DesignWare® Cores PCI Express Controller

Databook

DWC PCIe Root Port - Product Codes

Version 6.00a
June 2022

Copyright Notice and Proprietary Information

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

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

Synopsys, Inc.

www.synopsys.com

Contents

Revision History 15

Preface 17
Product Codes 18
Reference Documentation 21
Web Resources 22
Terms and Abbreviations 22
Synopsys Statement on Inclusivity and Diversity 27
Customer Support 27

Chapter 1
Product Overview 29
1.1 General Product Description 30
1.2 Applications 32
1.3 Features and Limitations 33
1.3.1 Gen6 Multistream Architecture 33
1.3.2 Gen5 and below 33
1.4 Frequency, Speed, and Width Support 36
1.4.1 Gen2/Gen3/Gen4/Gen5 Modes 36
1.4.2 Synthesis for 1000 MHz Clock Frequency 37
1.4.3 Supported Controller Configurations 40
1.4.4 Controller-PHY Compatibility 44
1.4.5 Frequency Step Module (CX_FREQ_STEP_EN =1) 49
1.4.6 Controller-PHY Integration 50
1.5 Deliverables 51
1.5.1 Verification Environment Overview 51
1.5.2 coreConsultant Overview 51
1.5.3 Synthesis Overview 52
1.6 Standards 53

Chapter 2
Architecture 55
2.1 Overview 56
2.2 RAM Requirements. 64
2.2.1 Overview 64
2.2.2 External RAM Verilog Instances 66

2.2.3 ECC Protection Support 67
2.2.4 RAM Sizes 67
2.2.5 Validating Multi-bit Read Enable 67
2.3 Clock Requirements 68
2.3.1 Overview 68
2.3.2 General Clock Relationships 70
2.3.3 Gen2, Gen3, Gen4, or Gen5 Speed Changing Considerations (CX_FREQ_STEP_EN =1) 73
2.3.4 AXI Bridge Clocking 74
2.3.5 Removing the Reference Clock 77
2.3.6 Additional Information 82
2.3.7 Clock Generation Module 83
2.4 Reset Requirements. 90
2.4.1 Overview 90
2.4.2 Implementing the PCIe Resets 95
2.4.3 Hot Reset 97
2.5 Receive Queues 100
2.5.1 Receive Queue Architecture 100
2.5.2 Configuring Your Queues 104

Chapter 3
Controller Operations 107
3.1 Initialization 109
3.1.1 Bus and Device Number 112
3.1.2 Segment ID Number 112
3.2 Link Establishment 115
3.2.1 Lane Resizing 115
3.2.2 Lane Reversal and Flipping 116
3.3 Transmit TLP Processing 120
3.3.1 Transmit TLP Arbitration in Gen6 120
3.3.2 Transmit TLP Arbitration 121
3.3.3 ACK/NAK Scheduling in Non-Flit Mode 125
3.3.4 Transmit Replay 126
3.3.5 Tx Interface Progression Detection (CX_INTERFACE_TIMER_EN =1) 128
3.3.6 MultiStream Transmission 130
3.4 Receive TLP Processing 134
3.4.1 Receive Filtering 134
3.4.2 Receive Routing 135
3.4.3 Error Handling 136
3.5 TLP Orthogonal Header Content (OHC). 138
3.6 Register Module, LBC, and DBI 141
3.6.1 Register Configuration Space Overview 142
3.6.2 Local Bus Controller (LBC) 145
3.6.3 Data Bus Interface (DBI) Access 146
3.6.4 Access Limitations 150
3.6.5 PHY Port Logic Registers 152
3.6.6 BAR Details 155
3.6.7 Generating Register Map 160
3.7 Reliability, Availability, and Serviceability (RAS) 162

3.7.1 RAS Wire Protection (ECRC) . . . 163
3.7.2 RAS Data Protection (RAS DP) . . . 164
3.7.3 RAS Debug, Error Injection, and Statistics (DES) . . . 177
3.8 Messages . . . 193
3.8.1 Message Generation . . . 194
3.8.2 Message Reception . . . 205
3.9 Interrupts . . . 210
3.9.1 Interrupt Reception (DSP) . . . 211
3.10 Flow Control . . . 216
3.10.1 Scaled Flow Control . . . 216
3.10.2 Shared Credits in Flit Mode . . . 217
3.11 Retry Mechanism . . . 219
3.12 Internal Address Translation Unit (iATU) . . . 220
3.12.1 Configuring the iATU . . . 221
3.12.2 Outbound Features and Limitations . . . 222
3.12.3 Outbound Limitations . . . 225
3.12.4 Outbound Basic Operation (Address Match Mode) . . . 226
3.12.5 Outbound Detailed Operation . . . 229
3.12.6 Outbound Programming . . . 234
3.12.7 Inbound Features . . . 236
3.12.8 Inbound Limitations . . . 237
3.12.9 Inbound Basic Operation . . . 238
3.12.10 Inbound Detailed Operation . . . 241
3.12.11 Inbound Programming . . . 246
3.13 Address Translation Services (ATS) . . . 248
3.14 Page Request Services (PRS) . . . 250
3.15 Peer-to-Peer Support (P2P) . . . 251
3.16 Gen2/Gen3/Gen4/Gen5 Speed Modes . . . 252
3.16.1 Gen2 5.0 GT/s Operation . . . 252
3.16.2 Gen3 8.0 GT/s Operation . . . 253
3.16.3 Gen4 16.0 GT/s Operation . . . 253
3.16.4 Gen5 32.0 GT/s Operation . . . 258
3.17 Power Management . . . 259
3.17.1 Overview . . . 260
3.17.2 L1 Substates . . . 266
3.17.3 L0s Entry and Exit Conditions . . . 278
3.17.4 L0p . . . 279
3.17.5 L1 Operation (Non-substates) . . . 284
3.17.6 L2 and L3 Power Down Entry and Exit Conditions (USP) . . . 291
3.17.7 Advanced Power Management and Power Domain Gating . . . 294
3.18 Completion Timeout Ranges . . . 312
3.19 TLP Processing Hints (TPH) . . . 313
3.19.1 Assumptions . . . 313
3.19.2 Limitations . . . 313
3.20 Atomic Operations (AtomicOps) . . . 314
3.20.1 AXI5 Support . . . 314
3.20.2 Internal Address Translation Unit (iATU) Support . . . 316
3.20.3 Atomic Routing - Egress Blocked Support . . . 316
3.21 TLP Prefix . . . 318

3.22 Separate Refclk Independent SSC (SRIS) . . . 321
3.23 Readiness Notifications (RN) . . . 322
3.23.1 FRS Queue . . . 323
3.24 Expansion ROM Validation . . . 324
3.25 Precision Time Measurement (PTM) . . . 325
3.25.1 PTM Timers . . . 325
3.25.2 Transmit and Receive Latencies . . . 325
3.25.3 Responder (DSP) Features . . . 328
3.25.4 Local PTM Clock Scaling . . . 330
3.26 Access Control Services (ACS) . . . 332
3.27 Management Component Transport Protocol (MCTP) . . . 334
3.28 Downstream Port Containment (DPC) . . . 335
3.29 Deferrable Memory Write (DMWr) . . . 337
3.29.1 Native PCIe Controller DMWr Support . . . 337
3.29.2 AXI Bridge Deferrable MemWr Support . . . 338
3.30 Enhanced Configuration Access Mechanism (ECAM) . . . 340
3.31 Native PCIe Enclosure Management (NPEM) . . . 342
3.32 Flit Performance Measurement Extended Capability . . . 343
3.32.1 Flit Latency Tracking . . . 343
3.32.2 LTSSM Latency Tracking . . . 345
3.33 Completion Queue Management . . . 349
3.33.1 Header and Data Completion Queue Size Calculation . . . 352

Chapter 4
Signal Interfaces . . . 355
4.1 Transmit Interfaces (XALI0/1/2) . . . 356
4.1.1 Transmit Client Interface Protocol Rules . . . 356
4.1.2 Transmit Client Transactions . . . 357
4.1.3 Transmit Address Alignment . . . 361
4.1.4 Target Completion Lookup Table . . . 363
4.2 FC-Transmit Interfaces (FC-XALI) . . . 365
4.2.1 Target Completion Lookup Table . . . 365
4.3 CCIX Transmit Interface (XALI_CCIX) . . . 366
4.3.1 CCIX Transmit Client Transactions . . . 366
4.3.2 CCIX TLP Header Fields and ccix_tlp_hdr Mapping . . . 367
4.4 Receive Bypass Interface (RBYP) . . . 369
4.4.1 Bypass Interface Protocol Rules . . . 369
4.4.2 Receive Bypass Interface for Gen6 Configuration . . . 369
4.5 Receive Request Interface (TRGT1) . . . 370
4.5.1 TRGT1 Protocol Rules . . . 370
4.5.2 TRGT1 Protocol Transactions . . . 370
4.5.3 TRGT1 Packet Grant and Halt . . . 371
4.6 FC-Receive Request Interface (FC-TRGT1) . . . 373
4.6.1 TRGT1 Interface Halting . . . 373
4.6.2 TRGT1 TLP Ordering Information . . . 373
4.7 CCIX Receive Request Interface (TRGT1_CCIX) . . . 374
4.7.1 CCIX TRGT1 Protocol Transactions . . . 374
4.7.2 CCIX TLP Header Fields and radm_ccix_hdr Mapping . . . 375

4.8 Data Bus Interface (DBI) 377
4.8.1 DBI Protocol Rules 377
4.8.2 DBI Protocol Transactions 377
4.9 External Local Bus Interface (ELBI) 379
4.9.1 ELBI Protocol Rules 379
4.9.2 ELBI Protocol Transactions 379
4.10 Vendor Message Interface (VMI) 381
4.10.1 VMI Protocol Rules 381
4.10.2 VMI Protocol Transactions 381
4.11 System Information Interface (SII) 382
4.12 PIPE Interface 383
4.12.1 64-bit PIPE 383
4.12.2 PIPE Interfaces 385
4.13 PHY Register Bus Interface (PRBI) 389
4.13.1 PRBI Protocol Transactions 389

Chapter 5
AMBA 393
5.1 AXI3/AXI4/AXI5 Support in PCIe Controller with AMBA Bridge 394
5.1.1 AXI Bridge Overview 394
5.1.2 AXI Bridge Architecture 400
5.1.3 AXI Bus Operations 414
5.1.4 Advanced AXI Reference Material 435
5.2 ACE-Lite Support in PCIe Controller with AMBA Bridge 445
5.3 DTIM Support in PCIe Controller with AMBA Bridge 447
5.3.1 Features 447
5.3.2 Limitations 448
5.3.3 AXI4-Stream Interface 449
5.3.4 RAMs 449
5.3.5 Programming Model 449
5.3.6 Error Handling 449
5.4 LTIM, Order Buffer, and MSI-GIC Interface 456
5.4.1 Overview and Supported Features 456
5.4.2 Local Translation Interface Master (LTIM) 458
5.4.3 Order Buffer 459
5.4.4 MSI-GIC 460
5.4.5 LTIM Mapping to AXI Manager 460

Chapter 6
Legacy DMA Controller 463
6.1 DMA Overview 464
6.1.1 Limitations 465
6.1.2 Feature List 466
6.2 DMA Architecture 468
6.2.1 Architecture Overview 468
6.2.2 Interrupts and Error Handling 469
6.2.3 Linked List Mode 475

6.2.4 Hardware Flow Control (CC_DMA_HSHAKE =1) 478
6.2.5 Native Controller Operation (without AXI Bridge) 481
6.3 Using the DMA 483
6.3.1 Source and Destination Address Registers (SAR, DAR) 483
6.3.2 DMA Transfer Size Register 485
6.3.3 Starting the DMA Transfer 485
6.3.4 Detecting the End of the DMA Transfer 485
6.3.5 Stopping the DMA Transfer (Software Stop) 486
6.3.6 TLP Generator Header Control 487
6.3.7 DMA Software Register Map 487
6.3.8 Programming Examples 490
6.4 Advanced DMA Information and Operation 501
6.4.1 Overview 501
6.4.2 Internal Architecture 504
6.4.3 DMA Ordering 505
6.4.4 Read Buffer 506
6.4.5 PCIe TAGs 507
6.4.6 Example Transfer 508
6.4.7 Error Handling 510
6.4.8 DMA Debug 512
6.4.9 Linked List Operation 513
6.4.10 Multichannel Arbitration 520

Chapter 7
HDMA Controller 523
7.1 HDMA Overview 524
7.1.1 Feature List 524
7.1.2 Limitations 526
7.2 HDMA Architecture 529
7.2.1 Architecture Overview 529
7.2.2 Interrupts and Error Handling 533
7.2.3 Linked List Mode 536
7.3 Using the HDMA 539
7.3.1 Source and Destination Address Registers (SAR, DAR) 539
7.3.2 HDMA Transfer Size Register 541
7.3.3 Starting the HDMA Transfer 541
7.3.4 Detecting the End of the HDMA Transfer 541
7.3.5 TLP Generator Header Control 542
7.3.6 Stopping the HDMA Transfer (Software Stop) 542
7.3.7 HDMA Software Register Map 543
7.4 Advanced HDMA Information and Operation 548
7.4.1 Overview 548
7.4.2 Error Handling 548
7.4.3 AXI Ordering (HDMA) 549
7.4.4 HDMA Debug 549

Chapter 8

Cache Coherent Interconnect for Accelerators (CCIX) 551
8.1 CCIX PCIe Controller Overview 552
8.1.1 Configuring the CCIX PCIe Controller 552
8.1.2 CCIX PCIe Controller Features 553
8.1.3 CCIX PCIe Controller Limitations 553
8.1.4 Frequency, Speed, and Width Support 554
8.2 CCIX PCIe Controller Operation 557
8.2.1 CCIX Extended Speed Mode Overview 557
8.2.2 PIPE Interface 560
8.2.3 CCIX Transaction Layer Receiver Operation 563
8.3 CXS Interface 565
8.3.1 CXS Overview 565
8.3.2 CXS Interface Clocking 566
8.3.3 CXS Interface Reset 567
8.3.4 CXS Transmitter Link Activation/Deactivation 567
8.3.5 CXS Receiver Link Activation/Deactivation 569
8.3.6 CXS Continuous Delivery Mode 571
8.3.7 CXS Power Management 572
8.3.8 CXS Data Protection 573
8.3.9 CXS Error Handling and Reporting 577
8.3.10 CXS Error Full Data (CXSERRORFULLDATA) 580
8.3.11 CXS Packet Constraints 580

Chapter 9
PCIe for USB4 583
9.1 PCIe Controller for USB4 Overview 584
9.1.1 Configuring PCIe Controller for USB4 584
9.1.2 Features 584
9.2 Precision Time Management (PTM) 587
9.3 Timer Adjustment 588
9.4 ASPM L1 Entry 589

Chapter 10
PCIe Integrity and Data Encryption (IDE) 591
10.1 PCIe IDE Features and Limitations 592
10.1.1 Features 592
10.1.2 Limitations 592
10.2 PCIe IDE Architecture Overview 594
10.2.1 Non-Multistream Architecture (When CX_MULTISTREAM_ARCH =0) 594
10.2.2 Multistream Architecture (When CX_MULTISTREAM_ARCH =1) 597
10.3 PCIe IDE Interfaces 601
10.3.1 PCIe IDE TX Interface 601
10.3.2 PCIe IDE RX Interface 602
10.3.3 PCIe IDE Message Interface 604
10.3.4 IDE Security Module Configuration Interface 605
10.4 PCIe IDE Error Handling and Reporting 607
10.5 PCIe and IDE Cores Interoperability 608

10.5.1 Configuring PCIe IDE 608

Chapter 11
Automotive 613
11.1 Overview 614
11.2 Automotive Safety Package 616
11.3 CDM Register Check. 617
11.4 Functional Safety Interrupt 620

Chapter 12
Parameter, Signal, Register, and Internal Parameter Descriptions 621

Appendix A
Advanced Information: Gen3/4/5 Equalization Details and Example 623
A.1 Equalization Overview and Synopsys-Specific Features 624
A.1.1 Preset Mapping Modes 626
A.1.2 Coefficient Tuning Feedback Modes 628
A.1.3 EQ Coefficient Request in FOM Mode (FCOEF) 632
A.1.4 Other Features and Limitations 635
A.1.5 Programming Registers 635
A.2 Detailed Equalization Procedure 637
A.2.1 Equalization Operations at the PHY Interface 637
A.2.2 Recovery.RcvrCfg 638
A.2.3 Recovery.Speed 638
A.2.4 Recovery.RcvrLock 638
A.2.5 Recovery.Equalization Phase 0 639
A.2.6 Recovery.Equalization Phase 1 639
A.2.7 Recovery.Equalization Phase 2 639
A.2.8 Recovery.Equalization Phase 3 (DSP/USP) 644
A.3 Usage Example 645

Appendix B
Advanced Information: Lane Reversal and Broken Lanes 649
B.1 Normal Link Establishment with Non-reversed/Reversed Lanes; and Wide/Narrow Ports 652
B.2 Link Down-sizing with Non-reversed/Reversed Lanes; and Wide/Narrow Ports. 655
B.3 Broken Lane Operation 658

Appendix C
Advanced Information: Loopback 663
C.1 Overview 664
C.2 Local Digital Loopback (PIPE) 665
C.2.1 Entering Local Loopback 665
C.2.2 In PIPE Loopback 665
C.2.3 Exiting Local Loopback 666
C.3 Local Analog Loopback. 667
C.4 Remote Digital Loopback 668
C.4.1 Entering Remote Digital Loopback 668
C.4.2 In Remote Loopback 668

C.4.3 Exiting Remote Loopback 669
C.5 Equalization for Loopback 670

Appendix D
Advanced Information: Lane Deskew 671
D.1 Conventional PCIe Deskew Requirements 672

Appendix E
Advanced Information: Clock and Data Crossing (CDC) 675
E.1 CDC Overview 676
E.2 AXI Bridge CDC Details 677
E.2.1 AXI Bridge RAM Architecture 677
E.2.2 LTIM and MSI-GIC Interface 677
E.3 SerDes Architecture CDC Details 679
E.4 Port Clocking and Input Synchonizers 680
E.5 CDC Reports 681

Appendix F
Advanced Information: Reset Domain Crossing (RDC) 683

Appendix G
Advanced Information: VC-Based Arbitration 685
G.1 VC-Based Arbitration Overview 686
G.2 VC-Based WRR Arbitration Programming Examples 688
G.2.1 Setting Up All VCs in the LPVC Group 688
G.2.2 Setting Up 4 VCs in the LPVC Group, 2 VCs in HPVC Group 688
G.2.3 Setting Up 5 VCs in the LPVC Group, 2 VCs in HPVC Group 689

Appendix H
Advanced Information: Advanced Filtering and Routing of TLPs 691
H.1 Filtering Rules 692
H.2 Filtering Algorithm 695
H.3 Root Port Routing Overview 698
H.4 Request TLP Routing Rules 699
H.5 Processing Illegal CFG TLPs and CFG1-CFG0 Conversion in Each PCI Express Port Type 701
H.5.1 RP Mode 701

Appendix I
Advanced Information: Advanced Ordering Information 703
I.1 PCIe Ordering Rules 704
I.2 Inbound (Receive) Order Enforcement 707
I.2.1 Receive Ordering Schemes 707
I.3 Outbound (Transmit) Order Enforcement 709
I.3.1 Ordering Across Outbound Interfaces 709
I.3.2 Outbound Order Enforcement on XALI0/1/2 Interfaces 709

Appendix J
Advanced Information: How to Turn Off Unused Lanes 711
J.1 Conventional PCIe 712

Appendix K

Advanced Information: Advanced Error Handling for Received TLPs 715
K.1 Routing of Request TLPs with Errors (Native PCIe Controller) 716
K.1.1 Store-and-Forward Queue Mode 716
K.1.2 Bypass Queue Mode 717
K.1.3 Disabling of Error Filtering 718
K.2 Routing of Request TLPs with Errors (AMBA Bridge Configuration) 719
K.2.1 Disabling of Error Filtering 719
K.3 Routing of Completions with Errors (Native PCIe Controller) 720
K.3.1 Store-and-Forward Queue Mode 720
K.3.2 Bypass/Cut-Through Queue Mode 721
K.4 Routing of Completions with Errors (AMBA Bridge Configuration) 723
K.5 Application Error Reporting Interface 724

Appendix L
Advanced Information: Calculating Gen1 Throughput 727
L.1 PCI Express Bandwidth and Throughput 728
L.1.1 Configuring the PCI Express Controller with Respect to Throughput 728
L.2 Effective Throughput 730
L.2.1 Effect of Link Layer Flow Control and ACK/NAK DLLPs in Non-Flit Mode 731
L.2.2 Other Factors Impacting Throughput 731

Appendix M
Advanced Information: Calculating AXI Bridge Throughput 733
M.1 AXI Bridge Throughput Measurements 734
M.1.1 Areas for Maximizing AXI Bridge Performance 736
M.1.2 Known Performance Limitations in AXI 736

Appendix N
Advanced Information: Advanced Routing of Received Messages 739
N.1 Routing of Received Messages 740
N.1.1 Message Reception on TRGT1 741
N.1.2 Message Reception Considerations when Using AXI Bridge 741

Appendix O
Advanced Information: Replay Buffer Sizing 745

Appendix P
Advanced Information: Endianness 747

Appendix Q
Advanced Information: Area, Power Estimates, and RAM Sizes 751
Q.1 Area Estimation 752
Q.2 Power Estimation 753
Q.3 RAM Sizing 754
Q.3.1 Sizing the AXI Outbound Completion Composer RAM 754
Q.4 SR-IOV Gate Count Costs 757

Appendix R
Advanced Information: Flow Control Credit Calculation 759
R.1 Calculation of Flow Control Latency 760
R.2 Calculation of Initial Flow Control Credits and Receive Buffer Sizes 761

Appendix S
Advanced Information: Serialization Queue Almost Full Threshold 763
S.1 Serialization Queue Almost Full Threshold 765
S.1.1 Data Rates 765
S.1.2 NACK Request to Replayed TLP Latency 768
S.1.3 Almost Full Threshold Calculation 771
S.2 Serialization Queue Input and Output Data Rates 773

Revision History

Revision history for PCI Express RP Controller Databook:

Version	Date	Description
6.00a	June 2022	■ A full list of functional (RTL) changes is in the “Release Notes: PCI Express Cores” document. ■ https://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/DWC_pcie_ctl_rp_-databook_change.pdf ■ The change-tracked version does not indicate minor changes that are made for readability, formatting, and so on. It only indicates changes to functionality, new features, and major rewrites. ■ The Signal Descriptions, Parameter Descriptions, Register Descriptions, and Internal Parameter Descriptions chapters are available in the *Reference Manual* book from 5.90a release onwards.

Preface

A single coreKit is provided for DesignWare PCI Express EP, RP, DM, and SW controller ports. Using the Device Type parameter (`CC_DEVICE_TYPE`) in coreConsultant GUI (as shown in Figure 1) you can configure the controller as a:

- 0 (EP): Endpoint
- 1 (RP): Root Port
- 2 (DM): Dual Mode (pin-selectable EP/RP)
- 3 (SW): Switch Port (pin-selectable Upstream/Downstream)

Attention

- The RP databook is not a description of the DM controller in RP mode.
- Unlike the Registers and Signal Descriptions chapters, the **Parameter Descriptions** chapter is common across all four *Reference Manuals*. The value of the `CC_DEVICE_TYPE` parameter in expressions should be interpreted according to the previous list.
- The parameter descriptions in **Parameter Descriptions** chapter of the *Reference Manual* include the **Enabled:** attribute which indicates the values required to be set on other parameters before you can change the value of the parameter in question.

Figure 1 Device Type Parameter in coreConsultant GUI

After the PCIe controller is configured as EP, RP, DM, or SW port, depending upon the features licensed, the PCIe controller can be identified as:

- **Native PCIe controller (without AMBA bridge)**: Basic PCIe controller which has its own non-standard, proprietary dedicated bus interface to your application.
- **PCIe controller with AMBA bridge**: PCIe controller which uses the AMBA bridge to interface to your application. To enable the AMBA bridge, set the parameter `AMBA_INTERFACE` greater than 1.

Note The controller does not support an AHB interface. Ignore all references to legacy AHB interface (if any) in the PCIe documents.

- **CCIX PCIe controller (without/with AMBA bridge)**: PCIe controller which has its own non-standard, proprietary dedicated CCIX interface to your application (when not using the AMBA bridge) or CCIX compliant AMBA bridge interface to your application (when using the AMBA bridge). To enable CCIX features, set the parameter `CX_CCIX_ENABLE` to 1.
- **Automotive PCIe controller (without/with AMBA bridge)**: ASIL B certified PCIe controller which supports features such as, CDM register check, TX interface progression detection, in addition to native PCIe controller features such as, Reliability, Availability, and Serviceability (RAS) and RAS Data Protection (RAS DP). To enable automotive features, set the parameter `CX_AUTOMOTIVE_ENABLE` to 1.
- **CCIX Automotive PCIe controller (without/with AMBA bridge)**: PCIe controller supporting both CCIX and Automotive features.
- **Embedded Endpoint Solution**: Embedded Endpoint or Switch DSP integrated Endpoint solution.
- **PCIe controller for USB4 (without/with AMBA bridge)**: The PCIe controller for USB4 connects to USB4 PCIe UP/DN adapter through an enhanced PIPE interface seamlessly.

Product Codes

This section lists the product codes and product names for DesignWare PCI Express controller ports.

Table 1 Native PCIe Controller (without AMBA Bridge)

Product Code	Product Name
C902-0	DWC PCIe 1.0 Standard
C903-0	DWC PCIe 1.0 Plus
C904-0	DWC PCIe 1.0 Premium
C908-0	DWC PCIe 2.0 Standard
C909-0	DWC PCIe 2.0 Plus
C910-0	DWC PCIe 2.0 Premium
F055-0	DWC PCIe 2.0 Premium II
C914-0	DWC PCIe 3.0 Standard
C915-0	DWC PCIe 3.0 Plus
C916-0	DWC PCIe 3.0 Premium
F059-0	DWC PCIe 3.0 Premium II
C920-0	DWC PCIe 4.0 Standard
C921-0	DWC PCIe 4.0 Plus

Product Code	Product Name
C922-0	DWC PCIe 4.0 Premium
F063-0	DWC PCIe 4.0 Premium II
C926-0	DWC PCIe 5.0 Standard
C927-0	DWC PCIe 5.0 Plus
C928-0	DWC PCIe 5.0 Premium
F067-0	DWC PCIe 5.0 Premium II

Table 2 PCIe Controller with AMBA Bridge

Product Code	Product Name
C905-0	DWC PCIe 1.0 Standard AMBA
C906-0	DWC PCIe 1.0 Plus AMBA
C907-0	DWC PCIe 1.0 Premium AMBA
C911-0	DWC PCIe 2.0 Standard AMBA
F056-0	DWC PCIe 2.0 Standard AMBA II
C912-0	DWC PCIe 2.0 Plus AMBA
F057-0	DWC PCIe 2.0 Plus AMBA II
C913-0	DWC PCIe 2.0 Premium AMBA
F058-0	DWC PCIe 2.0 Premium AMBA II
C917-0	DWC PCIe 3.0 Standard AMBA
F060-0	DWC PCIe 3.0 Standard AMBA II
C918-0	DWC PCIe 3.0 Plus AMBA
F061-0	DWC PCIe 3.0 Plus AMBA II
C919-0	DWC PCIe 3.0 Premium AMBA
F062-0	DWC PCIe 3.0 Premium AMBA II
C923-0	DWC PCIe 4.0 Standard AMBA
F064-0	DWC PCIe 4.0 Standard AMBA II
C924-0	DWC PCIe 4.0 Plus AMBA
F065-0	DWC PCIe 4.0 Plus AMBA II
C925-0	DWC PCIe 4.0 Premium AMBA
F066-0	DWC PCIe 4.0 Premium AMBA II

Product Code	Product Name
C929-0	DWC PCIe 5.0 Standard AMBA
F068-0	DWC PCIe 5.0 Standard AMBA II
C930-0	DWC PCIe 5.0 Plus AMBA
F069-0	DWC PCIe 5.0 Plus AMBA II
C931-0	DWC PCIe 5.0 Premium AMBA
F070-0	DWC PCIe 5.0 Premium AMBA II

Table 3 CCIX PCIe Controller

Product Code	Product Name
C932-0	DWC CCIX 25G Premium
F071-0	DWC CCIX 25G Premium II
C933-0	DWC CCIX 25G Premium AMBA
F072-0	DWC CCIX 25G Premium AMBA II
D375-0	DWC CCIX 32G Premium
F073-0	DWC CCIX 32G Premium II
D376-0	DWC CCIX 32G Premium AMBA
F074-0	DWC CCIX 32G Premium AMBA II

Table 4 Automotive PCIe Controller

Product Code	Product Name
C934-0	DWC AP PCIe 1.0 Premium
C935-0	DWC AP PCIe 1.0 Premium AMBA
C936-0	DWC AP PCIe 2.0 Premium
F077-0	DWC AP PCIe 2.0 Premium ||
C937-0	DWC AP PCIe 2.0 Premium AMBA
F078-0	DWC AP PCIe 2.0 Premium AMBA II
C938-0	DWC AP PCIe 3.0 Premium
F079-0	DWC AP PCIe 3.0 Premium ||
C939-0	DWC AP PCIe 3.0 Premium AMBA
F080-0	DWC AP PCIe 3.0 Premium AMBA II

Product Code	Product Name
C940-0	DWC AP PCIe 4.0 Premium
F081-0	DWC AP PCIe 4.0 Premium ||
C941-0	DWC AP PCIe 4.0 Premium AMBA
F082-0	DWC AP PCIe 4.0 Premium AMBA II
E808-0	DWC AP PCIe 5.0 Premium
F083-0	DWC AP PCIe 5.0 Premium ||
E809-0	DWC AP PCIe 5.0 Premium AMBA
F084-0	DWC AP PCIe 5.0 Premium AMBA II

Table 5 CCIX Automotive PCIe Controller

Product Code	Product Name
C942-0	DWC AP CCIX 25G Premium
F075-0	DWC AP CCIX 25G Premium II
C943-0	DWC AP CCIX 25G Premium AMBA
F076-0	DWC AP CCIX 25G Premium AMBA II

Table 6 PCIe Controller for USB4

Product Code	Product Name
E347-0	DWC USB4 PCIe Premium
E348-0	DWC USB4 PCIe Premium AMBA

Table 7 PCIe Controller for HPC

Product Code	Product Name
F537-0	DWC PCIe 4.0 Premium AMBA HPC
F538-0	DWC PCIe 5.0 Premium AMBA HPC

Reference Documentation

After installing the controller, the DWC PCI Express documents can be found under: `$DESIGNWARE_HOME/iip/DWC_pcie_ctl/latest/doc`

When you create a workspace in coreConsultant,

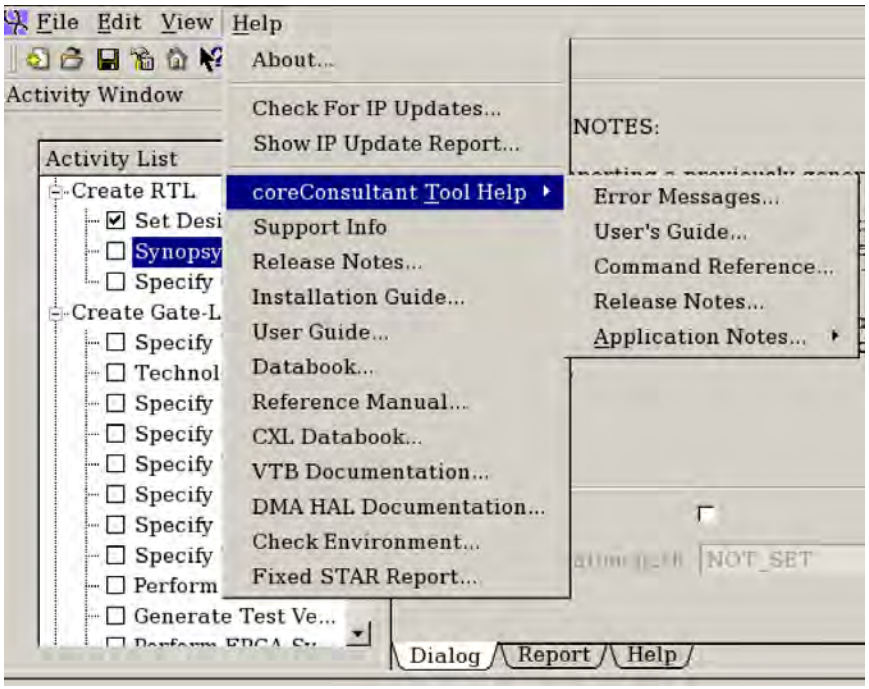
- Pre-configuration, the DWC PCI Express documents for all four ports are available in `<work-space>/doc`.

- Post-configuration, only the documents corresponding to your configured controller port are available in `<workspace>/doc`.

When viewing documents from coreConsultant GUI (as shown in Figure 2),

- Pre-configuration: The coreConsultant GUI points to the DM port documents.
- Post-configuration: The coreConsultant GUI points to your configured controller port documents.

Figure 2 Viewing Documents from coreConsultant GUI



Note For parameter descriptions, signal descriptions, register descriptions, and internal parameter descriptions, see *Reference Manual*.

Web Resources

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

Terms and Abbreviations

The following terms are used throughout this document:

Table 8 Terms and Descriptions

Term	Description
x1/x2/x4/x8/x16	1/2/4/8/16 lanes
BAR	Base Address Register
Application Software/Application Logic	Application software is a single program or a collection of small programs intended to perform certain tasks. In this document Application Software/Application logic is used to identify customer specific application software/logic, for example customer implementation to handle Rx/TX/VDM TLPs.
CCIX	Cache Coherent Interconnect for Accelerators
CDM	Configuration Dependent Module This is an internal block in the native controller that has the PCIe configuration registers and some user-accessible registers.
CPL	Completion
C-PCIe	Conventional PCI Express
CRPI	Control Register Parallel Interface The control registers in some Synopsys PHYs (for example, C8/C10/E12/E16) can be accessed through this interface.
CXPL	Common Xpress Port Logic The internal controller module that implements the majority of the PCI Express protocol layers.
CXS	An Interface for streaming CCIX traffic.
DBI	Data Bus Interface You can use this interface to locally access the controller's internal registers in the CDM. You can optionally connect a local CPU or controller to this port.
DLLP	Data Link Layer Packet
DRS	Device Readiness Status
DSP	Downstream port Refers to Root Port (RP) and/or Switch Downstream Port.
DWORD	Double Word (32 bits)
Flit Mode (FM)	New encoding mode, TLP framing rules, Ordered Set definitions
FRS	Function Readiness Status
FRSQ	FRS Queue
Gen6	6th Generation of PCIe protocol with per-lane speed of 64GT/s
Host System	System in which the host CPU resides
Host Software	Software pertaining to the host system
iATU	Internal Address Translation Unit

Term	Description
IDE	Integrity and Data Encryption
IDLE Flit	Flit containing only NOP TLPs and NOP2 DLLP
iMSI-RX	Integrated MSI-Receive module
Inbound traffic	PCIe transactions that enter the controller from the wire side of the controller (PCIe wire). These transactions are delivered to your application side.
LTR	Latency Tolerance Reporting
MTU	Maximum Transfer Unit Specifies the maximum packet payload size supported. This indicates the maximum allowed transfer size for a write or completion.
MultiStream Architecture	Describes the ADM architecture to enable streaming of parallel TLPs in the transmit and receive datapaths. This feature is enabled by parameter CX_MULTISTREAM_ARCH.
MultiStream Transmission	This feature enables per-VC XALI interfaces on the transmit datapath. Enables dedicated interfaces to access the VC arbiter within the transmit datapath of the controller. This feature is enabled by parameter CX_MULTISTREAM.
Native PCIe controller	Identifies the basic PCIe controller (without AXI Bridge) which has its own non-standard, proprietary dedicated bus interface to your application.
NF	Number of Functions The value “1” represents one function.
NFM	Non-Flit Mode PCIe Gen5 encoding
NOP TLP	1DW sent by transaction layer when no TLPs to send
NOP2 DLLP	Null DLLP
NP	Non-posted
NRZ	Non-Return to Zero
NW	Number of Double Words The value “1” represents a 32-bit DWORD.
OHC[A-E]	Orthogonal Header Content
Outbound traffic	Transactions that enter the controller from your application side of the controller. These transactions are passed to the native controller, where they are sent out onto the PCIe wire.
P	Posted
Page Boundary	Specifies the address page boundary size supported by the AXI bridge. No packet can have an address that crosses the specified address boundary.
PAM-4	Pulse Amplified Modulation with 4 levels
PCIe	PCI Express

Term	Description
PIPE	Standard PIPE interface between the PCI Express PHY and the controller. If you set PHY_TYPE to be the Synopsys PHY, then the PHY is included inside the controller.
PMBC	PIPE Message Bus Controller This module controls the PIPE Message Bus interface. It is enabled only when `CX_PIPE_VER >=3` or `CX_MAX_PCIE_SPEED >=4`.
PMC	Power Management Controller
PRC	PHY Register Controller
PRBI	PHY Register Bus Interface This interface facilitates direct access of PHY registers by your application through DBI or CFG requests. Note: Only Synopsys PHYs supporting Control Register Parallel Interface (CRPI), for example Synopsys C8/C10/E12/E16 PHYs can be accessed through this interface.
PTM	Precision Time Measurement feature as defined in *PCI Express Base Specification, Revision 6.0, Version 1.0*.
RADM	Receive Application-Dependent Module
RAS	Reliability, Availability, and Serviceability A collection of features aimed at error protection, error and event logging, and debug.
RAS DP	Data Protection aspect of RAS
RAS-DES/RASDES	Debug, Error injection, and Statistics aspects of RAS
RC	Root Complex A defined System Element that includes at least one Host Bridge, Root Port, or Root Complex Integrated Endpoint.
RP	PCI Express Root Port (RP) controller
SBDF	Segment Bus Device Function
SII	System Information Interface Exchanges system information between the controller and your application.
Software	Software is often divided into application software and system software.
System Software	System software includes System Firmware (BIOS, UEFI), Operating System, VMM, power management services, device drivers, user-mode services, kernel mode services, platform vendor's add-on to the Operating System. It is also responsible for managing hardware components and providing basic non-task-specific functions. Configuration Software is an example of system software responsible for accessing Configuration Space and configuring the PCI/PCIe bus.
TLP	Transaction Layer Packet

Term	Description
TRGT1/RBYP	Receive Target 1 and Bypass Interfaces These are the RX interfaces used to connect the native controller to your application. RBYP is used by queues that are in bypass mode. When you configure the controller with an AXI bridge, then these interfaces are no longer visible as they now become an internal connection between the native PCIe controller and the AXI bridge. Your application receives inbound TLPs on AXI bridge, instead of TRGT1/RBYP.
TRGT1_CCIX	Receive Target 1 CCIX Interface This RX interface is used to connect the native controller to your CCIX application.
User Application	Generic term for customer specific software (system as well as application, depending on the context).
USP	Upstream port Refers to Endpoint and/or Switch Upstream Port.
VC	Virtual Channel
VMI	Vendor Message Interface Allows your application to request a transmission of a vendor message independent from the XALI client interfaces.
VSEC	Vendor Specific Extended Capability Most of the Synopsys implementation-specific registers (not defined by the PCI-SIG PCIe specification) are located in the Port Logic space beginning at 0x700. From release 4.40a onwards, all new implementation-specific register groups are implemented as VSECs in the PCIe Extended Capability Structure address space.
XALI0	Transmit Client 0 Interface Transmit interface for outbound request or CPL TLPs. When you configure the controller with an AXI bridge, then this interface is no longer visible as it now becomes an internal connection between the native PCIe controller and the AXI bridge. Your application transmits outbound requests or CPL TLPs on AXI bridge, instead of XALI0.
XALI1	Transmit Client 1 Interface Additional application transmit interface, identical to XALI0. The usage of XALI0 and XALI1 is up to your application. For example, an application might use XALI0 for requests and XALI1 for completions. When you configure the controller with an AXI bridge, then this interface is no longer visible as it now becomes an internal connection between the native PCIe controller and the AXI bridge. Your application transmits outbound requests or CPL TLPs on AXI bridge, instead of XALI1.
XALI2	Transmit Client 2 Interface An optional transmit client interface, identical to XALI0 and XALI1. An example usage of the three transmit client interfaces is to use one for posted requests, one for non-posted requests, and one for completions. When you configure the controller with an AXI bridge, then this interface is no longer visible as it now becomes an internal connection between the native PCIe controller and the AXI bridge. Your application transmits outbound requests or CPL TLPs on AXI bridge, instead of XALI2.

Term	Description
XALI_CCIX	Transmit Client CCIX Interface Transmit interface for outbound CCIX TLPs.
XADM	Transmit Application-Dependent Module

Synopsys Statement on Inclusivity and Diversity

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

Customer Support

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

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

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

  a. https://solvnetplus.synopsys.com

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

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

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

     Make sure to include the following:

     - **Product L1:** DesignWare Cores

- **Product L2:** PCI Express

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

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

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

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

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

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

1

Product Overview

This chapter gives an overview of the PCI Express controller. The topics in this section are:

- "General Product Description" on page 30
- "Applications" on page 32
- "Features and Limitations" on page 33
- "Frequency, Speed, and Width Support" on page 36
- "Deliverables" on page 51
- "Standards" on page 53

1.1 General Product Description

This DesignWare PCI Express controller provides a solution to implement a PCI Express Root Port for a PCI Express Root Port application.

Attention

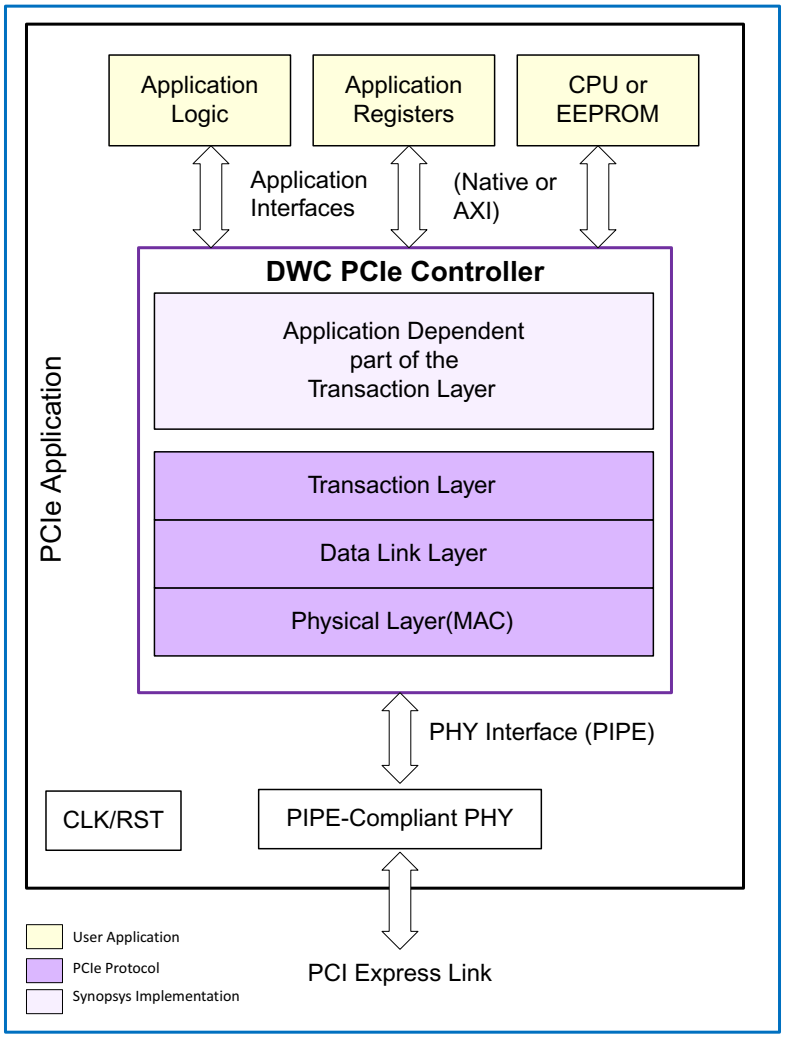
The Synopsys PCI Express IP does not implement a full Root Port.

To implement a full Root Port, you must add your application logic (connected to the application client interface or AXI interface) and other support logic (including clock and reset generation). For more information, see “Integrating with your Application RTL” section in the “Integrating the controller with the PHY, or Application RTL, or Verification IP” chapter of the *User Guide*.

The PCIe controller implements the three PCI Express protocol layers (Transaction layer, Data Link layer, and the MAC portion of the physical layer) in synthesizable RTL. It also implements the application-dependent functionality of the PCI Express Transaction Layer for packet transmission, which is located between your application logic and the PCI Express protocol layers.

A complete PCI Express port solution includes the controller, an analog PHY macro, and application logic to source and sink data. The physical layer is split across the PIPE and controller such that the MAC functionality (LTSSM, lane-to-lane deskew) is in the controller and the PHY functionality is implemented in the PIPE-compliant PHY. The PHY is outside of the controller, interfacing through the standard PIPE interface. For more information, see “Integrating the Controller with the PHY” section in the “Integrating the Controller with the PHY, or Application RTL, or Verification IP” chapter of the *User Guide*.

Figure 1-1 PCIe Application



1.2 Applications

Typical applications for a PCI Express component built with the controller include:

- Motherboard components for desktop and mobile computers
- Graphics devices
- Add-in cards for desktop and mobile computers
- Components and add-in cards in server applications
- Embedded applications
- Data communications equipment
- Telecommunications equipment
- Storage devices
- Wireless devices
- Other applications

Note The DM controller is not intended for use in a PCI Express Switch or as a Root Complex integrated Endpoint.

1.3 Features and Limitations

1.3.1 Gen6 Multistream Architecture

All features mentioned in "Gen5 and below" on page 33 carry forward to the Gen6 configured controller with the following exceptions:

- Lightweight Notification
- PHY Register Access
- Embedded DMA (only Hyper DMA (HDMA) is supported)
- 32-bit and 64-bit Internal Datapath (minimum internal datapath width is 128-bit)

The following additional features are supported in the Gen6 configured controller:

- Up to Gen1, Gen2, Gen3, Gen4, Gen5, or Gen6 (x1, x2, x4, x8, or x16 lanes)
- 128, 256, 512 or 1024-bit Internal Datapath operating at 62.5,125, 250, 500, 625, 781.25, 1000, 1250, or 1562.5MHz
- Support for Flit Mode and Non-Flit Mode Link
- 64GT/s PAM-4
- Forward Error Correction (FEC)
- L0p Dynamic Link Re-sizing
- 14-bit Tags
- Flit Logging Extended Capability
- 1024-bit datapath width at AXI Manager/Subordinate ports
- Single Port AXI 2GHz operation
- Dual Port AXI Bridge (2 Manager / 2 Subordinates)
- Dual Port LTI (2 LTI Manager ports)
- MultiStream Transmit Interface
- MultiStream Receive Interface
- Optional
  - Flit Performance Measurement Extended Capability
  - Flit Error Injection Extended Capability

1.3.2 Gen5 and below

The Synopsys PCI Express controller supports:

- All non-optional features of the PCI Express Base Specification, Revision 6.0, Version 1.0
- The following optional features of the specification:
  - Address Translation Services (ATS) by your application
  - Page Request Services (PRS) by your application
  - ID-Based Ordering (IDO) by your application
  - TLP Processing Hints (TPH)

  - Atomic Operations
  - TLP Prefix
  - Extended Tag Support
  - Optimized Buffer Fill and Flush (OBFF)
  - Separate Refclk with Independent Spread Spectrum Clocking (SRIS)
  - Gen3 Receiver Impedance
  - Lightweight Notifications (LN)
  - Readiness Notifications (RN)
  - PCI Express Active State Power Management (ASPM)
  - PCI Express Advanced Error Reporting (AER) with Multiple Header Logging
  - Vital Product Data (VPD)
  - Expansion ROM Validation
  - Precision Time Measurement (PTM)
  - Access Control Services (ACS)
  - Native PCIe Enclosure Management (NPEM)
  - Downstream Port Containment (DPC)
  - Root Port Programmed I/O to DPC (eDPC)
  - Deferrable Memory Write (DMWr)
  - Enhanced Configuration Access Mechanism (ECAM)
  - Integrity and Data Encryption (IDE)
- Up to x16 Gen1, Gen2, Gen3, Gen4, or Gen5 lanes (x1, x2, x4, x8, or x16)
- 32, 64, 128, 256, or 512-bit Internal Datapath Operating at 62.5, 125, 250, 500, 625, 781.25, 1000, 1250, or 1562.5 MHz
- Automatic Integration with Most PHY types (8-bit, 16-bit, 32-bit, and 64-bit PIPE Width per Lane)
- Power Gating (UPF) Support
- Advanced Power and Clock Management
- Internal Address Translation Unit (iATU)
- Internal MSI-X Generation Module
- AXI3/AXI4/AXI5/ACE-Lite bridge
- Embedded DMA with Hardware Flow Control
- Embedded Hyper DMA (HDMA)
- Upconfigure Support
- RAS DES (Debug, Error Injection, and Statistics)
- RAS DP (Data Protection) for Datapath and RAMs using ECC or Parity
- ECRC Generation and Checking
- Multiple Virtual Channels (VCs), Traffic Classes (TCs), and Quality of Service (QOS)
- Bypass, Cut-through, and Store-and-forward Queue Modes for Rx TLPs
- Configurable Receive and Retry Buffer sizes

- Configurable Max_Payload_Size size (128 bytes to 4 KB)
- Configurable Filtering Rules for Posted, Non-posted, and Completion Traffic
- Configurable Filtering, I/O Filtering, Configuration Filtering and Completion Lookup/Timeout
- Three Application Transmit Clients
- MSI-X with Per-Vector Masking (PVM)
- Type 1 Configuration Space (configurable bus and device numbers for multiple port instances)
- Automatic Lane Reversal
- Application-initiated Manual Lane Reversal/flip for situations where controller does not detect Lane 0
- Transaction Filtering and Routing Look up
- PHY Register access (Only for PHYs supporting Control Register Parallel Interface)
- Automotive support
- Transmit Interface Progression Detection
- CDM Register Check
- CCIX Support
- CXS Support
- PCIe for USB4

Attention

Some features require an additional license. Licensing requirements are given in the Installation Guide.

Some features are not available or might have limitations, when you are using the AXI bridge. For more information, see "AXI Features and Limitations" on page 395.

Attention

In this version of the PCIe documents,

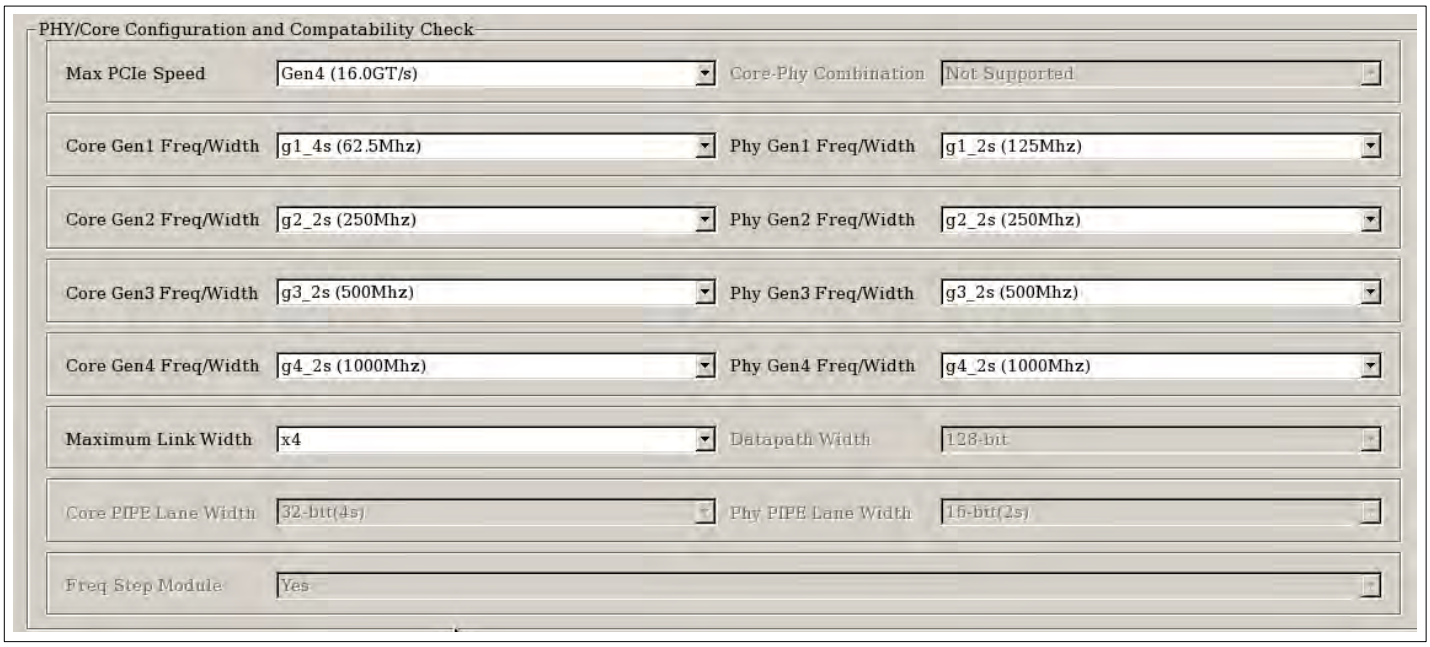
- AXI Master is the same as AXI Manager
- AXI Slave is the same as AXI Subordinate

1.4 Frequency, Speed, and Width Support

To set the frequency, datapath width, and speed modes of the controller, use the following configuration parameters in the coreConsultant GUI:

- Max Link Speed (`CX_MAX_PCIE_SPEED`)
- Controller Gen1, Gen2, Gen3, Gen4, Gen5, and Gen6 Freq/Width (`CX_MAC_SMODE_GEN1, CX_MAC_SMODE_GEN2, CX_MAC_SMODE_GEN3, CX_MAC_SMODE_GEN4, CX_MAC_SMODE_GEN5, and CX_MAC_SMODE_GEN6`)
- PHY Gen1, Gen2, Gen3, Gen4, Gen5, and Gen6 Freq/Width (`CX_PHY_SMODE_GEN1, CX_PHY_S-MODE_GEN2, CX_PHY_SMODE_GEN3, CX_PHY_SMODE_GEN4, and CX_PHY_SMODE_GEN5, and CX_PHY_SMODE_GEN6`)
- Maximum Link Width (`CX_NL`)

Figure 1-2 Conventional PCIe Frequency, Speed, and Width Configuration Parameters in coreConsultant GUI



After you have set these parameters, the coreConsultant tool cross-references your available licenses and automatically calculates the following values:

- Controller PIPE Lane Width (`CX_PIPE_WIDTH_` bits or `CX_NB` bytes)
- PHY PIPE Lane Width: (`CX_PHY_PIPE_WIDTH_` bits or `CX_PHY_NB` bytes)
- Datapath Width: (`CX_NW`/32 bits rounded up to multiple of `32` or `CX_NL*CX_NB` bytes)
- `CX_FREQ_STEP_DOWN_EN`. This is the enable for the "Frequency Step Module (CX_FREQ_STEP_EN =1)" on page 49.

1.4.1 Gen2/Gen3/Gen4/Gen5 Modes

The controller supports Dynamic frequency, Dynamic width, and Data throttling configurations for Gen2/Gen3/Gen4/Gen5 modes.

- **Dynamic Frequency (DF):** For DF configurations, the number of active symbols on the PIPE is constant; the frequency of the controller doubles each time as the controller transitions from Gen1 -> Gen2 -> Gen3 -> Gen4 -> Gen5 rates.
- **Dynamic Width (DW):** For DW configurations, the frequency of the controller is constant; the number of active symbols on the PIPE doubles each time as the controller transitions from Gen1 -> Gen2 -> Gen3 -> Gen4 rates.
- **Data Throttling (DT):** DT is a rate reduction mode for the PIPE interface that allows for a higher clock rate to be used at a slower data rate. It uses the `mac_phy_txdatavalid` and `phy_mac_rxdatavalid` signals to obtain the slower effective rate by toggling them every other clock (2:1 DT) or every fourth clock (4:1 DT). This controls the number of valid symbols per one, two, or four clock cycles.

  The `mac_phy_txdatavalid` and `phy_mac_rxdatavalid` signals are present in Gen3/Gen4/Gen5 speed modes. When DT is not used in a lower speed mode, these signals must be held high. In Gen3/Gen4/Gen5 speed modes, these signals are also used for regular de-assertion for 128b/130b encoding and decoding. For more information on the usage of the `mac_phy_txdatavalid` and `phy_mac_rxdatavalid` signals, see "Signal Interfaces" on page 355. However, DT is only enabled for some controller/PHY combinations as described in "Controller-PHY Compatibility" on page 44.

Note

- The controller and PHY can have different Gen2 modes. For example, you can configure the controller for Dynamic Frequency and the PHY for Dynamic Width. This is also true for Gen3, Gen4, and Gen5 speed modes.
- For more information, see "Gen2/Gen3/Gen4/Gen5 Speed Modes" on page 252.

1.4.2 Synthesis for 1000 MHz Clock Frequency

1 GHz operation is functionally supported as indicated in Table 1-2 on page 41. However, passing critical timing is dependent on your configuration, process technology, operating conditions, and sign-off corner, and may not be supported for all crosses of these variables. The default pipeline settings are configured to allow timing closure without incurring unnecessary latency. However, it may be necessary to enable the parameter `CX_CUSTOM_PIPELINING` to enable additional pipelines for your particular configuration. It is recommended that you assess the feasibility of closing timing with your specific scenario setup by adjusting your configuration parameters as suggested in Table 1-1.

You must follow the requirements for configuration parameter settings in Table 1-1.

Table 1-1 Timing Critical Parameter Setting for 1 GHz Clock Frequency

Feature	Parameter	Value	Suggestions
RAS DP Datapath Protection	CX_RASDP	3 or 4Data path protection with ECC	Use Parity for Data path protection (CX_RASDP = 1 or 2)
Address Translation Unit (ATU)	CX_INTERNAL_ATU_ENABLE CX_ATU_NUM_OUTBOUND_REGIONS CX_ATU_NUM_INBOUND_REGIONS	>64 Regions	Reduce the number of regions
Multiple Virtual Channels	CX_NVC	>2	Reduce the number of VCs

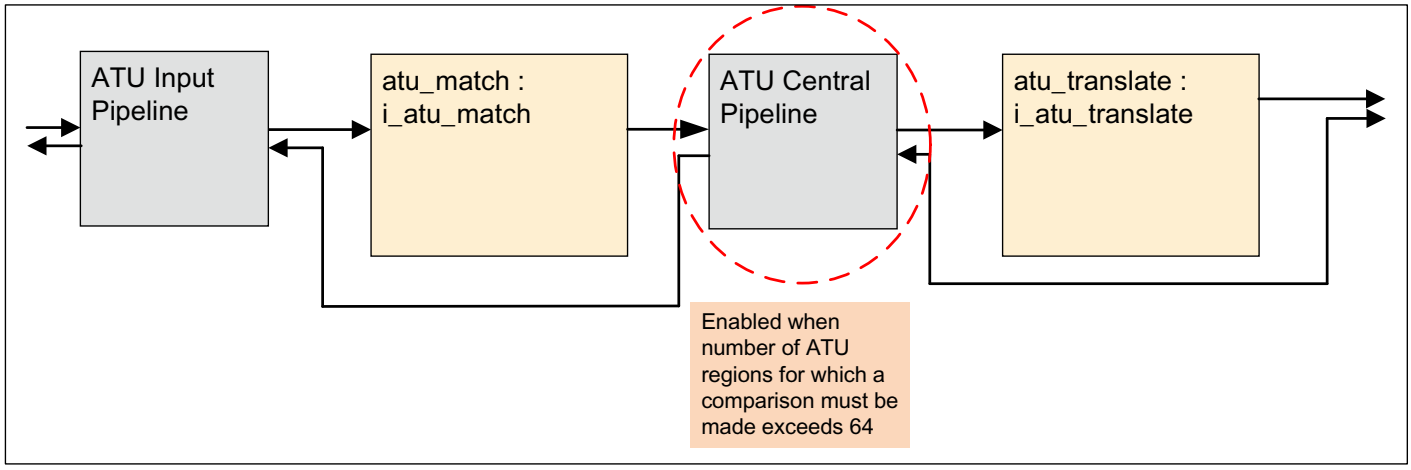
Feature	Parameter	Value	Suggestions
Multiple Functions	CX_NFUNC	> 8	Reduce the number of functions
Legacy DMA	CC_DMA_ENABLE	1	■ Reduce the number of tags (CC_MAX-_SLV_TAG & CC_MAX_M-STR_TAG_AXI) ■ Reduce number of DMA channels ■ Disable Byte Alignment feature (CC_D-MA_BA_ENABLE[a] =0)
SR-IOV	CX_SRIOV_ENABLE CX_MAX_VF_X	>128 VFs	■ Reduce number of VFsDisable DYNAM-IC_VF_ENABLE ■ Enable CX_VF_STRIDE_AL-WAYS_ONE
TLP Prefix	CX_NPRFX	>4	Reduce the number of TLP Prefixes

a. This is a hidden parameter and is not visible in coreConsultant GUI. The default value of CC_DMA_BA_ENABLE is '1'. You can change the value of this parameter through coreConsultant command line (using set_configuration_parameter command).

To discuss trade-offs, or optimizations to your configuration, contact Synopsys support through SolvNetPlus.

For Gen6, in Multistream Architecture, the controller adds an additional pipeline stage to allow timing closure when the number of regions exceed 64. This additional pipeline stage helps with critical paths through the iATU when the `core_clk` is greater than 1 GHz, as shown in Figure 1-3.

Figure 1-3 Synthesis of Gen6 Multistream Architecture at 1000 MHz



Floorplan Considerations for 1 GHz Clock Frequency (Controller Standalone)

The module and pin placement floorplan considerations for 1 GHz clock frequency are as follows:

Module Placement

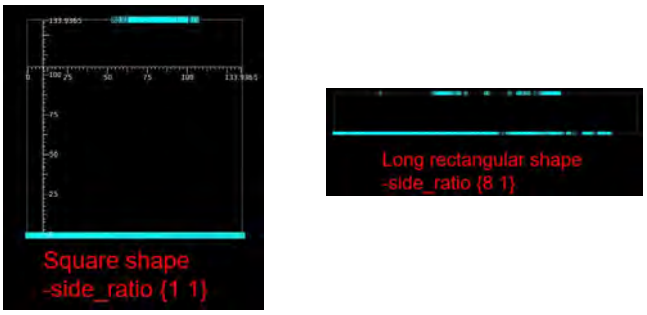
- Place application interface modules close to your application logic
  - AXI Configuration

        - u_axi_bridge
        - u_DWC_pcie_edma_axi_bridge
    - Native Configuration
        - u_DWC_pcie_core/u_xadm
        - u_DWC_pcie_core/u_radm
- Place physical layer modules close to PHY
    - u_DWC_pcie_core/u_cx_pl/u_xmlh
    - u_DWC_pcie_core/u_cx_pl/u_rmlh
    - u_DWC_pcie_core/u_cx_pl/u_smlh
    - u_DWC_pcie_core/gen_pipe_regif *
    - u_DWC_pcie_core/gen_freq_step *
    - u_DWC_pcie_core/u_lane_flip
    - u_DWC_pcie_core/u_pipe *

PIN Placement

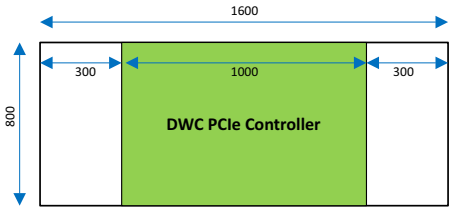
- Place PIPE interface pins close to PHY
    - PIPE Interface
    - PHY Register Bus Interface
- Reduce top level pin count by removing unused ports
- Place application interface pins close to your application logic
    - AXI Configuration
        - AXI Manager interface/AXI Subordinate interface
    - Native Configuration
        - Receive Bypass/Request Interface (RBYP/TRGT1)
        - Transmit Interfaces (XALI0/1/2)
    - Consider using DC-topo auto placement, in some cases it gives better results than manual placement.
- Aspect Ratio
    - Constrain the PCIe controller floorplan to a square shape as much as possible to ease timing as shown in Figure 1-4. Avoid a long rectangular shape as this may cause the tool to insert additional buffers thus creating long paths and timing challenges as shown in Figure 1-4.

Figure 1-4 Example: Square and Rectangular Shaped Controller Floorplan



- If a rectangular floorplan cannot be avoided, constraining the controller into a box as square as possible (using blockage) considering utilization within the overall floorplan helps ease timing. For example, if the floorplan is [1600 x 800], try constraining the PCIe Controller to [1000 x 800] as shown in Figure 1-5.

Figure 1-5 Example: Controller Floorplan Within a Rectangular Floorplan



1.4.3 Supported Controller Configurations

Table 1-2 lists configurations supported by the controller.

Table 1-2 Supported Controller Configurations

core_clk @ Gen{1,2,3,4,5,6} speed (in MHz)		Gen1 Speed Mode[c]	Gen2 Speed Mode	Gen3 Speed Mode	Gen4 Speed Mode	Gen5 Speed Mode	Gen6 Speed Mode	Max[a] Link Width (for each controller[b] Datapath width)					
								32-bit	64-bit	128-bit	256-bit	512-bit	1024-bit
Gen1 Configurations	250	g1_1s	NA	NA	NA	NA	NA	x4	x8	x16	NA	NA	NA
	125	g1_2s	NA	NA	NA	NA	NA	x2	x4	x8	x16	NA	NA
	62.5	g1_4s	NA	NA	NA	NA	NA	x1	x2	x4	x8	NA	NA
Gen2 Configurations	250, 250	g1_1s	g2_2s_dw	NA	NA	NA	NA	x2	x4	x8	x16	NA	NA
	125, 125	g1_2s	g2_4s_dw	NA	NA	NA	NA	x1	x2	x4	x8	x16	NA
	250, 500	g1_1s	g2_1s_df	NA	NA	NA	NA	x4	x8	x16	NA	NA	NA
	125, 250	g1_2s	g2_2s_df	NA	NA	NA	NA	x2	x4	x8	x16	NA	NA
	62.5, 125	g1_4s	g2_4s_df	NA	NA	NA	NA	x1	x2	x4	x8	x16	NA
Gen3 Configurations	250, 250, 250	g1_1s	g2_2s_dw	g3_4s_dw	NA	NA	NA	x1	x2	x4	x8	x16	NA
	125, 125, 125	g1_2s	g2_4s_dw	g3_8s_dw	NA	NA	NA	NA	x1	x2	x4	x8	NA
	250, 500, 1000[d]	g1_1s	g2_1s_df	g3_1s_df	NA	NA	NA	x4	x8	x16	NA	NA	NA
	125, 250, 500	g1_2s	g2_2s_df	g3_2s_df	NA	NA	NA	x2	x4	x8	x16	NA	NA
	62.5, 125, 250	g1_4s	g2_4s_df	g3_4s_df	NA	NA	NA	x1	x2	x4	x8	x16	NA

core_clk @ Gen{1,2,3,4,5,6} speed (in MHz)		Gen1 Speed Mode[c]	Gen2 Speed Mode	Gen3 Speed Mode	Gen4 Speed Mode	Gen5 Speed Mode	Gen6 Speed Mode	Max[a] Link Width (for each controller[b] Datapath width)					
								32-bit	64-bit	128-bit	256-bit	512-bit	1024-bit
Gen4 Configurations	250, 250, 250, 250	g1_1s	g2_2s_dw	g3_4s_dw	g4_8s_dw	NA	NA	NA	x1	x2	x4	x8	NA
	125, 250, 500, 1000[e]	g1_2s	g2_2s_df	g3_2s_df	g4_2s_df	NA	NA	x2	x4	x8	x16	NA	NA
	62.5, 125, 125, 250	g1_4s	g2_4s	g3_8s	g4_8s	NA	NA	NA	x1	x2	x4	x8	NA
	62.5, 125, 250, 500	g1_4s	g2_4s_df	g3_4s_df	g4_4s_df	NA	NA	x1	x2	x4	x8	x16	NA
Gen5 Configurations	62.5, 125, 250, 500, 1000[f]	g1_4s	g2_4s_df	g3_4s_df	g4_4s_df	g5_4s_df	NA	x1	x2	x4	x8	x16	NA
	62.5, 125, 125, 250, 500	g1_4s	g2_4s	g3_8s	g4_8s	g5_8s	NA	NA	x1	x2	x4	x8	NA
	62.5, 125, 250, 250, 500	g1_4s	g2_4s	g3_4s	g4_8s	g5_8s	NA	NA	x1	x2	x4	x8	NA
	125, 250, 250, 500, 1000[g]	g1_2s	g2_2s	g3_4s	g4_4s	g5_4s	NA	x1	x2	x4	x8	x16	NA
	125, 250, 500, 1000[h], 1000[i]	g1_2s	g2_2s	g3_2s	g4_2s	g5_4s	NA	x1	x2	x4	x8	x16	NA
	250, 500, 250, 500, 1000[j]	g1_1s	g2_1s	g3_4s	g4_4s	g5_4s	NA	x1	x2	x4	x8	x16	NA
Gen6 Configurations	125, 250, 250, 500, 1000,1000	g1_2s	g2_2s	g3_4s	g4_4s	g5_4s	g6_8s	NA	x1	x2	x4	x8	x16
	62.5, 125, 250, 500, 1000, 1000	g1_4s	g2_4s	g3_4s	g4_4s	g5_4s	g6_8s	NA	x1	x2	x4	x8	x16

a. The value in each cell indicates the maximum link width supported. You can configure the controller with a link width up to this value, and coreConsultant automatically calculates the datapath width.
b. For license scheme, see the “Checking License Requirements” section in the DWC PCI Express Controller Installation Guide.
c. The g value indicates the speed mode. The s value indicates the number of 8-bit symbols processed per clock cycle per lane by the controller PIPE interface, in the indicated speed mode. The dw/df value indicates how the speed change to that mode is achieved.
d. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.
e. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.
f. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.
g. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.
h. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.
i. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.
j. For more information, see “Synthesis for 1000 MHz Clock Frequency” on page 37.

Table 1-3 Supported Controller Configurations for PIPE SerDes Architecture

core_clk @ Gen{1,2,3,4,5,6} speed (in MHz)	
Gen3 Configurations	125, 125, 125
	125, 250, 500
	62.5, 125, 250
Gen4 Configurations	125, 250, 500, 1000
	62.5, 125, 125, 250
	62.5, 125, 250, 500
Gen5 Configurations	62.5, 125, 250, 500, 1000
	62.5, 125, 125, 250, 500
	62.5, 125, 250, 250, 500
	125, 250, 250, 500, 1000
	125, 250, 500, 1000, 1000
	250, 500, 250, 500, 1000

core_clk @ Gen{1,2,3,4,5,6} speed (in MHz)	
Gen6 Configurations	125, 250, 250, 500, 1000,1000
	62.5, 125, 250, 500, 1000, 1000

1.4.4 Controller-PHY Compatibility

The following tables indicate which combinations of controller configurations from Table 1-2 and PHYs are supported. In tables in this section, the sets of values for each row indicates how the controller moves from Gen1 to Gen2 speed mode, from Gen2 to Gen3 speed mode, from Gen3 to Gen4 speed mode, and from Gen4 to Gen5 speed mode, wherever applicable. The set of values also identifies the particular controller configuration in Table 1-2. The controller always uses the same mechanism (DF or DW) to go from Gen1 to Gen2, from Gen2 to Gen3, from Gen3 to Gen4, and from Gen4 to Gen5. However, some Gen3/Gen4/Gen5 PHYs use a mix of mechanisms and others support Data Throttling (DT) which is indicated by the `dt` postfix.

Table 1-4 Gen6 Controller-PHY Combinations Supported

		PHY	
		g1_2s **g2_2s_df** **g3_4s_dw** **g4_4s_df** **g5_4s_df** **g6_8s_dw**	**g1_4s** **g2_4s_df** **g3_4s_df** **g4_4s_df** **g5_4s_df** **g6_8s_dw**
CONTROLLER	g1_2s g2_2s_df g3_4s_dw g4_4s_df g5_4s_df g6_8s_dw	Yes	No
	g1_4s g2_4s_df g3_4s_df g4_4s_df g5_4s_df g6_8s_dw	No	Yes

Table 1-5 Gen5 Controller-PHY Combinations Supported

		PHY[a]									
		g1_2s g2_2s_df g3_2s_df g4_2s_df g5_2s_df	g1_2s g2_2s_df g3_2s_df g4_2s_df g5_4s_dw	g1_2s g2_2s_df g3_4s_dw g4_4s_df g5_4s_df	g1_4s g2_4s_df g3_4s_df g4_4s_df g5_4s_df	g1_1s_dt2 g2_1s g3_2s_dw g4_4s_dw g5_4s_df	g1_1s g2_1s_df g3_2s_dw g4_2s_df g5_4s_dw	g1_1s g2_1s_df g3_4s_dw g4_4s_df g5_4s_df	g1_1s_dt4 g2_1s_dt2 g3_4s_dt4 g4_4s_dt2 g5_4s	g1_4s g2_4s_df g3_8s_dw g4_8s_df g5_8s_df	g1_4s g2_4s_df g3_4s_df g4_8s_dw g5_8s_df
CONTROLLER	g1_1s g2_1s_df g3_4s_dw g4_4s_df g5_4s_df	✗	✗	✗	✗	✗	✗	✓	b ✓	✗	✗
	g1_2s g2_2s_df g3_2s_df g4_2s_df g5_4s_dw	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
	g1_2s g2_2s_df g3_4s_dw g4_4s_df g5_4s_df	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
	g1_4s g2_4s_df g3_4s_df g4_4s_df g5_4s_df	✗	✗	✗	✓	c ✓	✓	✓	✗	✗	✗

	PHY[a]									
	g1_2s **g2_2s_df** **g3_2s_df** **g4_2s_df** **g5_2s_df**	**g1_2s** **g2_2s_df** **g3_2s_df** **g4_2s_df** **g5_4s_dw**	**g1_2s** **g2_2s_df** **g3_4s_dw** **g4_4s_df** **g5_4s_df**	**g1_4s** **g2_4s_df** **g3_4s_df** **g4_4s_df** **g5_4s_df**	**g1_1s_dt2** **g2_1s** **g3_2s_dw** **g4_4s_dw** **g5_4s_df**	**g1_1s** **g2_1s_df** **g3_2s_dw** **g4_2s_df** **g5_4s_dw**	**g1_1s** **g2_1s_df** **g3_4s_dw** **g4_4s_df** **g5_4s_df**	**g1_1s_dt4** **g2_1s_dt2** **g3_4s_dt4** **g4_4s_dt2** **g5_4s**	**g1_4s** **g2_4s_df** **g3_8s_dw** **g4_8s_df** **g5_8s_df**	**g1_4s** **g2_4s_df** **g3_4s_df** **g4_8s_dw** **g5_8s_df**
g1_4s g2_4s_df g3_8s_dw g4_8s_df g5_8s_df	✗	✗	✓	✓	✗	✗	✗	✗	d ✓	✗
g1_4s g2_4s_df g3_4s_df g4_8s_dw g5_8s_df	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

a. Supports SerDes architecture (CX_PIPE_SERDES_ARCH =1) for the highlighted configurations.
b. DT PHY: doing Gen1 with 4:1 DT, Gen2 with 2:1 DT, Gen3 with 4:1 DT, Gen4 with 2:1, and obtaining Gen5 with no DT.
c. DT PHY: doing Gen1 with 2:1 DT, Gen2 with no DT, Gen3 using DW, Gen4 using DW, and obtaining Gen5 using DF.
d. Support for 64-bit (8s) PIPE is specific to Synopsys; the PIPE specification specifies a maximum of the maximum data width of 32 bits. When CX_PIPERX_MULTI_BLOCK =0, the controller only accepts 64-bit-aligned SKP OS. If you use this configuration, your PHY PCS must follow this requirement. When CX_PIPERX_MULTI_BLOCK =1, blocks can start any 32-bit boundary by extending bit-width of rxstartblock and rxsyncheader (phy_mac_rxstartblock: 2-bit per-lane and phy_mac_rxsyncheader 4-bit per-lane). For more information, see "64-Bit PIPE Receive Operation (CX_PIPERX_MULTI_BLOCK =1)" on page 384.

Table 1-6 Gen4 Controller-PHY Combinations Supported

		PHY[a]								
		g1_1s_dt2 g2_1s g3_2s_dw g4_2s_df	g1_1s_dt2 g2_1s g3_2s_dw g4_4s_dw	g1_1s g2_1s_df g3_2s_dw g4_2s_df	g1_1s g2_1s_df g3_4s_dw g4_4s_df	g1_1s g2_2s_dw g3_4s_dw g4_8s_dw	g1_2s g2_2s_df g3_2s_df g4_2s_df	g1_2s_dt2 g2_2s_dt2 g3_2s g4_2s_df	g1_4s g2_4s_df g3_4s_df g4_4s_df	g1_4s g2_4s_df g3_8s_dw g4_8s_df
CONTROLLER	g1_1s g2_2s_dw g3_4s_dw g4_8s_dw	✗	✗	✗	✗	b ✓	✗	✗	✗	✗
	g1_2s g2_2s_df g3_2s_df g4_2s_df	c ✓	✗	✓	✗	✗	✓	d ✓	✓	✗
	g1_4s g2_4s_df g3_4s_df g4_4s_df	✗	e ✓	✓	✓	✗	✓	✗	✓	✗
	g1_4s g2_4s_df g3_8s_dw g4_8s_df	✗	✗	✗	✗	✗	✗	✗	✓	✓

a. Supports SerDes architecture (CX_PIPE_SERDES_ARCH =1) for the highlighted configurations.
b. Support for 64-bit (8s) PIPE is specific to Synopsys; the PIPE specification specifies a maximum of the maximum data width of 32 bits. When CX_PIPERX_MULTI_BLOCK =0, the controller only accepts 64-bit-aligned SKP OS. If you use this configuration, your PHY PCS must follow this requirement. When CX_PIPERX_MULTI_BLOCK =1, blocks can start any 32-bit boundary by extending bit-width of rxstartblock and rxsyncheader (phy_mac_rxstartblock: 2-bit per-lane and phy_mac_rxsyncheader 4-bit per-lane). For more information, see "64-Bit PIPE Receive Operation (CX_PIPERX_MULTI_BLOCK =1)" on page 384.
c. DT PHY: doing Gen1 with 2:1 DT, Gen2 with no DT, Gen3 using DW, and obtaining Gen4 using DF.
d. DT PHY: doing Gen1 with 2:1 DT, Gen2 with 2:1 DT, Gen3 with no DT, and obtaining Gen4 using DF.
e. DT PHY: doing Gen1 with 2:1 DT, Gen2 with no DT, Gen3 using DW, and obtaining Gen4 using DW.

Table 1-7 Gen3 Controller-PHY Combinations Supported

		PHY[a]							
		g1_1s_dt2 g2_1s g3_2s_dw	g1_1s g2_2s_dw g3_4s_dw	g1_2s g2_2s_df g3_4s_dw	g1_2s g2_4s_dw g3_8s_dw	g1_1s g2_1s_df g3_1s_df	g1_2s g2_2s_df g3_2s_df	g1_2s_dt2 g2_2s_dt2 g3_2s	g1_4s g2_4s_df g3_4s_df
CONTROLLER	g1_1s g2_1s_df g3_1s_df	✗	✗	✗	✗	✓	✓	✗	✗
	g1_1s g2_2s_dw g3_4s_dw	✗	✓	✗	✗	✗	✗	✗	✗
	g1_2s g2_4s_dw g3_8s_dw	✗	✓	✓	b ✓	✗	✗	✗	✗
	g1_2s g2_2s_df g3_2s_df	c ✓	✗	✗	✗	✗	✓	d ✓	✗
	g1_4s g2_4s_df g3_4s_df	✗	✗	✗	✗	✗	✓	✗	✓

a. Supports SerDes architecture (CX_PIPE_SERDES_ARCH =1) for the highlighted configurations.
b. Support of a 64-bit (8s) PIPE is specific to Synopsys; the PIPE specification specifies a maximum of the maximum data width of 32 bits. The controller only accepts 64-bit-aligned SKP OS. If you use this configuration, your PHY PCS must follow this requirement
c. DT PHY: doing Gen1 with 2:1 DT, Gen2 with no DT, and obtaining Gen3 using DW.
d. DT PHY: doing Gen1 and Gen2 with 2:1 DT, Gen3 with no DT.

Table 1-8 Gen2 Controller-PHY Combinations Supported

		PHY				
		g1_1s g2_2s_dw	g1_2s g2_4s_dw	g1_1s g2_1s_df	g1_2s g2_2s_df	g1_4s g2_4s_df
CONTROLLER	g1_1s g2_2s_dw	✓	✗	✓	✗	✗
	g1_2s g2_4s_dw	✓	✓	✓	✓	✗
	g1_1s g2_1s_df	✗	✗	✓	✗	✗
	g1_2s g2_2s_df	✗	✗	✓	✓	✗
	g1_4s g2_4s_df	✗	✗	✓	✓	✓

Table 1-9 Gen1 Controller-PHY Combinations Supported

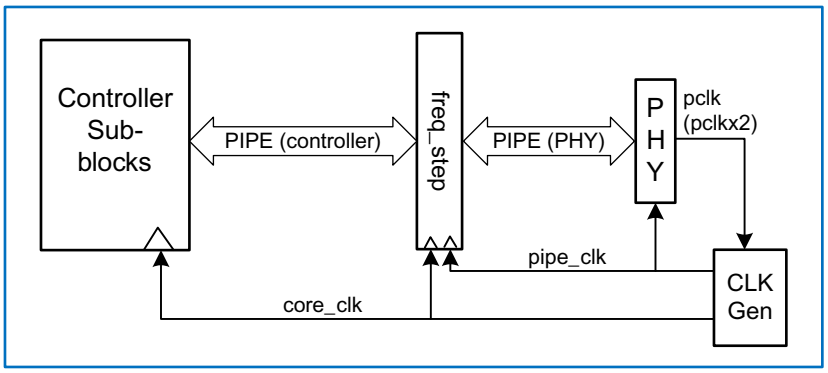
		PHY		
		g1_1s	g1_2s	g1_4s
CONTROLLER	g1_1s	✓	✗	✗
	g1_2s	✓	✓	✗
	g1_4s	✓	✓	✓

1.4.5 Frequency Step Module (CX_FREQ_STEP_EN =1)

When the controller PIPE lane width (`CX_NB`) and PHY PIPE lane width (`CX_PHY_NB`) are different, or when the controller and the PHY have different Gen2 speed modes, then the derived `CX_FREQ_STEP_EN` parameter is defined, and a PIPE adapter module[1] is added between the controller PIPE and the PHY PIPE. If the controller clock frequency is slower than the pipe clock frequency, the derived parameter `CX_FREQ_STEP_DOWN_EN` is defined. If the `core_clk` frequency is faster than the `pipe_clk` frequency, the derived `CX_FREQ_STEP_UP_EN` parameter is defined. In typical configurations (`CX_FREQ_STEP_EN =0`), the `pipe_clk` input in not present.

1. Located in workspace/src/common/freq_step.sv

Figure 1-6 Location of freq_step PIPE Adapter Module



This module steps up/down the signals at the PIPE interface and adjusts the controller PIPE width according to the PHY PIPE width if the `core_clk` frequency is slower or faster than the `pipe_clk` frequency.

1.4.6 Controller-PHY Integration

The PHY module resides outside of the controller, interfacing through the standard PIPE interface. For more information, see "Integrating the Controller with the PHY" in the *User Guide*.

1.5 Deliverables

The deliverables for the PCIe controllers include:

- Synthesizable RTL source code
- Synopsys coreConsultant tool for automated configuration, synthesis, and simulation
- Synopsys PHY simulation model
- A Verilog Testbench (VTB)
  - Allows you to re-execute the example tests on your configuration of the controller for the purpose of sanity checking your generated RTL in coreConsultant environment before moving the RTL to your SoC level verification.
  - Demonstrates connectivity and many types of transfers.
  - VTB is not intended for extensive verification.

Note The VC VIP library license shipped with the controller is intended only to verify the controller interfaces in a specific controller configuration out-of-the-box in the coreConsultant, to confirm that the controller meets the requirements. Do not use this VC VIP library license as a full-featured verification model at the subsystem or SoC level.

- Software Samples and Drivers
  - A Linux driver for PCIe RP controller is available at: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/pci/controller/dwc/pcie-designware.c

    This driver is provided by Synopsys as an example, and is not subject to warranty, support, or maintenance.
  - A software sample of the PCIe controller is available at: https://www.synopsys.com/dw/ipdir.php?c=dwc_pcie_controller_sw_samples

    The DesignWare PCIe IP software sample provides information about how you can use the pre-built images for testing the PCIe controller.

1.5.1 Verification Environment Overview

The verification environment for the controller includes a packaged VTB and separate VIP components. The VTB runs the packaged self-checking regression tests through coreConsultant to validate the configured controller. For more information, see the “Simulating the Controller” section in the *User Guide*.

1.5.2 coreConsultant Overview

The Synopsys coreConsultant tool allows you to configure, simulate, synthesize, and export the controller to your design flow. The final output from the coreConsultant design flow is a set of RTL files (the configured RTL controller) and scripts to allow you run various standalone tests within your own design flow. The coreConsultant features and tasks include:

- A Graphical User Interface (GUI) to guide you through design flow activities.

  All activities may also be performed through a command line interface.

- Interactive parameter selection.

  The coreConsultant tool checks for consistent settings and automatically derives any dependent parameters from your choices.
- Testbench simulation and synthesis configuration consistent with your parameter choices.

1.5.3 Synthesis Overview

The coreConsultant tool is your interface to Synopsys tools for ASIC synthesis of the controller. Synthesis scripts and constraints can be exported from coreConsultant and reused with non-Synopsys tools to synthesize the controller in your own design flow.

1.6 Standards

The PCIe controller implements the following standards:

- *PCI Express Base Specification, Revision 6.0, Version 1.0*

  Access to this specification requires membership in PCI-SIG.

  Download site:

  http://pcisig.com/specifications
- *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*

  Download site:

  http://www.intel.com/technology/pciexpress/devnet/resources.htm
- *PCI Express Card Electromechanical Specification, Revision 4.0*

  Access to this specification requires membership in PCI-SIG.

  Download site:

  https://members.pcisig.com/wg/PCI-SIG/document/download/8250

2

Architecture

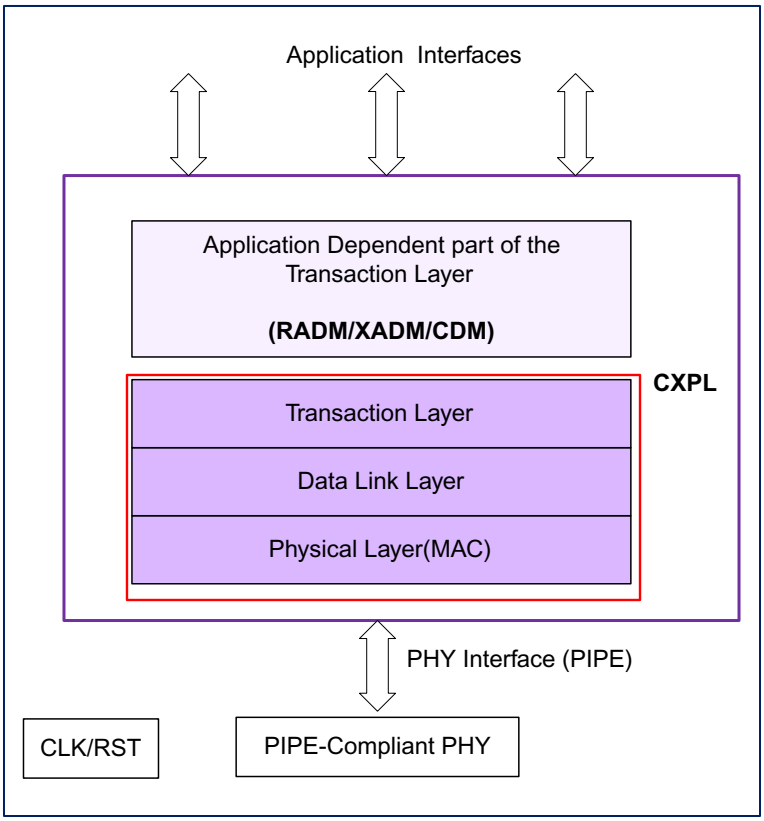
This chapter describes the architecture of the PCI Express controller. The topics in this section are:

- "Overview" on page 56
- "RAM Requirements" on page 64
- "Clock Requirements" on page 68
- "Reset Requirements" on page 90
- "Receive Queues" on page 100

2.1 Overview

The implementation of the PCIe protocol and mode-specific features is split across several modules.

Figure 2-1 PCIe Architecture Overview



- **Common Express Port Logic (CXPL) Module** implements the basic functionality for the PCI Express physical, link, and transaction layers. This module implements a large part 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. For more information, see "Integrating with the PHY in the PCI Express controller" in the *User Guide*.
- **Transmit Application-Dependent Module (XADM)** implements the application-specific functionality of the PCI Express transaction layer for packet transmission. Its functions include:
  - TLP Arbitration
  - TLP Formation
  - Flow Control (FC) Credit checking

  The transmit path uses a cut-through architecture. It does not implement transmit buffering/queues (other than the retry buffer).

  The controller maintains an internal Target Completion Lookup Table to store certain TLP header information from the Rx request. Your application can use this information for transmitting completions later on. For more information, see, "Target Completion Lookup Table" on page 363.

- **Transmit Application Interface Module (XAIM):** This module interfaces with the application for packet transmission. The module forms the PCIe TLP header and optionally implements Address Translation functionality.
- **Receive Application-Dependent Module (RADM)** implements application-specific functionality of the PCI Express transaction layer for packet reception. Its functions include:
  - Sorting/filtering of received TLPs. The filtering rules and routing are configurable.
  - Buffering and queuing of the received TLPs. For more information, see "Receive Queues" on page 100.
  - Routing of received TLP to the controller's receive interfaces.

  The RADM maintains a Receive Completion Lookup Table (LUT) for completion tracking and completion-timeout monitoring of Tx non-posted requests. It indicates a timeout when an expected Rx completion does not arrive within the timeout period.
- **Configuration-Dependent Module (CDM)** implements:
  - Standard PCI Express configuration space
  - Controller-specific register space (Port Logic Registers)

  For more information on the registers implementation in the CDM, see "Register Configuration Space Overview" on page 142.
- **Power Management Controller (PMC)** implements the power management features of the controller.
- **Data Bus Interface (DBI)**

  The DBI module provides a mechanism for a local CPU to access internal registers (in the CDM).
- **Message Generation Module (MSG_GEN)** transmits messages generated by the controller. For more information see, "Messages" on page 193.
- **Hot Plug Control Module (HOTPLUG)** implements logic to generate interrupts on hot plug events.
- **CXS Controller** is used for Coherent Multichip Links (CML).

Table 2-1 Output Signals Used to Notify Hot Plug Events

Mode	Output Signal Used
MSI	hp_msi
MSI-X	hp_msi
INTx	hp_int
PME[a]	hp_pme

a. The controller does not check if the PM state is D1, D2, or D3hot. It is up to your application to check the value on pm_dstate to make sure the device is in D1, D2, or D3hot.

Figure 2-2 RP Controller Block Diagram (without AXI Bridge Module)

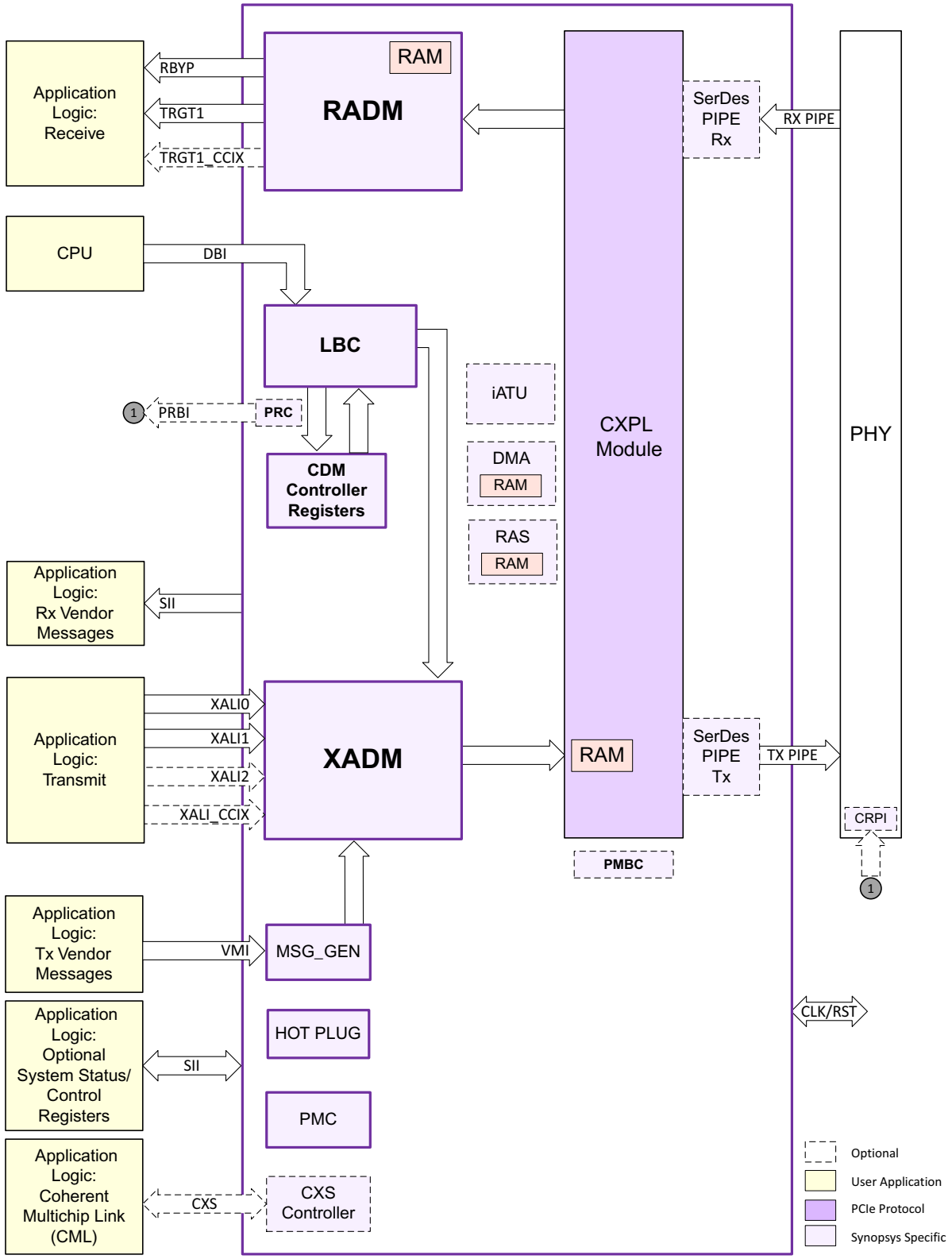


Figure 2-3 RP Controller Block Diagram (with AXI Bridge Module)

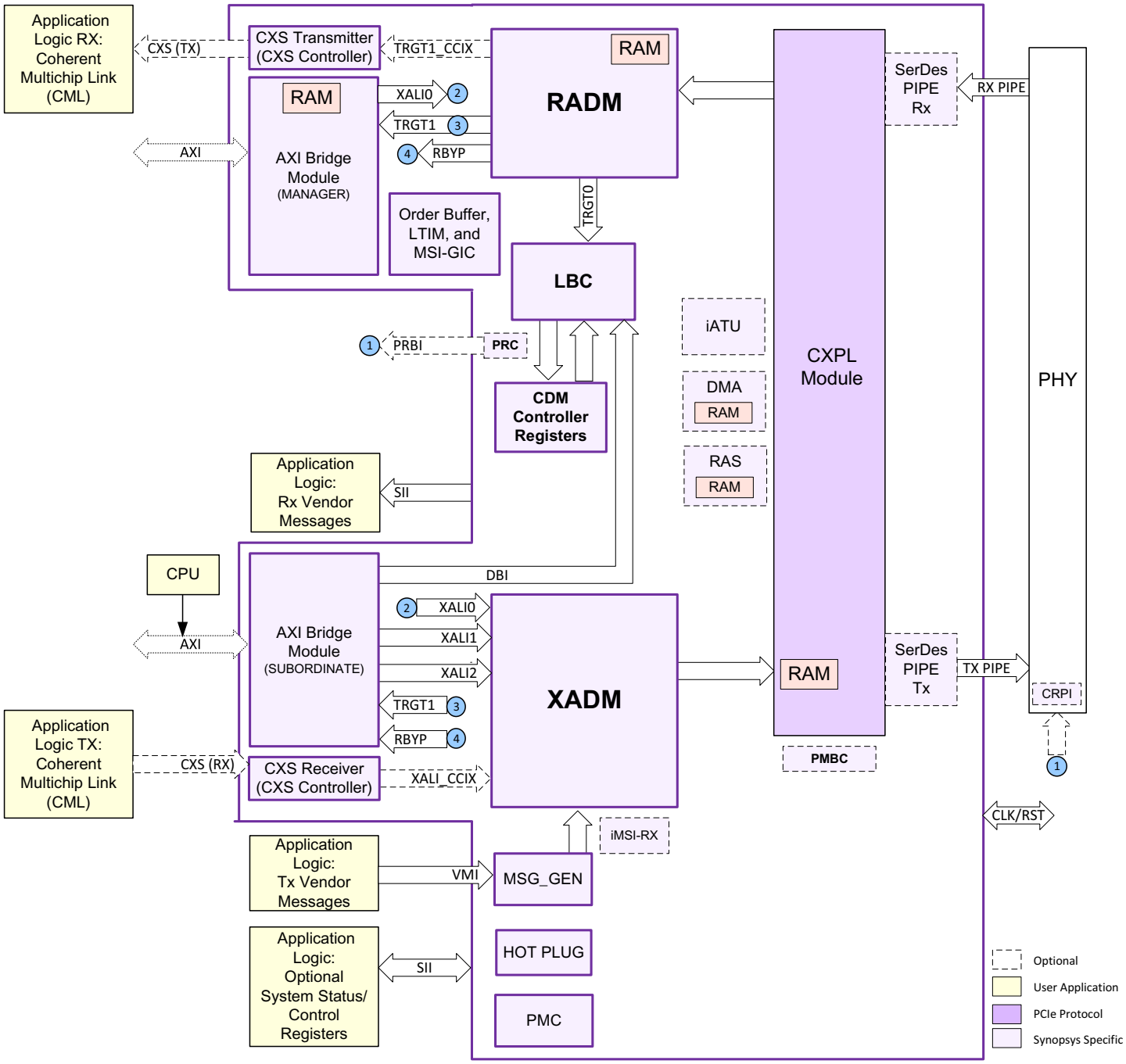


Figure 2-4 Gen6 Configured Controller Block Diagram (Without AXI)

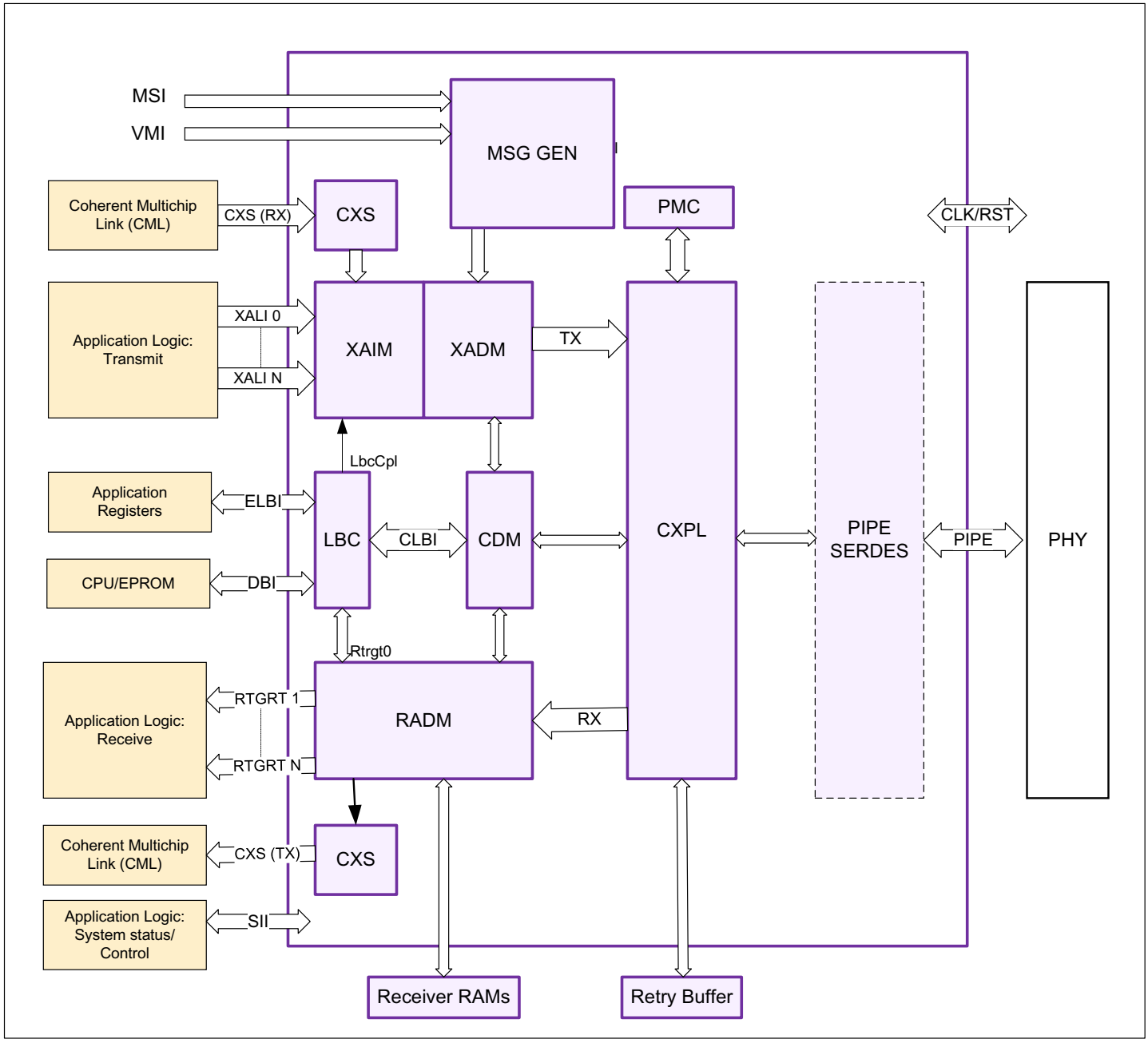


Figure 2-5 Gen6 Configured Controller Block Diagram (With AXI Bridge)

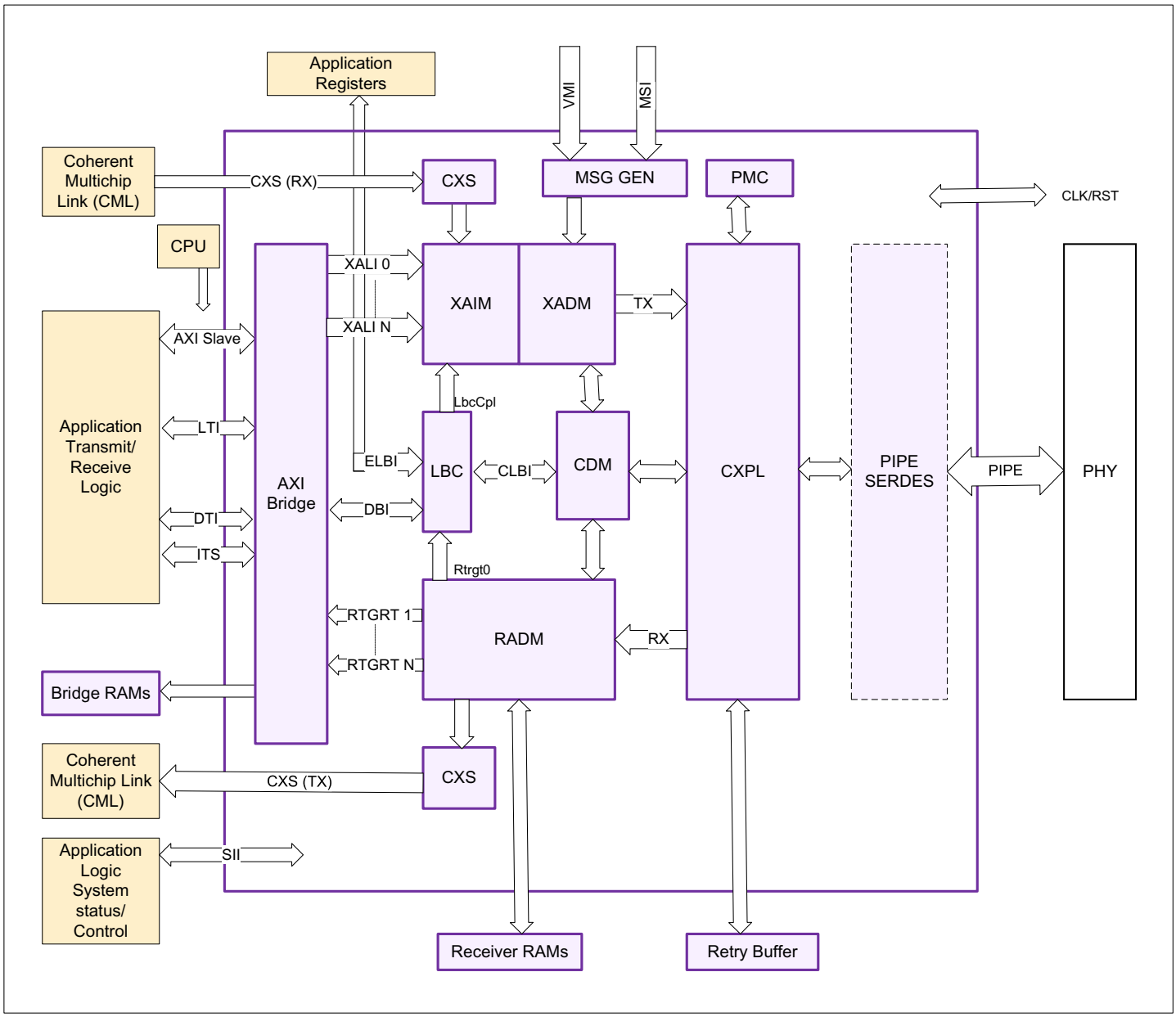


Table 2-2 Controller Interface Summary

Interface	Function
Transmit Client 0 Interface (XALI0)	XALI0 is a transmit interface used by your application for transmission of outbound requests or CPL TLPs. (when you are not using the AXI bridge)
Transmit Client 1 Interface (XALI1)	XALI1 is an additional application transmit interface, identical to XALI0. You can use XALI0 and XALI1 as per your application requirements. For example, your application might use XALI0 exclusively for transmission of outbound requests and XALI1 exclusively for transmission of completions.
Transmit Client 2 Interface (XALI2)	XALI2 is an optional transmit client interface, identical to XALI0 and XALI1. An example usage of the three transmit client interfaces can be, using XALI0 exclusively for posted requests, XALI1 exclusively for non-posted requests, and XALI2 exclusively for completions.
Transmit CCIX Interface (XALI_CCIX)	XALI_CCIX is a transmit interface used by your application for transmission of CCIX TLPs.
Receive Target 1 (TRGT1)/ Receive Bypass Interface (RBYP)	TRGT1 and RBYP are receive interfaces used to connect the native controller and your application (or AXI bridge when used). RBYP is used by queues that are in bypass mode. When you are using the AXI bridge, then these are internal interfaces.
Receive Target 1 CCIX Interface (TRGT1_CCIX)	TRGT1_CCIX receive interface is used to connect the controller and the CCIX logic of your application.
Data Bus Interface (DBI)	DBI is used by your application to access the controller's internal registers in the CDM.You can optionally connect a local CPU or controller to this port.
Message Signaled Interrupt (MSI) Interface	MSI Interface can be used by your application to send MSI requests to the controller, independent of XALI client interfaces.
MSI-X Interface	MSI-X Interface can be used by your application to send MSI-X requests to the controller, independent of XALI client interfaces.
Vendor Message Interface (VMI)	VMI can be used by your application for transmission of vendor messages, independent of the XALI client interfaces.
System Information Interface (SII)	SII is used for exchanging system information between the controller and your application.
PIPE	Standard PIPE interface (*PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*) between the PCI Express PHY and the controller.
PHY Register Bus Interface (PRBI)	This is an optional interface, that facilitates direct access of PHY registers by your application through DBI or CFG requests. **Note:** Only Synopsys PHYs supporting Control Register Parallel Interface, for example Synopsys C8/C10/E12/E16 PHYs can be accessed through this interface.
CXS Interface	CXS in the PCIe controller enables the use of PCIe IP in the implementation of Coherent Multi-chip Links.

Interface	Function
SerDes PIPE	PIPE SerDes functional block implements PIPE 5.1.1 SerDes architecture.

Hint

- TRGT0/TRGT1 refer to the two logical RX interfaces. RTRGT1 is a physical interface.
- TRGT1 is mapped to RTRGT1 in native controller configurations.
- TRGT1 is mapped to mstr_awaddr* in AXI configurations.
- TRGT0 can be mapped to the CDM.

2.2 RAM Requirements

This section covers RAM requirements for the controller. The following topics are discussed:

- "Overview"
- "External RAM Verilog Instances" on page 66
- "ECC Protection Support" on page 67
- "RAM Sizes" on page 67

2.2.1 Overview

- RAMs are either dual-port (write port A and read port B) or one-port (one shared read/write port).
  - In Gen6 configured controller all RAMs are dual-port, except iMSI-RX RAM.
  - In Gen5 and below all RAMs are dual-port, except the retry buffer and iMSI-RX RAMs.
- RAMs must have single-cycle access latency, that is, read data is expected on the next cycle after the address is supplied. The controller supports the RAM types in Figure 2-6 and Figure 2-7.
- RAMs do not have to be initialized.
- When using the UPF flow (`CX_ENHANCED_PM_EN =1`), all RAMs are in the `PD_VAUX` power domain and may be powered-down in L2 and L1.

Note UPF model for RAMs or isolation rules to cover the outputs of the RAMs are not provided when RAMs are powered down. You should ensure that RAM power gating or isolation is handled correctly in your application.

Figure 2-6 Types of Dual-Port RAMs Supported

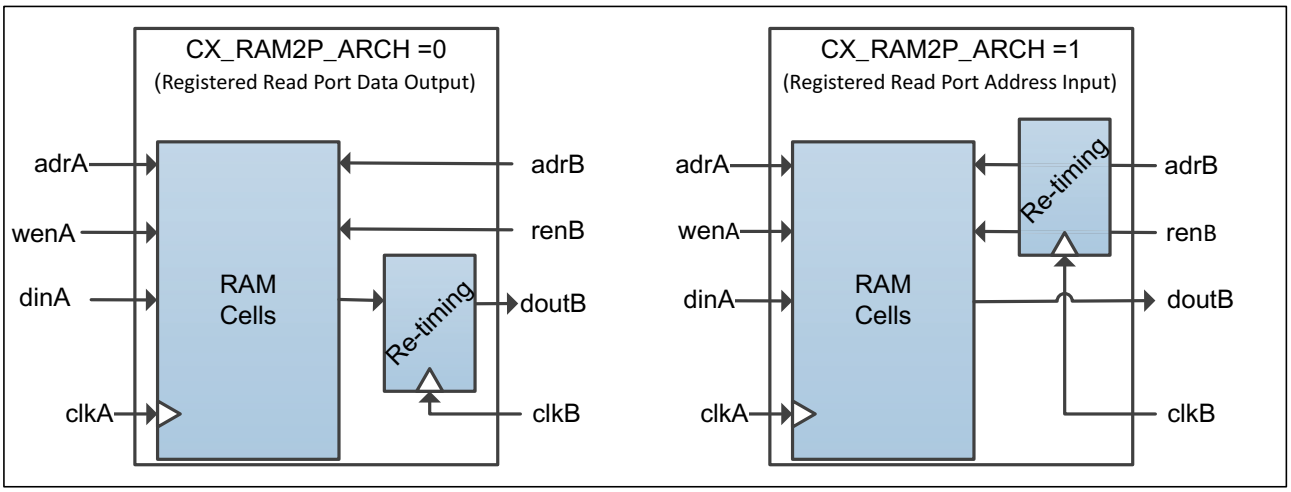
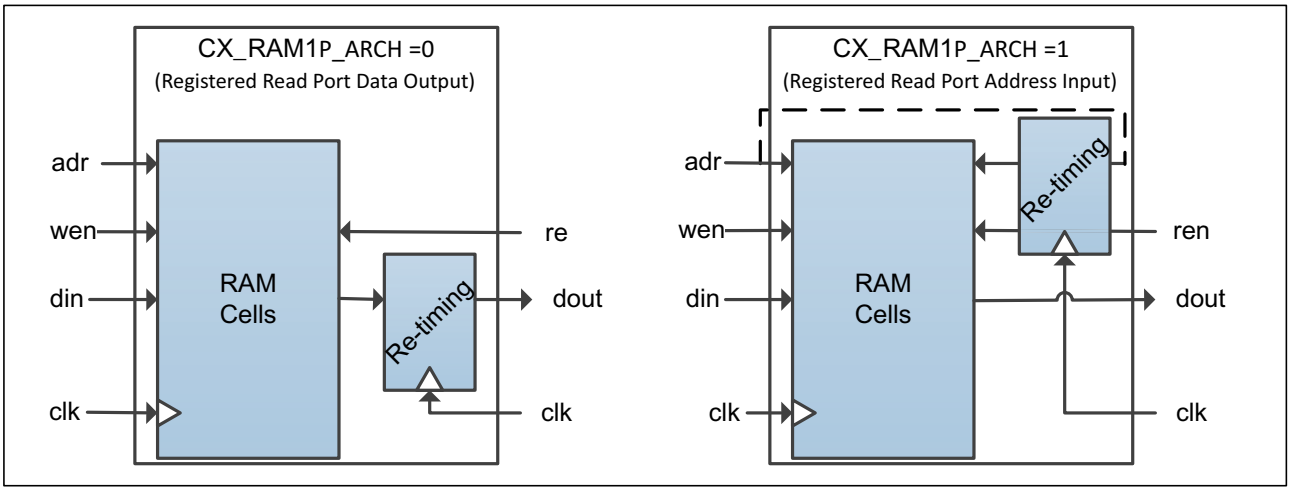


Figure 2-7 Types of One-Port RAMs Supported



RAM Implementation

Each module that uses RAM instantiates one of the following RAM wrappers:

- `ram_1p_wrapper.v` (for one-port RAM)
- `ram_1p_be_wrapper.v` (for one-port RAM with Byte Enables)
- `ram_2p_1c_wrapper.v` (for dual-port RAM with single clock)
- `ram_2p_2c_wrapper.v` (for dual-port RAM with two clocks)

Each RAM wrapper instantiates one of the following FPGA synthesizable simulation RAM models with the required width and depth parameters:

- `ram1p.v` (for one-port RAM)
- `ram2p.v` (for dual-port RAM)

In addition, each RAM wrapper instantiates a module to add RAM Read Data Latency to ease timing closure.

Table 2-3 shows the RAM wrappers and RAM models location in the your <workspace>.

Table 2-3 RAM Wrappers and Models Location

RAM Wrappers	RAM Models
`examples/rams/`	`examples/rams/`

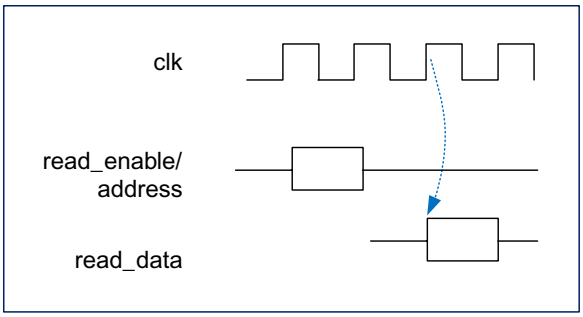
Note For ASIC implementation, you must instantiate your technology-specific RAM models from your RAM vendor inside the RAM wrappers.

RAM Timing

The RAM read cycle latency is the delay between the time at which the *read address* is presented by the controller to the RAM, and the time at which the *read data* is presented by the RAM to the controller. The default read access time of the RAM is one clock cycle.

To ease timing closure at high clock frequencies, external pipelines on the read address path or the read data path can be added to the RAM read path. You can configure the appropriate RAM read latency parameter (for more details on RAM read latency parameters, see "RAM Read Data Latency" section in the "Advanced RAM Config Parameters" table of the "Parameter Descriptions" chapter of the *Reference Manual*) so that the controller can sample the read data correctly. These parameters specify the number of cycles after which the controller expects RAM read data to appear. For example, if the AXI Manager Completion Buffer RAM read data is externally re-timed, then you can set `CC_MSTR_CPL_SEG_BUF_RAM_RD_LATENCY =2`.

Figure 2-8 RAM Read Interface Timing (CC_MSTR_CPL_SEG_BUF_RAM_RD_LATENCY =2)



Note By default, the AXI bridge registers RAM interfaces to facilitate timing closure. To reduce RAM write/read latencies, pipeline stages on the write/read control interface signals can be removed, subject to closing timing on these interfaces. Contact Synopsys support through SolvNetPlus, if the default RAM write/read latencies are of concern.

You can also set the following parameters to place a re-timing register at the output of the RAMs.

- `CX_RETRYSOTRAM_REGOUT`
- `CX_RETRYRAM_REGOUT`

The read data output of the Rx Queue RAMs is always re-timed/registered by the controller.

You can set the following parameter to place a register at the boundary between the Rx queue manager outputs and the Rx queue RAM inputs.

- `CX_RADM_INQ_MGR_REGOUT`

After you configure the `RAM1P_RD_ACCESS`, `RAM1P_ADDR_SU`, `RAM2P_RD_ACCESS`, and `RAM2P_ADDR_SU` parameters, the coreConsultant tool generates the timing requirements. For more information on accessing synthesis scripts and SDC, see "Synthesizing to a Device Outside of coreConsultant" in the "Exporting a Controller From coreConsultant to Your Chip Design Database" section of the *User Guide*.

2.2.2 External RAM Verilog Instances

For external RAMs, all RAM wrappers are instantiated within the following module:

`<workspace>/examples/pcie_iip_rams.v.`

2.2.3 ECC Protection Support

For more information, see "RAS Data Protection (RAS DP)" on page 164.

2.2.4 RAM Sizes

The address and data widths of the memory interfaces are automatically derived based on other design parameters. The coreConsultant tool generates a preliminary RAM report for your configuration in `workspace/report/RamSizes.html`.

Figure 2-9 Example of RAM Report

RAM name	Width	Depth	Address width	Byte Enables	Clock Domains	Instance name
AXI Master Completion Buffer RAM	303	640	10	No	core_clk	u_ib_mcpl_sb_ram
AXI Master Completion Asynchronous Clock Crossing FIFO RAM	415	16	4	No	mstr_aclk_gated core_clk	u_ib_mcpl_a2c_cdc_ram
AXI Master Non-Posted Read Request Buffer RAM	256	64	6	No	core_clk	u_ib_rreq_ordr_ram
AXI Master Posted Tracker Data Buffer RAM	144	32	5	No	core_clk	u_ib_wreq_ptrk_data_ram
AXI Master Posted Tracker Header Buffer RAM	345	16	4	No	core_clk	u_ib_wreq_ptrk_hdr_ram
AXI Master Read Queue Request Formation CDC Buffer RAM	264	16	4	No	core_clk mstr_aclk_gated	u_ib_rreq_c2a_cdc_ram
AXI Master Write Queue Request Formation CDC Buffer RAM	593	16	4	No	core_clk mstr_aclk_gated	u_ib_wreq_c2a_cdc_ram
AXI Slave TLP Completion Data Segment Composer FIFO RAM	303	640	10	No	core_clk slv_aclk_gated	u_ob_ccmp_data_ram
AXI Slave Non-Posted TLP Decomposer Header/Data FIFO RAM	284	64	6	No	slv_aclk_gated core_clk	u_ob_npdcmp_ram
AXI Slave Non-Posted Write Set-Aside Buffer RAM	239	16	4	No	slv_aclk_gated	u_slv_npw_sab_ram
AXI Slave Posted TLP Decomposer Data FIFO RAM	288	24	5	No	slv_aclk_gated core_clk	u_ob_pdcmp_data_ram
AXI Slave Posted TLP Decomposer Header FIFO RAM	244	16	4	No	slv_aclk_gated core_clk	u_ob_pdcmp_hdr_ram
RAS D.E.S. Event Counter RAM	64	19	5	No	muxd_aux_clk_g	u_ram_cdm_rasdes_ec_reg
RAS D.E.S. Time Based Analysis RAM	64	4	2	No	muxd_aux_clk_g	u_ram_cdm_rasdes_tba_reg
All Data Receive Queue RAM	151	1247	11	No	radm_clk_g	u3_ram_radm_qbuffer_data
All Header Receive Queue RAM	235	262	9	No	radm_clk_g	u0_ram_radm_qbuffer_hdr
Transmit Retry Buffer	168	638	10	No	core_clk	u_ram_1p_rbuf
Transmit 'Start Of TLP' (SOT) Buffer	16	1024	10	No	core_clk	u_ram_2p_sotbuf

2.2.5 Validating Multi-bit Read Enable

You can use the following output signals to identify the RAM portion read currently. These signals exist only when `ARC_256_P2` and `ARC_256` parameters are defined.

- `p_alt_hdrq_enb` -read enable for header queue buffers
- `p_alt_dataq_enb` -read enable for data queue buffers

The controller reads only one of the queue buffers - header or data, at a time and consequently asserts only one of the read enables at a time. This mechanism ensures that empty RAM is not read.

2.3 Clock Requirements

This section covers the following clock generation topics for the controller, focusing on conventional PCIe.

- "Overview"
- "General Clock Relationships" on page 70
- "Gen2, Gen3, Gen4, or Gen5 Speed Changing Considerations (CX_FREQ_STEP_EN =1)" on page 73
- "AXI Bridge Clocking" on page 74
- "Removing the Reference Clock" on page 77
- "Additional Information" on page 82
- "Clock Generation Module" on page 83

2.3.1 Overview

The PCIe controller requires the clocks outlined in Table 2-4.

Table 2-4 Generation of Individual Clock Signals by Your Clock Generation Logic

Clock	Notes
core_clk	■ This is the primary clock for the controller. ■ It can have a frequency of 62.5, 125, 250, 500, or 1000 MHz. For more information on supported clock frequencies and datapath widths, see "Frequency, Speed, and Width Support" on page 36. ■ The `core_clk` is derived from the `pipe_clk` output of the PHY (whose TX PLL uses the platform reference clock `REFCLK`). ■ The `core_clk` signal must be gated-off when `muxd_aux_clk` is being switched to run at low frequency.
ret_core_clk	■ The gated version of the `core_clk` for modules retained during L1 power gating. ■ The `ret_core_clk` frequency is the same as `core_clk`. ■ You can gate this clock during the L1 and L2 low power states under the same conditions as `core_clk`.
core_clk_ug	■ An ungated version of `core_clk` (called `core_clk_ug`) exists because a small number of controller registers must receive `core_clk` during switching to-and-from low-power states. ■ When `DWC_pcie_clkrst.v` has switched off `core_clk`, the controller uses `core_clk_ug` to detect and latch the PHY status from the pipe interface to complete the transition to the low-power state. After switching to the low-power state has completed, `core_clk_ug` might not be active because the PHY might stop providing `pipe_clk` (from which `core_clk` and `core_clk_ug` are derived).

Clock	Notes
aux_clk	■ This clocks the PMC. ■ It is derived from the platform AUX clk by `DWC_pcie_clkrst.v` which drives the controller's `aux_clk` input through the `muxd_aux_clk` wire. ■ When the controller output `pm_sel_aux_clk` is set to 1 the source of `aux_clk` should be switched to the low frequency clock and `core_clk` should be gated off. For more information, see "Removing the Reference Clock" on page 77. ■ When exiting L1/P2 you should increment the L1 Exit Latency in the Link Capabilities register by two or three `muxd_aux_clk` periods to allow time for `DWC_pcie_clkrst.v` to switch `muxd_aux_clk` back to `core_clk`. ■ The `aux_clk` and `core_clk` clocks must be completely synchronous during normal operation. You must ensure that the `aux_clk` and `core_clk` clock trees are balanced during layout. There are no lower or upper limits on the frequency of `aux_clk` unless you are using the "L1 Substates" on page 266. ■ The controller uses `aux_clk` for counting time during L1 substates. You must program the frequency of this clock into the `AUX_CLK_FREQ_OFF` register with a value in MHz to count real time. Frequencies lower than 1 MHz are possible, but with a loss of accuracy in the time counted.
aux_clk_g	■ This is a gated version of `aux_clk` that is used to clock the CDM registers and the LBC/DBI.
radm_clk_g	■ This is a gated version of `core_clk` that is used to clock the RADM (Rx filter and queues). ■ The `DWC_pcie_clkrst.v` module uses the `en_radm_clk_g` controller output[a] to enable `radm_clk_g`.
slv_aclk/ mstr_aclk/dbi_aclk	■ The AXI bridge assumes that its clocks are asynchronous to `core_clk` and implements the appropriate clock crossings through gray-encoded asynchronous FIFOs. ■ The upper frequency limit for the AXI clocks is set by the synthesis technology library that you use. ■ There are no known functional restrictions on the AXI clock to `core_clk` clock ratio. However, the maximum and minimum simulation-verified clock ratios are 16:1 and 1:16 respectively. ■ When your AXI clocks are synchronous to `core_clk`, you can eliminate all unnecessary clock crossing circuitry by disabling the `SLV_CLK_DIFF_ENABLE` and `MSTR_CLK_DIFF_ENABLE` configuration parameters. ■ You should take care when deriving these clocks from `pipe_clk` or `core_clk` if you are using a dynamic frequency Gen2, Gen3, Gen4, or Gen5 configuration. In these cases, the frequency of `pipe_clk` or `core_clk` changes across speed modes. In these cases, your AXI clock cannot be synchronous to `core_clk` and you must not set `SLV_CLK_DIFF_ENABLE` or `MSTR_CLK_DIFF_ENABLE` to '0'. ■ To save power when the AXI bridge is in low-power mode, you can gate off `slv_aclk`, `mstr_aclk`, or `dbi_aclk` when `slv_aclk_active`, `mstr_aclk_active`, or `dbi_aclk_active` is not asserted respectively. For more information, see "Clock Gating" on page 75. The ungated versions (*_ug) must never be gated. ■ `slv_aclk, mstr_aclk, dbi_aclk` are completely asynchronous; there are no signals passing between these clock domains.
slv_aclk_ug/ mstr_aclk_ug/ dbi_aclk_ug	■ These are ungated versions of `slv_aclk`, `mstr_aclk`, and `dbi_aclk`. ■ The ungated clocks are used by the AXI bridge to generate control signals `slv_aclk_active`, `mstr_aclk_active`, or `dbi_aclk_active` which gates `slv_aclk`, `mstr_aclk`, or `dbi_aclk` respectively. ■ You can de-assert `slv_aclk_active`, `mstr_aclk_active`, or `dbi_aclk_active` to gate off `slv_aclk`, `mstr_aclk`, or `dbi_aclk` respectively. For more information, see "Clock Gating" on page 75.

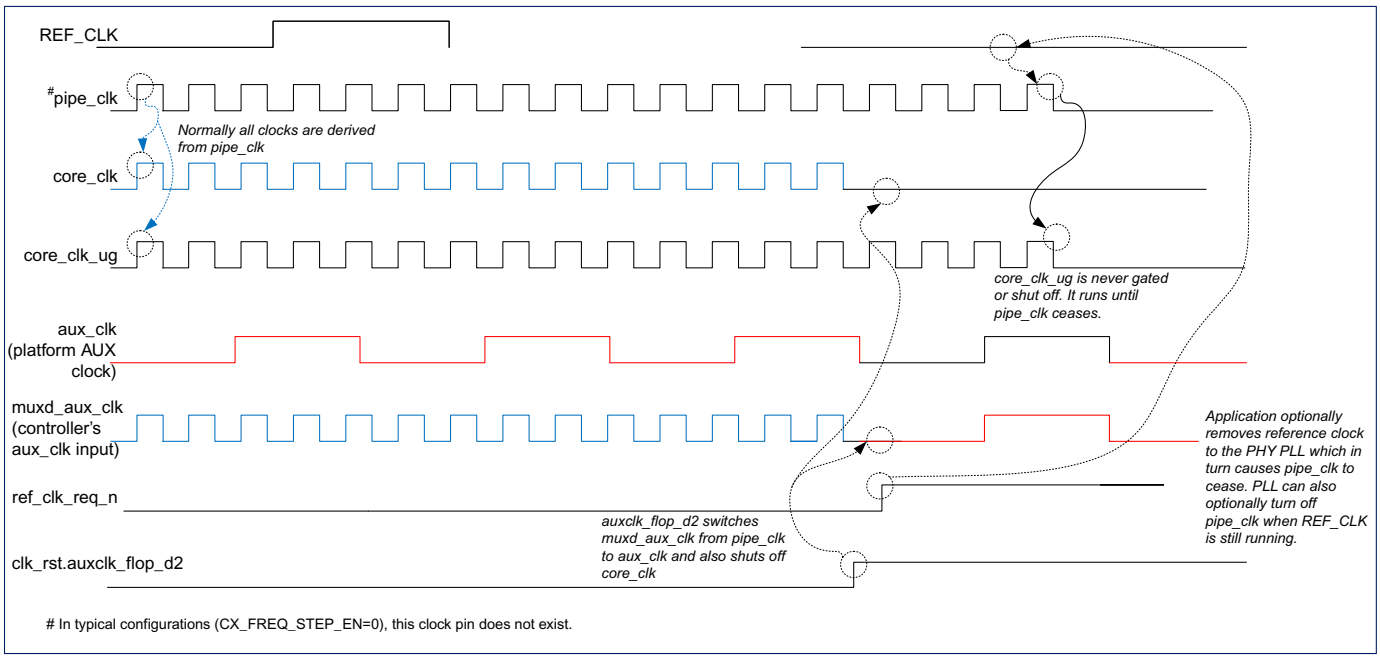
Clock	Notes
pipe_clk	■ In typical configurations (`CX_FREQ_STEP_EN =0`), this clock pin does not exist. ■ The `pipe_clk` input is used by the controller to clock the PIPE. ■ It can have a frequency of 62.5, 125, 250, 500, or 1000 MHz. ■ The PHY TX PLL generates `pipe_clk` from the platform reference clock. The PHY adapts and re-times RX data to `pipe_clk`. ■ The `core_clk` and `pipe_clk` clocks must have the same phase and have frequencies that are integer multiples of each other. ■ In normal operation (L0/D0), `core_clk` and `core_clk_ug` are derived from `pipe_clk`.
en_phy_reg_clk_g	■ Clock enable for `phy_reg_clk_g`. ■ When `en_phy_reg_clk_g` clock signal is 1, `phy_reg_clk_g` becomes active. ■ This clock exists only when `CX_PHY_VIEWPORT_ENABLE =1`.
phy_reg_clk_g	■ This clock is gated by `en_phy_reg_clk_g`. ■ This clock exists only when `CX_PHY_VIEWPORT_ENABLE =1`. **Note:** For Synopsys PHYs supporting Control Register Parallel Interface, this clock should be used as `cr_para_clk`, if the PHY registers are accessed through PRBI.
cxs_clk	■ This clocks the CXS Interface ■ `cxs_clk` is driven by `core_clk` when `CX_CXS_SYNCHRONOUS =1`
pipe_rx_clk	■ This clocks the SerDes PIPE per lane. ■ This clock exists only when `CX_PIPE_SERDES_ARCH =1`

a. The controller de-asserts this signal when there is no Rx traffic, Rx queues and pre/post-queue pipelines are empty, RADM completion LUT is empty, and there are no FLR actions pending. You must set the RADM_CLK_GATING_EN field in the CLOCK_GATING_CTRL_OFF register to enable this functionality; otherwise the en_radm_clk_g output is always be set to ‘1’.

2.3.2 General Clock Relationships

Figure 2-10 indicates the approximate relationships between the different clocks for the “Clock Generation Module” on page 83. It is not cycle accurate and is given for the purpose of illustration only. You must simulate the controller to obtain precise timings. For the purposes of simplicity, the diagram shows core_clk being derived from pipe_clk without division. For the meaning of signal labels, see “Clock Generation Module” on page 83.

Figure 2-10 Clock Waveforms with External PHY (Not Cycle Accurate)



2.3.2.1 Synchronous To Attribute

In the "Signal Descriptions" chapter of the *Reference Manual*, the "`Synchronous To:`" attribute indicates which controller clocks sample an input (or drive an output) when considering *all possible configurations*.

- In many cases an output has components driven (or an input sampled) by multiple clocks.
- The attribute lists all clocks which drive or sample for all possible configurations. It is an automatically generated list accumulated over all configurations.
- Your particular configuration might not have all of the clocks listed. If you have access to the Spyglass tool, then you can generate a configuration-specific report using the process outlined in the "Running Spyglass" section of the *User Guide*.
- When there is only one clock in the list, this clock is normally but not always the same as the clock that your application logic should use to clock (sample/drive) this pin.
- When there is more than one clock in the list:
- A {} is placed around the clock which is used in generating the synthesis in/out timing delay constraints and which is also used in Spyglass boundary CDC checking.
- In most cases, this is also the clock that you should use to drive/sample the pin in normal operational (non low-power) mode.
- If more[1] than one clock has a {}, that clock is configuration-dependent, and you should look in your SDC file (as explained in the "Synthesizing to a Device Outside of coreConsultant" section of the *User Guide*) to see which one is relevant.

1. Normally only applies to the PIPE outputs which are clocked on pipe_clk or core_clk depending on CX_FREQ_STEP_EN.

Figure 2-11 Snippet from Signal Descriptions Chapter Showing Synchronous To

client0_tlp_func_num[(PF_WD-1):0]	I	Indicates from which physical function (PF) the request is coming. The controller uses client0_tlp_func_num to form the requester ID for requests and the completer ID for completions. Function numbering starts at '0'. For a single-function device, set client0_tlp_func_num to 000b. **Exists:** !(AMBA_INTERFACE!=0) **Synchronous To:** aux_clk_g,{core_clk},None **Power Domain:** PD_VMAIN_SW **Active State:** High **Registered:** N/A **Validated by:** client0_tlp_hv is asserted

- When you see "perbitclk" in the clock list, it indicates that there is a different clock for some of the bits in the multi-bit port, and/or that some of the bits in a multi-bit port falls into the various categories as defined next by "None".
- When you see "None_as" in the clock list, it indicates the following and is equivalent/mapped to `core_clk` for CDC synthesis.
  - Asynchronous reset with synchronous de-assertion.
- When you see "None" in the clock list, it indicates any of the following[1] and is equivalent/mapped to `core_clk` for CDC and synthesis.
  - Direct or combinatorial feed-through.
  - Gated-off (that is, unclocked) inputs.
  - Unused inputs.
  - Hard-coded outputs.
  - Asynchronous outputs.
  - Asynchronous resets.

Note

For asynchronous inputs (excluding asynchronous resets), CDC synchronizers are added in the RTL. For more information, see "Port Clocking and Input Synchonizers" on page 680

- Asynchronous inputs cannot be identified by the "Synchronous To" attribute. This attribute identifies the sampling clock (synchronous or asynchronous).
- Asynchronous outputs are indicated with "Synchronous To: None".

Generalized Interface Clocking Guidelines Qualifying the Synchronous To Attribute

- All clocks (except AXI) are quasi-synchronous and depending on the situation, it is not incorrect to use any of them.
- The presence of `aux_clk` (not `aux_clk_g`) implies a connection to a low-power feature and where you see it listed, you can make use of that low-power feature by sampling/driving on `aux_clk`.
- **CDM Outputs (cfg_*/cdm_*)**
  - All are clocked out by `aux_clk_g` which runs slow in low-power mode.

1. In at least one of all possible configurations.

  - You might want to sample some of these outputs during L1 and take action.
  - However, it is not expected that you have your complete application running in low-power mode (on `aux_clk_g`) because this might result in higher and unnecessary power consumption.

- **Client Interfaces (client0/1/2_*)**
  - When driven from the `aux_clk` domain the `client0_tlp_hv` pin triggers exit from low power as per "L1 Operation (Non-substates)" on page 284.
  - The whole interface (all `client0_tlp_*` inputs) must be driven from the same clock because header info and so on, needs to be provided at the same time as `client0_tlp_hv`.
  - An alternative method to trigger L1 low-power exit is to use and clock `xfer_pending` on `aux_clk` and to drive the `client0_tlp_*` interface on `core_clk`.

- **TRGT1 Interface (trgt1_*)**
  - The controller clocks this interface on `radm_clk`. However, there are some status paths going back to the CDM (`aux_clk_g` and `core_clk`).
  - If your application interface is receiving requests and generating responses (on client0_tlp_*), it recommended to use `core_clk` to sample this interface and drive the response interface.
  - If your application interface is only receiving requests, it recommended to use `radm_clk` to sample this interface.

- **PIPE Interface (mac_phy_*/mac_phy_*)**
  - `pipe_clk` indicates a frequency step[1].
  - The PIPE interface also drives/samples logic on `aux_clk` for low-power entry and exit.

2.3.3 Gen2, Gen3, Gen4, or Gen5 Speed Changing Considerations (CX_FREQ_STEP_EN =1)

When the PHY is dynamic frequency and the controller is dynamic width, `pipe_clk` changes frequency when moving between modes. However, `core_clk` should not change frequency. Your synthesized controller logic operates at a fixed frequency, so it is imperative that your external clock generation and switching logic (see `DWC_pcie_clkrst.v`) never generates `core_clk` with a higher frequency during mode switching. The controller uses the `mac_phy_rate` output to negotiate the link data rate and waits for a pulse on the `phy_mac_phystatus` input to confirm that the PHY has accepted the requested rate. However, you must carefully consider the following facts when designing the logic that controls the `core_clk` switch selection between `pipe_clk` and the divided `pipe_clk`:

- The PHY can start generating `pipe_clk` at the newer frequency several clock cycles before it asserts `phy_mac_phystatus`.
- A typical anti-glitch clock switch does not switch the clock source until several clock cycles after it has been instructed to do so.

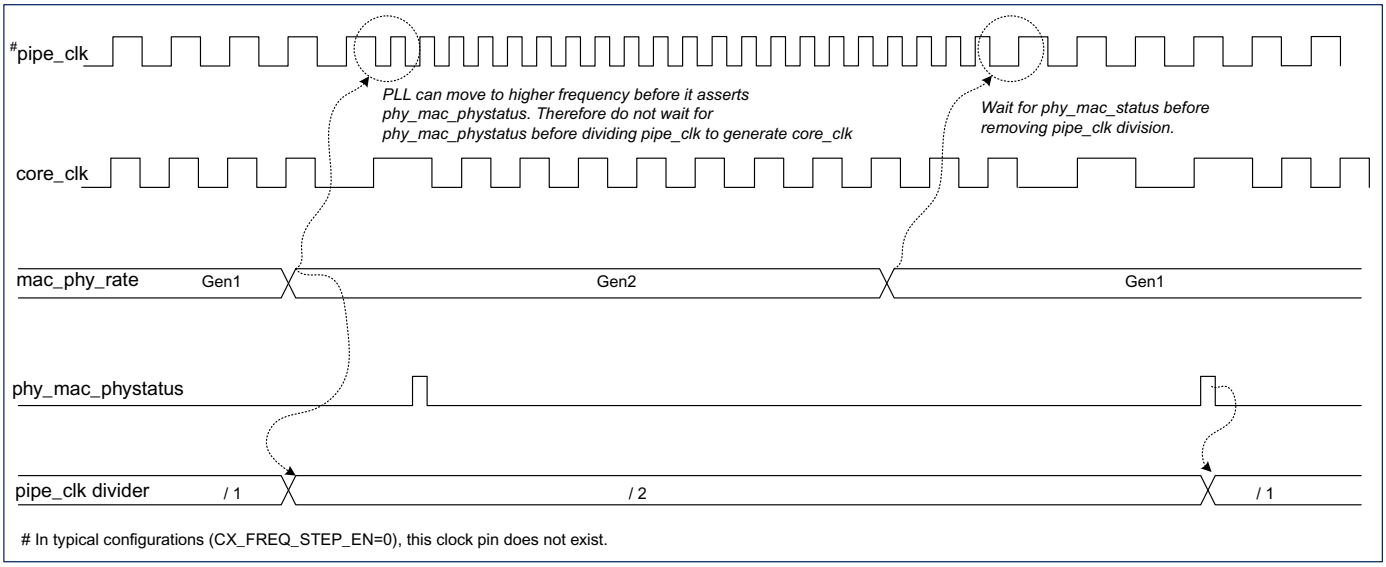
Therefore, your external clock generation and switching logic should:

- Start switching as soon as `mac_phy_rate` changes if moving to a faster speed mode. That is, do not switch too late.
- Delay switching until `phy_mac_phystatus` acknowledges `mac_phy_rate` if moving to a slower speed mode. That is, do not switch too early to ensure that `core_clk` does not have a frequency higher than what the controller is synthesized for.

1. PIPE outputs which are clocked on pipe_clk or core_clk depending on CX_FREQ_STEP_EN.

The `DWC_pcie_clkrst.v` module implements this feature.

Figure 2-12 Recommended Clock Divider Switching Points During Speed Changes (Not Cycle Accurate)



The controller uses `core_clk` to sample the `phy_mac_phystatus` input which the PHY generates at `pipe_clk`. Therefore when the PHY is dynamic frequency and the controller is dynamic width (that is, when `core_clk` is not equivalent to `pipe_clk`), you must ensure that:

- `core_clk` is toggling when the PHY updates `phy_mac_phystatus` for a speed change

OR

- your external logic holds `phy_mac_phystatus` (for the speed change) until `core_clk` toggles again.

2.3.4 AXI Bridge Clocking

The AXI bridge module has five[1] clock domains.

- `mstr_aclk`: AXI manager interface clock.
- `slv_aclk`: AXI subordinate interface clock.
- `dbi_aclk`: AXI Dedicated DBI slave interface clock when the dedicated DBI slave interface is present.
- `core_clk`: PCIe controller clock.
- `lti_clk`: LTI clock.

The AXI bridge assumes that `mstr_aclk`, `slv_aclk`, `lti_aclk`, and `dbi_aclk` are asynchronous to `core_clk` and it implements the appropriate clock crossings using pre-existing FIFOs and inserting extra FIFOs when necessary. For more information, see "Advanced Information: Clock and Data Crossing (CDC)" on page 675. When your AXI clocks are synchronous to `core_clk` then you can eliminate unnecessary clock crossing circuitry by disabling the `SLV_CLK_DIFF_ENABLE` and `MSTR_CLK_DIFF_ENABLE` configuration parameters.

1. There are also ungated versions of the mstr_aclk, slv_aclk, dbi_aclk, and lti_clk clocks.

Caution

Gen2/Gen3/Gen4/Gen5 Dynamic Frequency

When you are using a dynamic frequency Gen or Gen3 controller, the frequency of `core_clk` changes across speed modes. In these cases, your AXI clock cannot be synchronous to `core_clk`. You cannot set `SLV_CLK_DIFF_ENABLE` or `MSTR_CLK_DIFF_ENABLE` to '0'.

The upper frequency limit for the AXI clocks is set by synthesis technology library that you use. There are no known functional restrictions on the AXI clock to `core_clk` clock ratio. However, the maximum and minimum simulation-verified clock ratios are 16:1 and 1:16 respectively. AXI clock can be changed dynamically during normal operation, without the need of reset.

2.3.4.1 Clock Gating

To save power when the AXI bridge is in low-power mode[1], you can gate off `slv_aclk`, `mstr_aclk`, `lti_aclk`, or `dbi_aclk` when `slv_aclk_active`, `mstr_aclk_active`, `lti_aclk_active`, or `dbi_aclk_active` is de-asserted. For more information, see Figure 2-13 and Figure 2-14 on page 76. The ungated versions (`mstr_aclk_ug`, `slv_aclk_ug`, `lti_aclk_ug`, or `dbi_aclk_ug`) must never be gated because they are used to clock the AXI low-power FSM.

The controller de-asserts `mstr_aclk_active`, on completion of all the incoming requests. The AXI bridge asserts `mstr_aclk_active`, on receiving a new transaction from the wire.

When DMA is enabled, the controller also waits for all the DMA transfers to complete, before de-asserting `mstr_aclk_active`. The AXI bridge asserts `mstr_aclk_active` on either a DMA doorbell or on receiving a new transaction from the wire.

The controller de-asserts `slv_aclk_active` when your application does not intend[2] to initiate any traffic and all the transactions previously forwarded to the remove device have completed. It asserts `slv_aclk_active` when your application asserts `slv_*valid`. The operation of `dbi_aclk_active` is similar to that of `slv_aclk_active`.

1. AXI bridge enters low power mode after the native controller enters low-power mode.
2. slv_*valid de-asserted.

Figure 2-13 AXI Clock Gating Overview

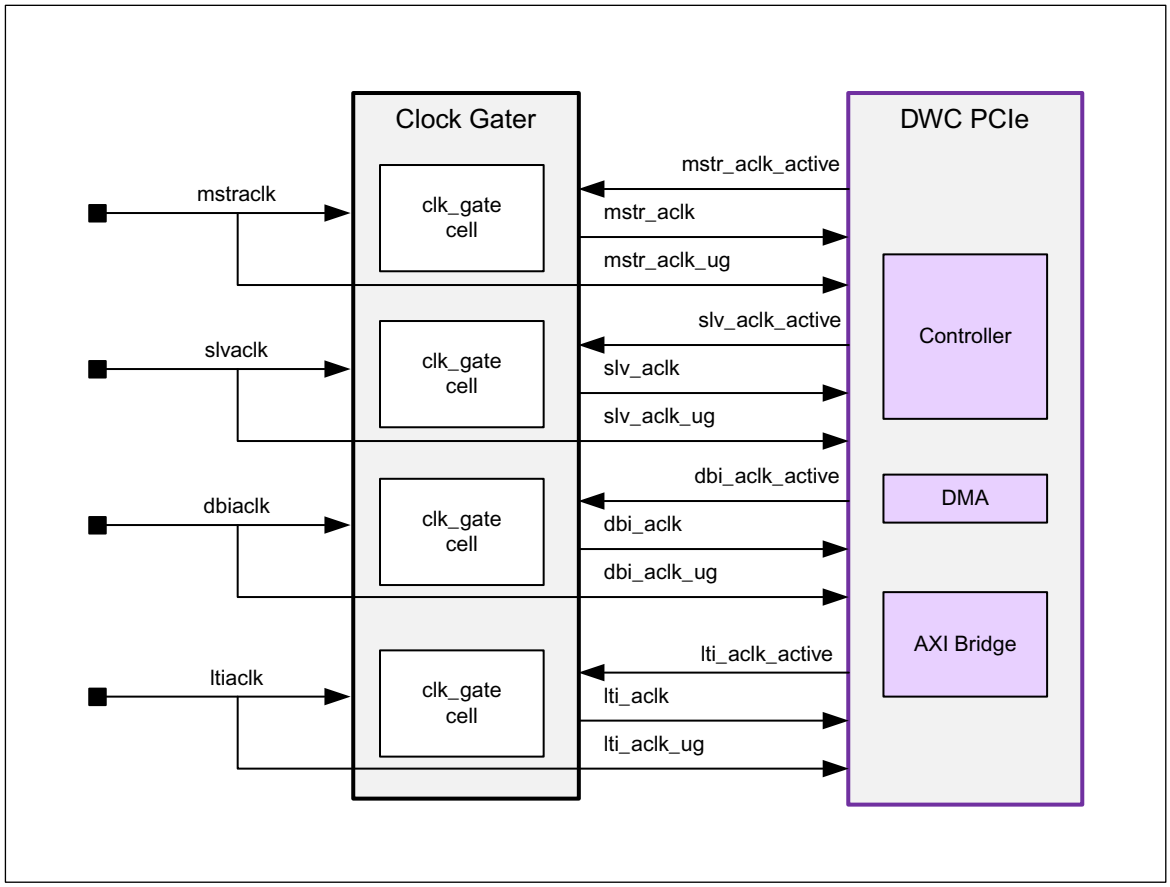
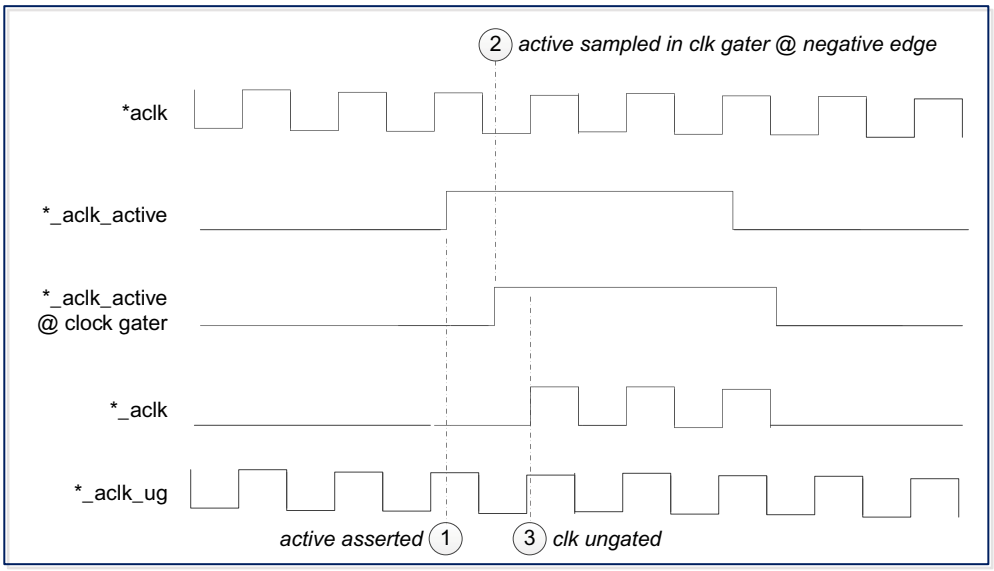


Figure 2-14 AXI Clock Gating Cell Behavior



Caution The clock gating cell should sample `*_aclk_active` in the negative edge of `*aclk` for gating the AXI clocks as shown in Figure 2-14. If you do not want to gate the AXI clocks feed the same clock to both `*_aclk` and `*_aclk_ug`.

2.3.5 Removing the Reference Clock

This section describes about the reference clock removal for the following:

- "Removing the Reference Clock for PCLK as PHY Output" on page 77
- "Removing the MAXPCLK for PCLK as PHY Input" on page 78

2.3.5.1 Removing the Reference Clock for PCLK as PHY Output

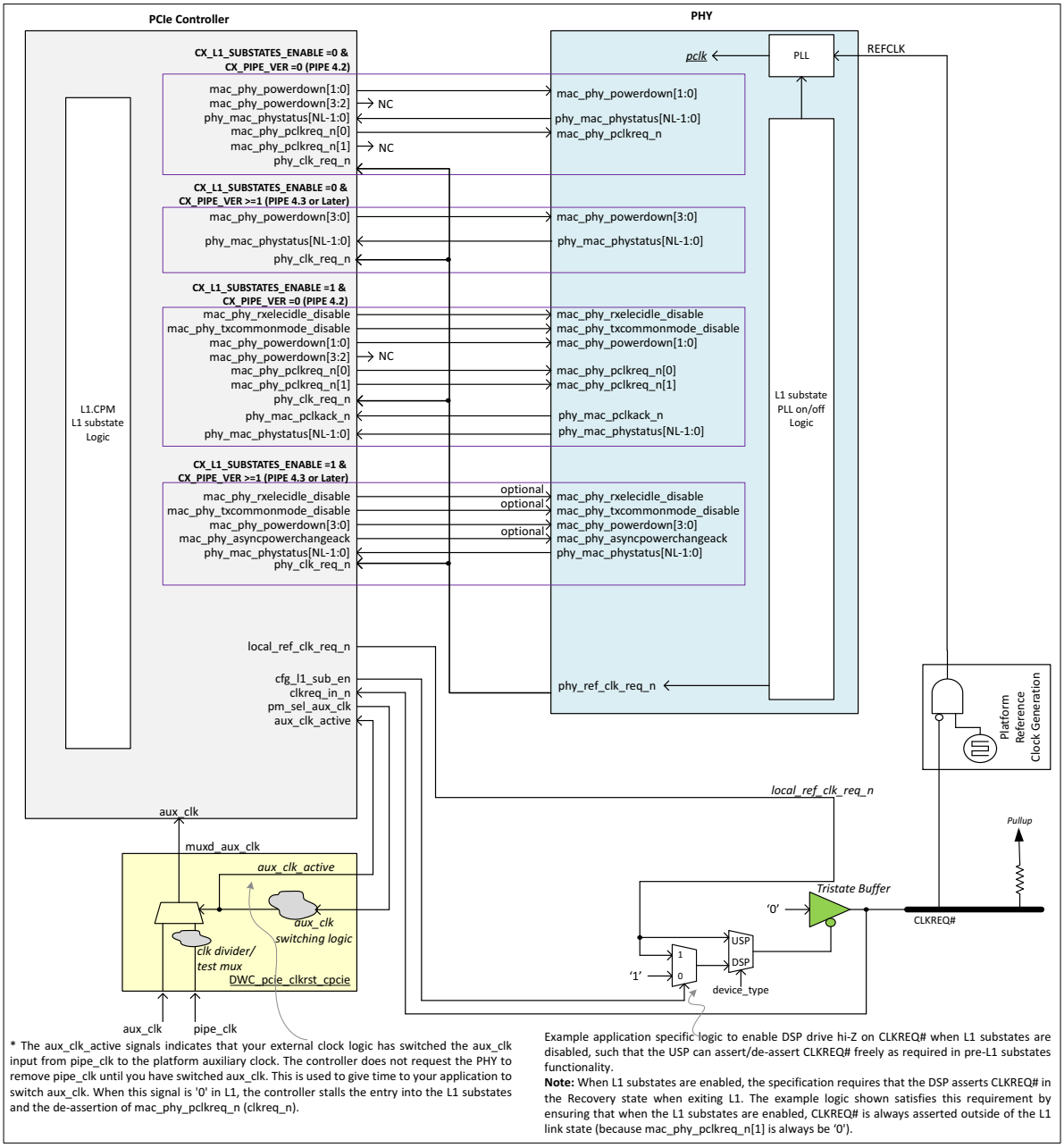
For PCLK as PHY Output, there are three schemes for removing the reference clock. For more information on low-power clocking see "Power Management" on page 259 and "Synchronous To Attribute" on page 71.

Table 2-5 REFCLK Removal Schemes

REFCLK Removal Scheme	Controller Outputs Used	Notes
L1 CLK PM (CX_L1_SUBSTATES_EN =0)	clk_req_n	"L1 Clock PM (L1 with REFCLK removal/PLL Off) Overview" on page 287
L1 CLK PM (CX_L1_SUBSTATES_EN =1)	mac_phy_pclkreq_n[0][a]	
L1 Substates (CX_L1_SUBSTATES_EN =1)	mac_phy_pclkreq_n[1]	"L1 Substates" on page 266 and "L1.1 and L1.2 Entry and Exit Conditions" on page 269

a. mac_phy_pclkreq_n signal is relevant only for PIPE 4.2 version.

Figure 2-15 Clock Removal Schemes for PCLK as PHY Output



2.3.5.2 Removing the MAXPCLK for PCLK as PHY Input

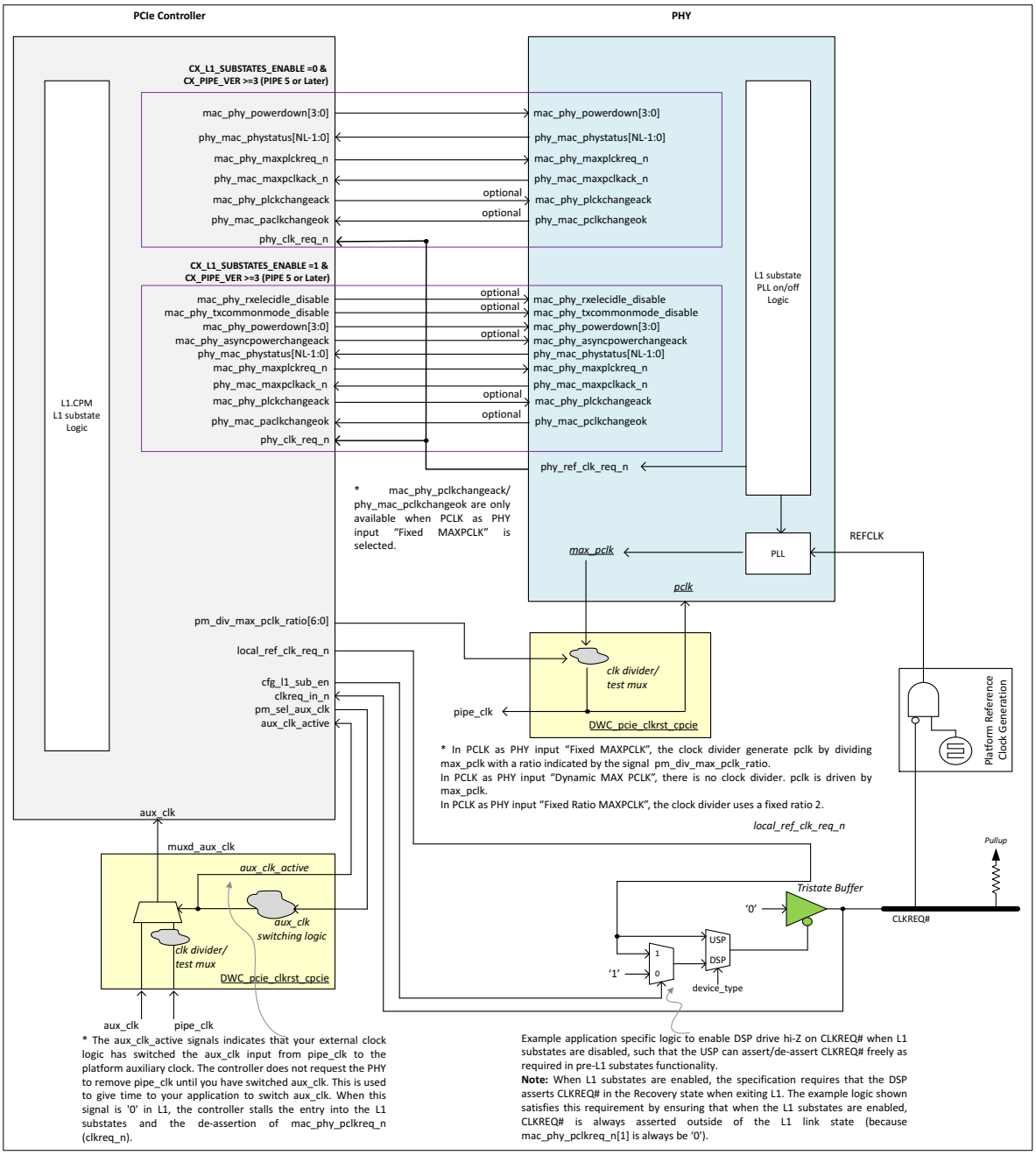
In PCLK as PHY input, all the schemes to remove the reference clock for PCLK as PHY output (mentioned in Table 2-6) are supported. You can also use the signals `mac_phy_maxpclkreq_n` and `phy_mac_maxpclkack_n` to remove MAXPCLK in low power mode. For more information on low-power clocking, see "Power Management" on page 259 and "Synchronous To Attribute" on page 71.

Table 2-6 REFCLK Removal Schemes

REFCLK Removal Scheme	Controller Outputs Used	Notes
L1 CLK PM (CX_L1_SUBSTATES_EN =0)	clk_req_n	"L1 Clock PM (L1 with REFCLK removal/PLL Off) Overview" on page 287
L1 CLK PM (CX_L1_SUBSTATES_EN =1)	mac_phy_pclkreq_n[0][a]	
L1 Substates (CX_L1_SUBSTATES_EN =1)	mac_phy_pclkreq_n[1]	"L1 Substates" on page 266 and "L1.1 and L1.2 Entry and Exit Conditions" on page 269

a. mac_phy_pclkreq_n signal is relevant only for PIPE 4.2 version.

Figure 2-16 Clock Removal Schemes for PCLK as PHY Input



PCLK Frequency Change

You can change the PCLK frequency using a handshake mechanism between controller and PHY involving the signals `phy_mac_pclkchangeok` and `mac_phy_pclkchangeack`.

Figure 2-17 Handshake, PCLK Frequency Change, and Clock Divider Control for Fixed MAXPCLK Mode (CX_PIPE_OPTIONAL_PCLKCHANGE_HS =0 or PIPE_OPTIONAL_PCLKCHANGE_HS =0)

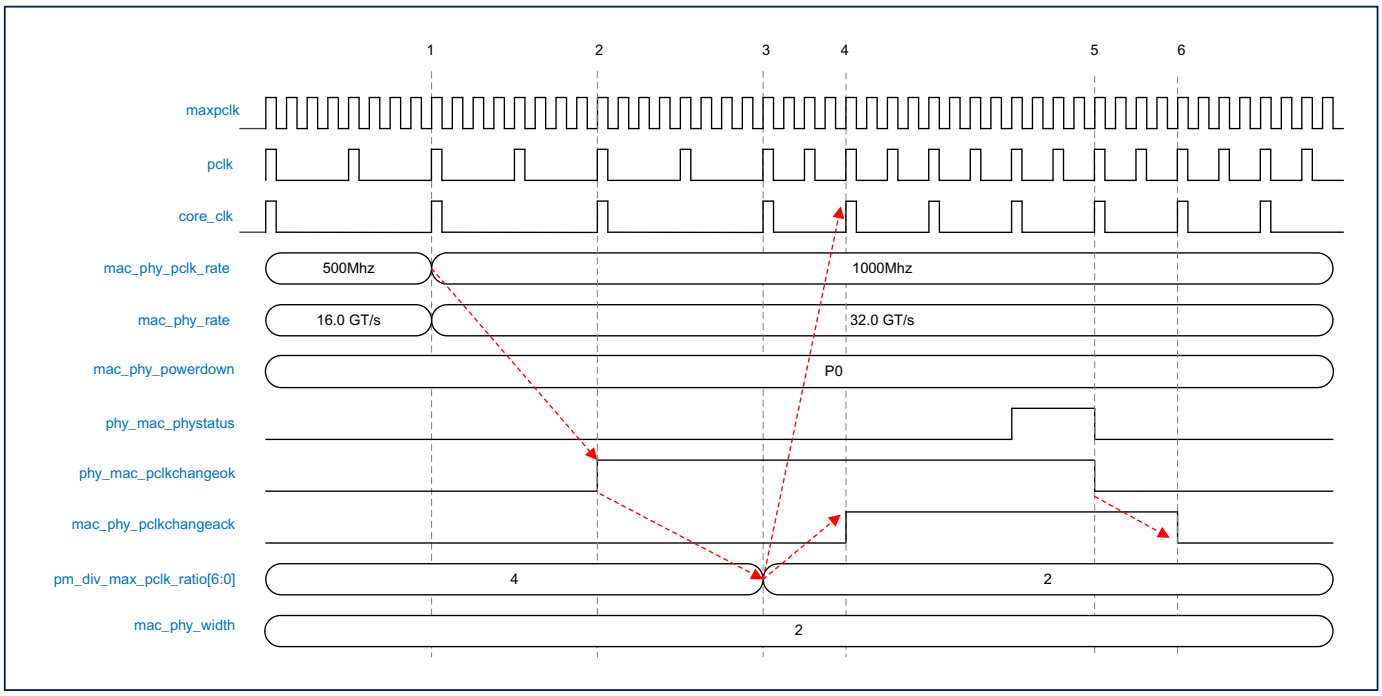
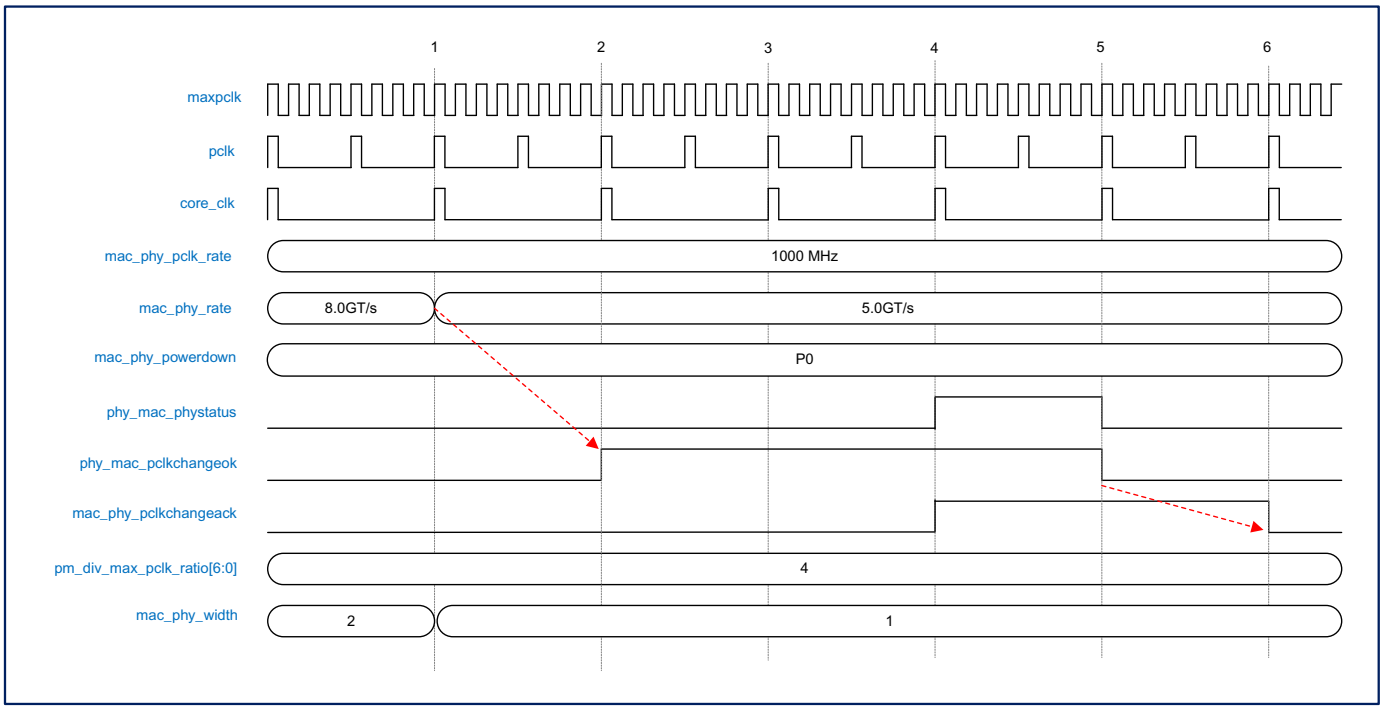


Figure 2-18 Handshake, PCLK Frequency Change, and Clock Divider Control for Fixed MAXPCLK Mode (CX_PIPE_OPTIONAL_PCLKCHANGE_HS =1 and PIPE_OPTIONAL_PCLKCHANGE_HS =1)

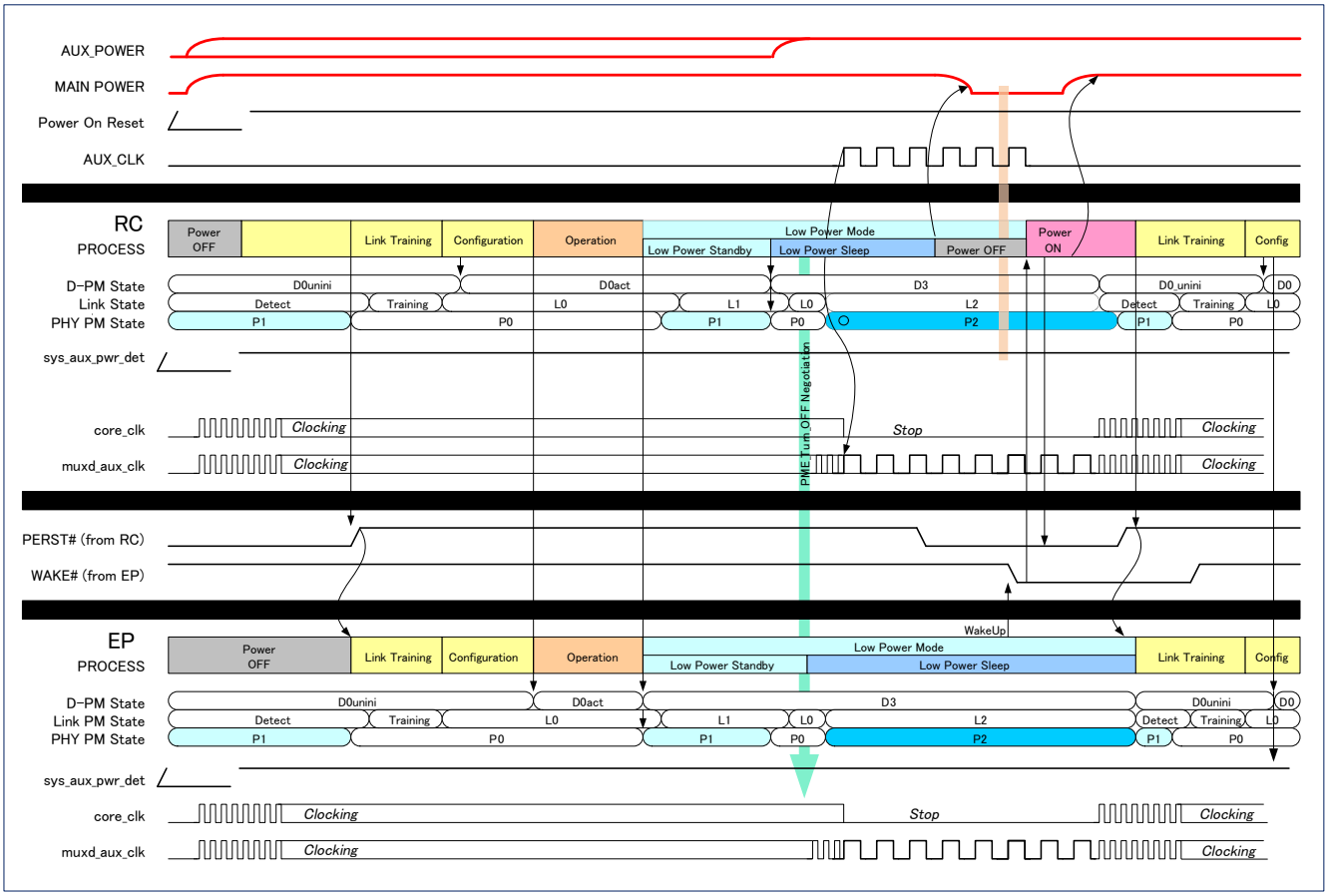


2.3.6 Additional Information

For more information on power gating and associated clock removal, see "Advanced Power Management and Power Domain Gating" on page 294.

For clock domain crossing (CDC) information, see "Advanced Information: Clock and Data Crossing (CDC)" on page 675.

Figure 2-19 Schematic Overview of L2 Low Power Process



Note When the controller transitions the PHY Powerdown state to P2 or P1.CPM, it simultaneously requests the source of aux_clk to be switched to the free running auxiliary clock. The clock switch design used in the clock and reset generation module of the controller requires two cycles of the core_clk to perform the clock switch. It is assumed that the PHY clock runs for a minimum number of cycles that is equivalent to two core_clk cycles after the controller has transitioned PHY Powerdown to P2 or P1.CPM. Notice that the PHY clock frequency could be different from core_clk frequency when frequency step is enabled.

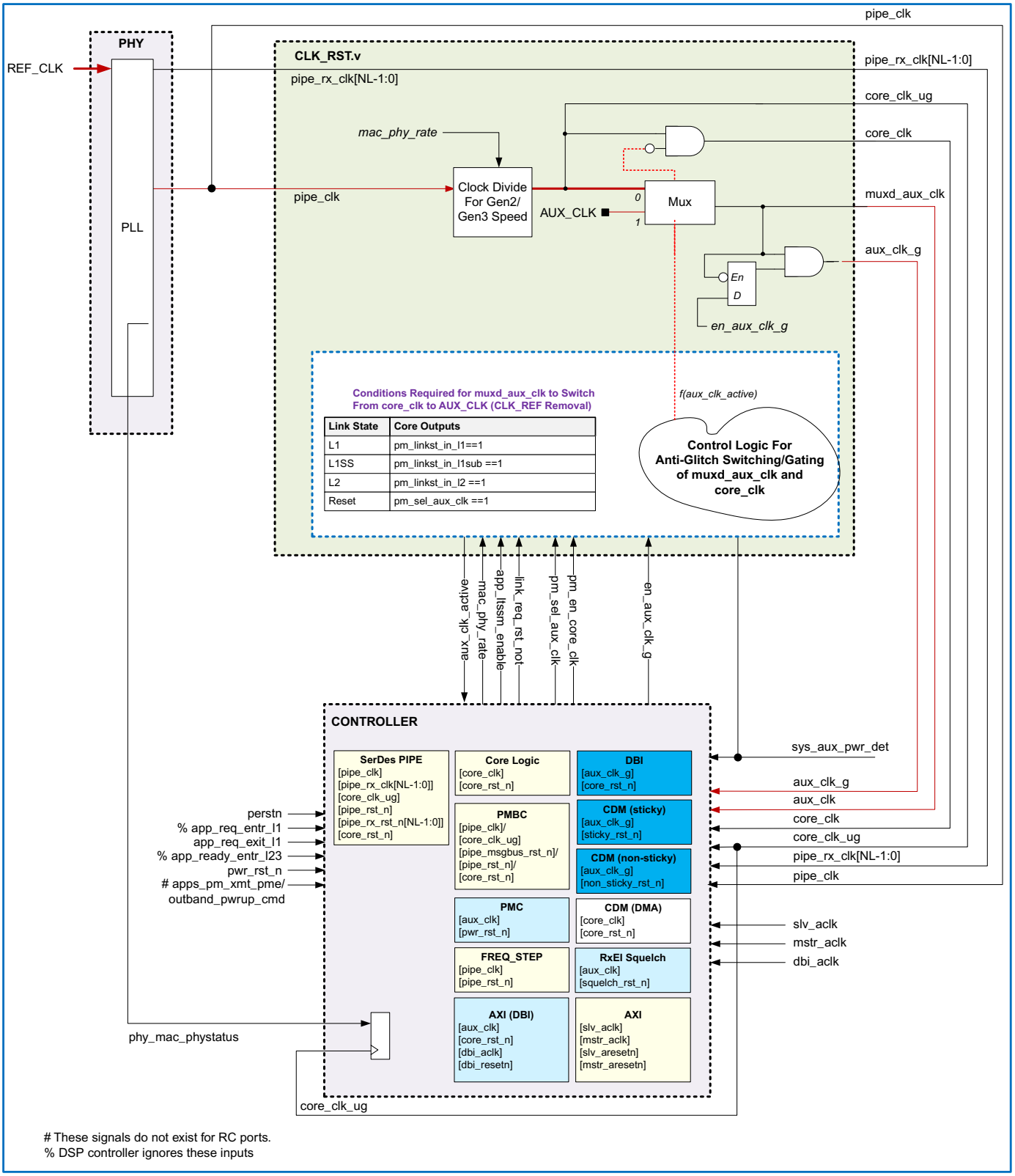
2.3.7 Clock Generation Module

The following circuits illustrate the clock generation for the controller, as used in the `DWC_pcie_clkrst.v` module.

Attention

The design example in `<workspace>/examples/DWC_pcie_clkrst.v` has been functionally verified, but has not been explicitly targeted for verification metrics. However, it has been used by Synopsys in the verification and regression environments for all controller configurations; including verification of all power transitions for the controller.Synopsys has only processed a small number of reference configurations of `DWC_pcie_clkrst.v` through the Synthesis and Spyglass flows. Therefore, your configuration of `DWC_pcie_clkrst.v` might not have been processed through these flows. The design example is shipped as example code. For that reason, it is not qualified for tape-out release. You must prove it in your SoC flow.

Figure 2-20 Clock Generation Overview in PCLK as PHY Output



The controller supports PCLK as PHY input for PIPE 5.0 and above according to the *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*. In PCLK as PHY input, the clock reset module generates PCLK based on MAXPCLK provided by the PHY, and drives it to the PHY. A clock divider in the clock reset module converts the MAXPCLK provided by the PHY to PCLK. The clock divider ratio varies based on the value of the parameter `CX_PIPE_PCLK_MODE`.

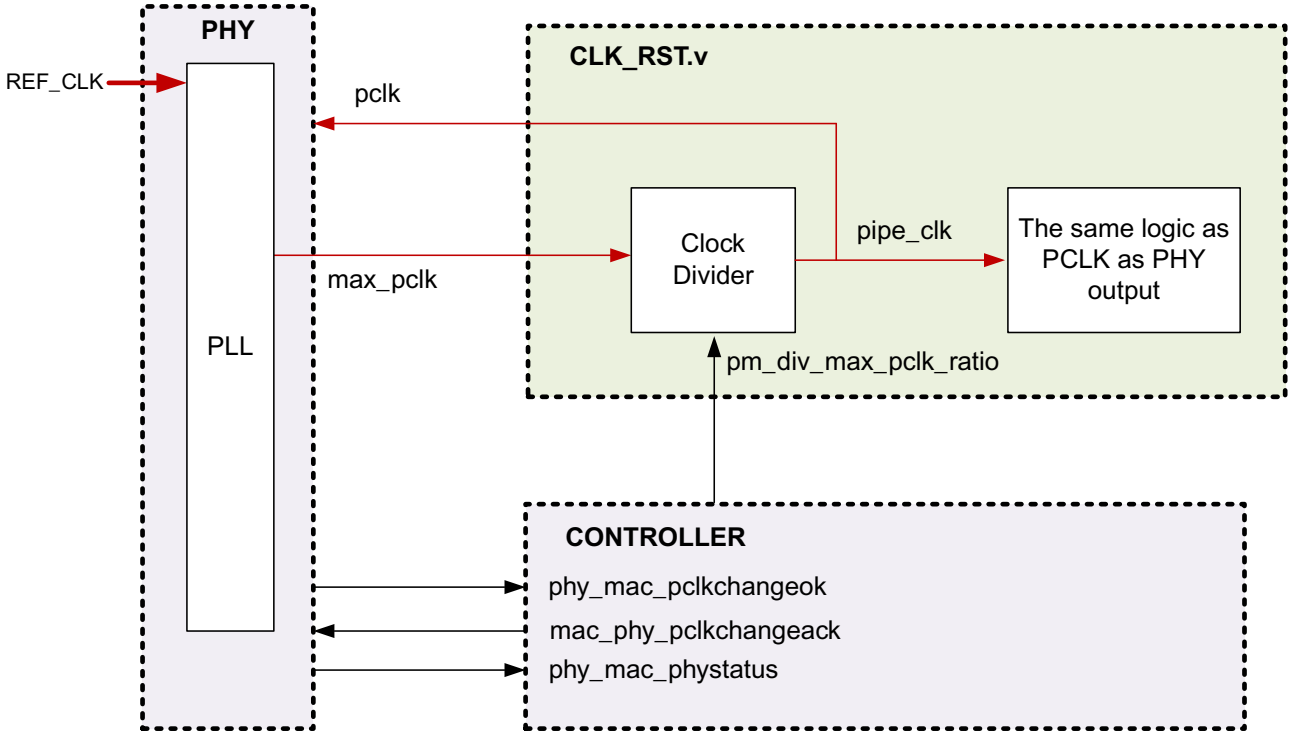
Table 2-7 Clock Divider Requirement for PCLK as PHY Input

PCLK as PHY Input Modes	Parameter Setting[a]	Clock Divider Ratio
Fixed MAXPCLK	CX_PIPE_PCLK_MODE =1	The clock divider ratio is controlled by the output pin pm_div_max_pclk_ratio of the controller. The controller calculates pm_div_max_pclk_ratio using the parameter CX_PHY_MAX_PCLK_FREQ_MHZ and the signal mac_phy_pclk_rate as follows: pm_div_max_pclk_ratio = CX_PHY_MAX_PCLK_FREQ_MHZ / mac_phy_pclk_rate For more information on usage of pm_div_max_pclk_ratio, see Figure 2-17.
Dynamic MAXPCLK	CX_PIPE_PCLK_MODE =2	The frequency ratio between MAXPCLK and PCLK is ‘1’. Clock divider is not required.
Fixed ratio MAXPCLK	CX_PIPE_PCLK_MODE =3	The frequency ratio between MAXPCLK and PCLK is fixed to ‘2’. A clock divider with fixed ratio ‘2’ is required.

a. For PCLK as PHY output, CX_PIPE_PCLK_MODE =0.

In PCLK as PHY input, the pclk signal is not generated in PHY. It is generated in clock reset module from `max_pclk`. The `max_pclk` could be generated from the PLL inside the PHY, as shown in Figure 2-21 or outside the PHY, as shown in Figure 2-22. The generation of the other clocks are the same as PCLK as PHY output.

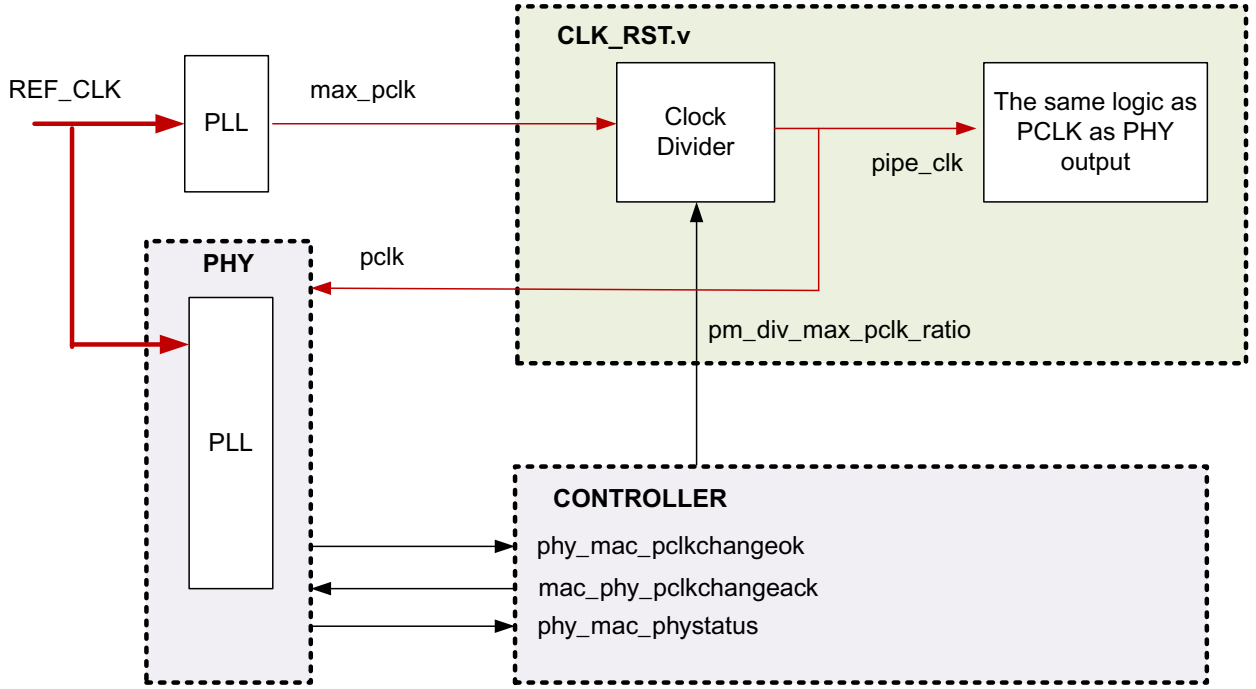
Figure 2-21 Clock Generation Overview in PCLK as PHY Input (PLL is Inside the PHY)



Note:

- The signals pm_div_max_pclk_ratio, phy_mac_pclkchangeok, and mac_phy_pclkchangeack are only used in PCLK as PHY input Fixed MAXPCLK.
- In PCLK as PHY input Dynamic MAXPCLK, there is no clock divider. The max_pclk drives pclk.
- In PCLK as PHY input Fixed Ratio MAXPCLK, the ratio for the clock divider is 2.
- You can switch the divider type from a pulse masking-based divider to a flop-based divider with 50% duty cycle by setting the CX_PIPE_PCLK_DIV_TYPE parameter to ‘1’.

Figure 2-22 Clock Generation Overview in PCLK as PHY Input (PLL is Outside the PHY)



Note:

- The signals pm_div_max_pclk_ratio, phy_mac_pclkchangeok, and mac_phy_pclkchangeack are only used in PCLK as PHY input Fixed MAXPCLK.
- In PCLK as PHY input Dynamic MAXPCLK, there is no clock divider. The max_pclk drives pclk.
- In PCLK as PHY input Fixed Ratio MAXPCLK, the ratio for the clock divider is 2.
- You can switch the divider type from a pulse masking-based divider to a flop-based divider with 50% duty cycle by setting the CX_PIPE_PCLK_DIV_TYPE parameter to ‘1’.

Figure 2-23 Clock Switch and Gating Logic

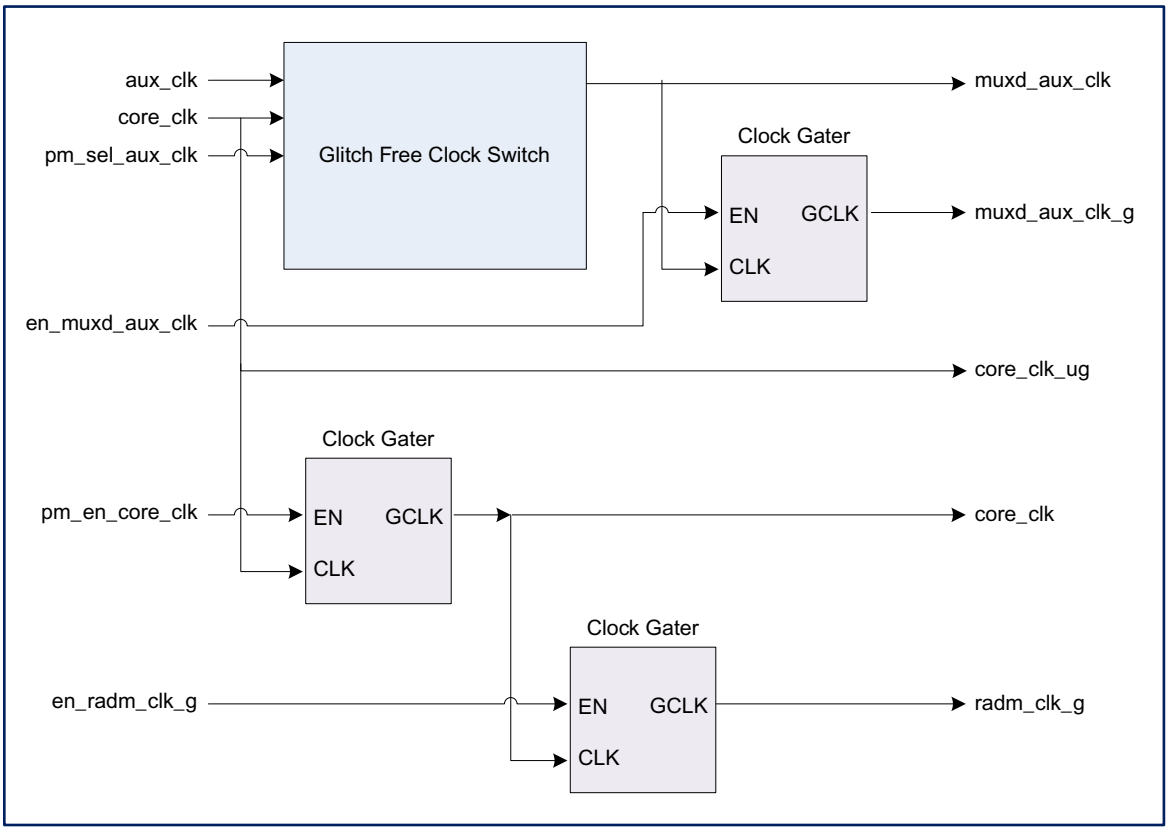


Figure 2-23 describes the clock switch and gating logic which functions as follows:

- The clock reset module implements a glitch free clock switch to permit the generation of the `aux_clk` input to the controller. The switch is controlled by the output `pm_sel_aux_clk`, which selects the `aux_clk` in certain low power states, and power on reset or fundamental reset.
- The clock reset module implements architectural clock gating on `core_clk`, which permits the disabling of `core_clk` in certain low power states as determined by the de-assertion of `pm_en_core_clk`.
- The clock for the RADM may be gated off when the signal `en_radm_clk_g` is de-asserted to allow architectural clock gating in the receive path.
- The clock `muxd_aux_clk_g` is a clock gated version of the `muxd_aux_clk`, which may be disabled when power gating is supported in L1.2.

For more information on the clock switch logic and implemented architectural clock gating, see the RTL of the clock and reset module.

Attention The frequency of the platform driven free running aux_clk must be slower than two times of the slowest frequency of core_clk (that is, the frequency of core_clk at Gen1 rate). For more information on the Gen1 core_clk frequency for your configuration, see "Supported Controller Configurations" on page 40.

Figure 2-24 Frequency Step Down Clock Divider

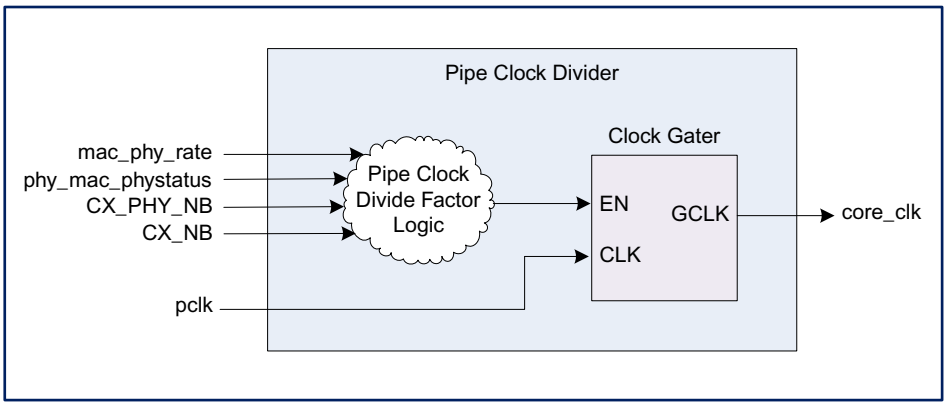


Figure 2-24 describes the frequency step down clock divider logic which functions as follows:

If the derived parameter `CX_FREQ_STEP_DOWN_EN` is defined, the `core_clk` frequency is slower than the `pipe_clk` frequency, in this scenario it is necessary to generate the `core_clk` as a divided down version of the `pipe_clk`.

- The following factors contribute to the selection of the `pipe_clk` division factor used to generate the `core_clk:`
  - Number of symbols per cycle for the controller and the PHY
  - The required link signaling rate as requested by the `mac_phy_rate` output of the controller

  For example, if the PHY supports two symbols per `pipe_clk` cycle at GEN1 speed while the controller supports four symbols per `core_clk` cycle at GEN1 speed, the divide by two is enabled.
- A counter running on `pclk` is used to count the required number of `pclk` cycles, the output of this counter in combination with the required division factor is used to enable the clock gating.
- The clock divider uses a pulse suppression technique to generate the divided version of the `pipe_clk`. This means that the generated `core_clk` do not have a 50% duty cycle.

For the detailed implementation of the divider circuit, see the RTL of the clock and reset generation module.

You could use a PLL/DPLL instead of the clock divider. This eases balancing of the `core_clk` , `radm_clk_g`, and `pipe_clk` clock trees as the PLL/DPLL can eliminate the divider insertion delay.

2.4 Reset Requirements

This section covers reset generation for the controller, focusing on conventional PCIe. The following topics are discussed:

- "Overview"
- "Implementing the PCIe Resets" on page 95
- "Hot Reset" on page 97

2.4.1 Overview

The `DWC_pcie_clkrst.v` module generates the resets as per Table 2-8.

Note: Sticky is a PCIe term and means that the register retains its value during some reset sequences. There are many reset inputs for the controller as describes in Table 2-8. All of these reset inputs are asserted on a cold/fundamental/power-on reset. Not all these reset inputs are asserted during a hot reset (when link is reset and most of controller logic is not reset but some small parts of the controller and some registers ARE reset). Any registers that you want to remain un-reset during a hot reset must be wired to the `sticky_rst_n` pin.

Table 2-8 Generation of Individual Reset Signals for each Reset Event

Reset Name	Clock Domain	Destination	Cold Reset (Power On Reset)	Warm Reset (PERST#)[abc]	L1.2 Power Gating[d]	Hot Reset (Link-down)[e f]
pwr_rst_n	aux_clk	Controller	Yes			
mstr_pwr_rst_n[g]	mstr_aclk	Controller	Yes			
slv_pwr_rst_n[h]	slv_aclk	Controller	Yes			
dbi_pwr_rst_n[i]	dbi_aclk	Controller	Yes			
sticky_rst_n	aux_clk_g	Controller	Yes	Yes	Yes[j]	
non_sticky_rst_n	aux_clk_g	Controller	Yes	Yes	Yes	Yes
core_rst_n	core_clk / radm_clk_g	Controller	Yes	Yes	Yes	Yes
pipe_rst_n[k]	pipe_clk	Controller	Yes	Yes	Yes	Yes
pipe_msgbus_rst_n[l]	pipe_clk	Controller	Yes	Yes		Yes
phy_rst_n	NA	PHY(PIPE)	Yes	Yes		Yes
phy_perst_n	NA	PHY(PMA)	Yes	Yes[m]		No
phy_reg_rst_n[n]	phy_reg_clk_ug	Controller	Yes	Yes	Yes	
ret_sticky_rst_n[o]	aux_clk_g	Controller	Yes	Yes		
ret_non_sticky_rst_n[p]	aux_clk_g	Controller	Yes	Yes		Yes
ret_core_rst_n[q]	core_clk	Controller	Yes	Yes		Yes

Reset Name	Clock Domain	Destination	Cold Reset (Power On Reset)	Warm Reset (PERST#)[abc]	L1.2 Power Gating[d]	Hot Reset (Link-down)[e f]
mstr_aresetn	mstr_aclk / mstr_aclk_ug	Controller	Yes	Yes	Yes	Yes
slv_aresetn	slv_aclk / slv_aclk_ug	Controller	Yes	Yes	Yes	Yes
dbi_aresetn	dbi_aclk / dbi_aclk_ug	Controller	Yes	Yes	Yes	Yes
cxs_rst_n	cxs_clk	Controller	Yes	Yes	Yes	Yes
pipe_rx_rst_n[r]	pipe_rx_clk	Controller	Yes	Yes	Yes	Yes

a. When CX_ENHANCED_PM_EN =1, the DWC_pcie_clkrst.v asserts all resets upon PERST# (perst_n) except pwr_rst_n. PERST# is the trigger for L2 power removal.
b. On PERST# assertion, all of the controllers resets apart from pwr_rst_n are asserted synchronous to aux_clk. If a register reset by core_rst_n, for example, fans in to a register reset by pwr_rst_n there is a reset domain crossing path. The path through the preset or clear pin of the register being reset to the data pin of register which is not being reset should be timed to ensure that there is no metastability on the register reset by pwr_rst_n.
c. The minimum duration the PCIe Power Management Controller (PMC) can handle is (6+DEFAULT_AUX_CLK_FREQ/62.5) +10 in slow aux_clk cycles.
d. CX_ENHANCED_PM_EN =1
e. TS protocol-based hot reset or link-down reset.
f. On link down reset the clock and reset module gates off the controller's clocks when the resets are being asserted. This is to ensure that the registers in the design are not being clocked when the resets are asynchronously asserted, thereby avoiding any Reset Domain Crossing (RDC) issues.
g. This signal exists only when MSTR_CLK_DIFF_ENABLE =1 && CX_ENHANCED_PM_EN =1
h. This signal exists only when SLV_CLK_DIFF_ENABLE =1 && CX_ENHANCED_PM_EN =1
i. This signal exists only when DBISLV_CLK_DIFF_ENABLE =1 && CX_ENHANCED_PM_EN =1
j. See ret_sticky_rst_n.
k. pipe_rst_n and pipe_clk are not used when CX_FREQ_STEP_EN =0.
l. pipe_msgbus_rst_n is not used when CX_PIPE_VER <3.
m. This behavior can be disabled at configuration time by setting DEFAULT_PHY_PERST_ON_WARM_RESET =0. This behavior can also be modified at runtime by changing the value of PHY_INTEROP_CTRL_OFF.PHY_PERST_ON_WARM_RESET to zero.
n. Exists when CX_PHY_VIEWPORT_ENABLE =1.
o. Exists when CX_ENHANCED_PM_EN =1. ret_<resetname> is the reset signal for the logic that is retained in L1.2. In L2, it is asserted identically to <resetname>. In L1.2, it is not asserted.
p. TS protocol-based hot reset or link-down reset.
q. TS protocol-based hot reset or link-down reset.
r. This signal exists only when CX_PIPE_SERDES_ARCH =1

If you want to delay link re-establishment (after reset) so that you can reprogram some registers through DBI, you must set `app_ltssm_enable =0` as shown in Figure 2-25. Unless `app_ltssm_enable` is de-asserted immediately in Detect state, the controller continues to link up. If `app_ltssm_enable` is de-asserted or the controller is reset when the controller is out of the Detect state, the LTSSM moves immediately back to the Detect state. This transition is not defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0* and might cause a PIPE protocol violation.

Figure 2-25 Delaying the Link Training After Cold Reset (Run a Simulation to Obtain Accurate Timing)

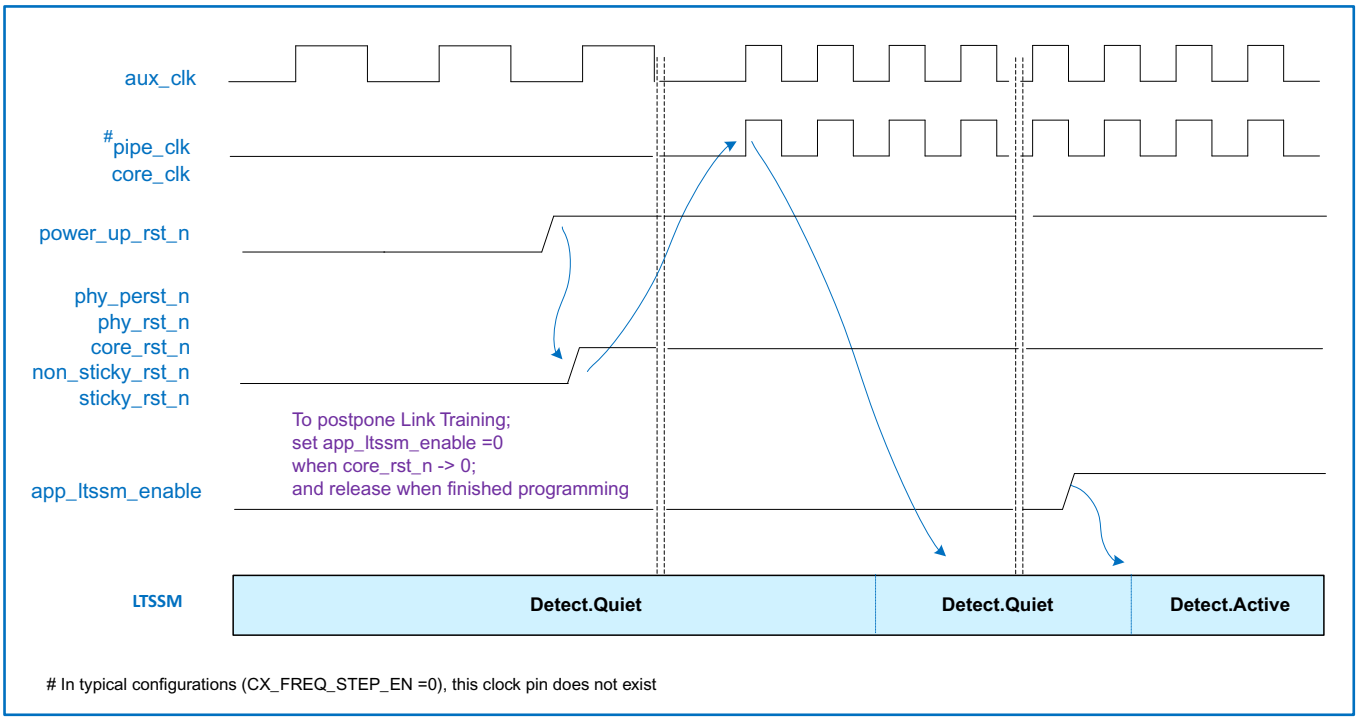
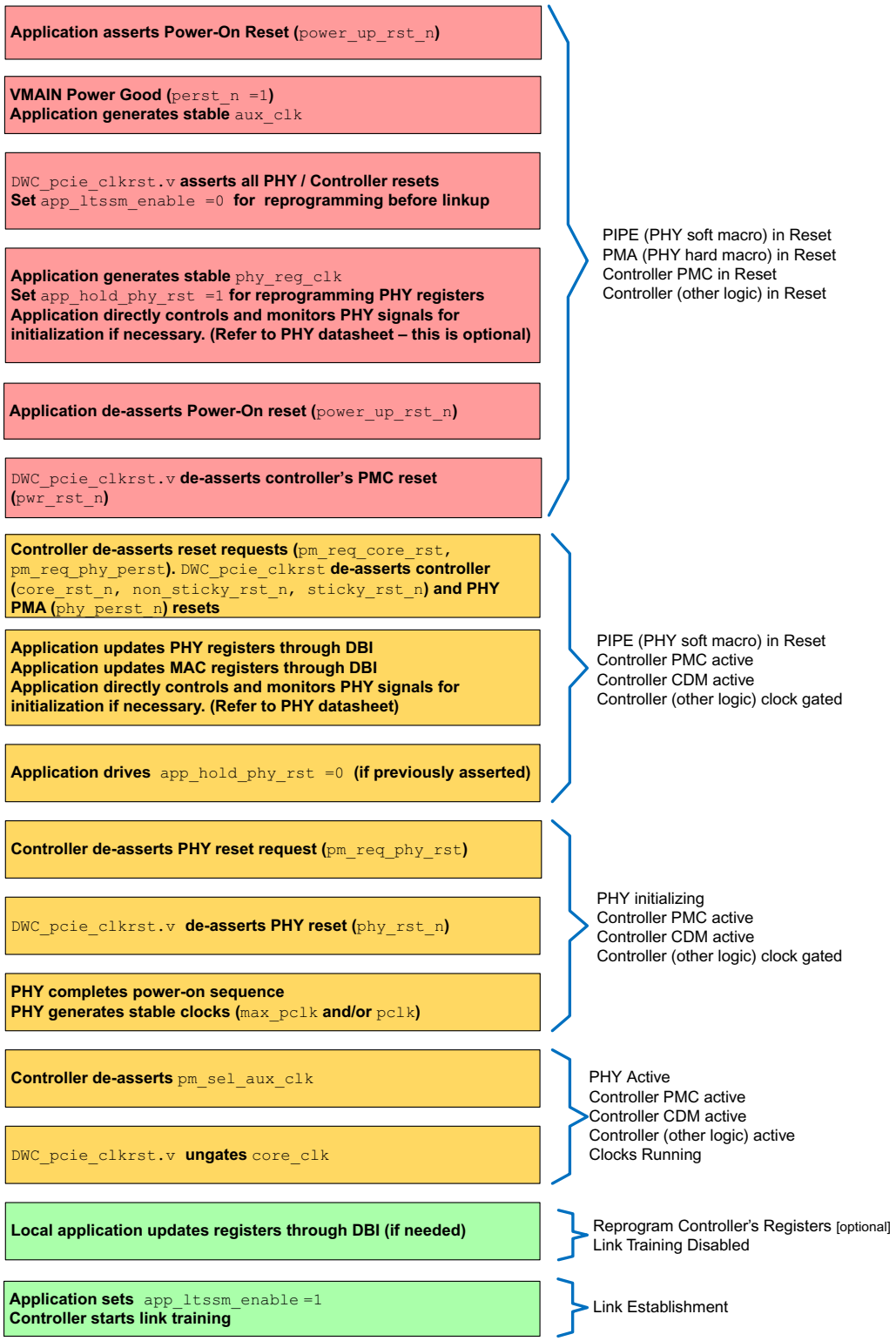
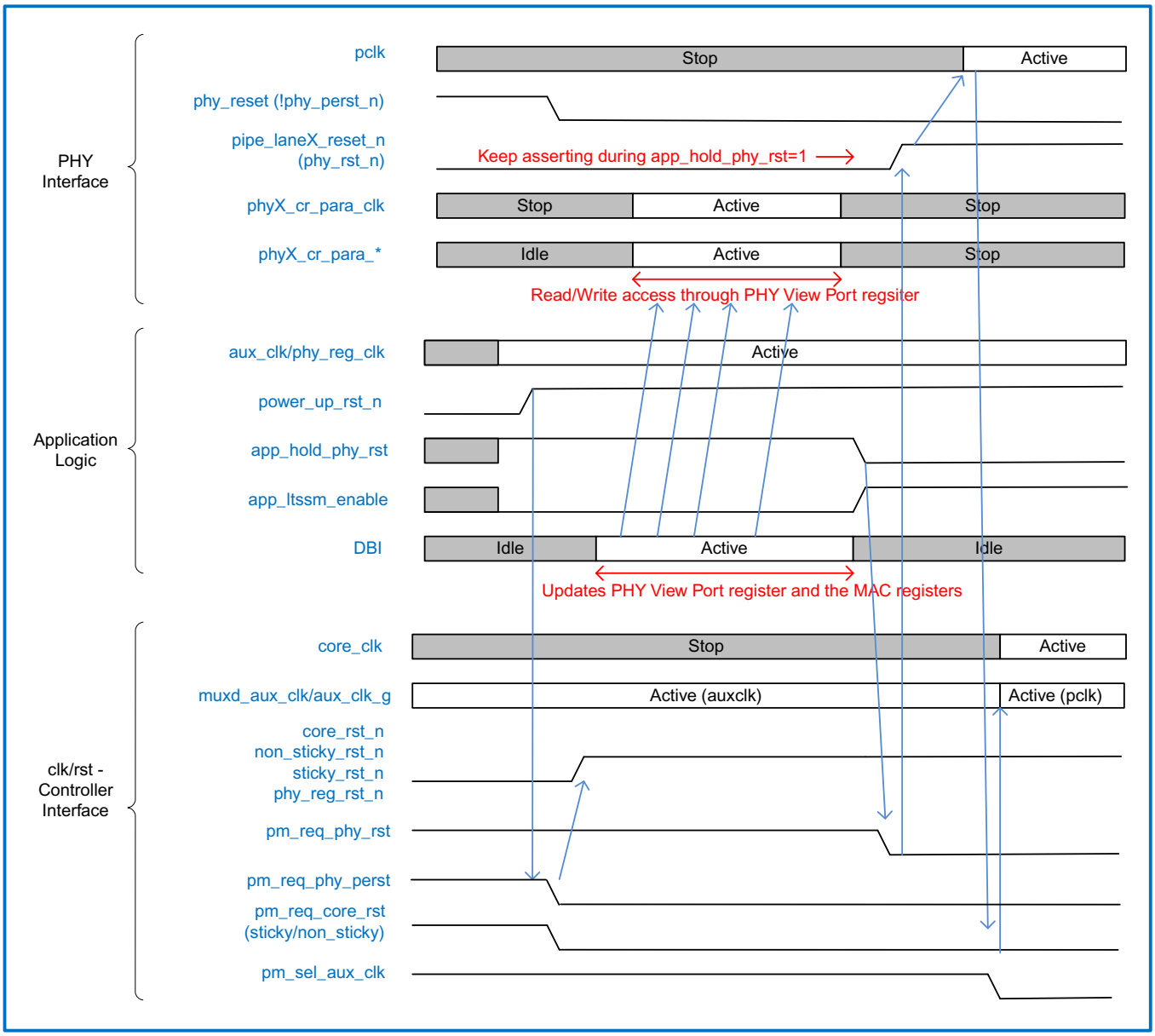


Figure 2-26 PCIe Cold Reset Sequence with Optional PHY Initialization (app_hold_phy_rst=1)



For PHY Initialization information when `CX_PHY_VIEWPORT_ENABLE =1`, see Figure 3-36 on page 111.

Figure 2-27 PHY Initialization After Cold Reset (CX_PHY_VIEWPORT_ENABLE =1)



Attention

- To obtain accurate timing of PHY initialization after cold reset you must run a simulation.
- The controller's PHY Viewport (PRBI) to PHY registers can have high access latency. If your application needs low latency to initialize Synopsys PHY registers or SRAM, then PRBI should not be used. Instead, you should design a direct interface to the PHY's SRAM or CR Bus. PRBI is recommended only for PHY debug.

2.4.2 Implementing the PCIe Resets

For more information on descriptions of each I/O signal, see the appropriate section in "Signal Interfaces" on page 355.

Figure 2-28 Reset Generation Overview

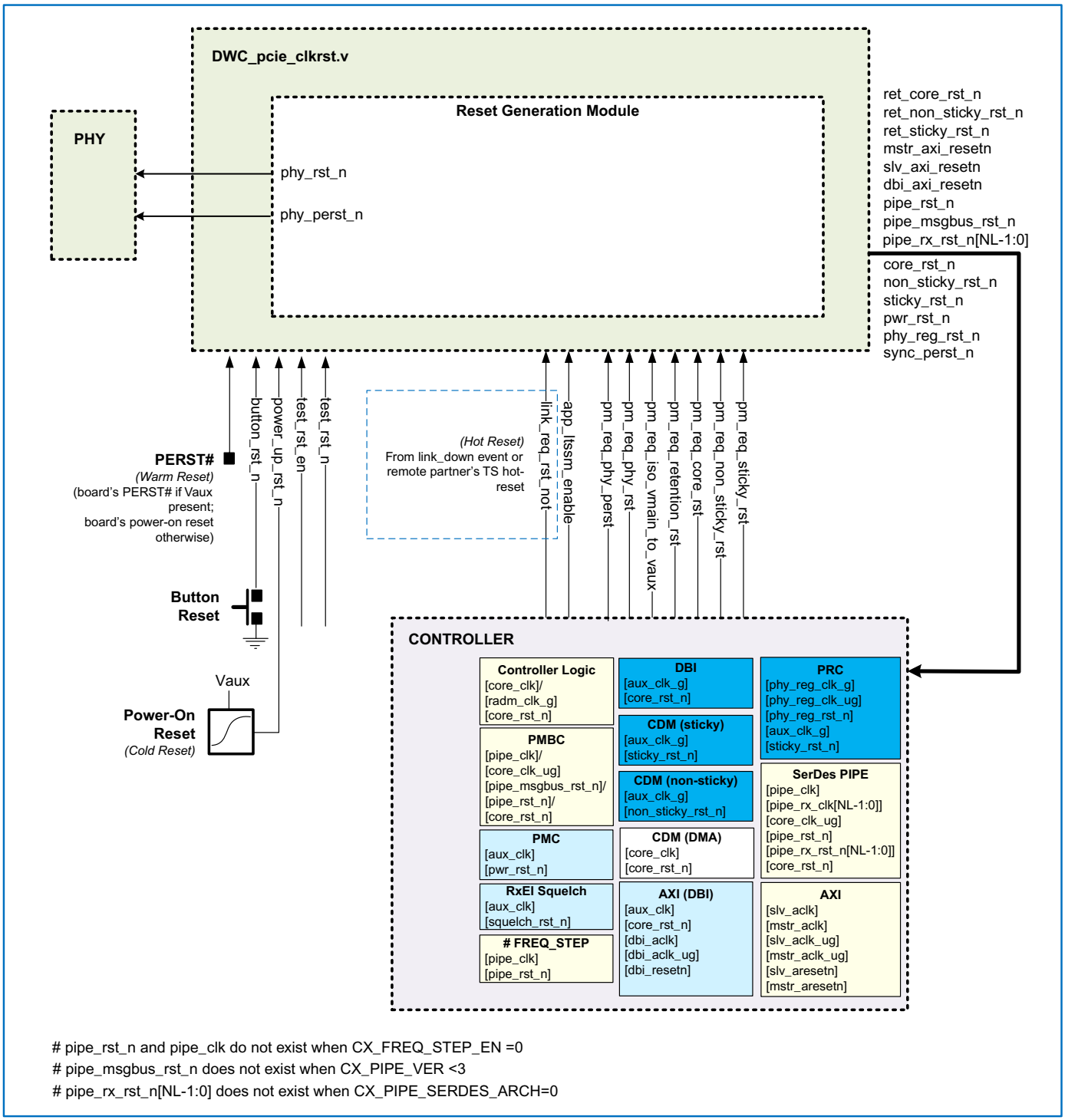
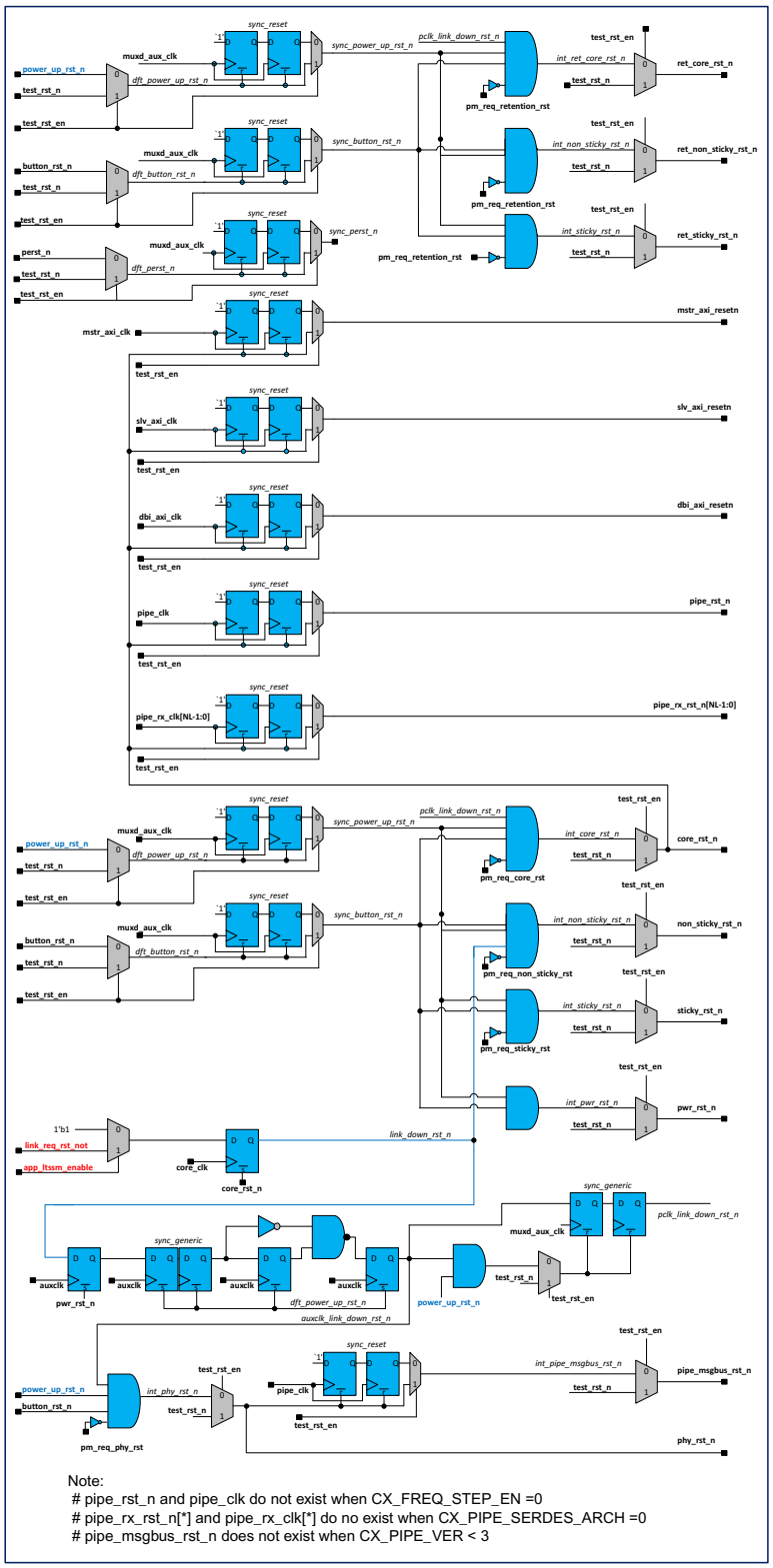


Figure 2-29 Reset Generation Module (Refer to RTL for Complete Design)



Note:
pipe_rst_n and pipe_clk do not exist when CX_FREQ_STEP_EN =0
pipe_rx_rst_n[*] and pipe_rx_clk[*] do no exist when CX_PIPE_SERDES_ARCH =0
pipe_msgbus_rst_n does not exist when CX_PIPE_VER < 3

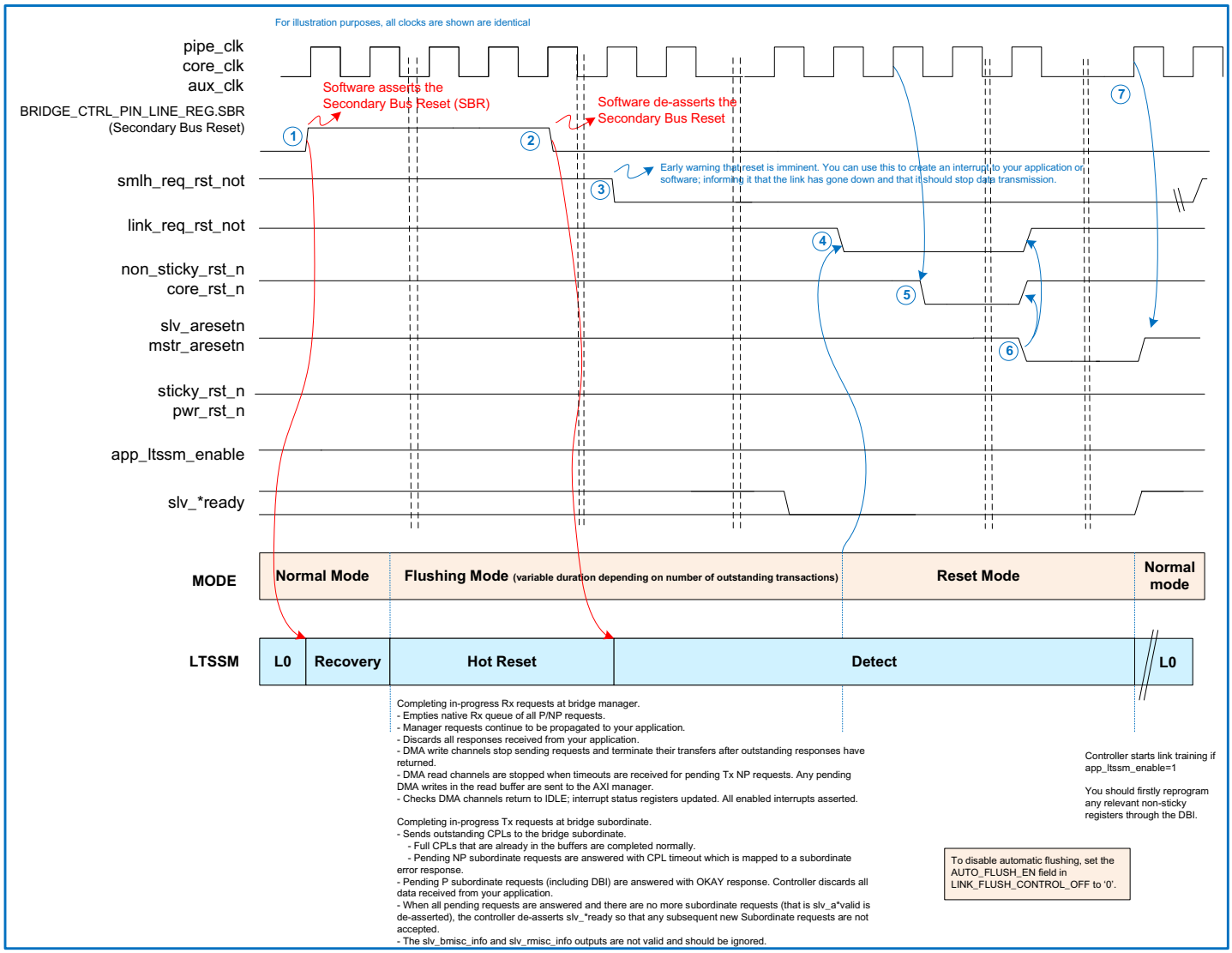
2.4.3 Hot Reset

A Downstream Port (DSP) can hot reset an Upstream Port (USP) by sending two consecutive TS1 ordered sets with the hot reset bit asserted, or by pulsing the `app_init_rst` input. Eventually, the DSP and USP assert `link_req_rst_not` to request external logic to reset them. Alternatively, during normal operation, the link might fail and go down. After this link-down event, the controller requests the `DWC_pcie_clkrst.v` module to hot-reset the controller. There is no difference in the handling of a link-down reset or a hot reset; the controller asserts the `link_req_rst_not` output requesting the `DWC_pcie_clkrst.v` module to reset the controller.

To prevent your application from hanging when the controller is reset, the controller must cleanly terminate all transactions that are still in-progress.

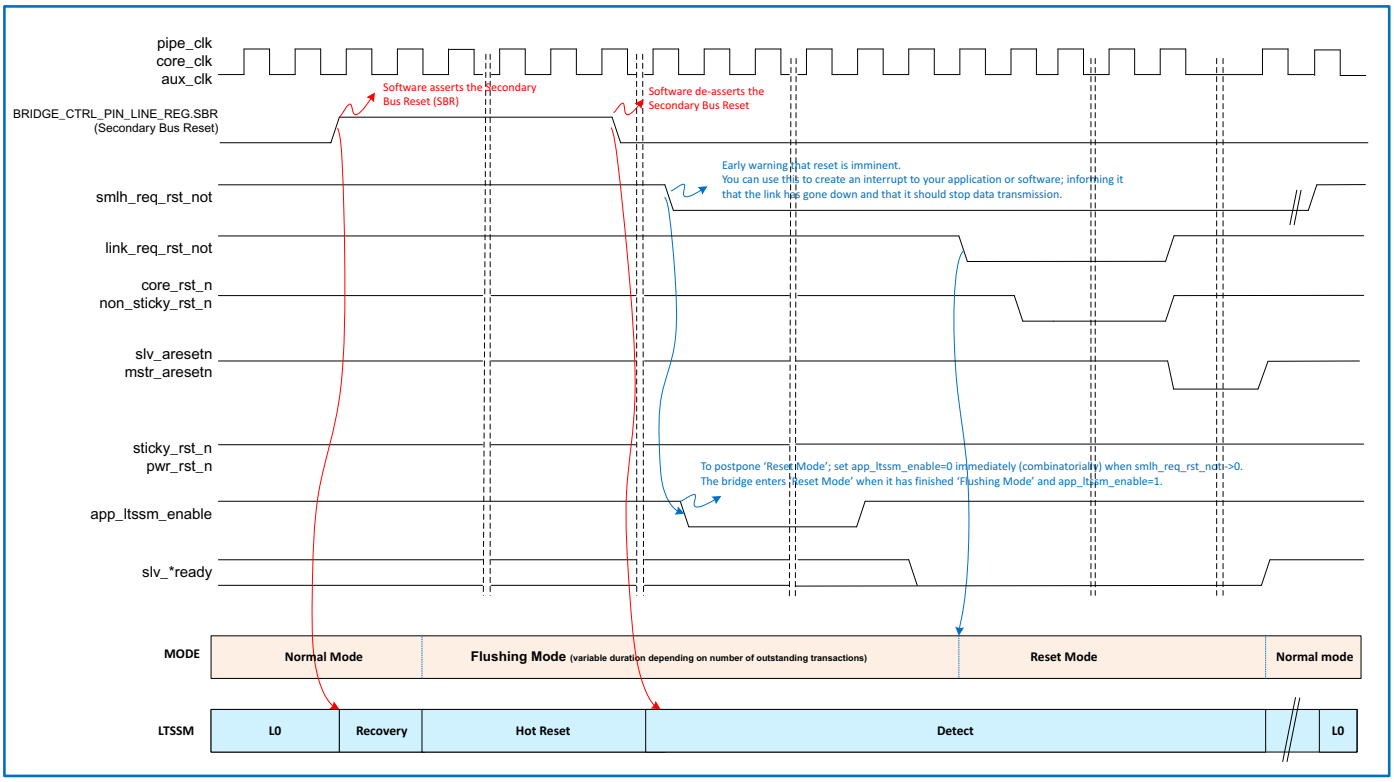
In AXI configurations with a segmented-buffer Rx queue architecture, the controller automatically executes this process before asserting `link_req_rst_not` as shown in Figure 2-30. To disable automatic flushing, set the `AUTO_FLUSH_EN` field in `LINK_FLUSH_CONTROL_OFF` to '0'.

Figure 2-30 AXI Bridge Flushing and Reset Process (Run a Simulation to Obtain Accurate Timing of Resets)



Normally, when all pending requests are answered and there are no more subordinate requests (`slv_a*valid` is de-asserted), the controller de-asserts `slv_*ready` so that any subsequent new subordinate requests are not accepted. If you want to delay the reset of the controller so that it continues to accept and terminate subordinate requests[1], or to read out some register status through the DBI, you must set `app_ltssm_enable =0` immediately (combinatorially) after `smlh_req_rst_not ->0`, as shown in Figure 2-31. Also applies to non-AXI configurations.

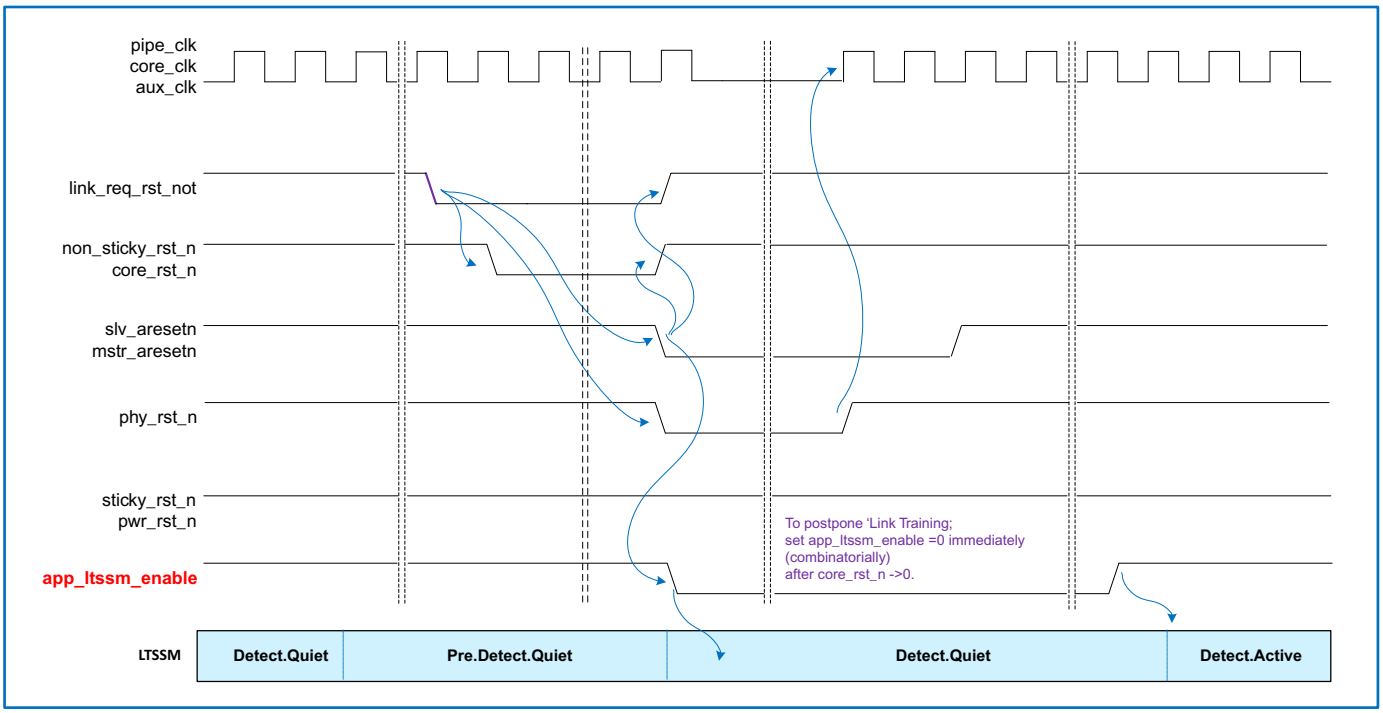
Figure 2-31 Delaying the Controller Hot Reset (Run a Simulation to Obtain Accurate Timing)



If you want to delay link re-establishment (after reset) so that you can reprogram some registers through DBI, you must set `app_ltssm_enable =0` immediately after `core_rst_n` as shown in Figure 2-32. Also applies to non-AXI configurations.

1. Transmitted by your application before smlh_req_rst_not ->0 and did not reach the bridge before link_req_rst_not ->0

Figure 2-32 Delaying the Link Training After Hot Reset (Run a Simulation to Obtain Accurate Timing)



Unless `app_ltssm_enable` is de-asserted immediately in Detect state, the controller continues to link up. If `app_ltssm_enable` is de-asserted or the controller is reset when the controller is out of the Detect state, the LTSSM moves immediately back to the Detect state. This transition is not defined in the PCIe Specification and might cause a PIPE protocol violation.

2.5 Receive Queues

This section covers the following buffering topics:

- "Receive Queue Architecture"
- "Configuring Your Queues" on page 104

2.5.1 Receive Queue Architecture

The receive queues use Quad-Double-Word (QDW) RAMs. There is a separate queue and a corresponding interface for posted, non-posted, and completion TLPs. The data bus width determines the number of TLPs that the queue can store in one clock cycle. Flit Mode protocol allows eight TLPs for the same FC to be placed in each flit half. The controller configuration and protocol ordering rules determine whether the controller reads one or two TLPs from a queue.

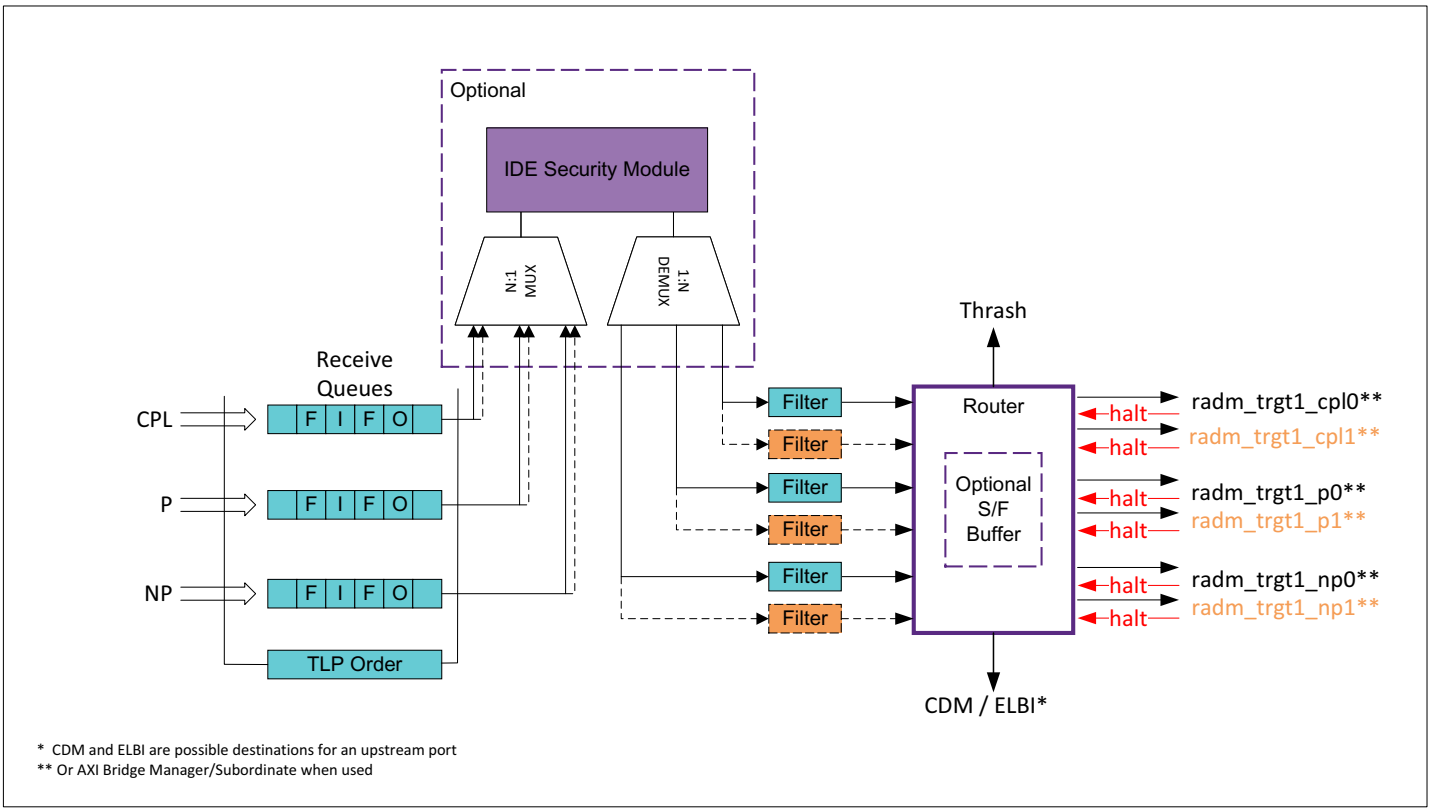
The number of QDW RAMs required for each queue is shown in Table 2-9 for a single VC configuration.

Table 2-9 Number of RAMs per Datapath Configuration

Datapath Width	Maximum Valid QDWs at FC Queue Input	Number of External RAMs per FC	Total Receive Queue RAMs
32[a]	2	2	6
64[b]	2	2	6
128	2	2	6
256	4	4	12
512	7	7	21
1024	14	14	42

a. NA for Gen6 configured controller
b. NA for Gen6 configured controller

Figure 2-33 Queue Architecture



- TRGT1 interfaces operate with an interface halt, and a pkt_grant/pkt_halt.
- For a Gen6 configured controller, the number of TRGT1 interfaces can be configured to be three (1 Posted, 1 Non-Posted, 1 CPL) or six (2 Posted, 2 Non-Posted, 2 CPL). The second set of FC interfaces are used to forward data when the ordering requirements are relaxed in the TLP (RO, IDO, VC).

Queue Modes (Native PCIe Controller)

The three possible queue modes for posted, non-posted, and completion TLPs are described in Table 2-10. Table 2-10 shows the division of functionality for these actions between the native PCIe controller (without AMBA bridge) and your application.

Table 2-10 Comparison of Queue Modes for Native PCIe Controller (without AMBA bridge)

	RADM_<P|NP|CPL>_QMODE_VCn		
	0x1 **Store-and-Forward (S/F)**	**0x2** **Cut-Through (C/T)[a]**	**0x4** **Bypass (B/P)**
Buffering Action	Controller stores all of the TLP in the buffer	Controller stores some of the TLP in the buffer	Controller stores none of the TLP in a buffer; there is no buffer
When Controller Forwards TLP to Your Application (Affects Latency)	At end	Immediately	Immediately

	RADM_<P|NP|CPL>_QMODE_VCn		
	0x1 **Store-and-Forward (S/F)**	**0x2** **Cut-Through (C/T)**[a]	**0x4** **Bypass (B/P)**
Application Halts TLP Delivery[b]	Yes	Yes	No
Credit Manager	Controller	Controller	Your application. Not necessary if infinite credits advertised.
Ordering Rules	Obeyed	Obeyed	Violated[c]
Error Checking	Controller forwards TLP to your application logic after checking is complete and error-free	Controller forwards TLP to your application logic before checking is complete	Controller forwards TLP to your application logic before checking is complete
Corrupt TLP Discarded By	Controller[d]	Your application[e]	Your application[f]

a. Cut-through mode decays into store-and-forward mode when the queues become full; if there is more than one TLP in the buffer when the queue is in cut-through mode, the TLPs being received are treated the same as store-and-forward mode. Only the first TLP in the queue behaves in cut-through mode.
b. For more information, see "TRGT1 Packet Grant and Halt" on page 371.
c. "CPL must not pass a previously-issued P" rule is violated, which is only a problem when you have a real "Producer-Consumer" scenario. For more information, see "PCIe Ordering Rules" on page 704.
d. This is the default operation. You can optionally configure the controller to forward the corrupt TLP (not recommended). For more information, see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.
e. This is the default operation. You can optionally configure the controller to forward the corrupt TLP (not recommended). For more information, see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.
f. When the controller signals an error or abort indication to your application at the end of the TLP, you must discard that TLP. For more information, see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.

The following tables show which queue modes can be used in which controller configurations. You can set the queue mode independently for each VC.

Table 2-11 Supported Queue Modes for 32-bit, 64-bit, 128-bit, and 512-bit Datapath (Default is Marked D)

	S/F	C/T	B/P
Posted	D	Y	Y[a b]
Non-Posted[c]	D	Y	-
Completion	Y	Y	D

a. In this configuration you cannot memory map the port logic registers.
b. B/P queue mode is not supported for 512-bit datapath configuration.
c. You cannot put the non-posted receive queue in bypass mode because the controller needs to manage credits for both internally consumed non-posted TLPs (for example CFG requests to internal registers in the CDM), as well as credit returns that are stalled while the controller is waiting to return completions to the remote link partner.

Table 2-12 Supported Queue Modes for 256-bit Datapath (Default is Marked D)

	S/F	C/T	B/P
Posted	Y	D	-

	S/F	C/T	B/P
Non-Posted	Y	D	-
Completion	Y	D	-

Queue Modes (AXI Bridge)

When you are using the AXI bridge, the controller takes care of all error handling, credit handling, and rollback actions for you.

Attention Receive queues in Cut-Through (C/T) are not fully supported for DMA configurations. For more information, contact Synopsys through SolvNetPlus.

The following tables show which queue modes can be used in which controller configurations. You can set the queue mode independently for each VC.

Table 2-13 Supported Queue Modes for 32-bit, 64-bit, 128-bit, and 512-bit Datapath (Default is Marked D)

	S/F	C/T	B/P
Posted	D	-	-
Non-Posted[a]	D	-	-
Completion	Y	Y[b]	D

a. You cannot put the non-posted receive queue in bypass mode because the controller needs to manage credits for both internally consumed non-posted TLPs (for example CFG requests to internal registers in the CDM), as well as credit returns that are stalled while the controller is waiting to return completions to the remote link partner.
b. Use this configuration when you want completions to obey the Producer-Consumer ordering rule.

Table 2-14 Supported Queue Modes for 256-bit Datapath (Default is Marked D)

	S/F	C/T	B/P
Posted	D	-	-
Non-Posted	D	-	-
Completion	D	Y	-

The queue configuration for the Gen6 configured controller has the following exceptions:

Bypass Queue Mode:

- Only Completion queue can be configured in this mode
- Optional IDE Security module is not permitted in the Rx data path
- Except 128-bit datapath all other datapath widths (256, 512, 1024) has buffer to store completions

Table 2-15 shows the queue mode support per TLP type.

Table 2-15 Gen6 Configured Controller: Supported Queue Modes (Default is marked D)

	S/F	C/T	B/P
Posted	Y	D	-
Non-Posted	Y	D	-
Completion	Y	Y	D

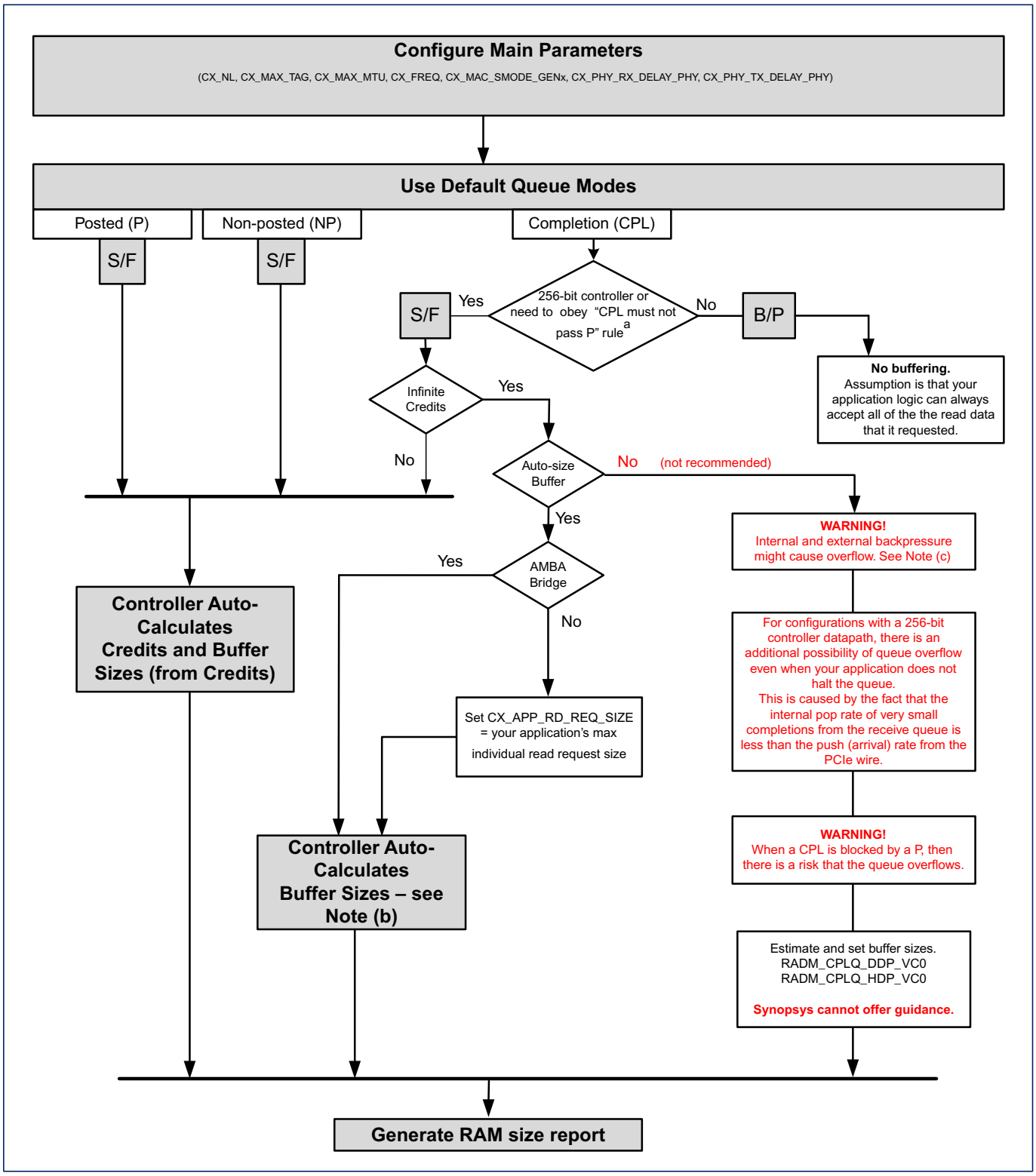
The optional IDE Security module processes one TLP per cycle. If enabled, Posted, Non-Posted, and Completion TLPs are serialized prior to being presented on the IDE Tx Interface. When the TLP has been processed the TLP is returned on the IDE Rx Interface, de-serialized and routed to the appropriate queue interface.

2.5.2 Configuring Your Queues

The coreConsultant tool automatically calculates the native controller's receive queue buffer sizes.

The sizing of the AXI bridge queues is discussed in "AXI Bridge Architecture" on page 400.

Figure 2-34 Guide to Configuring Your Receive Queues



Note

1. For more information, see “PCIe Ordering Rules” on page 704.
2. Approximate auto-calculated total RAM size (bytes) for Native PCIe controller configuration is:

   size = (CX_MAX_TAG + 1) * (CX_APP_RD_REQ_SIZE + 1)

   Note: For exact size, see the coreConsultant configuration report.
3. Approximate auto-calculated total RAM size (bytes) for AMBA configuration is:
   - (without DMA) size = (CC_MAX_SLV_TAG+1) * (CC_SLV_BURST_LEN * SLAVE_BUS_-DATA_WIDTH/8)
   - (with DMA) size = (CC_MAX_SLV_TAG+1) * max { CC_SLV_BURST_LEN * SLAVE_BUS_-DATA_WIDTH/8 , CX_MAX_MTU}

   Note: For exact size, see the coreConsultant configuration report.
4. A CPL buffer that advertises infinite credits and that is configured in store-and-forward or cut-through modes, is not protected from overflow by the "flow-control: credit-return" protocol. If you do not size the buffer to accept all of the data that you have requested, the queue might overflow when the P queue is blocked and the controller halts delivery of CPLs to your application when it is enforcing the CPL must not pass P ordering rule. If you do not have a local Producer-Consumer scenario, you can set the RO bit of the outbound NP requests to 1 so that CPL can overtake P requests in the queue. For more information, see “PCIe Ordering Rules” on page 704.
5. In the unlikely event that the buffer RAMs overflow, the controller asserts the radm_qoverflow output. If overflow is caused by a lack of credits for incoming TLPs (causing a TLP to be discarded), the controller sets the Receiver Overflow bit in the AER Status Register.

3

Controller Operations

This section describes the operations of the PCI Express controller. The topics in this section are:

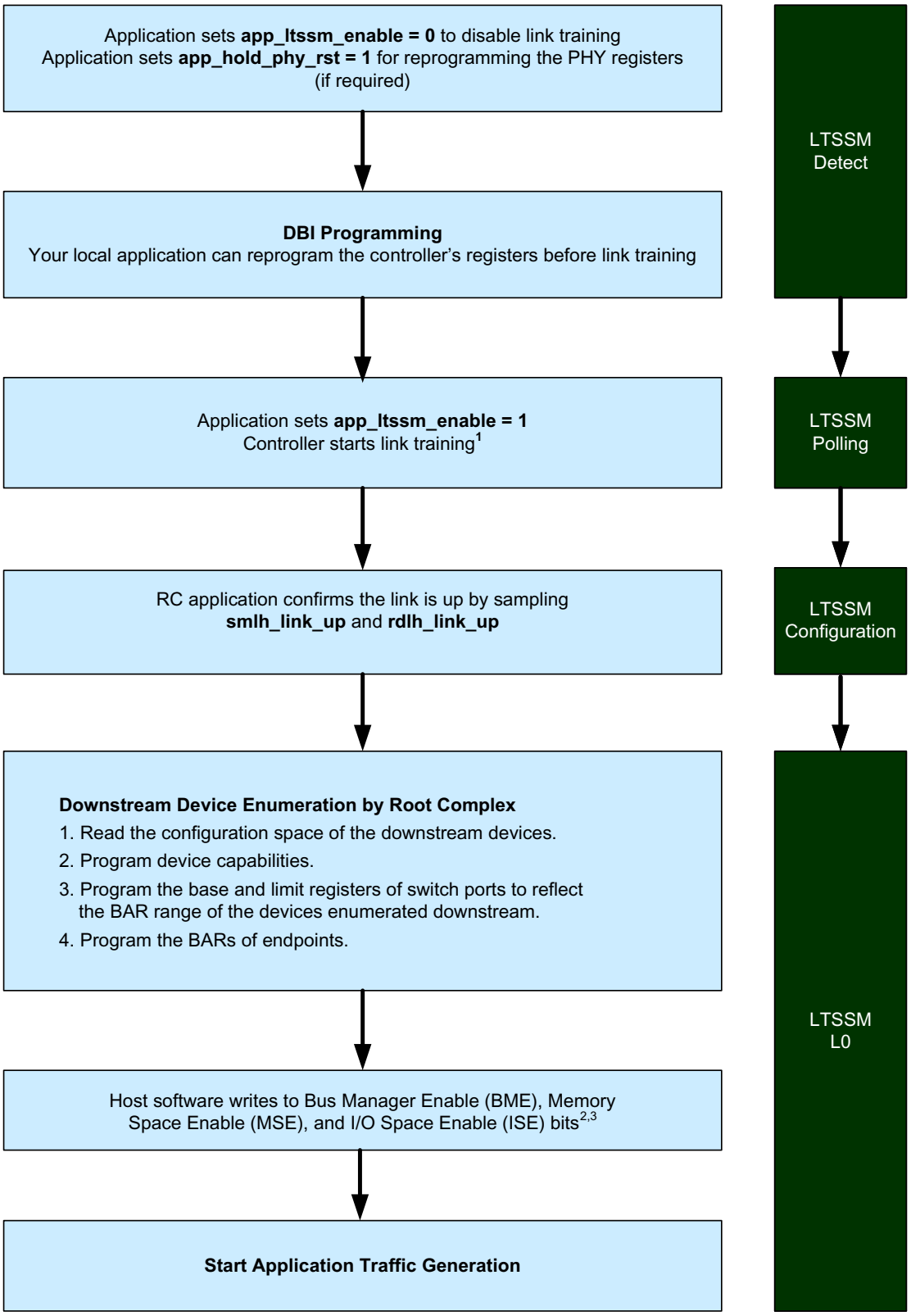
- "Initialization" on page 109
- "Link Establishment" on page 115
- "Transmit TLP Processing" on page 120
- "Receive TLP Processing" on page 134
- "TLP Orthogonal Header Content (OHC)" on page 138
- "Register Module, LBC, and DBI" on page 141
- "Reliability, Availability, and Serviceability (RAS)" on page 162
- "Messages" on page 193
- "Interrupts" on page 210
- "Flow Control" on page 216
- "Retry Mechanism" on page 219
- "Internal Address Translation Unit (iATU)" on page 220
- "Address Translation Services (ATS)" on page 248
- "Page Request Services (PRS)" on page 250
- "Peer-to-Peer Support (P2P)" on page 251
- "Gen2/Gen3/Gen4/Gen5 Speed Modes" on page 252
- "Power Management" on page 259
- "Completion Timeout Ranges" on page 312
- "TLP Processing Hints (TPH)" on page 313
- "Atomic Operations (AtomicOps)" on page 314
- "TLP Prefix" on page 318
- "Separate Refclk Independent SSC (SRIS)" on page 321
- "Readiness Notifications (RN)" on page 322
- "Expansion ROM Validation" on page 324
- "Precision Time Measurement (PTM)" on page 325

- "Access Control Services (ACS)" on page 332
- "Management Component Transport Protocol (MCTP)" on page 334
- "Completion Queue Management" on page 349
- "Downstream Port Containment (DPC)" on page 335
- "Native PCIe Enclosure Management (NPEM)" on page 342

3.1 Initialization

For Gen6 configured controller with `CX_FLIT_MODE_ENABLED =1`, the default behavior is to attempt Linkup in Flit Mode at Gen6 speed. To set different target speed, or Non Flit Mode, software must program controlling registers through the DBI interface during the "DBI Programming" phase as mentioned in Figure 3-35. For Downstream Ports, the Hierarchy Segment ID number must be provided through the `app_segment_id` input signal prior to link up.

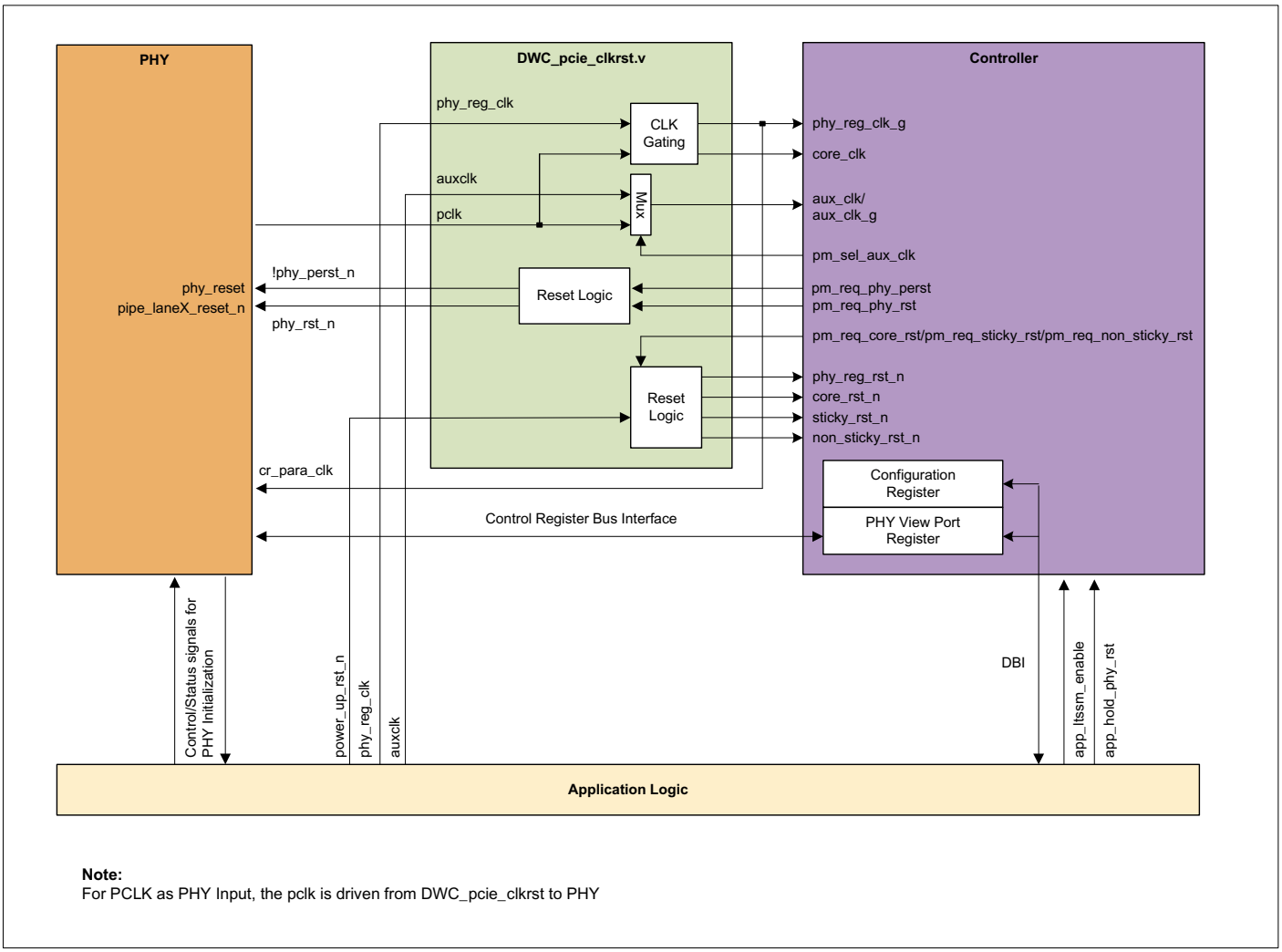
Figure 3-35 PCIe Initialization Steps



Note

1. For multiple VC configurations, only VC0 should be enabled before the link is up. Other VCs should be enabled after rdlh_link_up is asserted.
2. The application logic must not generate any MEM or I/O requests until the host software has enabled the BME. The Synopsys controller does not check this bit before transmitting requests. Therefore, your application logic must monitor the status of the BME on the `cfg_bus_master_en` output (or read it over the DBI).
3. For an RP with Type1 function, the controller does not block the received Requests or handle them as UR. Your application logic must check with the `cfg_bus_master_en` signal to block the upstream Mem or I/O requests and treat them as UR.
4. RP application logic must not generate any MEM or I/O requests until the host has enabled the MSE and ISE bits. The RP controller does not check these bits before transmitting requests. Therefore, your application should monitor the status of the MSE on the `cfg_mem_space_enable` output. There is no corresponding ISE output, so the RP application must be aware of when the host software writes to this bit. Your application could also read these bits over the DBI.

Figure 3-36 PHY Initialization



Attention The controller's PHY Viewport (PRBI) to PHY registers can have high access latency. If your application need slow latency to initialize Synopsys PHY registers or SRAM, then PRBI should not be used. Instead, you should design a direct interface to the PHY's SRAM or CR Bus. PRBI is recommended only for PHY debug.

For more information on cold reset sequence with PHY initialization see Figure 2-26 on page 93. For more information on PHY initialization after cold reset, see Figure 2-27 on page 94.

3.1.1 Bus and Device Number

For Downstream Ports, you must configure the bus and device number using the `app_bus_num` and `app_dev_num` input signals to set the bus and device number respectively, in the Requester ID. The `PL_APP_BUS_DEV_NUM_STATUS` register reflects the value of the bus and device number configured by your application.

3.1.2 Segment ID Number

For Gen6 configured controller in Flit mode, link Segment IDs are required on some TLP types and optional on others.

Upstream ports capture the Segment ID from reception of a CfgWr0 TLP. The value of the `DEV3_CAP_SEGMENT_CAPTURED` field in the `DEVICE_STATUS3_REG` is exported to the application through the output port `cfg_segment_id`.

For Downstream Ports that support Flit Mode, you must configure the Segment ID number using the `app_segment_id` input signal.

Segment IDs are populated as shown in Figure 3-38 to Figure 3-40.

Figure 3-37 Segment ID IO Signals

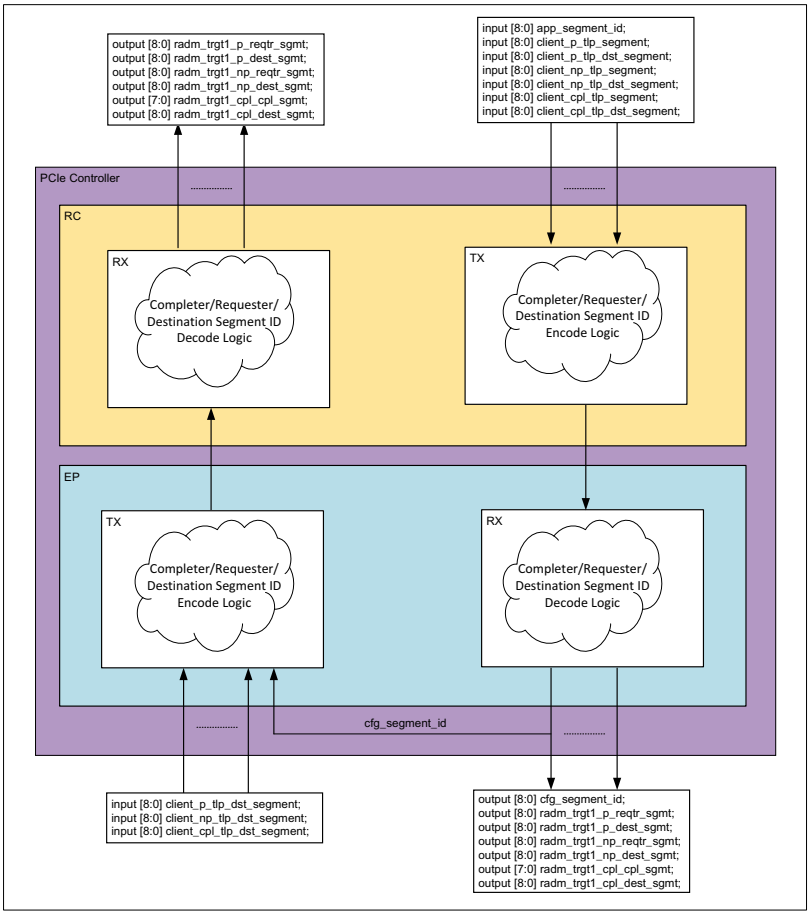


Figure 3-38 TLP Completer Segment for Root Port

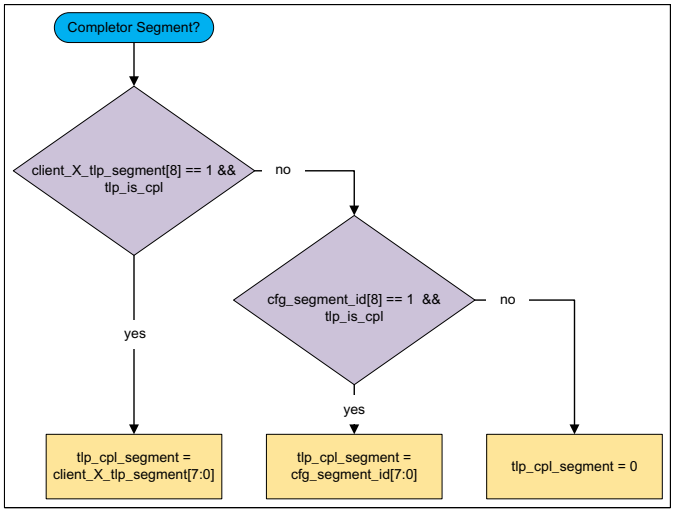


Figure 3-39 TLP Requester Segment for Root Port

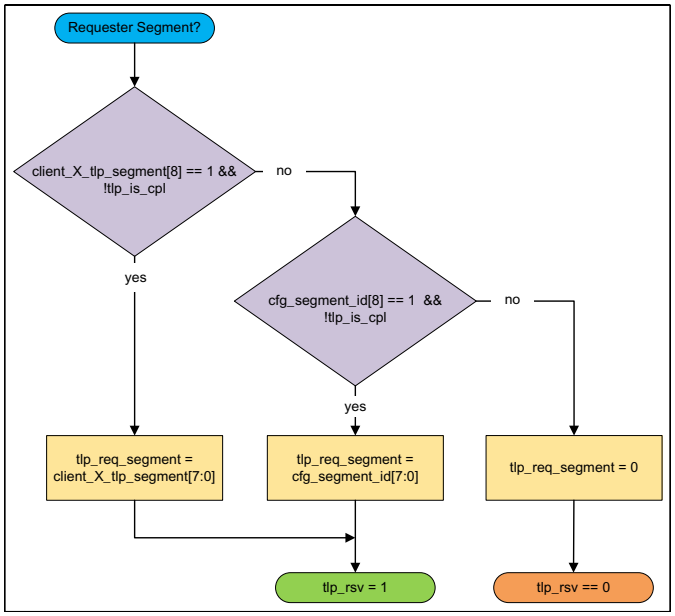
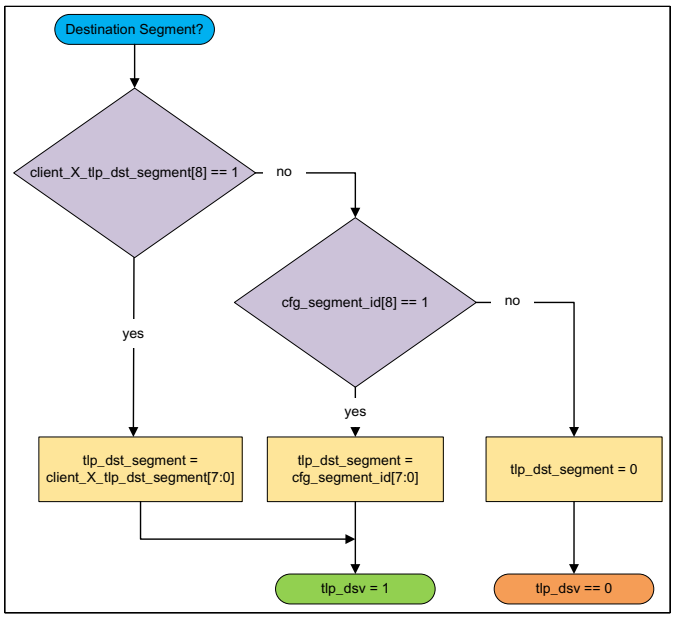


Figure 3-40 TLP Destination Segment for Root Port



3.2 Link Establishment

The controller implements the LTSSM function according to the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The controller supports link widths of x1, x2, x4, x8, and x16. It does not support x12.

When you have unused lanes in your system, you must turn off unused lanes using the hardware and software procedures outlined in "Advanced Information: How to Turn Off Unused Lanes" on page 711.

3.2.1 Lane Resizing

Initially, the controller tries to linkup at maximum width; you can restrict the maximum width as follows:

1. Program the `LINK_CAPABLE` field of `PORT_LINK_CTRL_OFF`.
2. Program the `NUM_OF_LANES` field of `GEN2_CTRL`.

To conserve power, your PHY should ignore any signaling on the non-operational lanes. The controller supports the PIPE turn off feature by using the `TxElecIdle` and `TxCompliance` signals only when the link resizing happens through detect state, and not by the Upconfigure capability.

The link partners can change the link width after the link is up. Resizing (up or down) normally occurs if both link partners advertised the Upconfigure Capability bit. You must set the `UPCONFIGURE_SUPPORT` field in `MULTI_LANE_CONTROL_OFF` so that the controller advertises '1' in the Upconfigure Capability bit of the TS2 OS. Default for this field comes from `DEFAULT_UPCONFIGURE_SUPPORT` parameter. Regardless of the value of the Upconfigure Capability bit and the value of `PCIE_CAP_HW_AUTO_WIDTH_DISABLE` field in the `LINK_CONTROL_LINK_STATUS_REG` register, the *PCI Express Base Specification, Revision 6.0, Version 1.0* permits either of link partners to initiate downsizing with link reliability reason. It can be initiated by programming the `RELIABILITY_LINK_WIDTH_CHANGE_ENABLE` field, the `TARGET_LINK_WIDTH[5:0]` field and the `DIRECT_LINK_WIDTH_CHANGE` field of the `MULTI_LANE_CONTROL_OFF` register. The link is not be upsized to the original link width again if the controller performs downsizing with link reliability reason when Upconfigure Capability bit is '0'. Resizing options are shown in Table 3-16.

Table 3-16 Link Resizing Options

	Remote Partner Initiates **Link Remains Up** **L0 -> Recovery -> Configuration**	**Software Initiates Locally[a]** **Link Remains Up** **L0 -> Recovery -> Configuration**
Upsizing[b]	Yes	Yes
Downsizing	Yes[c]	Yes

a. Previous versions of the Databook described a method of link resizing by taking the link down, and transitioning the LTSMM from L0 -> Recovery -> Loopback -> Detect. The controller still supports that method for legacy designs (see previous version of databook for more information), but new designs should use the method described here.
b. Called UpConfigure in the PCI Express Specification.
c. If the lanes are reversed, your application must manually flip the lanes to achieve downsizing for some configurations. For more information, see "Link Down-sizing with Non-reversed/Reversed Lanes; and Wide/Narrow Ports" on page 655.

The following steps show how you can initiate link resizing for non-reliability reason:

1. Ensure the link is in the L0 LTSSM state.
2. Program the `TARGET_LINK_WIDTH[5:0]` field of the `MULTI_LANE_CONTROL_OFF` register.
3. Program the `RELIABILITY_LINK_WIDTH_CHANGE_ENABLE` field of the `MULTI_LANE_CONTROL_OFF` register to 0.

4. Program the `DIRECT_LINK_WIDTH_CHANGE`[1] field of the `MULTI_LANE_CONTROL_OFF` register.

It is assumed that the `PCIE_CAP_HW_AUTO_WIDTH_DISABLE` field in the `LINK_CONTROL_LINK_STATUS_REG` register is 0.

Note For Gen3 and above, the controller does not send the first EIEOS before the first TS1 OS in Configuration.Linkwidth.Start state for the upconfiguring lanes because of the requirement that TxDataValid and TxElecidle signals must be aligned with other active lanes. After the first EIEOS, the upconfiguring lanes send all Ordered Sets including EIEOS, in the same manner as the other active lanes.

3.2.2 Lane Reversal and Flipping

The controller supports lane *reversal* and lane *flipping*. Depending on the configuration of the link partners and the topology of the link between them, the controller attempts to reverse and/or flip lanes to form a link in any of the following scenarios:

- The lanes are reversed by the physical layout of the PCIe link or chip package pinning.
- When connecting a narrower link partner into upper lanes of a wider controller.
- A lane is non-operational (broken).

  A lane is said to be non-operational if it cannot be detected in Detect LTSSM state or it cannot receive in Polling LTSSM state. This might be caused by any of (i) break on PCB (ii) electrical fault in transmitter (iii) electrical fault in receiver.

Lane auto *flip* occurs in Detect and is useful to connect physical lane `CX_NL-1` (or `CX_NL/2-1` or `CX_NL/4-1`) to logical Lane0 of the controller to form a link when physical Lane0 cannot be detected.

Lane auto *reverse* occurs in Configuration and is useful to form a link when logical Lane0 receives TS Ordered Sets from the link partner with the Lane Number field different from '0'.

Note To enable lane auto flip and reverse support:

- You must set configuration parameters `CX_LANE_FLIP_CTRL_EN =1` and `CX_AUTO_LANE_-FLIP_CTRL_EN =1`
- You must set `GEN2_CTRL_OFF` register fields `AUTO_LANE_FLIP_CTRL_EN =1` and `PRE_DE-T_LANE =0`

Figure 3-41 to Figure 3-43 show the muxing logic implemented in the controller as a function of parameters CX_LANE_FLIP_CTRL_EN and CX_AUTO_LANE_FLIP_CTRL_EN.

1. The controller clears the contents of this register after it has accepted the request.

Figure 3-41 Muxing Logic when CX_LANE_FLIP_CTRL_EN =1 and CX_AUTO_LANE_FLIP_CTRL_EN =1

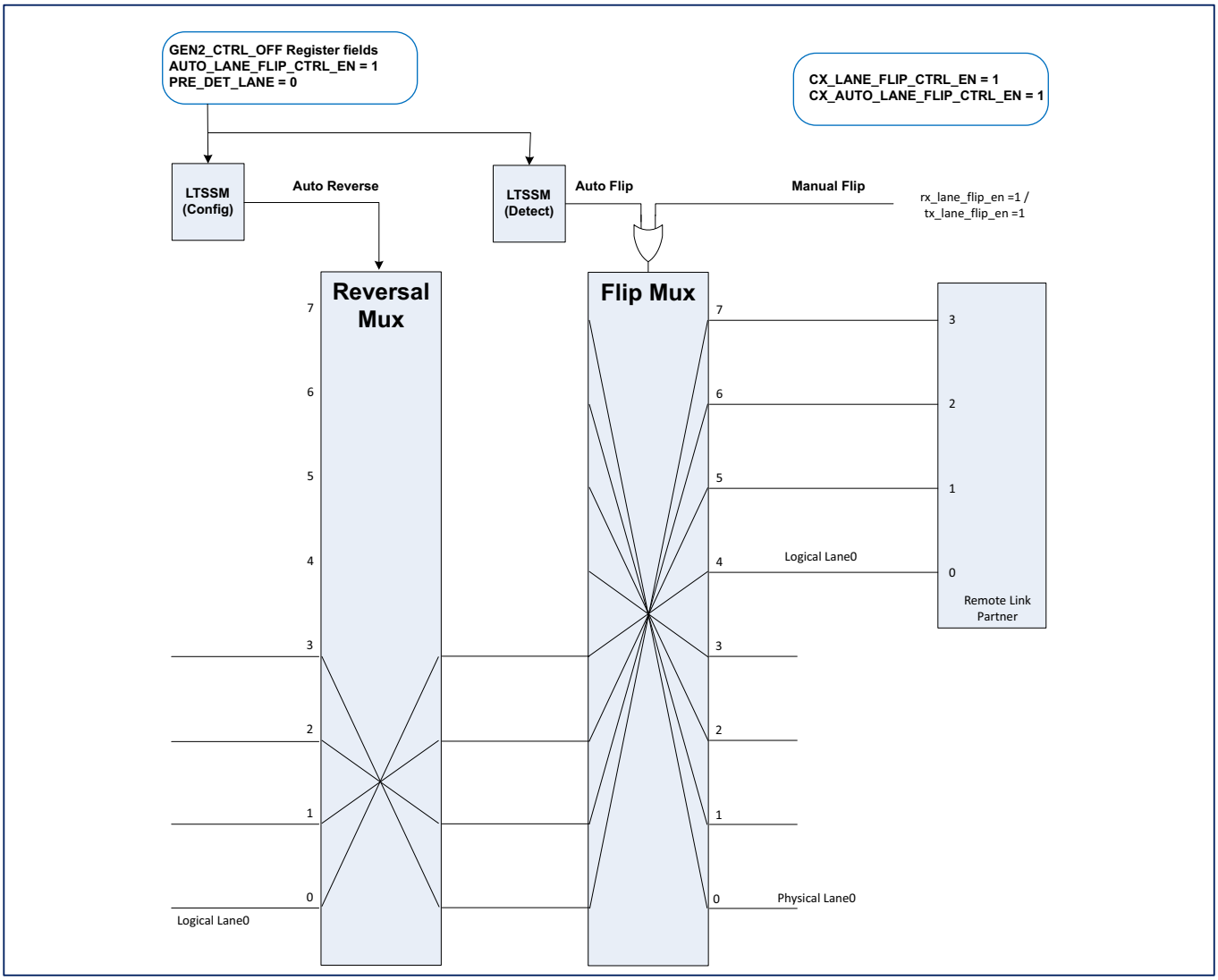


Figure 3-42 Muxing Logic when CX_LANE_FLIP_CTRL_EN =1 and CX_AUTO_LANE_FLIP_CTRL_EN =0

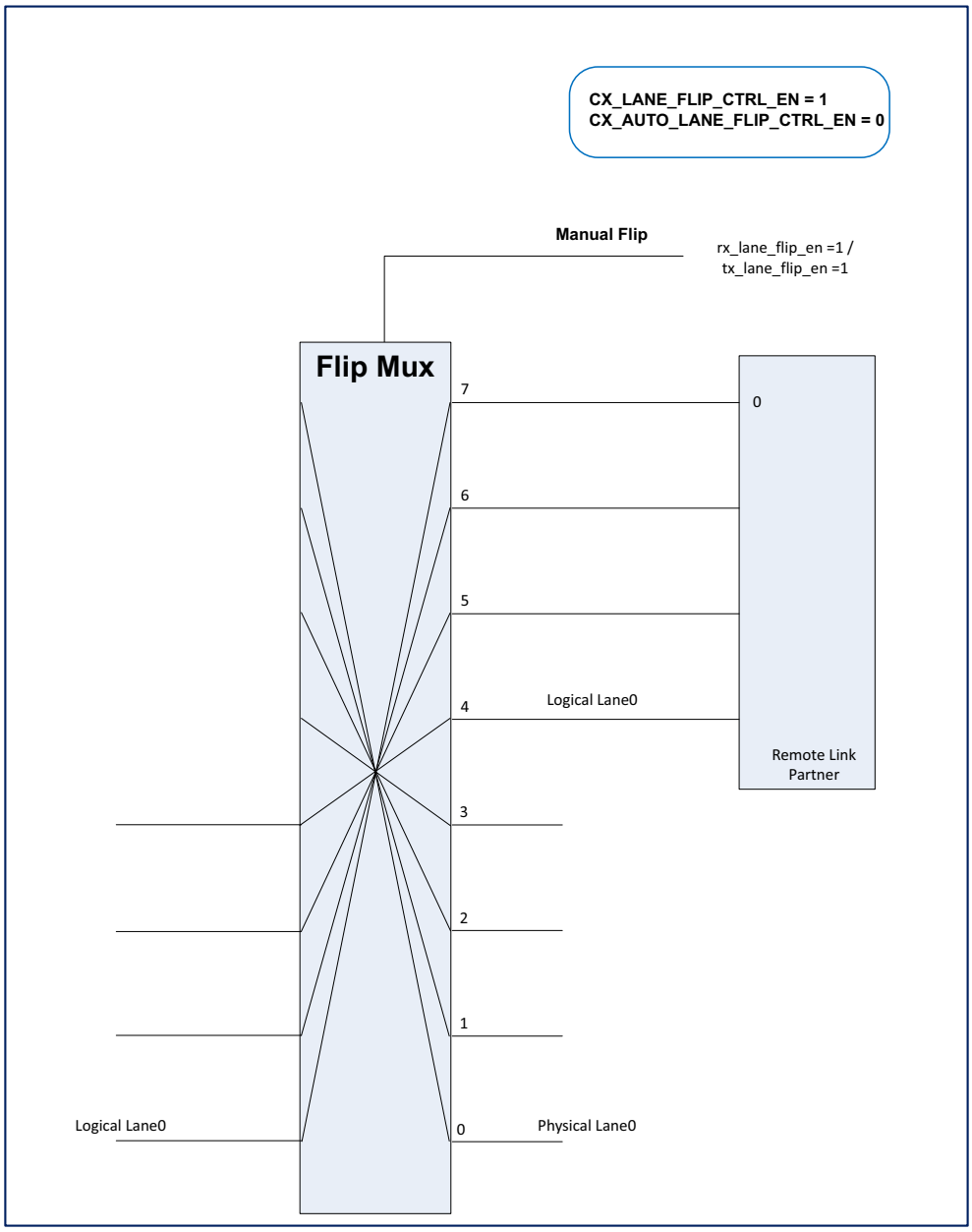
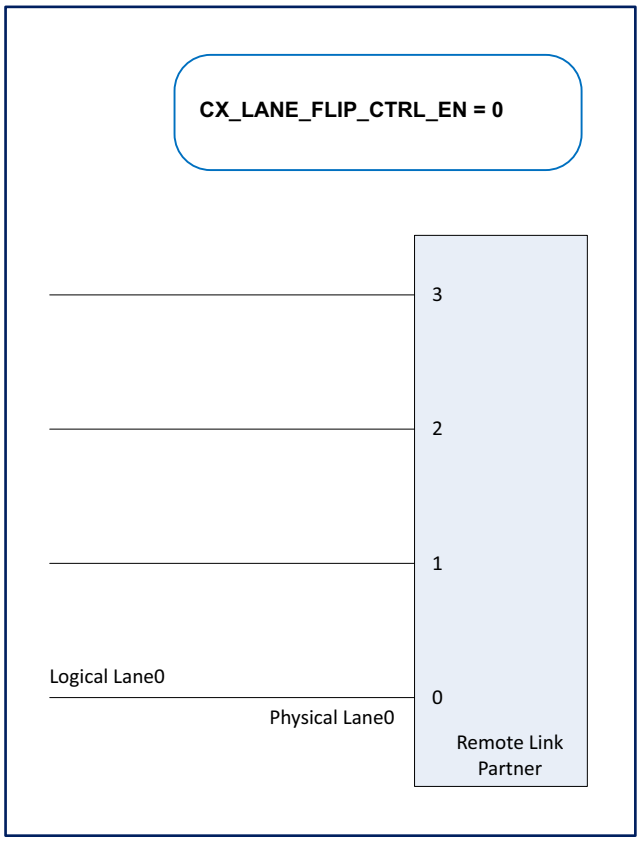


Figure 3-43 Muxing Logic when CX_LANE_FLIP_CTRL_EN =0



For more information, see "Advanced Information: Lane Reversal and Broken Lanes" on page 649.

Hint You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the "Integrating the Controller and VIP using VTB" section in the "Integrating the Controller with the PHY or Application RTL or Verification IP" chapter of the *User Guide*.

Force Lane Flip

When `CX_FORCE_LANE_FLIP_EN =1` and `GEN2_CTRL_OFF.FORCE_LANE_FLIP =1` you can forcefully connect the physical lane specified in `GEN2_CTRL_OFF.LANE_UNDER_TEST` tologicalLane0 of the controller after reset. Only x1 link can be formed using the force lane flip feature. This feature overrides all the other laneflip and reversal mechanisms.

The following conditions must be met for force lane flip to be effective:

- Parameters `CX_LANE_FLIP_CTRL_EN` and `CX_AUTO_LANE_FLIP_CTRL_EN` must be 'true'
- `PORT_LINK_CTRL_OFF.LINK_CAPABLE` must be set to '1'

3.3 Transmit TLP Processing

The following section describe the flow of transmit TLPs through the controller.

- "Transmit TLP Arbitration"
- "ACK/NAK Scheduling in Non-Flit Mode" on page 125
- "Transmit Replay" on page 126
- "Tx Interface Progression Detection (CX_INTERFACE_TIMER_EN =1)" on page 128
- "MultiStream Transmission" on page 130

Note

- The controller does not check the TLP for errors.
- The native PCIe controller does not check that the TLP payload size is less than the Maximum Payload Size limit.

  However, the AXI bridge module (if present) does guarantee that this limit is not exceeded.
- The native PCIe controller does not check that the TLP payload size is less than CX_MAX_MTU limit. Exceeding this limit overflows the retry buffer, resulting in data corruption.

  However, the AXI bridge module (if present) does guarantee that this limit is not exceeded.

3.3.1 Transmit TLP Arbitration in Gen6

In Gen6 configurations, the link might be operating in Flit Mode or non-Flit Mode. In both Flit Mode and non-Flit Mode the TLP arbitration follows the same mechanism.

In Flit Mode DLLPs are inserted into non-IDLE Flits, ACKs or NAKs are carried in any non-IDLE Flit which is not an explicit sequence number Flit.

In non-Flit Mode the DLLP arbitration and insertion follows the same mechanism as described in "Transmit TLP Arbitration".

The TLP arbitration is implemented in a round-robin fashion where each input to the arbiter carries a specific Flow Control Type (CPL, P, NP). The arbitrated TLPs are packed into Flits respecting the Flit packing rules. The DLLP arbitration uses a dynamic priority arbiter to select the DLLP to forward for insertion to the Flit.

Figure 3-44 TLP DLLP Transmission

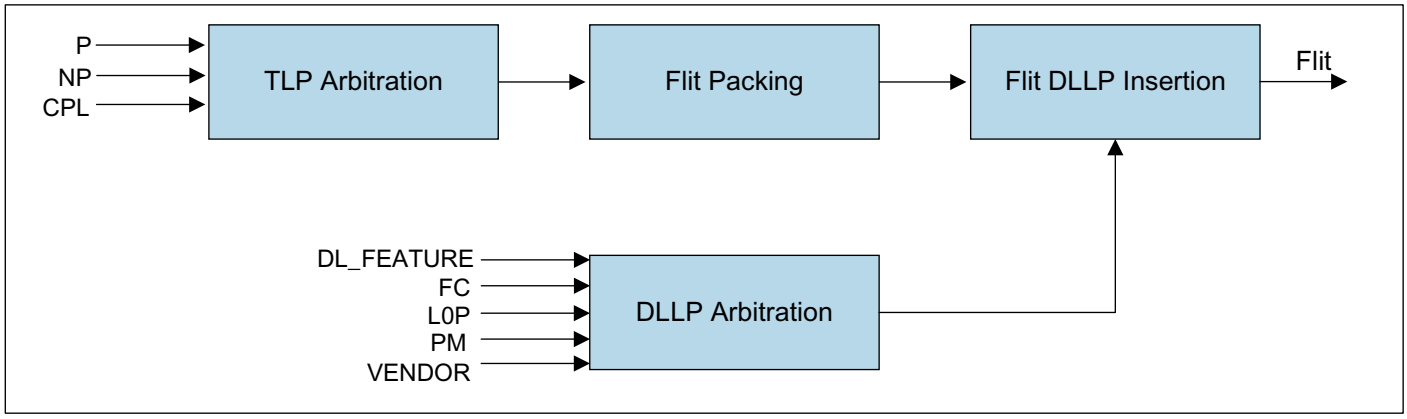


Table 3-17 shows the default priorities of DLLPs. The FC, PM, and Vendor Specific DLLPs are dynamic while DL Feature and L0P DLLPs remain fixed as highest priority.

Table 3-17 DLLP Arbitration Priorities

Priority	DLLP Type
1	DL Feature
2	L0P
3	FC
4	PM
5	Vendor Specific

In case the Flit is carrying Flit Marker, the DLLP presented by the DLLP arbitration is inserted in the next Flit after the Flit Marker has been sent.

In the event of a retry, Flits from the retry buffer take higher priority than Flits carrying TLPs. The DLLP arbitration and insertion behaves in the same manner with the DLLP being inserted into the Flits from the retry buffer.

3.3.2 Transmit TLP Arbitration

TLPs and DLLPs have equal priorities during transmit arbitration as shown in Figure 3-45. The priority of TLP types and DLLP types are shown in Table 3-18 and Table 3-19, respectively.

Figure 3-45 Transmit TLP/DLLP Arbitration

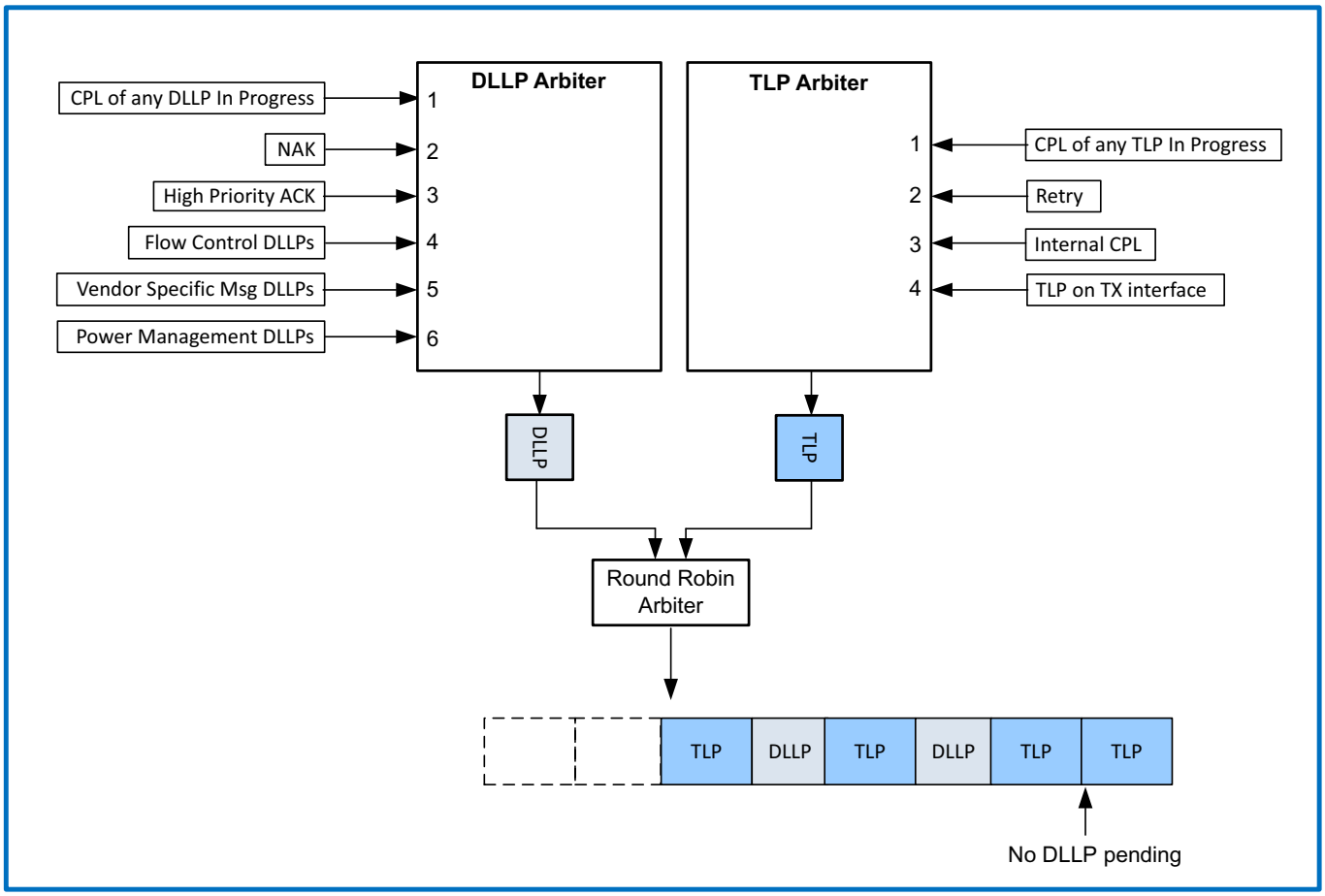


Table 3-18 TLP Arbitration Priority (1 = Highest, 4 = Lowest)

Priority	TLP Type
1	Completion of any TLP currently in progress
2	Retry buffer retransmissions
3	TLPs from the transaction layer (in the following order of priority): ■ Messages generated by the controller (including by your application through the MSI interface) ■ Completions generated by the controller for unsupported requests or completer aborts
4	TLPs on the transmit client interface (XALI0/1/2), or on the AXI Manager/Subordinate interface if the AXI bridge is being used

Table 3-19 DLLP Arbitration Priority (1= Highest, 6 = Lowest)

Priority	DLLP Type
1	Completion of any DLLP currently in progress

Priority	DLLP Type
2	NAK DLLP
3	High Priority ACK DLLP
4	Flow Control DLLP
5	Vendor Specific DLLP
6	Power management or any other Low Priority DLLP

3.3.2.1 Selecting Transmit Client Arbitration Scheme

All transmit client interfaces XALI0/1/2 are served using one of the three arbitration schemes when credit is available, regardless of the type of transaction. You can change the arbitration scheme using the `CX_XADM_ARB_MODE` configuration parameter when `CX_NVC >1`.

- **0: VC Based**. Provides a VC-based programmable weighted round robin arbitration (WRR) using two different arbitration methods for the two groups of VCs:
  - Strict Priority for the High-Priority VC (HPVC) group
  - RR or WRR for the Low-Priority VC (LPVC) group

  For more information, see "Advanced Information: VC-Based Arbitration" on page 685.
- **1: Round Robin (RR)**. Provides round robin arbitration between the three transmit clients. This is the default method.
- **2: Strict Priority**. Provides strict priority between the three transmit clients. XALI0 is lowest, XALI1 is higher, XALI2 (if implemented) is highest.

3.3.2.2 Effects of Flow Control Credits on Transmit Client Arbitration (Non-Multistream)

The controller checks that enough flow control (FC) credits are available in the remote device for the specific type of transaction (posted, non-posted, completion) 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.

For example when using the RR scheme, when a posted (P) transaction is presented onto XALI1 followed by a completion (CPL) on XALI0, and when credits permit, then the P transaction is transmitted onto the wire before the CPL. When the credit is not available then the CPL on XALI0 can pass the P transaction on XALI1 and be sent onto the wire. However, any non-posted (NP) or CPL TLPs on XALI1 (behind the blocked P) are blocked by the halted P. It is the responsibility of your application to make appropriate use of the three interfaces. There is no guarantee that order is preserved among client interfaces.

When your application is using a single transmit client interface for more than one TLP 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 might be available for that type. To avoid this situation, your application can use different transmit client interfaces for different request types (for example, XALI0 for posted requests, XALI1 for non-posted requests, and XALI2 for completions). Another way your application

can avoid this situation is to monitor the current FC credit availability on the `xadm_*_cdts` outputs from the controller and only generate requests for which FC credits are available.

When using `xadm_*_cdts` to monitor credit availability, your application must consider that it is possible that the controller might generate a message or completion TLP (which would change the number of credits available), between the time your application samples `xadm_*_cdts` and when your application generates the request.

To avoid this scenario, your application can use the Application Credit Control feature, for more information, see "Application Credit Control (Non-Multistream Architecture)" on page 124.

For more information on flow control, see "Flow Control" on page 216. For more information on TLP ordering, see "Advanced Information: Advanced Ordering Information" on page 703. For more information on how to select the client interfaces, see the "Configuration Guide" appendix in the *User Guide*.

Note

You can present any packet of any type on any interface at any time.

To maintain full throughput, the arbitration decision takes place prior to the presentation of the header of a packet on the XALI interface. Therefore, full throughput is only possible when sufficient credit of every type is available.

Due to pipelining the actual amount of credit required of each type is variable. To determine if sufficient credit is available, the controller waits for a TLP presentation. In this scenario, full throughput cannot be achieved.

Note

The DMA module when present, manages the use of these interfaces.

The AXI bridge module when present, manages the use of these interfaces.

3.3.2.3 Effects of Flow Control Credits on Transmit Client Arbitration (Multistream)

Similarly, to the Non-Multistream Architecture("Effects of Flow Control Credits on Transmit Client Arbitration (Non-Multistream)" on page 123), the controller checks for availability of enough flow control credits for the specific type of transaction before allowing the transmission of the TLP, prioritizing internally generated Completions and Messages. When the Multistream Architecture is used a TLP might be accepted at the XALI interface even in case of not enough credits for its transmission on the link. XALI interfaces are dedicated per VC/FC, and it is the responsibility of the application to drive the correct TLP VC and FC type on the correct interface.

3.3.2.4 Application Credit Control (Non-Multistream Architecture)

The optional application credit control feature (`APP_CREDIT_CTRL =1`) allows your application to monitor and control the credit expenditure. In addition to a reporting interface for all credit consumption, an interface is provided to grant your application control over transmission and credit consumption of internal controller generated messages and completions. Only when your application grants the credits for internally generated messages and completions, these TLPs enter arbitration. Using this feature your application can maintain an accurate credit count and define packet ordering to achieve the best performance when using a single transmit client interface for more than one TLP type (for example, posted and non-posted).

The credit control handshake between the controller and your application, as depicted in Figure 3-46, is as follows:

- Credit Request (controller => application)

  The controller requests your application to grant credits for internally generated messages/completions by asserting the output signals `xadm_crd_*_h_req`/`xadm_crd_*_d_req`/`xadm_crd_*_vc`[1].

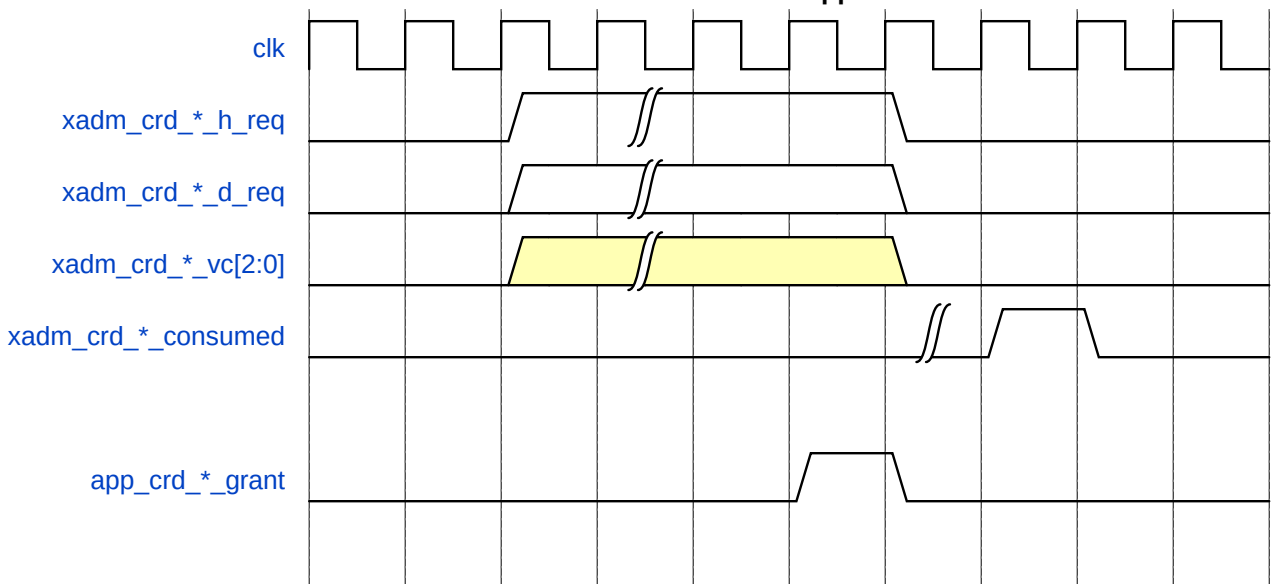
- Credit Grant (controller <= application)

  Only when your application grants the credits by asserting `app_crd_msg_grant`/`app_crd_cpl_grant` signals the internally generated messages/completions TLPs enter arbitration, and are immediately transmitted by the controller.

- Credit Consumption (controller => application)

  The output signal `xadm_*_consumed` indicates that the granted credit has been consumed by the controller.

Figure 3-46 Credit Control Handshake Between the Controller and Your Application



3.3.2.5 Application Credit Control (Multistream Architecture)

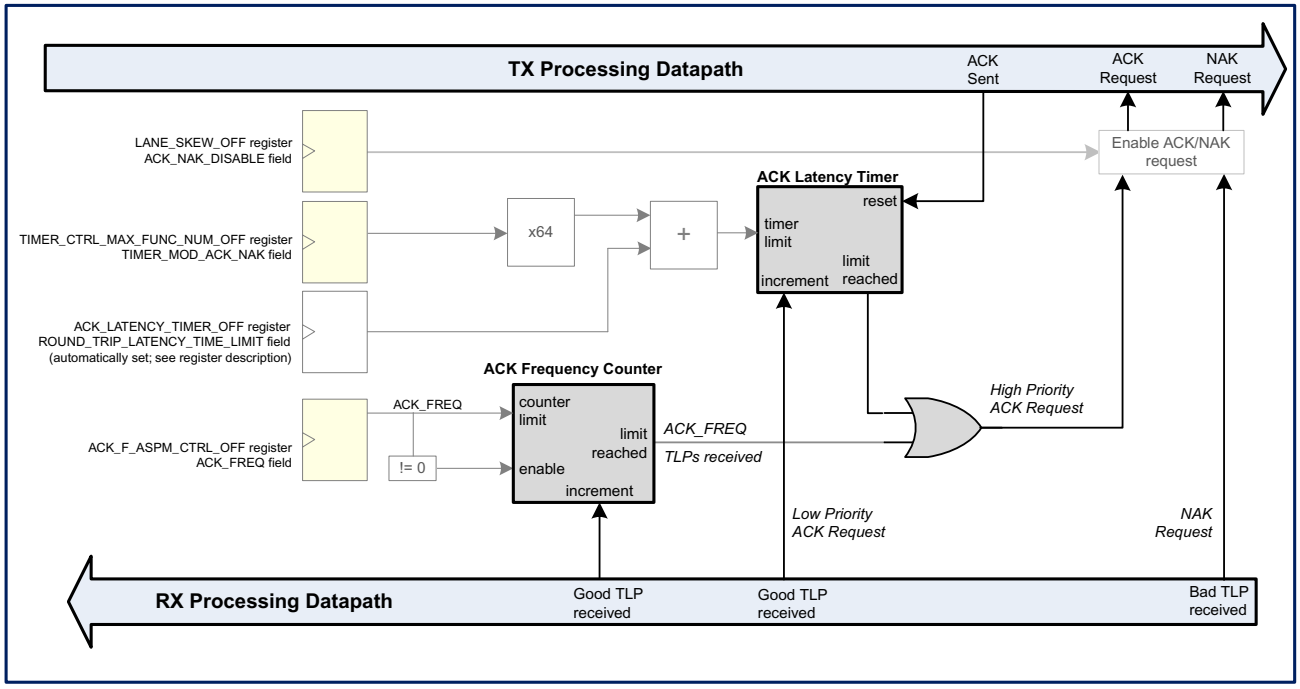
The Application Credit Control feature is not available when using the Multistream Architecture.

3.3.3 ACK/NAK Scheduling in Non-Flit Mode

Figure 3-47 shows how you can reprogram the controller to influence the sending of ACK and NAK DLLPs. When the link is idle, the controller sends an ACK DLLP every time it receives a good TLP. For a typical system, you do not have to modify the defaults, unless the remote device is executing unexpected replays.

1. Indicates the VC for the message/completion when multiple VCs exist.

Figure 3-47 Ack Frequency and Latency Timers Operation (Representative)



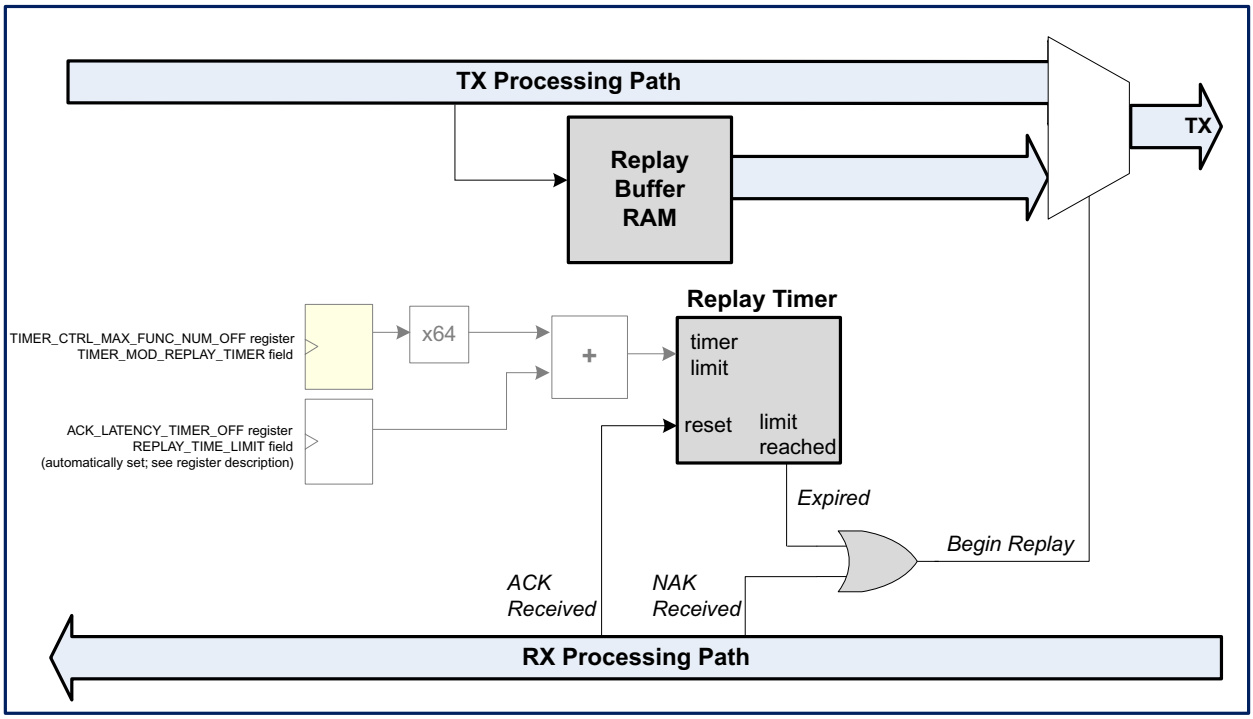
Note When a low priority ACK request is scheduled for transmission to the remote link partner, the controller waits until either the ACK Latency Timer or the ACK Frequency Counter expires, then converts the low priority ACK request to a high priority ACK request, and schedules it for transmission to the remote link partner.

3.3.4 Transmit Replay

3.3.4.1 Gen5 and Below Transmit Replay

You can modify the time-out value of the replay buffer as shown in Figure 3-48. For a typical system, you do not have to modify the default settings, unless the remote device is executing unexpected replays.

Figure 3-48 Replay Buffer and Time (Representative)



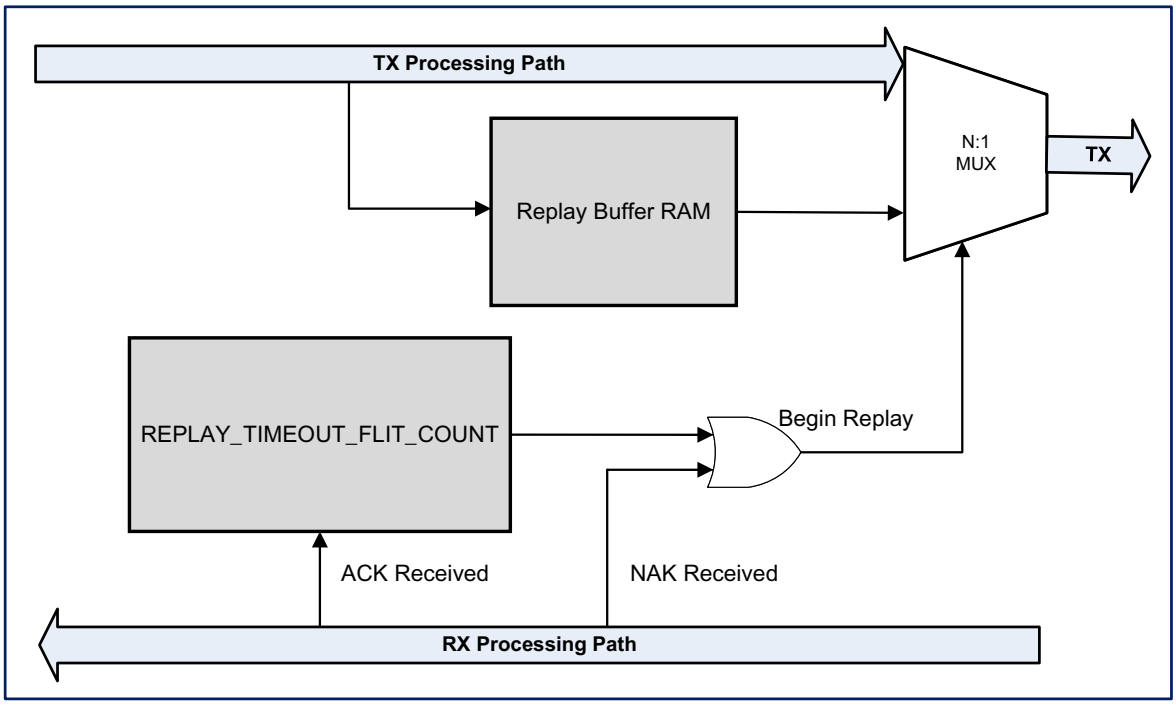
3.3.4.2 Flit Mode Transmit Replay

For Gen6 configured controller in Flit Mode, Flits are retried either due to

- Reception of selective or standard Nak for a Flit contained in the Replay Buffer RAM, or
- Expiration of the `REPLAY_TIMEOUT_FLIT_COUNT`, as defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

as shown in Figure 3-49.

Figure 3-49 Replay Buffer and Replay Flit Count Timeout (Representative)



3.3.5 Tx Interface Progression Detection (CX_INTERFACE_TIMER_EN =1)

This feature provides a mechanism for the controller to inform the application about the lack of progression on one of the data transmission interfaces XALI0/1/2 (without AXI bridge), or on AXI subordinate and manager interfaces. It also monitors the internal MSG_GEN interface and the LBC CFG completion interface. Progression on the data transmission interfaces is not expected to halt under normal operating circumstances; it might halt if soft errors or silicon failures occur, or if your application logic inadvertently hangs the TX or AXI application interfaces.

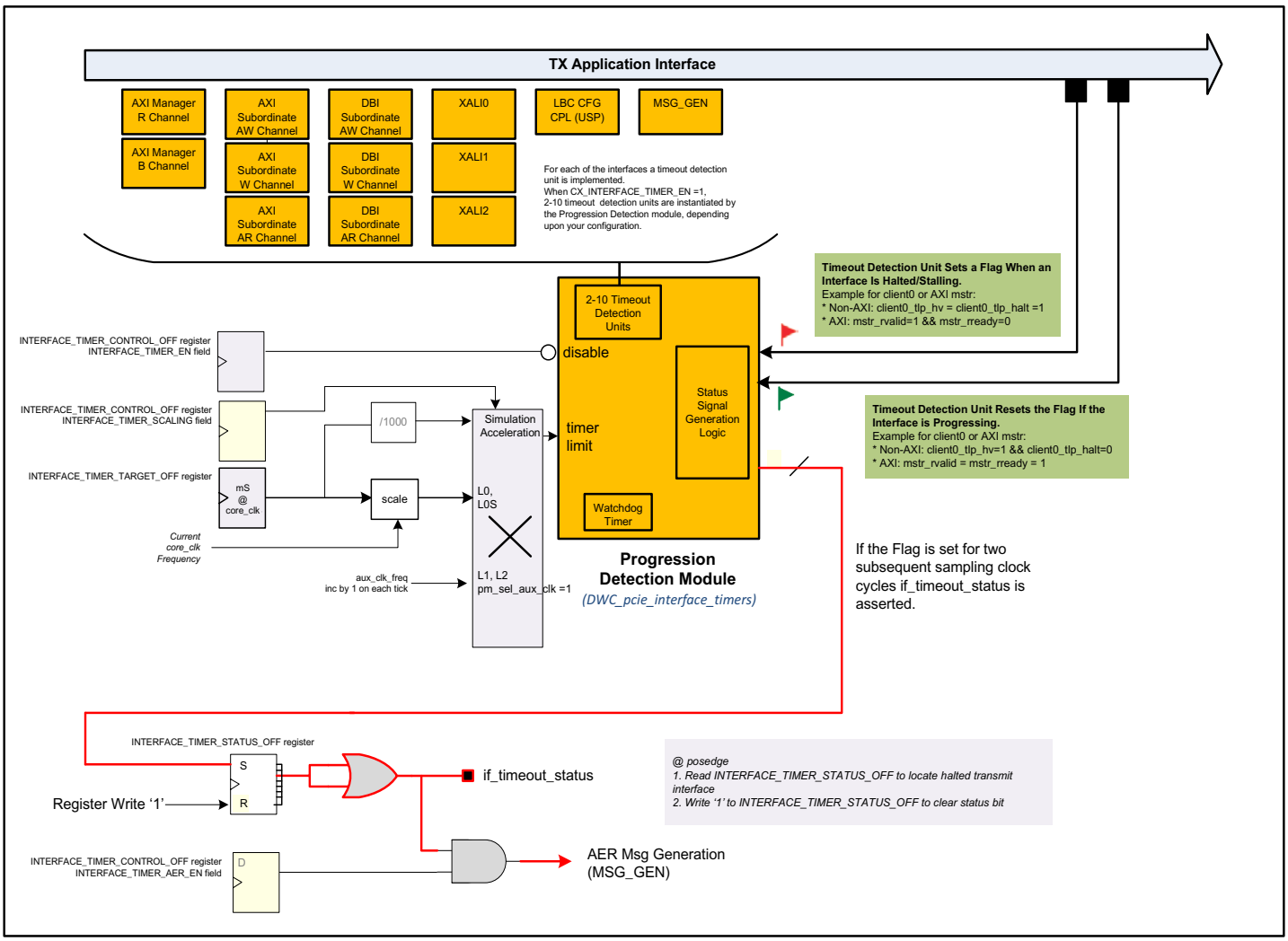
3.3.5.1 Tx Interface Progression Detection Features

- 2-10 timeout detection units instantiated depending on your configuration.
  - Timeout detection unit sets a flag when interface is stalling/waiting
  - Timeout detection unit resets the flag when interface is not stalling/waiting.
- Watchdog timer clocks the timeout detection units.
- Programmable watchdog timer monitoring window (through `INTERFACE_TIMER_TARGET_OFF` register).
- Controller automatically adjusts watchdog timer's incrementing interval when `core_clk` frequency changes after Gen1/2/3/4 speed change.
- Controller automatically adjusts watchdog timer's incrementing interval in L1 and L2 low-power states.
- Simulation acceleration mode where one micro-second is counted as one milli-second.
- Controller output `if_timeout_status` indicates a time-out event.
- Programmable generation of AER message after time-out event.

3.3.5.2 Tx Interface Progression Detection Operational Details

The interface progression timer module instantiates a watchdog timer and timeout detection units in the interfaces to be monitored. The application sets a window for monitoring transfer progression by programming the watchdog timer period through `INTERFACE_TIMER_TARGET_OFF` register.The watchdog timer signal is connected to the timeout detectionunits.When a change occurs in the watchdog timer signal, the timeout detection unit checks the interface signals to determine if the interface is halted, and sets a flag. For example, on XALI0/1/2,if `client*_tlp_hv =1` and `xadm_client*_halt =1` the timeout detection unit sets the flag. This flag is reset if the interface is progressing.If the flag is found to be set on the next change to the watchdog timer signal, interface timeout status signal (`if_timeout_status`) is asserted. This indicates to the application that a progression timeout has occurred on one of the interfaces. The application can then read the `INTERFACE_TIMER_STATUS_OFF` register to determine which interface is causing the timeout.

Figure 3-50 Tx Interface Progression Detection Operation



3.3.6 MultiStream Transmission

MultiStream Transmission provides multiple independent transmit interfaces. Each interface is capable of accepting traffic for transmission regardless of the state of other interfaces. The controller arbitrates between these interfaces within a VC and between VCs. A controller signal indicates when a TLP is transmitted.

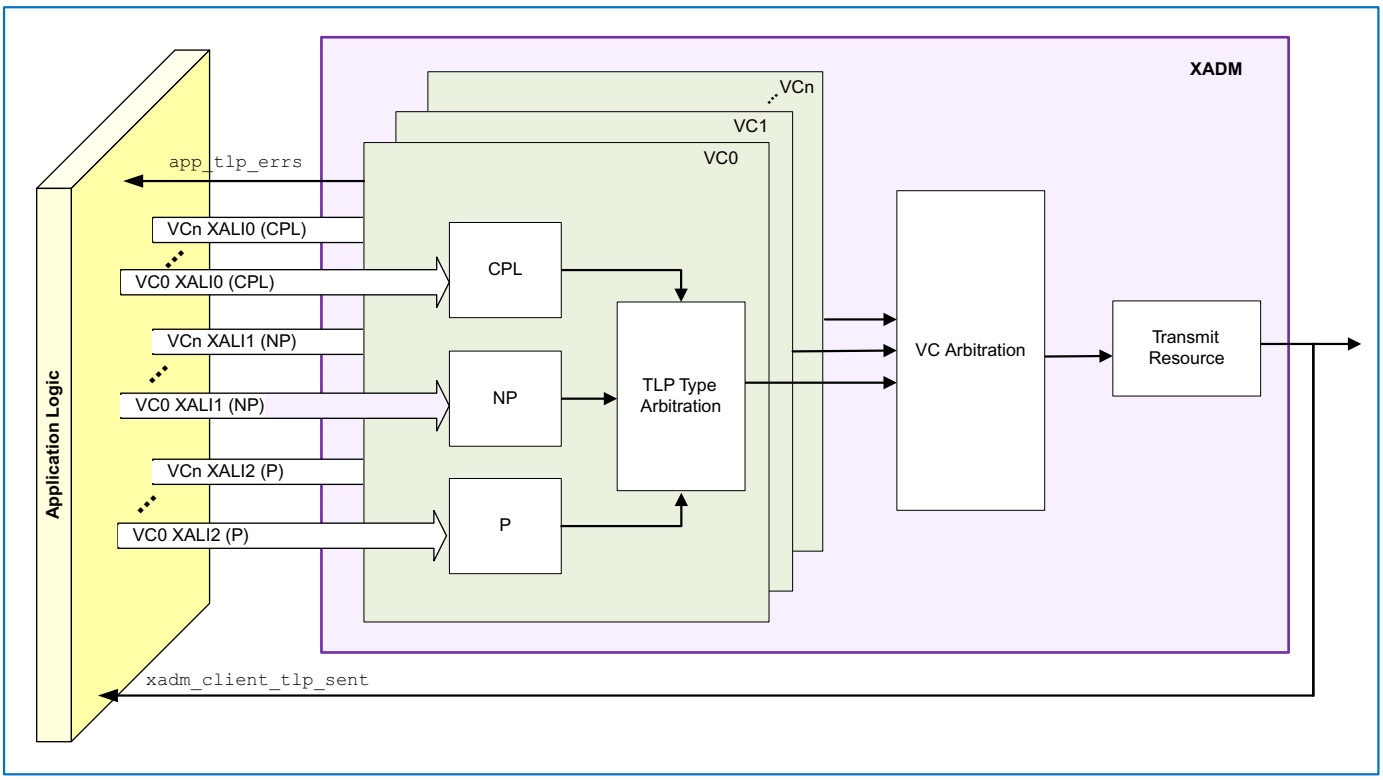
MultiStream transmission is enabled by setting the parameter `CX_MULTISTREAM` to '1'. When MultiStream is enabled dedicated logical XALI0/1/2 client interfaces are associated with each enabled virtual channel. The number of bits in each of the XALI0/1/2 interface signal varies based on the number of virtual channels enabled. The signal bits are split equally among the enabled virtual channels, thereby provisioning dedicated XALI0/1/2 client interfaces for each virtual channel. Dedicated XALI client interface for each virtual channel results in the asynchronous operation of virtual channels.

Each of the XALI interfaces handles a specific type of transaction. To transmit completion (CPL) transactions use XALI0, for non-posted (NP) transactions use XALI1, and for posted (P) transactions use XALI2. Using dedicated client interfaces for a specific type of TLP eases the timing and increases the performance, removing any dead cycle coming from possible interactions between the client interfaces.

3.3.6.1 TLP Arbitration

The controller accepts multiple transactions at the same time from the client interfaces and arbitrates them later on. The TLP to be transmitted is chosen after multiple arbitration stages within a VC and between VCs, as shown in Figure 3-51. Input TLPs are checked for consistency of their Traffic Class related to the interface Virtual Channel, to ensure that your application is transmitting correct TLP type on the client interfaces. The controller discards the TLP failing this check and generates an Uncorrectable Internal Error. The controller notifies your application of this event through the `app_tlp_errs` signal.

Figure 3-51 MultiStream Transmission TLP Arbitration



- **TLP Type Arbitration:** Within a VC, the TLP arbitration takes place according to TLP arbitration priorities described in Table 3-18. Internally generated messages and LBC completions have priority over application TLPs.
- **VC Arbitration**: Between VCs, the VC arbitration to select the TLP to be transmitted takes place as defined in section 6.3.3.2, "VC Arbitration-Arbitration Between VCs" of *PCI Express Base Specification, Revision 6.0, Version 1.0*. When MultiStream transmission is enabled, you can configure the VC Arbitration Table in the Virtual Channel Extended Capability structure of Function 0 to set the WRR phases. The WRR phases set for Function 0 apply to all the enabled functions.

  The controller arbitrates the VCs belonging to the Strict Priority group according to Strict Priority Arbitration Model. The controller arbitrates the VCs belonging to Low Priority Group, if present, using the Weighted Round Robin scheme configured in the VC Arbitration Select field of the Port VC Control and Status Register (that is, `VC_ARBI_SELECT` field of the `VC_STATUS_CONTROL_REG` register).

  - If multiple HPVCs are requesting, the controller grants TLP transmission right to HPVC with the highest VC ID
  - If no HPVC is requesting, and
    - If multiple LPVCs which have not used all the phases assigned to them in the VC Arbitration Table are requesting, the controller grants TLP transmission right to one of requesting LPVC in round robin order.
    - If all the requesting LPVCs have used all the phases assigned to them, the controller grants TLP transmission right to one of requesting LPVC in round robin order.

The controller may not transmit the TLPs in the order in which the XALI interface accepts them. The controller notifies your application about the arbitrated TLP using the output signal `xadm_client_tlp_sent`. Your application can use this information to enforce a specific order if required.

3.3.6.2 Gen6 MultiStream Transmission

The Gen6 configured controller contains independent transmit interfaces. Each XALI interface is tied to a single FC (Posted, Non-Posted, or Completion) and single VC. The minimum number of XALI interfaces is three (one Posted, one Non-Posted and one Completion).

The controller can be configured to have two sets of FC interfaces for the same VC. In this case you must specify the ordering relation for TLPs that belong to the same FC. A TLP must not include and gaps (bubbles) between the start of the TLP (`client*_tlp_hv`) and the end of the TLP (`client_*_tlp_eot`) Transmit TLP ordering information.

- There is no ordering relationship between TLPs from differing FC and VC
  - Posted can pass Non-Posted and Completion
  - Non-Posted can pass Posted and Completion
  - Completion can pass Posted and Non-Posted
- Once the TLPs have been serialized in the controller, buffered response is returned for each FC or VC
- When two interfaces exist for the same FC or VC (that is, two Posted interfaces for VC0), you must specify the order of the TLP through `client_vc<i>_p<n>_tlp_order_seqnum` input signal

TLPs can be stalled on the XALI interface if sufficient credits are not available to transmit the TLP. TLPs halted due to lack of credit can not cause a halt on a different FC TLP. For example, if Posted TLP is halted due to lack of credit, non-Posted and Completion TLPs continues to stream (assuming credit is available for these FCs). FC TLPs from the XALI interfaces are arbitrated using round-robin arbitration scheme.

Flit Mode Transmission

In a Flit Mode link (`CX_FLIT_MODE_ENABLED =1`), every Flit contains a DLLP. According to the protocol defined in Flit packing rules, the TLPs are framed into the Flit on the output of the Transaction Layer.

The controller does not transmit DLLP for Flits containing Flit Marker. In this case Flit Marker replaces DLLP bytes 5 to 2. When a Flit Marker is being transmitted the controller fills the remaining of the Flit after the TLP effected by the Flit Marker with NOPs.

3.3.6.3 Optional Features

- **Target Completion Lookup Table**: When Target Completion Look-up table is enabled, your application need not provide certain header information each time a CPL TLP is transmitted. For more information, see "Target Completion Lookup Table" on page 363.
- **Completion Queue Management:** When Completion Queue Management feature is enabled, non-posted TLPs are forwarded to the non-posted credits checker if RADM is in a position to receive a completion. For more information, see "Completion Queue Management" on page 349.
- **iATU:** When iATU is enabled, and there is a successful address match, the controller may convert posted TLPs on VC0 to non-posted TLPs. Type translation is allowed only for VC0 and TC change is not allowed. For more information, see "IO/MEM to Msg/MsgD Type Translation" on page 229.

3.3.6.4 Limitations

- DMA is not supported in this release.
- Application Credit Control (`APP_CREDIT_CTRL =1`) is not supported in this release.
- Client Pullback (`CLIENT_PULLBACK =1`) is not supported for MultiStream transmission.

3.4 Receive TLP Processing

This section describes the flow of received TLPs through the controller. The topics for this section are:

- "Receive Filtering"
- "Receive Routing" on page 135
- "Error Handling" on page 136

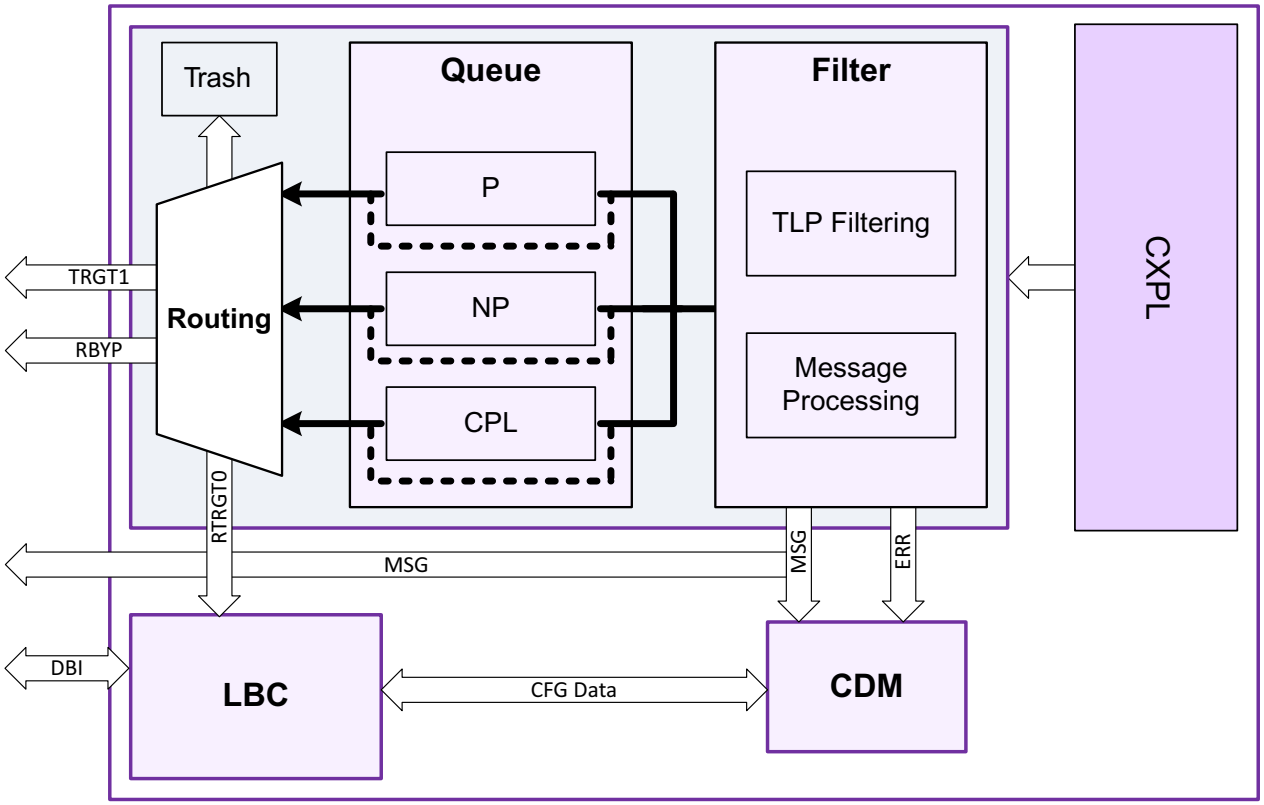
3.4.1 Receive Filtering

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

- Determine the status of a received TLP using filtering rules.
- Determine the destination of a received TLP based on the filtering status.
- Indicate the status of the received TLP using outputs.
- Report Errors to AER registers based on filter results. When more than one type of error is detected, Section 6.2.3.2.3, Error Pollution, of the *PCI Express Base Specification, Revision 6.0, Version 1.0* is followed.

The controller filters and routes received TLPs according to a set of rules determined by the TLP type based on the *PCI Express Base Specification, Revision 6.0, Version 1.0* and user-configurable filtering options. The filtering rules for a received TLP are affected by the configuration parameters (compile-time options), I/O signals (runtime options) and register values (runtime options).

Figure 3-52 RADM Block Diagram



The following general rules apply to all incoming TLPs that are not malformed. For more information on what happens to malformed TLPs, see "Error Detection for Received TLPs" on page 136. By default:

- For a function in device power states D1, D2, and D3hot, the controller only accepts CFG and MSG requests TLPs for that function. All other incoming request types for that function are treated as unsupported requests (UR).
- When the controller detects an error[1] in a received TLP, it normally performs the following:
  - Discards the TLP
  - Generates a completion (for non-posted requests) with the completion status set to CA or UR
  - Sets the status in the PCI-compatible Status register
  - Sets the status in the AER registers (when you enable AER)
- All error-free MSG requests are decoded internally, signaled on the SII interface and then dropped. When you want to have the decoded message also sent to the application interface, then see "Routing of Received Messages" on page 208.

Note For more information on advanced filtering, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691

3.4.2 Receive Routing

This section discusses how the RADM routes different TLP types to different receive interfaces depending on TLP type and filter result. When you are using the AXI bridge, then TRGT1 is replaced by the AXI bridge interfaces.

3.4.2.1 RP Routing Overview

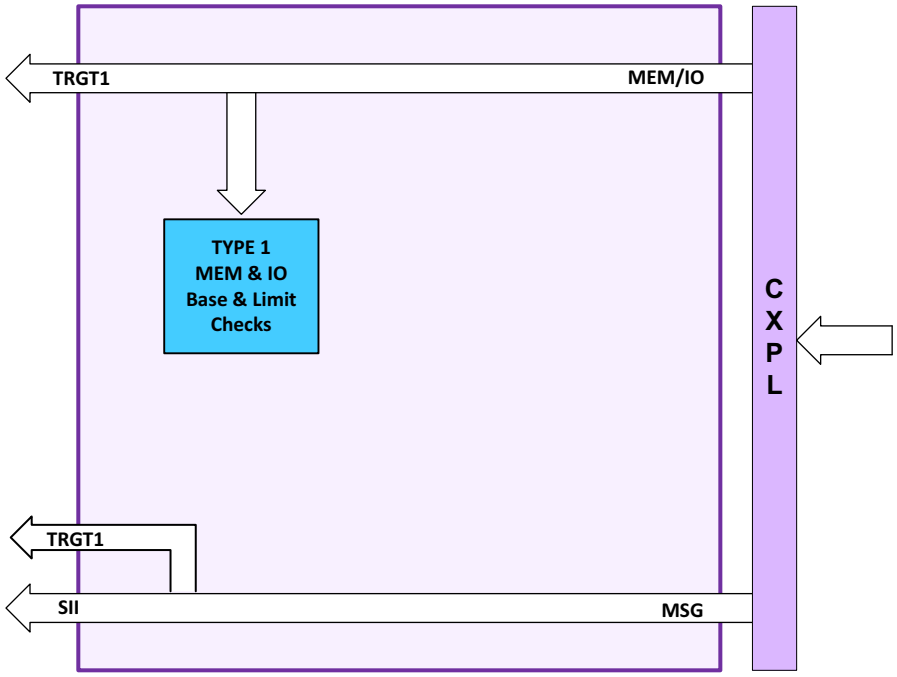
The possible destinations of a posted or non-posted request are TRGT1 and Discard[2]. By default:

- MEM requests outside of the memory range and prefetchable memory range as determined by the corresponding Base and Limit fields in the Type-1 header are routed to TRGT1.
- MSG requests are decoded internally, signaled on the SII interface, and then terminated.
- An RP does not expect to receive CFG or I/O requests.
- BARs are disabled and are not used.

The possible destinations of a completion TLP are TRGT1 and Discard.

1. Excluding TLPs targeted for forwarding (and not for local resources) that have ECRC errors.
2. Dropped or terminated. Serviced internally but not passed to your application

Figure 3-53 Default Request TLP Routing (Assuming no TLPs with CA/CRS/UR Error Status)



3.4.3 Error Handling

The controller supports the Baseline Capabilities, AER capabilities, and Advisory Non-Fatal Error Messaging as specified in Section 6.2, Error Signaling and Logging of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. These include Correctable and Uncorrectable (Fatal and Non-Fatal) errors.

For an RP, the reporting of most errors is internal to the root port. No external error notifications are generated. One exception to this (for example) is unsupported request (UR) completion status.

3.4.3.1 Error Detection for Received TLPs

The controller performs all mandatory error detections, the error reporting mechanism based on the *PCI Express Base Specification, Revision 6.0, Version 1.0*, and some optional error detections (using the `ENABLE_OPTIONAL_CHECKS` parameter). For more information of what error conditions contribute toward a UR or CA status, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691. When the controller detects an error[1] in a received TLP, it normally performs the following:

- Discards the TLP
- Generates a completion (for non-posted requests) with the completion status set to CA or UR
- Sets the status in the PCI-compatible status register
- Sets the status in the AER registers (when you enable AER)
- For malformed TLPs credit is returned based on the buffer space which has been consumed by the TLP

1. Excluding TLPs targeted for forwarding (and not for local resources) that have ECRC errors.

For more information see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.

3.4.3.2 AER Multiple Header Logging

The controller supports the AER multiple header logging as specified in Section 6.2.4.2, "Multiple Error Handling (Advanced Error Reporting Capability)" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

By default, the controller only logs the first header that caused an error. You can now set the number of logged headers using the `CX_HDR_LOG_DEPTH_n` parameter. The controller implements a FIFO queue of depth (`CX_HDR_LOG_DEPTH_n`) that interfaces with the AER Header Log registers (`HDR_LOG_i_OFF`). A valid queue entry is never overwritten. If your software does not read the header log registers fast enough, you can miss the new header log information when the queue is full.

3.5 TLP Orthogonal Header Content (OHC)

For the Gen6 Multistream Architecture, the controller supports the insertion of TLP OHCs (Orthogonal Header Content) as part of the support of Flit Mode. OHCs are additional header datawords (DWs) which are used to carry additional header information which could not otherwise be carried as part of the standard 3/4DW header for given Flow Control Types. The Transmitter TLP Header Formation Module has been updated to support these new OHC DWs. By default, OHCs are inserted for certain TLP types. Others can be inserted by way of a control register `OPTIONAL_OHC_CTRL`. For the fields for which OHCs must be inserted, the transmitter parses the incoming TLP Header Info/Prefix and extracts the relevant information. On detection of valid information, the appropriate OHCs are populated.

The Insertion process for the OHC-A type is given in Figure 3-54.

Figure 3-54 Insertion Process for OHC-A

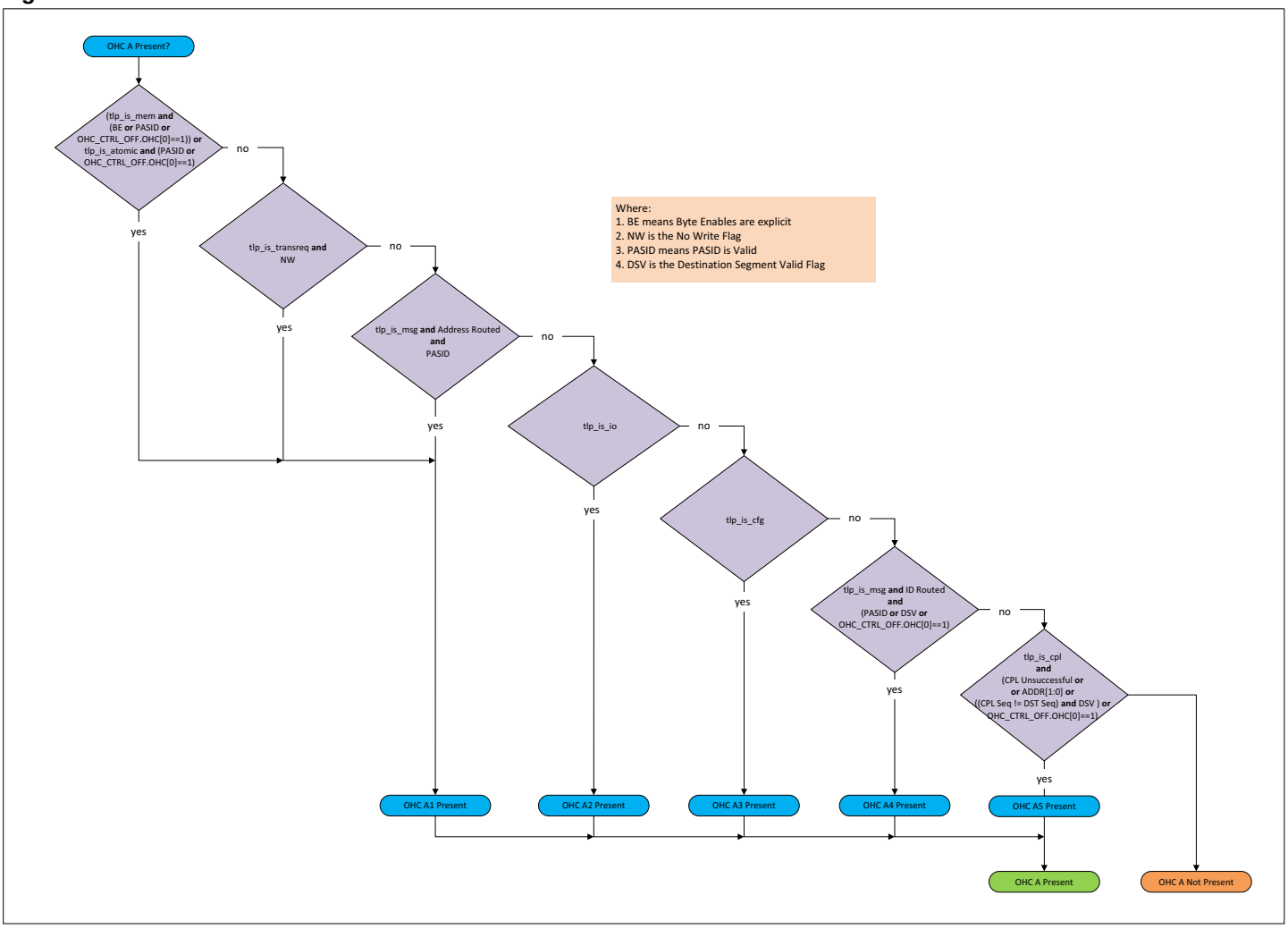


Figure 3-55 Insertion Process for OHC-B

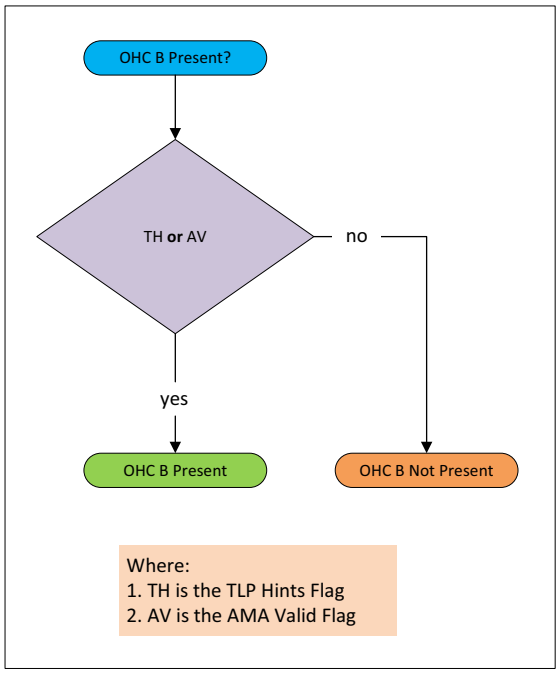


Figure 3-56 Insertion Process for OHC-C

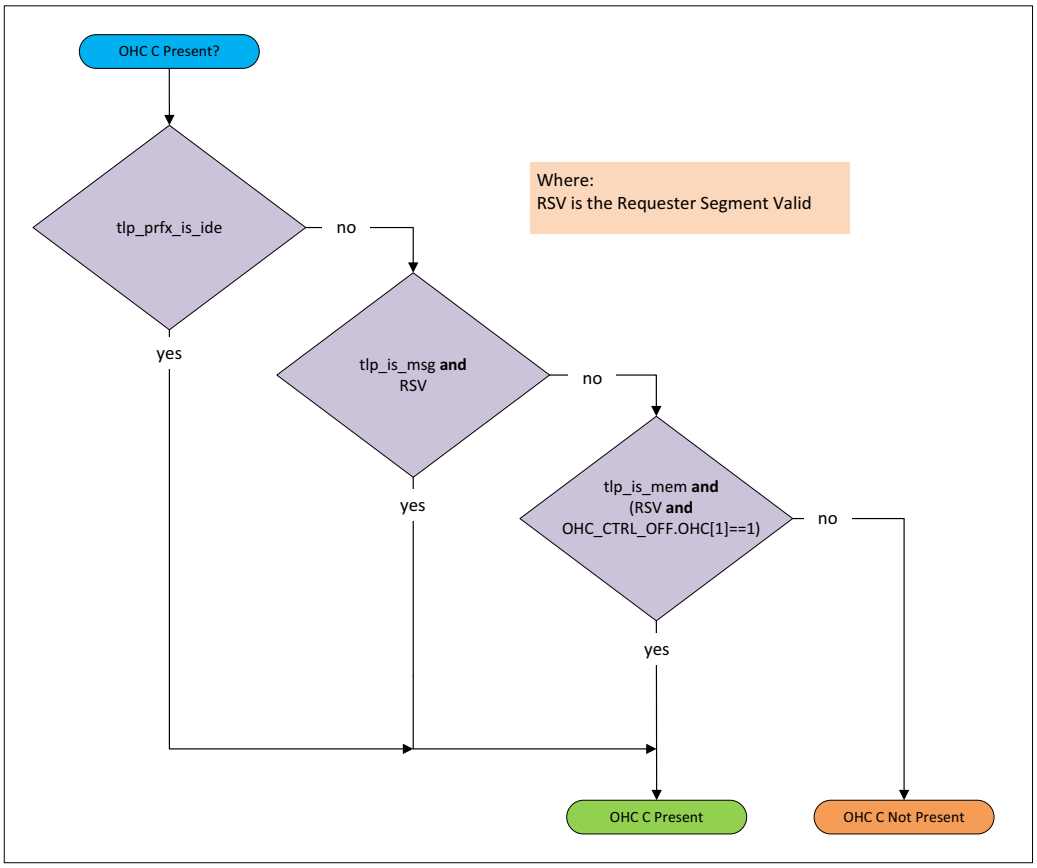
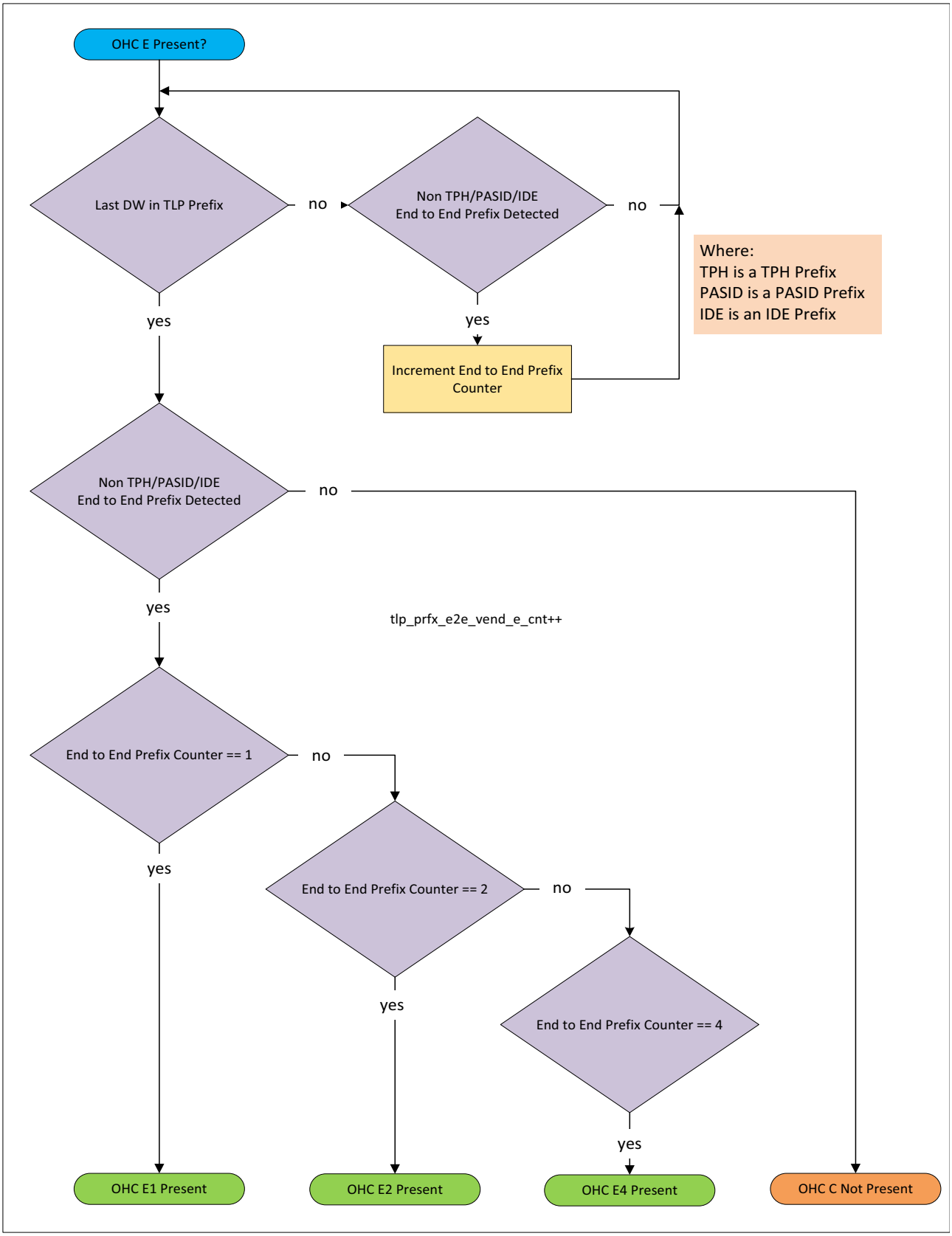


Figure 3-57 Insertion Process for OHC-E



3.6 Register Module, LBC, and DBI

This section discusses how the controller's register space is arranged and accessed. The following topics are covered in this section

- "Register Configuration Space Overview" on page 142
- "Local Bus Controller (LBC)" on page 145
- "Data Bus Interface (DBI) Access" on page 146
- "Access Limitations" on page 150
- "PHY Port Logic Registers" on page 152
- "BAR Details" on page 155
- "Generating Register Map" on page 160

3.6.1 Register Configuration Space Overview

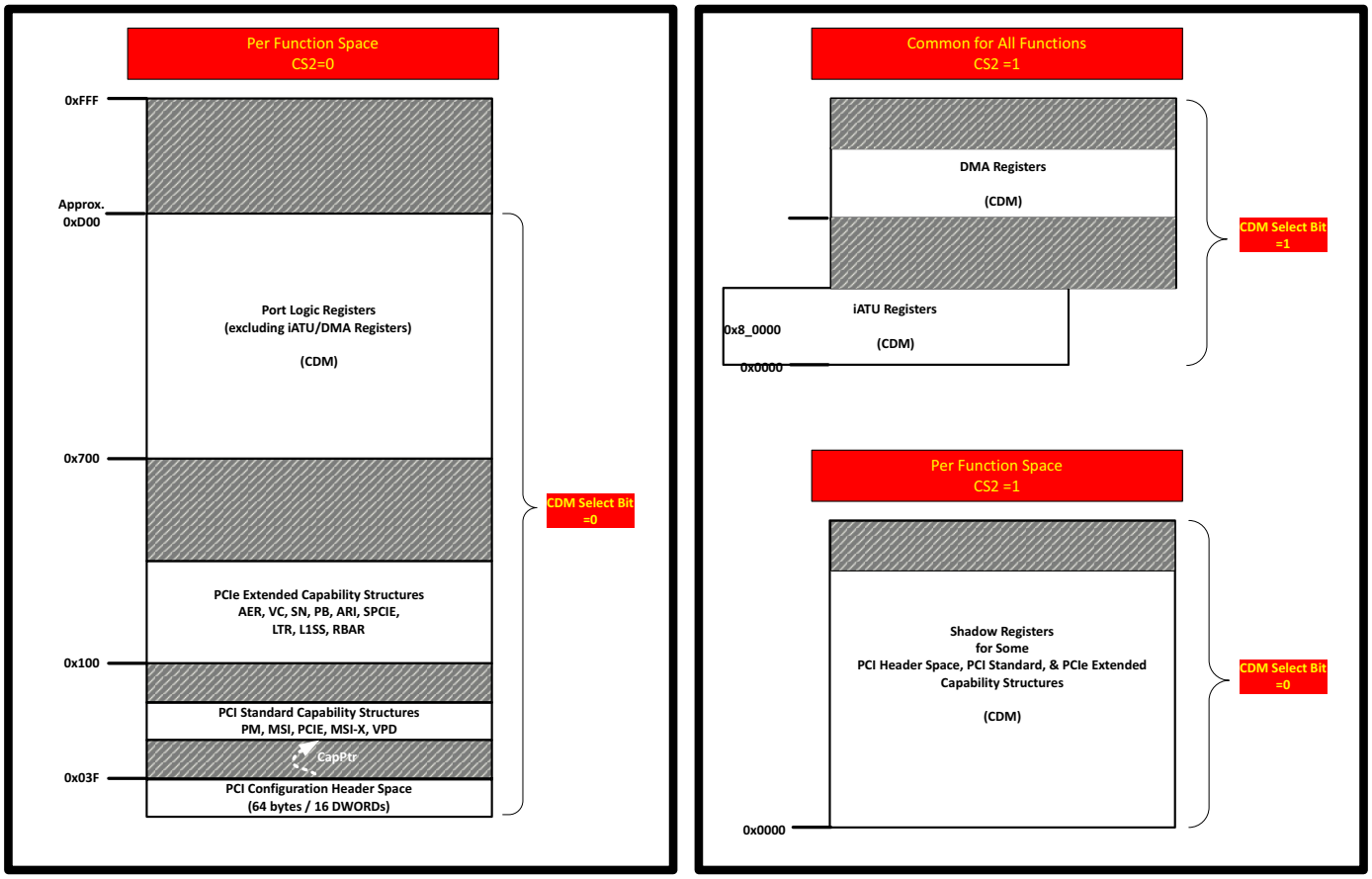
The controller has 4096 bytes of register space per function.

- Configuration Dependant Module (CDM) Registers

  Includes PCI Configuration Header, Standard and Extended Capabilities, and Port Logic registers.

  - PCI Configuration Header, Standard Capability, and Extended Capability Registers are PCIe controller configuration registers specified by the *PCI Express Base Specification, Revision 6.0, Version 1.0*.
  - Port Logic Registers (PL) registers are configuration registers which are not specified by the *PCI Express Base Specification, Revision 6.0, Version 1.0*, but which are Synopsys-specific. The port logic registers have specific pre-defined usages, and can optionally be removed from the controller hardware configuration.
    - There is one set of iATU port logic registers that is common for all functions.
    - There is one set of DMA port logic registers that is common for all functions.

Figure 3-58 Controller Configuration Space Layout



Capability configuration registers are in structures (groups) identified by a capability ID. The 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

  PCI compatible capability register groups begin at the configuration address stored in the capability pointer register at 0x34.
- PCI Express extended capability registers

  PCI Express extended capability register groups begin at address 0x100.

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 capability.

Attention

For releases previous to 4.80a, the DMA registers have been programmed through an indirect addressing scheme using an index register (DMA_VIEWPORT_OFF), to reduce the address footprint in the PCI Express extended configuration space. To use the legacy method of accessing DMA registers, contact Synopsys support through SolvNetPlus.

For releases previous to 4.80a, the iATU registers have been programmed through an indirect addressing scheme using an index register (iATU_VIEWPORT_OFF), to reduce the address footprint in the PCI Express extended configuration space. To use the legacy method of accessing iATU registers, contact Synopsys support through SolvNetPlus.

3.6.1.1 Register Types

There are three types of registers in the controller as shown in Table 3-20 on page 143.

Table 3-20 Types of Registers in PCIe Controller

Register Type	Description	Access/Addressing Method
Simple	Single register accessed directly by its address. Most of the registers are of this type.	Direct addressing method. You just supply the address.
Shadow	Another register exists at the same address as a Simple register. For example the BAR Mask registers have the same address as the BAR registers.	Normally can only be accessed through the DBI where you select between the two registers using the dbi_cs2 input. When you are using an AXI DBI subordinate, shadow registers can only be accessed through the DBI where you select between the two registers using the CS2 address bit. This is called DBI2, CS2, dbi_cs2, DBI_CS2, or Dbi2 access; all of these terms mean the same thing.
Viewport	Multiple (n) registers existing at the same address as a Simple register. For example Gen3 Coefficient Preset registers.	Indirect addressing method where you first write to a special index register to select which of the n registers at the address you want to access. You then proceed to write to the register address.

3.6.1.2 Register Views and Access

There are three views for most registers in the controller (see Table 3-21 on page 144), depending on how you access a register.

Table 3-21 Types of Register Views in PCIe Controller

View Type		Description	Typical Access Attributes (for example - R/W, read-only)
DBI	Dbi	Local (back-door) access to: ■ simple registers by your application logic through the DBI interface with: ■ dbi_cs =1, and ■ dbi_cs2 (or CS2) =0	■ Normally as per *PCI-SIG Specification*. ■ Some read-only[a] registers are permanently R/W. ■ Some read-only[a] registers can be made temporarily R/W when you write 1 to the `DBI_RO_WR_EN` bit of the `MISC_CONTROL_1_OFF` register.
	Dbi2	Local (back-door) access to some: ■ read only registers ■ shadow registers ■ iATU registers ■ DMA registers by your application logic through the DBI interface with: ■ dbi_cs =1, and ■ dbi_cs2 (or CS2) =1	■ Normally R/W ■ One exception is the BAR mask registers which are W (write-only).

a. As specified by the PCI-SIG specification.

In DSP mode, the controller register space is fully accessible from the DBI without any restrictions. There is no wire access in DSP Mode.

3.6.1.3 Writing to Read-Only Registers

Your application can write to some read-only (RO) and HwInit registers over the DBI when you set the "Write to RO Registers Using DBI" (`DBI_RO_WR_EN`) field of the `MISC_CONTROL_1_OFF` register to '1'. You must drive `app_dbi_ro_wr_disable` input pin to '0' to allow writes to `DBI_RO_WR_EN` field. Setting the `app_dbi_ro_wr_disable` input pin to '1' forces `DBI_RO_WR_EN` to '0' and disables writing to the `DBI_RO_WR_EN` field. The input pin `app_dbi_ro_wr_disable` exists only if `CX_PL_REG_DISABLE =0`. `DBI_RO_WR_EN` defaults to `CX_DBI_RO_WR_EN` parameter value at reset.

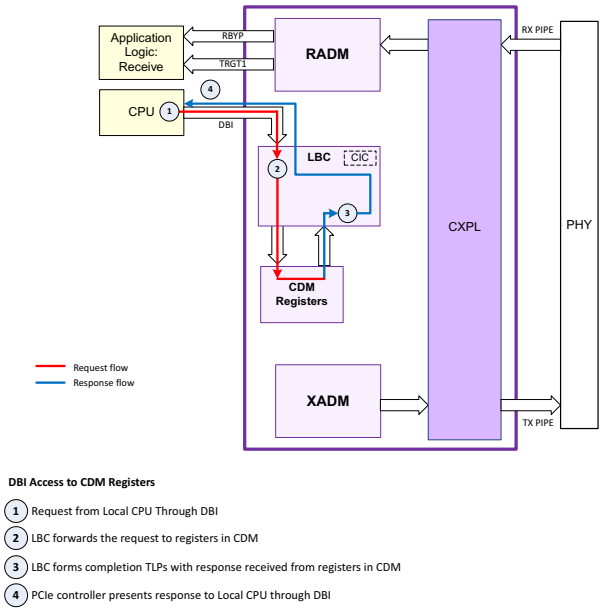
You should set this to 0. You can only program the PCIe configuration space registers (in the CDM) using the DBI. There is no wire access. In this case, when programming the Type-1 headers and capabilities, your host software can use the standard model of the configuration space as in the PCI-SIG specification. Therefore register fields marked RO do not become writable from the DBI. The exception to this is the SPCIE capability in a Gen3 configured device. The same applies to HwInit. RO registers never become writable over the wire, regardless of the setting of this register bit.

Your application can write to some shadow registers using DBI CS2 accesses. For more information on DBI CS2, see "Types of Register Views in PCIe Controller" on page 144.

3.6.2 Local Bus Controller (LBC)

The Local Bus Controller (LBC) module provides a mechanism for a local CPU (through the DBI) to access internal registers (in the CDM)

Figure 3-59 LBC Context (DSP)



3.6.3 Data Bus Interface (DBI) Access

This section discusses the Data Bus Interface (DBI). The topics are:

- "Overview" on page 146
- "Native Controller DBI Access Details" on page 147
- "AXI DBI Access Details" on page 148
- "Programming Examples" on page 149

3.6.3.1 Overview

The DBI can access all 4096 bytes (1024 DWORDs) of the PCI Express configuration space per function. DBI can also access iATU, DMA, MSI-X Table, and MSI-X PBA configuration space. This address space is fully accessible from the DBI without any restrictions.

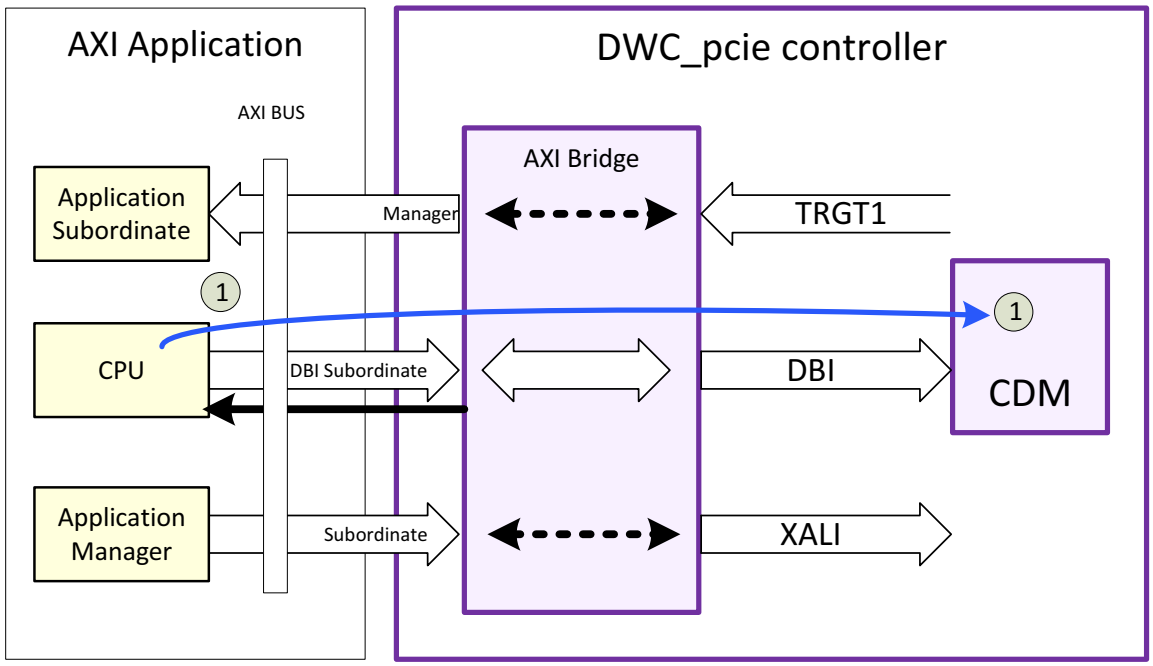
Native Controller DBI Access

Bit 0 of the DBI address input (`dbi_addr`) and chip select signal (`dbi_cs2`) determines the target of a DBI transaction as follows:

AXI Dedicated DBI Subordinate Access

When you are using the AXI dedicated DBI slave, then the DBI interface is not directly available to your application but is accessed through an AXI dedicated DBI slave interface. To enable DBI access through a AXI dedicated DBI slave interface, you must set the `DBI_4SLAVE_POPULATED` configuration parameter. You select the target of the DBI transaction using a bit of the AXI dedicated DBI slave address signal, whose position moves around. For more information, see "AXI DBI Access Details" on page 148.

Figure 3-60 AXI DBI Access with a Dedicated DBI Slave Interface (SHARED_DBI_ENABLED =0, DBI_4SLAVE_POPULATED =1)

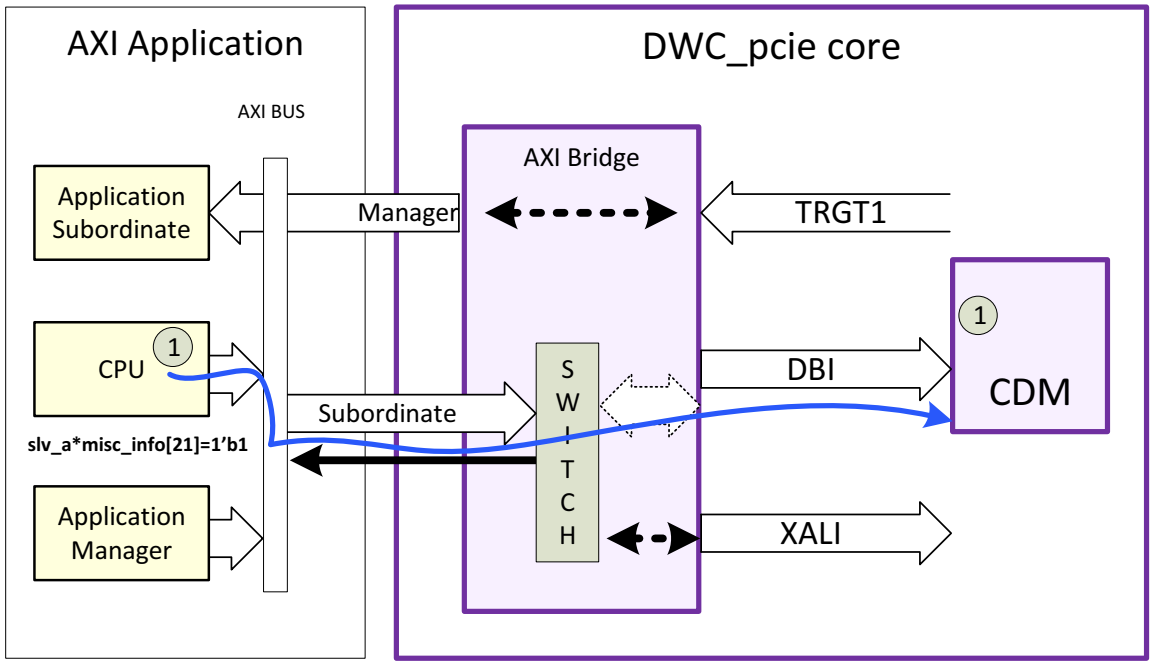


AXI Shared DBI Slave Access

To enable sharing of the existing slave interface (AXI only), you must enable the `SHARED_DBI_ENABLED` configuration parameter. You select the target of the DBI transaction using a bit of the AXI subordinate address signal whose position moves around. For more information, see "AXI DBI Access Details" on page 148.

When `slv_armisc_info[21]/slv_awmisc_info[21] =1`, the shared subordinate interface routes transactions to the CDM, instead of to the outbound PCIe transmit path at XALI1. Your application AXI manager could possibly decode an address range, or bits of the subordinate ID, to drive this bit.

Figure 3-61 AXI DBI Access with a Shared DBI Slave Interface (SHARED_DBI_ENABLED =1, DBI_4SLAVE_POPULATED =0)



Note

- You must not enable both the parameters SHARED_DBI_ENABLED and DBI_4SLAVE_POPULATED at the same time.
- When you are using the AXI bridge module, you can alternatively access the DBI through the native DBI interface (which is not connected to your AXI bus). To access the DBI through the native DBI interface, you must set SHARED_DBI_ENABLED =0 and DBI_4SLAVE_POPULATED =0.

3.6.3.2 Native Controller DBI Access Details

- The DBI address must be a DWORD address with `dbi_addr[1]=1b0`.
- Byte access within a DWORD for write requests is possible using `dbi_wr[3:0]`.
- The `dbi_wr[3:0]` bus also identifies the type of access as a write or read (4h0).

Table 3-22 Native Controller DBI Address Bus Layout for Accessing CDM (dbi_cs2 =0)

31-19	18-16	15-12	11-2	1	0
0	0x0	0	1 K-DWORD Register Address	0	0
The lower 12 bits are used to access the 4KB (1K DWORDs) of the PCI Express configuration space.					

Table 3-23 Native Controller DBI Address Bus Layout for Accessing iATU Configuration Registers(dbi_cs2 =1)

Access Type	31-20	19	18-17	16 -9	8	7-2	1	0
iATU	0		Reserved	Region Number Select	0: Outbound Region	Register Address	0	1
	0		Reserved	Region Number Select	1:Inbound Region	Register Address	0	1

Table 3-24 Native Controller DBI Address Bus Layout for Accessing DMA Configuration Registers(dbi_cs2 =1)

Access Type	31-20	19	18-17	16 -9	8	7-2	1	0
DMA	0	1	Reserved	DMA Address			0	1

3.6.3.3 AXI DBI Access Details

- The DBI address must be a DWORD address (`*addr[1:0] =2b00`) with the CDM selection bit of the address set to 0. The selection bit position is [13].
- In the AXI Shared DBI slave mode, you must set `slv_armisc_info[21]` or `slv_awmisc_info[21]` =1 to indicate that the request is for the DBI, and that is not a normal PCIe outbound request.
- When the number of Physical Functions (PFs) is greater than 32, you must program the `DBI_FUNCTION_BANK_CTRL_REG_OFF` register to select the appropriate function range for accessing ELBI mapped function configuration space through the AXI DBI interface.

Note The MSI-X Table and MSI-X PBA access through DBI interface is only supported for ELBI Data Bus Width of 32 bits (CX_LBC_NW =1)

Table 3-25 AXI DBI Address Bus Layout

Access Type	31-22	21	20	19	18-14	13	12	11-2	1	0
CDM	0							1 K-DWORD Register Address	0	0
iATU	0	1	1	0	iATU Address				0	0
DMA	0	1	1	1	DMA Address				0	0

Note You must set the AXI DBI Address Bus Layout bits that are not used by the PCIe controller (marked as ‘0’ in AXI DBI Address Bus Layout) to ‘0’. The PCIe controller does not report any error if you do not set these bits, or set these bits to any value other than ‘0’.

3.6.3.4 Programming Examples

The following programming examples are discussed:

- “Access From DBI (Native) (DSP)” on page 149
- “Access From DBI (AXI Dedicated DBI/Shared DBI) (DSP)” on page 149

3.6.3.4.1 Access From DBI (Native) (DSP)

You can access the following spaces without any restrictions:

- Port logic registers in the CDM by setting:
  - `dbi_addr[0] =0 and dbi_cs2=0`
- PCI configuration registers by setting:
  - `dbi_addr[0] =0 and dbi_cs2=0`

3.6.3.4.2 Access From DBI (AXI Dedicated DBI/Shared DBI) (DSP)

You can access the following spaces without any restrictions:

- Port logic registers in the CDM by setting:
  - `dbi_*addr[13]/[20]`[1] `=0`
- PCI configuration registers by setting:
  - `dbi_*addr[13]/[20] =0`

For more information, see “Native Controller DBI Access Details” on page 147.

1. Bit position is based on CX_NFUNC and not on device_type, so a DM controller with CX_NFUNC >1 or CC_DMA_ENABLE =1 or CX_INTERNAL_ATU_ENABLE =1 uses bit 20 (not bit 13) in RP mode.

3.6.4 Access Limitations

The following limitations exist for the DBI.

- CDM Access Restrictions:
  - Maximum payload Length =1 DWORD.
- DBI Data/Address Port Widths (without AXI bridge):
  - Data: 32-bit
  - Address: 32-bit

Additional AXI DBI Limitations

The following limitations exist when you configure the controller to use the AXI bridge module.

- The supported DBI data and address port sizes are as follow:

Table 3-26 Supported AXI Dedicated DBI Slave Port Widths

Data	Address
32	32

Table 3-27 Supported AXI Shared DBI Slave Port Widths

Data	Address
32, 64, 128, 256	32, 64[a]

a. When CX_SRIOV_ENABLE =1 or CX_ARI_ENABLE =1 the AXI Shared-DBI Slave Port Address width must be 64-bits.

- The DBI address must be a DWORD address.
- AXI transfer widths of 8, 16, and 32 bits are supported.The AXI dedicated DBI/shared DBI interface returns an error response to requests with width greater than 32 bits. The type of error response is programmable through `AMBA_ERROR_RESPONSE_DEFAULT_OFF` register[1].
- The CDM only supports 32-bit data access. The AXI dedicated DBI/shared DBI interface returns an error response to multi beat requests. The type of error response is programmable through `AMBA_ERROR_RESPONSE_DEFAULT_OFF` register[1].
- Byte access within a DWORD for DBI WRITE requests is possible for AXI.
  - Dedicated DBI: The write strobes `dbi_wstrb[3:0]` are sent to the internal (native controller) DBI interface.
  - Shared DBI: The relevant four strobes from `slv_wstrb` are sent to the internal (native controller) DBI interface. For example on a 64-bit data bus:
    - `slv_waddr` =0x0000 and `slv_wstrb[7:0]` =0x03 results in 0x3 at the internal (native controller) DBI interface.

1. The type of error response returned by the AXI dedicated/shared DBI slave is the same as that programmed for UR completions in the AMBA_ERROR_RESPONSE_DEFAULT_OFF register.

- `slv_waddr` =0x0004 and `slv_wstrb[7:0]` =0x30 results in 0x3 at the internal (native controller) DBI interface.

A zero-byte write over DBI is processed as a read and should never be issued. The AXI dedicated/shared DBI slave drops a zero-byte write before it reaches the internal (native controller) DBI and returns an error response. The type of error response is programmable through `AMBA_ERROR_RESPONSE_DEFAULT_OFF` register[1].

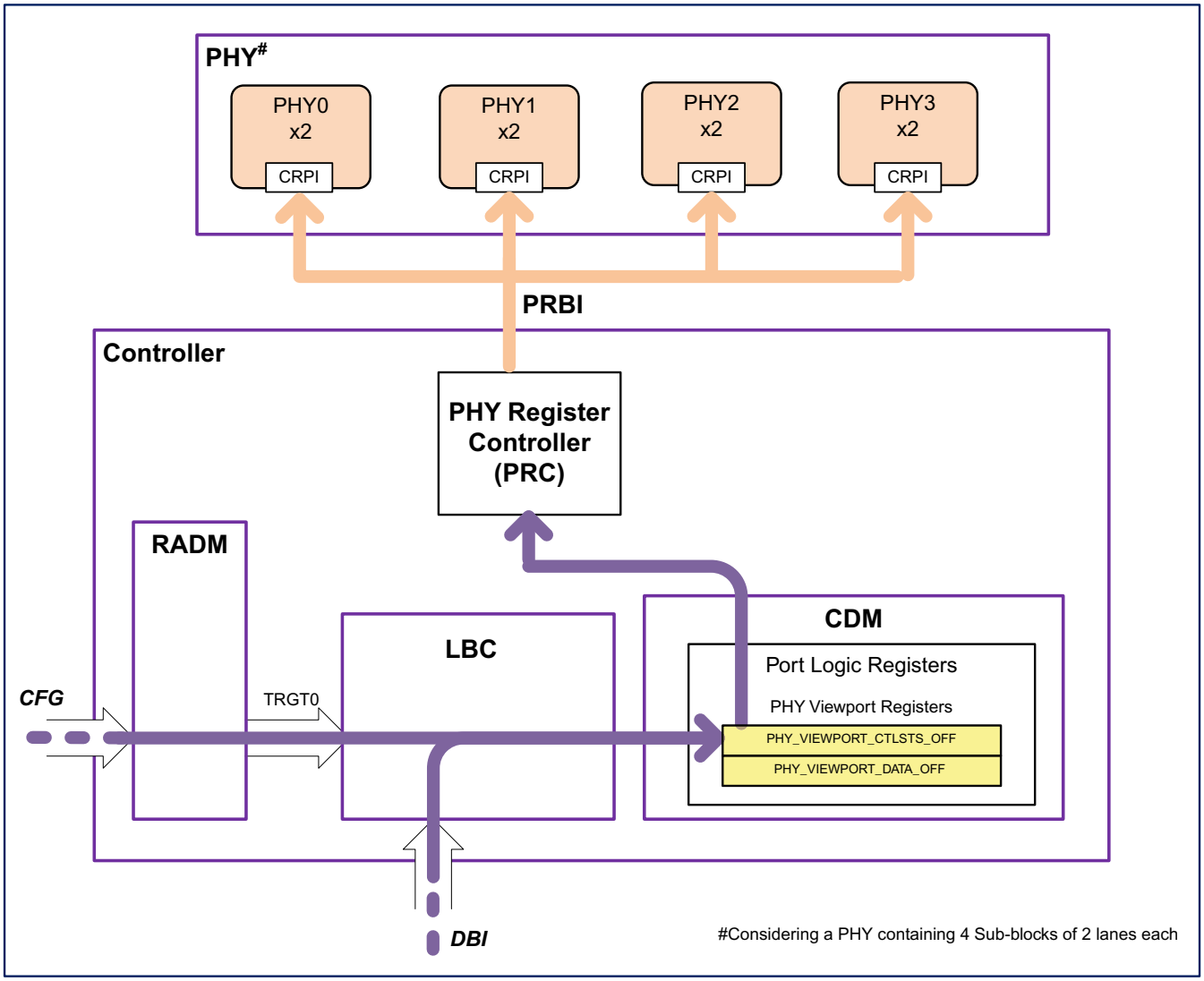
Hint

You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the "Integrating the Controller and VIP using VTB" section in the "Integration" chapter of the *User Guide*.

3.6.5 PHY Port Logic Registers

The controller implements `PHY_VIEWPORT_CTLSTS_OFF` and `PHY_VIEWPORT_DATA_OFF` registers which control the PHY Register Bus Interface (PRBI) to perform read and write operations on the PHY sub-blocks. These registers are enabled when `CX_PHY_VIEWPORT_ENABLE` parameter is set.

Figure 3-62 PHY Viewport Registers



Control registers of all the PHY sub-blocks map to a single `PHY_VIEWPORT_CTLSTS_OFF` register. To perform a read/write operation on the PHY, an indirect addressing mechanism is used, where first the `PHY_VIEWPORT_CTLSTS_OFF` register is programmed through DBI (`dbi_addr[0] =0` and `dbi_cs =1`) or CFG to select the PHY block/lane to read from/write to, and then actual read/write on the PHY sub-block happens.

Programming the PHY Viewport Registers to Read from PHY Registers

- Set PHY_VIEWPORT_ADDR and PHY_VIEWPORT_NUM fields of PHY_VIEWPORT_CTLSTS_OFF register.
- Set PHY_VIEWPORT_READ field of PHY_VIEWPORT_CTLSTS_OFF register to 1. When set, phy_cr_para_rd_en is asserted and read operation is initiated on the PRBI.

  Note: For DBI access, lbc_dbi_ack signal is asserted before the operation on the PRBI is completed. For CFG access, completion is returned before the operation on the PRBI is completed.
- When PHY_VIEWPORT_PENDING field of PHY_VIEWPORT_CTLSTS_OFF register becomes 0, check PHY_VIEWPORT_STATUS field of PHY_VIEWPORT_CTLSTS_OFF register to confirm that the read operation on PRBI has completed successfully.
- Read PHY_VIEWPORT_DATA_OFF register to fetch the data read from the PHY registers.

Programming the PHY Viewport Registers to Write to the PHY Registers

- Set PHY_VIEWPORT_ADDR, PHY_VIEWPORT_NUM, and PHY_VIEWPORT_BCWR fields of PHY_VIEWPORT_CTLSTS_OFF register.
- Set PHY_VIEWPORT_DATA_OFF register with the data to be written on the PHY. The signal phy_cr_para_wr_en is asserted to initiate a write operation on PRBI once the data is written to PHY_VIEWPORT_DATA_OFF register.

  Note: For DBI access, lbc_dbi_ack signal is asserted before the operation on the PRBI is completed. For CFG access, completion is returned before the operation on the PRBI is completed.
- When PHY_VIEWPORT_PENDING field of PHY_VIEWPORT_CTLSTS_OFF register becomes 0, check PHY_VIEWPORT_STATUS to confirm that the write operation on PRBI has completed successfully.

Write Operation: Role of PHY_VIEWPORT_ADDR, PHY_VIEWPORT_NUM, and PHY_VIEWPORT_BCWR

Table 3-28 shows the setting of PHY_VIEWPORT_BWCR, PHY_VIEWPORT_NUM and PHY_VIEWPORT_ADDR fields of the PHY_VIEWPORT_CTLSTS_OFF register, for a few combinations of PHY sub-block and lane writes. These settings are with reference to Consumer 8G PHY for TSMC 28 HPC.

Table 3-28 Write Operation: PHY_VIEWPORT_CTLSTS_OFF Field Values

Access	PHY_VIEWPORT_BWCR	PHY_VIEWPORT_NUM	PHY_VIEWPORT_ADDR			
			15	14-12	11-8	7-0
Write: Selected Sub-block, SelectedLane						
PHY*0*.lane**0**	0	0	0	Address	**0**	Address
PHY*0*.lane**1**	0	0	0	Address		Address
Write: Selected Sub-block, AllLanes						
PHY*0*.lane**0**/PHY*0*.lane**1**	0	0	**1**[a]	Address	NA	Address
PHY*1*.lane**0**/PHY*1*.lane**1**	0	1	**1**[a]	Address	NA	Address
Write: All Sub-blocks, SelectedLane						

Access	PHY_VIEWPORT_BWCR	PHY_VIEWPORT_NUM	PHY_VIEWPORT_ADDR			
			15	14-12	11-8	7-0
PHY*0*.lane**0** PHY*1*.lane**0** PHY*2*.lane**0** PHY*3*.lane**0**	*1*	NA[b]	0	Address	**0**	Address
PHY*0*.lane**1** PHY*1*.lane**1** PHY*2*.lane**1** PHY*3*.lane**1**	*1*	NA[b]	0	Address	**1**	Address

a. if PHY_VIEWPORT_ADDR[15] =1, all lanes inside sub-block selected by PHY_VIEWPORT_NUM are written.
b. if PHY_VIWEPORT_BWCR =1, PHY_VIWEPORT_NUM is not considered.

Access	PHY_VIEWPORT_BWCR	PHY_VIEWPORT_NUM	PHY_VIEWPORT_ADDR			
			15	14-12	11-8	7-0
Broadcast Write: All Sub-blocks,All Lanes						
PHY*0*.lane**0**/PHY*0*.lane**1** PHY*1*.lane**0**/PHY*1*.lane**1** PHY*2*.lane**0**/PHY*2*.lane**1** PHY*3*.lane**0**/PHY*3*.lane**1**	1	NA[a]	**1**	Address	NA	Address
PHY*0*.lane**0** PHY*1*.lane**0** PHY*2*.lane**0** PHY*3*.lane**0**	1	NA[a]	0	Address	**0**	Address

a. if PHY_VIWEPORT_BWCR =1, PHY_VIWEPORT_NUM is not considered.

For more information on cold reset sequence with PHY initialization, see . Figure 2-26 on page 93For more information on PHY initialization after cold reset, see Figure 2-27 on page 94.

3.6.6 BAR Details

In this section, the operation and programming of Base Address Registers (BARs) is described.

3.6.6.1 RP Mode

The following topics are discussed:

- "Overview" on page 155
- "BAR Sizing" on page 156
- "Disabling a BAR" on page 157
- "General Rules for BAR Setup (Fixed Mask or Programmable Mask Schemes Only)" on page 157
- "Memory BAR Sizes" on page 158
- "IO BAR Sizes" on page 158
- "BAR Mask Registers" on page 158
- "Example BAR Setup" on page 158
- "Expansion ROM BAR Mask Register (Offset: 0x38; same as the Expansion ROM BAR, but requires dbi_cs2/CS2 for write access)" on page 159

3.6.6.1.1 Overview

Base Address Registers (Offset: 0x100x14)

Two BARs are present but are not expected to be used. You should disable them (see "Disabling a BAR" on page 157) to avoid unnecessary memory assignment during device enumeration. If you do use a BAR, then you should program it to capture TLPs that are targeted to your local non-application memory space residing on TRGT1[1], and not for the application on TRGT1[2]. The BAR range must be outside of the three Base/Limit regions. The controller provides one pair of 32-bit BARs (BAR0 and BAR1). The BARs can be configured as follows:

- One 64-bit BAR: BAR0 and BAR1 are combined to form a single 64-bit BAR.
- Two 32-bit BARs: BAR0 and BAR1 are two independent 32-bit BARs.
- One 32-bit BAR: BAR0 is a 32-bit BAR and BAR1 is either disabled or removed from controller altogether to reduce gate count.

Note For more information about routing requests to either TRGT1 or TRGT0 on a BAR-by-BAR basis, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691. For more information on BAR operation, see "Receive Routing" on page 135.

The following sections describe how to set up the BAR types and sizes by programming values into the base address registers.

1. Or AXI manager interface.
2. Because in a DSP, there is no wire access to TRGT0 (ELBI or CDM) using CFG requests or BAR-matched MEM requests.

3.6.6.1.2 BAR Sizing

The BAR masks are used for indicating the amount of address space that each BAR requests from host software. The BAR masks determine which bits in each BAR are non-writable by host software, which determines the size of the address space claimed by each BAR. The BAR mask values indicate the range of low-order bits 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. Your application can write to all BAR bits to allow setting of memory, I/O, and other standard BAR options. There are three schemes that each BAR can use for setting the amount of address space that it requests.

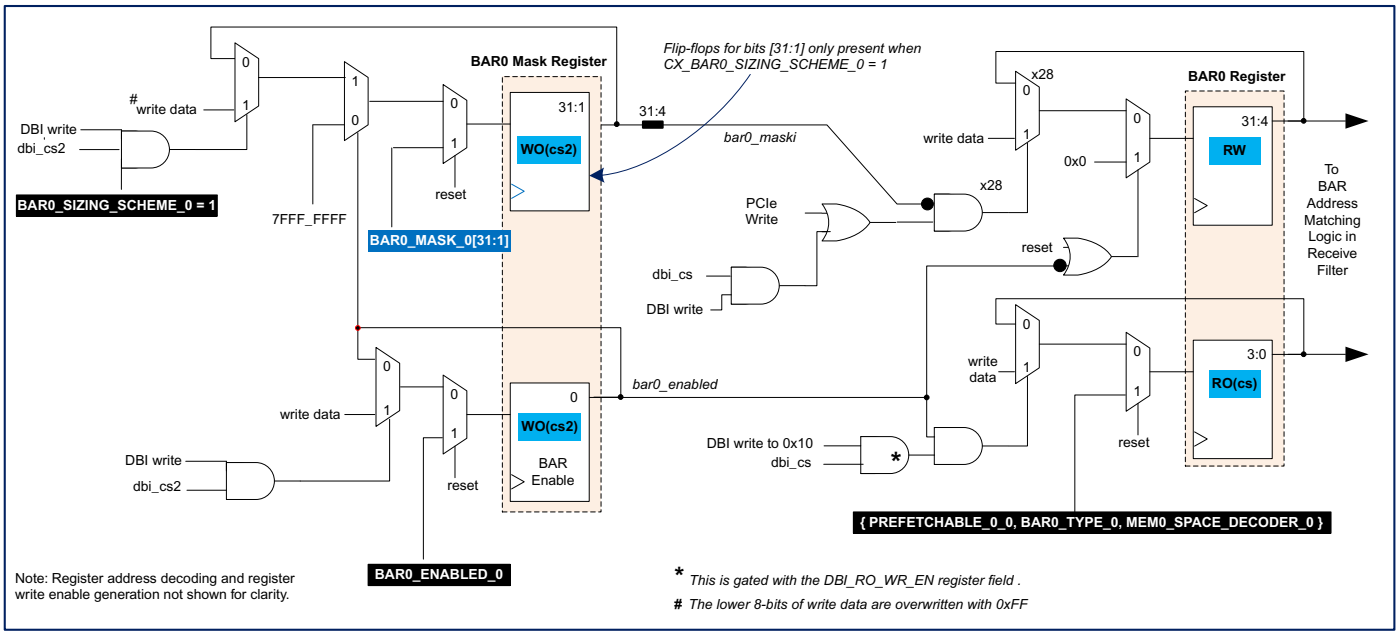
- **BARn_SIZING_SCHEME_0 =Fixed Mask (0)**

  You hardwire the mask using the `BARn_MASK_0` parameter. You cannot change it at runtime.

- **BARn_SIZING_SCHEME_0 =Programmable Mask (1)**

  Mask registers are instantiated in the controller. You set the default using `BARn_MASK_0`. You can change it at runtime through the DBI, at the same address as the corresponding BAR register, by asserting `dbi_cs2` (or the CS2 address bit when the AXI bridge is used). The BAR mask registers are only writable through the DBI, and not readable. They are not visible from the PCIe wire by the remote partner. When the BAR mask value for a BAR is less than that required for the BAR type, the controller automatically uses the minimum value for the BAR type.

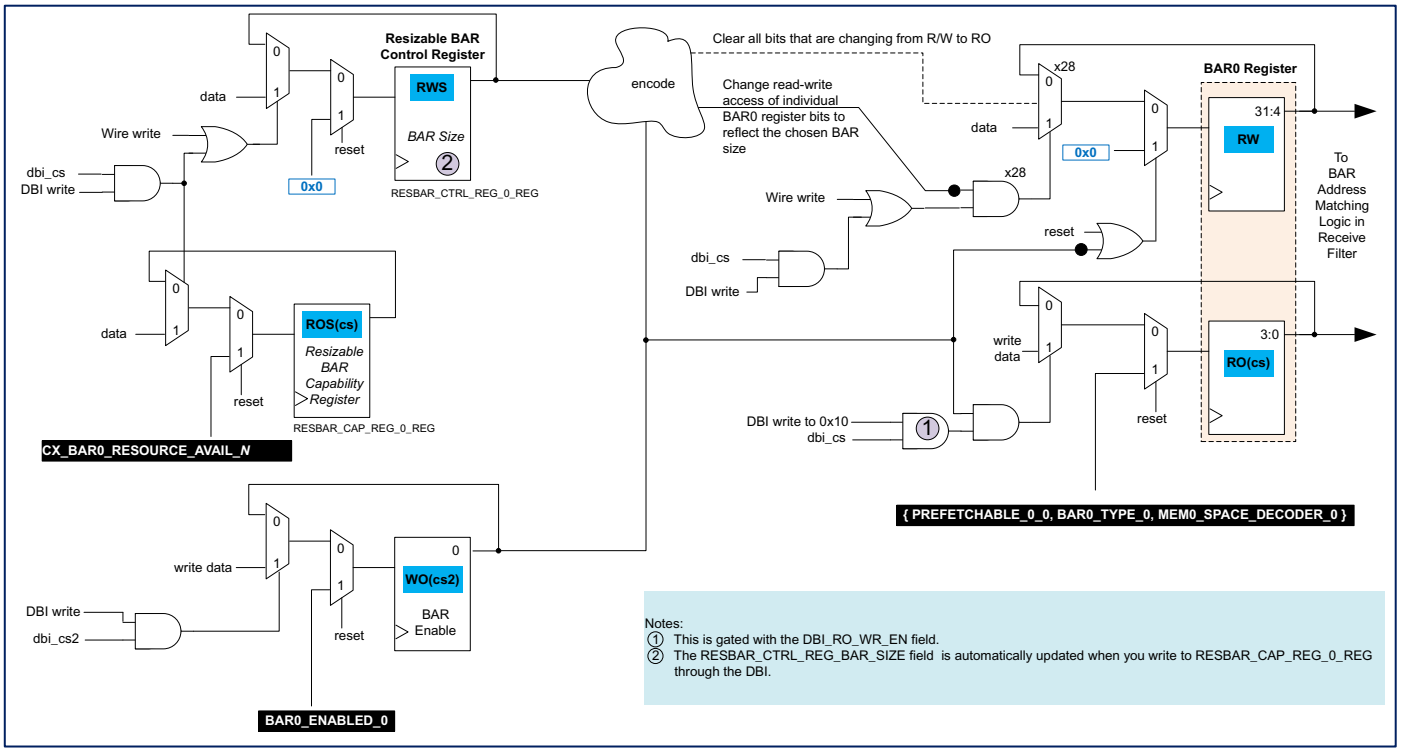
Figure 3-63 Fixed and Programmable Mask Example for 32-bit Memory BAR0 (CX_BAR0_SIZING_SCHEME_0 !=2)



- **BARn_SIZING_SCHEME_0 =Resizable BAR (2)**

  When you enable the PCI-SIG resizable BAR feature, the host software reads the Supported Resource Sizes field of the Resizable BAR Capability register.

Figure 3-64 Resizable BAR Example for 32-bit Memory BAR0 (CX_BAR0_SIZING_SCHEME_0 =2)



After it selects a BAR size, it writes this value to the BAR Size field of Resizable BAR Control register. From this field, the controller automatically derives the R/W mask for the BAR register. The host software can now start the normal BAR query process. You can change the BAR size at runtime, because the “Supported Resource Sizes” register field is writable through the local DBI (only).

3.6.6.1.3 Disabling a BAR

To disable a BAR (in any of the three schemes), your application can write 0 to the LSB of the BAR mask register. The BAR mask register is at the same address as the BAR register, and you can access it through the DBI (only) by asserting `dbi_cs2`(or the CS2 address bit when the AXI bridge is used).

3.6.6.1.4 General Rules for BAR Setup (Fixed Mask or Programmable Mask Schemes Only)

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:

- BAR0 and BAR1 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 controller hardware configuration to reduce gate count.
- An I/O BAR must be a 32-bit BAR and cannot be prefetchable.
- When BAR0 is configured as a 64-bit BAR:
  - BAR1 is the upper 32 bits of the combined 64-bit BAR formed by BAR0 and BAR1. Therefore, BAR1 must be disabled and cannot be configured independently.
  - BAR0 must be a memory BAR and can be either prefetchable or non-prefetchable.

  - The contents of the BAR0 mask register determine the number of writable bits in the 64-bit BAR, subject to the restrictions described in "BAR Mask Registers" on page 158. The BAR1 mask register contains the upper 32 bits of the BAR0 mask value.
  - BAR0 can be disabled by writing 0 to bit [0] of the BAR0 mask register.
- When BAR0 is configured as a 32-bit BAR:
  - You can configure BAR1 as an independent 32-bit BAR or remove BAR1 from the controller hardware configuration.
  - BAR0 can be configured as a memory BAR or an I/O BAR.
  - The contents of the BAR0 mask register determine the number of writable bits in the 32-bit BAR0, subject to the restrictions described in "BAR Mask Registers" on page 158.
  - BAR0 can be disabled by writing 0 to bit [0] of the BAR0 mask register.
- When BAR0 is configured as a 32-bit BAR, BAR1 is available as an independent 32-bit BAR according to the following rules:
  - BAR1 can be configured as a memory BAR or an I/O BAR.
  - The contents of the BAR1 mask register determine the number of writable bits in the 32-bit BAR1, subject to the restrictions described in "BAR Mask Registers" on page 158.
  - BAR1 can be disabled by writing 0 to bit [0] of the BAR1 mask register.
  - When BAR1 is not required in your design, you can remove BAR1 from the hardware configuration by setting both `BAR1_ENABLED_0` and `BAR1_MASK_TYPE_0` to 0.

3.6.6.1.5 Memory BAR Sizes

BAR bits [11:0] are always masked for a memory BAR. The *PCI Express Base Specification, Revision 6.0, Version 1.0* states that the minimum memory address range requested by a BAR is 128 bytes. In the *PCI Local Bus Specification,* it is recommended that devices that need less than 4 KB of address space should still consume 4 KB of address space in order to minimize the number of bits in the address decoder. The controller requires each memory BAR to claim at least 4 KB as it does not perform end address checking.

3.6.6.1.6 IO BAR Sizes

BAR bits [7:0] are always masked for an I/O BAR. The controller requires each I/O BAR to claim at least 256 bytes. The PCI Local Bus Specification allows I/O BARs to consume between 4 bytes and 256 bytes of address space. The controller only permits I/O BARs to consume 256 bytes of address space. This restriction is used in order to minimize the number of bits in the address decoder.

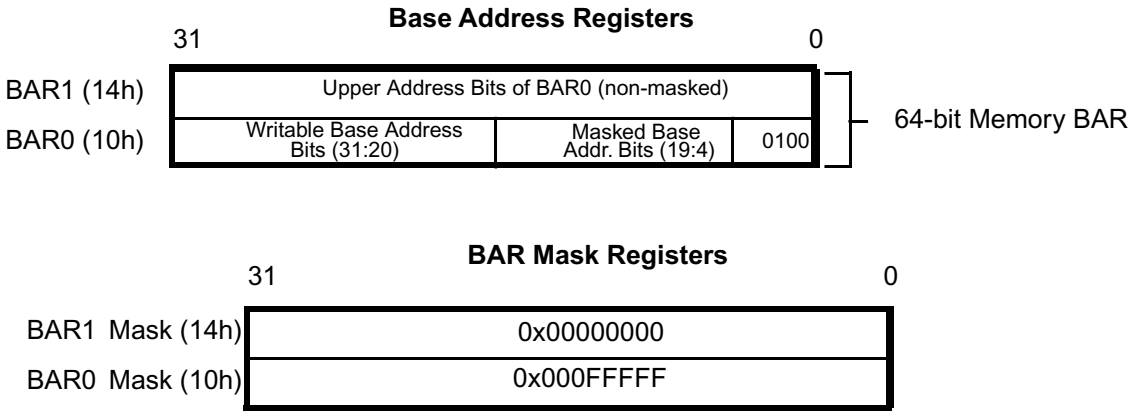
3.6.6.1.7 BAR Mask Registers

You can change the mask at runtime through the local DBI. The mask register is at the same address as the BAR register. The mask is a shadow register that is invisible to the PCIe wire but visible to the local processor over the DBI. Furthermore, it is only visible for a write not for a read. So you cannot read the mask register but you can write to it. The shadow register is accessed by asserting `dbi_cs2` and `dbi_cs`. If you assert dbi_cs (only) then you access the BAR which is the primary register at that location.

3.6.6.1.8 Example BAR Setup

Figure 3-65 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 3-65 Example Base Address Register Configuration



3.6.6.1.9 Expansion ROM BAR Mask Register (Offset: 0x38; same as the Expansion ROM BAR, but requires dbi_cs2/CS2 for write access)

Your local CPU can change the mask at runtime using the DBI. The mask register is at the same address as the BAR register. The mask is a shadow register that is invisible to the PCIe wire but visible to the local processor over the DBI. Furthermore, it is only visible for a write not for a read. So you cannot read the mask register but you can write to it. The shadow register is accessed by asserting CS2 and CS. If you assert CS (only) then you access the BAR which is the primary register at that location.

Base Address Registers (Offset: 0x100x14)

The Synopsys controller does not implement the optional BARs for the RP product. This is based on the assumption that the RP host probably has registers on some other internal bus and has knowledge and setup access to these registers already.

3.6.7 Generating Register Map

The detailed descriptions for each register are given in `DWC_pcie_ctl_rc_reference.pdf`

Attention

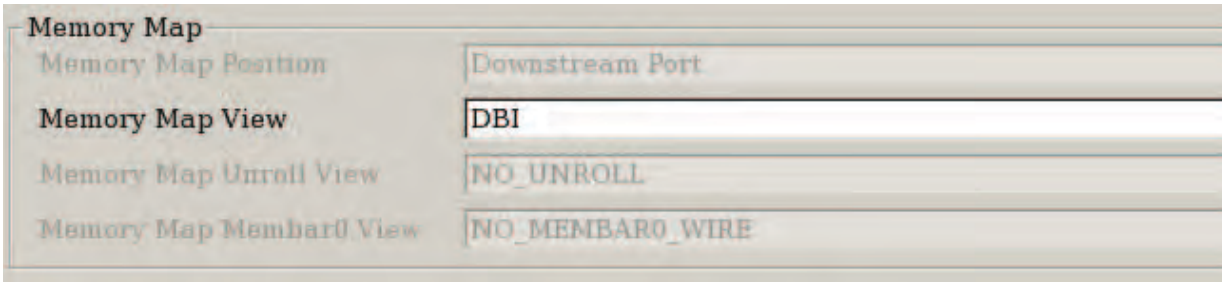
Generating Register Map Documentation

When you configure the controller in coreConsultant, you can access the register descriptions for your actual configuration at *workspace*/report/ComponentRegisters.html using the process described in the “Creating Optional Views and Reports” section in the *User Guide*. This report comes from the exact same source as the Databook but removes all the registers that are not in your actual configuration.

Table 3-29 Parameters to Consider for Generating DocBook XML and HTML Register Reports

Parameter Name	Values	Notes
CX_MEMORY_MAP_POSTION	0: Upstream Port 1: Downstream Port	The memory map of an Upstream port is different to that of a Downstream port. Using the CX_MEMORY_MAP_POSTION parameter you can generate the memory map either for the Upstream Port or the Downstream Port.
CX_MEMORY_MAP_VIEW	0: DBI 1: WIRE 2: DBI2	The registers for each port type, Upstream port/Downstream port can be accessed by the remote link partner through the wire or by your application through DBI (DBI or DBI2). For more information on the register views, see Table 3-21. The wire view of the registers is different to the DBI view. To choose the type of register view, use CX_MEMORY_MAP_VIEW parameter.
CX_UNROLL_VIEW	0: NO_UNROLL 1: UNROLL	You can generate the memory map view containing only ■ iATU registers ■ DMA registers
CX_MEMBAR0_VIEW	Note: Applicable for CXL configuration only.	

Figure 3-66 Memory Map Configuration Parameters in coreConsultant



Register Default Values

The register descriptions in this chapter indicate the default of the register after a cold reset, either as a specific numerical value or as the name of the design configuration parameter that sets the default. You can inspect the values of the configuration parameters in two different locations:

- In the Component Configuration report that is described in the Creating Optional Views and Reports section in the *User Guide*
- In `<workspace>/src/DWC_pcie_ctl_cc_constants.svh`

Rebuilding Capability Linked Lists (PFs only)

When your configuration software requires any set of standard PCI capability registers to be removed, it can overwrite the next capability pointers through the DBI and rebuild the linked list structure.

Note

- Even though there is a unique standard capabilities linked lists provided per function, specific capabilities cannot be included/excluded on a per-function basis. For example, MSI-X capability is included/excluded for all functions (PF) at the same time through the coreConsultant GUI.
- Even though there is a unique extended capabilities linked list provided per function, specific capabilities cannot be included/excluded on a per-function basis (in particular for LTR and VC, this is a non-fatal violation of a strict reading of the PCIe specification).
- Each function can have a different setup of a capability structure after it is included, although some features/settings are common across all functions.

3.7 Reliability, Availability, and Serviceability (RAS)

PCIe system Reliability, Availability, and Serviceability (RAS) ensures that failures in the underlying processes and hardware components do not cause any interruptions in the overall system operation. Synopsys PCIe RAS consists of four fundamental components:

- Reliable protocol architecture involving LCRC, ECRC, Flit FEC, Flit CRC, ACK/NAK handshake, and Replay mechanisms.
- Internal device-level Datapath and RAM protection (RAS DP) of data, header, and RAM control signals from application I/F to LCRC extraction/generation points.
- Debug, Error Injection, and Statistic gathering (RAS DES).
- Hot-plug usage to swap out components with uncorrectable errors.

This section discusses how the controller maintains the integrity of the link using different methods at different points. It also discusses error injection and traffic, error, and state logging. The following topics are covered.

- "RAS Wire Protection (ECRC)" on page 163
- "RAS Data Protection (RAS DP)" on page 164
- "RAS Debug, Error Injection, and Statistics (DES)" on page 177

3.7.1 RAS Wire Protection (ECRC)

The controller uses the optional ECRC checksum, as defined in the PCIe specification, to protect the wire part of the link. The controller inserts the ECRC in the transmit logic and checks it in the receive logic.

3.7.1.1 Generation

It is expected that the ECRC is not present in traffic that your application is presenting on the transmit interfaces of the controller. The controller generates and inserts the ECRC. When you are not using the AXI bridge, then you should set `client0_tlp_td` =0. To enable ECRC insertion:

- Set `CX_ECRC_ENABLE =AER_ENABLE` =1
- Set the `ECRC_GEN_EN` field of the `ADV_ERR_CAP_CTRL_OFF` register.

For native PCIe controller (without AMBA bridge), when you select the address alignment feature (`GLOB_ADDR_ALIGN_EN =1`), you must set `clientN_addr_align_en =0` for those TLPs that contain ECRC (TD bit =1). For more information, see "Address Alignment and ECRC" on page 363.

3.7.1.2 Checking

When the ECRC is present, the controller checks it and removes[1] it from the TLP. To enable ECRC checking:

- Set `CX_ECRC_ENABLE =AER_ENABLE =CX_ECRC_STRIP_ENABLE` =1
- Set the `ECRC_CHECK_EN` field of the `ADV_ERR_CAP_CTRL_OFF` AER register.

By default, when the controller detects a TLP, with an ECRC error, it performs the following:

- Discards[2] the TLP
- Generates a completion (for non-posted requests) with the completion status set to CA or UR
- Sets the status in the PCI-compatible status register
- Sets the status in the AER registers (when you enable AER)
  - When an ECRC error is detected by a PF, with ECRC checking enabled for any PF, ECRC error status is set for all PFs
  - ECRC error status is not set for VFs and reads zero

1. The controller does not remove the ECRC when CX_P2P_ENABLE =1. For more information, see the CX_ECRC_STRIP_ENABLE parameter.
2. You can program the controller to forward the TLP. For more information, see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.

3.7.2 RAS Data Protection (RAS DP)

The following topics are covered in this section:

- "Overview"
- "Features and Limitations" on page 168
- "RAS DP Registers" on page 171
- "Uncorrectable Error Handling" on page 174
- "Correctable Error Handling" on page 176

3.7.2.1 Overview

For the parts of the controller outside of ECRC/LCRC protection, you can use the RAS DP feature consisting of:

RAM Protection (ECC on data and Parity on address)

An ECC checksum is calculated and added to data that is written to the RAM. When the data is read from the RAM, then the controller recalculates the ECC and compares it against the value read from RAM. RAM ECC protection is provided for all of the RAMs in the controller, except for the RAS-DES RAM. ECC corrects single-bit errors and reports (without correction) multi-bit errors.

Datapath Protection (ECC or Parity)

- **Native PCIe Controller**

  For Tx traffic, your local application logic must add ECC or parity protection codes[1] to traffic using:

  - `client0_tlp_hdr_prot` and `client1_tlp_hdr_prot` for the TLP header.
  - Additional MSBs on `client0_tlp_data` and `client1_tlp_data` for the TLP data.
  - Additional MSBs on `client0_tlp_prfx` and `client1_tlp_prfx` for the TLP prefixes.

  For Rx traffic, your local application must check the protection codes on:

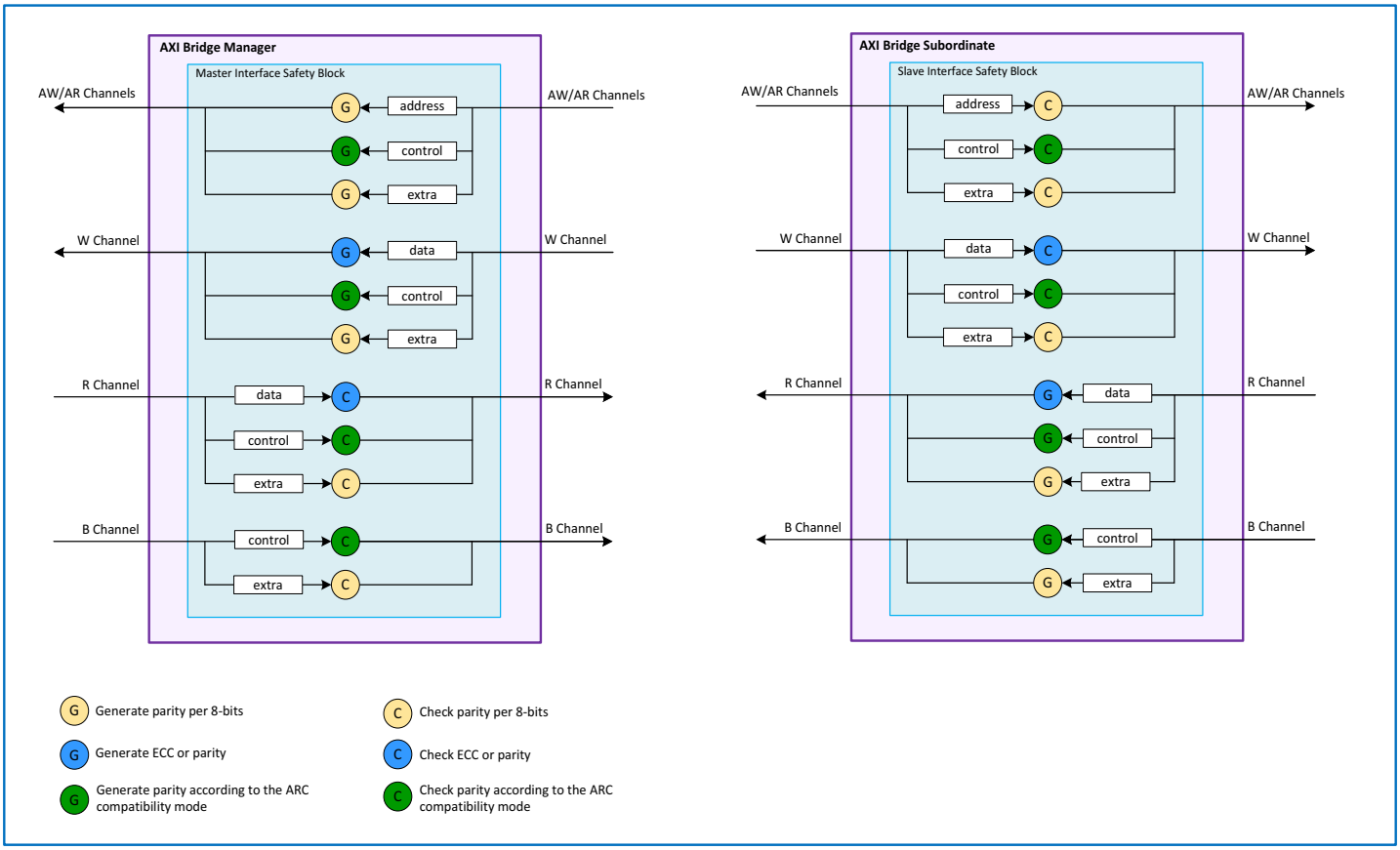
  - `radm_trgt1_hdr_prot` and `radm_bypass_hdr_prot` for the TLP header.
  - The MSBs on `radm_trgt1_tlp_data` and `radm_bypass_tlp_data` for the TLP data.
  - The MSBs on `radm_trgt1_tlp_prfx` and `radm_bypass_tlp_prfx` for the TLP prefixes.

- **AXI Interface (ECC or Parity)**

  You can use the `CX_AXI_IF_TYPE_OF_PROT` parameter to configure AXI interface protection. The AXI Interface Safety Block (ISB) in the AXI bridge master/subordinate generates and checks ECC or parity as shown in Figure 3-67.

1. For more information on how to calculate the protection codes, see the description of these signals in the "Signal Descriptions" chapter of the *Reference Manual*.

Figure 3-67 AXI Interface Safety Block (ISB)



The `[slv|mstr]_*user` signals of an AXI read/write channel are used to exchange the parity or ECC bits between the controller and your application.The controller aggregates `[slv|mstr]_*user` signal bits as extra, control, data or address protection bits. Table 3-30 describes the `[slv|mstr]_*user` signal content based on the `CX_AXI_IF_TYPE_OF_PROT` configuration parameter value. For more information on AXI interface signals, see the "Signal Descriptions" chapter of the *Reference Manual*.

Table 3-30 Relation Between the CX_AXI_IF_TYPE_OF_PROT Value and [slv|mstr]_*user Signal Content

CX_AXI_IF_TYPE_OF_PROT Value	[slv|mstr]_*user Signal Content
0[a] (Legacy Mode)	Includes the parity bits for data or address only.
1	Includes the parity bits for extra, control, data or address.
2	Includes the parity bits for extra and control; and the ECC bits for data or the parity bits for address.

a. *valid_p and *ready_p signals do not exist when CX_AXI_IF_TYPE_OF_PROT =0.

The control protection bits in the `[slv|mstr]_*user` signal are compatible with the DesignWare Cores AXI interface ISB ARC processor parity implementation. In case a control signal does not exist, the position of its parity bit in the `[slv|mstr]_*user` signal is taken by the parity bit of the subsequent control signal, as shown in Table 3-31.

Table 3-31 Example *_wuser Control Signal Parity Mapping for AXI3 and AXI4

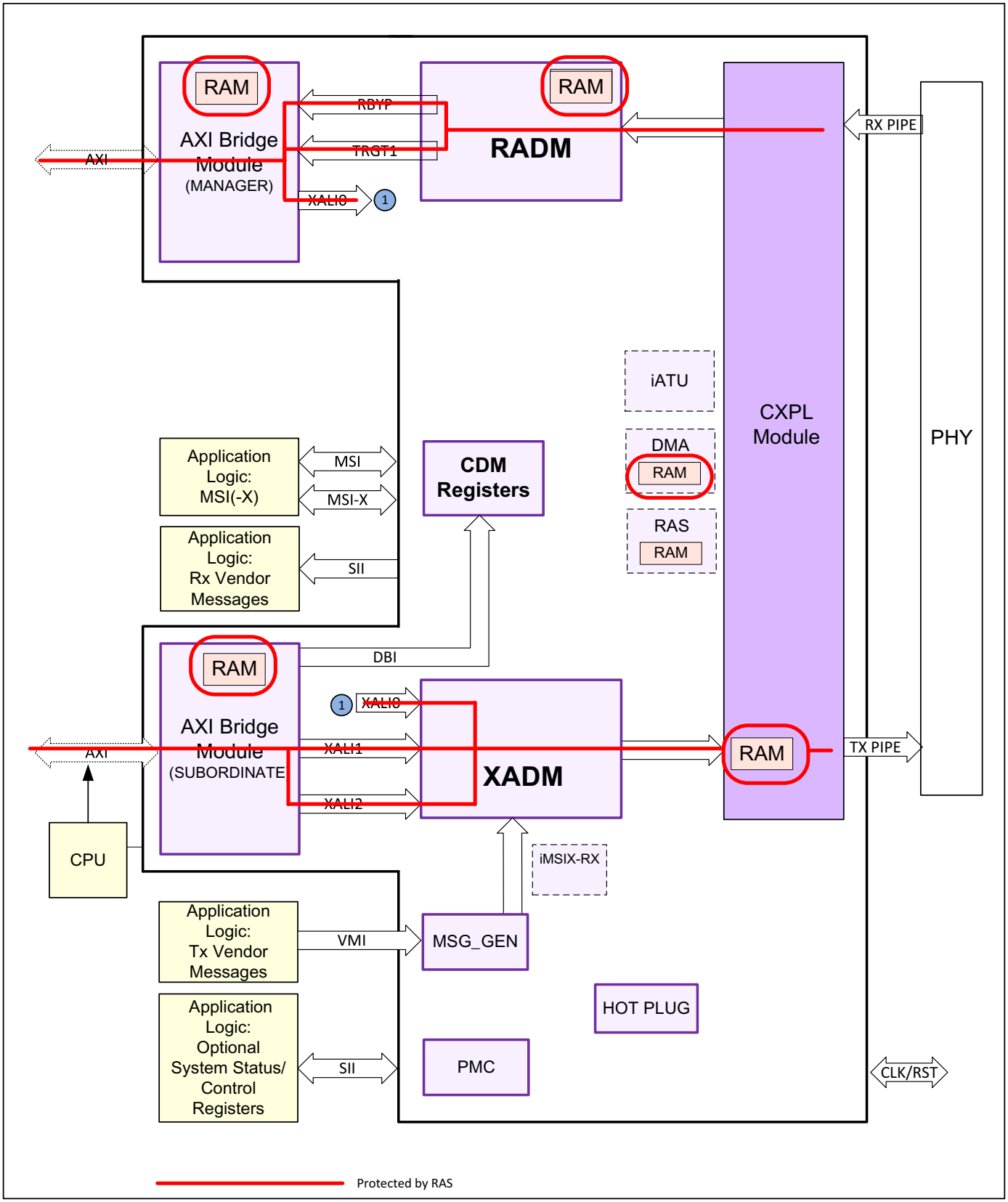
	Control Signals Taken Into Consideration	
***_wuser Control Signal Parity Bit**	**AXI3**	**AXI4**
*_wuser[CC_*_BUS_DATA_PROT_WIDTH]	*_wid	*_wstrb
*_wuser[CC_*_BUS_DATA_PROT_WIDTH+1]	*_wstrb	*_wlast
*_wuser[CC_*_BUS_DATA_PROT_WIDTH+2]	*_wlast	NA

Attention

You can find a reference RAS DP wiring template that has an example implementation of the application logic required to drive the controller's RAS DP inputs and the expected outputs in: `<workspace>/examples/pcie_rasdp_glue.sv`. This file is for reference purpose only, it must not be used as is.

The basic ECC module is based on http://www.synopsys.com/dw/ipdir.php?c=DW_ecc.

Figure 3-68 RAS DP Protection Coverage



3.7.2.2 Features and Limitations

Table 3-32 describes RAS data protection features and limitations.

Table 3-32 RAS Data Protection Features and Limitations

RAM Protection Features (CX_RASDP_RAM_PROT =1)	
RAM Data Protection	ECC[a]
RAM Types Protected	Internal/External
# Errors Detectable	1, 2
# Errors Correctable	1
1-bit errors are Correctable	Yes
2-bit errors are Uncorrectable	Yes
Programmable Correction Enable/Disable Per Region[b]	Yes
Corrected Data Rewritten to RAM	No
Controller Regenerates ECC Code at Output of RAM When Uncorrectable (2-bit) Error Detected	No
ECC Protection Range	
32-bit for 32-bit controller datapath width	Yes
64-bit all other controller datapath widths	Yes
ECC Code Size (8 bits ECC per 64 bits of RAM data)	
<= 26-bit RAM Data	6 bits
<= 57-bit RAM Data	7 bits
<= 64-bit RAM Data	8 bits
Optional Pipeline Stage	No
RAM Address Protection	Parity
Number of Bits per Address Bus	1
Address Parity Generated by Controller	Yes
Address Parity Checked by Controller[c]	No

a. See Limitations.
b. Using the RASDP_ERROR_PROT_CTRL_OFF register. Not available when CX_MULTISTREAM_ARCH =1 (correction always enabled)
c. The controller does not check the address parity. RAM address parity checking and error handling is application specific.

Datapath/Interface Protection Features (CX_RASDP >0)	
Datapath (Header and Data) Protection	Parity/ECC

Datapath/Interface Protection Features (CX_RASDP >0)	
Protection From Application Interfaces[a] to LCRC Insertion/Extraction	
Controller Expects Application to Supply Parity/ECC on TX Path Interfaces	Yes
Controller Strips Parity/ECC After LCRC Insertion on TX Path	Yes
Controller Inserts Parity/ECC Before LCRC Extraction on RX Path	Yes
Controller Passes Parity/ECC to Application on RX Path	Yes
Controller Checks and Recalculates Protection Codes at all Internal Processing (Data Manipulation) Steps[b]	Yes
# Errors Detectable	
ECC	1,2
Parity	1
# Errors Correctable	
ECC	1
Parity	0[c]
Programmable ECC Correction Enable/Disable Per Region[d]	Yes
Configurable ECC Correction On/Off Using CX_RASDP	Yes
ECC Protection Range	
32-bit for 32-bit controller datapath width	Yes
64-bit all other controller datapath widths	Yes
ECC Code Size (8 bits ECC per 64 bits[e]of datapath)	
32-bit Datapath	7 bits
64-bit Datapath	8 bits
128-bit Datapath	16 bits
256-bit Datapath	32 bits
512-bit Datapath	64 bits
Parity Protection Range	
8-bit	Yes
32-bit	No
64-bit	No
Selectable Odd/Even Parity	Yes

a. See Limitations.

b. Detailed report of all check points is available at http://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/RASDP_Check-Points.pdf.
c. The controller pulses app_parity_err[2:0]output bus. Your application can OR the bus bits and drive the Uncorrectable Error bit of the app_err_bus input. This is not required when RAS DP error mode is enabled because the controller handles the parity error as an uncorrectable error in this mode.
d. Using the RASDP_ERROR_PROT_CTRL_OFF register. Not available when CX_MULTISTREAM_ARCH =1 (correction always enabled).
e. Except for 32-bit datapath.

Statistics and Error Injection Features	
Error Injection (Per TLP)	
Programmable[a] 1-bit or 2-bit Injection	Yes
1-bit and 2-bit automatic error injection on the full datapath	Yes[b]
Programmable Continuous or Fixed-Number (n) Injection Modes	Yes[c]
Programmable Global Enable/Disable	Yes
Programmable Selectable Location Where Injection Occurs	Yes
Programmable simultaneous error injection on all the checkers (broad-cast mode)	Yes[d]
Error Statistics[e]	
Counts # Correctable Errors	Yes
Counts # Uncorrectable Errors	Yes
Reports Location of First and Last Errors (Correctable and Uncorrectable)	Yes
Reports RAM Addresses[f] of Detected Errors (Correctable and Uncorrectable)	Yes

a. Using the RASDP_ERROR_INJ_CTRL_OFF register.
b. Only when CX_MULTISTREAM_ARCH =1.
c. Only when CX_MULTISTREAM_ARCH =0.
d. Only when CX_MULTISTREAM_ARCH =1.
e. A single RAS DP error might increment more than one counter.
f. RASDP_RAM_ADDR_UNCORR_ERROR_OFF and RASDP_RAM_ADDR_CORR_ERROR_OFF registers when CX_MULTISTREAM_ARCH =0; RAS_DP_ERROR_TRACER_LOG_REPORT_2_OFF if CX_MULTISTREAM_ARCH =1.

Limitations (Native PCIe Controller)	
iMSIX-TX Module RAMs Protected	No
RAS DES RAM Protected	No
DBI Datapath (DBI Access to CDM) Protected	No
MSI/MSI-X Interfaces Protected	No
Vendor Message Interface (VMI) Protected	No
iMSIX-TX Module Datapath Protected	No

Limitations (AXI)	
AXI Configuration Support[a b]	

Limitations (AXI)	
ECC (RAM)[c]	Yes
ECC[d] (Datapath)	Yes
Parity (Datapath) on Payload Data	Yes
Parity (Datapath) on Header[e]	Yes
Error Injection	No

a. When RAS is enabled RTRGT1 address width (FLT_Q_ADDR_WIDTH) can be either 32 or 64, other values are not supported by the controller.
b. When an uncorrectable RAS.DP error is detected by the AXI logic, the following scenarios apply:-- If the payload data is corrupted but the bridge is functional, no reset is required to resume normal operation.- If the header is corrupted and bridge is in an inconsistent state a hot/cold reset is required to resume normal operation.
c. RAM protection cannot be enabled without datapath protection.
d. ECC protection is not supported when CX_AXI_IF_TYPE_OF_PROT =0.
e. The complete TLP payload data is protected, but only the address part of the header is fully protected inside the AXI bridge. The remainder of the header (carried on the *_misc_info_* signals) is not protected everywhere inside the AXI bridge and is routed without parity or ECC. It is however fully protected in the rest (native part) of the controller.

Limitations (DMA)	
DMA Configuration Support	
ECC (Read Buffer RAM)[a]	Yes
ECC (DMA Context RAM)	Yes
ECC (Datapath)	No
Parity (Datapath)	Yes
Error Injection	No

a. A single RAS DP error might increment more than one counter.

3.7.2.3 RAS DP Registers

The controller has a set of RAS DP registers as defined in Table 3-33. For detailed descriptions of how you use them, see the "Register Descriptions" chapter of the *Reference Manual*.

Table 3-33 RAS DP Vendor Specific Extended Capability (VSEC Registers) when CX_MULTISTREAM_ARCH =0

Register	Description
RASDP_EXT_CAP_HDR_OFF	Vendor-Specific Extended Capability Header.
RASDP_VENDOR_SPECIFIC_HDR_OFF	Vendor Specific Header.
RASDP_ERROR_PROT_CTRL_OFF	Error Protection Control.
RASDP_CORR_COUNTER_CTRL_OFF	Corrected error (1-bit ECC) counter selection and control.
RASDP_CORR_COUNT_REPORT_OFF	Corrected error (1-bit ECC) counter data.

Register	Description
RASDP_UNCORR_COUNTER_CTRL_OFF	Uncorrected error (2-bit ECC and parity) counter selection and control.
RASDP_UNCORR_COUNT_REPORT_OFF	Uncorrected error (2-bit ECC and parity) counter data.
RASDP_ERROR_INJ_CTRL_OFF	Error injection control.
RASDP_CORR_ERROR_LOCATION_OFF	Corrected errors locations.
RASDP_UNCORR_ERROR_LOCATION_OFF	Uncorrectable errors locations.
RASDP_ERROR_MODE_EN_OFF	RAS DP error mode enable.
RASDP_ERROR_MODE_CLEAR_OFF	RAS DP error mode clear.
RASDP_RAM_ADDR_CORR_ERROR_OFF	RAM Address where a corrected error (1-bit ECC) has been detected.
RASDP_RAM_ADDR_UNCORR_ERROR_OFF	RAM Address where an uncorrected error (2-bit ECC) has been detected.

Table 3-34 RAS DP Vendor Specific Extended Capability (VSEC Registers) when CX_MULTISTREAM_ARCH =1

Register	Description
RASDP_EXT_CAP_HDR_OFF	Vendor-Specific Extended Capability Header
RASDP_VENDOR_SPECIFIC_HDR_OFF	Vendor Specific Header
RASDP_ERROR_TRACER_CAPABILITY_OFF	Error tracer capability register
RASDP_ERROR_TRACER_CTRL_OFF	Error tracer control register
RASDP_ERROR_TRACER_FILTER_CTRL_OFF	Error tracer filter control register
RASDP_ERROR_TRACER_LOG_CTRL_OFF	Error tracer log control register
RASDP_ERROR_TRACER_LOG_REPORT_[0-3]_OFF	Error tracer log report register (0-3)
RASDP_ERROR_INJECTION_CTRL	Error injection control register

Figure 3-69 RAS DP Register Access to Error Counters (Same Method Applies to Uncorrectable Errors) when CX_MULTISTREAM_ARCH =0

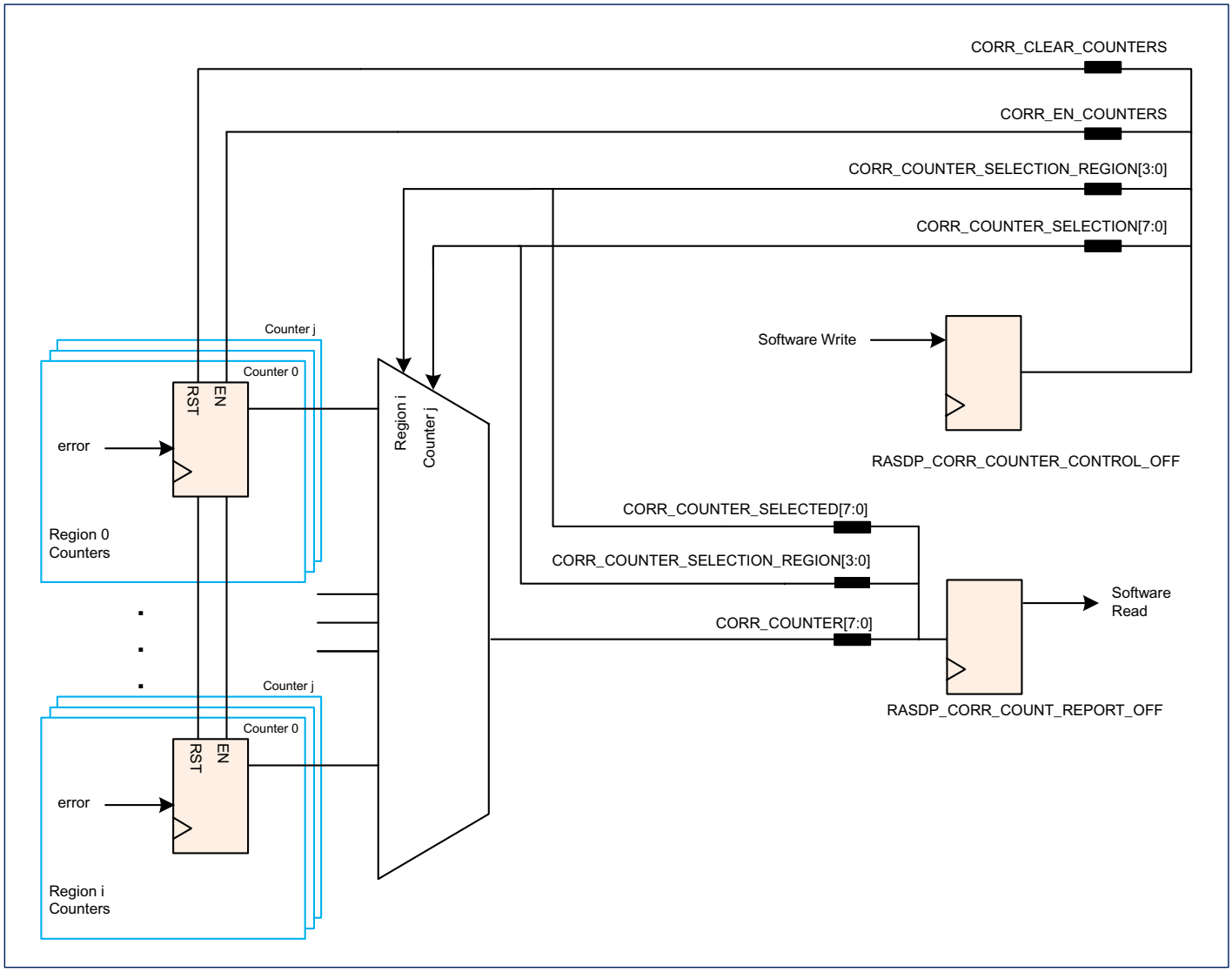
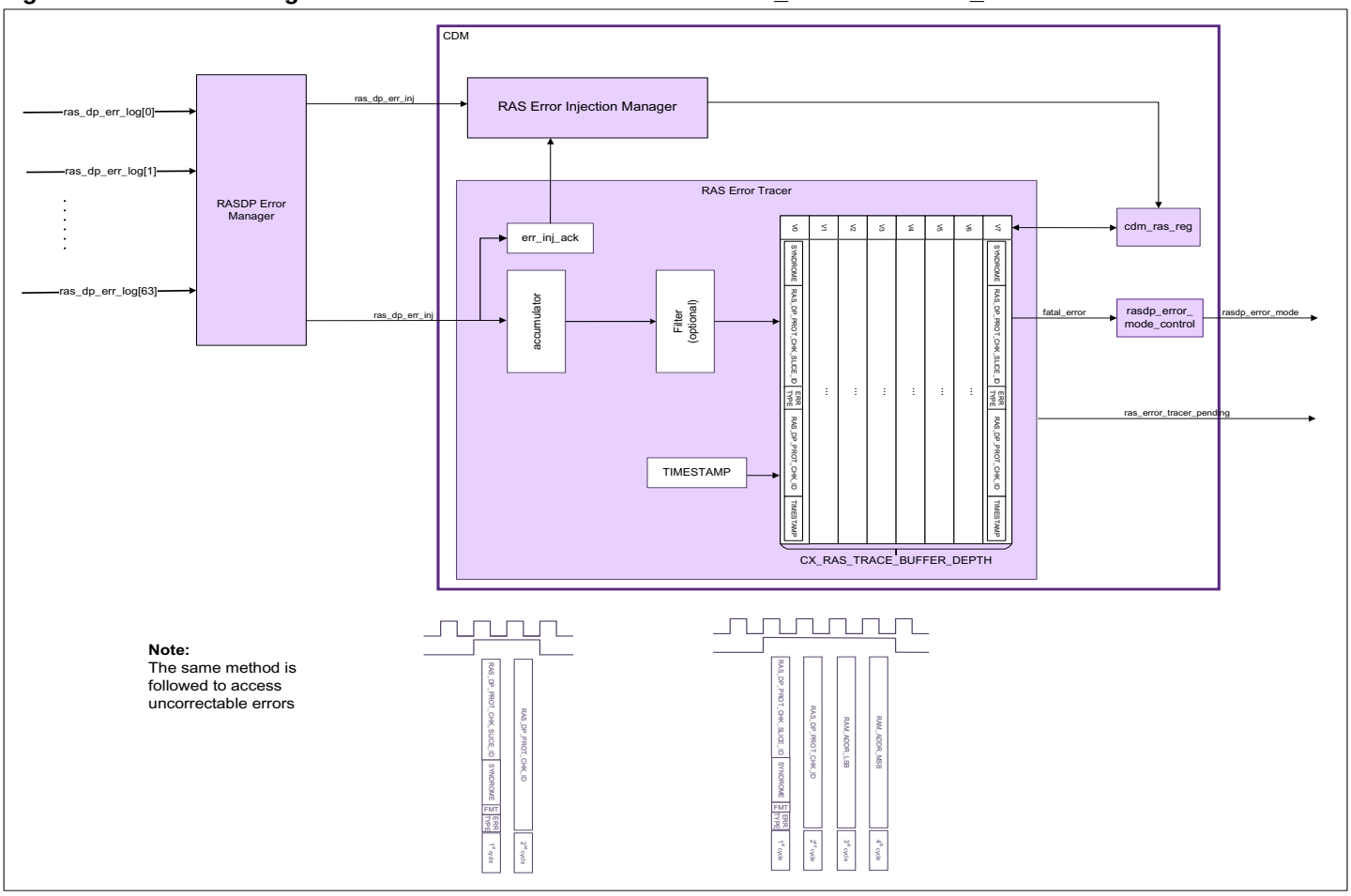


Figure 3-70 RASDP Register Access to Error Counters when CX_MULTISTREAM_ARCH =1



Hint You can inspect the VTB test case (`ft_rasdp.sv`) at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the “Integrating the Controller and VIP using VTB” section in the “Integration” chapter of the *User Guide*.

3.7.2.4 Uncorrectable Error Handling

Upon detection of first Uncorrectable error (RAM and Datapath/Interface Protection), depending on value of `AUTO_LINK_DOWN_EN` field of the `RASDP_ERROR_MODE_EN_OFF` register, the controller takes one of the two actions:

- Forces the PCIe link to go down (`AUTO_LINK_DOWN_EN =1`), or
- Enters RAS DP Error Mode (`AUTO_LINK_DOWN_EN =0`)

PCIe Link Down

If the controller is configured to cause the link to go down upon detection of an Uncorrectable error, the controller resets and resumes normal operation.

RAS DP Error Mode

On entering RAS DP error mode, the controller sets `cfg_rasdp_error_mode =1` (or `slv_rasdp_error_mode/mstr_rasdp_error_mode`). Wire access to CDM/ELBI registers is suspended to prevent corruption of the register configuration space.

TLP handling in RAS DP Error mode:

- Rx TLP Handling
  - Rx Errored TLPs are Forwarded to Your Application
  - Subsequent Rx TLPs not Guaranteed to be Correct; you Must Discard all TLPs[1]
- Tx TLP Handling (32-bit, 64-bit Datapath configurations)
  - Bad TLP is Nullified[2] (EDB Inserted and LCRC Inverted) and Transmitted
  - Subsequent TLPs are Nullified and Transmitted
  - In-progress Replay is Nullified and Transmitted
  - Subsequent Replays are Silently Dropped
- Tx TLP Handling (128-bit, 256-bit and 512-bit datapath configurations which can carry multiple TLPs per clock cycle)
  - Bad TLP is Silently Dropped[3]
  - Subsequent TLPs are Silently Dropped
  - In-progress Replay is Nullified and Transmitted
  - Subsequent Replays are Silently Dropped

Note: The controller attempts a best-effort recovery in RAS DP Error Mode (after detection of an uncorrectable error). Synopsys reserves the right to modify this best-effort behavior if deemed necessary.

Credits accuracy is not retained under uncorrectable error conditions. The controller implements a best-effort approach in accounting for credits under uncorrectable error conditions. However, credit inaccuracies are unavoidable in such conditions. This might result in performance degradation or triggering of overflow protection mechanisms.

Error Logging:

- Uncorrectable[4] Error is Logged
- Uncorrectable Internal Error Reported (AER)[5]

Your application can instruct the controller to exit RAS DP error mode by writing to `RASDP_ERROR_MODE_CLEAR_OFF` register through DBI. Your application must perform any cleanup it may deem necessary prior to instructing the controller to exit RAS DP error mode. In RAS DP error mode, some Uncorrectable errors can be severe enough to eventually cause the link to go down. If link down occurs, the controller resets and resumes normal operation.

1. In RAS DP error mode, TLPs are not guaranteed to be correct. You must discard all TLPs when this signal is high.
2. The controller might nullify TLPs that are received prior to the bad TLP.
3. The controller might drop TLPs that are received prior to the bad TLP.
4. 2-bit ECC and all parity errors.
5. The controller only reports uncorrectable errors when it exits RAS DP error mode.

3.7.2.5 Correctable Error Handling

The controller takes the following action to handle correctable (RAM and Datapath/Interface Protection) errors:

- Correctable[1] Error is Logged
- Correctable Internal Error Reported (AER)
- TLP is Modified/Corrected
- Errors Reported When Detected (Immediately)
- Error Correction Enable/Disable Control (Per TX/RX Direction and Per-Layer)[2]

1. 1-bit ECC errors.
2. If the controller is capable of single bit error correction, disabling ECC error correction does not change the nature of the errors and their reporting. A single-bit error is reported as a correctable error although correction might be disabled.

3.7.3 RAS Debug, Error Injection, and Statistics (DES)

You enable the basic DES operation with the `CX_RAS_DES_ENABLE` parameter. To enable advanced features, you have to enable the respective parameters in addition to `CX_RAS_DES_ENABLE` parameter.

3.7.3.1 Debug

To facilitate system level debugging the controller provides the following features:

- General Diagnostic Support (Legacy Internal Probe Points).
- Silicon Debug Support (Control and Status of PIPE, Deskew, Ack/Nak, Reversal, FC, EQ, LTSSM, Replay)

General Diagnostic Support

The basic diagnostic operation is enabled when the `CX_RAS_DES_ENABLE` parameter is enabled. Your application can use the signals in Table 3-35 to control and check the diagnostic results.

Table 3-35 General Diagnostic Signals

Signal Name	Description
diag_ctrl_bus	Diagnostic control bus.
diag_status_bus	This bus provides information regarding all the important status signals from each controller module.

Silicon Debug Support

You enable this feature with the `CX_RAS_DES_SD_ENABLE` parameter. Your application can use the outputs in Table 3-36 and registers in Table 3-37.

Table 3-36 Silicon Debug Outputs

Output	Description
cdm_ras_des_sd_info_common	This is common debug signal bus that provides RAS D.E.S. silicon debug information.
cdm_ras_des_sd_info_li (for i = 0; i < CX_NL)	This is Lane*i* debug signal bus that provides RAS D.E.S. silicon debug information.
cdm_ras_des_sd_info_vi (for i = 0; i < CX_NVC)	This is VC*i* debug signal bus that provides RASD.E.S. silicon debug information.

Table 3-37 Vendor Specific and Silicon Debug Registers

Register	Description
Vendor Specific Registers	
RAS_DES_CAP_HEADER_REG	Vendor-Specific Extended Capability Header.
VENDOR_SPECIFIC_HEADER_REG	Vendor-Specific Header.

Register	Description
Silicon Debug Registers	
SD_CONTROL1_REG on	Silicon Debug Control1.
SD_CONTROL2_REG on	Silicon Debug Control2.
SD_STATUS_L1LANE_REG	Silicon Debug Status(Layer1 Per-lane).
SD_STATUS_L1LTSSM_REG	Silicon Debug Status(Layer1 LTSSM).
SD_STATUS_PM_REG	Silicon Debug Status(PM).
SD_STATUS_L2_REG	Silicon Debug Status(Layer2).
SD_STATUS_L3FC_REG	Silicon Debug Status(Layer3 FC).
SD_EQ_CONTROL1_REG	Silicon Debug EQ Control1.
SD_EQ_STATUS1_REG	Silicon Debug EQ Status1.
SD_EQ_STATUS2_REG	Silicon Debug EQ Status2.
SD_EQ_STATUS3_REG	Silicon Debug EQ Status3.

In addition, you can also use the non-optional silicon debug outputs in Table 3-38 and registers in Table 3-39 that always exist regardless of parameter settings.

Table 3-38 Pre-existing Silicon Debug Outputs

Output	
rdlh_link_up	Data link layer up/down indicator.
brdg_slv_xfer_pending	AXI Slave non-DBI transfer pending status.
brdg_dbi_xfer_pending	AXI Slave DBI transfer pending status.
edma_xfer_pending	DMA transfer pending status.
radm_xfer_pending	Receive request pending status.
rtlh_rfc_upd	Indicates that the controller received a flow control update DLLP.
rtlh_rfc_data	The data from a received flow control update DLLP.
radm_q_not_empty	Level indicating that the receive queues contain TLP header/data.
radm_qoverflow	Pulse indicating that one or more of the P/NP/CPL receive queues have overflowed.
cxpl_debug_info[63:0]	State of selected internal signals,
cxpl_debug_info_ei[15:0]	State of selected internal signals in relation to electrical idle (EI) at the receiver.

Table 3-39 Pre-existing Silicon Debug Registers

Register	Fields
TX_P_FC_CREDIT_STATUS_OFF	TX_P_HEADER_FC_CREDIT TX_P_DATA_FC_CREDIT
TX_NP_FC_CREDIT_STATUS_OFF	TX_NP_HEADER_FC_CREDIT TX_NP_DATA_FC_CREDIT
TX_CPL_FC_CREDIT_STATUS_OFF	TX_CPL_HEADER_FC_CREDIT TX_CPL_DATA_FC_CREDIT
QUEUE_STATUS_OFF	RX_QUEUE_NON_EMPTY TX_RETRY_BUFFER_NE RX_TLP_FC_CREDIT_NON_RETURN
PL_DEBUG0_OFF	DEB_REG_0
PL_DEBUG1_OFF	DEB_REG_1
GEN3_EQ_COEFF_LEGALITY_STATUS_OFF	GEN3_EQ_VIOLATE_COEF_RULES
VC0_P_RX_Q_CTRL_OFF	VC0_P_TLP_Q_MODE VC0_P_HEADER_CREDIT VC0_P_DATA_CREDIT
VC0_NP_RX_Q_CTRL_OFF	VC0_NP_TLP_Q_MODE VC0_NP_HEADER_CREDIT VC0_NP_DATA_CREDIT
VC0_CPL_RX_Q_CTRL_OFF	VC0_CPL_TLP_Q_MODE VC0_CPL_HEADER_CREDIT VC0_CPL_DATA_CREDIT
GEN2_CTRL_OFF	GEN1_EI_INFERENCE PRE_DET_LANE NUM_OF_LANES FAST_TRAINING_SEQ
PIPE_LOOPBACK_CONTROL_OFF	PIPE_LOOPBACK
ACK_LATENCY_TIMER_OFF	REPLAY_TIME_LIMIT ROUND_TRIP_LATENCY_TIME_LIMIT
VENDOR_SPEC_DLLP_OFF	VENDOR_SPEC_DLLP
PORT_FORCE_OFF	CPL_SENT_COUNT LINK_STATE FORCE_EN FORCED_LTSSM LINK_NUM

Register	Fields
ACK_F_ASPM_CTRL_OFF	ENTER_ASPM L1_ENTRANCE_LATENCY L0S_ENTRANCE_LATENCY ACK_FREQ
PORT_LINK_CTRL_OFF	TRANSMIT_LANE_REVERSALE_ENABLE EXTENDED_SYNCH BEACON_ENABLE LINK_CAPABLE LINK_RATE FAST_LINK_MODE LINK_DISABLE DLL_LINK_EN RESET_ASSERT SCRAMBLE_DISABLE VENDOR_SPECIFIC_DLLP_REQ
LANE_SKEW_OFF	DISABLE_LANE_TO_LANE_DESKEW
IMER_CTRL_MAX_FUNC_NUM_OFF	UPDATE_FREQ_TIMER TIMER_MOD_ACK_NAK TIMER_MOD_REPLAY_TIMER MAX_FUNC_NUM
SYMBOL_TIMER_FILTER_1_OFF	MASK_RADM_1 DISABLE_FC_WD_TIMER EIDLE_TIMER
FILTER_MASK_2_OFF	MASK_RADM_2
AMBA_MUL_OB_DECOMP_NP_SUB_REQ_CTRL_OFF	OB_RD_SPLIT_BURST_EN
QUEUE_STATUS_OFF	TIMER_MOD_FLOW_CONTROL_EN TIMER_MOD_FLOW_CONTROL
GEN3_RELATED_OFF	USP_SEND_8GT_EQ_TS2_DISABLE AUTO_EQ_DISABLE GEN3_DC_BALANCE_DISABLE GEN3_DLLP_XMT_DELAY_DISABLE GEN3_EQUALIZATION_DISABLE RXEQ_PH01_EN EQ_REDO EQ_EIEOS_CNT EQ_PHASE_2_3 DISABLE_SCRAMBLER_GEN_3 GEN3_ZRXDC_NONCOMPL

Register	Fields
GEN3_EQ_LOCAL_FS_LF_OFF	GEN3_EQ_LOCAL_FS GEN3_EQ_LOCAL_LF
GEN3_EQ_PSET_COEF_MAP__i (for i = 0; i <= 10)	GEN3_EQ_POST_CURSOR_PSET GEN3_EQ_CURSOR_PSET GEN3_EQ_PRE_CURSOR_PSET
GEN3_EQ_PSET_INDEX_OFF	GEN3_EQ_PSET_INDEX
GEN3_EQ_CONTROL_OFF	GEN3_EQ_PSET_REQ_AS_COEF GEN3_EQ_FOM_INC_INITIAL_EVAL GEN3_EQ_PSET_REQ_VEC GEN3_EQ_EVAL_2MS_DISABLE GEN3_EQ_PHASE23_EXIT_MODE GEN3_EQ_FB_MODE
GEN3_EQ_FB_MODE_DIR_CHANGE_OFF	GEN3_EQ_FMDC_MAX_POST_CUSROR_DELTA GEN3_EQ_FMDC_MAX_PRE_CUSROR_DELTA GEN3_EQ_FMDC_N_EVALS GEN3_EQ_FMDC_T_MIN_PHASE23

3.7.3.2 Error Injection

You enable this feature with the `CX_RAS_DES_EINJ_ENABLE` parameter. The controller provides support for your application to inject the following types of errors:

- CRC Error
- Sequence Number Error
- DLLP Error
- Symbol DataK Mask Error or Sync Header Error
- FC Credit Update Error
- TLP Duplicate/Nullify Error
- Specific TLP Error

Your application can use the registers in Table 3-40 to set up error injection.

Table 3-40 Error Injection Vendor Specific Extended Capability (VSEC) Registers

Register	Description
RAS_DES_CAP_HEADER_REG	Vendor-Specific Extended Capability Header.
VENDOR_SPECIFIC_HEADER_REG	Vendor-Specific Header.
.....other registers (for example, Event Counter, Time-based Analysis, Error Injection)	
EINJ_ENABLE_REG	Error Injection Enable.

Register	Description
EINJ0_CRC_REG	Error Injection Control 0 (CRC Error).
EINJ1_SEQNUM_REG	Error Injection Control 1 (SeqNum Error).
EINJ2_DLLP_REG	Error Injection Control 2 (DLLP Error).
EINJ3_SYMBOL_REG	Error Injection Control 3 (Symbol Error).
EINJ4_FC_REG	Error Injection Control 4 (FC Credit Error).
EINJ5_SP_TLP_REG	Error Injection Control 5 (Specific TLP Error).
EINJ6_COMPARE_POINT_H0_REG	Error Injection Control 6 (Compare Point H0).
EINJ6_COMPARE_POINT_H1_REG	Error Injection Control 6 (Compare Point H1).
EINJ6_COMPARE_POINT_H2_REG	Error Injection Control 6 (Compare Point H2).
EINJ6_COMPARE_POINT_H3_REG	Error Injection Control 6 (Compare Point H3).
EINJ6_COMPARE_VALUE_H0_REG	Error Injection Control 6 (Compare Value H0).
EINJ6_COMPARE_VALUE_H1_REG	Error Injection Control 6 (Compare Value H1).
EINJ6_COMPARE_VALUE_H2_REG	Error Injection Control 6 (Compare Value H2).
EINJ6_COMPARE_VALUE_H3_REG	Error Injection Control 6 (Compare Value H3).
EINJ6_CHANGE_POINT_H0_REG	Error Injection Control 6 (Change Point H0).
EINJ6_CHANGE_POINT_H1_REG	Error Injection Control 6 (Change Point H1).
EINJ6_CHANGE_POINT_H2_REG	Error Injection Control 6 (Change Point H2).
EINJ6_CHANGE_POINT_H3_REG	Error Injection Control 6 (Change Point H3).
EINJ6_CHANGE_VALUE_H0_REG	Error Injection Control 6 (Change Value H0).
EINJ6_CHANGE_VALUE_H1_REG	Error Injection Control 6 (Change Value H1).
EINJ6_CHANGE_VALUE_H2_REG	Error Injection Control 6 (Change Value H2).
EINJ6_CHANGE_VALUE_H3_REG	Error Injection Control 6 (Change Value H3).
EINJ6_TLP_REG	Error Injection Control 6 (Packet Error).

Hint

You can inspect the VTB test case `ft_ras_des_einj.sv` at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see “Integrating the Controller and VIP using VTB” section in the “Integration” chapter of the *User Guide*.

Limitations

You can use the Error Injection6 only when both the conditions are satisfied.

- The RAS Datapath Protection (DP) is disabled (`CX_RASDP =0`, `CX_RASDP_RAM_PROT =0`), and
- The address translation is disabled (`ADDR_TRANSLATION_SUPPORT_EN =0`)

3.7.3.3 Statistics

To facilitate collection of statistics the controller provides the following features:

- Time Based Analysis (RX/TX data throughput and time spent in each low-power LTSSM state)
- Event counters (Error and Non-Error)

Time Based Analysis

You enable this feature with the `CX_RAS_DES_TBA_ENABLE` parameter. Using this feature you can obtain information regarding RX/TX data throughput and time spent in each low-power LTSSM state by the controller. Your application can use the outputs in Table 3-41 and registers in Table 3-43.

Table 3-41 Time-based Analysis Signals

Signal Name	
app_ras_des_tba_ctrl	This bus controls the start/end of time based analysis.
cdm_ras_des_tba_info_common	This is common event signal bus that provides RAS D.E.S. time based analysis information.

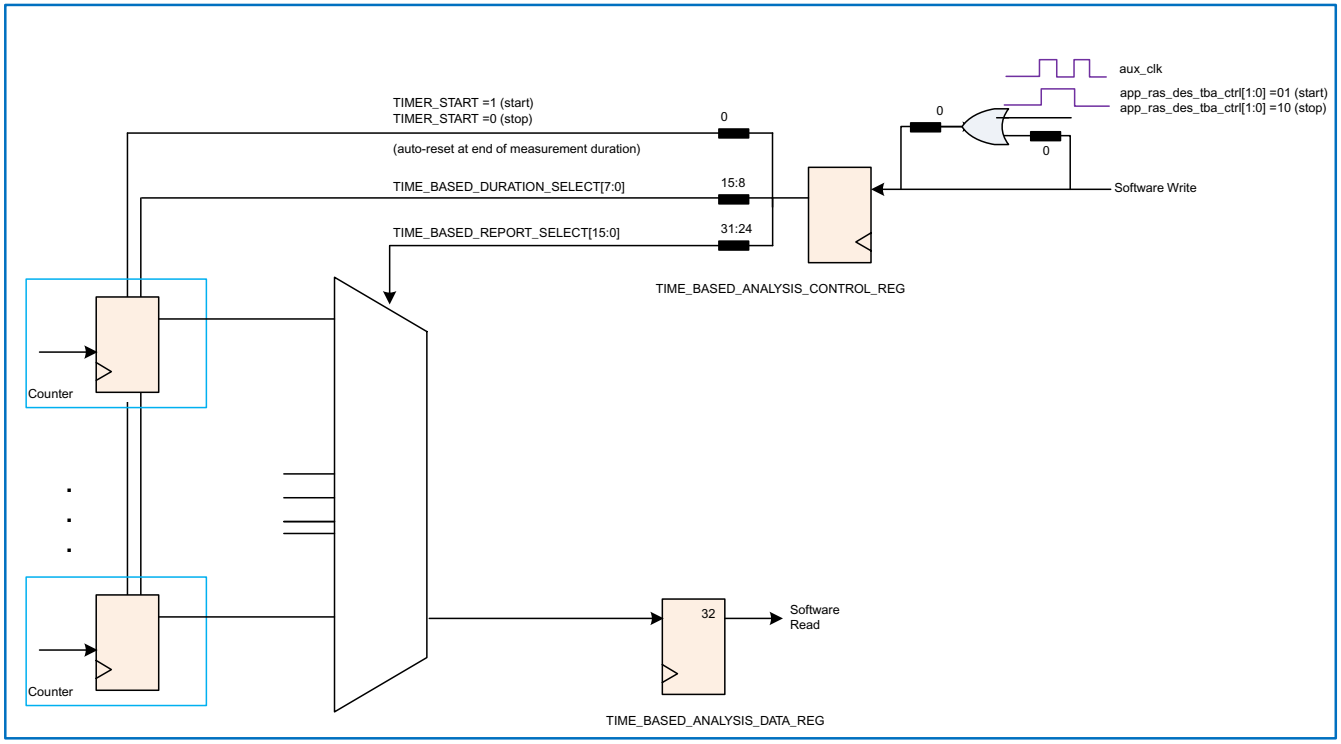
Table 3-42 Time-based Analysis RAM Signals

Signal Name	
cdm_ras_des_tb_ram_*	This signal bus provides RAS D.E.S. time based analysis information for the RAMs.

Table 3-43 Time-based Analysis Vendor Specific Extended Capability (VSEC) Registers

Register	Description
RAS_DES_CAP_HEADER_REG	Vendor-Specific Extended Capability Header
VENDOR_SPECIFIC_HEADER_REG	Vendor-Specific Header.
.....other registers (for example, Event Counter).....	
TIME_BASED_ANALYSIS_CONTROL_REG	Time-based Analysis Control.
TIME_BASED_ANALYSIS_DATA_REG	Time-based Analysis Data.

Figure 3-71 Time-based Analysis Register Access



The counters measure, for example, what percentage of time does the controller stay in L0 in a one second window (configurable through `TIME_BASED_DURATION_SELECT` field). The measurement range of each Event in Group#0 is shown in Figure 3-72.

Figure 3-72 Measurement Ranges of Time-based Analysis Events in Group#0

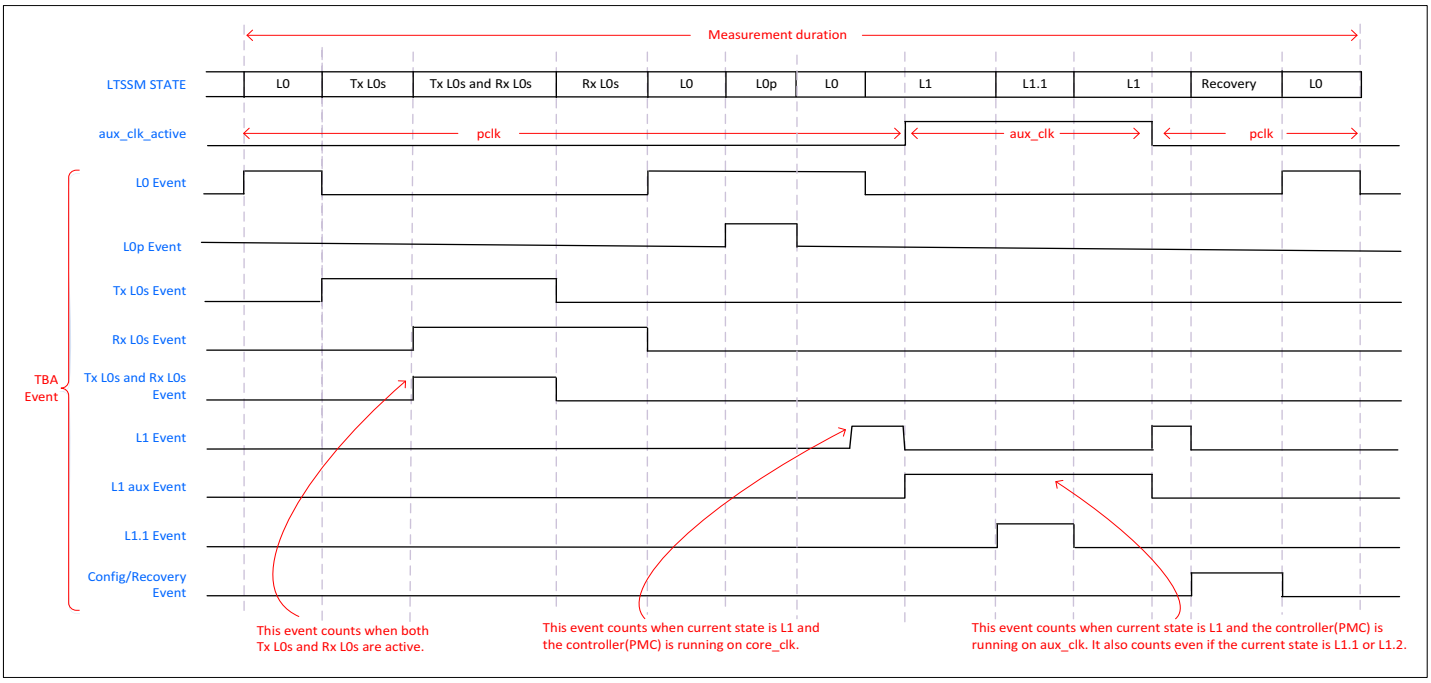


Table 3-44 Time-based Analysis Counter Group #0 (32-bit Low-Power Cycle Counter (RAM)

Event #	% of Measurement Window Spent in LTSSM State	Note
0x01	Tx L0s	
0x02	Rx L0s	
0x03	L0	
0x04	L1	The controller (PMC) is running on core_clk
0x05	L1.1	
0x06	L1.2	
0x07	Configuration/Recovery	
0x08	L1 aux	The controller (PMC) is running on aux_clk
0x09	Tx L0s and Rx L0s (in non-flit mode or L0p in flit mode)	

Table 3-45 Time-based Analysis Counter Group #1 (32-bit Throughput 4-DWORD Counter)

Event #	Amount of Data Processed (Units of 16 bytes)	Note
0x00	Tx PCIe TLP Data Payload	
0x01	Rx PCIe TLP Data Payload	

Event #	Amount of Data Processed (Units of 16 bytes)	Note
0x02	Tx CCIX TLP Data Payload	PCIe Compatible TLP: CCIX TLP Data Payload consists of the data payload + TLP Header DW3CCIX Optimized TLP: CCIXTLP Data Payload consists of the data payload + 1 DW header
0x03	Rx CCIX TLP Data Payload	PCIe Compatible TLP: CCIX TLP Data Payload consists of the data payload + TLP Header DW3CCIX Optimized TLP: CCIXTLP Data Payload consists of the data payload + 1 DW header

Note

- **L2 Exit:** All RASDES analysis results are cleared when the controller exits from L2.
- **Power Gating:** RASDES results exhibit undefined behavior if the RASDES analysis is active during power gating. The PMC does not store RASDES analysis results. On removing power from the RAMs connected to cdm_rasdes_ec_ram*/cdm_rasdes_tba_ram*, RASDES analysis results are lost.

Event Counters

You enable this feature with the `CX_RAS_DES_EC_ENABLE` parameter. Your application can use the outputs in Table 3-46 and registers in Table 3-48.

Table 3-46 Event Counter Signals

Output	Description
cdm_ras_des_ec_info_common	This is a common signal bus that provides RASD.E.S. event counter information.
cdm_ras_des_ec_info_i(for i = 0; i <CX_NL))	This is Lane*i* event signal bus that provides RASD.E.S. event counter information.

Table 3-47 Event Counter RAM Signals

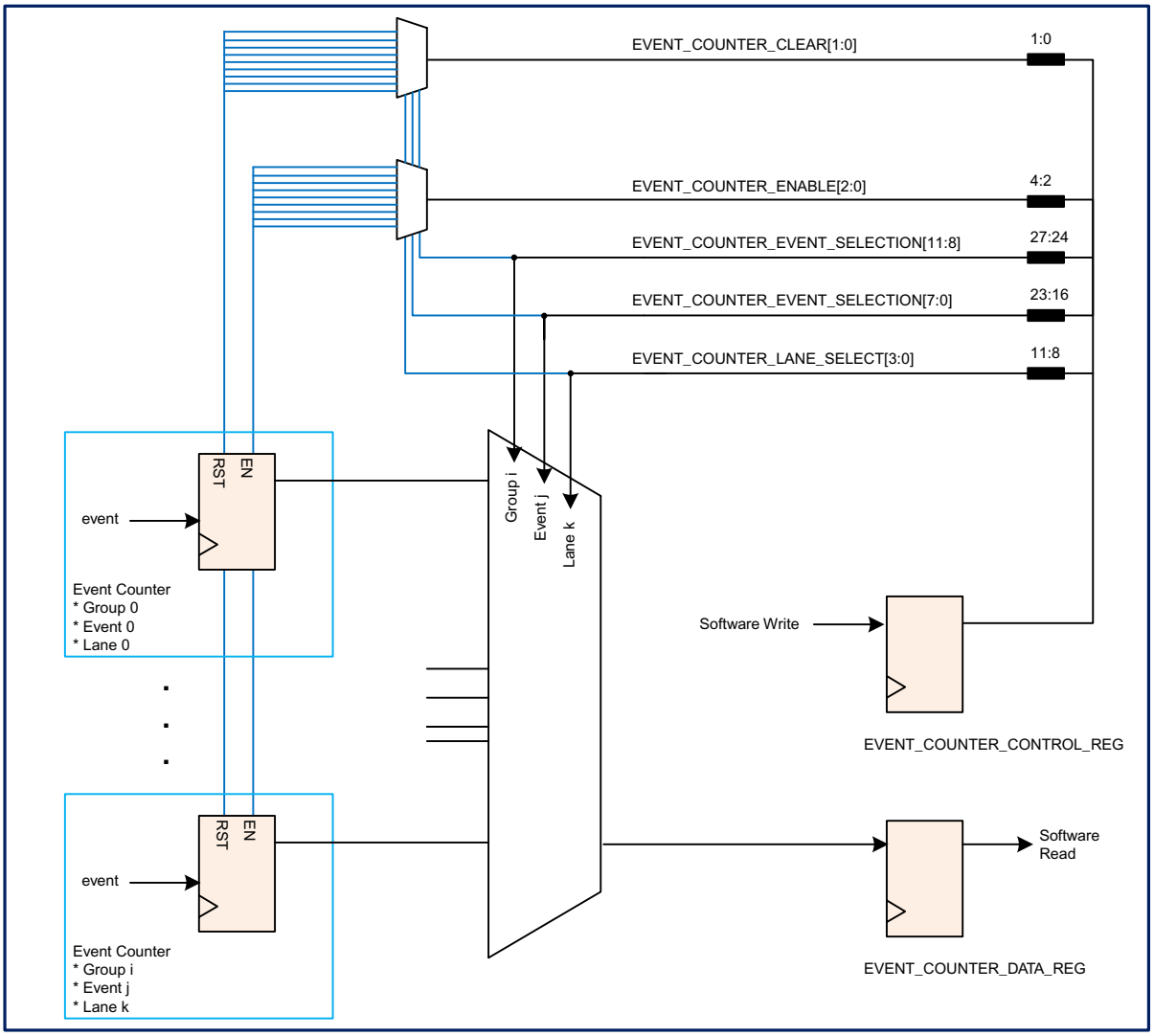
Signal Names	Description
cdm_ras_des_ec_ram_*	This signal bus provides RAS D.E.S. Event Counter information for the RAMs.

Table 3-48 Event Counter VSEC Registers

Register	Description
RAS_DES_CAP_HEADER_REG	Vendor-Specific Extended Capability Header
VENDOR_SPECIFIC_HEADER_REG	Vendor-Specific Header
EVENT_COUNTER_CONTROL_REG	Event Counter Control

Register	Description
EVENT_COUNTER_DATA_REG	Event Counter Data

Figure 3-73 Event Counter Register Access



Event Counter Data Details

The following tables provide the Group/Event numbers which are required when using the `EVENT_COUNTER_CONTROL_REG` viewport control register to read each of the event counters. Some of the event counters only exist for specific configurations. Groups 5-7 are implemented in RAM as indicated. The group # is set by `EVENT_COUNTER_EVENT_SELECTION[11:8]`and the event # is set by `EVENT_COUNTER_EVENT_SELECTION[7:0]`

Table 3-49 Event Counter Group #0 (4-bit Layer1 Error Counter Per-Lane)

Event #	Description	Note
0x00	EBUF Overflow	-
0x01	EBUF Under-run	-
0x02	Decode Error	-
0x03	Running Disparity Error	-
0x04	SKP OS Parity Error	Gen3 only
0x05	SYNC Header Error	Gen3/4/5
0x06	Rx Valid de-assertion	Rx Valid de-assertion without EIOSs
0x07	CTL SKP OS Parity Error	Gen4 or above
0x08	1st Retimer Parity Error	Gen4 or above
0x09	2nd Retimer Parity Error	Gen4 or above
0x0A	Margin CRC and Parity Error	Gen4 or above

Table 3-50 Event Counter Group #1 (8-bit Layer1 Error Counter Common-Lane)

Event #	Description	Note
0x00	Reserved	-
0x01	Reserved	-
0x02	Reserved	-
0x03	Reserved	-
0x04	Reserved	-
0x05	Detect EI Infer	L0 to Recovery transition trigger
0x06	Receiver Error	-
0x07	Rx Recovery Request	When the controller receives TS1 OS in L0 state. L0 to Recovery transition trigger
0x08	N_FTS Timeout	Timeout when the controller's Rx condition is in Rx_L0.FTS. L0 to Recovery transition trigger
0x09	Framing Error	Gen3 or above
0x0A	Deskew Error	L0 to Recovery transition trigger
0x0B	Received EIOS when L0s is not supported and not directed L1 or L2	L0 to Recovery transition trigger
0x0C	Framing Error in L0 State	Gen3 or above. L0 to Recovery transition trigger

Event #	Description	Note
0x0D	Deskew Uncompleted Error	L0 to Recovery transition trigger

Table 3-51 Event Counter Group #2 (8-bit Layer2 Error Counter Common-Lane)

Event #	Description	Note
0x00	BAD TLP	-
0x01	LCRC Error	-
0x02	BAD DLLP	-
0x03	Replay Number Rollover	L0 to Recovery transition trigger
0x04	Replay Timeout	-
0x05	Rx Nak DLLP	-
0x06	Tx Nak DLLP	-
0x07	Retry TLP	-

Table 3-52 Event Counter Group #3 (8-bit Layer3 Error Counter Common-Lane)

Event[a] #	Description	Note
0x00	FC Timeout	L0 to Recovery transition trigger
0x01	Poisoned TLP	Any function
0x02	ECRC Error	Any function
0x03	Unsupported Request	Any function
0x04	Completer Abort	Any function
0x05	Completion Timeout	Any function

a. Malformed TLP, FC & DL Protocol errors are excluded because of low frequency.

Table 3-53 Event Counter Group #4 (4-bit Layer1 Non-Error Counter Per-Lane)

Event #	Description	Note
0x00	EBUF SKP Add	-
0x01	EBUF SKP Del	-

Table 3-54 Event Counter Group #5 (32-bit Layer1 Non-Error Counter [RAM] Common Lane)

Event #	Description	Note
0x00	L0 to Recovery Entry	-
0x01	L1 to Recovery Entry	-

Event #	Description	Note
0x02	Tx L0s Entry	-
0x03	Rx L0s Entry	-
0x04	ASPM L1 reject	Send PM_Active_State_Nak
0x05	L1 Entry	-
0x06	L1 CPM	CX_L1_SUBSTATES_ENABLE
0x07	L1.1 Entry	CX_L1_SUBSTATES_ENABLE
0x08	L1.2 Entry	CX_L1_SUBSTATES_ENABLE
0x09	L1 short duration	(CX_L1_SUBSTATES_ENABLE) No. of times link entered L1.0 state but exited to L0 without entering L1.1 or L1.2 states
0x0A	L1.2 abort	(CX_L1_SUBSTATES_ENABLE) No. of times link entered L1.2_Entry state but is forced to exit to L1.0 then L0 because it saw clkreq# asserted while in this state
0x0B	L2 Entry	-
0x0C	Speed Change	Gen2
0x0D	Link width Change	x2 (including L0p in Flit mode)
0x0E	Reserved	-

Table 3-55 Event Counter Group #6 (32-bit Layer2 Non-Error Counter [RAM])

Event #	Description	Note
0x00	Tx Ack DLLP	-
0x01	Tx Update FC DLLP	-
0x02	Rx Ack DLLP	-
0x03	Rx Update FC DLLP	-
0x04	Rx Nullified TLP	-
0x05	Tx Nullified TLP	-
0x06	Rx Duplicate TLP	-

Table 3-56 Event Counter Group #7 (32-bit Layer3 Non-Error Counter [RAM])

Event #	Description	Note
0x00	Tx Memory Write	-

Event #	Description	Note
0x01	Tx Memory Read	-
0x02	Tx Configuration Write	-
0x03	Tx Configuration Read	-
0x04	Tx IO Write	-
0x05	Tx IO Read	-
0x06	Tx Completion without data	-
0x07	Tx Completion w data	-
0x08	Tx Message TLP	PCIe VC only
0x09	Tx Atomic	-
0x0A	Tx TLP with Prefix	-
0x0B	Rx Memory Write	-
0x0C	Rx Memory Read	-
0x0D	Reserved	-
0x0E	Reserved	-
0x0F	Rx IO Write	-
0x10	Rx IO Read	-
0x11	Rx Completion without data	-
0x12	Rx Completion w data	-
0x13	Rx Message TLP	PCIe VC only
0x14	Rx Atomic	-
0x15	Rx TLP with Prefix	-
0x16	Tx CCIX TLP	CCIX VC Only (PCIe Compatible TLP or CCIX Optimized TLP)
0x17	Rx CCIX TLP	CCIX VC Only (PCIe Compatible TLP or CCIX Optimized TLP)
0x18	Tx Deferrable Memory Write TLP	-
0x19	Rx Deferrable Memory Write TLP	-

Table 3-57 Event Counter Group #8 (8-Bit CXL Error Counter Common-Lane (CXL Only))

Event#	Description	Note
0x00	ALMP Replication Error	L0 to Recovery transition trigger
0x01	Unexpected ALMP error	L0 to Recovery transition trigger

Event#	Description	Note
0x02	NUM_RETRY reached MAX_NUM_RETRY	L0 to Recovery transition trigger

Hint

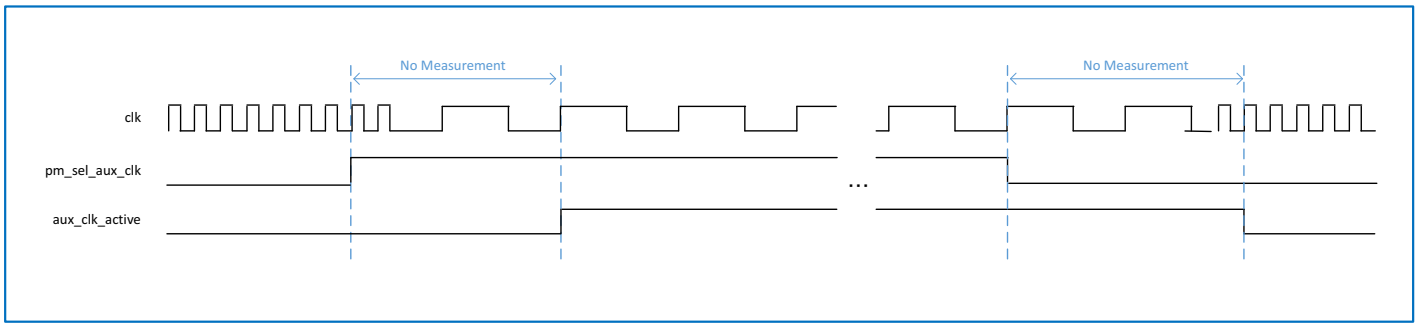
You can inspect the VTB test case `ft_ras_des_event_count.sv` at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the “Integrating the Controller and VIP using VTB” section in the “Integration” chapter of the *User Guide*.

Limitations

If the controller (PMC) is running on `aux_clk` during the measurement:

- The measurement timers (`TIME_BASED_DURATION_SELECT`: 1 ms, 10 ms, …, 2 s, 4 s) have measurement error less than 4 us.
- The measurement timers and event timers of all the events are stopped temporarily when `aux_clk_active/pm_sel_aux_clk` are controlled. Figure 3-74 describes the timing details when `aux_clk_active/pm_sel_aux_clk` are controlled.

Figure 3-74 No Measurement Period



3.8 Messages

This section describes the processing of messages through the controller. The following topics are discussed:

- "Message Generation" on page 194
- "Message Reception" on page 205

For a proper understanding of messages you should be familiar with Section 2.2.8, "Message Request Rules" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

- Messages (Msg/MsgD[1]) are posted transactions.
- Vendor Defined messages are Msg / MsgD.
- PTM messages are Msg / MsgD.
- Set Slot Power Limit, Invalidate Request, and LN messages are MsgD.
- All other messages are Msg.

For more information, see "Interrupts" on page 210.

1. With payload.

3.8.1 Message Generation

The following topics are discussed in this section:

- "Overview"
- "Vendor Defined Message (VDM) Generation" on page 196
- "OBFF Message Generation" on page 197
- "OBFF Wake Signal Encoder Reference Design" on page 198
- "Application Msg/MsgD Programming Examples" on page 201
- "Byte Mapping of Third and Fourth Message Header DWORDs at I/O Interfaces" on page 202

3.8.1.1 Overview

Messages that are transmitted by the controller are derived from the following sources:

- the controller (automatically)
- the controller (under the control of your application)
- your application
- the host/client software

Note

For internally generated messages, or for messages generated at any of the SII interfaces:

- ID Based Ordering (IDO) is not supported.
- The controller does not check the messages for TLP errors; instead it sends the TLP as presented on the message interfaces.

Figure 3-75 Message Transmission: RP

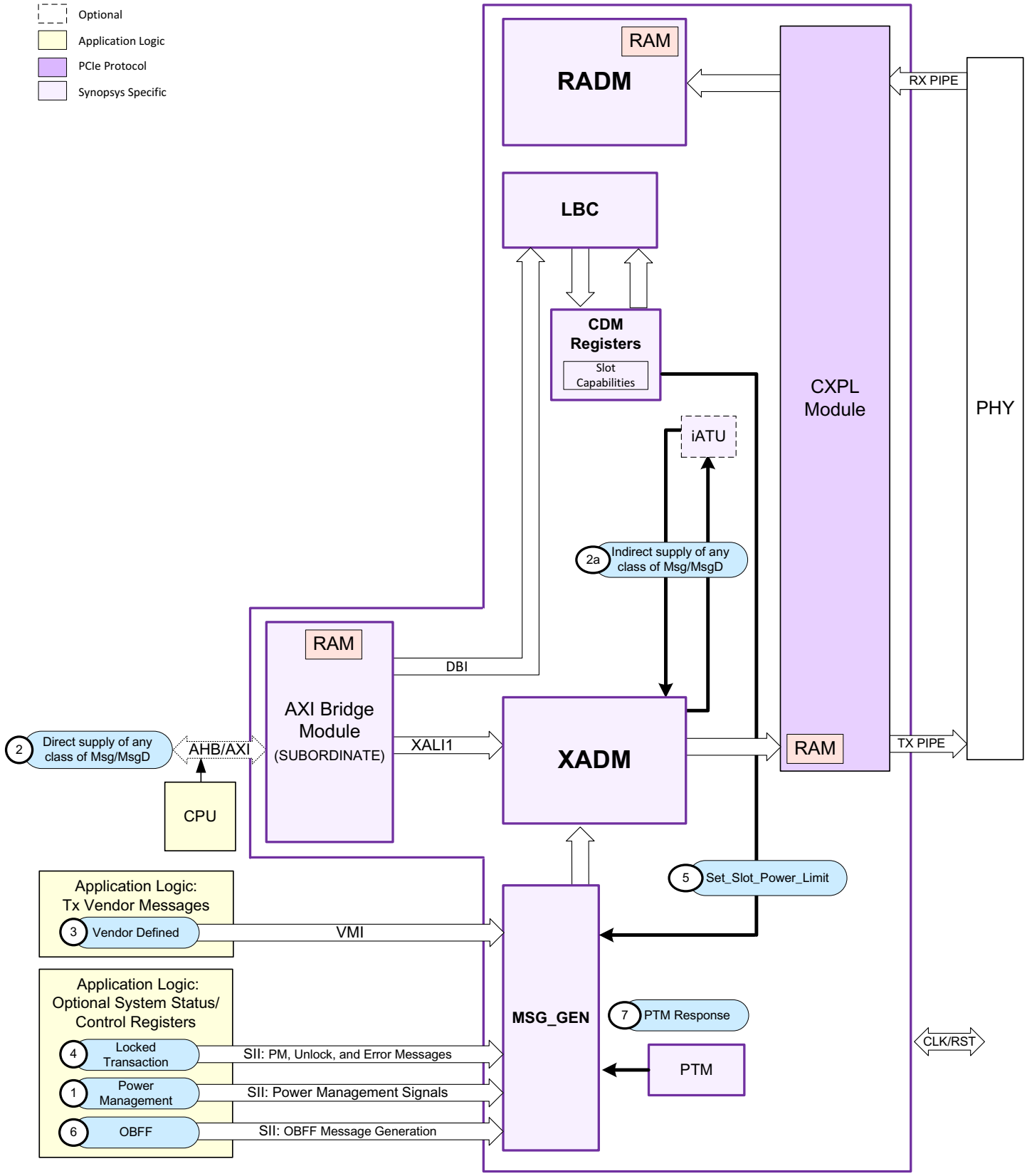


Table 3-58 Types of Transmitted Messages

Index	Message Source (Type)	Details
1	Power Management interface (Msg)	For more information, see "Power Management" on page 259.
2	Direct Supply of any class of message (Msg/MsgD)	XALI0/1/2 For more information on how to generate a message at the XALI0/1/2 interface, see "Transmit Client Interface Protocol Rules" on page 356 and "Application Msg/MsgD Programming Examples" on page 201.
2a	Indirect Supply of any class of message (Msg/MsgD)	For more information on generating Msg/MsgD from MWr/IOWr using an internal address translation unit (ATU), see "IO/MEM to Msg/MsgD Type Translation" on page 229. For more information on the supported methods of Msg/MsgD generation, see Table 3-60 and Table 3-61.
3	Vendor Defined (Msg[a])	The controller generates Vendor Defined messages in response to requests on the VMI. For more information, see "Vendor Message Interface (VMI)" on page 381.
4	Locked Transaction (Msg)	Unlock message, triggered by Root Complex application logic through the `app_unlock_msg` pin.
5	Set_Slot_Power_Limit (MsgD)	Set_Slot_Power_Limit message, triggered by writing to the Slot Capabilities register through the DBI, or when LTSSM enters L0.
6	Optimized Buffer Flush Fill (OBFF) (Msg)	OBFF messages, triggered by your application logic through the `app_obff_obff_msg_req`/`app_obff_idle_msg_req`/`app_obff_cpu_active_msg_req` inputs. For more information, see "OBFF Message Generation" on page 197.
7	PTM Response (Msg/MsgD)	For more information, see "Precision Time Measurement (PTM)" on page 325.

a. MsgD transmission not possible on Vendor Message Interface (VMI). For more information, see "Vendor Defined Message (VDM) Generation" on page 196. However, it is possible through (4).

3.8.1.2 Vendor Defined Message (VDM) Generation

VDMs can be generated by your application using any of the following methods:

- The controller generates VDMs in response to requests on the VMI.
- The VMI can be used to send Msg only. It does not support message with payload (MsgD).
- The inputs `ven_msg_fmt[1:0]` and `ven_msg_len[9:0]` should be set to 0x1 and 0x0 respectively to indicate 4-DWORD header and no payload.
- VDMs created by you at the VMI are always subject to translation by the "Internal Address Translation Unit (iATU)" on page 220.

Note

It is preferable to use the SII interface to send a VDM.

The AXI bridge subordinate can also support message transmission of a VDM without any modifications. A full set of limitations for the AXI bridge is available at "AXI Bridge Limitations" on page 398.

- Direct supply of Msg and MsgD TLPs at XALI0/1/2 (or AXI bridge subordinate when present).
- Direct supply of I/O and MEM TLPs at XALI0/1/2 (or AXI bridge subordinate when present) to be converted to VDM.

  The optional internal address translation unit (iATU) can convert I/O and MEM TLPs to VDM TLPs. For more information, see "IO/MEM to Msg/MsgD Type Translation" on page 229.

Caution

You must use 64-bit addressing when you are using the Direct supply method of VDM generation. For more information, see "Byte Mapping of Third and Fourth Message Header DWORDs at I/O Interfaces" on page 202.

3.8.1.3 OBFF Message Generation

You can enable OBFF message generation by setting `CX_OBFF_SUPPORT`. You can generate OBFF messages using any of the following methods:

- The controller generates OBFF messages in response to requests made by your application logic on the SII `app_obff_*_msg_req` inputs.
- Direct supply of Msg TLPs at XALI0/1/2 (or AXI bridge subordinate when present).
- Direct supply of I/O and MEM TLPs at XALI0/1/2 (or AXI bridge subordinate when present) to be converted to OBFF messages. The optional iATU) can convert I/O and MEM TLPs to Msg TLPs. For more information, see "MEM or I/O Match Modes" on page 238.

You must not send an OBFF message when the OBFF Enable field of the Device Control 2 Register (`DEVICE_CONTROL2_DEVICE_STATUS2_REG`) is 00. When this field is 00, the controller does not block transmission of OBFF messages that you generate. The value of the OBFF Enable field is reflected on the `cfg_obff_en[1:0]` output.

When you configure the controller to support either variation (A or B) of OBFF messages (through the OBFF Enable field of the Device Control 2 Register (`DEVICE_CONTROL2_DEVICE_STATUS2_REG`), and your application sends an OBFF message when the controller is in the L0s or L1 TX states, then the controller transitions to the L0 state and transmits the message. According to the *PCI Express Base Specification, Revision 6.0, Version 1.0*, the controller should drop a Variation A message in this scenario. Therefore, your application must not send Variation A OBFF messages when the controller is in the L0s or L1 TX states. Your application must monitor the `pm_linkst_in_l0s`, `pm_linkst_in_l1sub`, and `pm_linkst_in_l1` outputs. For the purpose of controller optimization the controller does not do this check; your application must do the check.

Note It is preferable to use the SII interface to send OBFF messages.
The bridge AXI subordinate can also support message transmission of OBFF messages without any modifications. A full set of limitations for the AXI bridge is available at "AXI Bridge Limitations" on page 398.

3.8.1.4 OBFF Wake Signal Encoder Reference Design

An optional example WAKE Signal Encoder design, that you can integrate into your application logic, is provided at `<workspace>/examples/obff_wake_ref*`. It translates OBFF requests (from the same source that is driving the controller inputs `app_obff_obff_msg_req`, `app_obff_cpu_active_req`, and `app_obff_idle_msg_req`) to WAKE# patterns

Note You must verify the integration of the example encoder/decoder in your design.

Figure 3-76 Encoder Example Design in (DSP)

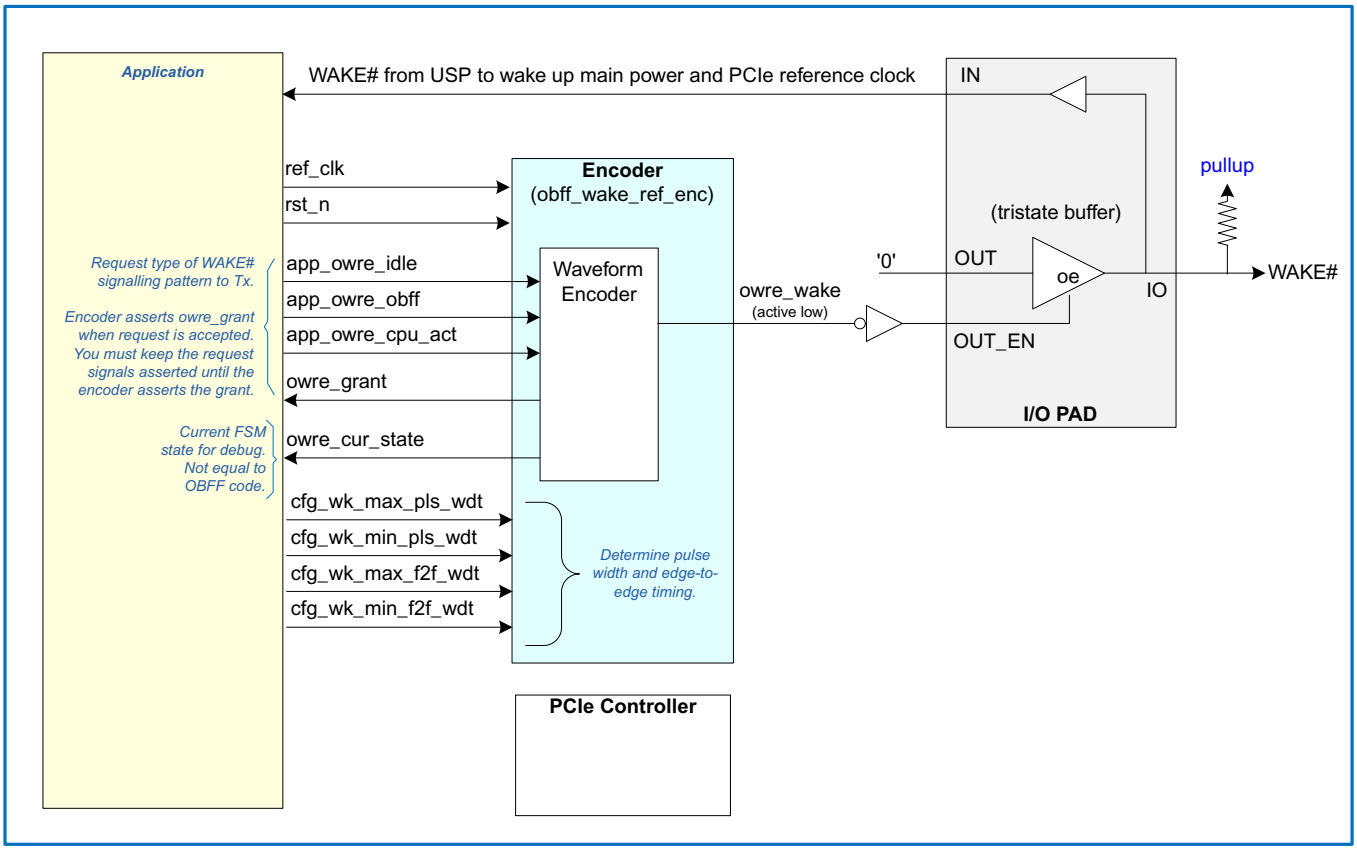
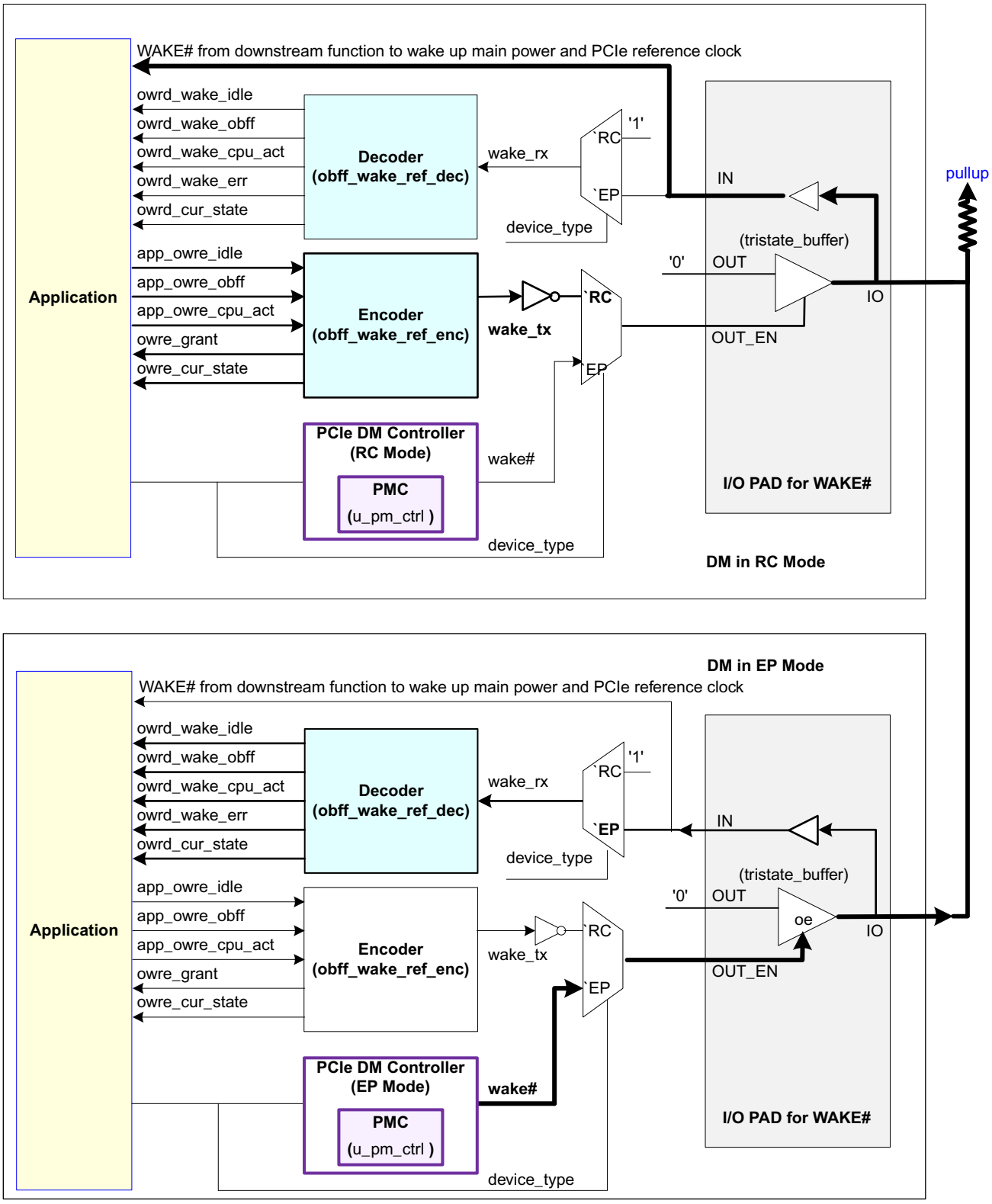


Table 3-59 OBFF WAKE# AC Specification (Table 2.6.2 of PCIe Specification)

Symbol	Parameter	Min	Max	Unit	Relationship to Timing Inputs
T_{WKRF}	WAKE# rise-fall time		100	ns	NA
$T_{WAKE\text{-}TX\text{-}MIN\text{-}PULSE}$	Minimum WAKE# pulse width; applies to both active-inactive-active and inactive-active-inactive cases.	300		ns	cfg_wk_min_pls_wdt * T_{CK_PERIOD}[a]
N/A	Maximum WAKE# pulse width; applies to both active-inactive-active and inactive-active-inactive Set to less than 50% of {cfg_wk_max_f2f_wdt * T_{CK_PERIOD} }		500	ns	cfg_wk_max_pls_wdt * T_{CK_PERIOD}
$T_{WAKE\text{-}FALL\text{-}FALL\text{-}CPU\text{-}ACTIVE}$	Time between two falling WAKE# edges when signalling CPU Active.	700	1000	ns	min = cfg_wk_min_f2f_wdt * T_{CK_PERIOD} max = cfg_wk_max_f2f_wdt * T_{CK_PERIOD}

a. Period of the clock this is running to detect wakeup events.

Figure 3-77 Wake Event from EP to RP (Unrelated to OBFF Signaling)



3.8.1.5 Application Msg/MsgD Programming Examples

The following tables enumerate the different ways that your application can generate Msg and MsgD TLPs.

Table 3-60 Msg Without Payload Transmission Methods

Application Interface	Description	Application I/O Signals
AXI	Direct Supply using a Msg transaction.	slv_awmisc_info[4:0] =MSG slv_wstrb[3:0] =0000 slv_awmisc_info[20:13] =message code slv_awmisc_info_p_tag[5:0] =tag slv_awmisc_info_hdr_34dw[63:32] =3rd TLP header DWORD slv_awmisc_info_hdr_34dw[31:0] =4th TLP header DWORD slv_awmisc_info_atu_bypass =1
AXI	Indirect Supply (iATU) using a MWr[a] transaction. The iATU[a] needs to be configured to translate MWr to Msg TLPs.	slv_awmisc_info[4:0] =MEM slv_wstrb[3:0] =0000 slv_awmisc_info_hdr_34dw[63:32] =3rd TLP header DWORD slv_awmisc_info_hdr_34dw[31:0] =4thTLP header DWORD
XALI0/1/2	Direct Supply using a Msg transaction.	client1_tlp_type =MSG client1_tlp_fmt[1:0] =01 client1_tlp_byte_len =0 client1_tlp_byte_en =message Code client1_tlp_addr[31:0] =header bytes 12-15 client1_tlp_addr[63:32] =header bytes 8-11
XALI0/1/2	Indirect Supply (iATU) using an MWr[a] transaction. The iATU needs to be configured to translate MWr to Msg TLPs. A MWr with an “effective length of 0”[b] is converted to Msg and all otherMWrTLPs are converted to MsgD.	client1_tlp_type =MSG client1_tlp_byte_len =0 client1_tlp_byte_en =00000000
VMI	The controller generates VendorDefinedmessages in response to requests on the VMI. For more information, see “Vendor Message Interface (VMI)” on page 381.	

a. When the “payload inhibit” feature of the iATU is used, then slv_wstrb does not need to be set. Similarly, for iATU header and tag substitution; the iATU can be set up to fill these fields in the outgoing TLP.
b. For AXI, the determination of effective length takes the first and last byte enables into account using slv_wstrb for AXI. When not using the AXI bridge, an effective length of “0” is determined by looking for a “0” at client_tlp_fmt[1]. When you are only translating the address of MSG TLPs then use client0_tlp_byte_en to provide the message code.

Table 3-61 Msg With Payload Transmission Methods

Application Interface	Description	Application I/O Signals
AXI	Direct supply using an MsgD transaction.	slv_amisc_info[4:0] =MSG slv_awmisc_info[20:13] = message code slv_awmisc_info_p_tag[5:0] =tag slv_awmisc_info_hdr_34dw[63:32] =3rd TLP header DWORD slv_awmisc_info_hdr_34dw[31:0] =4th TLP header DWORD slv_awmisc_info_atu_bypass =1 slv_awaddr[1:0] =2'b00
AXI	Indirect Supply (iATU) using a MWr[a] transaction. The iATU[a] needs to be configured to translate MWr to MsgD TLPs.	slv_amisc_info[4:0] =MEM slv_awmisc_info_hdr_34dw[63:32] =3rd TLP header DWORD slv_awmisc_info_hdr_34dw[31:0] =4th TLP header DWORD slv_awaddr[1:0] =2’b00
XALI0/1/2	Direct Supply using a Msg transaction.	client1_tlp_type =MSGD client1_tlp_fmt[1:0] =11 client1_tlp_byte_len !=0 client1_tlp_byte_en =message Code client1_tlp_addr[31:0] =header bytes 12-15 client1_tlp_addr[63:32] =header bytes 8-11
XALI0/1/2	Indirect Supply (iATU) using a MWr transaction The iATU needs to be configured to translate MWr to MsgD TLPs.	client1_tlp_type =MEM client1_tlp_byte_len !=0
VMI	MsgD is not supported.	

a. When the “payload inhibit” feature of the iATU is used, then slv_wstrb does not need to be set. Similarly, for iATU header and tag substitution; the iATU can be set up to fill these fields in the outgoing TLP.

3.8.1.6 Byte Mapping of Third and Fourth Message Header DWORDs at I/O Interfaces

The following figure indicates the byte mapping between the message generation interfaces and the third and fourth DWORDs of the message TLP header. For more information, see:

- Descriptions of `client0_tlp_addr` and `client0_tlp_data` at “Transmit Interfaces (XALI0/1/2)” on page 356.
- Description of `ven_msg_data` at “Vendor Message Interface (VMI)” on page 381.
- “Advanced Information: Endianness” on page 747.

Figure 3-78 Transmitted 3rd and 4th DWORD Message Header Byte Mapping at Interfaces

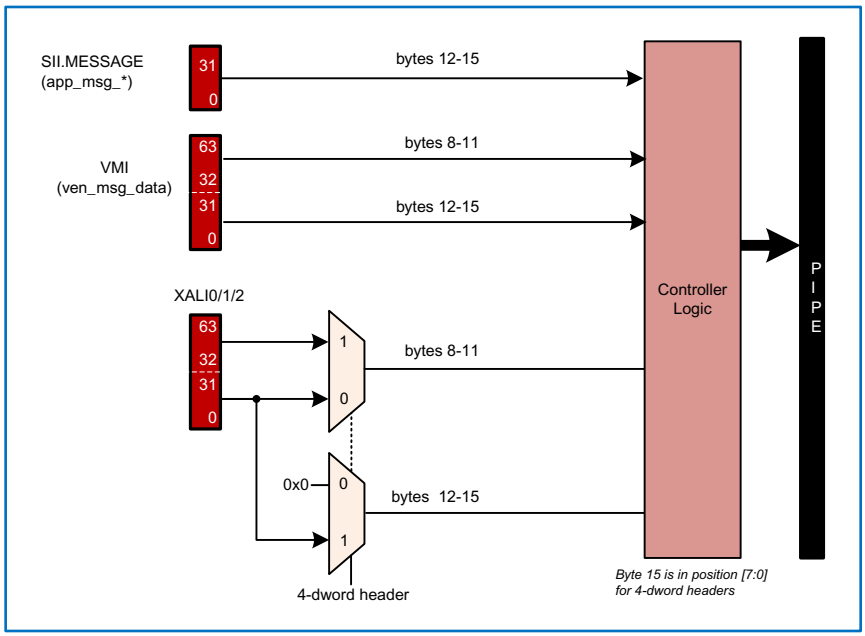
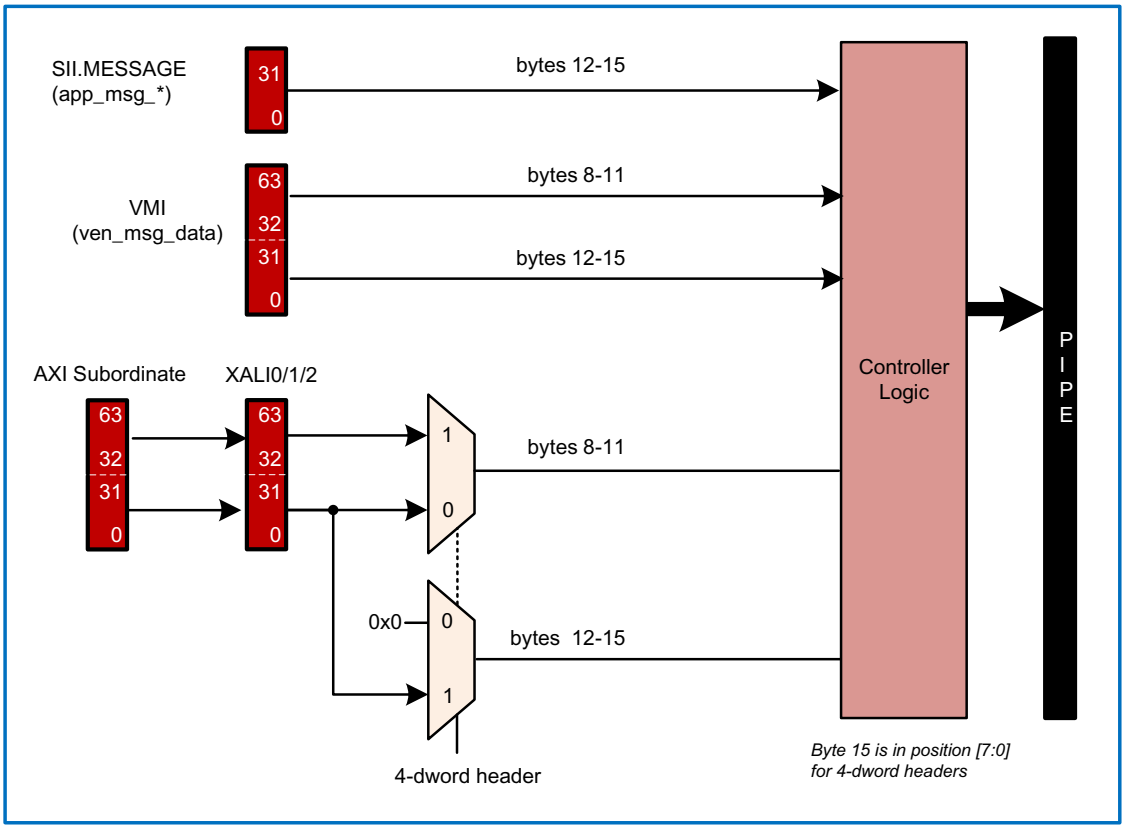


Figure 3-79 Transmitted 3rd and 4th DWORD Message Header Byte Mapping at Interfaces (with AXI Bridge)



Note	In 32-bit address AXI configurations; the controller transmits all '0's instead of slv_awmisc_info_hdr_34dw[63:32].

Caution	You must ensure that messages generated by your application do not trigger decomposition as described in “AXI Decomposition Rules” on page 435.

3.8.2 Message Reception

The following topics are discussed in this section:

- "Overview"
- "Message Reception I/O Interfaces" on page 207
- "Routing of Received Messages" on page 208
- "Latency Tolerance Reporting (LTR) Message Reception" on page 208
- "Further Information" on page 209

3.8.2.1 Overview

The PCI Express controller can receive the following types of messages. The index in the first column refers to the circled numbers in the following diagrams.

Table 3-62 Types of Received Messages

Index	Message Source (Type)	Details
1	Power Management (Msg)	For more information, see "Power Management" on page 259.
2	Error Signaling from Downstream Component (Msg)	COR_ERR / ERR_NONFATAL / ERR_FATAL
3	Vendor Defined (Msg/MsgD)	For more information, see "Routing of Received Messages" on page 208.
5	Legacy PCI Interrupts from Downstream Devices (Msg)	
6	LTR Request and Clear (Msg)	For more information, see "Latency Tolerance Reporting (LTR) Message Reception" on page 208.
9	ATS Request (Msg) ATS Invalidation Completion (Msg)	For more information, see "Address Translation Services (ATS)" on page 248.
10	DRS/FRS (Msg)	For more information, see "Readiness Notifications (RN)" on page 322.
11	PTM Request (Msg)	For more information, see "Precision Time Measurement (PTM)" on page 325.

Figure 3-80 Message Reception

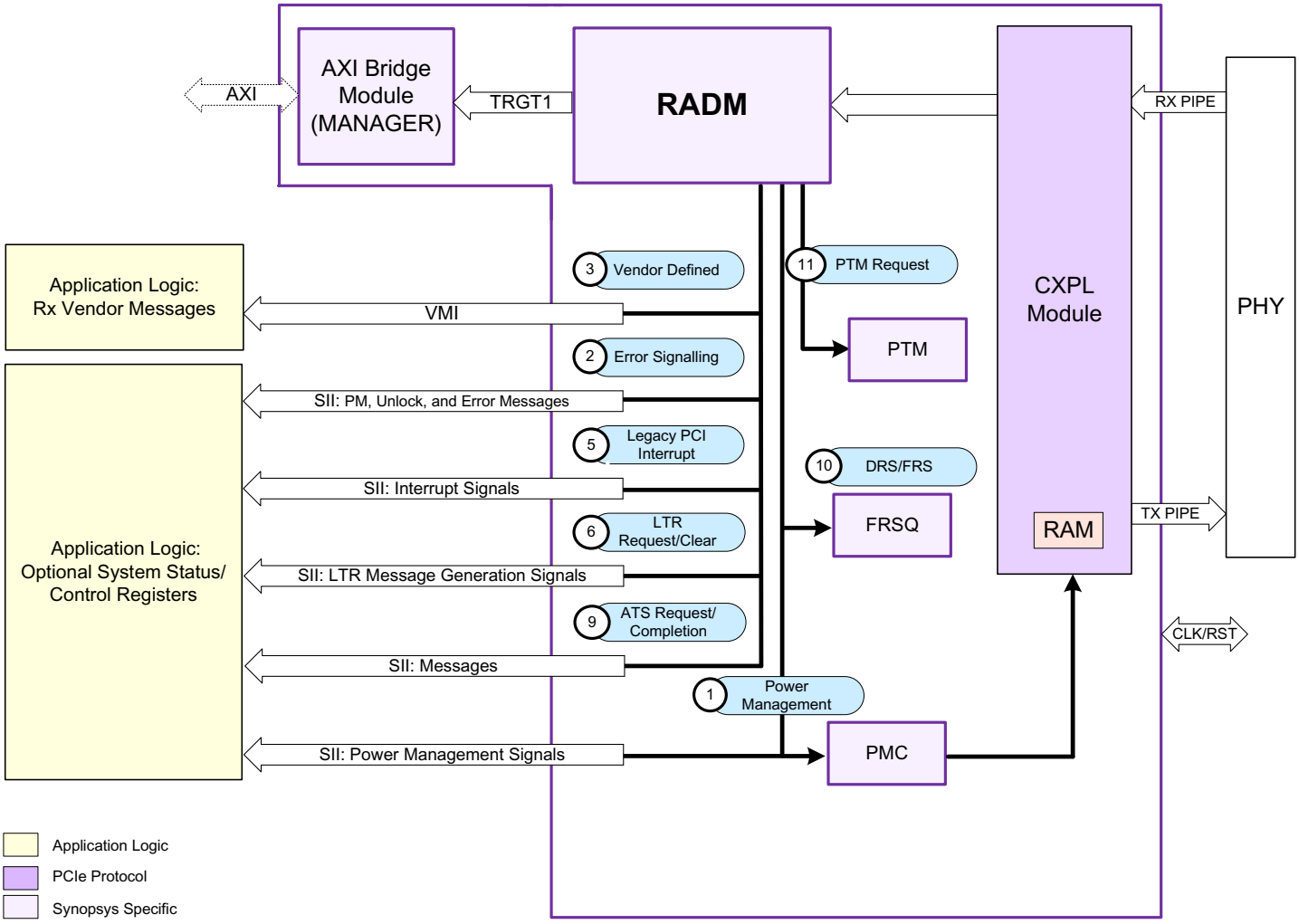
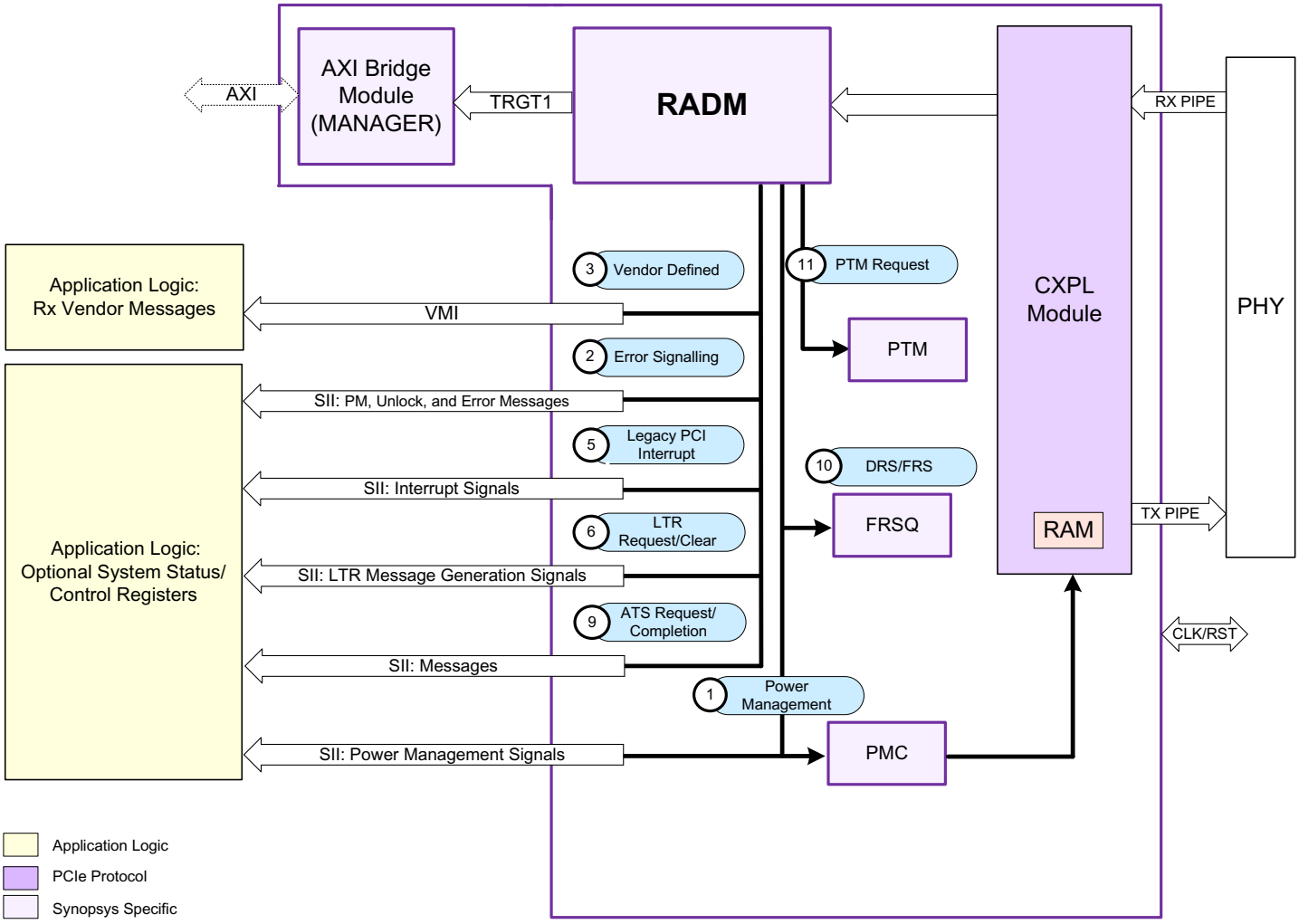


Figure 3-81 Message Reception: RP Mode



For definitions of acronyms used for block and interface names, see "Terms and Abbreviations" on page 22.

3.8.2.2 Message Reception I/O Interfaces

The RADM filter provides a message interface that is grouped as part of the System Information Interface (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 your application logic on the SII. Some of the message reception signals are also used in "Interrupts" on page 210 and "Reliability, Availability, and Serviceability (RAS)" on page 162.

3.8.2.2.1 Slot Power Limit Indication

The application can identify when the controller

- receives a Set_Slot_Power_Limit and
- updates the `PCIE_CAP_SLOT_POWER_LIMIT_VALUE` field of the `SLOT_CAPABILITIES_REG` register,

using the `radm_slot_pwr_limit` output signal from the controller. The application can collect the payload of the Set_Slot_Power_Limit message through `radm_slot_pwr_payload` output signal.

3.8.2.3 Routing of Received Messages

All[1] error-free MSG requests are decoded internally, signaled on the SII interface, and then dropped (not forwarded to your application[2] on TRGT1 (or AXI bridge Master when present). When a MSG request is filtered with UR/CA/CRS status, the TLP is always dropped.

Note

Summary of Message Delivery

- By default received messages[a] (Msg/MsgD) are delivered (without the payload for MsgD) on the SII interface and are not[b] delivered on the TRGT1 (or AXI bridge master) interface. If you also want the message to be delivered on the TRGT1 (or AXI bridge Master) interface, then you must clear the corresponding filter mask bit.
- The SII message reception interface provides[c] the requester ID and message type (from the first and second TLP DWORDs) and the contents of the third and fourth TLP DWORDs.

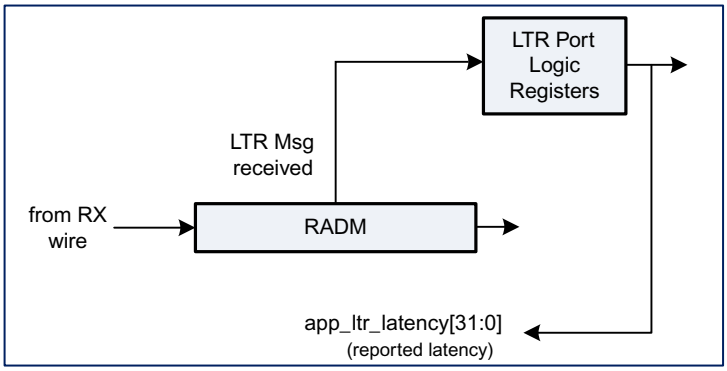
For more information, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.

a. Except PTM and DRS/FRS
b. The exception to this is an ATS Invalidation Request (in EP mode) which is too big and is delivered on the TRGT1 interface regardless of filter mask settings
c. Except PTM and DRS/FRS

3.8.2.4 Latency Tolerance Reporting (LTR) Message Reception

You can enable LTR message reception by setting `CX_LTR_M_ENABLE`. When the controller receives an LTR message, it pulses the `radm_msg_ltr` output. You can access the LTR message information on the radm_msg_payload output, which is part of the SII message interface. The actual values in the LTR message received by this Downstream Port are captured in the port logic LTR Latency Register (`PL_LTR_LATENCY_OFF`). They are also reflected on the `app_ltr_latency[31:0]`output.

Figure 3-82 LTR Registers



When you configure the controller with a 256-bit datapath, and when two LTR messages are received in the same clock cycle (back-to-back), the earlier message is discarded.

1. Except PTM and DRS/FRS.
2. Vendor TYPE0 messages generate an UR error.

Note It is preferable to use the SII interface to receive LTR messages. The AXI manager cannot support reception of LTR messages.
A full set of limitations for the AXI bridge is available at "AXI Bridge Limitations" on page 398.

3.8.2.5 Further Information

- For specific information on how the handling of a message is affected by its error status, see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715 and"Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.
- For more information on optional address translation of VDMs, see "MSG/MSGD Match Mode" on page 239.
- For more information on ordering considerations, see "Advanced Information: Advanced Ordering Information" on page 703.

3.9 Interrupts

The following section describes the processing of interrupts in the controller. You should be familiar with the different types of interrupts as specified in Sections 6.1.1, “Rationale for PCI Express Interrupt Model”, 6.1.4, “Message Signaled Interrupt (MSI/MSI-X) Support”, 6.1.2, “PCI Compatible INTx Emulation”, and 6.1.3, “INTx Emulation Software Model” of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

- “Interrupt Reception (DSP)” on page 211

3.9.1 Interrupt Reception (DSP)

The following topics are discussed in this section:

- "PCI Legacy Interrupts"
- "PCI Express Hot-Plug Logic Interrupt and Wakeup" on page 211
- "iMSI-RX: Integrated MSI Receiver [AXI Bridge]" on page 211

3.9.1.1 PCI Legacy Interrupts

The controller decodes received Assert_INTx/Deassert_INTx messages and pulses the `radm_inti_asserted` and `radm_inti_deasserted` outputs where i=a,b,c,d.

3.9.1.2 PCI Express Hot-Plug Logic Interrupt and Wakeup

When MSI or MSI-X is enabled, the controller asserts the `hp_msi` output upon detection of any of these hot plug events:

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

When INTx interrupt mode is enabled, the controller asserts the `hp_int` output upon detection of any of the hot-plug events. When PME mode is enabled, the controller asserts the `hp_pme` output upon detection of any of the hot-plug events. The controller does not check if the PM state is D1, D2, or D3hot. Your application must check the value on `pm_dstate` to make sure the device is in D1, D2, or D3hot upon receipt of a `hp_pme` notification.

3.9.1.3 iMSI-RX: Integrated MSI Receiver [AXI Bridge]

The AXI bridge provides an integrated MSI reception module to detect and terminate inbound MSI requests (received on the RX wire). It is enabled by setting `CX_MSI_CTRL_ENABLE` =1. Rather than propagating MSI MWr TLPs onto the AXI bus through the master interface, the MSI packets are captured, terminated in the AXI bridge, and an interrupt is signaled locally through the `msi_ctrl_int` output.

The iMSI-RX is programmed with an address (`MSI_CTRL_ADDR_OFF` and `MSI_CTRL_UPPER_ADDR_OFF`) that is used as the system MSI address. When an inbound MWr request is passed to the AXI bridge and matches this address as well as the conditions specified for an MSI memory write request, an MSI interrupt is detected. When this MWr is about to be driven onto the AXI bridge master interface[1], it is dropped and never appears on the AXI bus.

1. To guarantee Posted requested ordering.

Caution

- The MSI interrupt is generated on `msi_ctrl_int` afterthepreviousPosted transaction's AXI B response is received. Itis then synchronized with `core_clk`. Therefore, the subsequentNP write transaction might appear sooner than MSI interrupt in theAXI `mstr_aclk` domain.
- If you want poisoned MSI MemWr TLP be forwarded to your application over AXI bridge, do not use iMSI-RX module.

The iMSI-RX decodes the MSI MWr data payload to determine which Endpoint device (EP) sent the MSI and which interrupt vector the MSI corresponds to. When a valid interrupt has been decoded, the `msi_ctrl_int` output is asserted. This output remains asserted when any MSI interrupt is pending. It is only de-asserted when there is no MSI interrupt pending.

Features

- Provides support for up to eight EPs. Each supported EP has a set of interrupt enable, mask, and status registers (`MSI_CTRL_INT_*`).
- Guarantees correct AXI ordering with respect to other inbound posted writes by generating the MSI interrupt only after your application AXI subordinate acknowledges responses of previous posted TLPs.
- A maximum of 32 interrupts are supported per EP.
- Optional 32-bit register driven general purpose outputs (`msi_ctrl_io[31:0]`). For more information, see the `MSI_GPIO_IO_OFF` register.

MSI Request Detection Criteria

An MSI interrupt request is detected when the controller receives a MWr TLP that satisfies these conditions:

- Header attributes bits are 0x0. No snoop (NS) and relaxed ordering (RO) fields must be 0x0.
- Length field is 0x01 to indicate a payload of one DWORD.
- First byte enable (FBE) is 4bxx11 (enabling the first two bytes of the payload.)
- Last byte enable (LBE) is 4b0000.
- TLP address corresponds to systems chosen MSI address as programmed in the `MSI_CTRL_ADDR_OFF` and `MSI_CTRL_UPPER_ADDR_OFF` port logic registers. This register is not the MSI Lower 32 Bits Address Register which is part of the PCIe MSI capability register.

In addition to these criteria the MWr must also pass the receive filtering rules as outlined in "Receive Filtering" on page 134.

Programming and Usage Model

- The host CPU configures the MSI capabilities of all endpoints through CFG requests.
  - Program the MSI Data Register (`MSI_CAP_OFF_08H_REG` or `MSI_CAP_OFF_0CH_REG` in the MSI capability) as follows to allow the iMSI-RX to decode the interrupt source.

Table 3-63 MSI Data Register (MSI_CAP_OFF_08H_REG or MSI_CAP_OFF_0CH_REG) Programming

15:8	7:5	4:0
Not Used	EP Number Allows each EP to be identified within the system, For example, EP#5 is programmed with 3b101	Interrupt Vector Number ■ Identifies the interrupt source within each EP ■ Programmed to 5b00000. ■ Set by MSI generation logic to identify each interrupt source in real time ■ Supports up to 32 Vectors

- Program the MSI Capability Lower 32 Bits Address Register (`MSI_CAP_OFF_04H_REG`) of every EP with one common MSI address.
- The host CPU configures the iMSI-RX through the local DBI bus.
  - Program the iMSI-RX's `MSI_CTRL_ADDR_OFF` and `MSI_CTRL_UPPER_ADDR_OFF` registers with the same value as in `MSI_CAP_OFF_04H_REG`.
  - Read the MSI capability of each EP to determine the number of vectors enabled in each EP and uses this information to program the iMSI-RX's interrupt enable register (`MSI_CTRL_INT_0_EN_OFF`). This register allows up to 32 MSI interrupt vectors to be enabled within the iMSI-RX for a given EP. It is the responsibility of the host CPU to read the contents of the Multiple Message Enable field in an EPs MSI capability structure and program that EPs Interrupt Enable register in the iMSI-RX appropriately. For example when the Multiple Message Enable is 3b100 for Endpoint #0 (which corresponds to 16 enabled interrupt vectors), the host CPU should program `MSI_CTRL_INT_0_EN_OFF` with 0x0000FFFF.

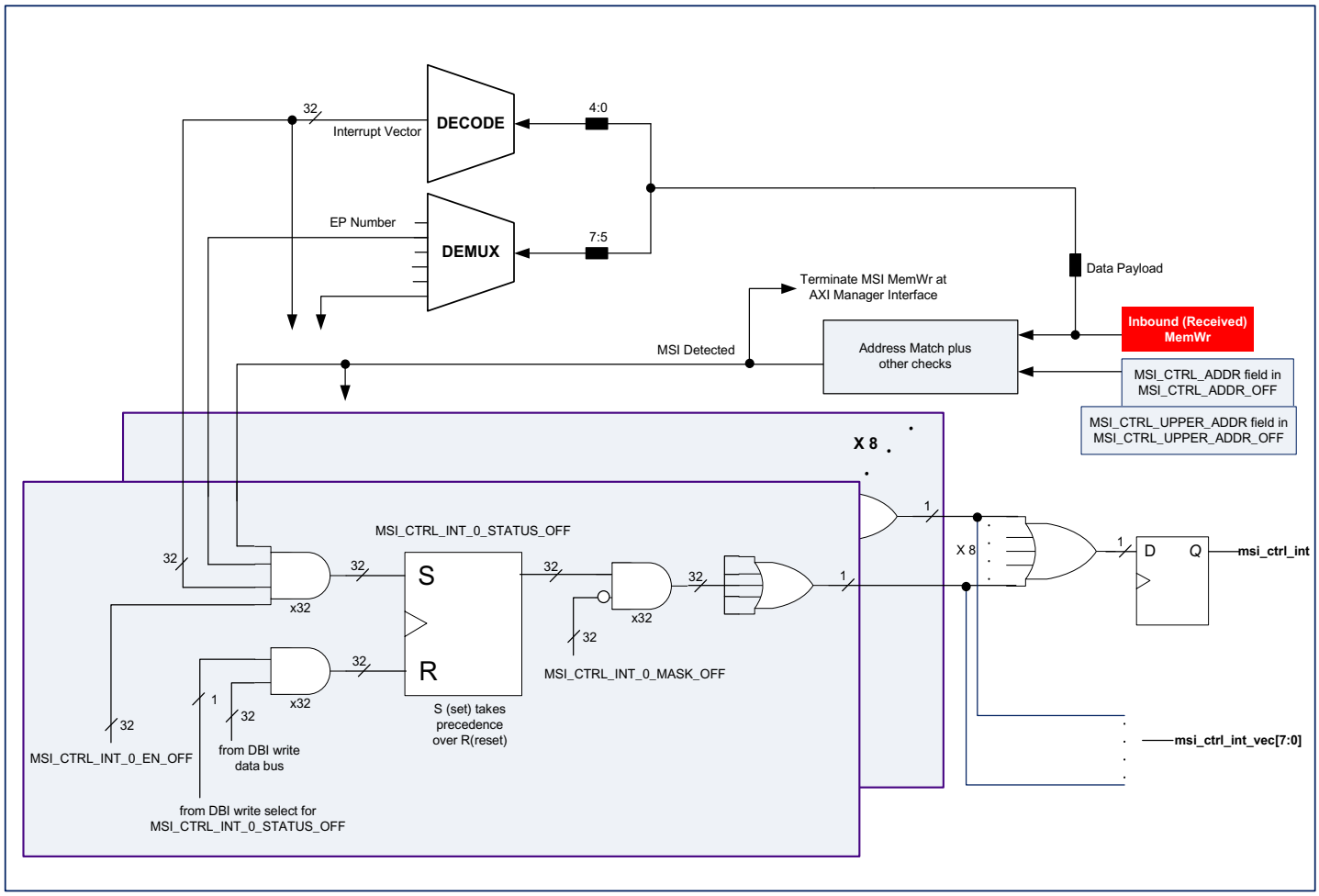
Hint

- When you include the iMSI-RX in your controller (by setting CX_MSI_CTRL_ENABLE =1), it is active and terminates all received MSI MemWr unless you deactivate it. The iMSI-RX is deactivated when all of the eight MSI_CTRL_INT_0_EN_OFF.... MSI_CTRL_INT_7_EN_OFF registers have a value of 0x0.

Processing of Detected Interrupts

The MSI controller decodes the MSI MWr data payload to determine which EP sent the MSI and to which interrupt vector it corresponds.

Figure 3-83 Architectural Representation: iMSI-RX Interrupt Detection Process



- When the decoded interrupt vector is enabled and not masked, then the controller sets the corresponding bit in the iMSI-RX Interrupt#0 Status Register (`MSI_CTRL_INT_0_STATUS_OFF`) and asserts the top-level controller output `msi_ctrl_int`. This signal remains asserted until the host CPU clears the status bit by writing a 1 to the status bit. Writing a 0 has no effect.
- When any status bit remains set, then `msi_ctrl_int` remains asserted. The interrupt status register provides a status bit for up to 32 interrupt vectors per Endpoint. When the decoded interrupt vector is enabled but is masked, then the controller sets the corresponding bit in interrupt status register but the it does not assert the top-level controller output `msi_ctrl_int`.
- When an MSI interrupt vector is received from an Endpoint but that vector has not been enabled in the corresponding iMSI-RX Interrupt#0 Enable Register (`MSI_CTRL_INT_0_EN_OFF`), then the controller does not set any bit in the interrupt status register and it does not assert `msi_ctrl_int`.
- In addition, when no interrupts have been enabled in any of the eight interrupt enable registers, then all MSI detection logic is disabled and valid MSI MWr request TLPs are not terminated in the bridge but are passed by the AXI manager interface to the AXI bus.
- The iMSI-RX Interrupt#0 Mask Register (`MSI_CTRL_INT_0_MASK_OFF`) allows the host to mask a given MSI interrupt vector. When a MSI interrupt vector is received for a masked interrupt vector, then the controller sets the corresponding bit in the interrupt status register but it does not assert `msi_ctr_intl` because the interrupt vector is masked.

- Note: This masking is local to the iMSI-RX and is not part of Per Vector Masking (PVM) in any of the downstream Endpoints. The contents of the interrupt mask and interrupt status registers are used to drive the `msi_ctrl_int` output.
- When any status bit is set and the interrupt vector is not masked, then the controller asserts `msi_ctrl_int`. As long as any interrupt status bit is set and not masked, `msi_ctrl_int` remains asserted.

Alternative Data Register Programming

The programming of the MSI Data Register (`MSI_CAP_OFF_08H_REG` or `MSI_CAP_OFF_0CH_REG`) as described in Table 3-63 assumes that each Endpoint enables 32 vectors. When one or more endpoints only support 16 or less interrupt vectors, then it is possible to share a single set of interrupt enable, interrupt mask, and interrupt status registers among those endpoints.

For example, assume that Endpoint #1 enables 16 interrupt vectors and Endpoint #2 also enables 16 interrupt vectors. When the MSI data register for Endpoint #1 is programmed to 0x20 and the MSI data register for Endpoint#2 is programmed to 0x30, as in Table 3-64, then the status for each Endpoint is in interrupt status register#1. Endpoint#1s vectors are in the lower 16 bits and Endpoint#2s vectors are in the upper 16 bits. Manipulating the MSI data register format in this manner allows the interrupt controller to support more than eight endpoints using the eight interrupt status registers provided.

Table 3-64 Alternative Data Register Programming

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Not Used								Reg N			EP#	Interrupt Vector Number			
Endpoint #1									0	0	1	0	0	0	0	0
Endpoint #2									0	0	1	1	0	0	0	0

3.10 Flow Control

The flow control mechanism is divided into two phases: initialization and update. The controller automatically performs both of these phases with minimal support required from your application. By default the RADM is responsible for returning flow control credits when it reads data out of the receive queues.

The controller provides optional I/O signals to enable your application to handle flow control returns in an application-specific manner. For error scenarios in bypass queue mode your application can decide whether to return credits or not based on state of the Bypass/RTRGT1 Interface signals described in Table 3-65.

Table 3-65 Bypass/RTRGT1 Interface Signals to Consider for Credit Return (Native PCIe controller)

Bypass/RTRGT1 Interface Signal	State	Credit Return Policy
radm_bypass_tlp_abort radm_trgt1_tlp_abort	Asserted	Credit needs to be returned
radm_bypass_dllp_abort radm_trgt1_dllp_abort	Asserted	Credit need not be returned
radm_bypass_ecrc_err radm_trgt1_ecrc_err	Asserted	Credit need not be returned. But, it is strongly recommended to return the credit if the TLP is not ambiguous.

For more information, see "System Information Interface (SII)" on page 382 and "Advanced Information: Advanced Error Handling for Received TLPs" on page 715. The controller does not return flow control credits for packets that have data link layer errors.

Calculation of Initial Credits and Receive Buffer Sizes

The default buffer sizes and credits for each TLP type are automatically calculated from the number of lanes, the controller datapath width, the maximum PCIe payload (`CX_MAX_MTU`), flow control update latencies, internal delays, and the PHY latency. For more information, see "Advanced Information: Flow Control Credit Calculation" on page 759.

You can determine the transmit posted, non-posted, and completion credits that are advertised by the receiver at the other end of the link by reading the following port logic registers:

- Transmit Posted FC Credit Status (`TX_P_FC_CREDIT_STATUS`)
- Transmit Non-Posted FC Credit Status (`TX_NP_FC_CREDIT_STATUS`)
- Transmit Completion FC Credit Status (`TX_CPL_FC_CREDIT_STATUS`)

Note You can override the default flow control update latency timer value using the FC Latency Timer Override Value field (TIMER_MOD_FLOW_CONTROL) in the Queue Status register (Q_STATUS).

3.10.1 Scaled Flow Control

The controller supports Scaled Flow Control (set the `DL_FEATURE_EN =1` parameter followed by `FC_SCALE_EN =1` parameter) as described in *PCI Express Base Specification, Revision 6.0, Version 1.0*. Scaled

flow control increases the number of credits prior to link-up by a scaling factor. Scaled flow control has the following features and limitations:

- This feature applies individually to all P, NP, or CPL TLPs.
- This feature applies to all VCs.
- The allocated initial credits scale down to 127 header credits and 2047 data credits, if the link partner does not support scaling.
- The default scaling factor depends on the number of lanes, maximum supported speed, delay factor, and latency factor.
- When scaling, the minimum number of header and data credits must be calculated as follows:
  - Minimum Number of Header Credits =32 * Scaling Factor, and
  - Minimum Number of Data Credits =512 * Scaling Factor
- At larger scaling factors, it is not possible to return single credits.

Note The default scaling factor of non-posted data is always 1.

- When scaling, the header and data credits range are as shown in Table 3-66

Table 3-66 Credits According to Scaling Factors

Scaling Factor	Header Scaling Credits	Data Scaling Credits
1	1-127	1-2047
4	128-508	2048-8188
16	509-2032	8189-32752

- When manually enabling Flow Control Scaling on a new configuration by editing the configuration file (without going through the coreConsultant flow):
  - If you are modifying the Header and Data credits, the scaling factors must be specified.
  - If the Header and Data credits have default values, there is no need to add the scaling factors.
- Due to a potential deadlock scenario, if an MRIOV_INIT is received during Data Link (DL) Feature setup, the link immediately exits DL Feature, and starts transmitting INITFCs.

  Note: Synopsys does not support, or recognize MRIOV DLLPs, outside of this scenario.
- For Gen4 and above, you can set `DL_FEATURE_EXCHANGE_EN` field of `DATA_LINK_FEATURE_-CAP_OFF` register to '0' to facilitate link up with legacy Endpoint devices that do not support Scaled Flow Control.

3.10.2 Shared Credits in Flit Mode

The controller supports shared credits in Flit Mode according to the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

- In Flit Mode, the controller advertises only not-merged shared credits for Posted, Non-Posted, and Completion FCs according to resources available in their respective buffers. Dedicated credits are not

used and Zero dedicated credits are advertised. As a consequence, the controller is not supposed to receive any TLP targeting dedicated FCs nor, more in general, any TLP including the Flit Mode Local TLP Prefix.

- In Flit Mode, the controller always consumes Shared Credits for transmitting a TLP if enough Shared Credits are available, otherwise it consumes Dedicated Credits and generates a Flit Mode Local TLP Prefix with the 'TLP uses Dedicated Credits' field Set.
- Shared Flow Control Usage Limit is not supported.
- The controller supports inbound Optimized Update FC, but it does not generate outbound Optimized Update FC.

3.11 Retry Mechanism

The controller supports a TLP-based Retry Mechanism when operating in Non-Flit Mode and a Flit-based Retry Mechanism when operating in Flit Mode according to the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

- The TX retry buffer RAM is reused for both Flit Mode and Non-Flit Mode.
- In Flit Mode:
  - Both Standard NAK and Selective NAK are supported.
  - The controller does not support RX retry buffer. As a consequence, Selective NAK requests are not sent.

3.12 Internal Address Translation Unit (iATU)

The controller uses the iATU to implement a local address translation scheme that replaces the TLP address and TLP header fields in the current TLP request header. The topics for this section are:

- "Configuring the iATU" on page 221
- "Outbound Features and Limitations" on page 222
- "Outbound Limitations" on page 225
- "Outbound Basic Operation (Address Match Mode)" on page 226
- "Outbound Detailed Operation" on page 229
- "Outbound Programming" on page 234
- "Inbound Features" on page 236
- "Inbound Limitations" on page 237
- "Inbound Basic Operation" on page 238
- "Inbound Detailed Operation" on page 241
- "Inbound Programming" on page 246

3.12.1 Configuring the iATU

The configuration parameters relevant to the iATU are described in Table 3-67.

Table 3-67 iATU Configuration Parameters

Parameter Name	Parameter Label	Value Range	Default
CX_INTERNAL_ATU_ENABLE	iATU Enable	1, 0	0
CX_ATU_NUM_OUTBOUND_REGIONS	Number of Outbound Translation Regions	256,255,.....3,2,1,0	2
CX_ATU_NUM_INBOUND_REGIONS	Number of Inbound Translation Regions	256,255,.....3,2,1,0	2
CX_ATU_MIN_REGION_SIZE	Minimum Size of Translation Region	4, 8, 16, 32, 64 kB	64 kB
CX_ATU_MAX_REGION_SIZE	Maximum Size of Translation Region	4 GB, 8 GB...16 EB	4 GB

Table 3-68 Other Configuration Parameters that Indirectly Affect iATU Operation

Parameter Name	Description
AMBA_INTERFACE !=0	AXI Bridge Enabled
CC_DMA_ENABLE =1	DMA Controller Enabled
CX_NFUNC >1	Multifunction
CX_SRIOV_ENABLE =1	SR-IOV Enabled

3.12.2 Outbound Features and Limitations

Address translation is used for mapping different address ranges to different memory spaces supported by your application. A typical example maps your application memory space to PCI memory space. The iATU also supports type translation. Without address translation, your application address is passed unmodified to the TLPs directly through the Tx application interface. You can program the iATU to implement your own outbound address translation scheme without external logic.

Outbound Features:

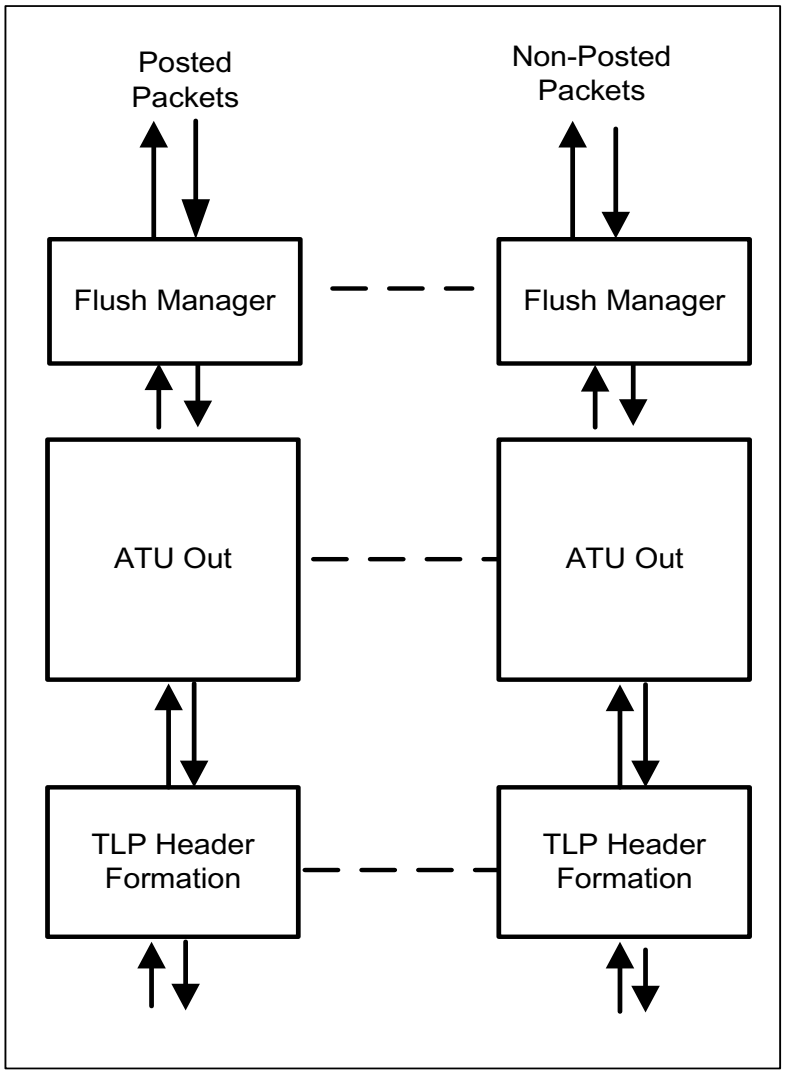
- Address Match mode operation for MEM and I/O, CFG, and MSG TLPs. No translation for completions.
- Supports type translation through TLP type header field replacement for MEM or I/O types to MSG/CFG types.
  - Includes posted to non-posted translation (for example, MWr to CfgWr0)
  - No translation from completions
- Programmable TLP header field replacement.
  - TYPE/TC/AT/ATTR/MSG-Code/TH/PH/ST
- Multiple (up to 256) address regions programmable for location and size.
- Programmable enable/disable per region.
- Automatic FMT field translation between three DWORDs and four DWORDs for 64-bit addresses.
- Invert Address Matching mode to translate accesses outside of a successful address match.
- Configuration Shift mode. Optimizes the memory footprint of CFG accesses destined for the Rx application interface in a multifunction device.
- Response code which defines the completion status to return for accesses matching a region.
- Supports regions from 4 KB to 16 EB in size.The minimum and maximum region sizes are determined by `CX_ATU_MIN_REGION_SIZE` and `CX_ATU_MAX_REGION_SIZE`. Smaller region sizes consume extra decode logic.

  For AXI, this feature is applicable only when `SLAVE_BUS_ADDR_WIDTH = 64`; If `SLAVE_BUS_ADDR_WIDTH < 64`, `CX_ATU_MAX_REGION_SIZE` is restricted to 4GB.
- Payload Inhibit marks all TLPs as having no payload data.
- Header Substitution replaces bytes 8 to 11 (for 3 DWORD header) or bytes 12 to 15 (for 4 DWORD header), inclusive, of the outbound TLP header.
- Tag Substitution of the outbound TLP tag field.
- Function number bypass mode to allow function number information to be supplied from your application transmit interface while translating the address and other attributes of the TLP.
- DMA bypass mode to allow TLPs which are initiated by the embedded DMA engine, to pass through the iATU untranslated.
- TLP Header fields bypass mode to allow header information to be supplied from your application transmit interface or, if AMBA is configured, from the AMBA sideband bus while translating the address and Type of the TLP.

Caution The default behavior of the ATU when there is no address match in the outbound direction or no TLP attribute match in the inbound direction, is to pass the transaction through.

In the Gen6 controller, type translation from MEM/IO to MSG is not supported due to the fact that Posted to Non-Posted or Non-Posted to Posted translations are not permitted.

Figure 3-84 Internal Address Translation Unit



Outbound Limitations:

In the Multistream Architecture, due to the fact the Outgoing ATU predicts the TLP_FMT, based on the TLP_TYPE, a situation can arise in FM (Flit-Mode) where the user attempts to convert a MemRd32 TLP to a TLP with TLP_TYPE = 0. The table underneath explains the values of the TLP_FMT/TLP_TYPE in both NFM/FM.

	NFM (FMT,Type)	FM (Type)
NOP	NA	0000_0000
MemRd32	000_00000	0000_0011
MemRd64	001_00000	0010_0000

If the PCIe Link is configured in FM, the user must ensure that in the case of an ATU address match, the TLP_TYPE to be converted must not be set to 00000 if not attempting to sent a TLP with Data (type[6] = 1) or a TLP with a 4-DW header (type[5] = 1). Otherwise a NOP TLP will be constructed. In this case, the System Software must be aware of the Link Mode and configure the PCIe CDM accordingly.

3.12.3 Outbound Limitations

- For configurations where there is no integrated AXI bridge present, it is required that your application provides additional tag management when you have programmed the iATU to do posted to non-posted type conversions (from MWr to CfgWr or IOWr). The tag field in the resulting CfgWr/IOWr TLP does not have a valid tag number because MWr had no tag. This can be overcome by providing a valid tag field on the XALI* client interface for the original MWr transaction and tracking/releasing that tag when the completion arrives. In addition, the transaction is issued on the PCIe wire and receives a completion in response as I/O and CFG writes are non-posted. Your application must accept this completion.
- Generation of ATS requests and messages not supported (no facility to insert NW and iTAG fields).

3.12.4 Outbound Basic Operation (Address Match Mode)

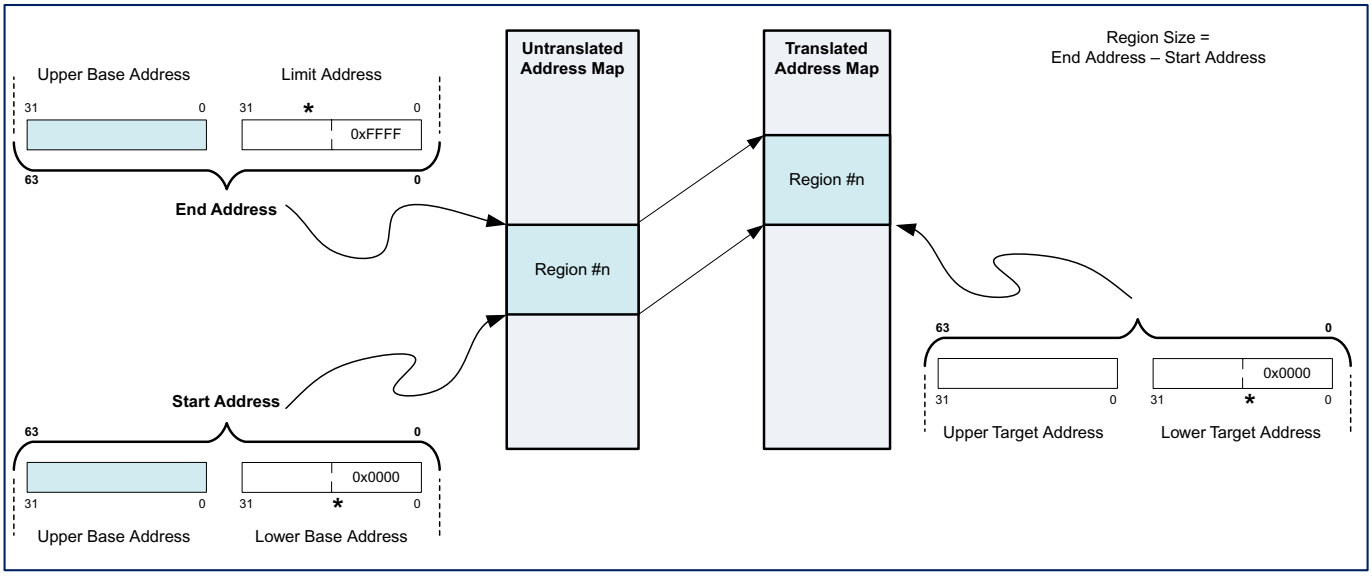
The address field of each request MEM and I/O TLP is checked to see if it falls into any of the enabled[1] address regions defined by the Start and End addresses as defined in Figure 3-87 on page 228. When an address match is found, then the TLP address field is modified as follows:

```
Translated Address = Original Address - Base Address + Target Address
```

and the TYPE, TC, AT, TH, PH, ST, Function Number, and ATTR header fields are replaced with the corresponding fields in the `IATU_REGION_CTRL_1_OFF_OUTBOUND_0` register. When your application (XALI0/1/2) address field matches more than one of the `CX_ATU_NUM_OUTBOUND_REGIONS` address regions, then the first (lowest of the numbers from 0 to `CX_ATU_NUM_OUTBOUND_REGIONS`-1) enabled region to be matched is used. For more information on what happens when there is no address match, see "No Address Match Result" on page 231. This operational mode (called Address Match Mode) is always used for outbound translation. Figure 3-85 to Figure 3-87 provides more information on this translation process.

Note The iATU does not translate DBI requests.

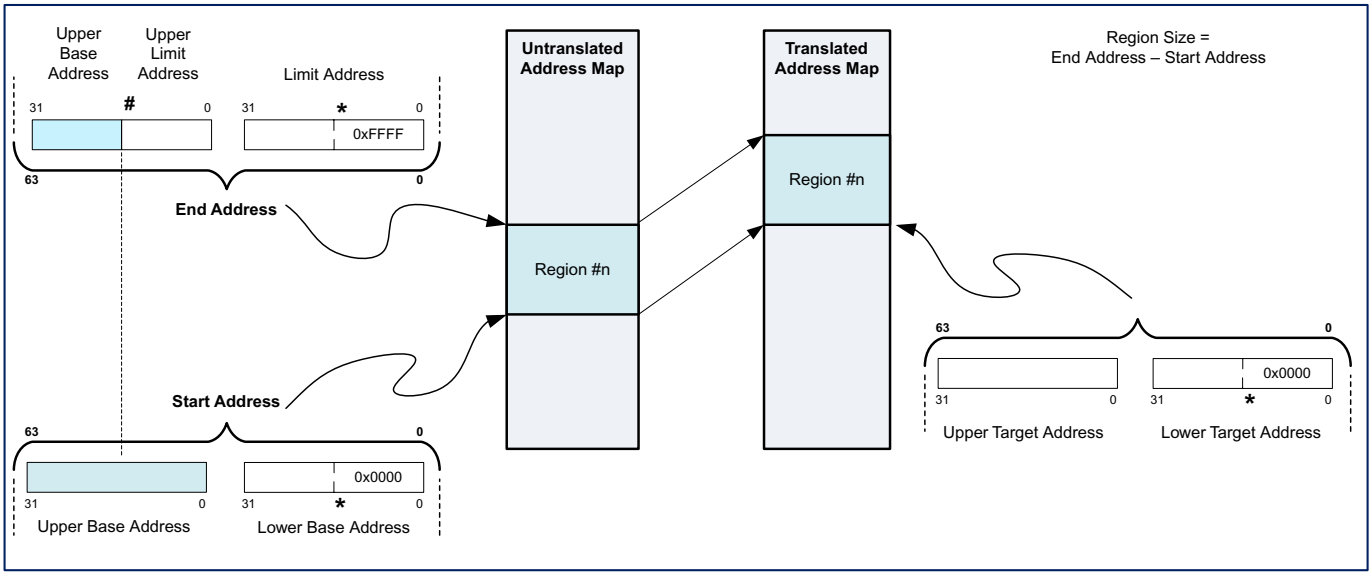
Figure 3-85 64-bit Region Mapping: Outbound and Inbound (Address Match Mode); INCREASE_REGION_SIZE=0



Note: * is $\log_2$(`CX_ATU_MIN_REGION_SIZE`)

1. When the Region Enable bit of the Region Control 2 register is '0', then that region is not used for address matching.

Figure 3-86 64-bit Region Mapping: Outbound and Inbound (Address Match Mode); INCREASE_REGION_SIZE=1



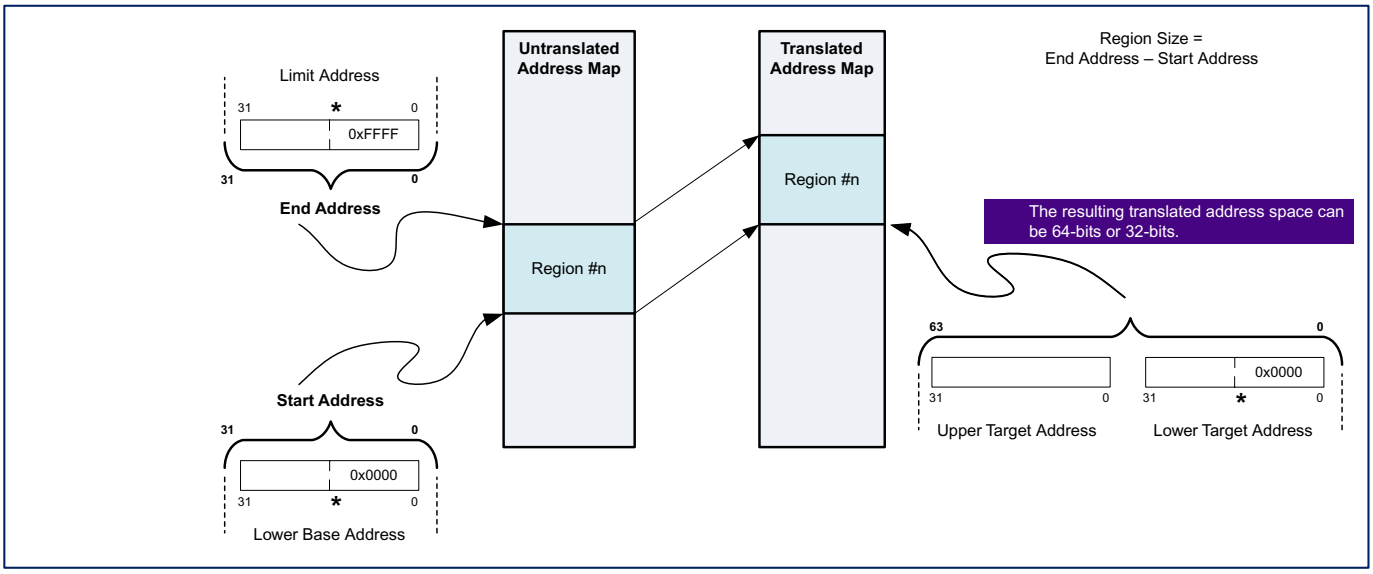
Note: * is $\log_2$(`CX_ATU_MIN_REGION_SIZE`)

The `CX_ATU_MIN_REGION_SIZE` configuration parameter (4 kB - 64 kB) specifies the minimum size of an address translation region. For example, when set to 64 kB, the lower 16 bits of the Base and Target registers are zero, the lower 16 bits of the Limit register is 0xFFFF, and all address regions are aligned on 64 kB boundaries. More precisely, the lower `log2(CX_ATU_MIN_REGION_SIZE)` bits are zero. In Figure 3-86, # is `CX_ATU_MAX_REGION_SIZE`; if =0, Upper Limit Address register does not exist.

The `INCREASE_REGION_SIZE` field in `IATU_REGION_CTRL_1_OFF_OUTBOUND_i` determines if the maximum region size is 4 GB or is set by the `CX_ATU_MAX_REGION_SIZE` configuration parameter. This only applies to 64-bit addressing.

Caution The combination of region size and number of regions must not consume the maximum space that is addressable with your 32-bit or 64-bit system address.

Figure 3-87 32-bit Region Mapping: Outbound and Inbound (Address Match Mode)



Note: * is $\log_2$(`CX_ATU_MIN_REGION_SIZE`)

3.12.5 Outbound Detailed Operation

The following advanced topics are discussed:

Default Operation

- "RID BDF Number Replacement"
- "IO/MEM to Msg/MsgD Type Translation" on page 229
- "IO/MEM to CFG Type Translation" on page 230
- "FMT Translation" on page 230
- "Writing to a MRdLk Region" on page 230
- "No Address Match Result" on page 231

Optional Features

- "CFG Shift" on page 231
- "Address Match Invert" on page 231
- "General Bypass" on page 231
- "DMA Bypass" on page 231
- "Function Number Translation Bypass" on page 232
- "TLP Header Field Translation Bypass" on page 232
- "Payload Inhibit" on page 232
- "Header Substitution" on page 233
- "Tag Substitution" on page 233

3.12.5.1 RID BDF Number Replacement

Overview: When there is a successful address match on an outbound TLP, then the function number used in generating the function part of the requester ID[1] field of the TLP is taken from the 3-bit Function Number field of the `IATU_REGION_CTRL_1_OFF_OUTBOUND_i` register. To override this behavior, use the "Function Number Translation Bypass" on page 232.

3.12.5.2 IO/MEM to Msg/MsgD Type Translation

Overview: The iATU supports TYPE translation/conversion of MEM and I/O TLPs to Msg/MsgD TLPs. When there is a successful address match on an outbound MEM TLP, and the translated TLP type field is MSG (that is, the type field of the `IATU_REGION_CTRL_1_OFF_OUTBOUND_i` register is 10xxx), then the message code field of the TLP is set to the value in the Message Code field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register. A MWr with an effective length[2 3] of '0' is converted to Msg and all other MWr TLPs are converted to MsgD. For more information on generating messages, see "Message Generation" on page 194.

1. Uses the 8:5:3 bit PCI Bus.Device.Function (BDF) format.
2. For AXI, the determination of effective length takes the first and last byte enables into account using slv_wstrb.
3. In the native PCIe controller (without AMBA bridge) configuration, an effective length of '0' is determined by looking for a '0' at client_tlp_fmt[1]. When you are only translating the address of MSG TLPs, use client0_tlp_byte_en to provide the message code.

In the Gen6 controller, Non-Posted to Posted (and Posted to Non-Posted) translation is not supported as there is a dedicated FC channel for each request type. It is not possible to convert between FC types on a dedicated channel.

Caution The iATU does not translate outbound messages.

Usage Scenario: This is useful for applications that are unable to generate Msg/MsgD TLPs.

3.12.5.3 IO/MEM to CFG Type Translation

Overview: The iATU supports translation of I/O and MEM TLPs to CFG TLPs. The 16-bit BDF is located at bits [31:16] of the translated address where:

```
Translated Address = Original Address - Base Address + Target Address
```

As an example

```
Original Address[31:16] = {13h0,function_no[2:0]}
Base Address[31:16] = 16h0
Target Address[31:16] = {bus_no[7:0],device_no[4:0],3h0}
```

then

```
Translated Address[31:16] = BDF ={bus_no[7:0],device_no[4:0],function_no[2:0]}
```

For CFG transactions created directly by your application (as opposed to the iATU), you must ensure that the BDF field does not match any programmed iATU address region or else unintentional type translation could occur.

Usage Scenario: This is useful for applications that are unable to generate CFG TLPs.

3.12.5.4 FMT Translation

Overview: The iATU automatically sets the TLP format field for three DWORDs when it detects all zeros in the upper 32 bits of the translated address. Otherwise, it sets it to four DWORDs when it detects a 64-bit address (that is, when there is a '1' in the upper 32 bits of the translated address). When the original address and the translated address are of different format, the iATU ensures that the TLP header size matches the translated address format.

3.12.5.5 Writing to a MRdLk Region

Overview: When there is a successful address match for an outbound write and the type header field in the `IATU_REGION_CTRL_1_OFF_OUTBOUND_i` register is "00001" indicating a locked MEM transfer, then the controller sets the type field to "0000" (MEM).

3.12.5.6 No Address Match Result

Overview: When there is no address match then the address is untranslated but the TLP header information (for fields that are programmable) comes from the relevant fields on the application transmit interface XALI*[1].

3.12.5.7 CFG Shift

Overview: The iATU uses bits [27:12] of the original address to form bits [31:16] (BDF location) of the outgoing CFG TLP. This supports the Enhanced Configuration Address Mapping (ECAM) mechanism (Section 7.2.2 of the *PCI Express Base Specification, Revision 6.0, Version 1.0*) by allowing all outgoing I/O and MEM TLPs (that have been translated to CFG) to be mapped from memory space into any 256 MB region of the PCIe configuration space.

Usage Scenario: This is useful for applications that implement Enhanced Configuration Address Mapping (ECAM) mechanism (Section 7.2.2 of the *PCI Express Base Specification, Revision 6.0, Version 1.0*).

Feature Configuration: To enable CFG shift feature, set the `CFG_SHIFT_MODE` field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_0` register to '1'.

3.12.5.8 Address Match Invert

Overview: In normal operation an address match on an outbound TLP occurs when the untranslated address is in the region bounded by the base address and limit address. When the invert feature is enabled, an address match occurs when the untranslated address is not in the region bounded by the base address and limit address.

Feature Configuration: To enable Invert feature, set the `INVERT_MODE` field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register to '1'.

3.12.5.9 General Bypass

Overview: When you set `client0_tlp_iatu_bypass`[2], the iATU does not process that transaction.

3.12.5.10 DMA Bypass

Overview: When you do not want the iATU to translate outbound requests that are generated by the DMA, you must implement one of the following approaches:

- Ensure that the combination of DMA channel address programming and iATU control register programming causes no translation of DMA traffic to be done in the iATU, or
- Enable the DMA bypass mode to allow request TLPs which are initiated by the embedded DMA controller to pass through the iATU untranslated

Feature Configuration: To enable DMA bypass, set the `DMA_BYPASS` field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register to '1'.

1. or AXI slave
2. or slv_*misc_info_atu_bypass (when AXI bridge is enabled)

3.12.5.11 Function Number Translation Bypass

Overview: When function number translation bypass is enabled and region address is matched, the function number of the translated TLP is taken from your application transmit interface and not from the Function Number field of the `IATU_REGION_CTRL_1_OFF_OUTBOUND_i` register .

Hint

You can directly map an AXI ID to a PCIe function number as follows:

- Concatenate the AXI ID onto the MSBs of the AXI slave address (you need to configure the controller for a 64-bit AXI address).
- Use these upper MSBs to match to a unique region, with each region being assigned to a different function number.

Usage Scenario: This is useful for AXI-based SR-IOV applications where the upper bits of the AXI ID are used to identify the source (PF or VF) of a request. These upper bits of the ID can be mapped to the function number of the slave request PF/VF `misc_info` buses of the controller. There is then no need for the iATU to translate the PF and VF information.

Feature Configuration: To enable function number translation bypass, set the `FUNC_BYPASS` field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register to ‘1’.

3.12.5.12 TLP Header Field Translation Bypass

Overview: When TLP header field bypass is enabled and region address is matched, header fields of the translated TLP are taken from your application transmit interface or, if AMBA is configured, from the AMBA sideband bus (`slv_awmisc_info`) and not from the corresponding fields of the `IATU_REGION_CTRL_1_OFF_OUTBOUND_i` or `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` registers. The header fields are TC, AT, PH, TH, ST, and Attr (IDO, RO and NS) fields.

Feature Configuration: To enable TLP header field bypass, set the `TLP_HEADER_FIELDS_BYPASS` field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register to '1'.

3.12.5.13 Payload Inhibit

Overview: When payload inhibit is enabled and region address is matched, the iATU converts transactions with data payload to TLPs without payload data by forcing the TLP header Fmt[1] bit =0, regardless of the application inputs [1].

Note: This feature is not supported for MemWr/IOWr with data payload greater than 1DW.

Caution

This feature must not be used for outbound transactions with data poisoning, because the EP bit is set for TLPs without data payload, this is a violation of data poisoning rules defined in section “2.7.2.2. Rules For Use of Data Poisoning” of the *PCI Express Base Specification*, *Revision 6.0*, *Version 1.0*.

1. such as slv_wstrb

Usage Scenario: This is useful for Vendor Defined Msg (without data); your application sends a MWr [1] but does not set the byte write enables to 0.

Feature Configuration: To enable payload inhibit, set the `INHIBIT_PAYLOAD` field of `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` to '1'.

3.12.5.14 Header Substitution

Overview: When header substitution is enabled and region address is matched, the iATU fully substitutes bytes 8-11 (for 3 DWORD header) or bytes 12-15 (for 4 DWORD header) of the outbound TLP header with the contents of the `LWR_TARGET_RW` field in `IATU_LWR_TARGET_ADDR_OFF_OUTBOUND_i`.

Note: For a Vendor Defined Msg/MsgD having 4 DWORD header, when `HEADER_SUBSTITUTE_EN =1`, the 4th DWORD (bytes 12-15) is replaced with the contents of `LWR_TARGET_RW` field in the `IATU_LWR_TARGET_ADDR_OFF_OUTBOUND_i` register. But regardless of the value of `HEADER_SUBSTITUTE_EN`, the 3rd DWORD is replaced with the contents of `UPPER_TARGET_RW` field in the `IATU_UPPER_TARGET_ADDR_OFF_OUTBOUND_i` register. So for a Vendor Defined Msg, BDF (or reserved) and Vendor ID must be programmed in `UPPER_TARGET_RW`.

Usage Scenario: This is useful for Vendor Defined Msg/MsgD and ATS transactions[2] which is normally inefficient requiring a very large iATU region.

Feature Configuration: To enable header substitution, set the `HEADER_SUBSTITUTE_EN` field in `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register to '1'.

3.12.5.15 Tag Substitution

Overview: When tag substitution is enabled and region address is matched, the iATU substitutes the `TAG` field of the outbound TLP header with the contents of the TAG field in `IATU_REGION_CTRL_2_OFF_OUTBOUND_i`.

Note: Your application must not attempt to perform TAG substitution for outgoing non-posted TLPs.

`CX_TPH_ENABLE =1`: TAG substitution for MWr does not occur because this field (byte 6) in the TLP header is the ST field. ST substitution can still take place using the `MSG_CODE` field in `IATU_REGION_CTRL_2_OFF_OUTBOUND_i`.

`CX_10BITS_TAG =1`: For 10-bit tags, TAG substitution only applies to the 8 least significant bits of the Tag field (bits 7:0 of Byte 4 of the TLP Header). T9 and T8 of the Header are not substituted.

Usage Scenario: This is useful for translation from AXI MWr to Vendor Defined Msg/MsgD.

Feature Configuration: To enable tag substitution, set the `TAG_SUBSTITUTE_EN` field of the `IATU_REGION_CTRL_2_OFF_OUTBOUND_i` register to '1'.

1. over the AXI interface
2. over the AXI interface

3.12.6 Outbound Programming

This section discusses the register programming interface and provides details of programming examples.

3.12.6.1 iATU Outbound Programming Overview

You can access the iATU registers through the DBI interface or through BAR Matched Mem/IO requests. The following registers are used for programming the iATU.

Table 3-69 iATU Register Map

Address	Label (i = 0 to CX_ATU_NUM_OUTBOUND_REGIONS-1)	Description
0x000	IATU_REGION_CTRL_1_OFF_OUTBOUND_*i*	Region Control 1
0x004	IATU_REGION_CTRL_2_OFF_OUTBOUND_*i*	Region Control 2
0x008	ATU_LWR_BASE_ADDR_OFF_OUTBOUND_*i*	Lower Base Address
0x00C	IATU_UPPER_BASE_ADDR_OFF_OUTBOUND_*i*	Upper Base Address
0x010	IATU_LIMIT_ADDR_OFF_OUTBOUND_*i*	Limit Address
0x014	IATU_LWR_TARGET_ADDR_OFF_OUTBOUND_*i*	Lower Target Address
0x018	IATU_UPPER_TARGET_ADDR_OFF_OUTBOUND_*i*	Upper Target Address
0x01C	IATU_REGION_CTRL_3_OFF_OUTBOUND_*i*	Region Control 3
0x020	IATU_UPPR_LIMIT_ADDR_OFF_OUTBOUND_*i*	Upper Limit Address

Table 3-70 iATU Register Address Bus Layout

Register Bit	31-20	19	18-17	16-9	8	7-2	1	0
Value	0	0	Reserved	Region	Inbound/Outbound	Register Address	0	1

Table 3-71 Example: Address Table for Accessing iATU Outbound Region 1 Configuration Registers

Address	Register Name	Register Bit Value							
		31-20	19	18-17	16-9	8	7-2	1	0
0x200	Region Control 1	0	0	0	1	000			
0x204	Region Control 2	0	0	0	1	004			
0x208	Lower Base Address	0	0	0	1	008			
0x20C	Upper Base Address	0	0	0	1	00C			
0x210	Limit Address	0	0	0	1	010			
0x214	Lower Target Address	0	0	0	1	014			
0x218	Upper Target Address	0	0	0	1	018			

Attention

For releases previous to 4.80a, the iATU registers have been programmed through an indirect addressing scheme using an index register (iATU_VIEWPORT_OFF), to reduce the address footprint in the PCI Express extended configuration space. To use the legacy method of accessing iATU registers, contact Synopsys support through SolvNetPlus.

Caution

You can inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the "Integrating the Controller and VIP using VTB" section in the "Integration" chapter of the *User Guide*.

Caution

Dynamic iATU Programming with AXI Bridge Module

You must not update the iATU registers while operations are in progress on the AXI bridge subordinate interface.

3.12.6.2 Outbound iATU Programming Example

Define Outbound Region 1 as:

64 kB I/O region from `0x80000000_d000000` to `0x80000000_d000ffff`, to be mapped to `0x00010000` in the PCIe I/O space.

1. Setup the Region Base and Limit Address Registers.

   Write `0xd0000000` to Address {`0x208`} to set the Lower Base Address.

   Write `0x80000000` to Address {`0x20C`} to set the Upper Base Address.

   Write `0xd000ffff` to Address {`0x210`} to set the Limit Address.
2. Setup the Target Address Registers.

   Write `0x00010000` to Address {`0x214`} to set the Lower Target Address.

   Write `0x00000000` to Address {`0x218`} to set the Upper Target Address.
3. Configure the region through the Region Control 1 Register.

   Write `0x00000002` to Address {`0x200`} to define the type of the region to be I/O.
4. Enable the region.

   Write `0x80000000` to Address {`0x204`} to enable the region.

3.12.7 Inbound Features

Address translation is used for mapping different address ranges to different memory spaces supported by your application. A typical example maps your application memory space to PCI memory space. The iATU supports type translation. Without address translation, your application address is passed from the TLPs directly through the application interface. You can program the iATU to implement your own inbound address translation scheme without external logic.

- Programmable Match mode operation for MEM, I/O, CFG, and MSG TLPs. No translation for completions.
- Selectable BAR Match mode operation for I/O and MEM TLPs.
  - TLPs that are not error-free (ECRC, malformed and so on) are not translated.[1]
- Programmable TLP header field matching.
  - TYPE/TD/TC/AT/ATTR/MSG-Code/TH/PH/ST
- Multiple address regions programmable for location and size.
- Programmable enable/disable per region.
- Automatic FMT field translation between three DWORDs and four DWORDs for 64-bit addresses.
- Invert Address Matching mode to translate accesses outside of a successful address match.
- ECAM Configuration Shift mode to allow a 256 MB CFG1 space to be located anywhere in the 64-bit address space.
- Supports regions from 4 KB to 16 EB in size.The minimum and maximum region sizes are determined by `CX_ATU_MIN_REGION_SIZE` and `CX_ATU_MAX_REGION_SIZE`. Smaller region sizes consume extra decode logic.

  For AXI, this feature is applicable only when `SLAVE_BUS_ADDR_WIDTH =64`; If `SLAVE_BUS_ADDR_WIDTH <64`, `CX_ATU_MAX_REGION_SIZE` is restricted to 4GB.
- Single Address Location to allow all TLPs to be translated to a single address location.
- Msg Type Match Mode to allow matching of any TLP of type Message.

Caution The default behavior of the ATU when there is no address match in the outbound direction or no TLP attribute match in the inbound direction, is to pass the transaction through.

1. If DEFAULT_TARGET =1, UR/CA TLPs which are BAR matched for an ATU region are forwarded to the TRGT1 interface and translated.

3.12.8 Inbound Limitations

- Generation of ATS requests and messages not supported (no facility to insert NW and iTAG fields).

3.12.9 Inbound Basic Operation

This section discusses how the iATU processes inbound requests. The topics are:

- "Overview"
- "MEM or I/O Match Modes" on page 238
- "CFG Match Mode" on page 239
- "MSG/MSGD Match Mode" on page 239

3.12.9.1 Overview

The following translation rules and limitations apply:

- When there is no match, then the address is untranslated. In addition,
  - TLPs that are not error-free (ECRC, malformed, and so on) are not translated.
- Address translation of completions is not supported in Address Match mode.

The setting of the `MATCH_MODE` field in `IATU_REGION_CTRL_2_OFF_INBOUND_i` determines how iATU inbound matching is done for each TLP type.

Table 3-72 Determination of Match Mode

TLP Type	MATCH_MODE =0	MATCH_MODE =1
MEM or I/O	Address Match Mode	Reserved
MSG/MSGD	Address Match Mode	Vendor ID Match Mode

Note The main difference between inbound and outbound iATU operation is that the TLP type is never changed in the inbound direction. Instead, the type field is used for more precise matching. Other fields can also be optionally used to further refine the matching process.

3.12.9.2 MEM or I/O Match Modes

Address Match Mode

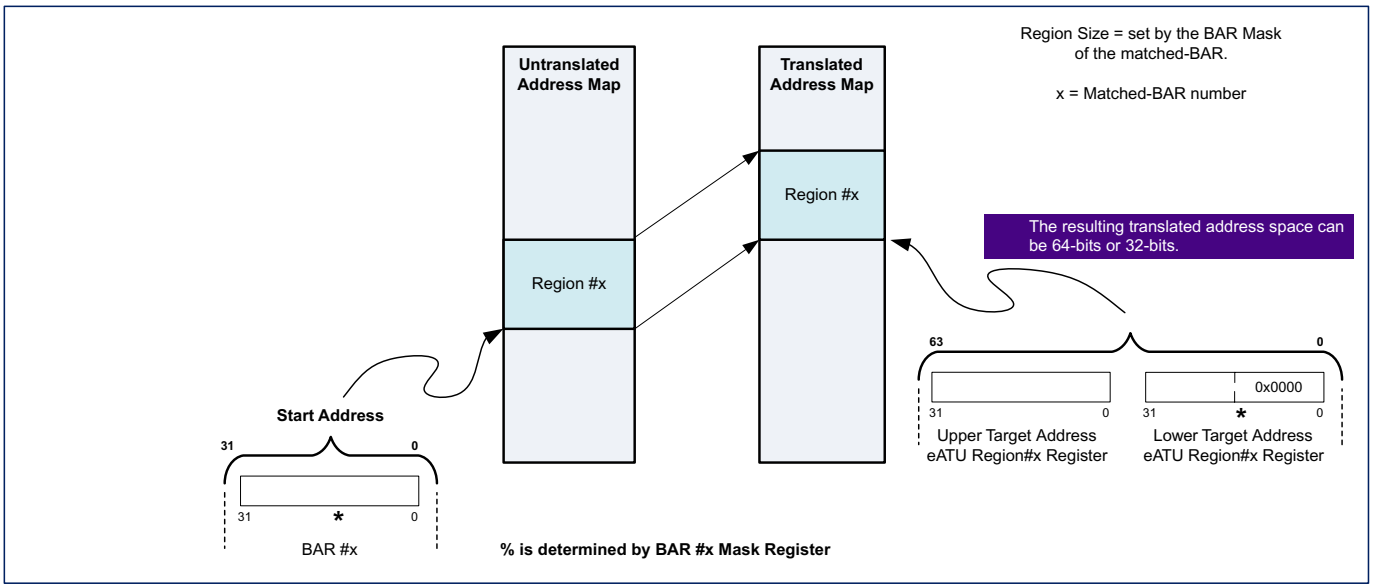
The operation is similar to "Outbound Basic Operation (Address Match Mode)" on page 226. The address field of each request TLP is checked to see if it falls into any of the enabled[1] address regions as shown in Figure 3-88. When an address match is found then the TLP address field is modified as follows:

```
Address = Address - Base Address + Target Address
```

When the TLP address field matches more than one of the `CX_ATU_NUM_INBOUND_REGIONS` address regions, then the first (lowest of the numbers from 0 to `CX_ATU_NUM_INBOUND_REGIONS` - 1)enabled region to be matched is used.

1. When the Region Enable bit of the Region Control 2 register is 0, then that region is not used for address matching.

Figure 3-88 iATU Address Region Mapping: Inbound (BAR Match Mode), 32-bit BAR



3.12.9.3 CFG Match Mode

The controller normally routes CFG TLPs (to the internal CDM or ELBI) without translation. The iATU only translates CFG0 TLPs that the controller has routed to the TRGT1.

Accept Mode

In Accept Mode, the controller always accepts CFG0 TLPs even when the CFG bus number does not match the current bus number of the device. The routing ID of received CFG0 TLPs are ignored when determining a match.

Routing ID Match Mode

The operation is similar to "Outbound Basic Operation (Address Match Mode)" on page 226. The routing ID of the inbound CFG0 TLP must fall within the Base and Limit of the defined iATU region for matching to proceed. The iATU interprets the routing ID (Bytes[1] 8-11 of TLP header) as an address. This corresponds to the upper 16 bits of the address in MEM and I/O transactions.

3.12.9.4 MSG/MSGD Match Mode

Address Match Mode

The third and fourth header DWORDs are treated as an address and are compared against the iATU Region Base and Limit Address registers. For vendor defined messages this allows specific messages to be filtered into memory at the target address. The Upper Base address should be set to Bus.Device.Function (BDF) and Vendor ID. The Lower Base address can be used as a filter for specific messages.

1. For more information, see Figure 2-18 Request Header Format for Configuration Transactions in Section 2.2.7 Memory, I/O, and Configuration Request Rules of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

Vendor ID Match Mode

This mode is relevant for ID-routed vendor defined messages. The iATU ignores the routing ID (BDF) in bits [31:16] of the third DWORD of the TLP header[1], but compares it against the vendor ID in bits [15:0] of the third DWORD of the TLP header (bytes[2] 10 and 11). This allows vendor defined messages to be filtered against specific vendor IDs without needing to know the BDF number which might vary depending on the PCI topology. Bits [15:0] of the Region Upper Base register should be programmed with the required vendor ID as follows:

```
Region Upper Base[15:8] =byte 10
Region Upper Base[7:0] =byte 11
```

The lower base and limit register should be programmed to translate TLPs based on vendor-specific information in the fourth DWORD of the TLP header.

Hint

For more information on generating messages, see "Message Generation" on page 194.

For a proper understanding of messages, you should be familiar with Section 2.2.8, Message Request Rules of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

1. In Figure 2-25 Header for Vendor_Defined Messages in Section 2.2.8.6, Vendor_Defined Message of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.
2. In Figure 2-25 Header for Vendor_Defined Messages in Section 2.2.8.6, Vendor_Defined Message of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

3.12.10 Inbound Detailed Operation

The following advanced topics are discussed:

Default Operation

- "FMT Translation"

Optional Features

- "Optional Matching Fields" on page 241
- "Single Address Location" on page 242
- "Response Code" on page 244
- "Message Type Match" on page 244
- "Fuzzy Type Match" on page 245
- "Invert" on page 245

3.12.10.1 FMT Translation

The iATU automatically sets the TLP format field for three DWORDs when it detects all zeroed in the upper 32 bits of the translated address. Otherwise it sets it to four DWORDs when it detects a 64-bit address (when there is a 1 in the upper 32 bits of the translated address). When the original address and the translated address are of a different format then the iATU ensures that the TLP header size matches the translated address format.

3.12.10.2 Optional Matching Fields

Overview: In address and BAR match modes, a successful address/BAR match can be optionally gated by successful matching of the programmable TLP header fields (per region) mentioned in Table 3-73. When the associated Match Enable bit is set, the TLP header field of an inbound TLP is matched to the programmed TLP header field value.

Feature Configuration: Table 3-73 provides information on the registers to consider for TLP header field matching.

Table 3-73 Registers to Consider for TLP Header Field Matching

TLP Header Field	Match Enable Bit		TLP Header FieldValue to Match	
	Field	Register	Field	Register
TC	TC_MATCH_EN	IATU_REGION_CTRL_2_OFF_INBOUND_i	TC	IATU_REGION_CTRL_1_OFF_INBOUND_i
TD	TD_MATCH_EN		TD	
ATTR	ATTR_MATCH_EN		ATTR	
TH	TH_MATCH_EN		TH	
AT	AT_MATCH_EN		AT	
PH	PH_MATCH_EN		PH	
ST	MSG_CODE_MATCH_EN		MSG_CODE	IATU_REGION_CTRL_2_OFF_INBOUND_i
Message Code	MSG_CODE_MATCH_EN		MSG_CODE	

Note Address translation only proceeds when compares on all enabled field are successful.

3.12.10.3 Single Address Location

Overview: Single address location feature has two modes of operation, fixed and circular buffer.

Fixed: When enabled and region address matches, the TLPs translates to a single address location as determined by the target address register of the iATU region.

Received Message TLP Header Bytes 8 - 15 at TRGT1 Interface

Bytes 8 to 15 of the received TLPs header destined for TRGT1 are made available on `radm_trgt1_hdr_uppr_bytes` irrespective of TLP translation. When `radm_trgt1_hdr_uppr_bytes_valid` is asserted your application should use `radm_trgt1_hdr_uppr_bytes` along with `radm_trgt1_data`. The `radm_trgt1_hdr_uppr_bytes_valid` is set only when an ATU match occurs for Single Address Location enabled region.

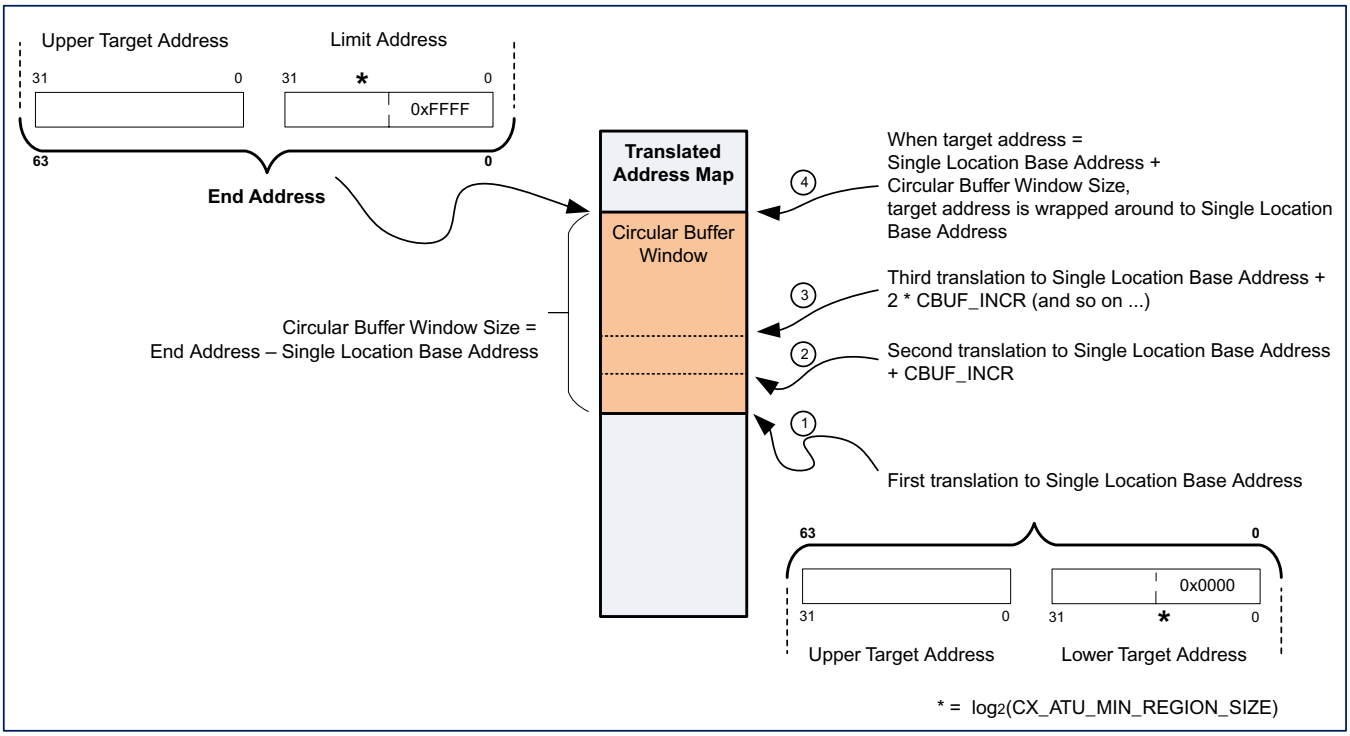
For example, if a received MsgD matches a Single Address Location enabled ATU region, bytes 8 to 15 of the MsgD header are available on `radm_trgt1_hdr_uppr_bytes` when `radm_trgt1_hdr_uppr_bytes_valid` is asserted, and `radm_trgt1_hv` is true. The available `radm_trgt1_hdr_uppr_bytes` can be concatenated with `radm_trgt1_data` and used by your application.

Note

- The width of `radm_trgt1_hdr_uppr_bytes` is `FLT_Q_ADDR_WIDTH` bits.
- For `radm_trgt1_hdr_uppr_bytes[63:32]` to exist `FLT_Q_ADDR_WIDTH` mustbe64.
- If 32 <`FLT_Q_ADDR_WIDTH` < 64, `radm_trgt1_hdr_uppr_bytes[FLT_Q_ADDR_WIDTH-1:32]` =0.

Circular Buffer: In this mode, the first TLP translates to a single address location as determined by the target address register of the iATU region. But, for subsequent TLPs, the target address increments by a programmable increment size (`CBUF_INCR`), each time a region address matches. You can program the increment size using `CBUF_INCR` field of the `IATU_LIMIT_ADDR_OFF_INBOUND_i` register (the maximum increment size is 8192; this incorporates the maximum VDM data payload of 4096 and 2 DWs of header `radm_trgt1_hdr_uppr_bytes`). On reaching the circular buffer window size, the target address wraps around, and resets to single location base address. The controller uses the output signal `radm_trgt1_atu_cbuf_err` to notify your application if the region matched received VDM size is greater than the programmed circular buffer increment size (`CBUF_INCR`).

Figure 3-89 Circular Buffer Operation



Note The controller output port radm_trgt1_atu_sloc_matchcan be used as IRQs for any VDM received which is processed in theiATU using the circular buffer feature.

Usage Scenario: The main usage scenario is translation of Messages (such as Vendor Defined or ATS Messages) to MWr TLPs when the AXI bridge is enabled.

Feature Configuration: To enable single address location feature, set the `SINGLE_ADDR_LOC_TRANS_EN` field of the `IATU_REGION_CTRL_2_OFF_INBOUND_i` register to '1'. Table 3-74 describes additional configuration settings for choosing the single address location operation mode.

Steps to Program ATU Circular Buffer Registers

1. Write to Upper Target Address, Lower Target Address, and Limit Address (all inbound) registers for start and end address values of Circular Buffer window for the ATU region.
2. Set `SINGLE_ADDR_LOC_TRANS_EN =1` to enable single location address.
3. Write the desired increment value to the `CBUF_INCR` field of the `IATU_LIMIT_ADDR_OFF_INBOUND_i` register. This enables the circular buffer feature.

Table 3-74 Setting the Single Address Location Operation Mode

Fixed	Circular Buffer
IATU_LIMIT_ADDR_OFF_INBOUND_i. CBUF_INCR =0	IATU_LIMIT_ADDR_OFF_INBOUND_i. CBUF_INCR >0
	IATU_REGION_CTRL_2_OFF_INBOUND_i. SINGLE_ADDR_LOC_TRANS_EN =1
	CX_ATU_SLOC_CBUF =1(default)
	CX_ATU_MAX_REGION_SIZE =0

3.12.10.4 Response Code

Overview: Based on the setting of the response code field of the `IATU_REGION_CTRL_REG_2_INBOUND_i` register (2b00: Normal RADM filter response, 2b01: UR, or 2b10: CA), the controller sets the completion status field of completion TLPs, sent in response to successfully matched non-posted TLPs. When the response code field is set to 2b00, then the normal receive filter response for this TLP is used.

Note Response Code feature is not available in regions where Single Address Location Enable is set.

Feature Configuration: To enable response code feature, set the `RESPONSE_CODE` field of the `IATU_REGION_CTRL_2_OFF_INBOUND_i` register!='00'.

3.12.10.5 Message Type Match

Overview: When this feature and "Single Address Location"is enabled, the iATU matches Msg/MsgD TLP Type field with TYPE field of `IATU_REGION_CTRL_1_OFF_INBOUND_i` register. If "Fuzzy Type Match"is also enabled, then any Msg received is matched (that is, Msg Type's sub-field r[2:0], which specifies the Message routing mechanism, is ignored).

Caution The Message should be consumed by your application before the next message arrives as all messages go to the same address.

Note If `SINGLE_ADDRESS_LOCATION_TRANSLATE_EN` is set for any region, then you must ensure that the same TLP cannot be matched in any other region where `SINGLE_ADDRESS_LOCATION_TRANSLATE_EN` is not set. If this happens `radm_trgt1_hdr_uppr_bytes` could have incorrect data.

Usage Scenario: This feature can be used for translation of VDM or ATS messages when AXI bridge is configured on client interface.

Feature Configuration: To enable message type match feature, set the l field of the `IATU_REGION_CTRL_2_OFF_INBOUND_i` register to ='1'.

3.12.10.6 Fuzzy Type Match

Overview: When this feature is enabled, the iATU relaxes the matching of the TLP type field against the expected type field so that:

- CfgRd0 and CfgRd1 TLPs are seen as identical. Similarly with CfgWr0 and CfgWr1.
- MWr, MRd, and MRdLk TLPs are seen as identical.
- The routing field of MsgD TLPs is ignored.
- Atomic Ops TLPs - FetchAdd, Swap, and CAS are seen as identical.

For example, CFG0 in the type field in the `IATU_REGION_CTRL_1_OFF_INBOUND_i` register matches against an inbound CfgRd0, CfgRd1, CfgWr0, or CfgWr1 TLP. To enable this feature, set the Fuzzy Type Match Mode bit of the `IATU_REGION_CTRL_OFF_2_INBOUND_i` register.

Feature Configuration: To enable fuzzy type match feature, set the `FUZZY_TYPE_MATCH_CODE` field of the `IATU_REGION_CTRL_2_OFF_INBOUND_i` register to ='1'.

3.12.10.7 Invert

Overview: Normally an address match on an inbound TLP occurs when the untranslated address is in the region bounded by the Base address and Limit address. When the invert feature is enabled, an address match occurs when the untranslated address is not in the region bounded by the Base address and Limit address.

Usage Scenario: This feature is supported only when all regions of that type are set to address match mode.

Feature Configuration: To enable invert feature, set the `INVERT_MODE` field of the `IATU_REGION_CTRL_2_OFF_INBOUND_i` register to ='1'.

3.12.11 Inbound Programming

This section discusses the register programming interface and provides details of programming examples.

3.12.11.1 iATU Inbound Programming Overview

You can access the iATU registers through the DBI interface or through BAR Matched Mem/IO requests. The following registers are used for programming the iATU.

Table 3-75 iATU Register Map

Address	Label (i = 0 to CX_ATU_NUM_INBOUND_REGIONS-1)	Description
0x100	IATU_REGION_CTRL_1_OFF_INBOUND_*i*	Region Control 1
0x104	IATU_REGION_CTRL_2_OFF_INBOUND_*i*	Region Control 2
0x108	ATU_LWR_BASE_ADDR_OFF_INBOUND_*i*	Lower Base Address
0x10C	IATU_UPPER_BASE_ADDR_OFF_INBOUND_*i*	Upper Base Address
0x110	IATU_LIMIT_ADDR_OFF_INBOUND_*i*	Limit Address
0x114	IATU_LWR_TARGET_ADDR_OFF_INBOUND_i	Lower Target Address
0x118	IATU_UPPER_TARGET_ADDR_OFF_INBOUND_*i*	Upper Target Address
0x11C	IATU_REGION_CTRL_3_OFF_INBOUND_*i*	Region Control 1
0x120	IATU_UPPR_LIMIT_ADDR_OFF_INBOUND_*i*	Upper Limit Address

Table 3-76 iATU Register Address Bus Layout

Register Bit	31-20	19	18-17	16-9	8	7-2	1	0
Value	0	0	Reserved	Region	Inbound/Outbound	Register Address	0	1

Table 3-77 Example: Address Table for Accessing iATU Inbound Region 1 Configuration Registers (Native DBI)

Address	Register Name	Register Bit Value						
		31-20	19	18-17	8	7-2	1	0
0x300	Region Control 1	0	0	0	101			
0x304	Region Control 2	0	0	0	105			
0x308	Lower Base Address	0	0	0	109			
0x30C	Upper Base Address	0	0	0	10D			
0x310	Limit Address	0	0	0	111			
0x314	Lower Target Address	0	0	0	115			
0x318	Upper Target Address	0	0	0	119			

Address	Register Name	Register Bit Value						
		31-20	19	18-17	8	7-2	1	0
0x31C	Region Control 3	0	0	0	119			
0x320	Upper Limit Address	0	0	0	119			

Attention

For releases previous to 4.80a, the iATU registers have been programmed through an indirect addressing scheme using an index register (iATU_VIEWPORT_OFF), to reduce the address footprint in the PCI Express extended configuration space. To use the legacy method of accessing iATU registers, contact Synopsys support through SolvNetPlus.

Caution

You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the “Integrating the Controller and VIP using VTB” section in the “Integration” chapter of the *User Guide*.

Caution

Dynamic iATU Programming with AXI Bridge Module

You must not update the iATU registers while operations are in progress on the AXI bridge subordinate interface.

3.12.11.2 Inbound Programming Examples

Address Match Mode Example

Define Inbound Region 0 as, MEM region matching TLPs with addresses in the range `0x00010000` to `0x0005ffff` mapped to `0x1000_0000_2000_0000` - `0x1000_0000_2004_ffff` in your application memory space.

1. Setup the Region Base and Limit Address Registers.

   Write `0x00010000` to Address {`0x108`} to set the Lower Base Address.

   Write `0x00000000` to Address {`0x10C`} to set the Upper Base Address.

   Write `0x0005ffff` to Address {`0x110`} to set the Limit Address
2. Setup the Target Address Registers.

   Write `0x20000000` to Address {`0x114`} to set the Lower Target Address.

   Write `0x10000000` to Address {`0x118`} to set the Upper Target Address.
3. Configure the region through the Region Control 1 Register.

   Write `0x00000000` to Address {`0x100`} to define the type of the region to be MEM.
4. Enable the region.

   Write `0x80000000` to Address {`0x104`} to enable the region in address match mode.

3.13 Address Translation Services (ATS)

The controller implements ATS (`CX_ATS_ENABLE`) with the following features and limitations.

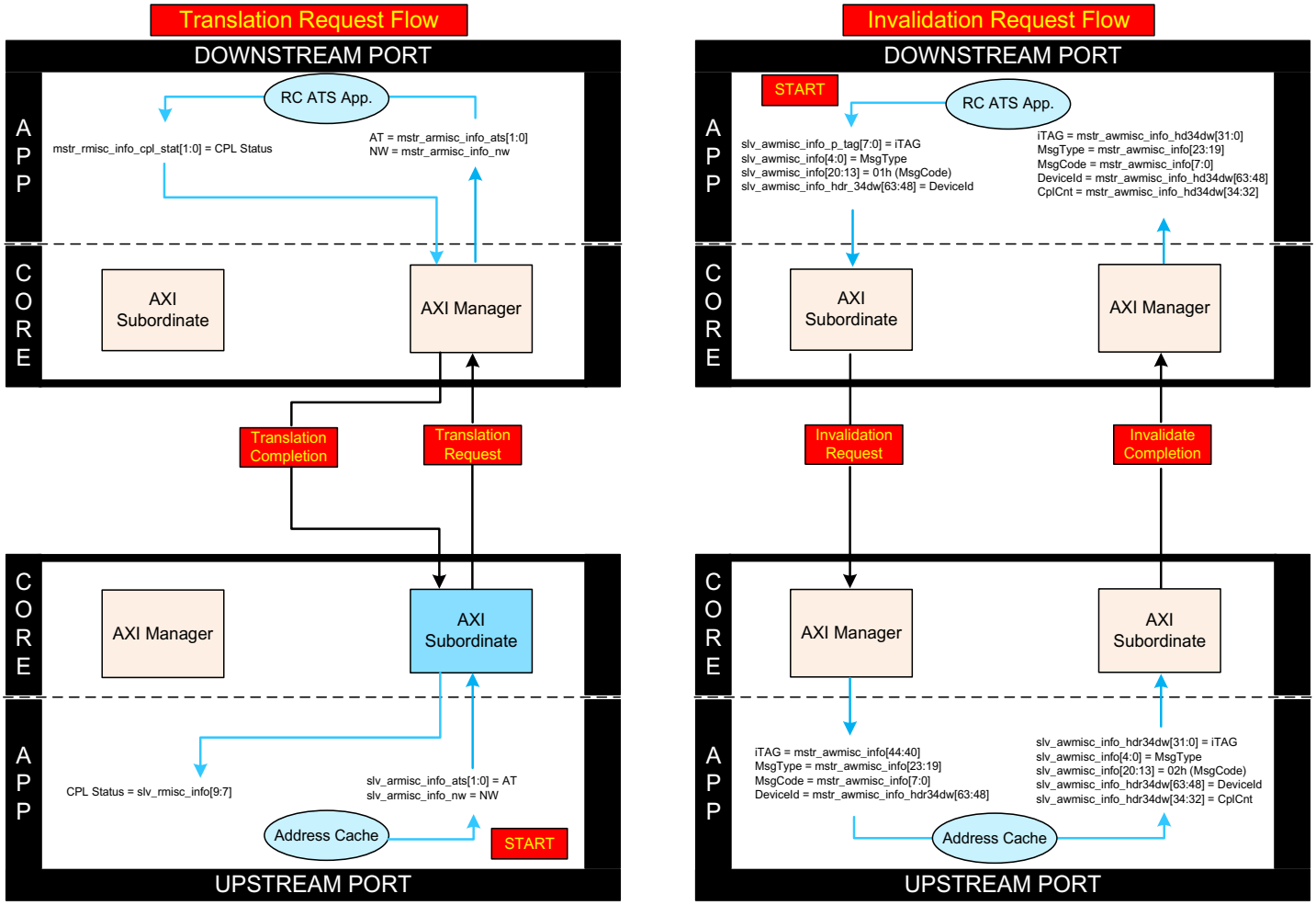
- To send an ATS TLP, you must supply it on an application transmit client interface (XALI0/1/2) taking note of the `client*_tlp/_ats/_nw` inputs, or on the AXI bridge subordinate interface as per Figure 3-90.

  You can also send an Invalidation Completion using the VMI[1].
- The controller routes all Rx ATS TLPs to the TRGT1[2] receive interface (note `radm_trgt1_tlp_/ats/_nw` outputs), or to the AXI bridge master interface (note the `mstr_a*misc_info_ats/mstr_a*misc_info_nw` outputs).
- The controller also routes all Rx ATS TLPs (except ATS Invalidate Requests and ATS Invalidate Completions) to the SII interface.

  The controller does not pass Rx ATS Invalidate Requests and ATS Invalidate Completions to the SII interface because they are too big. All ATS Invalidate Requests and ATS Invalidate/Completions and pass through to TRGT1 (or AXI bridge Master) regardless of the filtering rule settings discussed in "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691".
- When the AXI bridge is enabled:
  - Translation requests (AT =01) are supported for a single address only (length =two DWORDs).
  - Requests for multiple translations are not supported.
- The controller does not implement the ATS cache.

1. ven_msg_type[4:0]=0x12; ven_msg_code[7:0]=0x02; ven_msg_data[63:0]={ITAG Vector, CC, Device ID}
2. There is no need for an NW bit on BYP because the non-posted request RX queue cannot be put into bypass mode.

Figure 3-90 AXI ATS Sideband Signal Usage



3.14 Page Request Services (PRS)

The controller includes the PRS capability structure (in RP mode) when you set `CX_PRS_ENABLE =1` so that your RP application can receive PRS messages.

3.15 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 Endpoint to another[1], instead of strictly flowing up and down the PCIe hierarchy. The PCI Express specification describes this capability in terms of routing peer-to-peer (P2P) transactions between ports. Though identified by the PCIe specification, the mechanism and support of this feature is considered to be outside the scope of the base specification. To enable P2P support in RP mode, you must set the `CX_P2P_ENABLE` configuration parameter. Enabling P2P support modifies the operation of the controller in the following ways:

- The controller distinguishes between locally generated transmit traffic and P2P traffic, and only tracks locally generated requests in the receive completion LUT.
- Only received completions corresponding to requests originated by the device are subject to the normal receive completion LUT processing. Completions of P2P requests do not reference the LUT.
- The controller preserves the original requester or completer IDs for P2P transmit traffic while automatically adding the appropriate ID for locally generated traffic.
- All TLP header fields are preserved in both transmit and receive directions. Therefore the controller provides all reserved fields at receive interfaces and expects your application to drive them at the client interfaces.
- The controller optionally appends ECRC fields to locally generated TLPs, passing P2P traffic along without modification.

The following limitations apply with respect to the P2P feature:

- No ECRC stripping.
- The controller does not split transmit packets. Your application must handle this condition when the peer-to-peer packet exceeds the max payload limitation.

Attention

Supporting P2P with AXI Bridge Module

You cannot enable the CX_P2P_ENABLE parameter when you configure the controller with the AXI. This means that the previously mentioned features for supporting P2P are not available in the controller. However, you can still support P2P operation by implementing similar features in your application logic. Assuming (for example) that you have two Root Ports connected together by your AXI application logic. Then P2P transactions appear to terminate and originate from your application even though the device generating the originating requests and the target completer are EPs on different PCIe hierarchical trees.

Full information of I/O signals and timing requirements for the P2P are described in “Signal Descriptions” chapter of the *Reference Manual*.

1. In this mode, the RP is effectively acting as a Switch.

3.16 Gen2/Gen3/Gen4/Gen5 Speed Modes

This section discusses the Gen2/Gen3/Gen4/Gen5 speed mode operation.

3.16.1 Gen2 5.0 GT/s Operation

The controller supports all of the non-optional Gen2 5.0 GT/s features defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The interface between the controller and the PHY is compliant with *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*. The controller supports two different mechanisms of achieving the Gen2 rate: dynamic frequency and dynamic width.

- For dynamic frequency configurations, the number of active symbols on the PIPE is constant and the frequency of the controller doubles each time as the controller transitions from Gen1 => Gen2 rate.
- When supporting dynamic width, the clock frequency of the controller remains constant and the number of active symbols on the PIPE doubles each time as the controller transitions from Gen1 => Gen2 rate.

Note For more information on supported clock frequencies and datapath widths, see "Frequency, Speed, and Width Support" on page 36.

3.16.1.1 Gen2 Speed Changing

When you set the default of the Directed Speed Change field of the Link Width and Speed Change Control register (`GEN2_CTRL_OFF. DIRECT_SPEED_CHANGE`) using the `DEFAULT_GEN2_SPEED_CHANGE` configuration parameter to '1', then the speed change is initiated automatically after link up, and the controller clears the contents of `GEN2_CTRL_OFF. DIRECT_SPEED_CHANGE`. If you want to prevent this automatic speed change, then write a lower speed value to the Target Link Speed field of the Link Control 2 register (`LINK_CONTROL2_LINK_STATUS2_REG. PCIE_CAP_TARGET_LINK_SPEED`) through the DBI before link up.

To manually initiate the speed change then:

- Write to `LINK_CONTROL2_LINK_STATUS2_REG. PCIE_CAP_TARGET_LINK_SPEED` in the local device
- De-assert `GEN2_CTRL_OFF. DIRECT_SPEED_CHANGE` in the local device
- Assert `GEN2_CTRL_OFF. DIRECT_SPEED_CHANGE` in the local device

The controller uses the `mac_phy_rate` output to negotiate the link data rate. It 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.

For `core_clk` considerations when changing speed, see "Gen2, Gen3, Gen4, or Gen5 Speed Changing Considerations (CX_FREQ_STEP_EN =1)" on page 73.

Note As per the *PCI Express Base Specification, Revision 6.0, Version 1.0*, the controller does not implement the optional Compliance Receive bit for Gen1 configurations.

3.16.2 Gen3 8.0 GT/s Operation

The controller supports all of the non-optional Gen3 8.0 GT/s features defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The interface between the controller and the PHY is compliant with *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*. The controller supports two different mechanisms of achieving the Gen3 rate: dynamic frequency and dynamic width.

- For dynamic frequency configurations, the number of active symbols on the PIPE is constant and the frequency of the controller doubles each time as the controller transitions from Gen1 => Gen2 => Gen3 rates.
- When supporting dynamic width, the clock frequency of the controller remains constant and the number of active symbols on the PIPE doubles each time as the controller transitions from Gen1 => Gen2 => Gen3 rates.

Note For more information on supported clock frequencies and datapath widths, see "Frequency, Speed, and Width Support" on page 36.

3.16.2.1 Gen3 Speed Changing

See "Gen2 Speed Changing" on page 252.

3.16.2.2 Gen3 8.0 GT/s Transmitter Equalization

The controller performs link equalization during link training to improve signal quality by adjusting the transmitter and receiver equalization parameters for each lane on each side of the link. There are three modes for the mapping of presets to coefficients, selectable at configuration time using the `CX_GEN3_EQ_PSET_COEF_MAP_MODE` parameter:

- Dynamic PHY: The coefficients are dynamically mapped in the PHY. This is the default mode.
- Dynamic MAC: The coefficients are dynamically mapped in the MAC.
- Programmable Table: The coefficients are programmed in a table by your application.

There are two feedback modes for determining the optimal equalization settings, programmable using the Gen3 EQ Control Register (`GEN3_EQ_CONTROL_OFF`).

- Figure of merit (FOM)
- Direction change with optional convergence support

Hint For more information on the EQ procedure, see "Advanced Information: Gen3/4/5 Equalization Details and Example" on page 623.

3.16.3 Gen4 16.0 GT/s Operation

The controller supports the Gen4 16.0 GT/s feature defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The interface between the controller and the PHY is compliant with *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*.

The controller supports two different mechanisms of achieving the Gen4 rate: dynamic frequency and dynamic width.

- For dynamic frequency configurations, the number of active symbols on the PIPE is constant and the frequency of the controller doubles each time as the controller transitions from Gen1 => Gen2 => Gen3 => Gen4 rates.
- When supporting dynamic width, the clock frequency of the controller remains constant and the number of active symbols on the PIPE doubles each time as the controller transitions from Gen1 => Gen2 => Gen3 rates => Gen4.

Note For more information on supported clock frequencies and datapath widths, see "Frequency, Speed, and Width Support" on page 36.

3.16.3.1 Gen4 Speed Changing

See "Gen2 Speed Changing" on page 252.

3.16.3.2 Gen4 16.0 GT/s Transmitter Equalization

The controller performs link equalization during link training to improve signal quality by adjusting the transmitter and receiver equalization parameters for each lane on each side of the link. There are three modes for the mapping of presets to coefficients, selectable at configuration time using the `CX_GEN3_EQ_PSET_COEF_MAP_MODE` parameter:

- Dynamic PHY: The coefficients are dynamically mapped in the PHY. This is the default mode.
- Dynamic MAC: The coefficients are dynamically mapped in the MAC.
- Programmable Table: The coefficients are programmed in a table by your application.

There are two feedback modes for determining the optimal equalization settings, programmable using the Gen3 EQ Control Register (`GEN3_EQ_CONTROL_OFF`).

- Figure of merit (FOM)
- Direction change with optional convergence support

Hint For more information on the EQ procedure, see "Advanced Information: Gen3/4/5 Equalization Details and Example" on page 623.

3.16.3.3 Gen4 Lane Margining

The controller supports all mandatory PIPE message bus interface features specified in *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*, except for elastic buffer depth control and generation of read command from the controller to PHY. Figure 3-91 to Figure 3-97 describe the lane margining operation. The figures are for reference purposes only and may not be aligned with actual implementation.

Figure 3-91 Start Margining

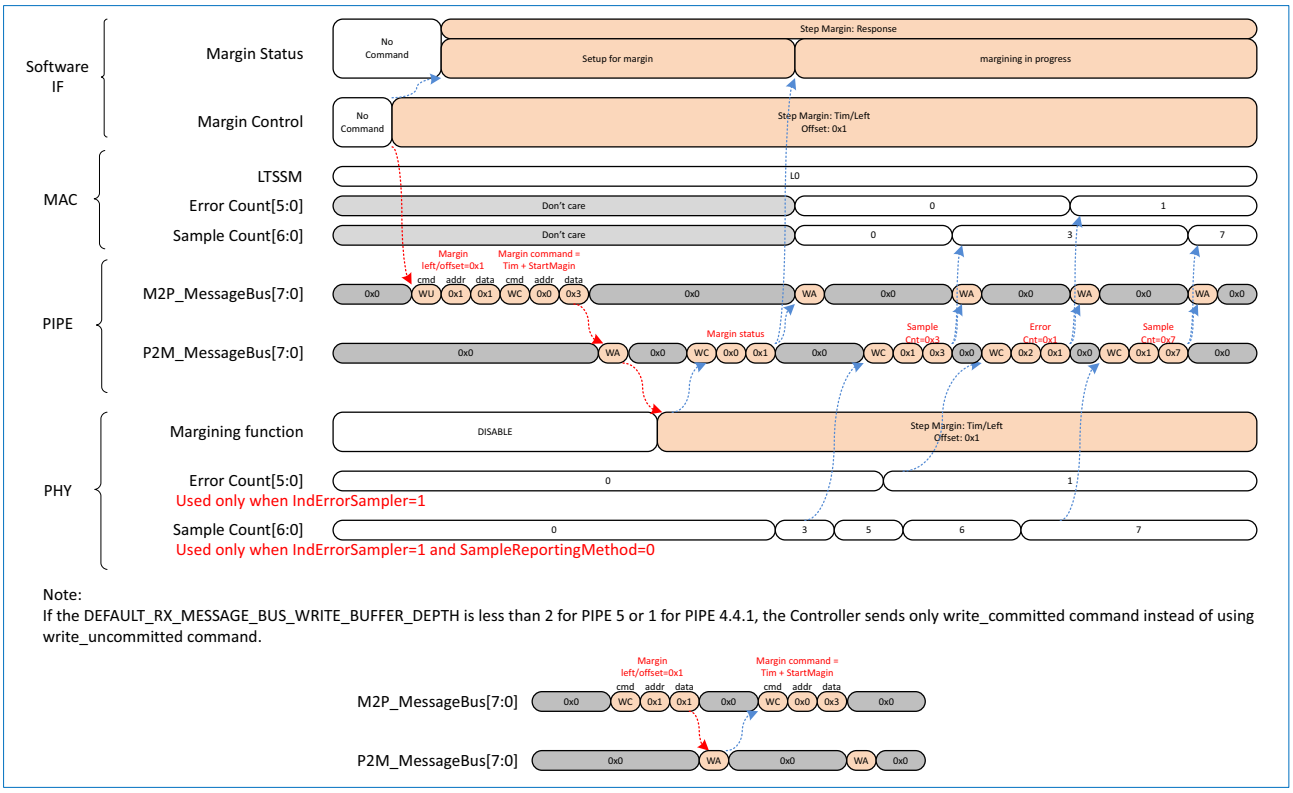


Figure 3-92 Stop Margining

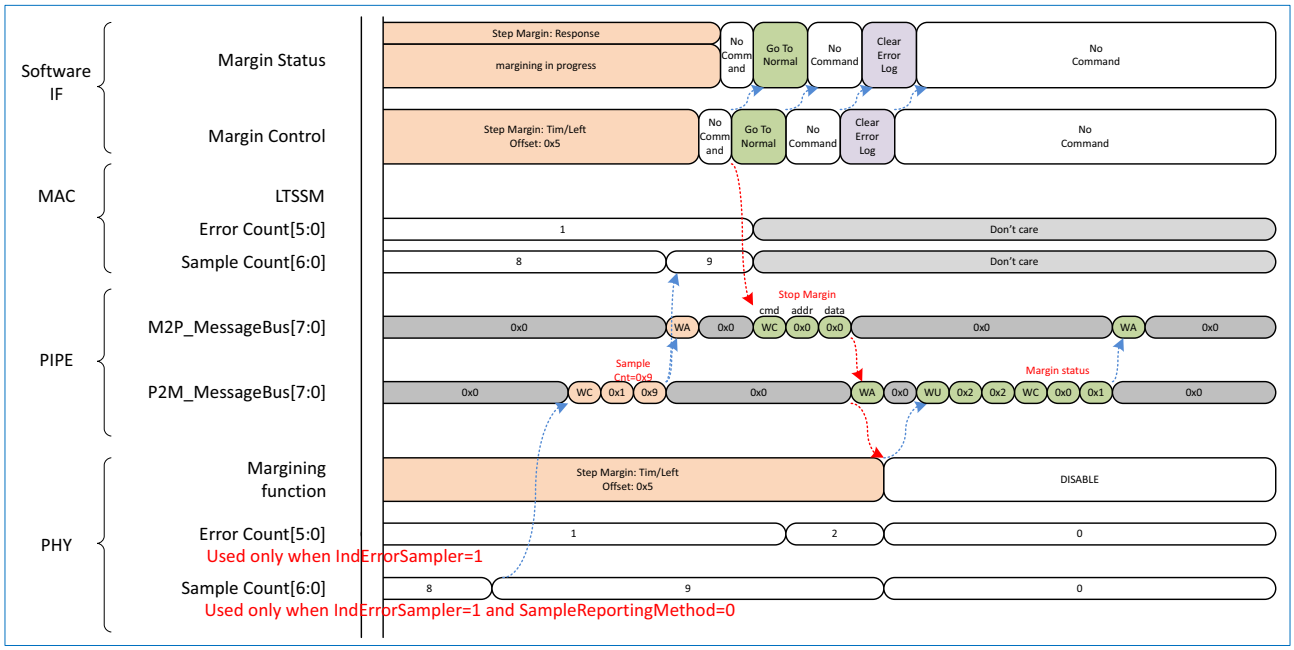


Figure 3-93 Offset Change

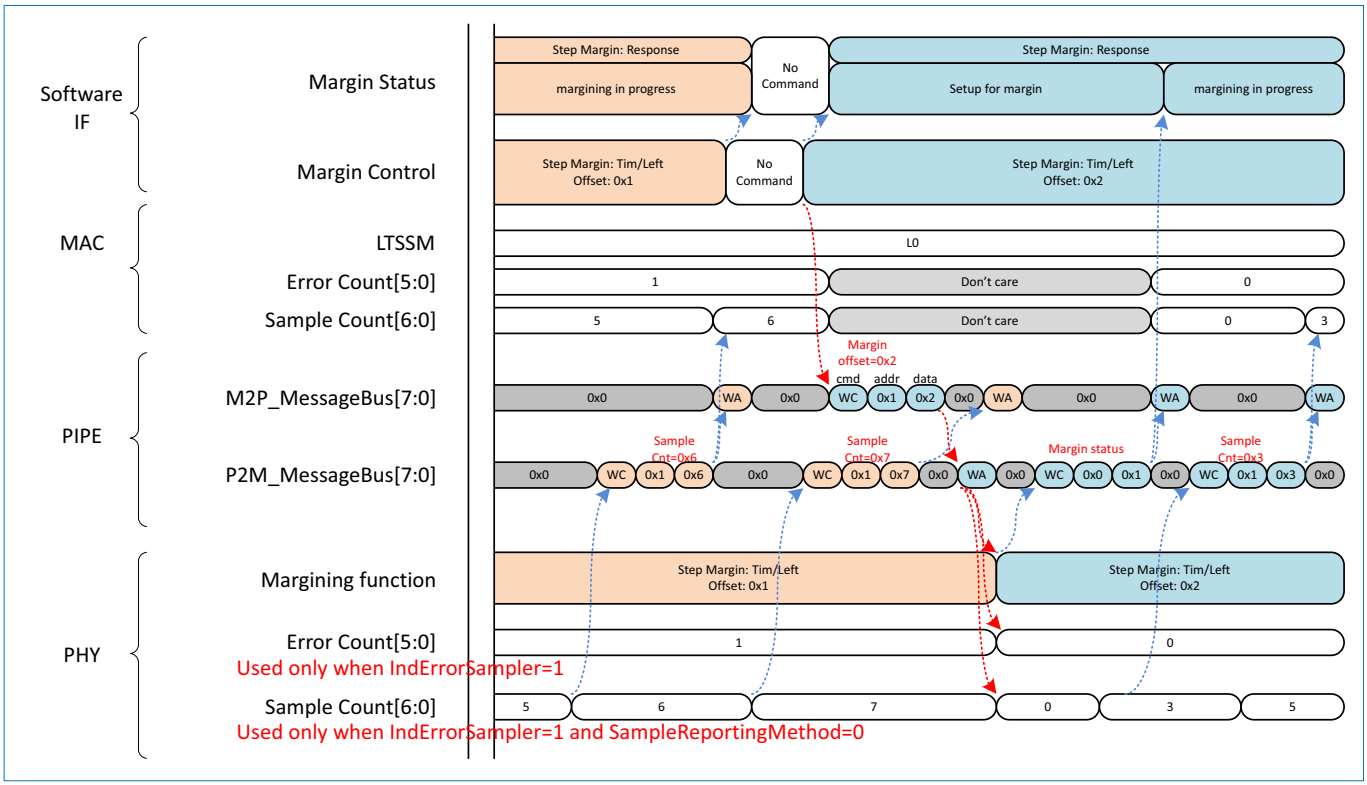


Figure 3-94 NAK Response

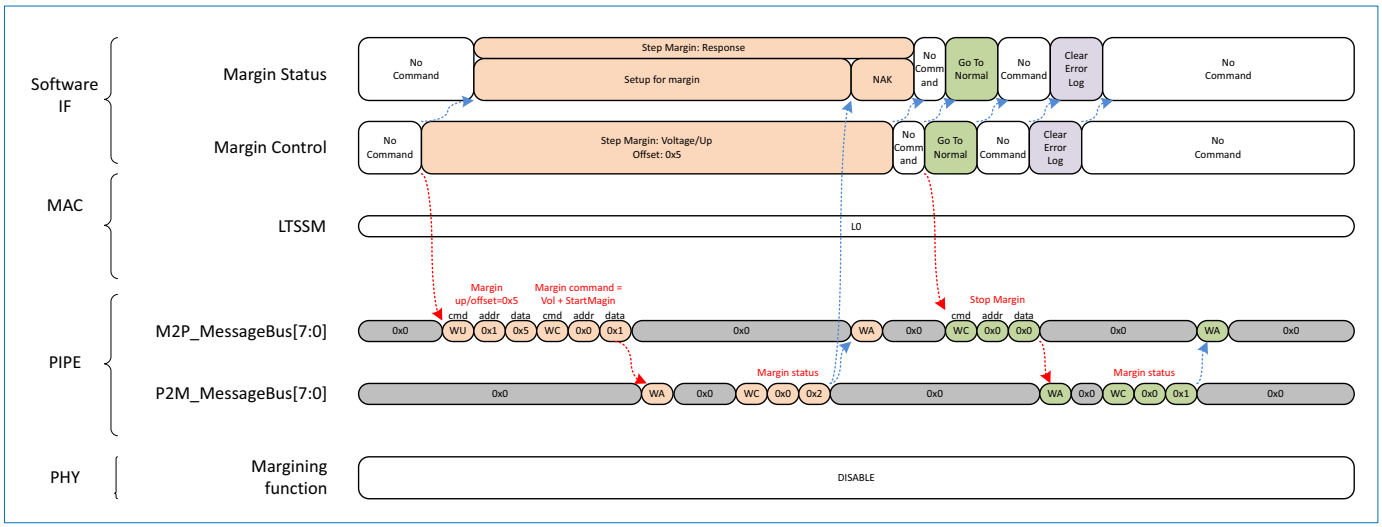
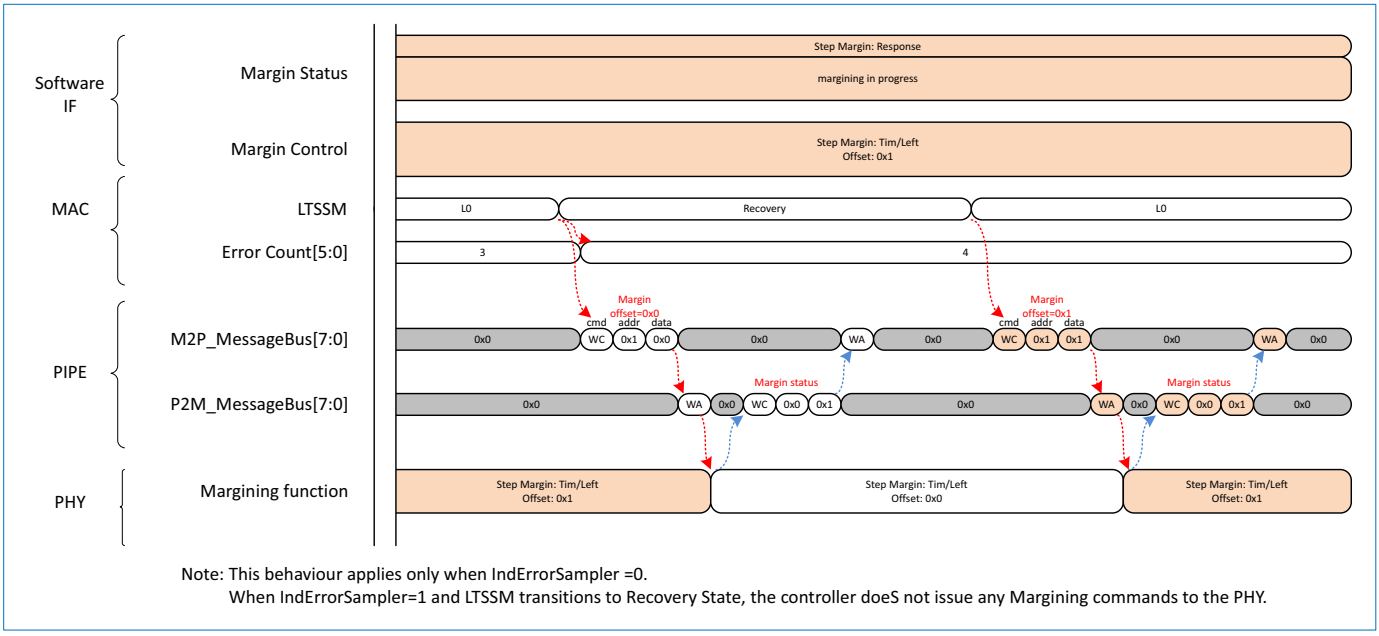


Figure 3-95 Recovery Transition (IndErrorSampler =0)



Note: This behaviour applies only when IndErrorSampler =0.
When IndErrorSampler=1 and LTSSM transitions to Recovery State, the controller doeS not issue any Margining commands to the PHY.

Figure 3-96 Clear Error Log

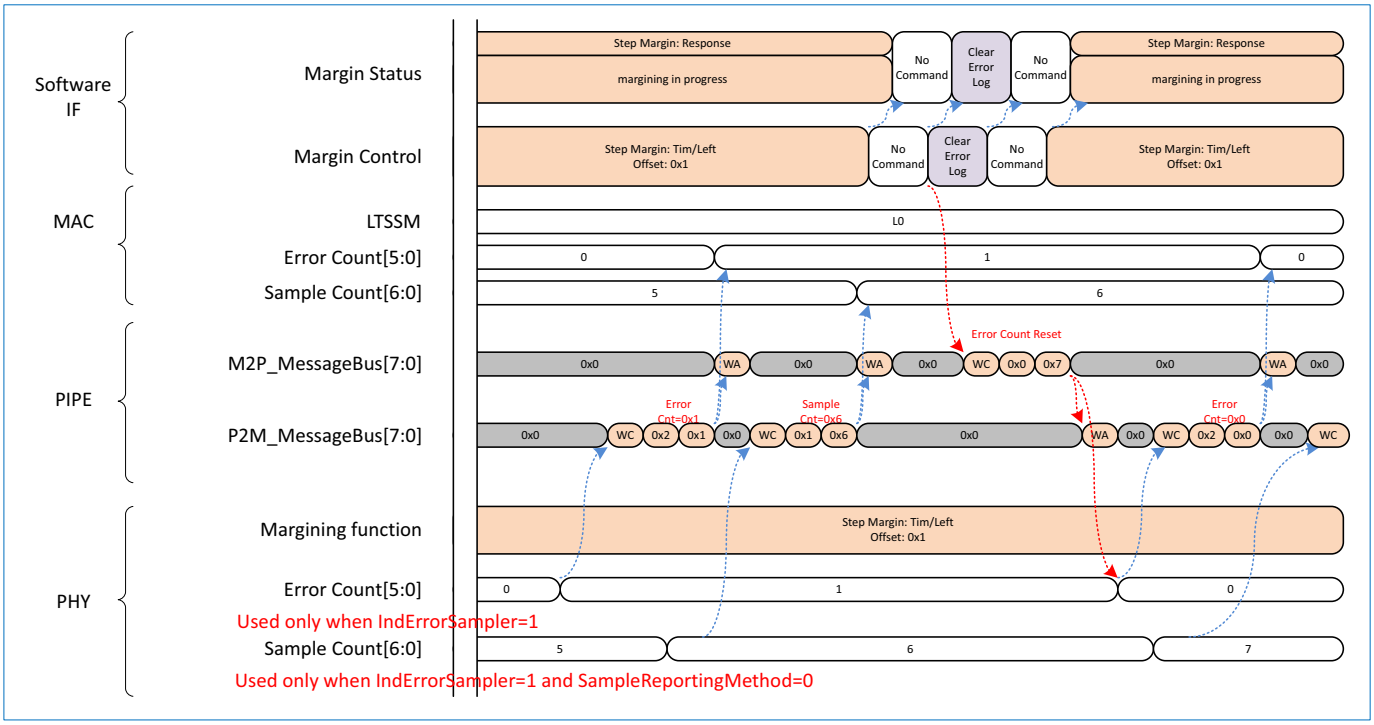
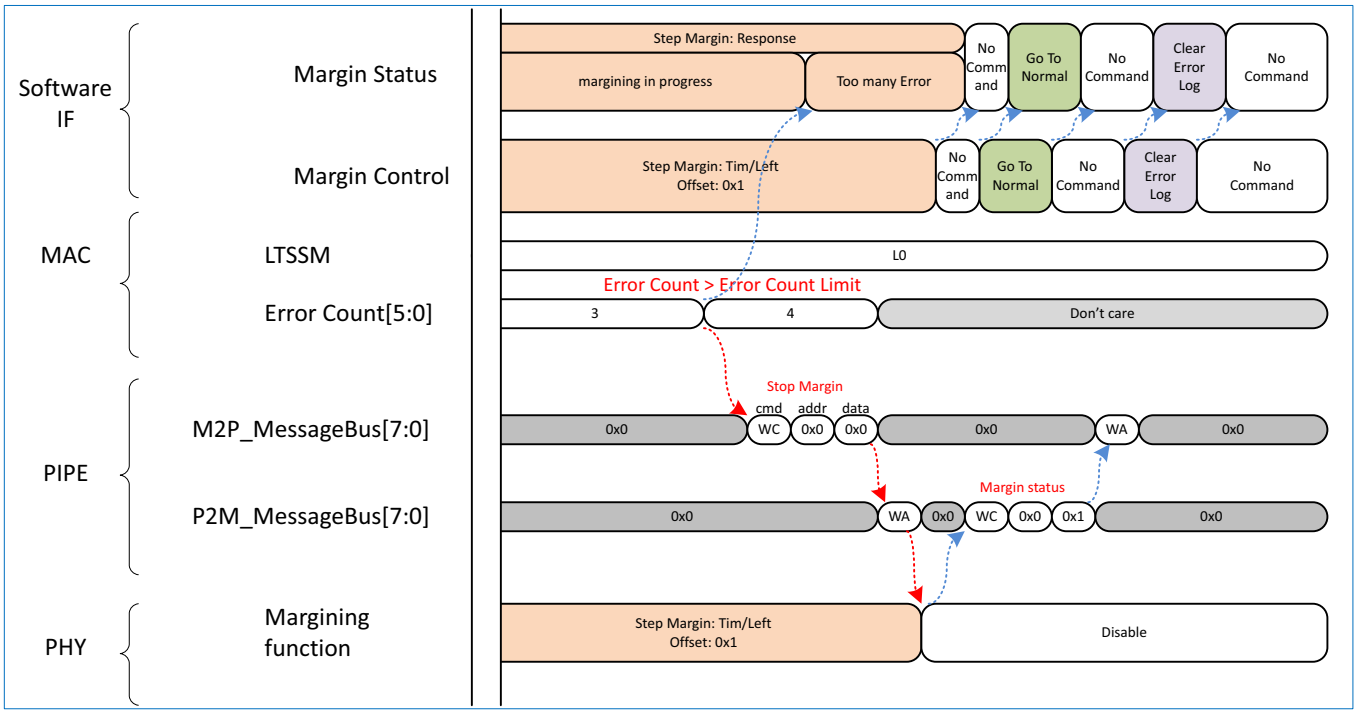


Figure 3-97 Exceeding Error Limit



3.16.4 Gen5 32.0 GT/s Operation

The controller supports the Gen5 32.0 GT/s feature defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The interface between the controller and the PHY is compliant with *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*. The controller supports only one mechanism of achieving the Gen5 rate in this release, that is, dynamic frequency.

For dynamic frequency configurations, the number of active symbols on the PIPE is constant and the frequency of the controller doubles each time as the controller transitions from Gen1 -> Gen2 -> Gen3-> Gen4 -> Gen5 rates.

3.16.4.1 Gen5 Speed Changing

See "Gen2 Speed Changing" on page 252.

3.16.4.2 Gen5 32.0 GT/s Transmitter Equalization

See "Gen4 16.0 GT/s Transmitter Equalization" on page 254.

3.16.4.3 Lane Margining

See "Gen4 Lane Margining" on page 254.

3.17 Power Management

This section discusses the power management (PM) features of the controller. For a proper understanding of PCIe power management, you should read Chapter 5, Power Management of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The following topics are discussed:

- "Overview" on page 260
- "L1 Substates" on page 266
- "L0s Entry and Exit Conditions" on page 278
- "L0p" on page 279
- "L1 Operation (Non-substates)" on page 284
- "L2 and L3 Power Down Entry and Exit Conditions (USP)" on page 291
- "Advanced Power Management and Power Domain Gating" on page 294

3.17.1 Overview

The controller supports two categories of PM operations to control the device state (D-state) and link state.

- **Software PCI Compatible PM (PCI-PM)**
  - **D-state PM of Function**

    The host software can direct the function to enter any of the D1, D2, or D3 low-power states. It does this by writing to the Power Management Control and Status Register (PMCSR) in the PCI-PM capability structure.
  - **D-state PM of Link**

    Link states are not visible to PCI-PM legacy compatible software, and are derived from the power management D-states of the components connected to that link. The action of changing the D-state in the PMCSR indirectly causes a change in the link power state. The L1 state is entered whenever all functions of a USP on a link are programmed to a non-D0 state. The entry into L2 and L3 states is initiated by the DSP.
  - **Clock PM (L1 with REFCLK removal/PLL Off)**

    This is an optional feature that enables components on a link to further reduce idle power consumption while the link is in L1, by turning off the PLL.
- **Native PCIe PM Mechanisms**
  - **Active State PM (ASPM)**

    When the USP is in L0 and detects idleness on the link for a specific amount of time, it automatically transitions the link to the L0s or L1 (optional) power state.
  - **L1 Substates**

    This is an optional PCIe feature that enables components on a link to further reduce idle power consumption while the link is in L1, including almost complete removal of power for the high speed PHY circuits.

3.17.1.1 Relationship Between D-state, Link State, and PHY State

The link state and D-state determine what power and clock supplies are on or off. For implementation information of low-power clocking logic, see "Clock Requirements" on page 68.

Table 3-78 Power/Clock Supply Status in Low-Power States of Interest (DSP or Single Function non-ARI USP)

D-State	Permissible Quiescent Link States	Link State Change Trigger (see Table 3-79)	Active Power Supplies	Active Clock Supplies[a]	pipe_clk core_clk	PHY State
$D0_{unitialized}$	L0	NA	V_{main}[b] V_{aux}	REFCLK AUXCLK	ON	P0, P0S
$D0_{active}$	L0	ASPM				
	L0S					
	L1				ON[c]	P1
D1, D2, $D3_{hot}$	L1	PCI-PM				

D-State	Permissible Quiescent Link States	Link State Change Trigger (see Table 3-79)	Active Power Supplies	Active Clock Supplies[a]	pipe_clk core_clk	PHY State
$D3_{cold}$[d]	L2	PCI-PM	V_{aux}	AUXCLK	OFF	P2/OFF[e]
	L3[f]		None	None		

a. REFCLK is the platform reference clock for the PHY TX PLL. AUXCLK is the platform low-power clock. For more information, see "Clock Requirements" on page 68
b. Optionally removable in L2 and L1.2 using "Advanced Power Management and Power Domain Gating" on page 294.
c. You can remove REFCLK (and pipe_clk/core_clk). For more information, see "Removing the Reference Clock" on page 77
d. Transitioned from D3hot to D3cold by removal of Vmain. D3cold is entered by the controller in response to PME_Turn_Off MSG
e. When CX_P2NOBEACON_ENABLE =1 and PHY_INTEROP_CTRL_OFF.P2NOBEACON_ENABLE =1, mac_phy_powerdown drives P2.NoBeacon encoding, instead of P2 encoding, when the link goes to L2.
f. A function can be transitioned from L2 to L3 by the removal of Vaux.

Table 3-79 Link-State-Change Trigger Capabilities for Each Port Type

Port Type	Can Trigger L0/L0s Entry	Can Trigger L1 Entry	Can Trigger L2/L3 Entry
USP	Yes	Yes, on any of these triggers[a]: ■ Software programs non-D0 state (PCI-PM) ■ USPs idle timer expires (ASPM) ■ Application requests the USP to enter L1[b]	No
DSP	Yes	No[c]	Yes[d]

a. You can prevent/delay the controller from entering L1-ASPM (not L1-PCI-PM) by asserting app_xfer_pending.
b. USP application asserts the app_req_entr_l1 input. A DSP controller ignores this input.
c. DSP can indirectly initiate L1 by writing non-D0 value to PMCSR of the USP link partner. A non-D0 write to the USP PMCSR always triggers a transition to L1.
d. L2/L3 entry is initiated after the RP PM software transitions a device into a D3 state and subsequently instructs the DSP to transmit PME_Turn_Off message TLP to initiate the removal of power and clocks.

Table 3-80 indicates the different levels of power savings (obtained by switching off circuits outside the controller) that can be expected from executing L1 in different ways. Higher exit latencies are associated with higher power savings. Power saving and latency are PHY implementation specific, and may be determined by consulting your PHY documentation.

Table 3-80 L1 Execution Modes and Permissible Power States of PHY Circuits

L1 Execution Mode	Permissible PowerStates for PHY Circuits		
	PLL/REFCLK	RX Electrical Idle Detector	TX Common Mode
L1 without Clock PM	ON	ON	ON
L1 with Clock PM	OFF	ON	ON
L1.1	OFF	OFF	ON
L1.2	OFF	OFF	OFF

Note For more information on power consumption figures, see "Advanced Information: Area, Power Estimates, and RAM Sizes" on page 751.

For more information on optimizing system resumption time after exiting low-power states, see "Readiness Notifications (RN)" on page 322.

3.17.1.2 Device State (D-state) Power Management of a Function (USP)

The host software can direct the downstream device function to enter any of the low-power states:

- D1 (light sleep)
- D2 (deep sleep)
- D3hot (full of; reference clock optionally removed; V_{main} normally available)
- D3cold (full of; reference clock removed; V_{main} removed; V_{aux} optionally available)

It does this by writing to the PMCSR in the PCI-PM capability structure. This action also indirectly causes the downstream component (USP) to change the link power state. The controller negotiates with the link partner using PM TLPs and DLLPs before entering the corresponding low-power link state. The controller exits the device low-power state (and returns to D0) when it detects a CFG access to a device function, or when it detects a PME such as an event on the `outband_pwrup_cmd`[1] input signal. For a function in device power states D1, D2, and $D3_{hot}$, the controller only accepts CFG and MSG requests TLPs for that function. All other incoming request types for that function are treated as unsupported requests (UR). For more information see Section 5.3.3, "Power Management Event Mechanisms" of the *PCI Express Base Specification, Revision 6.0, Version 1.0.*

3.17.1.3 Active State Power Management (ASPM)

You must enable ASPM through the ASPM Control field in the Link Control register. When the USP is in L0 and detects idleness on the link for a specific amount of time (determined by the `L0S_ENTRANCE_LATENCY` field[2] in the `ACK_F_ASPM_CTRL_OFF` register), it automatically transitions the link to the L0s power state. Each link direction (TX and RX) is processed separately. The controller exits L0s when it has traffic waiting to be sent.

When the USP is in L0 or L0s and it detects idleness on the link (both directions) for a specific amount of time (determined by the `L1_ENTRANCE_LATENCY` field in `ACK_F_ASPM_CTRL_OFF`), it automatically transitions the link to the L1 power state. The link partners must negotiate entry into the L1 link power state using PM TLPs and DLLPs. The L1 state achieves greater power savings but the exit latency from this state is higher than from L0s. Instruct an USP controller (only) to enter L1 by asserting the `app_req_entr_l1` input signal. The USP exits L1 when it has traffic waiting to be sent or when it receives a PME (such as an event on the `app_req_exit_l1` input).

If the ASPM exit latencies from L0s or L1 are too high for the overall PCIe system data latency requirements, the host software disables ASPM using the `PCIE_CAP_ACTIVE_STATE_LINK_PM_CONTROL` field of the `LINK_CONTROL_LINK_STATUS_REG` register. The controller reports the exit latencies using the Link

1. Or on apps_pm_xmt_pme as these inputs are functionally identical.
2. Set this by writing to ACK_F_ASPM_CTRL_OFF register or by setting DEFAULT_L0S_ENTR_LATENCY or DEFAULT_L1_ENTR_LATENCY parameters.

Capability registers. You set these by writing to these registers or by setting the `DEFAULT_L0S_EXIT_LATENCY` and `DEFAULT_L1_EXIT_LATENCY` parameters.

3.17.1.4 ASPM Fine-grained Control

You can enable the following fine-grained control of ASPM L1 entry:

- "ASPM Top-level Control" on page 263
- "ASPM L1 Link State Entry Control" on page 263

3.17.1.4.1 ASPM Top-level Control

You can enable this feature by setting the `CX_ASPM_L1_EN_OUTPUT` parameter to '1'. Use the `cfg_aspm_l1_enable` output signal to check the value of the `PCIE_CAP_ACTIVE_STATE_LINK_PM_CONTROL` field of the `LINK_CONTROL_LINK_STATUS_REG` register, which provides the current ASPM enable state.

3.17.1.4.2 ASPM L1 Link State Entry Control

The upstream application disables the ASPM L1 entry timer by setting

1. The `CX_ASPM_L1_TIMER_ENABLE` parameter to '0'
2. The `ASPM_L1_TIMER_ENABLE` field of the `ACK_F_ASPM_CTRL_OFF` register to '0'

When the ASPM L1 entry timer is disabled, the controller starts the ASPM L1 entry negotiation when the following conditions are met:

- `ASPM_L1_TIMER_ENABLE` field of the `ACK_F_ASPM_CTRL_OFF` register is set to '1'(per Host configuration)
- `app_xfer_pending` input signal is '0'
- No TLP pending on transmission path
- No DLLP pending
- No pending transaction on Client* or RADM* interfaces
- No pending transaction on ELBI (if accessed from the wire)
- No pending OBFF sideband signals
- No eDMA transfer pending (if applicable)
- Target completion LUT is in idle state
- RADM CPL LUT is in idle state
- Transmission side has enough credits

When the controller asserts the `pm_aspm_l1_enter_ready` output signal, all the conditions mentioned in this section are satisfied except `app_xfer_pending =0`. The value of `pm_aspm_l1_enter_ready` output signal is valid when the `pm_master_state` output signal indicates that the link is in L0 or L0s state.

The application can use `pm_aspm_l1_enter_ready` and `app_xfer_pending` signals to gate or enable ASPM L1:

- `pm_aspm_l1_enter_ready` output signal indicates that the datapath is idle and the controller is ready to enter ASPM L1

- `app_xfer_pending` input signal from the application indicates that the link can proceed to ASPM L1

After the ASPM L1 entry negotiation starts, it is not affected, if the application asserts the `app_xfer_pending` signal to '1' , even if `pm_master_state` = WAIT_PMCSR_CPL_SENT, or L1_BLOCK_TLP, or L1_WAIT_LAST_TLP_ACK, or L1_WAIT_PMDLLP_ACK, or WAIT_LAST_PMDLLP, or PREP_4L1, or L1_LINK_ENTR_L1. If the application asserts the `app_xfer_pending` input signal to '1' after the `pm_master_state` =L1, the controller exits ASPM L1 immediately.

Assert the `app_xfer_pending` input signal immediately after the L1 exit (so that the controller does not enter L1 state again unexpectedly) to control the ASPM L1 entry.

Note

- pm_aspm_l1_enter_ready output signal indicates, in real-time, if the controller meets the internal conditions to start L1 entry negotiation.
- app_xfer_pending input signal indicates the application layer status of the pending TLPs.
- pm_aspm_l1_enter_ready and app_xfer_pending signals are not dependent on each other.
- After the application de-asserts app_xfer_pending signal, to ensure that there is no pending transaction and the application is ready for ASPM L1 entry, the controller starts L1 negotiation right after it asserts the pm_aspm_l1_enter_ready signal, given the ASPM L1 entry timer is disabled.
- Controller initiates ASPM L1 entry after both app_xfer_pending signal becomes '0' and pm_aspm_l1_enter_ready signal becomes '1'.

Note

Legacy settings compatibility

- The ASPM_L1_TIMER_ENABLE field of the ACK_F_ASPM_CTRL_OFF register is not compatible with the legacy CX_ASPM_TIMEOUT_ENTR_L1_EN parameter, which must be left at its default value of '1'.
- The application must not set the ASPM_L1_TIMER_ENABLE field of the ACK_F_ASPM_CTRL_OFF register to '0' if you set the CX_ASPM_TIMEOUT_ENTR_L1_EN parameter to '0'.
- When the ACK_F_ASPM_CTRL_OFF register is '0', the application drives the app_req_entr_l1 input signal to '0'.

When the ASPM_L1_TIMER_ENABLE field of the ACK_F_ASPM_CTRL_OFF register is set to '1', and CX_ASPM_TIMEOUT_ENTR_L1_EN parameter is set to '1', the application uses pm_aspm_l1_enter_ready and app_xfer_pending signals to gate or enable ASPM L1 timer:

- Use the pm_aspm_l1_enter_ready output signal to notify that the datapath is idle and the controller is ready to start ASPM L1 timer
- Use the app_xfer_pending input signal from the application to notify that the link can start ASPM L1 timer
- When ASPM L1 timer expires and if the controller still asserts the pm_aspm_l1_enter_ready, the controller enters L1

3.17.1.5 Registers Shadowed in Power Management Controller

Table 3-81 lists the registers shadowed in the power management controller (PMC) and the reason for shadowing them.

Table 3-81 Registers Shadowed in PMC

Register	Field	Description	Reason for Shadowing
DEVICE_CONTROL_DEVICE_STATUS	PCIE_CAP_AUX_POWER_PM_EN	Enable aux power support for a function.	The *PCI Express Base Specification, Revision 6.0, Version 1.0* specifies that the value of this bit must be preserved in the D3cold state. In L2 when power is removed the CDM loses its context. This bit is preserved in the PM under AUX power and fed back to the CDM.
VF_CON_STATUS_REG	PME_ENABLE	Indicates if a function is permitted to generate a PME.	PME message request, if enabled, can trigger a wakeup from L2 if main power has been removed.
VF_CON_STATUS_REG	POWER_STATE	Indicates the D-state of a function.	Used for entry to low power states.
VF_LINK_CAPABILITIES_REG	PCIE_CAP_CLOCK_POWER_MAN	If set to 1 this indicates that an Upstream Port tolerates the removal of reference clock when the link is inL1 or L2/L3 Ready.	Used for the removal of reference clock inL1 or L2/L3 Ready.
AUX_CLK_FREQ_OFF	AUX_CLK_FREQ	The frequency of the aux_clk in MHz.	This field is used to scale certain timers in the power management block. It needs to be shadowed because the CDM is reset when power is removed in L2.
GEN3_RELATED_OFF	GEN3_ZRXDC_NONCOMPL	GEN3 Receiver impedance zrx-dc not compliant.	This bit enables a 100ms timer which can trigger exit from L1.
PHY_INTEROP_CTRL_OFF	P2NOBEACON_ENABLE	P2.NoBeacon enable bit.	The shadowed register field is used to pick the PHY power down encoding; P2.NoBeacon instead of P2.

3.17.2 L1 Substates

This section discusses the L1 link substates. The following topics are discussed:

- "L1.1 and L1.2 Entry and Exit Conditions" on page 269
- "L1 Exit Latency (PIPE 4.2)" on page 275
- "L1 Exit Latency (PIPE 4.3 or Later)" on page 276

The L1 substates are applicable in both the ASPM and PCI-PM L1 link states. L1 substates management utilizes a per-link sideband signal called `CLKREQ#`. The controller supports this feature (with the additional I/O as shown in "Removing the Reference Clock" on page 77) when you set `CX_L1_SUBSTATES_ENABLE =1`.

Note To prevent L1 substates entry you can either perform a configuration write to change the settings of the L1 SubstatesControl1register or set `app_l1sub_disable =1`.

Assumptions and Limitations

- The `CLKREQ#` signal is not directly controlled by the controller. You must implement external logic to do so; see "Removing the Reference Clock" on page 77 for examples.
- During L1 substates it is assumed that `core_clk` is turned off and that `aux_clk` is active. It is required that your application switches `aux_clk` to a low frequency free running clock on entry into L1.
- The controller uses `aux_clk` for counting time during L1 substates. You must program the frequency of this clock into the `AUX_CLK_FREQ_OFF` register with a value in MHz to count real time. Frequencies lower than 1 MHz are possible, but with a loss of accuracy in the time counted.
- When the electrical idle detection circuitry is disabled it is assumed that the PHY holds the signal `phy_mac_rxelecidle` to '1'.

PIPE Operation

The controller supports three interface options for L1 substates.

- PIPE 4.2 + Sideband signals (`CX_PIPE_VER =0`)
  - The controller uses the sideband signals `mac_phy_rxelecidle_disable`, `mac_phy_txcommon-mode_disable`, and `phy_mac_pclkack_n`.
- PIPE 4.3 or later (`CX_PIPE_VER >0`) (PCLK as PHY Output)
  - The controller and PHY use the `mac_phy_powerdown[3:0]` output to indicate the PHY power state.
  - You can specify the power-down encodings using the `CX_PIPE43_P1CPM_ENCODING`, `CX_PIPE43_P1_1_ENCODING`, and `CX_PIPE43_P1_2_ENCODING` parameters.
  - You can specify P1.CPM entry sequence using the `CX_PIPE43_P0_P1CPM` parameter.
  - You can specify P1.1 (P1.2) exit sequence using the `CX_PIPE43_P1CPM_P1` parameter.
  - The sideband signals `mac_phy_rxelecidle_disable` and `mac_phy_txcommonmode_disable` exist on top of the controller. PHY can optionally use these signals.
  - The sideband signal `phy_mac_pclkack_n` does not exist on the top of the controller.
- PIPE 5.0 or later (PCLK as PHY Input)

- The controller and PHY use the `mac_phy_powerdown[3:0]` output to indicate the PHY power state.
- You can specify the power-down encodings using the `CX_PIPE43_P1CPM_ENCODING`, `CX_PIPE43_P1_1_ENCODING`, and `CX_PIPE43_P1_2_ENCODING` parameters.
- You can specify P1.CPM entry sequence using the `CX_PIPE43_P0_P1CPM` parameter.
- The option to set `CX_PIPE43_P1CPM_P1` for P1.1 (P1.2) exit sequence is not supported.
- The sideband signals `mac_phy_rxelecidle_disable` and `mac_phy_txcommonmode_disable` exist on top of the controller. PHY can optionally use these signals.
- The sideband signal `phy_mac_pclkack_n` does not exist on the top of the controller.
- The sideband signals `mac_phy_maxpclkreq_n/phy_mac_maxpclkack_n` exists on top of controllers. They are used for the removal of MAXPCLK in lower power mode.

Note Contact your PHY vendor regarding L1 substates operation because the PIPE specification does not clarify all details.

Figure 3-98 PIPE 4.3 L1 SS Operation (CX_PIPE43_ASYNC_HS_BYPASS =0 / CX_PIPE43_P0_P1CPM =0/ CX_PIPE43_P1CPM_P1 =0) for PCLK as PHY Output

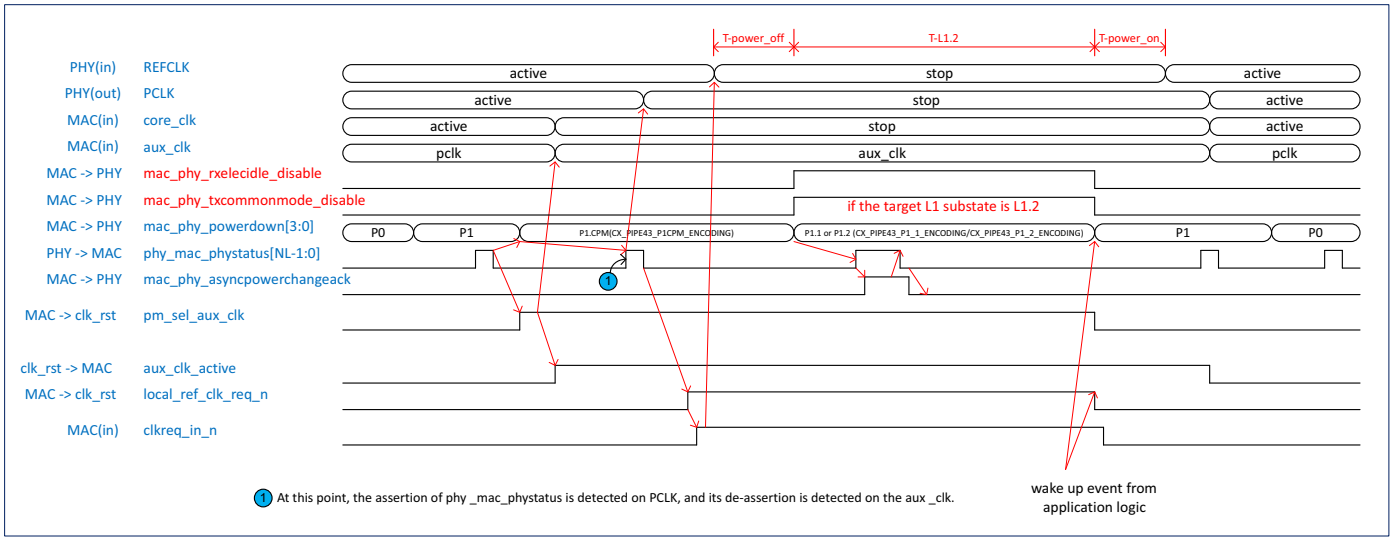


Figure 3-99 PIPE 5.0 L1 SS Operation (CX_PIPE43_ASYNC_HS_BYPASS =0 / CX_PIPE43_P0_P1CPM =0/ CX_PIPE43_P1CPM_P1 =0) for PCLK as PHY Input

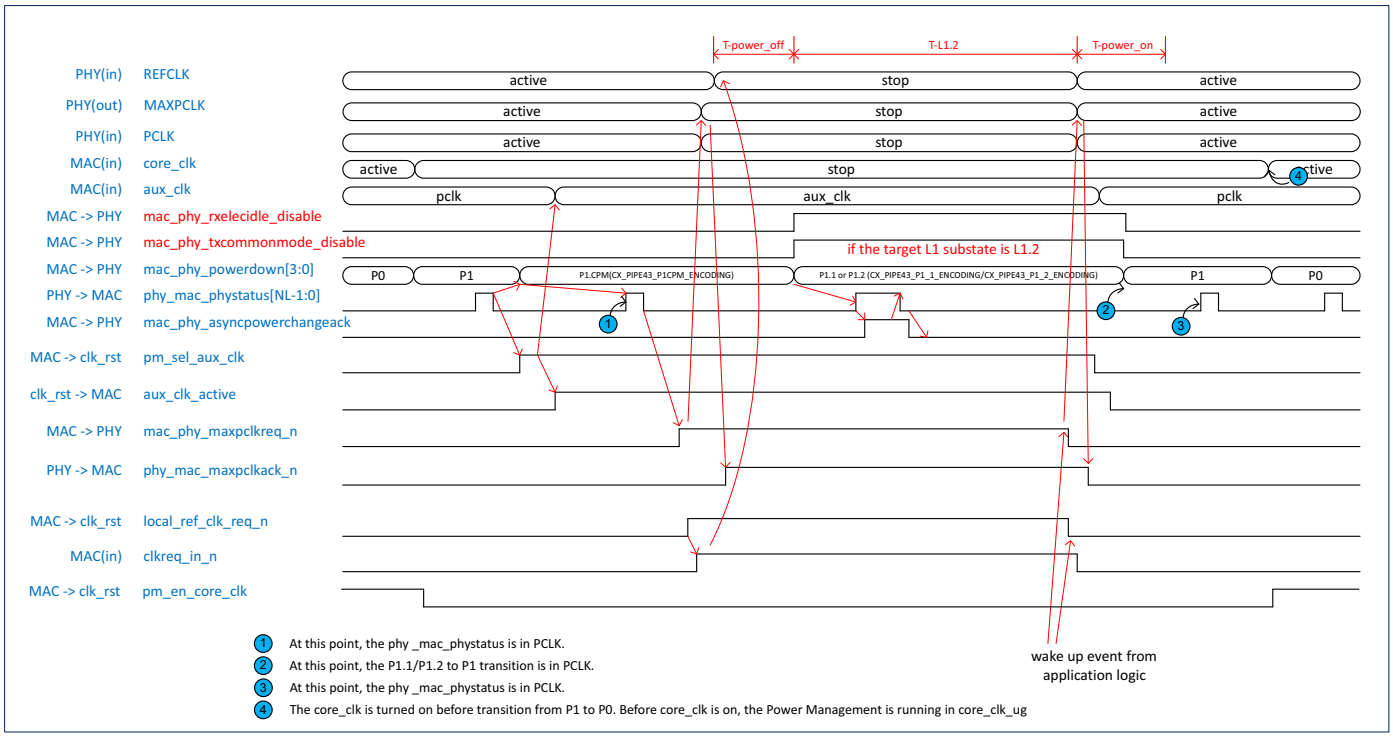


Figure 3-100 PIPE 4.3 L1 SS Operation (CX_PIPE43_ASYNC_HS_BYPASS =0 /CX_PIPE43_P0_P1CPM =1/ CX_PIPE43_P1CPM_P1 =1)

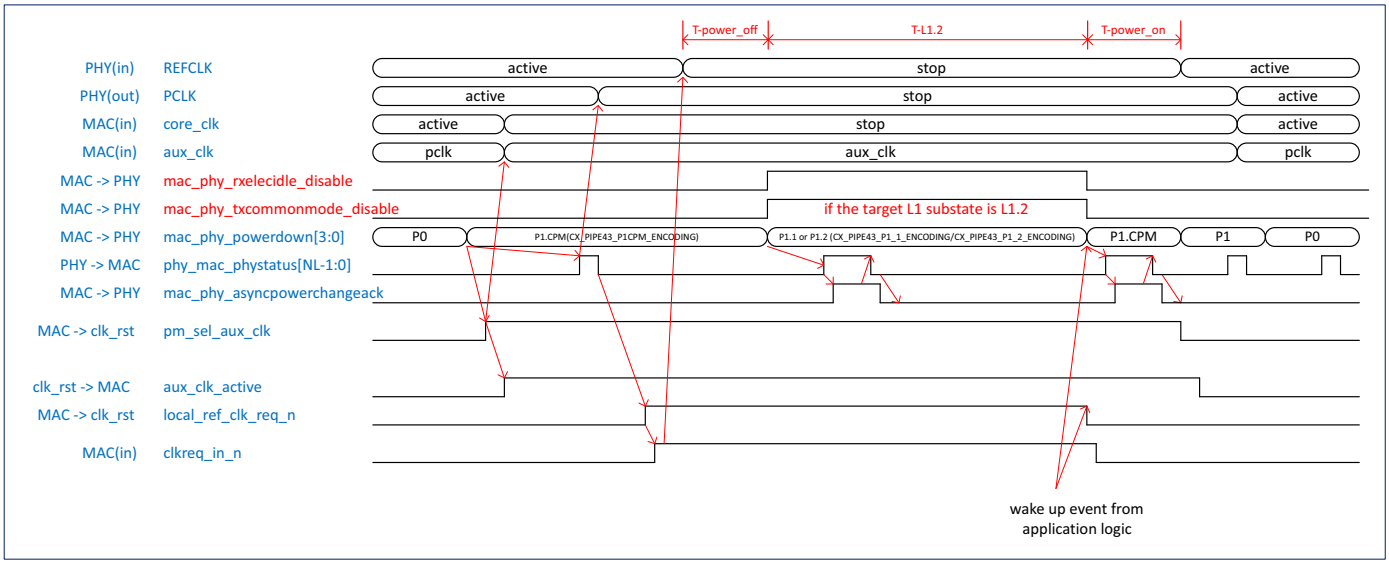
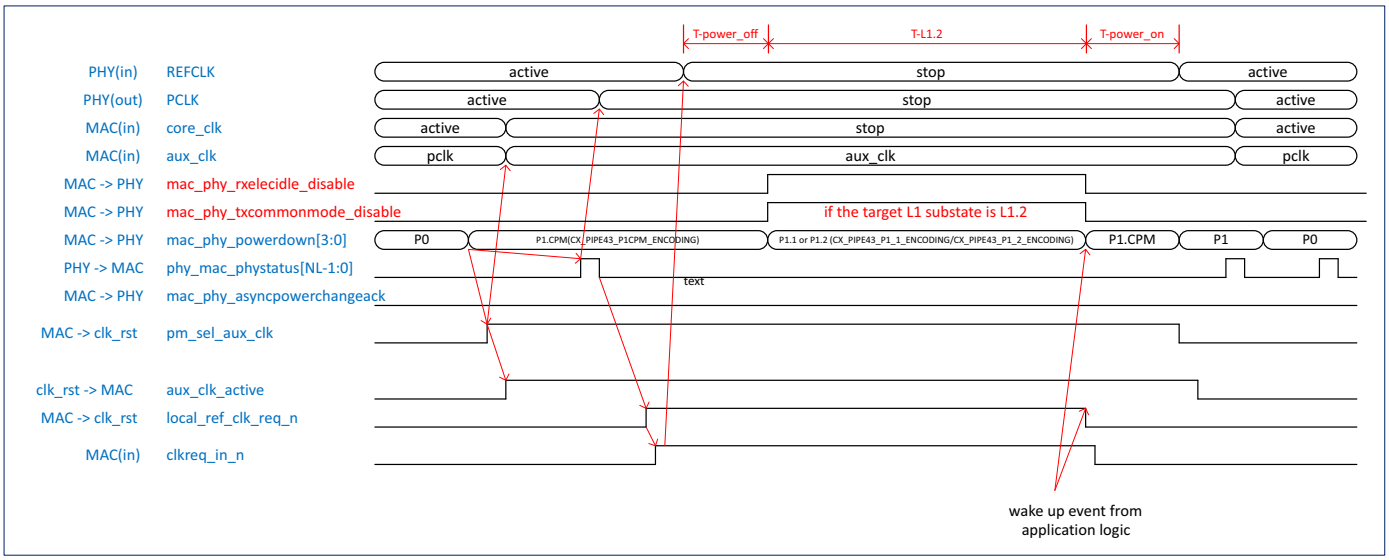


Figure 3-101 PIPE 4.3 L1 SS Operation (CX_PIPE43_ASYNC_HS_BYPASS =1 / CX_PIPE43_P0_P1CPM =1/ CX_PIPE43_P1CPM_P1 =1)



3.17.2.1 L1.1 and L1.2 Entry and Exit Conditions

This section discusses the L1 link state. The following topics are discussed

- "Overview"
- "PIPE 4.2 L1 Substates Entry" on page 270
- "PIPE 4.2 L1 Substates Exit (Locally Initiated)" on page 270
- "PIPE 4.2 L1 Substates Exit (Remotely Initiated)" on page 270
- "PIPE 4.3 or Later L1 Substates Entry" on page 270
- "PIPE 4.3 or Later Substates Exit (Locally Initiated)" on page 271
- "PIPE 4.3 or Later Substates Exit (Remotely Initiated)" on page 271
- "PIPE 5.0 or Later L1 Substates Entry" on page 272
- "PIPE 5.0 or Later Substates Exit (Locally Initiated)" on page 272
- "PIPE 5.0 or Later Substates Exit (Remotely Initiated)" on page 273
- "L1 Substates Software Control" on page 273

3.17.2.1.1 Overview

After the link has entered L1 through the normal L1 negotiation, the USP can initiate the sequence for entering the target L1 substate (L1.1 or L1.2) by tri-stating its `CLKREQ#` output buffer. The entry sequence can only proceed if the DSP is also tri-stating its `CLKREQ#` output buffer, resulting in the bidirectional `CLKREQ#` signal being pulled up to 1. Otherwise `CLKREQ#` remains asserted at 0 and the link state stays in L1. The exit sequence can be initiated by either ports by asserting `CLKREQ#` to 0. For each port there are two cases to consider, the first where the exit is initiated locally, the second where the exit is initiated remotely. The following sections provide details on the signaling that occurs between the PCIe controller as the MAC, the PHY, and the `CLKREQ#` output buffer, for entering and exiting L1 substates. L1 substates management

utilizes a per-link sideband signal called `CLKREQ#`. After all of the entry conditions for a L1 substate are met, the controller asserts the `mac_phy_rxelecidle_disable` output. When the target L1 substate is L1.2, it also asserts the `mac_phy_txcommonmode_disable` output.

3.17.2.1.2 PIPE 4.2 L1 Substates Entry

1. Controller de-asserts `mac_phy_pclkreq_n[1]`
2. PHY de-asserts `phy_mac_pclkack_n` when ready for REFCLK removal, PCLK is off
3. CLKREQ# output buffer tri-stated
4. Controller waits for `clkreq_in_n` to de-assert
5. Controller asserts `mac_phy_rxelecidle_disable` (and `mac_phy_txcommonmode_disable` if the target L1 substate is L1.2)
6. PHY disables receiver Electrical Idle detection, `phy_max_rxelecidle` is maintained stable to 1, if `mac_phy_txcommonmode_disable` =1 the PHY also disables transmitter Common Mode.

Note: If PHY does not de-assert `phy_mac_pclkack_n` within 3us in step 2, controller asserts `mac_phy_pclkreq_n` again.

3.17.2.1.3 PIPE 4.2 L1 Substates Exit (Locally Initiated)

1. Any of the normal L1 exit conditions in "L1-PM/L1-ASPM Exit" on page 286 are met
2. Controller asserts `mac_phy_pclkreq_n[1]`
3. Controller de-asserts `mac_phy_rxelecidle_disable` (and `mac_phy_txcommonmode_disable` if asserted)
4. PHY re-enables receiver Electrical Idle detection, if disabled the PHY also re-enables transmitter Common Mode
5. `CLKREQ#` output buffer is asserted
6. PHY asserts `phy_mac_pclkack_n` when PCLK is on
7. Controller transitions the PHY to P0 in the usual way

3.17.2.1.4 PIPE 4.2 L1 Substates Exit (Remotely Initiated)

1. The remote link partner asserts `CLKREQ#`
2. Controller input `clkreq_in_n` asserts
3. Controller asserts `mac_phy_pclkreq_n[1]`
4. Controller de-asserts `mac_phy_rxelecidle_disable` (and `mac_phy_txcommonmode_disable` if asserted)
5. PHY re-enables receiver Electrical Idle detection, if disabled the PHY also re-enables transmitter Common Mode
6. PHY asserts `phy_mac_pclkack_n` when PCLK is on
7. Controller transitions the PHY to P0 in the usual way

3.17.2.1.5 PIPE 4.3 or Later L1 Substates Entry

1. MAC transitions the PHY to P1
2. PHY asserts `phy_mac_phystatus` for P1 entry

3. Controller transitions the PHY to P1.CPM and asserts `pm_sel_aux_clk` to turn off `core_clk`
4. PHY asserts `phy_mac_phystatus` for P1.CPM entry and CLK_RST asserts `aux_clk_active`
5. Controller de-asserts `local_ref_clk_req_n`
6. `CLKREQ#` output buffer tri-stated
7. Controller waits for `clkreq_in_n` to de-assert
8. Controller transitions the PHY to P1.1 (or P1.2)
9. PHY asserts `phy_mac_phystatus` for P1.1 (or P1.2)
10. Controller asserts `mac_phy_asyncpowerchangeack`
11. PHY de-asserts `phy_mac_phystatus`
12. Controller de-asserts `mac_phy_asyncpowerchangeack`

Note: When `CX_PIPE43_P0_P1CPM` =1, steps 1- 2 are bypassed and when `CX_PIPE43_ASYNC_HS_BYPASS` =1, steps 9 - 12 are bypassed.

3.17.2.1.6 PIPE 4.3 or Later Substates Exit (Locally Initiated)

1. Any of the normal L1 exit conditions in "L1-PM/L1-ASPM Exit" on page 286 are met
2. Controller asserts `local_ref_clk_req_n`
3. `CLKREQ#` output buffer is asserted
4. Controller transitions the PHY to P1.CPM
5. PHY asserts `phy_mac_phystatus` for P1.1 (or P1.2)
6. Controller asserts `mac_phy_asyncpowerchangeack`
7. PHY de-asserts `phy_mac_phystatus`
8. Controller de-asserts `mac_phy_asyncpowerchangeack`
9. Controller transitions the PHY to P1 and de-asserts `pm_sel_aux_clk` to turn on `core_clk`
10. PHY asserts `phy_mac_phystatus` for P1 entry and CLK_RST de-asserts `aux_clk_active`
11. Controller transitions the PHY to P0 in the usual way

Note:

- When `CX_PIPE43_P1CPM_P1` =0, steps 4 - 8 are bypassed and when `CX_PIPE43_ASYNC_HS_BYPASS` =1, steps 5 - 8 are bypassed.
- Step 2 and step 4 happen at the same time. The signal `local_ref_clk_req_n` could be asserted 1 or 2 cycles later than the transition from P1.1 (or P1.2) to P1.CPM. But, it is expected that the Phystatus for the Powerdown transition is responded after the change in `local_ref_clk_req_n`.

3.17.2.1.7 PIPE 4.3 or Later Substates Exit (Remotely Initiated)

1. The remote link partner asserts CLKREQ#
2. Controller input `clkreq_in_n` asserts
3. Controller asserts `local_ref_clk_req_n`
4. Controller transitions the PHY to P1.CPM
5. PHY asserts `phy_mac_phystatus` for P1.1 or (P1.2)
6. Controller asserts `mac_phy_asyncpowerchangeack`

7. PHY de-asserts `phy_mac_phystatus`
8. Controller de-asserts `mac_phy_asyncpowerchangeack`
9. Controller transitions the PHY to P1 and de-asserts `pm_sel_aux_clk` to turn on `core_clk`
10. PHY asserts `phy_mac_phystatus` for P1 entry and `DWC_pcie_clkrst.v` de-asserts `aux_clk_activ`
11. Controller transitions the PHY to P0 in the usual way

Note: When `CX_PIPE43_P1CPM_P1` =0, steps 4 - 8 are bypassed and when `CX_PIPE43_ASYNC_HS_BYPASS` =1, steps 5 - 8 are bypassed.

3.17.2.1.8 PIPE 5.0 or Later L1 Substates Entry

1. MAC transitions the PHY to P1
2. PHY asserts `phy_mac_phystatus` for P1 entry and turns off `core_clk` by de-asserting `pm_en_core_clk`
3. Controller transitions the PHY to P1.CPM and asserts `pm_sel_aux_clk`
4. PHY asserts `phy_mac_phystatus` for P1.CPM entry and CLK_RST asserts `aux_clk_active`
5. Controller asserts `mac_phy_maxpclkreq_n` to initiate MAXPCLK removal
6. The PHY or the PLL control logic turns off the MAXPCLK and asserts `phy_mac_maxpclkack_n`
7. Controller de-asserts `local_ref_clk_req_n`
8. CLKREQ# output buffer is tri-stated
9. Controller waits for `clkreq_in_n` to de-assert
10. Controller transitions the PHY to P1.1 (or P1.2)
11. PHY asserts `phy_mac_phystatus` for P1.1 (or P1.2)
12. Controller asserts `mac_phy_asyncpowerchangeack`
13. PHY de-asserts `phy_mac_phystatus`
14. Controller de-asserts `mac_phy_asyncpowerchangeack`

Note: When `CX_PIPE43_P0_P1CPM =1`, steps 1- 2 are bypassed and when `CX_PIPE43_ASYNC_HS_BYPASS =1`, steps 11 - 14 are bypassed.

3.17.2.1.9 PIPE 5.0 or Later Substates Exit (Locally Initiated)

Any of the normal L1 exit conditions in "L1-PM/L1-ASPM Exit" on page 286 are met

1. Controller asserts `local_ref_clk_req_n`
2. CLKREQ# output buffer is asserted
3. Controller de-asserts `mac_phy_maxpclkreq_n` and requests MAXPCLK
4. The PHY or the PLL control logic turns on the MAXPCLK and de-asserts `phy_mac_maxpclkack_n`
5. Controller de-asserts `pm_sel_aux_clk` and CLK_RST de-asserts `aux_clkactive`
6. Controller transitions the PHY to P1
7. PHY asserts `phy_mac_phystatus` for P1 entry
8. Controller transitions the PHY to P0 in the usual way and asserts `pm_en_core_clk` to enable `core_clk`

Note: P1.1(P1.2) ->P1.CPM->P1 `(CX_PIPE43_P1CPM_P1 =1)` is not supported in PCLK as PHY Input.

3.17.2.1.10 PIPE 5.0 or Later Substates Exit (Remotely Initiated)

1. The remote link partner asserts CLKREQ#
2. Controller input `clkreq_in_n` asserts
3. Controller asserts `local_ref_clk_req_n`
4. Controller de-asserts `mac_phy_maxpclkreq_n` and requests MAXPCLK
5. The PHY or the PLL control logic turns on the MAXPCLK and de-asserts `phy_mac_maxpclkack_n`
6. Controller de-asserts `pm_sel_aux_clk` and CLK_RST de-asserts `aux_clk_active`
7. Controller transitions the PHY to P1
8. PHY asserts `phy_mac_phystatus` for P1 entry
9. Controller transitions the PHY to P0 in the usual way and asserts `pm_en_core_clk` to enable `core_clk`

Note: P1.1(P1.2) ->P1.CPM->P1 `(CX_PIPE43_P1CPM_P1 =1)` is not supported in PCLK as PHY Input.

3.17.2.1.11 L1 Substates Software Control

To facilitate L1 substates entry, your application must,

1. Set `app_l1sub_disable =0`
2. Set `app_clk_pm_en =1` and `PCIE_CAP_EN_CLK_POWER_MAN` field in `LINK_CONTROL_LINK_STATUS_REG =1(only for PIPE 4.3 or later)`
3. Set `app_clk_req_n =1`, to indicate that your application logic is ready to have reference clock removed, and the controller can switch to `aux_clk`

When the controller enters L1, the target L1 substate depends on the following programmable register fields:

- PM Control/Status Register
  - DSP: The DLLP type that is used by USP to request L1
- L1 Substates Control 1 Register
  - ASPM PM L1.1 Enabled
  - ASPM PM L1.2 Enabled
  - PCI PM L1.1 Enabled
  - PCI PM L1.2 Enabled

After the DSP controller enters L1, it uses the USPs DLLP L1 request type to determine if L1 is entered in ASPM mode or PCI-PM mode. Table 3-82 shows the target L1 substate as a function of the relevant programmable register fields.

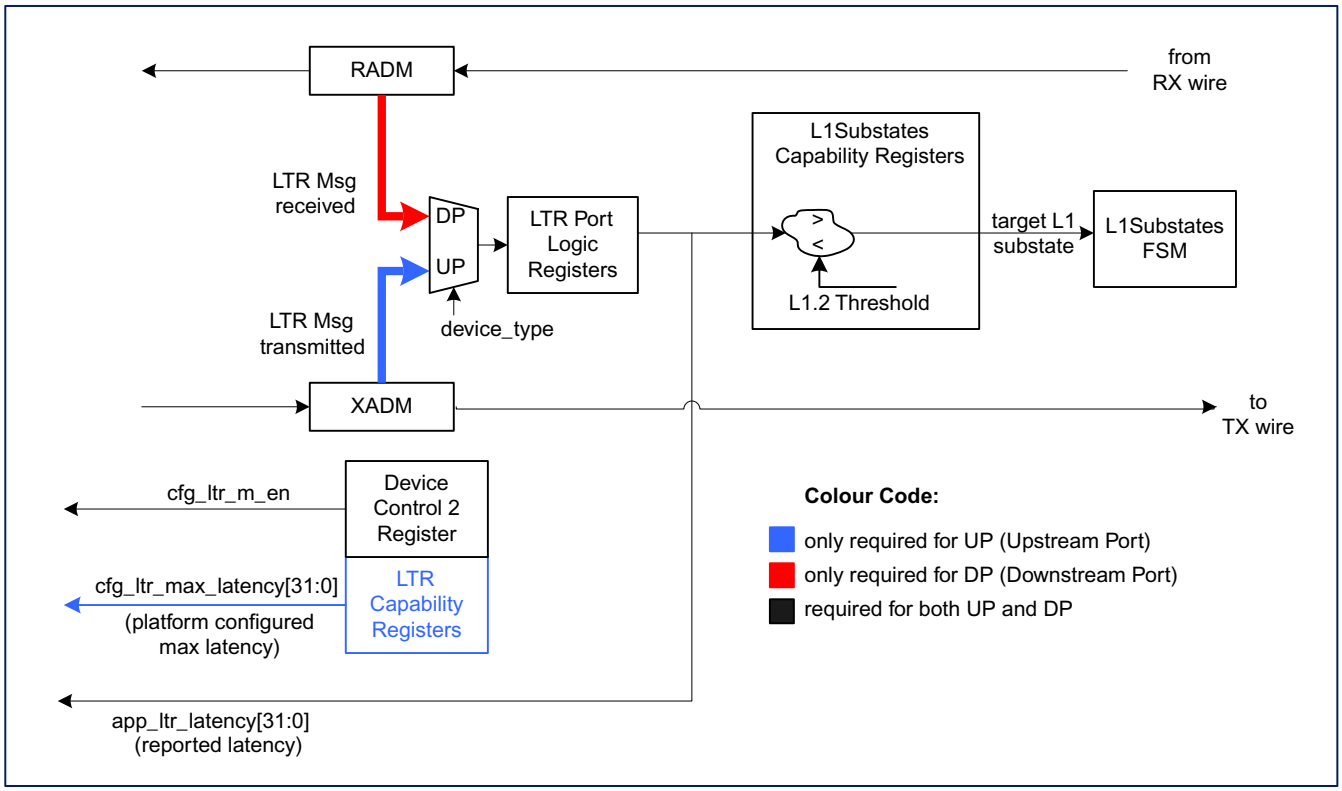
Table 3-82 Target L1 Substate as a Function of Software Controls (DSP)

DLLP Received	ASPM PM L1.1 Enabled	ASPM PM L1.2 Enabled	Reported LTR >= L1.2 Threshold	PCI PM L1.1 Enabled	PCI PM L1.2 Enabled	Target L1 Substate
PM_Enter_L1	-	-	-	0	0	L1
	-	-	-	1	0	L1.1
	-	-	-	-	1	L1.2
PM_Active_State_Request_L1	0	0	-	-	-	L1
	1	0	-	-	-	L1.1
	0	1	0	-	-	L1
	1	1	0	-	-	L1.1
	-	1	1	-	-	L1.2

The Reported LTR is the maximum of the snoop/nosnoop latency values embedded in LTR messages transmitted by the Upstream Ports or received by the Downstream Port. The controller stores these values in the port logic LTR Latency Register (`PL_LTR_LATENCY_OFF`). When the requirement bit in the message is 0, the latency value is considered infinite (that is, the check with the threshold always pass).

For more information on LTR message reception, see "Latency Tolerance Reporting (LTR) Message Reception" on page 208.

Figure 3-102 Role of LTR in L1 Substates



3.17.2.2 L1 Exit Latency (PIPE 4.2)

This section provides a breakdown of the L1 exit latency when `CX_L1_SUBSTATES` is enabled for L1 substates or L1 Clock PM in a PIPE 4.2 system.

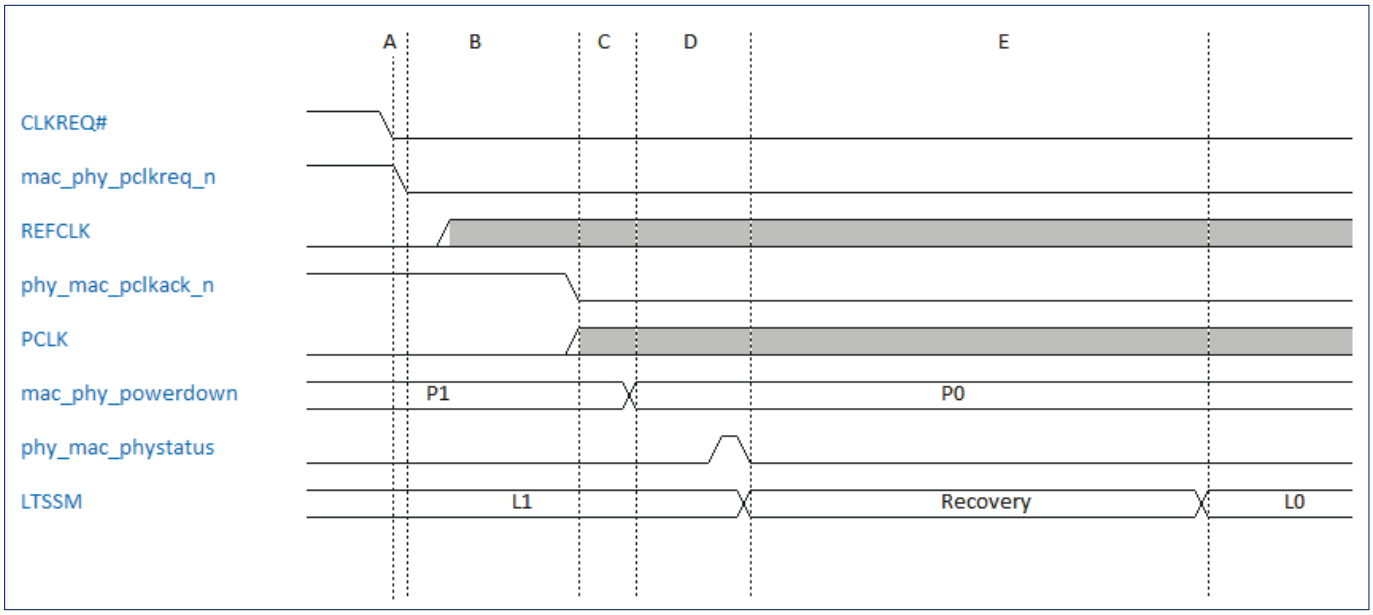
Table 3-83 L1 (PIPE 4.2) Substates and L1 Clock PM Exit Latency Component Values

	A	B	C	D	E
Event in Figure 3-103	CLKREQ# -> mac_phy_pclkreq_n	mac_phy_pclkreq_n -> phy_mac_pclkack_n	pipe_clk stable -> mac_phy_powerdown	mac_phy_powerdown -> phy_mac_phystatus & LTSSM change	Recovery -> L0
Description	0	Time required to produce a stable pipe_clk.	Includes time to switch aux_clk back to pipe_clk and ungating of core_clk.	Time required after pipe_clk is stable to power up circuits for P0	-

	A	B	C	D	E
Values when exiting L1.1	0	400ns to reactivate REFCLK + 15 us	Estimate: 3 aux_clk cycles (slow freq) + 3 core_clk cycles.	Time to P0, ~ 2 us	Time to execute Recovery
Values when exiting L1.2	0	T_power_on to reactivate REFCLK + 15 us			T_common mode[a] (~ 255 us) + time to execute Recovery

a. T_commonmode is only used for DSP.

Figure 3-103 L1 Exit (PIPE 4.2) Latency Components



3.17.2.3 L1 Exit Latency (PIPE 4.3 or Later)

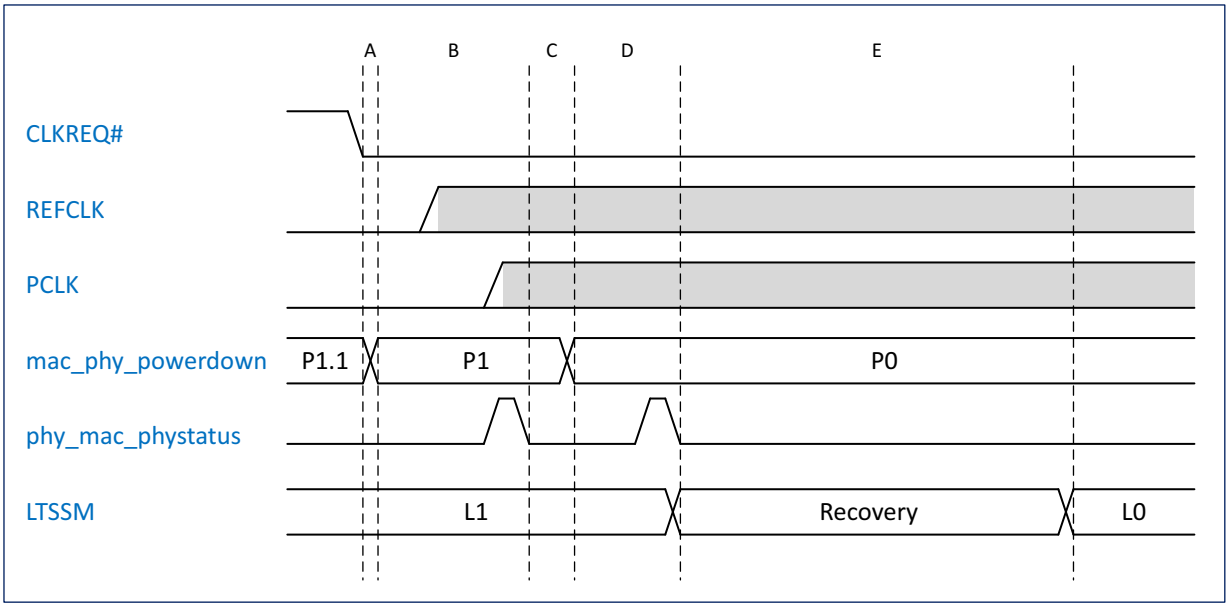
This section provides a breakdown of the L1 exit latency when `CX_L1_SUBSTATES` is enabled for L1 substates or L1 Clock PM in a PIPE 4.3 or later system.

Table 3-84 L1 (PIPE 4.3 or Later) Substates and L1 Clock PM Exit Latency Component Values

	A	B	C	D	E
Event in Figure 3-104	CLKREQ# -> mac_phy_powerdown	mac_phy_powerdown -> phy_mac_phystatus (pipe_clk stable)	pipe_clk stable -> mac_phy_powerdown	mac_phy_powerdown -> phy_mac_phystatus & LTSSM change	Recovery -> L0
Description	0	Time required to produce a stable pipe_clk.	Includes time to switch aux_clk back to pipe_clk and ungating of core_clk.	Time required after pipe_clk is stable to power up circuits for P0	-
Values when exiting L1.1	0	400ns to reactivate REFCLK + 15 us	Estimate: 3 aux_clk cycles (slow freq) + 3 core_clk cycles.	Time to P0, ~ 2 us	Time to execute Recovery
Values when exiting L1.2	0	T_power_on to reactivate REFCLK + 15[a] us			T_common mode (~ 55 us) + time to execute Recovery

a. The 15us time includes the P1.1 to P1.CPM transition time for PIPE 4.4.

Figure 3-104 L1 (PIPE 4.3 or Later) Exit Latency Components



3.17.3 L0s Entry and Exit Conditions

This section discusses the L0s link state. The following topics are discussed:

- "Overview"
- "L0s Entry"
- "L0s Exit"

3.17.3.1 Overview

L0s is a low-power state enabled by ASPM. ASPM controls entry into L0s for the transmitter. The remote device controls entry into L0s for the receiver.

3.17.3.2 L0s Entry

All of these condition must be met:

- ASPM L0s is enabled through the ASPM Control field in the Link Control register.
- L0s entry conditions as defined in Section 5.4.1.1.1, "Entry into the L0s State" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*, exist for a duration of time (determined by the `L0S_ENTRANCE_LATENCY` field in `ACK_F_ASPM_CTRL_OFF`).
- No higher stage of power-down requested.

3.17.3.3 L0s Exit

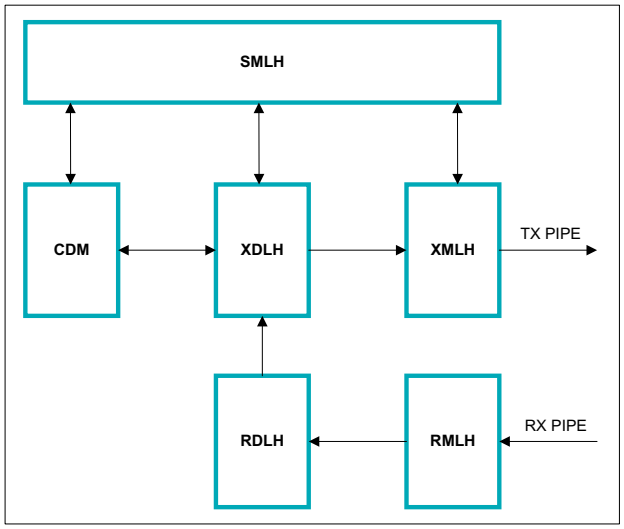
Any of these condition can be met:

- A DLLP or TLP is pending to be sent.
- L1 entry conditions as defined in Section 5.4.1.2.1, "Entry into the L1 State" of the *PCI Express Base Specification, Revision 6.0, Version 1.0* are satisfied.
- PCIe link partner requests to enter into link recovery.

3.17.4 L0p

L0p is a part of the L0 state and intended as a power savings state. The controller supports L0p only in Flit Mode (`CX_FLIT_MODE_ENABLED =1`) according to *PCI Express Base Specification, Revision 6.0, Version 1.0*

Figure 3-105 L0p State Transition Diagram



3.17.4.1 CDM

The following registers are associated for L0p operation. For more information, see "Register Description" chapter of the *Reference Manual*.

- `DEVICE_CAPABILITIES3_REG` (Device 3 Capabilities Register)
- `DEVICE_CONTROL3_REG` (Device 3 Control Register)
- `DEVICE_STATUS3_REG` (Device Status 3 Register)
- `LINK_CONTROL_LINK_STATUS_REG` (Link Control and Link Status Register)
- `PORT_LOGIC_CTRL_2_OFF` (Port Link Control 2 Register)

If the software sets a target link number to the Target Link Width bits in the Device 3 Control register field in LTSSM L0 sate, the controller performs L0p operation. After the L0p operation is completed, the controller changes the Negotiated Link Width bits in Link Control and Link Status Register field to a target link width.

3.17.4.2 XDLH/RDLH

The XDLH/RDLH module receives the L0p requirement from CDM or remote partner.

If the XDLH/RDLH module receives the L0p requirement from CDM:

- Making a Link Management DLLP with the L0p.cmd bits as "L0p Request" and sending it
- Receiving a Link Management DLLP with the L0p.cmd bits as "L0p Request Ack" or "L0p Request Nak"

If the module receives the L0p requirement from remote partner:

- Receiving a Link Management DLLP with the L0p.cmd bits as “L0p Request”
- Sending a Link Management DLLP with the L0p.cmd bits as “L0p Request Ack” or “L0p Request Nak”

The controller responds with an L0p Ack and Nak based on the following rules:

- A Port is permitted to Nak an L0p Request for a lower width than the current width if L0p.Priority is not set.
- If a Port is requesting link width down-size due to thermal throttling or reliability reasons, then it should set L0p.Priority in the L0p Request. The Link partner must Ack priority L0p Requests if it is not requesting an L0p width change or going to request an L0p width change in the next Flit, which must show up in the Transmitter pins in 100 ns with the L0p.Priority set in the L0p Request.
- Any Port requesting a link width down-size that receives an Ack from its Link partner is responsible for initiating the up-size request eventually when the underlying conditions no longer prohibit the link from operating at full width.
- If both Ports are simultaneously requesting link width change the following rules apply to determine which request wins. A Port is permitted to consider a pending request for L0p as simultaneous as long as it has the L0p request scheduled to appear in its Transmitter pins within 100 ns. The Port that does not win must Ack the request to the winning Port through the proper L0p response.
  - If both requests have L0p.Priority set, the one with the lower width wins.
  - If one request has L0p.Priority set, it wins
  - If neither side has L0p.Priority set: the one with the higher width wins; else if both sides are requesting the same width, the Downstream Port wins
- A Port Is permitted to Nak a request for a lower width than the current L0p width if L0p.Priority is not set.
- An entry to Recovery results in the Link going to its maximum configured width.

Note A Link Management DLLP contains a L0p.Priority bit. You must program this bit using CTRL_2_L0P_PRIO field in PORT_LOGIC_CTRL_2_OFF register.

The controller sets the CTRL_2_L0P_PRIO field value to L0p.Priority bit of Link Management DLLP.

3.17.4.3 SMLH

The SMLH module negotiates with the L0p capable bit during Configuration.Complete state when `CX_FLIT_MODE_ENABLE =1` and LinkUp =0. The controller sets the L0p capable bit and the Remote L0p Supported bit in the Device Status 3 Register to ‘1 when:

- The device sends TS2 Ordered Sets with the Link Upconfigure or L0p Capability bit (Symbol 4 bit 6) set to ‘1’, and
- Receives eight consecutive TS2 Ordered Sets with the Link Upconfigure or L0p Capability bit set to ‘1’

3.17.4.4 XMLH/RMLH

The controller changes a Link width dynamic in LTSSM L0 state based on the following rules:

Link width down-size:

Each Port follows the steps independently after the Link width down-configure request has been 'Ack'ed:

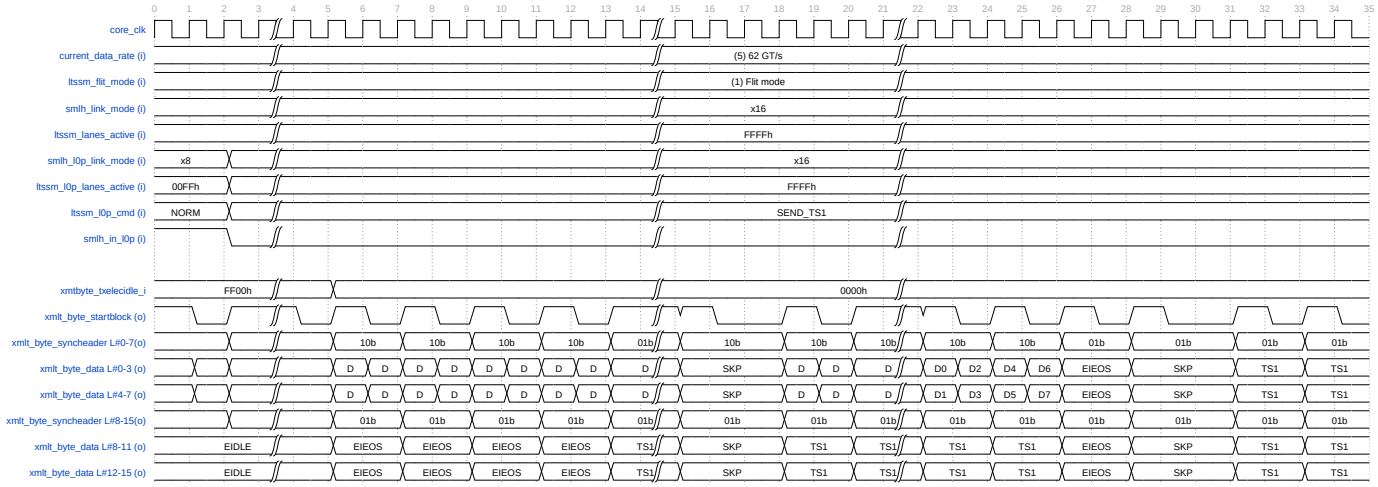
- On the next scheduled SKP Ordered Set interval, the Lanes that are being turned off sends an EIOSQ instead of a SKP Ordered Set
- The Lanes that sent an EIOSQ go to Electrical Idle after the EIOSQ is sent. The Lanes that sent a SKP Ordered Set resume transmitting Flits.
- During the down-size negotiation, an EIOS must be received on all lanes to be deactivated at the same time after adjusting for lane to lane skew; else the LTSSM must enter Recovery.

Link width up-size:

- The up-sizing action must be initiated by the requesting port. The non-requesting Port waits for detection of Electrical Idle exit on the lanes to be activated before starting the up-sizing actions
- Data Stream continues on the active Lanes
- For the Lanes to be activated the following sequence must be followed:
  - SKP Ordered Sets must be scheduled at the same time as with the active Lanes
  - Transmit TS1 Ordered Sets meeting the requirements for TS1 Ordered Sets.
- If eight consecutive TS1 or TS2 Ordered sets are received on all Lanes that are to be activated, the transmitter must transition to sending TS2 Ordered Sets meeting the requirements for TS2 Ordered Sets.
- After receiving eight consecutive TS2 Ordered Sets and sending at least 16 TS2 Ordered sets after the receipt of one TS2 Ordered Set on all the Lanes to be activated, the Port sends an SDS Ordered Set sequence if the data rate is 8.0 GT/s or higher just prior to sending the next scheduled SKP Ordered Set that will be sent (on the active as well as to be activated Lanes).
- Lane to Lane de-skew must be completed by the Receiver using the SKP Ordered Set across all the currently active as well as the Lanes being activated that have received eight consecutive TS2 Ordered Set and the SDS Ordered Set, if the data rate is 8.0 GT/s or higher.

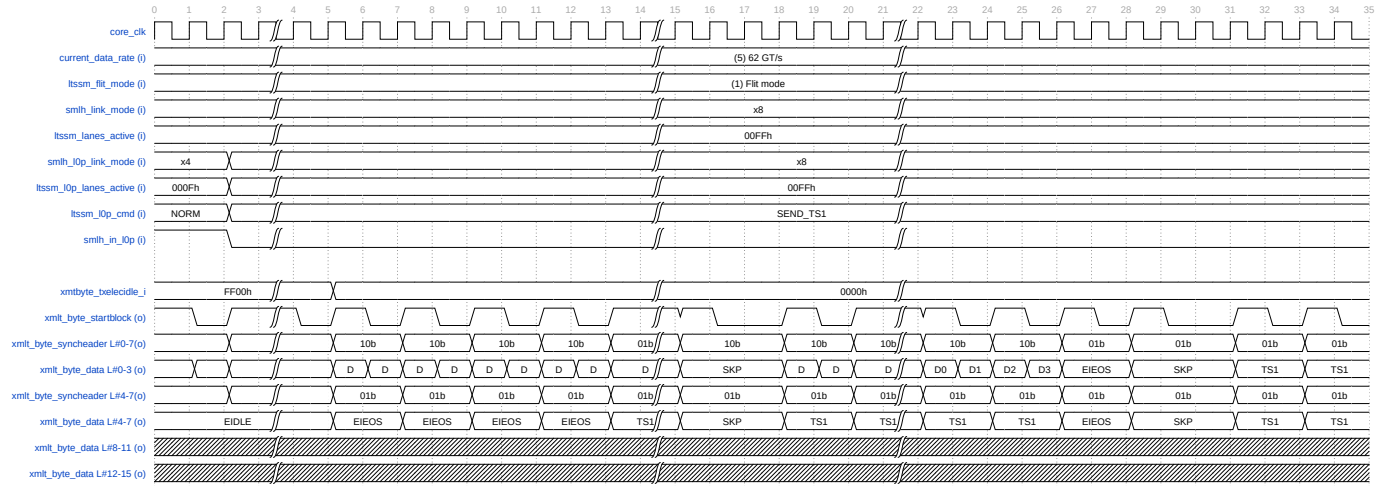
3.17.4.5 Transmitter Operation

Figure 3-106 Tx Transition to Recovery State From x8 to x16 at Gen6 Rate



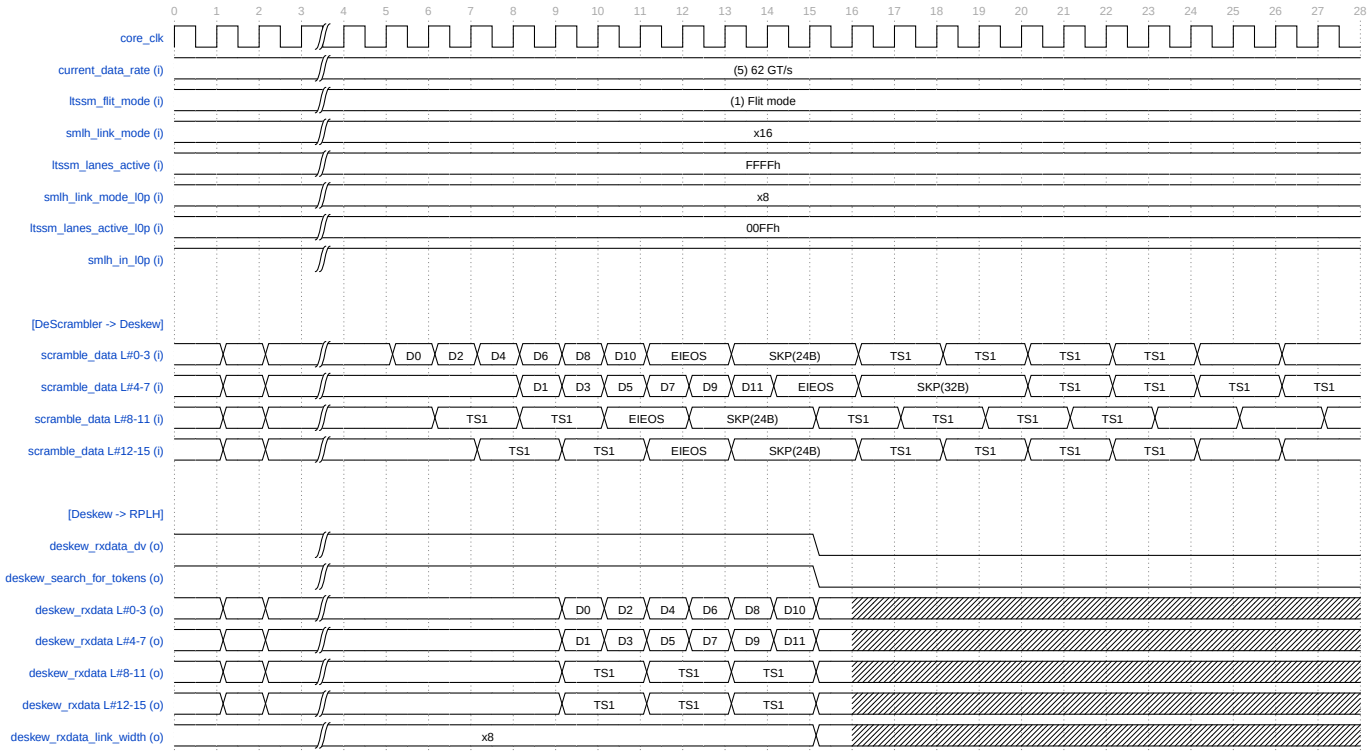
1. At clock cycle 2, the `ltssm_cmd_i` signal is changed from NORM to SEND_TS1
2. At clock cycle 5:
   - Flit data is transmitted with currently link width
   - An EIEOSQ is sent first on next active lanes and followed by TS-OSs
3. At clock cycle 16, an SKP-OS must be sent on all active lanes at same time
4. At clock cycle 26, an EIE-OS is sent on all active lanes at same time
5. At clock cycle 28, an SKP-OS is sent on all active lanes at same time
6. After clock cycle 31, TS-OSs for recovery state are sent

Figure 3-107 Tx Transition to Recovery State From x4 to x8 at Gen6 Rate



3.17.4.6 Receiver Operation

Figure 3-108 Rx Transition to Recovery State From x8 to x16 at Gen6 Rate



1. Deskew module infers link width changes when TS-OS are received on the Lanes to be activated.
2. At clock cycle 15, the `deskew_rxdata_dv` and the `deskew_search_for_tokens` signals are de-asserted if an EIE-OS is received on all active lanes.

3.17.5 L1 Operation (Non-substates)

The following topics are discussed:

- "L1 (ASPM/PM) Entry and Exit Conditions"
- "L1 Clock PM (L1 with REFCLK removal/PLL Off) Overview" on page 287
- "L1 Clock PM (L1 with REFCLK removal/PLL Off); Entry and Exit Conditions" on page 289

3.17.5.1 L1 (ASPM/PM) Entry and Exit Conditions

This section discusses the L1 link state. L1 is a low-power state enabled either by ASPM (L1-ASPM) or by the software changing the D-state (L1-PM). The L1 state is a bi-directional link low-power state and both link partners must negotiate to go to this state.

- "L1-ASPM Overview"
- "L1-ASPM Entry (Scenario 1): L1 Idle Timeout in L0s" on page 284
- "L1-ASPM Entry (Scenario 2): L1 Idle Timeout in L0" on page 285
- "L1-ASPM Entry (Scenario 3): Application Controlled (USP only)" on page 285
- "L1-PM Entry" on page 286
- "L1-PM/L1-ASPM Exit" on page 286

3.17.5.1.1 L1-ASPM Overview

The L1-ASPM entry negotiation handshake uses PM_Active_State_Request_L1 DLLPs, PM_Request_Ack DLLPs, and PM_Active_State_Nak MSG TLPs. There are three scenarios that cause the controller to enter L1 under ASPM conditions.

3.17.5.1.2 L1-ASPM Entry (Scenario 1): L1 Idle Timeout in L0s

The controller enters L1-ASPM when all of these conditions are met in the USP:

- ASPM L1 and L0s are enabled through the ASPM Control field in the Link Control register.
- The `ENTER_ASPM` field of `ACK_F_ASPM_CTRL_OFF` is set to '0' and the link state is L0s for both link partners,

  or

  the `ENTER_ASPM` field of `ACK_F_ASPM_CTRL_OFF` is set to '1'.
- L1 entry conditions as defined in Section 5.4.1.2.1, "Entry into the L1 State" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*, exist for a duration of time (determined by the `L1_ENTRANCE_LATENCY` field in `ACK_F_ASPM_CTRL_OFF`).
- No higher stage of power-down requested.
- There are no in-progress transactions[1] in the controller such as:
  - Rx queue P/NP Request TLPs
  - Rx queue CPL TLPs
  - Outstanding/expected wire completions

1. Created by your application, or remote link partner, or the embedded DMA. DBI requests do not prevent L1 entry because the CDM registers (including DMA) are clocked off aux_clk_g.

- Outstanding/expected application completions at native PCIe controller receive interfaces
- Outstanding/expected application completions at the AXI manager interface, when present

- There are no pending requests at the AXI subordinate interface, that is, `slv_a*valid` must be '0'.
- There are no pending DMA transfers.

3.17.5.1.3 L1-ASPM Entry (Scenario 2): L1 Idle Timeout in L0

The controller enters L1-ASPM when all of these conditions are met in the USP:

- ASPM L1 is enabled and L0s is not enabled through the ASPM Control field in the Link Control register.
- Link state is L0.
- L1 entry conditions as defined in Section 5.4.1.2.1, "Entry into the L1 State" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*, exist for a duration of time (determined by the `L1_ENTRANCE_LATENCY` field in `ACK_F_ASPM_CTRL_OFF`).
- No higher stage of power down-requested.
- There are no in-progress transactions in the controller such as:
  - Rx queue P/NP Request TLPs
  - Rx queue CPL TLPs
  - Outstanding/expected wire completions
  - Outstanding/expected application completions at native PCIe controller receive interfaces
  - Outstanding/expected application completions at the AXI manager interface, when present
- There are no pending requests at the AXI subordinate interface, that is, `slv_a*valid` must be '0'.
- There are no pending DMA transfers.

3.17.5.1.4 L1-ASPM Entry (Scenario 3): Application Controlled (USP only)

The controller enters L1-ASPM when all of these conditions are met in the USP:

- ASPM L1 is enabled through the ASPM Control field in the Link Control register.
- Your application logic pulses the `app_req_entr_l1` input.
- There are no in-progress transactions[1] in the controller such as:
  - Rx queue P/NP Request TLPs
  - Rx queue CPL TLPs
  - Outstanding/expected application completions at native PCIe controller receive interfaces
  - Outstanding/expected application completions at the AXI manager interface, when present
- There are no pending requests at the AXI subordinate interface, that is, `slv_a*valid` must be '0'.
- There are no pending DMA transfers.

1. Created by your application or remote link partner, or the embedded DMA. DBI requests do not prevent L1 entry because the CDM registers (including DMA) are clocked off aux_clk_g.

3.17.5.1.5 L1-PM Entry

The power management state of a link is determined by the D-state of the USP. When you change the device state of the USP to D1, D2, or D3hot by writing to the PMCSR, the controller must initiate a link state transition to L1. L1 state is a bi-directional link low-power state. Both link partners must negotiate to go this state. The L1-PM negotiation handshake uses the following DLLPs:

- PM_Enter_L1
- PM_Request_Ack

All of these condition must be met:

- L1 entry conditions are satisfied as defined in Section 5.3.2.1, "Entry into the L1 State" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

3.17.5.1.6 L1-PM/L1-ASPM Exit

The controller exits L1-PM/L1-ASPM if any of these conditions is met:

- A DLLP or TLP is pending to be sent.
- Link partner is requesting exit from L1.
- Your application asserts the `app_xfer_pending` input.
- Your application is requesting to send traffic by asserting `client0_tlp_hv` (non-AXI).
- Your application is requesting to send traffic by asserting `slv_a*valid` (AXI).
- Your application enables a DMA read or write channel, or DMA controller is requesting data.
- Your application (DSP) initiates link disable, or link retrain (by setting `PCIE_CAP_LINK_DISABLE` or `PCIE_CAP_RETRAIN_LINK` field in `LINK_CONTROL_LINK_STATUS_REG` to 1).
- Your application (DSP) initiates hot reset by either:
  - setting `RESET_ASSERT` field in `PORT_LINK_CTRL_OFF` to 1, or
  - setting `SBR` field in `BRIDGE_CTRL_INT_PIN_INT_LINE_REG` to 1, or
  - toggling `app_init_rst`
- Your application (DSP) requests transmission of Unlock message.
- Your application requests a speed change (by setting `DIRECT_SPEED_CHANGE` field in `GEN2_C-TRL_OFF` to 1).
- Your application requests link width change (by setting `DIRECT_LINK_WIDTH_CHANGE` field in `MULTI_LANE_CONTROL_OFF` is set to 1).
- Any activity in the RADM, as indicated by `radm_idle` being set to 1'b0.

Note When you trigger the FLR during L1-PM, the function exits from L1 through `radm_idle`, while the Link tries to enter L1 again if the D state is non-D0 state. Ensure that the function exits from non-D0 state before you trigger the FLR. Otherwise, the Link iterates between L1-PM entry and L1-PM exit during the FLR process.

Note

L1 Exit

- The CDM space is accessible in L1, but some CDM register bits do not produce the desired results.
- Writing to CDM register bits which trigger LTSSM state transitions in L1 does not have the desired effect because the LTSSM clock is not running in L1.
- `radm_idle` is set to 1'b0 if `ADV_ERR_CAP_CTRL_OFF[ECRC_CHK_EN]` is set to 1'b1 during L1.

Caution

DMA Doorbell L1

- Any DMA register can be accessed in L1 because the DMA registers are clocked on `aux_-clk_g`.
- When `core_clk` is removed in L1 substates or L1 clock power management, `aux_clk_g` is used.

DMA Doorbell L1 Exit

- Doorbelling any DMA read/write channel triggers L1 exit.
- Writing to other DMA registers does not trigger L1 exit.

3.17.5.2 L1 Clock PM (L1 with REFCLK removal/PLL Off) Overview

This feature allows you to shut down the PLL and the reference clock during standard (non-substates) L1. Table 3-85 lists the inputs and outputs that are considered for L1 Clock PM.

Table 3-85 Inputs and Outputs to Consider for L1 Clock PM

CX_L1_SUBSTATES_EN	CX_PIPE_VER	I/O Signals
0	0	`clk_req_n` `phy_clk_req_n`

	1	`clk_req_n`
		`mac_phy_powerdown[3:0] =CX_PIPE43_P1CPM_ENCODING`
		`phy_mac_phystatus`

1	0	`mac_phy_pclkreq_n[0]` `phy_mac_pclkack_n`
	1	`mac_phy_powerdown[3:0] =CX_PIPE43_P1CPM_ENCODING` `phy_mac_phystatus`

For PIPE 4.3 (`CX_PIPE_VER` =1),

- The controller and PHY use `mac_phy_powerdown[3:0]` to indicate the PHY power state.
- You can specify the power-down encodings using the `CX_PIPE43_P1CPM_ENCODING` parameter.
- You can specify P1.CPM entry sequence using the `CX_PIPE43_P0_P1CPM` parameter.

For a USP, host software uses the Enable Clock Power Management bit in the Link Control register to enable this feature. For a DSP, this register bit is hard coded to '0' and cannot be used to control this behavior. Your application can use the `app_clk_pm_en` input to dynamically control whether to execute L1 with or without Clock PM. The `CLKREQ#` signal is not directly controlled by the controller. You must implement external logic to do so; see "Removing the Reference Clock" on page 77 for examples.

Figure 3-109 PIPE 4.3 L1 CPM Operation (CX_PIPE43_P0_P1CPM =0) for PCLK as PHY Output

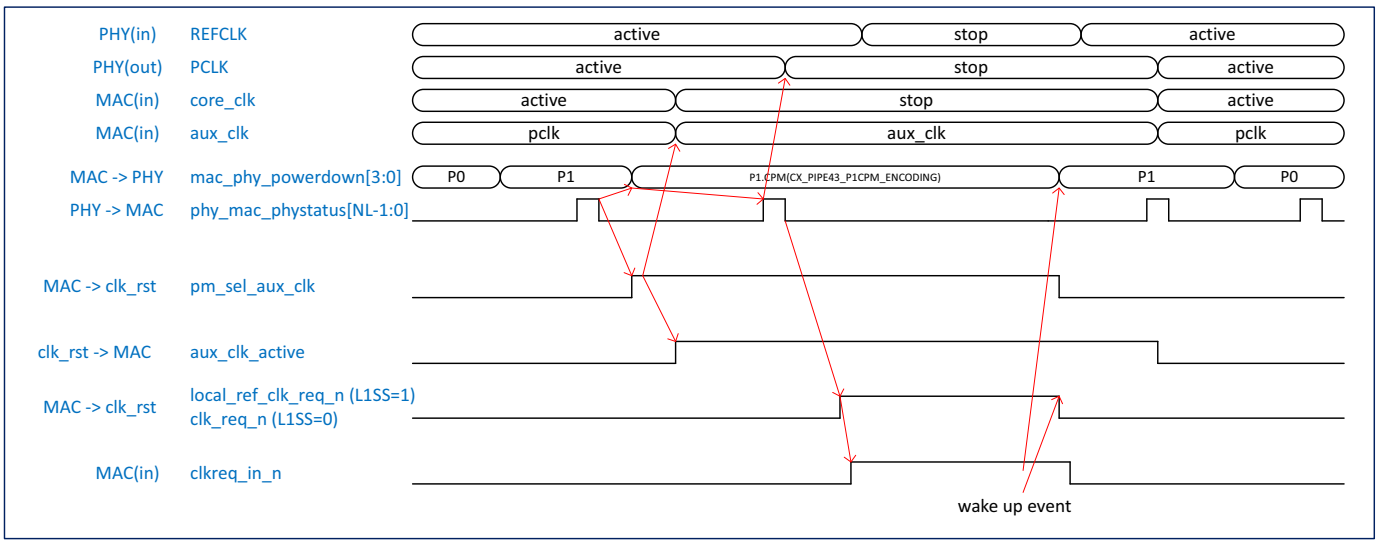


Figure 3-110 PIPE 5.0 L1 CPM Operation (CX_PIPE43_P0_P1CPM =0) for PCLK as PHY Input

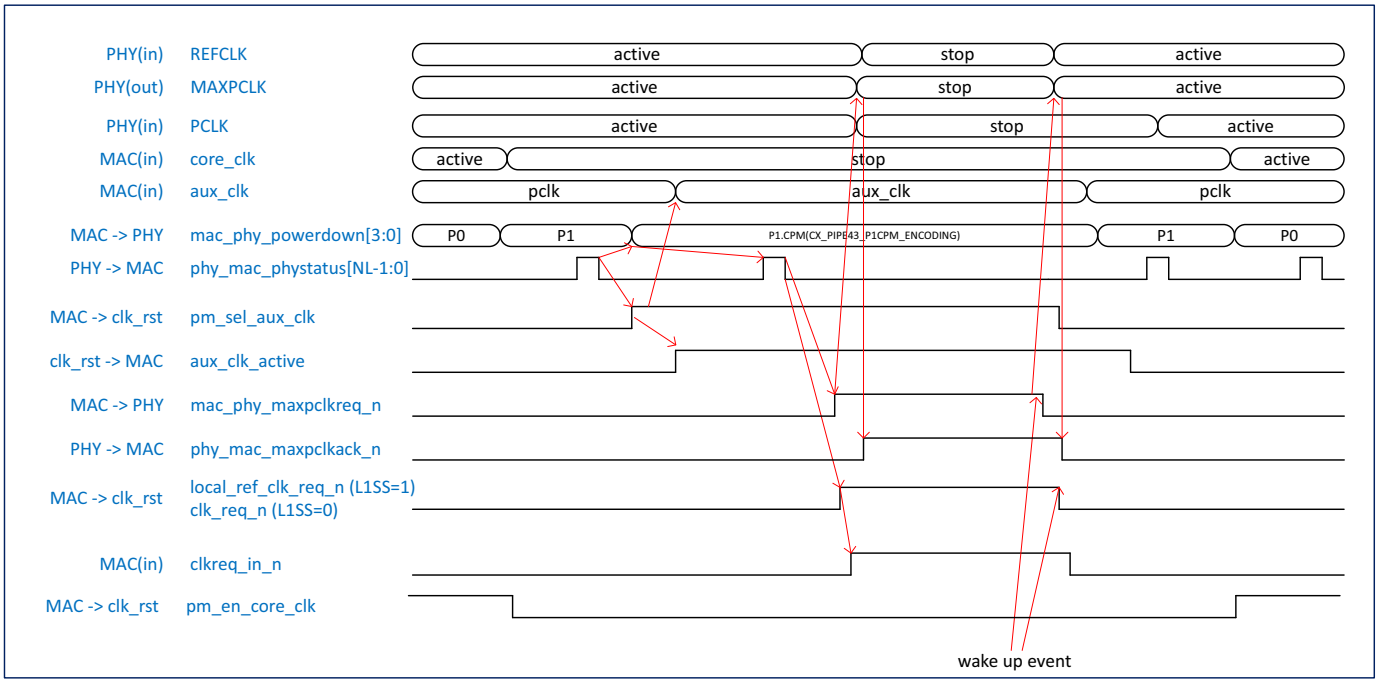


Figure 3-111 PIPE 4.3 L1 CPM Operation (CX_PIPE43_P0_P1CPM =1) for PCLK as PHY Output

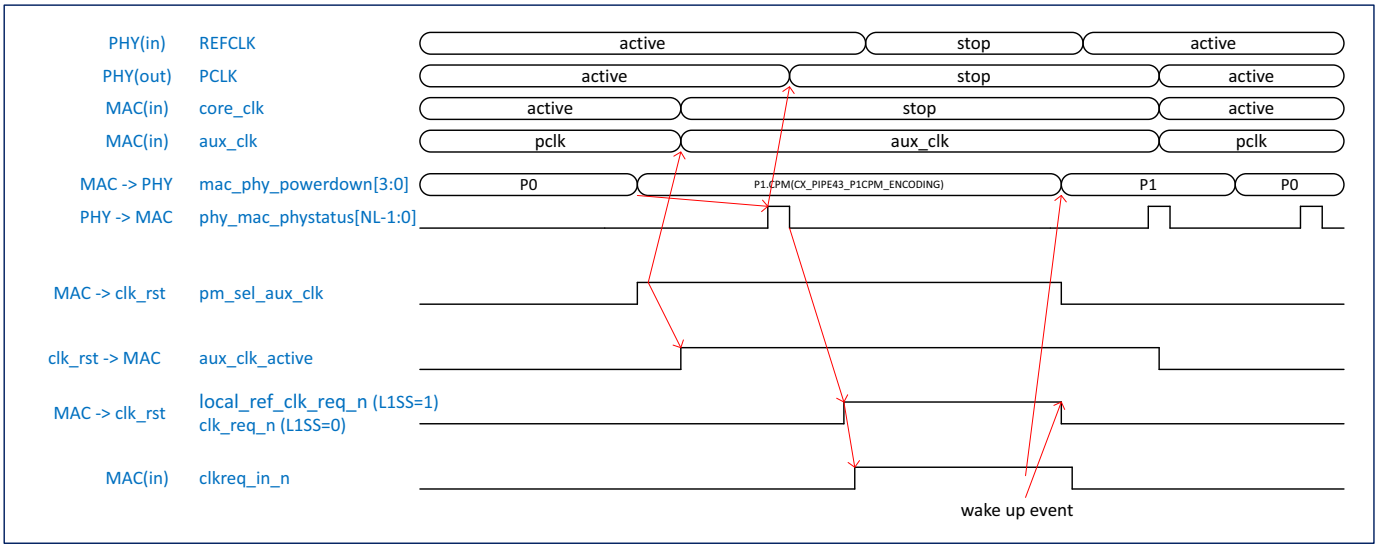
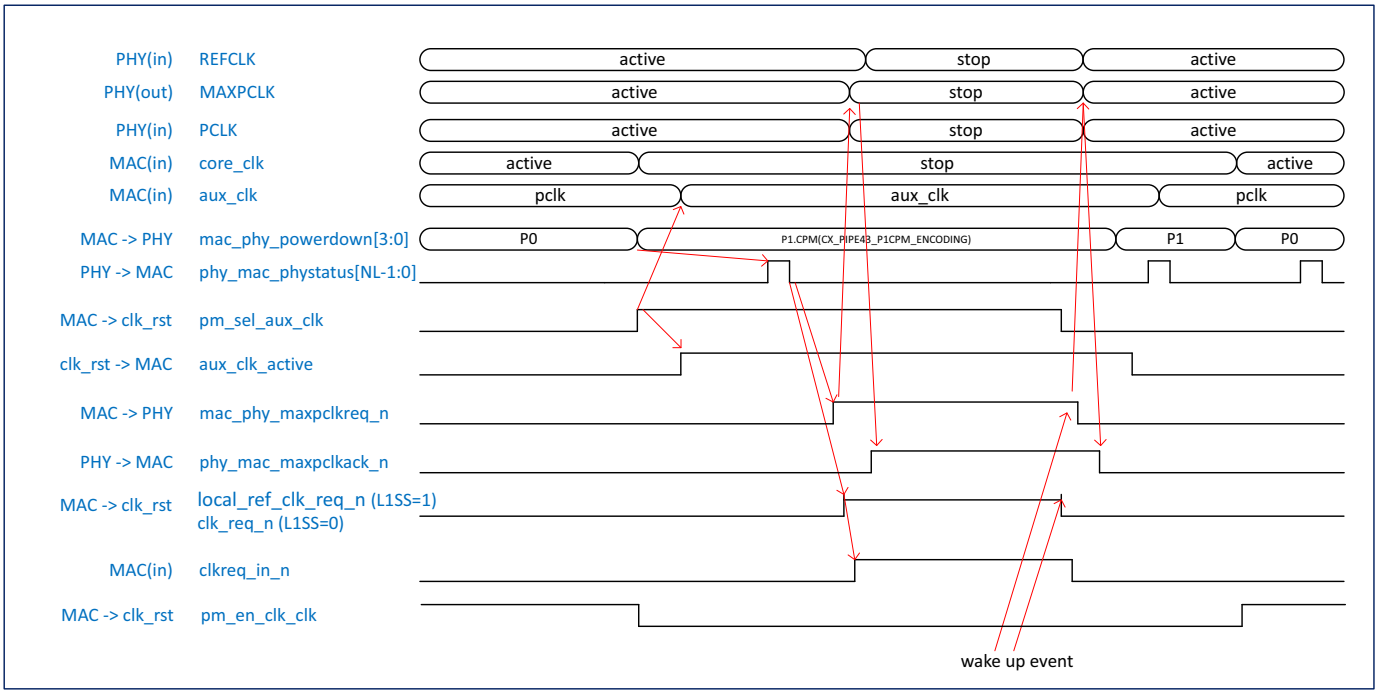


Figure 3-112 PIPE 5.0 L1 CPM Operation (CX_PIPE43_P0_P1CPM =1) for PCLK as PHY Input



3.17.5.3 L1 Clock PM (L1 with REFCLK removal/PLL Off); Entry and Exit Conditions

To enable the controller (Upstream Port only) to execute L1 with Clock PM:

- The Support Clock Power Management bit in the Link Capabilities register must be set. The default value of this register is controlled by the `DEFAULT_CLK_PM_CAP` parameter. For Downstream Ports it is hardcoded to 0, for Upstream Ports it can be accessed through the DBI.
- The Enable Clock Power Management bit in the Link Control register must be set.
- You must set the `app_clk_pm_en` input to 1. The controller only samples `app_clk_pm_en` when L1 is entered.

L1 with Clock PM and L1 substates work orthogonal to each other. L1 with Clock PM uses the `mac_phy_pclkreq_n[0]` signaling, and L1 substates uses the `mac_phy_pclkreq_n[1]`signaling. However, L1 substates takes precedence over Clock PM within the controllers PM state machine. This means that when the entry conditions for any L1 substate are satisfied (as per "L1 Substates Software Control" on page 273), then the controller's LTSSM executes the corresponding L1 substate protocol.

For more information on L1 exit latency, see "L1-PM/L1-ASPM Exit" on page 286.

3.17.6 L2 and L3 Power Down Entry and Exit Conditions (USP)

This section discusses the L2 and L3 link states. The following topics are discussed:

- "Overview"
- "L2/L3 Entry"
- "L2/L3 Exit"

3.17.6.1 Overview

L2/L3 entry is initiated[1] after the RP calls power management software to initiate the removal of power and clocks. USPs of devices in D0, D1, D2, and D3hot must respond to the receipt of a PME_Turn_Off MSG TLP by transmitting a PME_TO_Ack MSG TLP. The device must then request a link transition to L2/L3_Ready. L2/L3_Ready is a bi-directional link power down state. If your application is not ready to be shut-down, it must keep the `app_ready_entr_l23` input de-asserted. This causes the controller to delay sending the PM_Enter_L23 DLLP and thereby stalling the negotiation handshake that uses the following DLLPs:

- PM_Enter_L23
- PM_Request_Ack

When your application is eventually ready for transitioning to D3cold, that is, loss of main power and reference clock, L2/L3_Ready is entered and the downstream device begins preparation for the power and clock removal. After main power has been removed, the link transitions to L2 if Vaux is provided, or it transitions to L3 if no Vaux is provided.

3.17.6.2 L2/L3 Entry

All of these condition must be met:

- PME_Turn_Off/PME_TO_Ack handshake has been completed.
- Your USP application is ready to be turned off; `app_ready_entr_l23 =1`.
- After sending the PME_TO_Ack, the USP initiates the L2/L3 Ready transition protocol by sending the PM_Enter_L23 DLLP. The RP responds with the PM_Request_Ack.

3.17.6.2.1 PME_Turn_Off Handshake Control

You can control the L2 entry negotiation using the `CX_APP_CTRL_PME_TO_ACK` parameter.

1. In preparation for removing the main power source, your RP application asserts the apps_pm_xmt_turnoff input which causes the RP to broadcast the PME_Turn_Off MSG TLP to all USPs.

Figure 3-113 PME Control State Machine When CX_APP_CTRL_PME_TO_ACK =1

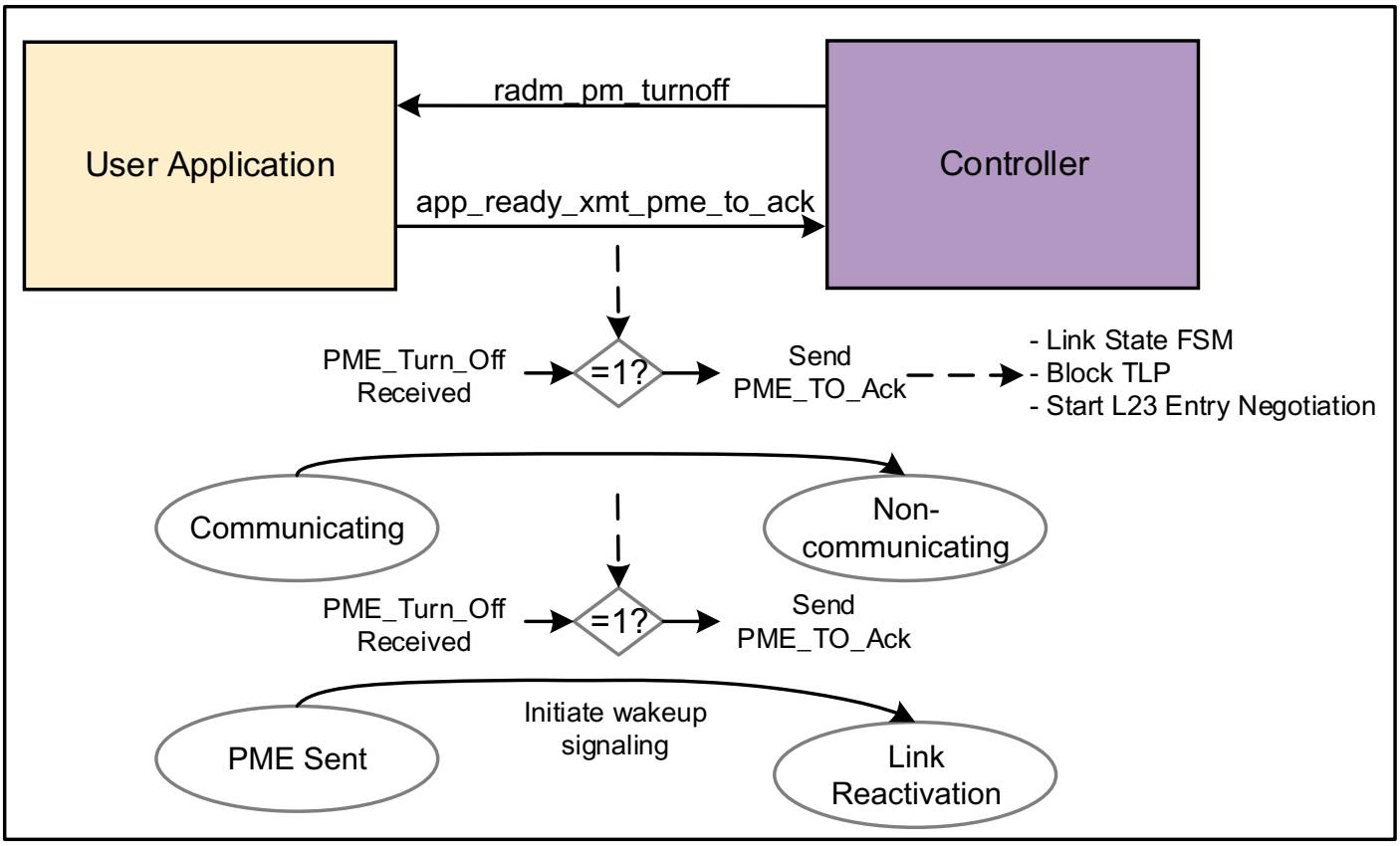
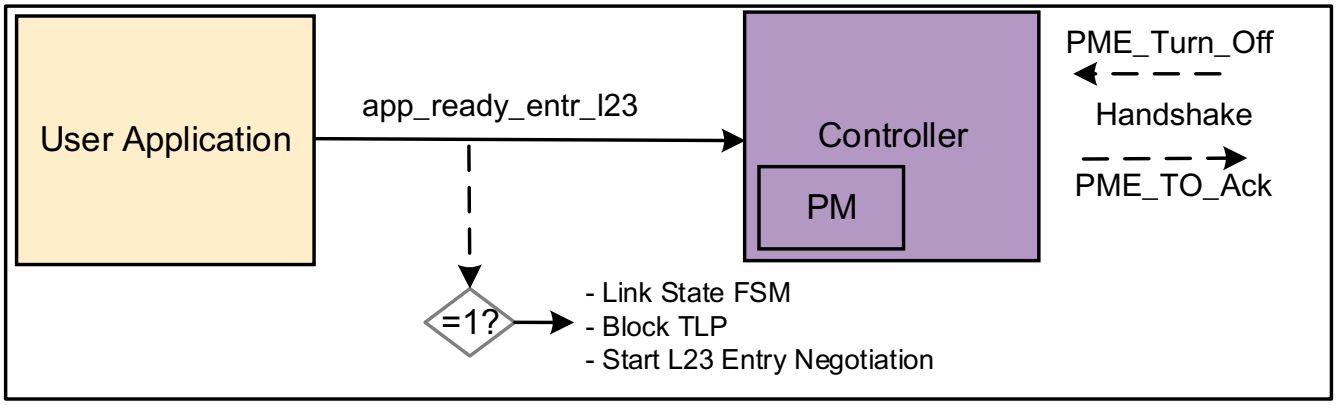


Figure 3-114 PME Control State Machine When CX_APP_CTRL_PME_TO_ACK =0



When the `CX_APP_CTRL_PME_TO_ACK` parameter is set to '1', the application controls PME_Turn_Off and PME_TO_Ack handshake as follows:

1. The `radm_pm_turnoff` output signal indicates when the Upstream Port receives a PME_Turn_Off message from the remote link partner.
2. The `app_ready_xmt_pme_to_ack` input signal controls the PME_Turn_Off and PME_TO_Ack handshake if certain tasks must be performed before going to L23.
3. The controller sends the PM_TO_Ack in response to PM_Turn_Off after `app_ready_xmt_pme_to_ack` input signal is asserted to '1'.

4. After the application asserts `app_ready_xmt_pme_to_ack` input signal, it must be kept asserted until L2 entry has completed.

Note

L2 entry is considered to be completed if any of the following conditions are met:

- smlh_ltssm_state =L2_IDLE or
- pm_master_state =L23RDY or
- pm_linkst_in_l2 =1

Constraint:

The `app_ready_xmt_pme_to_ack` signal is exclusive to `app_ready_entr_l23`. When you set the `CX_APP_CTRL_PME_TO_ACK` parameter to '1', `app_ready_xmt_pme_to_ack` is provided to the application. When the `CX_APP_CTRL_PME_TO_ACK` parameter is set to '0', `app_ready_entr_l23` is provided to the application.

Caution

After the controller asserts the radm_pm_turnoff output signal, application must assert the app_ready_xmt_to_ack input signal within 1 ms. If this is violated, the Downstream Port might enter L2 state without waiting for the Upstream Port to enter the L2/L3 Ready state, thus powering-off VMAIN. The Endpoint behavior in this circumstance is undefined.

3.17.6.3 L2/L3 Exit

Any of these condition can be met:

- When the USP is programmed[1] with capability to support PME, your application can assert the `apps_pm_xmt_pme` input to request the controller to wakeup. The USP then sends a PM_PME MSG TLP to the RP which calls the PM software to transition the USP out of the D3 state.
- Device is programmed with capability to support PME and your application requests the controller to wakeup by triggering a native hot-plug event.
- Link partner is requesting exit from L2/L3.

The controller directs the PHY to generate beacon signaling when a PCIe device initiates a wake-up event.

1. PME_ENABLE bit is set in CON_STATUS_REG, and the PME_SUPPORT bit is set in CAP_ID_NXT_PTR_REG for the corresponding D-state for which the function is currently in.

3.17.7 Advanced Power Management and Power Domain Gating

This section discusses advanced power management and power gating. The following topics are discussed:

- "Advanced Power Management Overview"
- "C-PCIe System-Initiated L2 Power Gating" on page 297
- "C-PCIe L2 Power Gating + Autonomous L1.2 Power Gating (No Retention Registers)" on page 300
- "C-PCIe L2 Power Gating + Autonomous L1.2 Power Gating (Using Retention Registers)" on page 303
- "C-PCIe Autonomous L1.2 Power Gating (No Retention Registers)" on page 306
- "C-PCIe Autonomous L1.2 Power Gating (Using Retention Registers)" on page 308
- "UPF Files" on page 310
- "P2.NoBeacon Support" on page 310

3.17.7.1 Advanced Power Management Overview

This feature allows you to implement power domain gating in your SoC design. It provides all the necessary clock, reset, and power gating controls.

Table 3-86 Advanced Power Management Features and Assumptions

Description	Supported
C-PCIe Features	
System-initiated L2 Power Gating ■ Power is removed from all of the controller (including all CDM registers) except the Power Management Controller (PMC) and DWC_pcie_clkrst.v.	✓
Autonomous L1 Power Gating	✗
Autonomous L1.1 Power Gating	✗
Autonomous L1.2 Power Gating Using Power Islands	✓
Autonomous L1.2 Power Gating Using Retention Registers ■ (Requires Specific Technology Library)	✓
Optional Wait for External Application/PHY to Save Any State Before Power Removal in L1.2 ■ The PHY or your application can use the pm_save_state_reqand pm_restore_state_req outputs to retain any useful information in the always-on memory. For example, the PHY configuration registers.	✓
Prevent L1.2 Power Removal by Setting app_l1_pwr_off_en =0	✓
PIPE Status Frozen Using Controller Logic (no Change in Value) in L1.2 ■ The controller freezes all PIPE outputs at the values prior to power removal using shadow registers in the RTL. This includesGen3 equalization data. The controller does not redo Gen3 equalization after restoration of power.	✓

Description	Supported
FSM To Control Isolation, Clock and Reset Generation, Power Sequencing	✓
Override All Isolation in Test Mode by Setting test_bypass_lp =1	✓
Power Removal Conditions	
Power Removal in L1.2 Waits for Pending Transactions to Complete The controller does not request power removal in L1.2 (and subsequently L1.2 and L1.2 power removal) if: ■ There are in-progress transactions in the controller (including the AXI bridge), or ■ There are outstanding/expected completions from the wire or application; as indicated by the Rx Completion LUT and Target Completion LUT respectively, or ■ DMA is enabled In addition, the AXI bridge subordinate temporarily de-asserts its ready signals when entering the power removal phase. **Note**: DBI requests do not prevent L1 entry because the CDM registers (including DMA) are clocked off aux_clk_g. However, outstanding DBI transactions forces the controller to keep the link in L1 (with core_clk gated off) and power gating deactivated.	✓
Power Restoration Conditions	
Power Restoration in L1.2 ■ The controller does not restore power until it is required to exit L1.2 (as determined by the criteria in "L1-PM/L1-ASPM Exit" on page 286)or there is a DBI access as described in the next row.	✓
Power Restoration in L1.2 Directly Triggered by DBI Access to CDM. ■ DBI access to the CDM registers (which are on aux_clk_g) when the controller is L1.2 power-gated causes the controller to exit power-gating without exiting L1.2 and waking up the link. When DBI access is completed, power-gating is resumed. DBI access does not cause the controller to exit L1.2 (or L1).	✓
Power Restoration in L1.2 Indirectly Triggered by Application Activity Causing L1.2 Exit ■ The controller does not restore power until it is required to exit L1.2. The controller exits L1.2 (and L1) when your application is requesting to send traffic by asserting client0_tlp_hv (non-AXI) or slv_a*valid (AXI). This consequently triggers restoration of power. For more information, see "L1-PM/L1-ASPM Exit" on page 286.	✓
Power Restoration in L1.2 Indirectly Triggered by SII Activity Causing L1.2 Exit ■ The controller does not restore power until it is required to exit L1.2. Activity on any of the SII or ancillary interfaces does not trigger L1.2 exit. Therefore activity on any of the SII or ancillary interfaces do not trigger restoration of power.	✗
Assumptions/Limitations	
PHY Autonomously Generates its Own Reset When Exiting L1.2 Power Gating ■ The controller does not reset the PHY when exiting L1.2 power gating.	✓
PCIe PHY Autonomously Saves State Internally ■ The controller does not save/restore any PHY configuration information.	✓

Description	Supported
Power-gating Interoperability with PHYs Using L1 Clock Power Management (CPM) ■ L1.CPM with power-gating is not supported if the PHY is power-gating its receiver electrical idle detection circuitry in that state; this causes violations of the base specification as exit is not possible with RxElecIdle de-assertion.	✘
UPF Features	
Configuration-specific UPF File Provided	✓
PHY UPF File Provided (Example) ■ Sample UPF assumes PHY is off in L2 ■ Sample UPF assumes PHY is on in L1.2	✓
SoC Integration UPF File Provided (Example)	✓
Optional Level Shifting on Always-on PD_VAUX (PMC) Boundary in L2	✓
Powered-Down Outputs Isolated (Not in Controller but at System/Device Level in UPF)	✓
Powered-Down Inputs from PHY Clamped in Controller	✓
VTB Simulation Support	✓
Retention Registers Supported	
L1.2 Retention Register Type ■ 2-Pin Save/Restore ■ 1-Pin Save/Restore ■ 1-Pin Save-Only ■ 1-Pin Restore-Only ■ 0-Pin	✓ ✓ ✓ ✘ ✘

Hint You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html` for programming examples. For more information on VTB, see the “Integrating the Controller and VIP using VTB” section in the “Integration” chapter of the *User Guide*.

3.17.7.2 C-PCIe System-Initiated L2 Power Gating

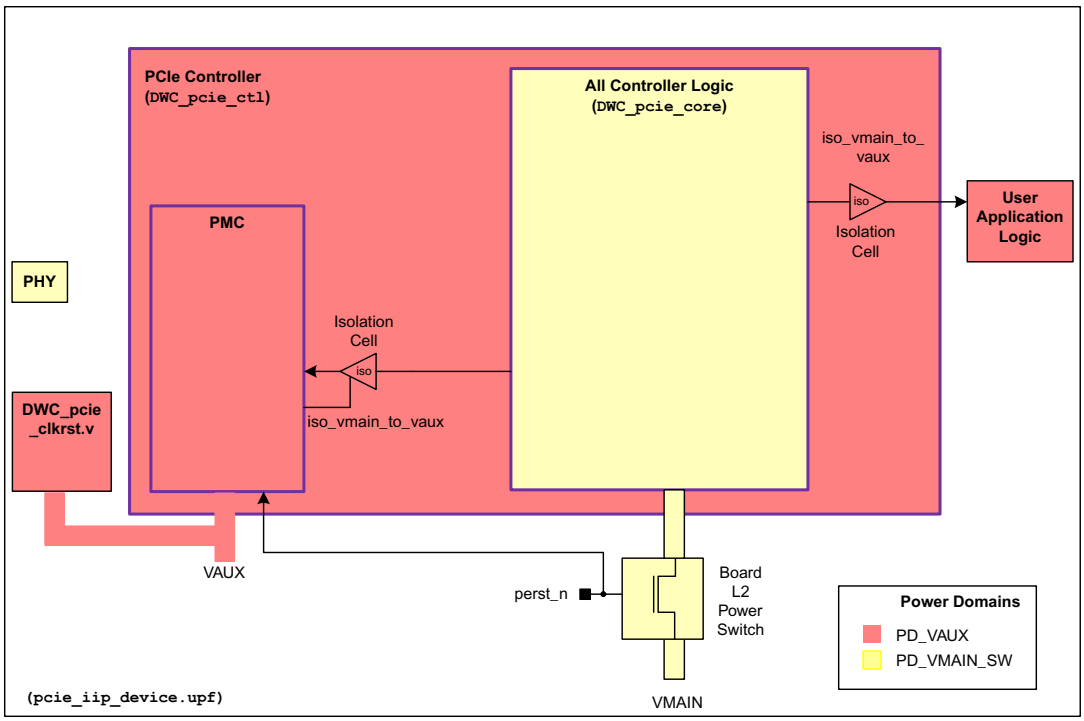
When the controller MAC is in L2 (after being programmed to D3), the system might decide to power-down the primary supply of VMAIN by asserting `perst_n`. Therefore, the assertion of `perst_n` is an indication that the system power controller is going to switch off VMAIN.

- `CX_L2_PG_EN =1`
- `CX_L1_SUBSTATES_ENABLE =0 || (CX_L1_SUBSTATES_ENABLE =1 && CX_L12_PG_EN =0)`

Table 3-87 L2 Configuration

Link State	Power Supply	
	AUX	MAIN
L0/L1	ON	ON
L2	ON	OFF
L3	OFF	OFF

Figure 3-115 L2 Symbolic Overview



Note If AMBA bridge is enabled, in addition to DWC_pcie_core, DWC_pcie_edma_amba_bridge module is instantiated in PD_VMAIN_SW power domain.

Figure 3-116 L2 Simple Timing

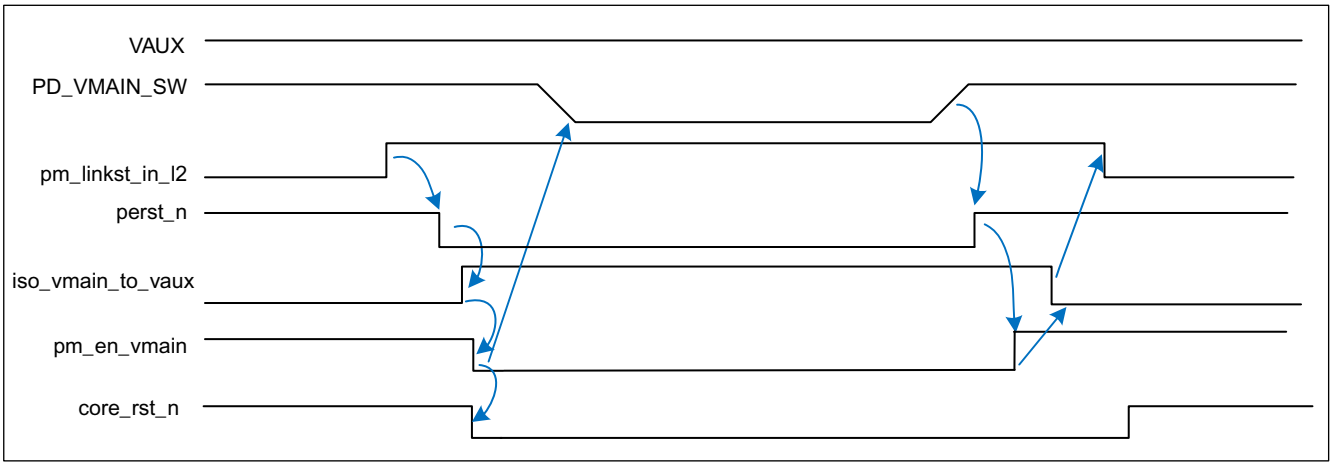
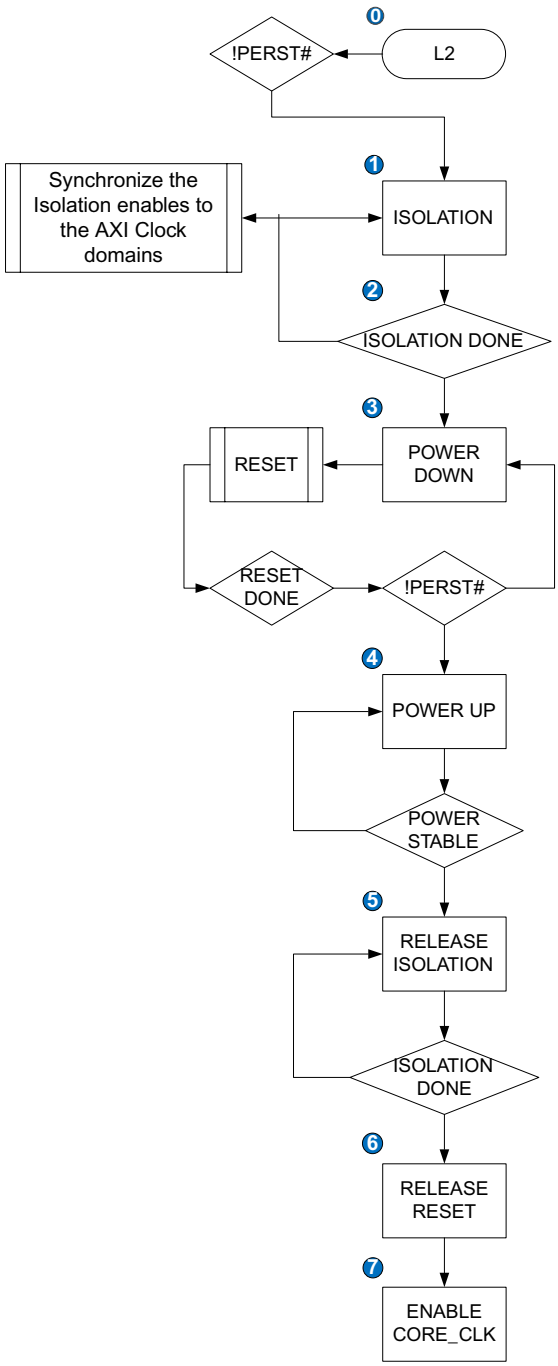


Figure 3-117 L2 Power Gating



L2 Power Gating Process

0 Pre-requisite
Negotiation for L2 entry has been successfully completed (The controller asserts the output pm_linkst_in_12 when this is the case)

1 The controller waits for the input perst_n to be set to 1'b0, before it begins the process of isolating the always on domain.

2 The controller isolates the outputs from its switchable power domains
- If AMBA bridge is present and asynchronous to the core_clk, the isolation enable is synchronized to the AMBA clock domains.
- The controller waits for all isolation enables to be asserted before proceeding with power removal.

3 The controller sets the output pm_en_vmain to 1'b0 to indicate the it is ready for main power removal
- The controller asserts the reset request outputs to request the assertion of the controller resets.
- When the resets have, all been asserted the controlled remains in the power down state until a wakeup event occurs.

4 When an L2 wakeup event occurs, the upstream port will assert the wake output to request the restoration of main power
- Your application should restore the main power and set the controller's perst_n input to 1'b1 when the main power is stable.
- When the controller samples perst_n at 1'b1, it begins the power-up sequence.

5 The isolation enable signals are de-asserted. The controller will wait for all isolation enables to be de-asserted before proceeding with the wakeup sequence.

6 The reset requests are de-asserted.

7 The controller sets the output pm_sel_aux_clk to 1'b0 to turn back on the core_clk and switch aux_clk back to core_clk.

3.17.7.3 C-PCIe L2 Power Gating + Autonomous L1.2 Power Gating (No Retention Registers)

There is a power switch in the UPF to allow the power management circuitry of the controller to automatically switch off the primary power supply to the selected parts of the design. Certain parts[1] of the design may still need to retain the state when power is removed. The state retention task is achieved by creating always on supply islands for modules requiring power in L1 sub-states. There are seven L1ON power islands (representing different parts of the RTL hierarchy) defined in the UPF. You can merge these into one power domain (PD_L1ON).

- `CX_L2_PG_EN =1`
- `CX_L12_PG_EN =1`
- `CX_L1_SUBSTATES_ENABLE =1 && app_l1_pwr_off_en =1`
- `CX_L1_RETENTION =0`

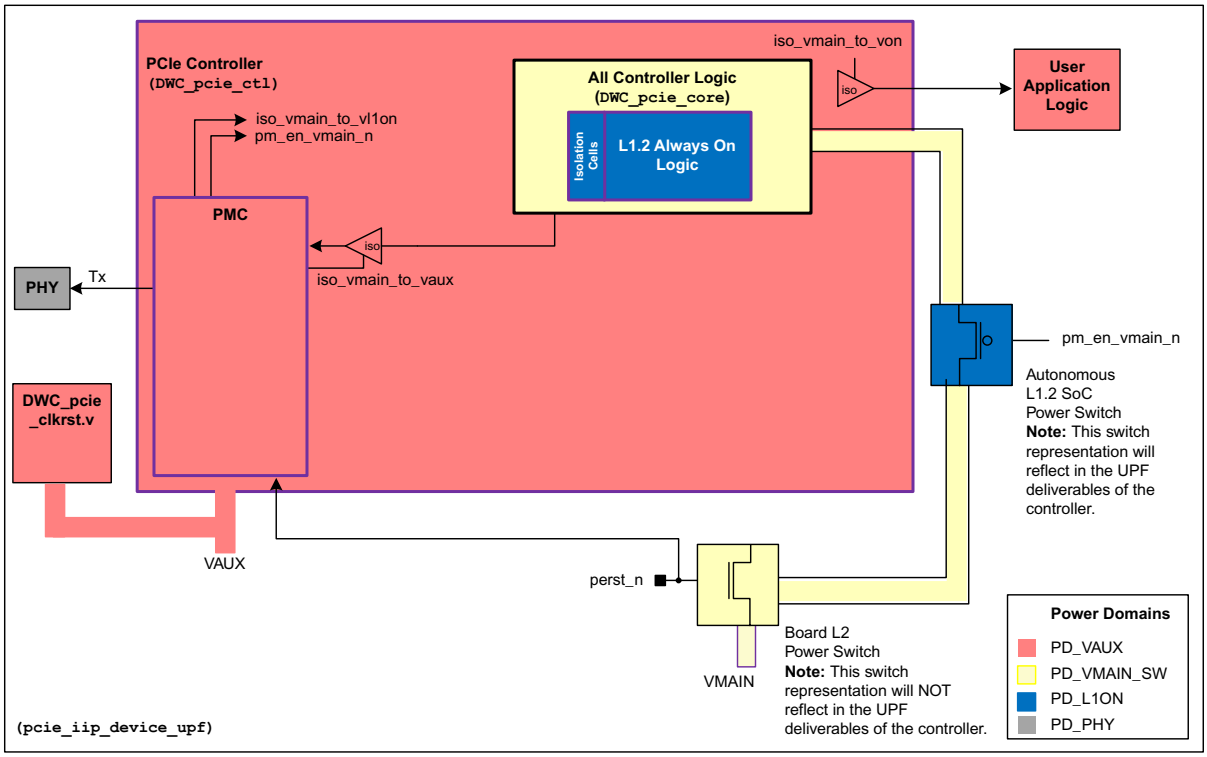
Table 3-88 L2 + L1.2_PD Configuration

Link State	Power Supply	
	AUX	MAIN
L0/L1/L1.1	ON	ON
L1.2	ON	ON[a]
L2	ON	OFF
L3	OFF	OFF

a. In L1.2 mode the MAIN power domain inside the controller is completely switched off. It might be ON at the SoC level, but in PCIe controller context it is switched off.

1. Such as CDM register block, Gen3 EQ settings, credits, lane reversal information and so on.

Figure 3-118 L2 + L1.2_PD Symbolic Overview



Note If AMBA bridge is enabled, in addition to DWC_pcie_core, DWC_pcie_edma_amba_bridge module is instantiated in PD_VMAIN_SW power domain.

Note If the board does not have access to the signal pm_en_vmain (this signal indicates the controller is ready for the power gating), the board must make sure the delay between perst_n assertion and power switch off the power is not less than the following amount of time:

- When AXI bridge is not enabled, the minimum delay must be:

  9 * muxed_aux_clk period
- When AXI bridge and AXI bridge isolation is enabled, the minimum delay must be:

  3 * the maximum value (mstr_aclk_ug, slv_aclk_ug, dbi_aclk_ug) period + 9 * muxed_aux_clk period

Figure 3-119 L2 + L1.2_PD Simple Timing

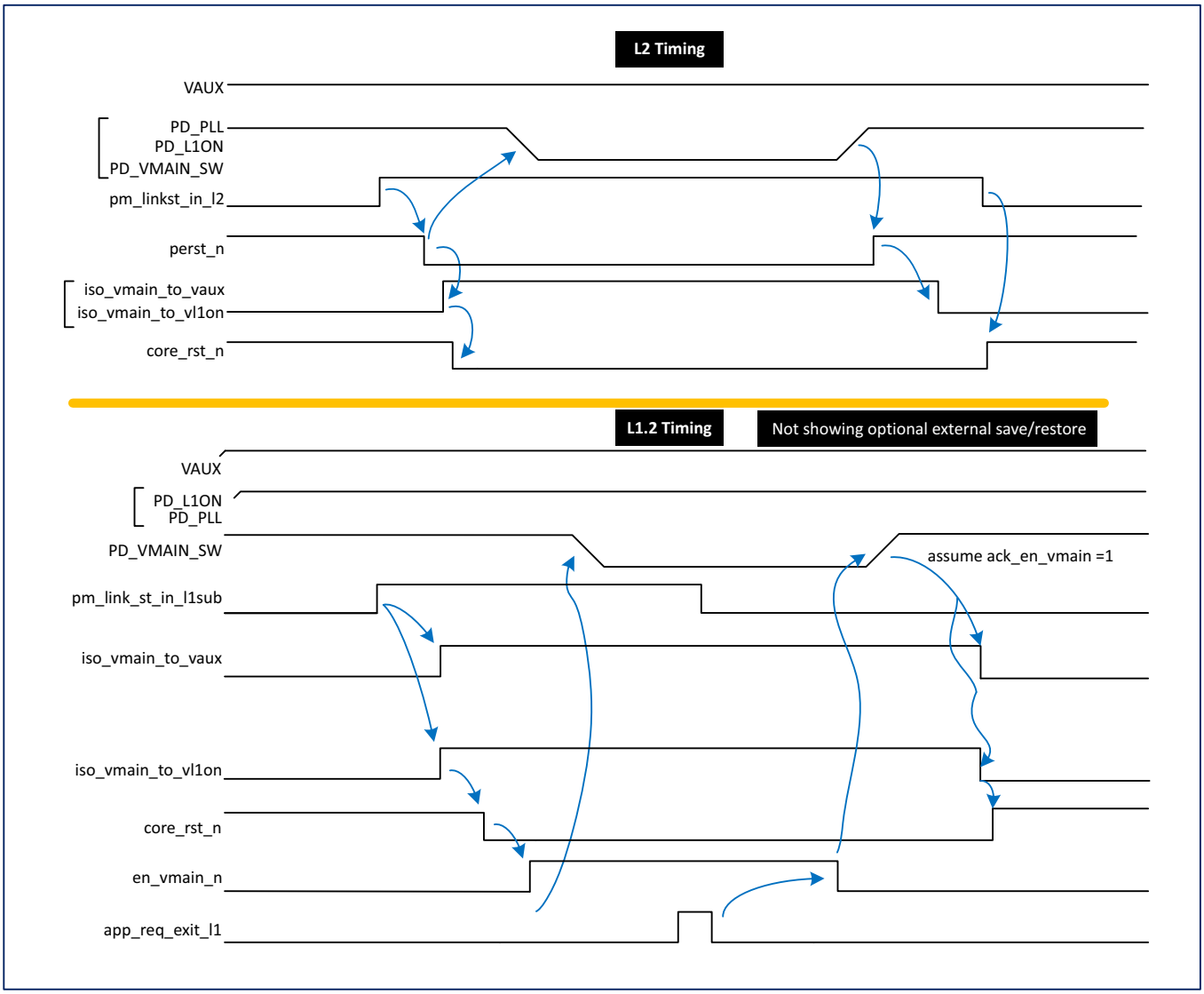
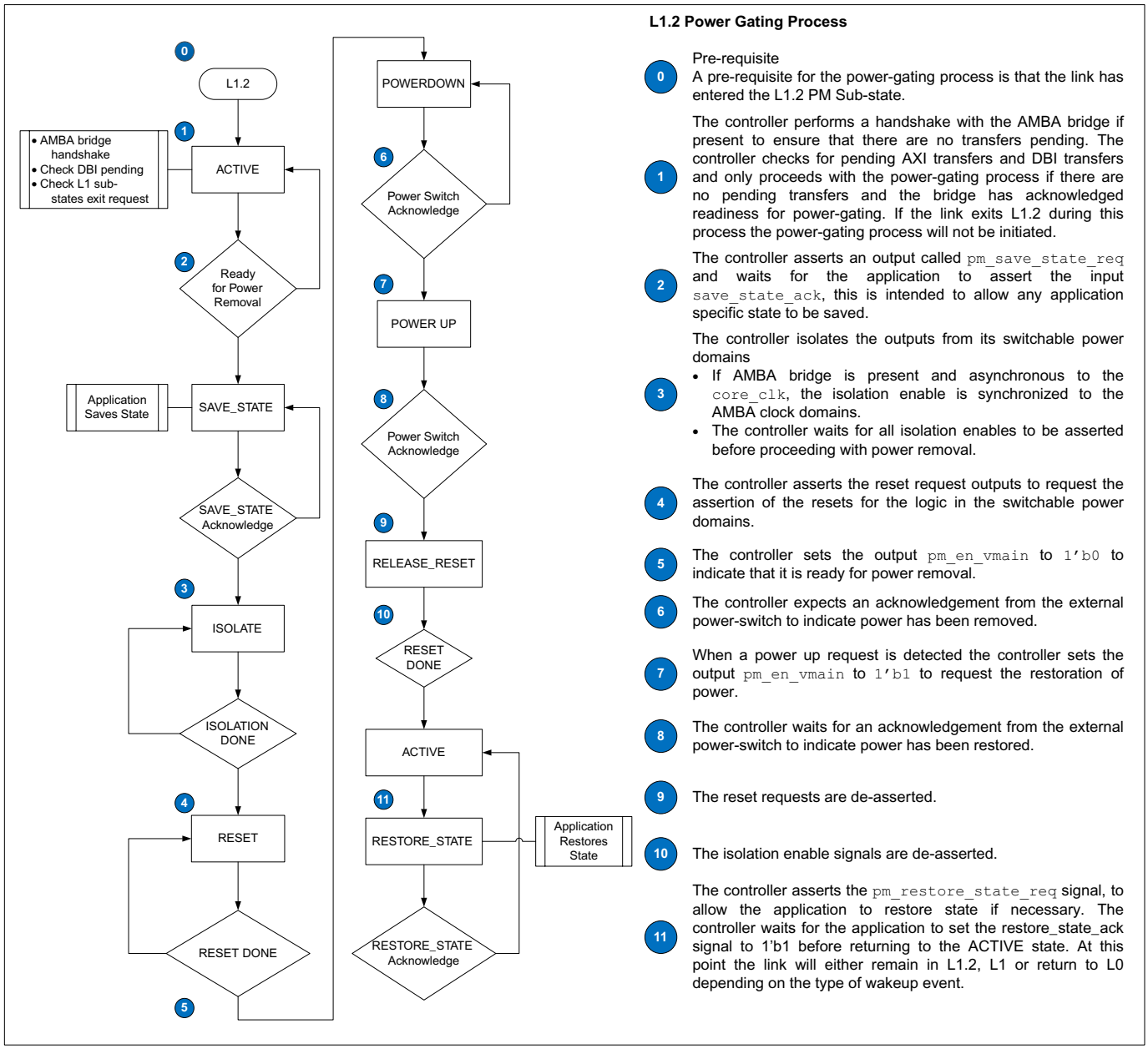


Figure 3-120 L1.2_PD Detailed Timing



3.17.7.4 C-PCIe L2 Power Gating + Autonomous L1.2 Power Gating (Using Retention Registers)

There is a power switch in the UPF to allow the power management circuitry of the controller to automatically switch off the primary power supply to selected parts of the design. Certain parts[1] of the design may still need to retain the state when power is removed. The state retention task is achieved by implementing (using your technology vendors library) retention registers in selected modules.

1. Such as CDM register block, Gen3 EQ settings, credits, lane reversal information and so on.

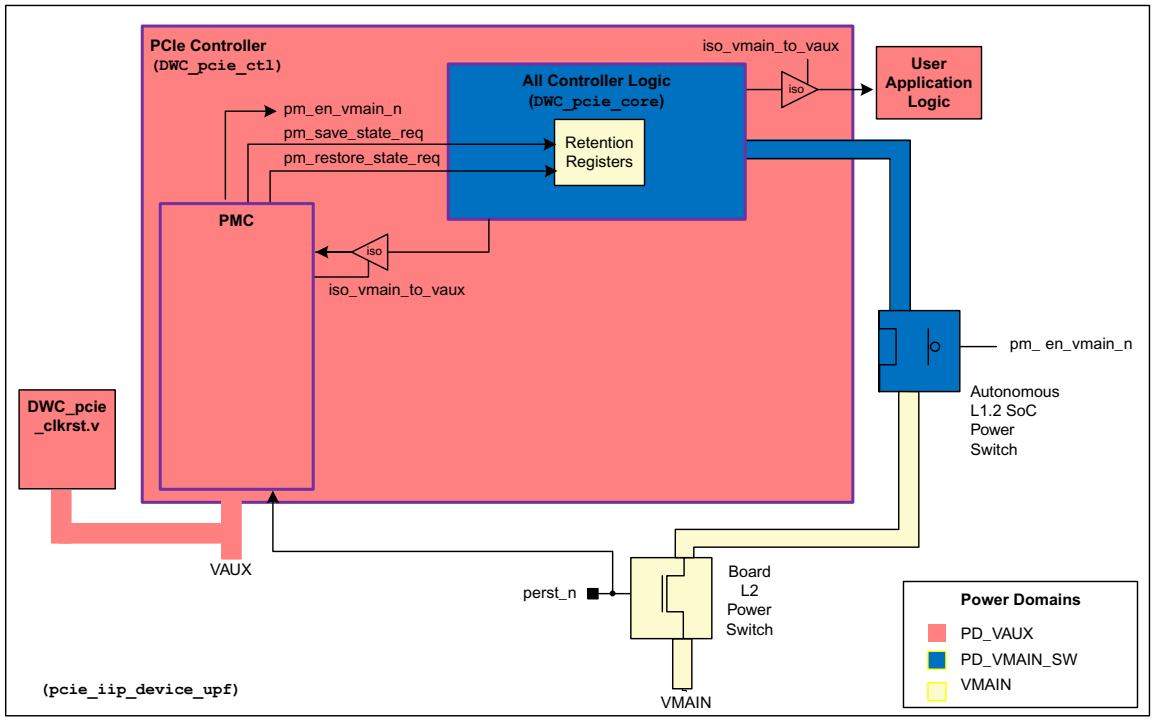
- CX_L2_PG_EN =1
- CX_L12_PG_EN =1
- CX_L1_SUBSTATES_ENABLE =1 && app_l1_pwr_off_en =1
- CX_L1_RETENTION =1

Table 3-89 L2 + L1.2_RET Configuration

Link State	Power Supply	
	AUX	MAIN
L0/L1/L1.1	ON	ON
L1.2	ON	ON[a]
L2	ON	OFF
L3	OFF	OFF

a. In L1.2 mode the MAIN power domain inside the controller is completely switched off. It might be ON at the SoC level, but in PCIe controller context it is switched off.

Figure 3-121 L2 + L1.2_RET Symbolic Overview



Note If AMBA bridge is enabled, in addition to DWC_pcie_core, DWC_pcie_edma_amba_bridge module is instantiated in PD_VMAIN_SW power domain.

Figure 3-122 L2 + L1.2_RET Simple Timing

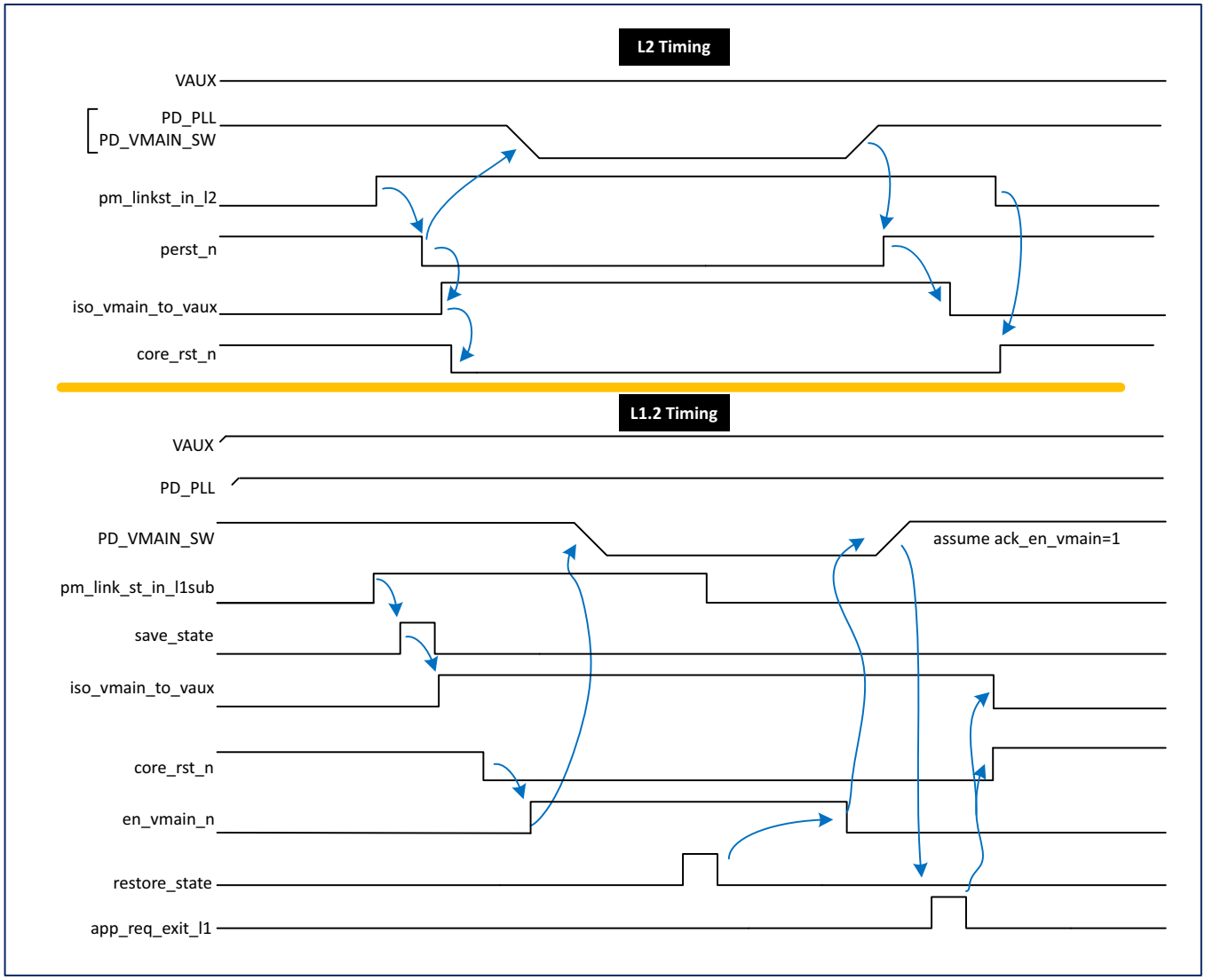
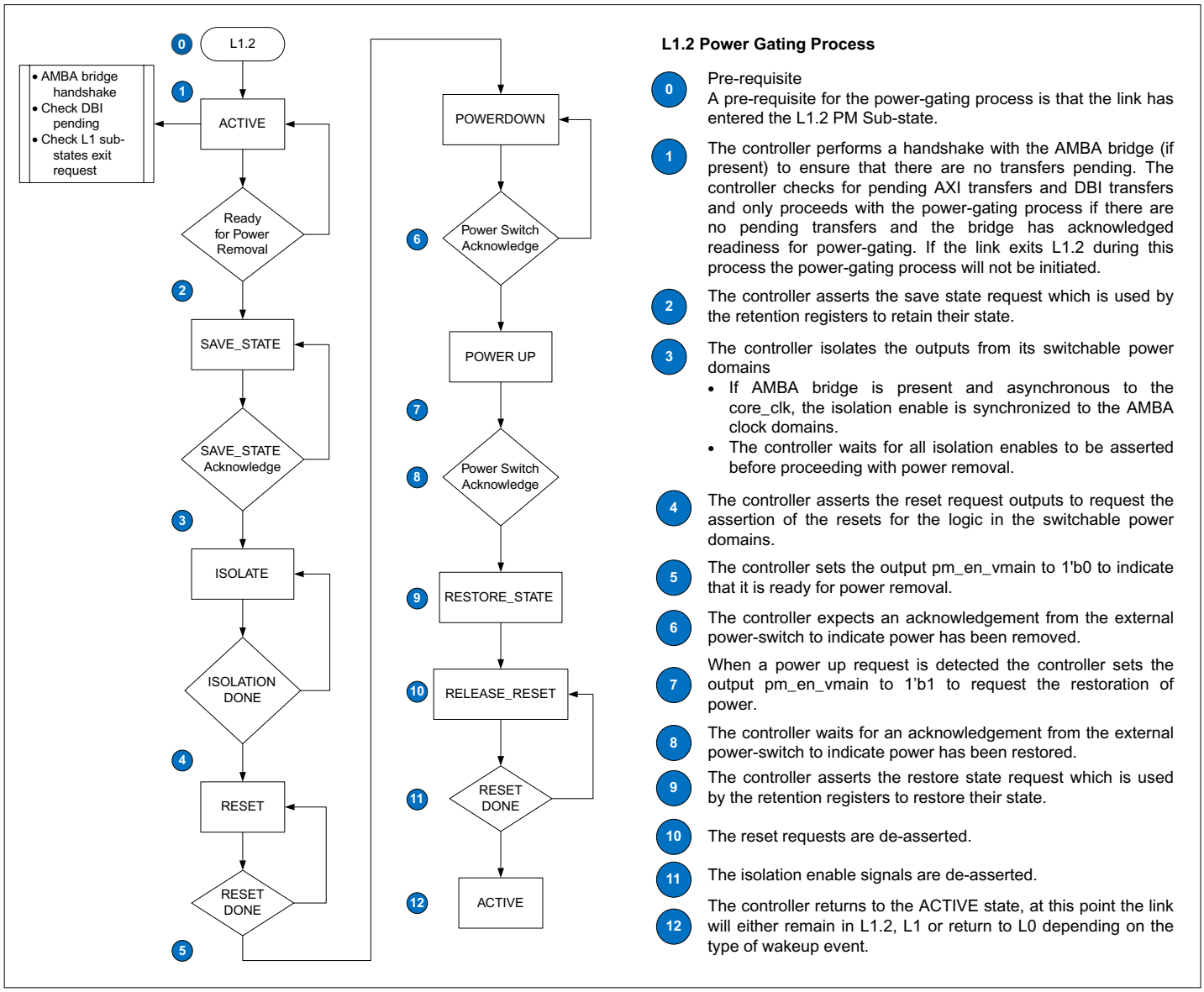


Figure 3-123 L1.2_RET Detailed Timing



3.17.7.5 C-PCIe Autonomous L1.2 Power Gating (No Retention Registers)

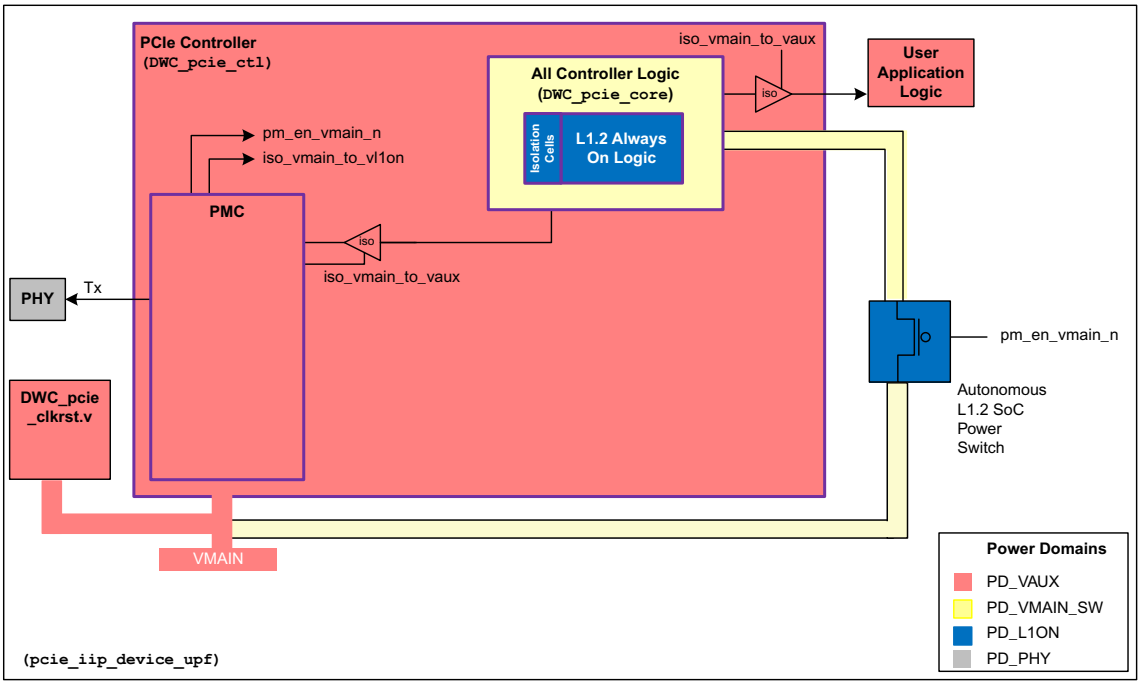
This scheme is similar to "C-PCIe L2 Power Gating + Autonomous L1.2 Power Gating (No Retention Registers)" on page 300 except that there is no always-on power supply available in L2.

- `CX_L2_PG_EN =0`
- `CX_L12_PG_EN =1`
- `CX_L1_SUBSTATES_ENABLE =1 && app_l1_pwr_off_en =1`
- `CX_L1_RETENTION =0`

Table 3-90 L1.2_PD Configuration

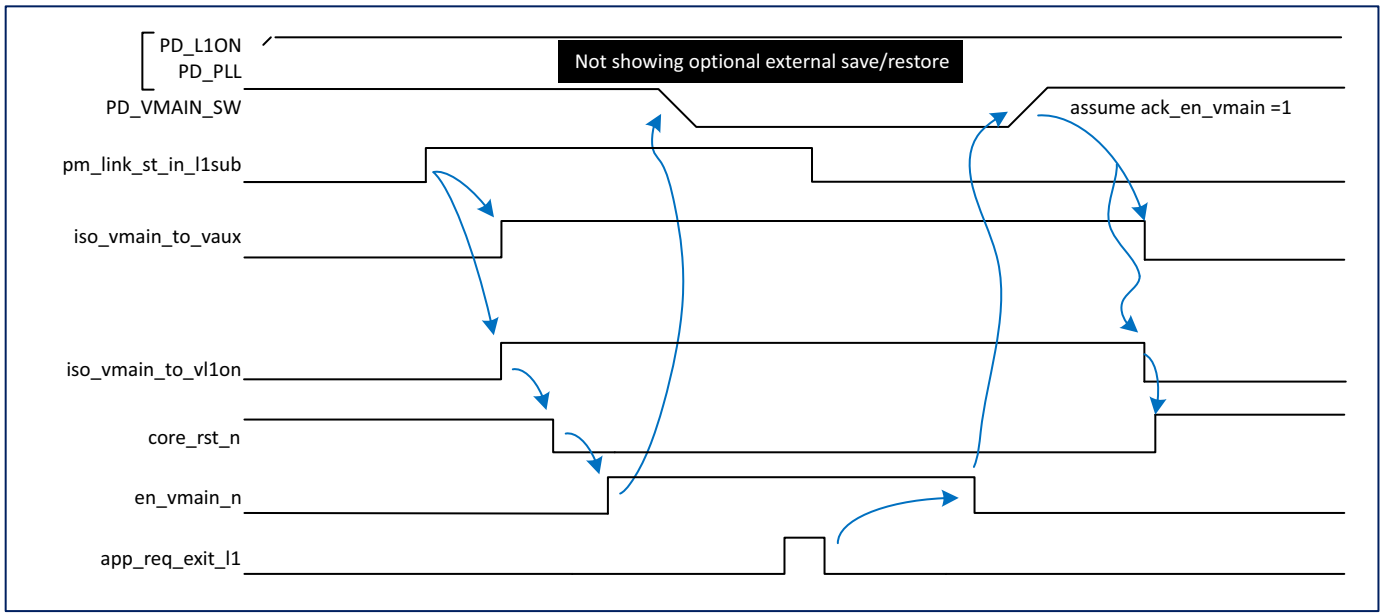
Link State	Power Supply	
	AUX	MAIN
L0/L1/L1.1	Not present	ON
L1.2	Not present	ON
L2/L3	Not present	OFF

Figure 3-124 L1.2_PD Symbolic Overview



Note If AMBA bridge is enabled, in addition to DWC_pcie_core, DWC_pcie_edma_amba_bridge module is instantiated in PD_VMAIN_SW power domain.

Figure 3-125 L1.2_PD Simple Timing



3.17.7.6 C-PCIe Autonomous L1.2 Power Gating (Using Retention Registers)

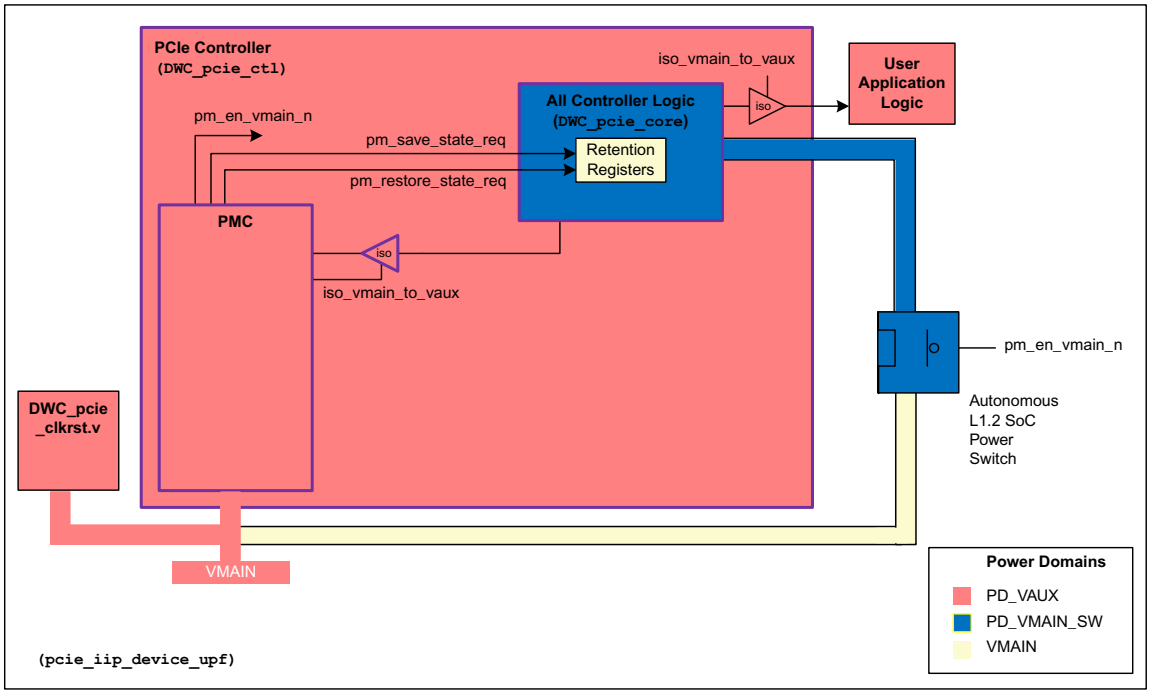
This scheme is similar to "C-PCIe L2 Power Gating + Autonomous L1.2 Power Gating (Using Retention Registers)" on page 303 except that there is no always-on power supply available in L2.

- `CX_L2_PG_EN =0`
- `CX_L12_PG_EN =1`
- `CX_L1_SUBSTATES_ENABLE =1 && app_l1_pwr_off_en =1`
- `CX_L1_RETENTION =1`

Table 3-91 L1.2_RET Configuration

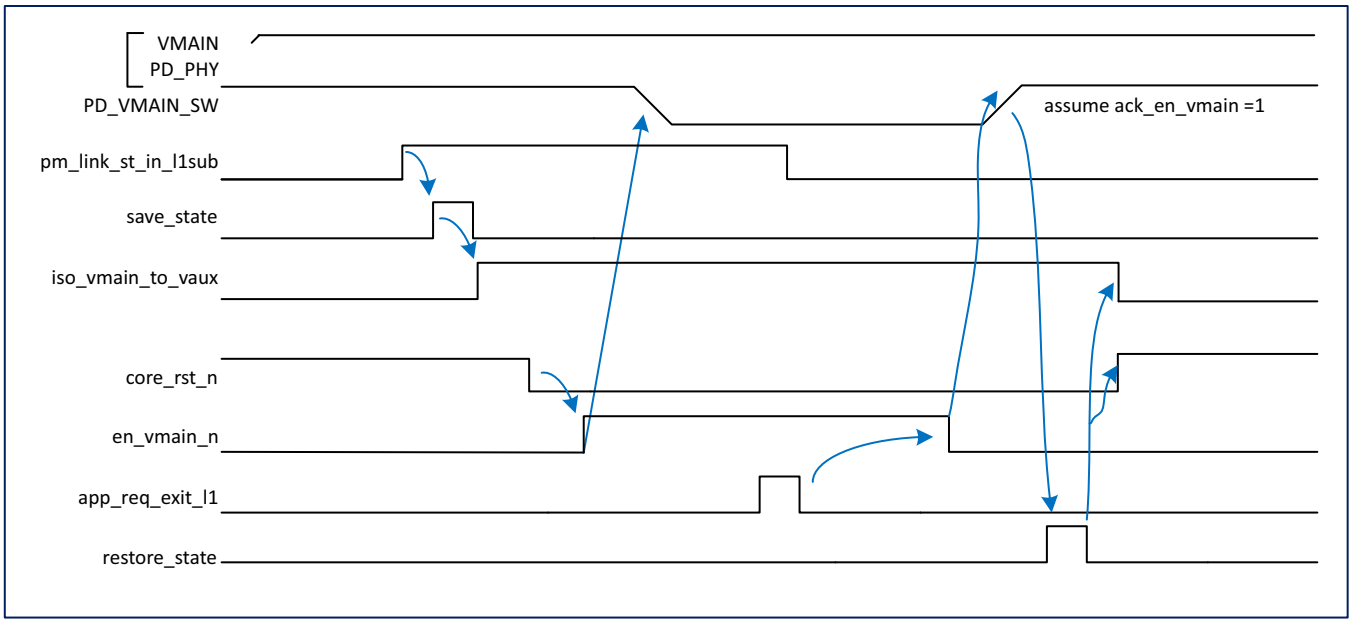
Link State	Power Supply	
	AUX	MAIN
L0/L1/L1.1	Not present	ON
L1.2	Not present	ON
L2/L3	Not present	OFF

Figure 3-126 L1.2_RET Symbolic Overview



Note If AMBA bridge is enabled, in addition to DWC_pcie_core, DWC_pcie_edma_amba_bridge module is instantiated in PD_VMAIN_SW power domain.

Figure 3-127 L1.2_RET Simple Timing



3.17.7.7 UPF Files

A set of configuration-specific UPF files which are supported by VTB simulations and synthesis are created by the coreConsultant tool in `workspace/upf`. The main UPF file defines:

- Power domains
- Isolation rules
- Level shifter rules
- Retention cell technology cell name. You must edit the UPF and replace this with the actual name of the retention registers in your technology vendor s library.
- Power supplies
- Power states
- Power state table

Note For more information, see the “UPF Flow and Methodology” section in the *User Guide.*

3.17.7.8 P2.NoBeacon Support

P2.NoBeacon is a custom powerdown encoding supported by Synopsys PHYs that enables deeper power saving as compared to the P2 state. To enable the controller to drive the PHY’s P2.NoBeacon state you must set the parameter `CX_P2NOBEACON_ENABLE` to ‘1’.

When this feature is enabled,

- The controller/PHY exits from L2/P2 only when PERST# is de-asserted. Beacon is not supported as an exit condition by the PHY, and is ignored
- `mac_phy_powerdown` drives P2.NoBeacon encoding, instead of P2 encoding, when the link goes to L2

To configure P2.NoBeacon encoding use `CX_PIPE43_P2NOBEACON_ENCODING` parameter. The default value of this parameter is 4'b1111. To disable this feature, set `P2NOBEACON_ENABLE` field of the `PHY_INTEROP_CTRL_OFF` register to '0'.

3.18 Completion Timeout Ranges

Timeout ranges are supported as defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The controller supports all ranges. The Device Control 2 Register has a default equal to the default in the specification: 0000b Default range: 50 µs to 50 ms. When the default is used, then the timeout is in Range B: 0101b: 16 to 55 ms. This range is chosen for the default because the *PCI Express Base Specification, Revision 6.0, Version 1.0* states, "It is strongly recommended that the completion Timeout mechanism not expire in less than 10 ms". Table 3-92 illustrates the specification values versus the PCI Express controller values for the ranges.

Attention As the PCIe specification states, This mechanism is intended to be activated only when there is no reasonable expectation that the completion is returned, and should never occur under normal operating conditions.

Table 3-92 Comparison of PCIe Specification Ranges and Synopsys PCIe Controller Completion Timeout Ranges (CX_CPL_TO_RANGES_ENABLE =1)

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

Note If CX_CPL_TO_RANGES_ENABLE =0, the default timeout range is 8 to 12 ms.

3.19 TLP Processing Hints (TPH)

TLP Processing Hints (TPH) allows your application to provide or process hints on a per transaction basis to optimize the processing of transactions targeting memory space. Posted and non-posted requests targeting memory space can use the PH and 2-byte ST fields in the header. For more information, see the *PCI Express ECN TLP Processing Hints* document.

You can enable TPH globally for all physical functions (PF) by setting the `CX_TPH_ENABLE` parameter.

3.19.1 Assumptions

- When your application is generating requests, it must retrieve the steering tag (ST) from the software programmable ST table. The ST table can be located in the TPH capability structure or combined with the two most significant bytes of the Vector Control register of each MSI-X Table entry in the MSI-X capability structure. When using the interrupt vector mode of operation, and when steering tags are located in the TPH capability structure, then your application must decode the MSI/MSI-X interrupt vector number and select the correct value from the ST table.
- The application is responsible for passing only valid values to the controller when TPH is enabled or disabled. The controller does not perform any signal checks or corrections.

3.19.2 Limitations

- The controller does not support the Extended TPH feature.
- When TPH is enabled, the device is enabled as both a requester and a completer. No control is given to support these features individually.
- In the Interrupt Vector Mode of operation:
  - If `TPH_ST_TABLE_SIZE_# < MSIX_TABLE_SIZE_#` then your application is permitted to do any of the following:
    - Not use TPH for certain transactions
    - Use TPH with a Steering Tag of 0
    - Use TPH with an implementation defined mechanism used to select a ST value from the ST Table.
  - If the `TPH_ST_TABLE_SIZE_#` is larger than the enabled range of interrupt vector numbers, then ST Table Entries corresponding to out-of-range interrupt vector numbers are ignored by the function.
- There is no TPH support in the "Internal Address Translation Unit (iATU)" on page 220. The iATU is not aware of TPH field. Therefore type translation is not possible. However, address translation based on address or BAR matching is possible, and all TPH TLP fields are preserved during translation.

3.20 Atomic Operations (AtomicOps)

You can enable support for AtomicOps by setting the `CX_ATOMIC_ENABLE` parameter. AtomicOps defines the following PCIe request TLPs:

- Fetch and Add (FetchAdd)
- Unconditional Swap (Swap)
- Compare and Swap (CAS)

FetchAdd and Swap supports 32-bit and 64-bit operand sizes. CAS supports 32-bit, 64-bit, and 128-bit operand sizes. When you enable AtomicOps, the controller receives and forwards AtomicOp TLPs to your application, and generates AtomicOp TLPs in response to requests from your application on the XALI client interfaces. Your application must generate the required completion for any AtomicOps that are passed to it. The PCI Express controller does not generate completions for AtomicOps.

Four bit fields in the Device Capabilities 2 Register allow your software to discover the completer capabilities for PCIe functions, and the AtomicOp routing capabilities of routing elements. Two bit fields in the Device Control 2 Register allow your software to program a function, enabling it to:

- Initiate AtomicOp requests

  or
- Block the egress port of a routing element (SW) to avoid forwarding an AtomicOp request to a component that should not receive it.

Limitations

- When you enable 128-bit Atomic CAS instructions (through the `CX_ATOMIC_128_CAS_EN` configuration parameter), you must configure the controller (through the `RADM_NP_DCRD_VCn` configuration parameter) to advertise more than one non-posted data credit on at least one VC.
- AtomicOp TLP type cannot be nullified on TX.

3.20.1 AXI5 Support

For AMBA AXI5 configurations, the AXI bridge module seamlessly transports the PCIe AtomicOp transactions to the AXI interconnect, and vice-versa when `AMBA_INTERFACE =AXI5` and `CX_ATOMIC_ENABLE =1`. The AXI subordinate translates AXI5 atomic transactions to outbound PCIe AtomicOp TLPs, and the AXI manager translates inbound PCIe AtomicOp TLPs to AXI5 atomic transactions. All PCIe AtomicOps are supported.

Table 3-93 describes the mapping between the PCIe AtomicOp TLPs and AXI5 atomic transactions. Figure 3-128 shows inbound and outbound PCIe AtomicOp TLP to AXI5 atomic transaction translation mechanism.

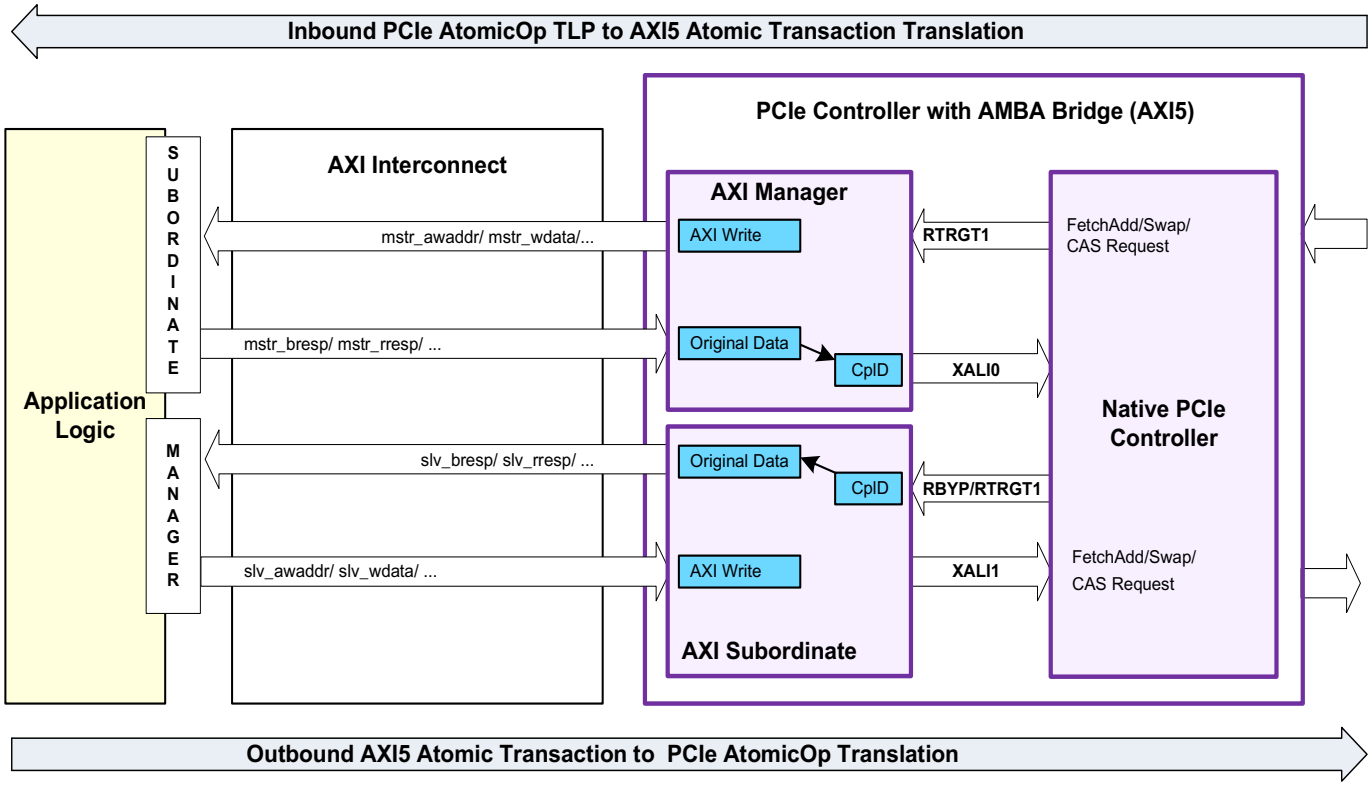
Table 3-93 PCIe AtomicOp TLP and AXI5 Atomic Transaction Mapping

Inbound path (PCIe to AXI)		Outbound path (AXI to PCIe)	
PCIe AtomicOp TLP	**AXI5 Transaction[a]**	**AXI5 Transaction**	**PCIe AtomicOp TLP**
FetchAdd[b]	AtomicLoad	AtomicLoad[c]	FetchAdd
Swap[d]	AtomicSwap	AtomicSwap	Swap

Inbound path (PCIe to AXI)		Outbound path (AXI to PCIe)	
PCIe AtomicOp TLP	**AXI5 Transaction[a]**	**AXI5 Transaction**	**PCIe AtomicOp TLP**
CAS[e]	AtomicCompare[f]	AtomicCompare	CAS
		AtomicStore	Not Supported

a. Unique IDs (as per AXI specification guidelines) are assigned to AXI write transactions.
b. AXI manager converts FetchAdd to INCR burst type request.
c. Only ADD atomic transaction operator is supported. CLR, EOR, SET, SMAX, SMIN, UMAX, and UMIN atomic transaction operators are not supported.
d. AXI manager converts Swap to INCR burst type request.
e. AXI manager adjusts the burst type either to INCR or WRAP, depending upon the address alignment of the request.
f. The transaction data includes both compare and swap values.

Figure 3-128 PCIe AtomicOp TLP to AXI5 Atomic Transaction Translation



Note When AXI shared DBI access is enabled (SHARED_DBI_ENABLED =1), an AtomicOp request must set slv_awmisc_info[21] =0 to indicate that the request is a normal PCIe outbound request, and not a DBI access.

Limitations

- For AtomicLoad transaction to PCIe FetchAdd AtomicOp TLP translation, only ADD atomic transaction operator is supported. CLR, EOR, SET, SMAX, SMIN, UMAX, and UMIN atomic transaction operators are not supported.
- AXI5 atomic transaction data size of 1-byte and 2-byte is not supported.

3.20.2 Internal Address Translation Unit (iATU) Support

The "Internal Address Translation Unit (iATU)" on page 220 supports AtomicOps. Outbound address translation is provided through address matching. By correctly programming the type field, a matched address can be translated into an AtomicOp. The setting of the format field is automatically handled within the controller. When AXI bridge and AtomicOp is enabled, ATU does not translate outbound requests (AXI to PCIe).

For inbound address matching, you can program the iATU to also match other fields, including the type field. The iATU also supports fuzzy match mode to allow a group of requests to be matched to a partial type field. When fuzzy mode is set (bit 27 of the iATU Control 2 Register, and the `type[4:2]` field (iATU Control 1 Register) is set to 3'b011, all AtomicOp requests are matched.

3.20.3 Atomic Routing - Egress Blocked Support

When AtomicOp Egress Blocking is enabled through the Device Control 2 Register in an egress port (A in Figure 3-129), and an AtomicOp request is scheduled to or go out that port, then that port must:

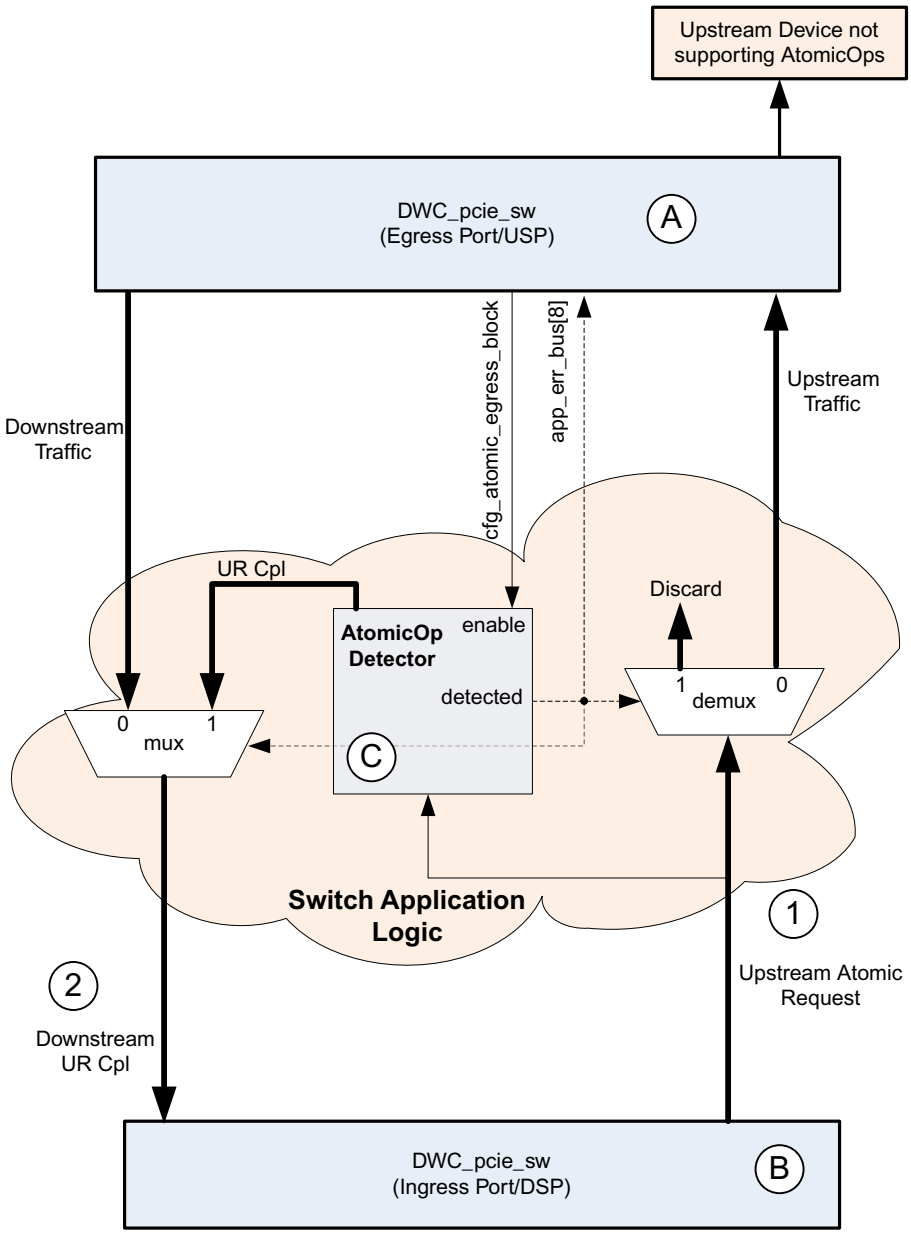
- Handle the request as an AtomicOp Egress Blocked error by setting the appropriate bit in the Uncorrectable Error Status Register.
- Return a completion with a completion status of UR.

A possible way to do this (using a switch as an example) is:

- Set As AtomicOp Egress Blocking bit in the Device Control 2 Register.
- Design an AtomicOp Detector (C in Figure 3-129) in your application logic that:
  a. Detects an AtomicOp forwarded by the ingress/Downstream Port (B in Figure 3-129).
  b. Discards (blocks) that AtomicOp.
  c. Generates downstream completion with status of UR[1], for the blocked AtomicOp.
  d. Sets the AtomicOp Egress Blocked error status bit in the egress port using the `app_err_bus[8].`

1. The unsupported request detected bit in the Device Status register is not set.

Figure 3-129 Example of AtomicOp Egress Blocking Using Switch



3.21 TLP Prefix

The TLP prefix mechanism extends the TLP by adding DWORDs in front of the TLP header to carry additional information. The controller supports this feature when you set the parameter `CX_NPRFX >0`. The controller forwards TLP prefixes unaltered. It validates each prefix by checking its format field before forwarding it. Each validated prefix has its type field read to determine if it is Local or End-End. Only End-End prefixes are included in the ECRC (if enabled). The controller analyzes the extracted prefixes to ensure that the TLP prefix rules are met.

Features and Limitations

- The controller supports a maximum of seven prefixes, including a maximum of four End-End prefixes
- A TLP (of any type) can contain from 0 to `CX_NPRFX` TLP prefixes
- When a combination of Local and End-End TLP prefixes are present in a TLP, Local TLP prefixes must precede any End-End TLP prefixes
- There is no prefix support for messages which are handled over the SII (such as LTR messages) or over the VMI (vendor-defined messages)
- When the AXI bridge is enabled, and the request is split, the corresponding TLP prefix (`slv_awmisc_info_tlpprfx/slv_armisc_info_tlpprfx)` is present on all the split requests

For more information on Prefix usage when PCIe IDE (`CX_IDE_ENABLE`) is enabled, see "PCIe Integrity and Data Encryption (IDE)" on page 591.

End-End TLP Prefix Blocking (DSP mode only)

When enabled, your application must report blocking errors on the SII. It must set the End-End TLP Prefix Blocking bit field in the Device Control 2 Register (`DEVICE_CONTROL2_DEVICE_STATUS2_REG`) in ports that support End-End TLP Prefixes but that do not support forwarding of End-End TLP Prefixes.

TLP Prefix Completion/Response Signals

The following internal parameters must be enabled so that the signals corresponding to TLP prefix read and write responses in the AXI manager and subordinate interfaces are available:

- `CC_MSTR_RESP_TLP_PREFIX_ENABLE`
- `CC_SLV_RESP_TLP_PREFIX_ENABLE`

Request/Response TLP Prefix Examples

Figure 3-130 to Figure 3-132 show the mapping between TLP prefix DWORDs and the bits of input signals *misc_info_tlpprfx/*_tlp_prfx.

Figure 3-130 Example 1: CX_NPRFX =7; 2 End-End Prefixes are Present

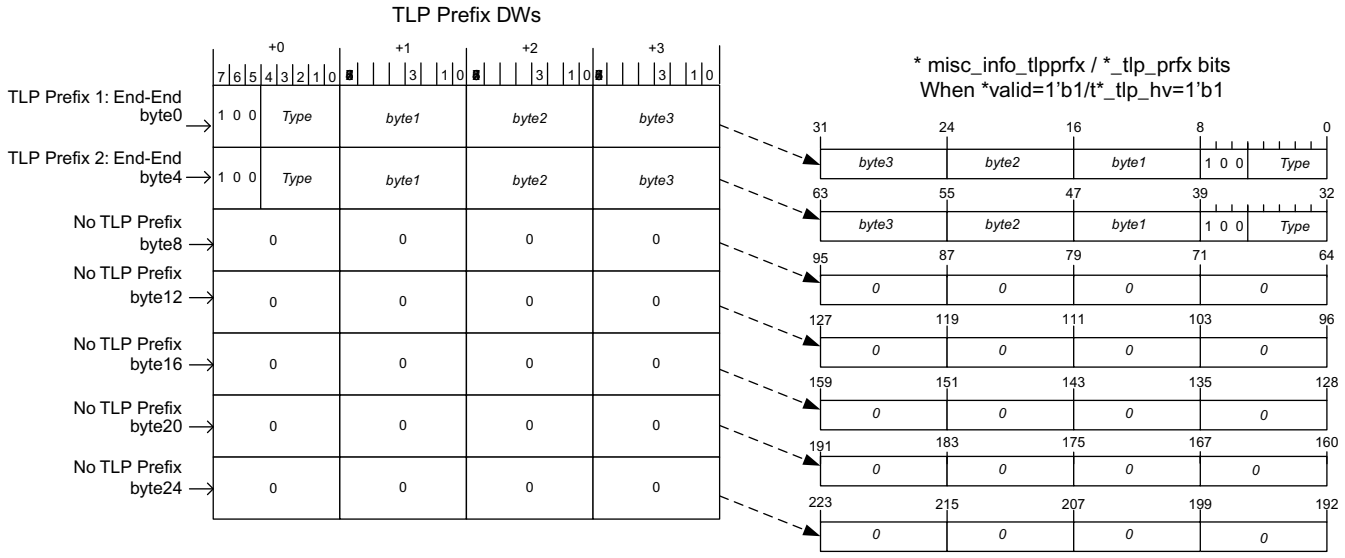


Figure 3-131 Example 2: CX_NPRFX =5; 4 End-End and 1 Local Prefixes are Present

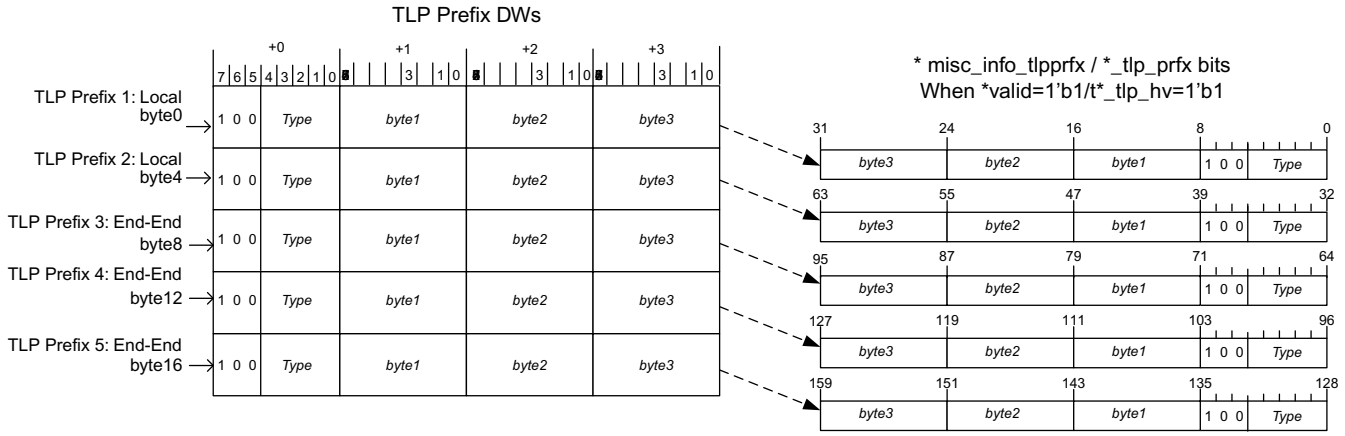
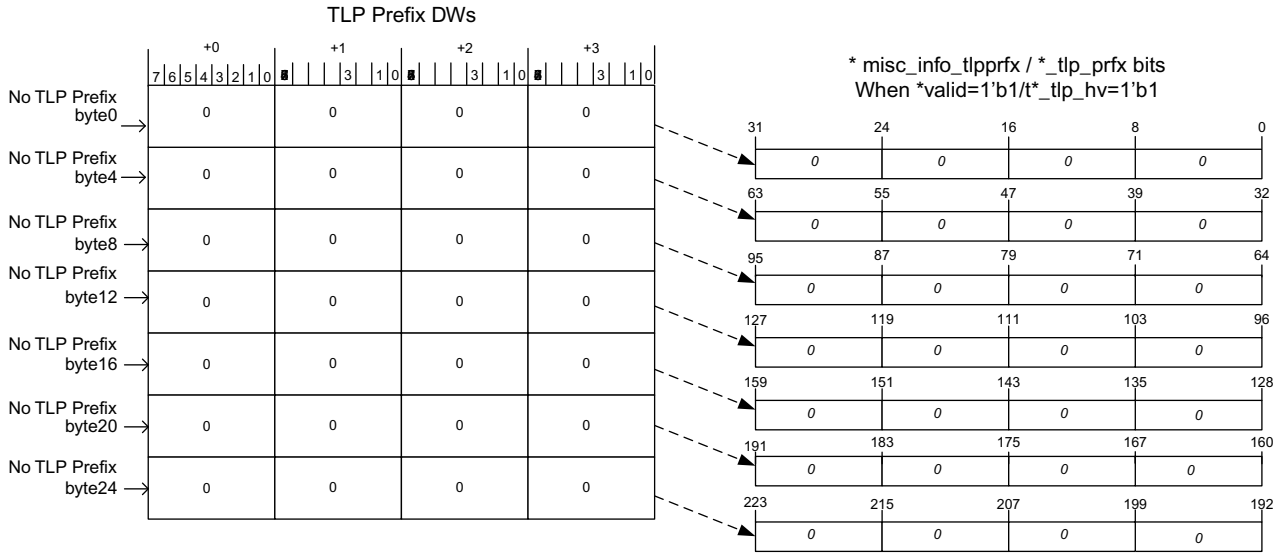


Figure 3-132 Example 3: CX_NPRFX=7; No TLP Prefixes are Present



3.22 Separate Refclk Independent SSC (SRIS)

You can enable support for the Separate Refclk with Independent Spread Spectrum Clocking (SRIS) feature as defined by the *PCI Express Base Specification, Revision 6.0, Version 1.0*, by setting `CX_SRIS_SUPPORT =1`. This also populates the `app_sris_mode` input pin. When `CX_SRIS_SUPPORT =1` and `app_sris_mode =1`, then L0s is not supported and must not be advertised in the ASPM Support Capability Registers.

Table 3-94 SKP Ordered Set Intervals in SRIS and non-SRIS Modes

CX_SRIS_SUPPORT	app_sris_mode	SKP Ordered Set Intervals (8b/10b Encoding)	SKP Ordered Set Intervals (128b/130b Encoding)
0	NA	Normal (between 1180 and 1538 Symbol times)	Normal (between 370 to 375 Blocks)
1	0	Normal	Normal
1	1	Short (less than 154 Symbol times)	short (less than 38 Blocks)

Note The controller does not implement the Lower and Upper “SKP OS Generation Supported Speeds Vector” fields in the Link Capabilities 2 register because the controller architecture does not benefit from a lower SKP frequency.

3.23 Readiness Notifications (RN)

The controller supports the optional Readiness Notification features as outlined in Table 3-95. These features provide multiple mechanisms for a device or function to notify the host software that it is configuration-ready. A configuration-ready function always responds to a valid configuration request with a completion indicating successful completion status, and not CRS. For more information, see Readiness Notification in the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

Table 3-95 Controller Operations

Parameter Name	Description
CX_RN_IMM_EN	**Enable Immediate Readiness.** When you set the Immediate Readiness bit in the PCI header Status Register, the function is configuration-ready. Software is exempt from all requirements to delay configuration accesses following any type of reset or exit from low-power states.
CX_RN_RTR_EN	**Enable Readiness Time Reporting (RTR).** RTR provides an optional mechanism for describing the time required for a device or function to become configuration-ready. Software is permitted to issue requests to the device or function (following any type of reset or exit from low-power states) after waiting for the time advertised in this capability and need not wait for the (longer) times required elsewhere.
CX_RN_FRS_SUPPORTED	**Enable Function Readiness Status (FRS).** FRS provides an optional mechanism for messaging the host software when a function has become configuration-ready. The controller autonomously sends a Vendor-Defined Message (VDM) with no payload when any of these events occur: ■ Completion of D3hot to D0 transition ■ DRS message received (by Downstream Port) 1. When multiple FRS events occur in close proximity: ■ Sometimes the controller may transmit a single FRS. ■ The reason field in the transmitted message may contain a code from any of the events. 2. Software is permitted to issue requests to the function (following any of these events) after receiving an FRS message from this function and need not wait for the (longer) times required elsewhere in the specification for these events. 3. The DSP controller asserts the `cfg_up_drs_to_frs` output and sends an FRS message with the reason code set to “DRS Message Received” when: ■ It receives a DRS message, and ■ `PCIE_CAP_DRS_SIGNALING_CONTROL` in `LINK_CONTROL_LINK_STATUS_REG` is 2b10

Parameter Name	Description
CX_RN_DRS_SUPPORTED	**Enable Device Readiness Status (DRS).** DRS provides an optional mechanism for messaging the host software when a device has become configuration-ready. The controller autonomously sends a Vendor-Defined Message (VDM) with no payload when any of these events occur: ■ Exit from Cold, Warm, or Hot Reset ■ Exit from Loopback or Disabled ■ Exit from L2/L3_Ready ■ Port transitions from DL_Down DL_Up status 1. Software is permitted to issue requests to the function (following any of these events) after receiving an DRS message from this function and need not wait for the (longer) times required elsewhere in the specification for these events. 2. The DSP controller asserts the `cfg_drs_msi` output when all of the following are true: ■ It receives a DRS message ■ `PCIE_CAP_DRS_SIGNALING_CONTROL` in `LINK_CONTROL_LINK_STATUS_REG` is 2b01 ■ MSI or MSI-X is enabled

3.23.1 FRS Queue

When `CX_RN_FRS_SUPPORTED =1`, the controller implements an FRS queue (FRSQ) to collect FRS messages received by the root port or generated by the root port (in response to received DRS messages). The oldest FRS message is visible in the FRS message queue register (`FRS_MESSAGE_QUEUE_OFF`) of the FRSQ capability which can be read through the DBI. Writing to this register clears the oldest FRS message from the queue. The controller asserts the `frsq_msi` output when:

- It receives an FRS message or the FRSQ overflows
- `FRS_INTERRUPT_ENABLE` field in `FRSQ_CONTROL_FRSQ_STATUS_OFF =1`
- You have enabled MSI or MSI-X.

You can configure the queue depth using the `CX_RN_FRS_QUEUE_MAX_DEPTH` parameter.

3.24 Expansion ROM Validation

The Expansion ROM Validation is a mechanism for reporting the results of implementation specific validation of ROM contents. The controller supports this optional mechanism as mentioned in the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

After your application checks the ROM contents, you must update the Expansion ROM Validation Status and Expansion ROM Validation Details in the Expansion ROM BAR Register. That is, `ROM_BAR_VALIDATION_STATUS` and `ROM_BAR_VALIDATION_DETAILS` fields of the `TYPE1_EXP_ROM_BASE_REG` register. You can update these fields using:

- DBI Write, or
- The `exp_rom_validation_status`, and `exp_rom_validation_details` signals. For more information, see "SII:ROM Validation Signals" section in the "Signal Descriptions" chapter of the *Reference Manual*.

3.25 Precision Time Measurement (PTM)

The controller supports the optional PTM feature (`CX_PTM_ENABLE` =1) as defined in Section 6.22, Precision Time Measurement (PTM) Mechanism of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. This enables the coordination of timing information across multiple devices with independent local time-bases. The PTM synchronization is initiated from a downstream requester (USP) to an upstream responder (DSP). The controller handles the entire timestamp update process without software intervention. Each current ($t_{\#n}$) and previous timestamp ($t_{\#n-1}$) value as well as other context information (clock) can be read from the PTM VSEC registers by software. The controller enables Enhanced Precision Time Management (ePTM) for all the devices, by default.

3.25.1 PTM Timers

The controller implements 1us and 100us timers as per the requirements of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The controller also implements a 10 ms timer which is used for automatic update of PTM context by the Requester.

3.25.2 Transmit and Receive Latencies

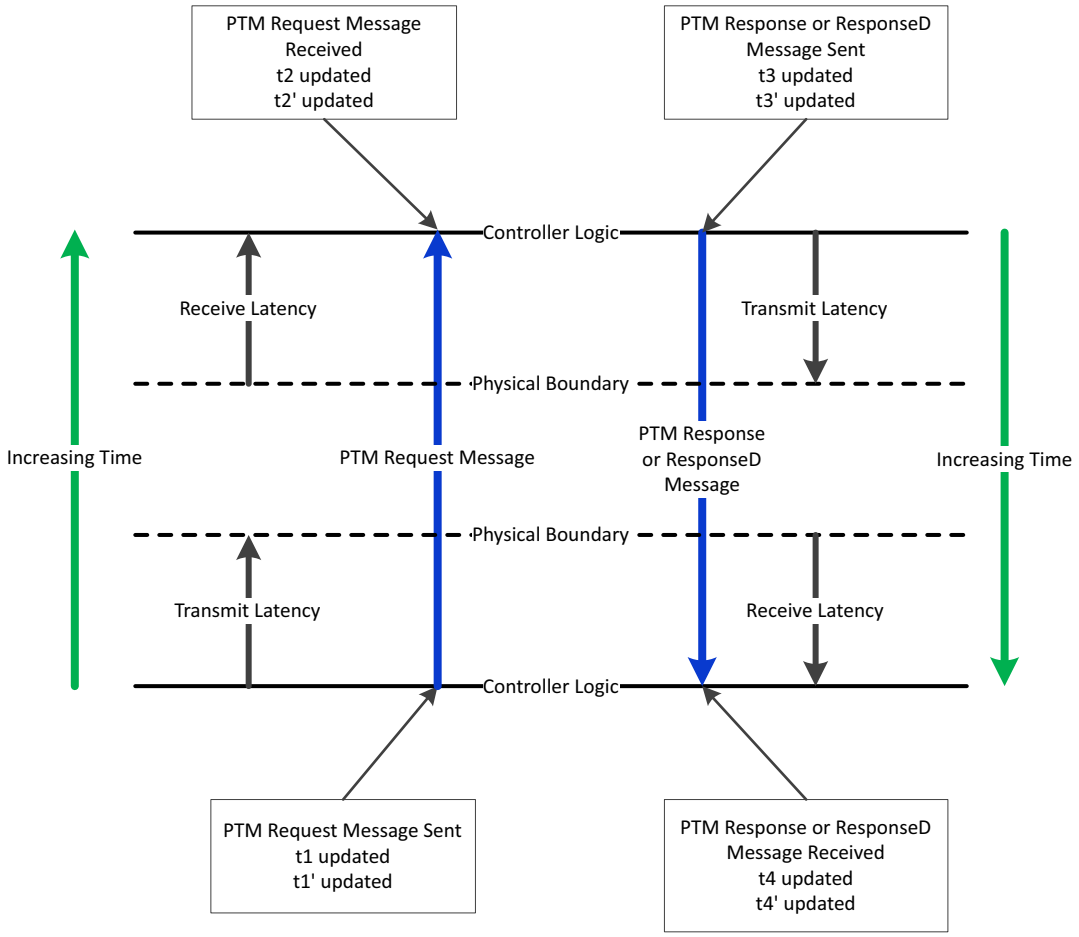
A key operational goal of PTM is to stamp the time at which a message is transmitted from the physical boundary of the PCIe device. The Tx latency is the time that it takes a message to travel from the boundary of the controller to the boundary of the PCIe device. It is added to the time at which the message left the controller. The Rx latency is the time that it takes a message to travel from the boundary of the PCIe device to the boundary of the controller. It is subtracted from the time at which the message reaches the controller. The Latency registers may be written to at any time with a more accurate value if required.For example:

- t1 = Local PTM Clock time when message leaves controller + `PTM_REQ_TX_LATENCY_OFF.PTM_REQ_TX_LATENCY`
- t3 = Local PTM Clock time when message leaves controller + `PTM_RES_TX_LATENCY_OFF.PTM_RES_TX_LATENCY`
- t2 = Local PTM Clock time when message received by controller - `PTM_RES_RX_LATENCY_OFF.PTM_RES_RX_LATENCY`
- t4 = Local PTM Clock time when message received by controller - `PTM_REQ_RX_LATENCY_OFF.PTM_REQ_RX_LATENCY`

Note: The Tx latency includes all delays from the moment timestamp is captured, to the time the message reaches the port pins. The Rx latency includes all delays from the time the message reaches the port pins, to the time the timestamp is captured. These delays depend on the characteristics of the PHY used, skew on the link, and deviation between the reference clocks of the two ports. There may be a dependency on link width and the current link speed also.

For best accuracy the Tx and Rx latency values should be programmed during initialization, for each supported link speed. For more information, see "Programming Tx and Rx Latencies for Each Link Speed" on page 326. The controller automatically selects the Tx and Rx latency based on current link speed.

Figure 3-133 Tx and Rx Latencies



3.25.2.1 Programming Tx and Rx Latencies for Each Link Speed

The Tx and Rx Latency registers are programmed through an indirect addressing scheme using an index register (`PTM_[RES|REQ]_LATENCY_REG_SEL_OFF`), to reduce the address footprint in the PCI Express extended configuration space. The index register has Responder/Requester Latency Register Write Select field (`PTM_[RES|REQ]_LATENCY_REG_SEL`), referred to as index, to determine which Tx or Rx Latency viewport register (`PTM_RES_[TX|RX]_LATENCY_OFF`) to program/read for each link speed. Table 3-96 provides the index to link speed mapping.

Table 3-96 Index to Link Speed Mapping

PTM_[RES|REQ]_ESM_SEL	PTM_[RES|REQ]_LATENCY_REG_SEL	Link Speed (GT/s)
PCI Express Speed Mode		
0	0	2.5
0	1	5

PTM_[RES|REQ]_ESM_SEL	PTM_[RES|REQ]_LATENCY_REG_SEL	Link Speed (GT/s)
0	2	8
0	3	16
0	4	32
CCIX ESM Speed Mode (CX_CCIX_ESM_SUPPORT =1)[a]		
1	0	20
1	1	25

a. Additional ESM Speeds

If the Tx and Rx latency registers are not programmed, the default latency values are the same for all speeds, and are set by configuration parameters, as follows:

- Tx Latency =`CX_PHY_TX_DELAY_PHY` + `CX_PHY_TX_DELAY_MAC`
- Rx Latency =`CX_PHY_RX_DELAY_PHY` + `CX_PHY_RX_DELAY_MAC`

Note: The default values for Tx and Rx latency registers are defined in clock cycles. You must program the appropriate registers to set the latencies in ns.

Programming Examples

Example 1: To setup PTM Requester Tx Latency to 56 ns and Rx Latency to 334ns for 2.5 GT/s:

1. Setup speed mode (PCI Express or CCIX ESM):

   Write `1'b0` to `PTM_REQ_ESM_SEL`, to set PCI Express speed mode.
2. Setup the index:

   Write `32'h0000_0000` to `PTM_REQ_LATENCY_REG_SEL`, to select 2.5 GT/s Latency viewport Tx and Rx registers.
3. Write the Tx/Rx Latency values:

   Write `32'h0000_0038` to `PTM_REQ_TX_LATENCY`, to set 2.5 GT/s Tx latency to 56 ns.

   Write `32'h0000_014E` to `PTM_REQ_RX_LATENCY`, to set 2.5 GT/s Rx latency to 334 ns.

Example 2: To setup PTM Responder Tx Latency to 24ns and Rx Latency to 115ns for 8 GT/s:

1. Setup speed mode (PCI Express or CCIX ESM):

   Write `1'b0` to `PTM_RES_ESM_SEL`, to set PCI Express speed mode.
2. Setup the index:

   Write `32'h0000_0002` to `PTM_RES_LATENCY_REG_SEL`, to select 8 GT/s Latency viewport Tx and Rx registers.
3. Write the Tx/Rx Latency value:

   Write `32'h0000_0018` to `PTM_RES_TX_LATENCY`, to set 8 GT/s Tx latency to 24 ns.

   Write `32'h0000_0073` to `PTM_RES_RX_LATENCY_OFF` register, to set 8 GT/s Rx latency to 115 ns.

Example 3: To setup PTM Requester Tx Latency to 7ns and Rx Latency to 18ns for 20 GT/s (`CX_CCIX_ESM_SUPPORT =1`)

1. Setup speed mode (PCI Express or CCIX ESM)

   Write `1'b1` to `PTM_REQ_ESM_SEL`, to set CCIX ESM speed mode
2. Setup the Index

   Write `32'h0000_0004` to `PTM_REQ_LATENCY_REG_SEL`, to select 20 GT/s Latency viewport Tx and Rx registers
3. Write the Tx/Rx Latency value

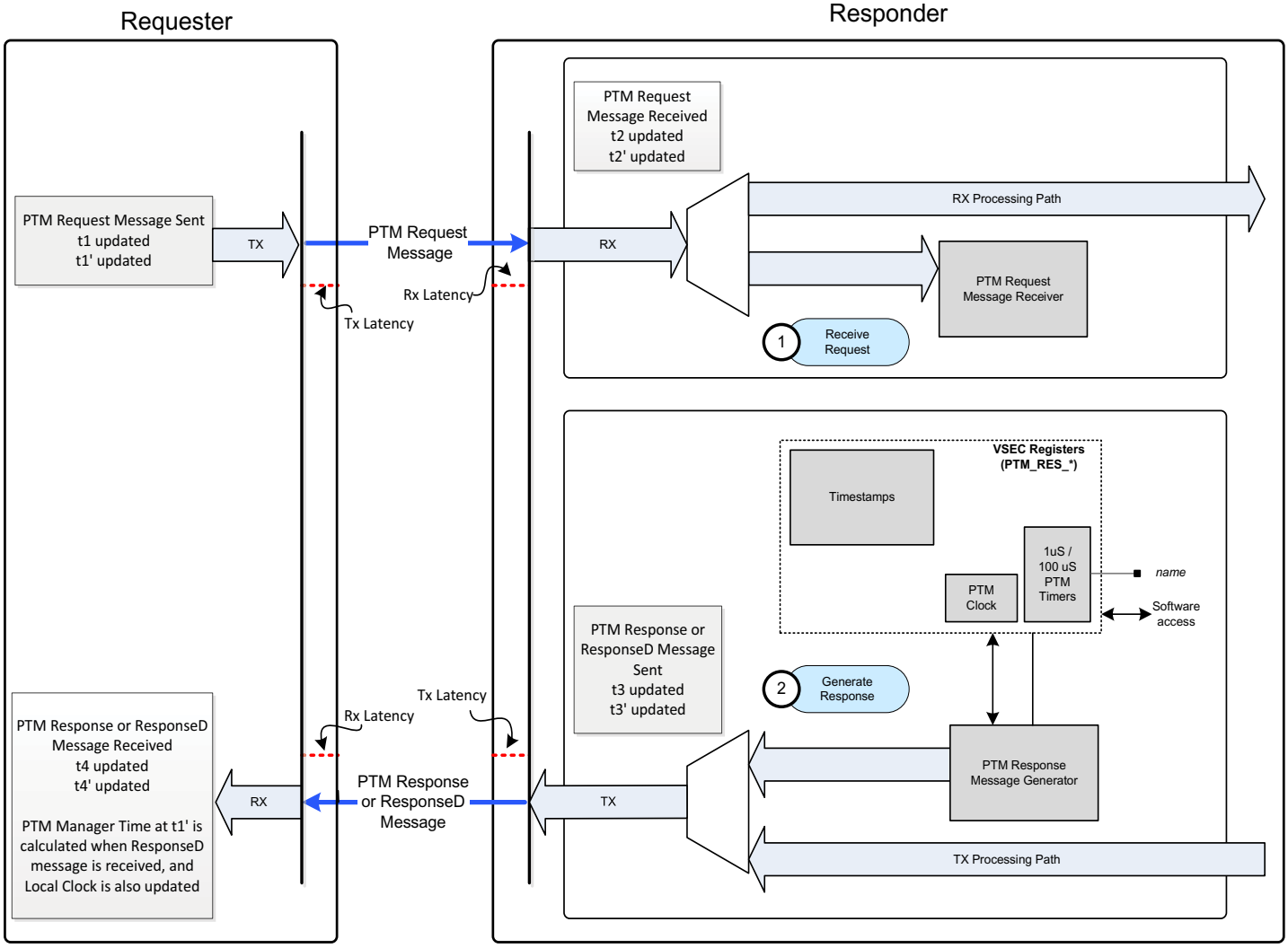
   Write `32'h0000_0007` to `PTM_REQ_TX_LATENCY`, to set 20 GT/s Tx latency to 7nsWrite `32'h0000_0012` to `PTM_REQ_RX_LATENCY`, to set 20 GT/s Rx latency to 18ns

Note You can inspect the VTB test case `tl_ptm_latency.sv` at `<workspace>/doc/html/vtb/testenv/classtl__ptm__latency.html.` For more information on how to determine latency values by simulation, see “Detailed Description” section.

3.25.3 Responder (DSP) Features

- Integrated hardware for scheduling of PTM responses to dialog requests when context is valid
- Timestamp generation when PTM Messages are detected

Figure 3-134 PTM Responder (DSP)



- Local PTM Clock
  - If `CX_PTM_EXTERNAL_MASTER_TIME =1` an external master time can be used to set the clock value

    Note: The external master time is latched when `ptm_external_master_strobe` signal is set to 1.
  - Can also be updated by software
  - Can be read by software
  - Available through top-level port
- Context automatically invalidated when:
  - `core_clk` stops, or runs at the wrong frequency (for example, in L1 substates or when the link speed is changing), or
  - When PTM is disabled by clearing the `PTM_ENABLE` bit in the `PTM_CONTROL_OFF` register

  When the controller asserts `ptm_responder_rdy_to_validate` signal, your application can re-validate the context by either,

  - [CX_PTM_EXTERNAL_MASTER_TIME =1]: Updating the local PTM clock using the ptm_external_master_time and ptm_external_master_strobe signals, which automatically asserts the ptm_context_valid signal, or
  - [DBI Access]: Updating the local PTM clock by writing to PTM_RES_LOCAL_LSB_OFF and PTM_RES_LOCAL_MSB_OFF registers, and then setting the PTM_RES_CCONTEXT_VALID field of PTM_RES_CONTROL_OFF to 1, which automatically asserts ptm_context_valid signal
- Root Support
  - Ideally a PTM root behaves like a responder with permanently valid context. However,
    - If the PTM root enters low power modes or changes link frequency, PTM context is invalidated (indicated by ptm_context_valid =0)
    - Following PTM context invalidation, the application must re-validate PTM context, by using either the PTM master time interface signals or the PTM responder VSEC registers, as described in the previous list item
  - PTM messages must not contain TLP prefixes
- Context information stored in PTM Responder VSEC (PTM_RES_*_OFF):
  - Provides access to timestamps of most recent context refresh
  - Provides notification when context is valid
  - Programmable Tx and Rx latency values for each link speed, automatically selected based on current link speed
- When duplicate PTM TLP is received or a replay TLP is sent, the responder invalidates historical timestamps (t3 - t2)

3.25.4 Local PTM Clock Scaling

PTM clock scaling feature allows the local PTM clock to increment a programmable non-integer nanosecond amount in each controller clock period. This allows the PTM local time to remain coherent if there is a frequency deviation between PTM Requester and Responder. PTM clock scaling also allows the controller to use PTM when the clock period is a non-integer nanosecond multiple, for example when running at CCIX ESM data rates.

3.25.4.1 PTM Responder

For the PTM Responder, your application fully controls PTM local clock updates. If the external master clock runs from the same source clock as the PCIe controller, clock scaling is not needed. However, for applications where the external master and PCIe controller run on unrelated clock sources, it may be necessary to scale the local PTM clock period.

To enable this feature set the PTM_RES_SCALED_CLOCK_T_EN field of PTM_RES_SCALED_CLOCK_T_OFF register to '1' (this bit can only be set when ptm_context_valid is set). You can set a 24-bit value consisting of an 8-bit integral part (nanoseconds) and 16-bit fractional part ($1/2^{16}$ nanoseconds) using PTM_RES_SCALED_CLOCK_T_INT and PTM_RES_SCALED_CLOCK_T_FRAC fields of the PTM_RES_SCALED_CLOCK_T_OFF register, which represents the scaled local PTM clock period.

The clock scaling procedure is as follows:

- Your application initializes the PTM Responder with a known time from an external master

- The PTM Responder counts local PTM time using the Nominal PTM Clock Period, which your application can read from the `PTM_RES_NOM_CLOCK_T_OFF` register
- Your application waits for a known time and reads the local PTM clock value again
- Your application calculates the scaled PTM clock period using the deviation between the external master clock and the local PTM clock, and writes the scaled PTM clock period to PTM Responder's Scaled PTM Clock Period register (`PTM_RES_SCALED_CLOCK_T_OFF`), which represents the scaled local PTM clock period

If the PTM context becomes invalid your application must reiterate the steps listed in this section.

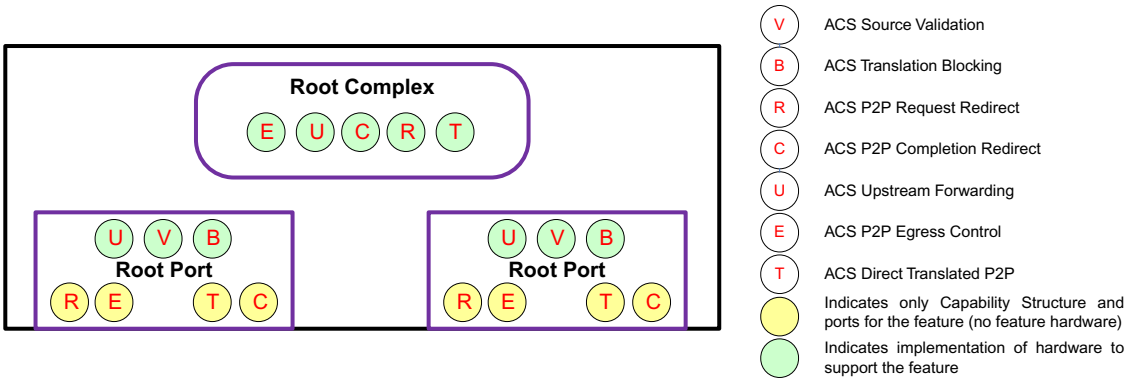
3.25.4.2 Use of Scaled Clock Period and Clock Rate Changes

The PTM Requester and PTM Responder scaled clock period enable bits are cleared when the `core_clk` rate changes, and can only be set when the clock rate change is complete. The `core_clk` rate is controlled by LTSSM, using the `mac_phy_rate` signal. When `mac_phy_rate` changes, the register bits are cleared. When the current data rate has been updated to match `mac_phy_rate`, setting of the bits is enabled.

3.26 Access Control Services (ACS)

The controller supports Access Control Services (`CX_ACS_ENABLE =1`) as described in *PCI Express Base Specification, Revision 6.0, Version 1.0*.

Figure 3-135 ACS Overview



Features and Limitations:

- A DSP (single function or multi-function device) supports all access control types listed in Table 3-97.
- Per-port egress control vector mask can be set using the parameter `CX_ACS_DW_EGRESS_CTRL_MASK_#` for DSP. Setting a 1 in the egress control vector mask makes the corresponding bit in the egress control vector read only.

  Note: The egress control vector masks can be written through the DBI using the `dbi_cs2` select at the same address as the egress control vector.

Table 3-97 ACS Violation Error Handling DSP

Access Control Type	Parameter to be set in addition to CX_ACS_ENABLE	ACS Violation Error Handling
ACS Source Validation (V)	`CX_ACS_SRC_VALID`	■ The controller returns CA status CPL for NP requests. ■ The controller reports AER: ACS Violation Error. Note: For NP Requests, If ACS violation is non-fatal, Advisory Non-Fatal Error is reported. ■ The controller sets the Signaled Target Abort bit in the PCI compatible Status, or Secondary Status register.
ACS Translation Blocking (B)	`CX_ACS_AT_BLOCK`	
ACS Upstream Forwarding (U)	`CX_ACS_UP_FORWARD`	■ Not an error condition.[a]

Access Control Type	Parameter to be set in addition to CX_ACS_ENABLE	ACS Violation Error Handling
ACS P2P Request Redirect (R)[b]	`CX_ACS_P2P_REQ_REDIRECT_n` `(n =0; n <=CX_NFUNC-1)`	■ The controller does not take any action. ■ Your application logic should generate CPL for NP requests. ■ Your application logic can use the controller signal `app_err_bus[12]` to detect ACS violation, and take action accordingly.
ACS P2P Completion Redirect (C)[a]	`CX_ACS_P2P_COMPL_REDIRECT_n` `(n =0; n <=CX_NFUNC-1)`	
ACS P2P Egress Control (E)	`(n =0; n <=CX_NFUNC-1)`	
ACS Direct Translated P2P (T)	`CX_ACS_P2P_DIRECT_TRANSL_n` `(n =0; n <=CX_NFUNC-1)`	

a. When CX_ACS_UP_FORWARD =1, DSP forwards TLPs within the memory/IO ranges upstream. When CX_ACS_UP_FORWARD =0, these TLPs are treated as UR.
b. It is not used by the controller internally to gate peer-to-peer traffic.

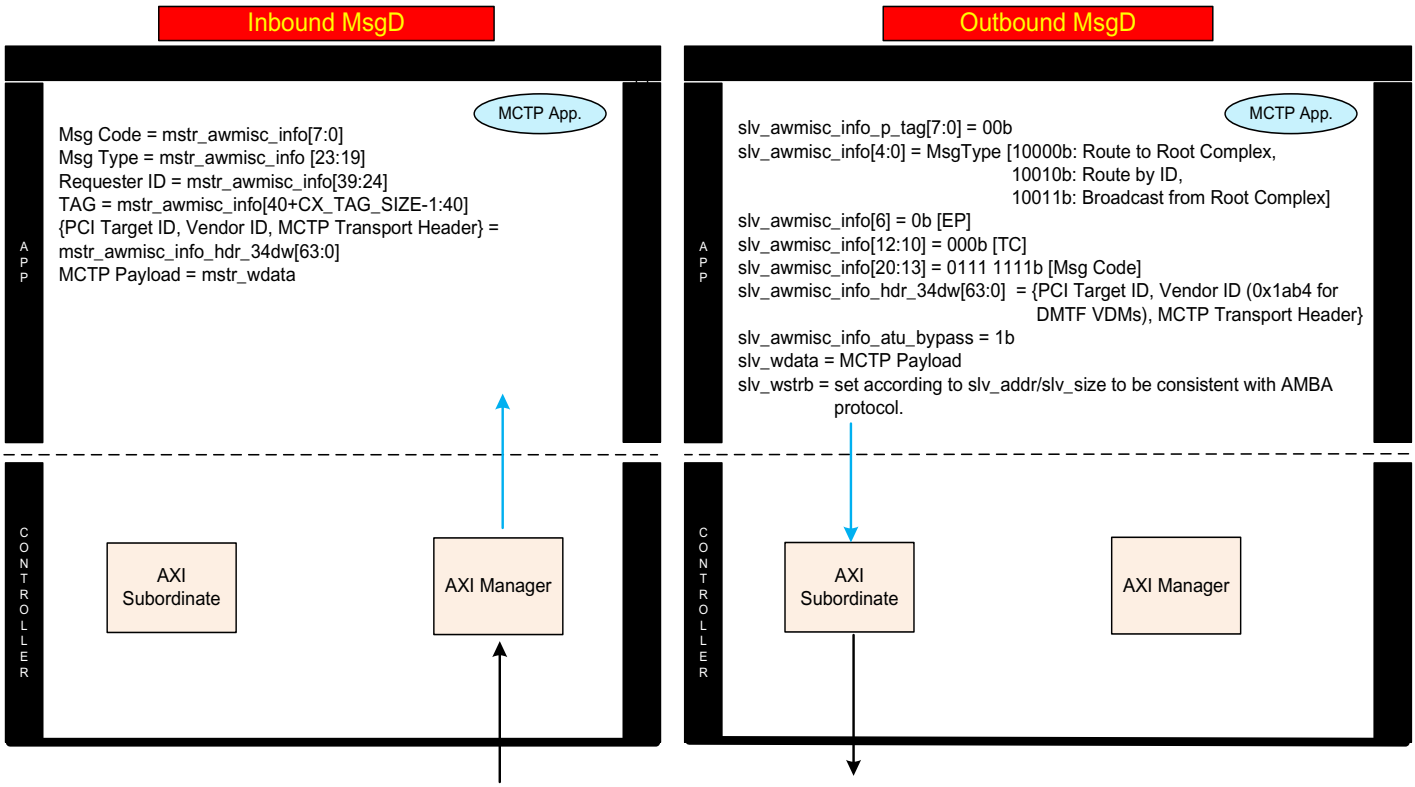
3.27 Management Component Transport Protocol (MCTP)

MCTP messages are treated as Type1 VDMs with Data by the controller. MCTP messages use the MCTP VDM code value (0000b) that uniquely differentiates MCTP messages from other Distributed Management Task Force (DMTF) VDMs. MCTP messages can be transmitted/received using AXI bridge sideband signals, as shown in Figure 3-136.

Limitations:

- Outbound MCTP message decomposition is not supported because the AXI bridge does not support decomposition of MsgD. Your application must split MCTP messages into MCTP packets with payload data size less than Max_Payload_Size to prevent any unprecedented data corruption.
- For inbound MCTP packets, the payload data size is limited to (`CC_MSTR_MTU`). MCTP message assembly is not supported by the controller. Your application must implement logic to assemble MCTP packets to form MCTP message.

Figure 3-136 AXI Sideband Signal Usage for Inbound/Outbound MCTP Messages



3.28 Downstream Port Containment (DPC)

The controller supports the optional Downstream Port Containment (DPC) feature of a Downstream Port as defined in *PCI Express Base Specification, Revision 6.0, Version 1.0* when the parameter `CX_DPC_ENABLE` is set to '1'. When a DPC enabled Downstream Port detects an unmasked uncorrectable error at or below the port, it halts the traffic below the port to avoid the potential spread of any data corruption. The Downstream Port reports the errors to your application.

The DPC is triggered as required and described in section 6.2.10 of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

When DPC is triggered, the LTSSM of the controller is directed to the Disabled state. The LTSSM stays in Disabled state until your application clears the DPC Status register. When cleared, the LTSSM proceeds to the Detect state.

Features and Limitations

- DPC interrupts are supported
- DPC ERR_COR Signaling is supported
- Root Port Programmed I/O (RP PIO) Error Controls are supported
- Auto Slot Power Limit Disable is supported

Error Handling

The Root Error Status (`ROOT_ERR_STATUS_OFF`) register is updated depending on the values of the DPC Trigger Enable (`DPC_CAP_CONTROL_OFF.DPC_DPC_TRIG_EN`) and DPC Trigger Status (`DPC_STATUS_ERR_SOURCE_ID_OFF.DPC_DPC_TRIG_STATUS`) register fields as described in Table 3-98.

Table 3-98 Impact of DPC Trigger Enable and DPC Trigger Status on Root Error Status Register

DPC Trigger Enable	DPC Trigger Status	Impact on Root Error Status Register Fields
01	0	■ ERR_COR_RX and MUL_ERR_COR_RX can be updated by all correctable error messages ■ NON_FATAL_ERR_MSG_RX can be updated by received ERR_NONFATAL
	1	■ ERR_COR_RX and MUL_ERR_COR_RX can be updated by internally generated correctable error and not by received ERR_COR
	0 or 1	■ ERR_FATAL_NON_FATAL_RX and MUL_ERR_FATAL_NON_FATAL_RX cannot be updated by received ERR_FATAL or ERR_NONFATAL ■ FATAL_ERR_MSG_RX cannot be updated by received ERR_FATAL
10	0	■ ERR_COR_RX and MUL_ERR_COR_RX can be updated by all correctable error messages
	1	■ ERR_COR_RX and MUL_ERR_COR_RX can be updated by internally generated correctable error and not by received ERR_COR
	0 or 1	■ ERR_FATAL_NON_FATAL_RX and MUL_ERR_FATAL_NON_FATAL_RX cannot be updated by received ERR_FATAL or ERR_NONFATAL ■ NON_FATAL_ERR_MSG_RX cannot be updated by received ERR_NONFATAL

The AER Uncorrectable Error Status (`UNCORR_ERR_STATUS_OFF`) register is updated depending on the values of the DPC Trigger Enable (`DPC_CAP_CONTROL_OFF.DPC_DPC_TRIG_EN`) and DPC Trigger Status (`DPC_STATUS_ERR_SOURCE_ID_OFF.DPC_DPC_TRIG_STATUS`) register fields. The controller updates AER Uncorrectable Error Status register only after your application clears DPC Trigger Status as specified in Section 6.2.5 of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. Hence, till your application clears DPC Trigger Status, the AER Uncorrectable Error Status register stores errors corresponding to the TLP which triggered DPC.

Auto Slot Power Limit Disable

When Auto Slot Power Limit Disable field in the Slot Control register is set (`SLOT_CONTROL_SLOT_STATUS.PCIE_CAP_AUTO_SLOT_PWR_LMT_DIS =1)`, the controller does not automatically send Set_Slot_Power_Limit message. Hence, Set_Slot_Power_Limit message is not sent automatically by a Downstream Port when the link transitions from a non-DL_Up status to a DL_Up status.

Flushing Mechanism for AXI Configurations

When the controller enters the DPC,

- It uses 'Automatic Graceful Flushing of Bridge Before Reset When Link Goes Down' feature.
- Your application can access the registers during flushing.

Flushing Mechanism for Native Controller Interfaces

When the controller enters DPC,

1. It requests your application to flush all downstream TLPs by asserting the output signal `rstctl_core_flush_req`.
2. After your application completes flushing, it should return an acknowledgment by asserting the input signal `app_flush_ack`. The controller can then be reset when it exits DPC.

While the controller is being flushed,

1. The receive completion timeout output signals provide header information for the flushed outstanding expected completions which are stored in Receive Completion LUT in the RADM. For more information on the receive completion timeout output signals, see "SII: Receive Control/CPL Timeout Signals" in the "Signal Descriptions" chapter of the *Reference Manual*.
2. These completions have UR or CA status depending on the value of the `DPC_DPC_CPL_CNTRL` field of the `DPC_CAP_CONTROL_OFF` register. Your application can also use the output signal `cfg_dpc_cpl_cntrl` to determine the completion status.

eDPC (RP PIO Error Controls Support)

You can enable the RP PIO error controls by setting the internal parameter `CX_DPC_RP_PIO_EXTNS` to '1'.

RP PIO error control mechanism includes fine-grained controls for managing Programmed I/O (PIO) error, and Completion synthesis to avoid Completion Timeouts for outstanding Non-Posted Requests.

3.29 Deferrable Memory Write (DMWr)

You can enable Deferrable Memory Write (DMWr) support in the controller by setting the parameter `CX_DEF_MEM_WR_ENABLE` to '1'.

Note Configuration Request Retry Status (CRS) is renamed as Request Retry Status (RRS). RRS has the same value and meaning as CRS for Configuration requests. In addition, RRS is also used to defer DMWr requests.

3.29.1 Native PCIe Controller DMWr Support

DMWr Outbound Requests

The controller manages the DMWr outbound requests as Non-Posted requests with respect to credits and completion qtimeout. Software is allowed to initiate DMWr requests only when,

- The controller is configured as a DMWr requester (`CX_DEF_MEM_WR_REQ_EN =1`) and the DMWr Requester Enable field of the DEV3 Control register is set (`DEVICE_CONTROL3_REG.DEV3_CAP_DM-WR_REQ_EN =1`).
- The controller is configured as a DMWr router (`CX_DEF_MEM_WR_ROUTING_EN =1`) and Egress Blocking field of the DEV3 Control register is disabled (`DEVICE_CONTROL3_REG.DEV3_CAP_DM-WR_EGRESS_BLK =0`).

DMWr Inbound Requests

The controller manages the DMWr inbound requests as Non-Posted requests with respect to credits, completion filtering, and address matching. The controller can process inbound requests only when,

- The controller is configured as a DMWr completer (`CX_DEF_MEM_WR_CPL_EN =1`). A Switch or a Root Port controller that is configured as a DMWr router (`CX_DEF_MEM_WR_ROUTING_EN =1`)

Inbound DMWr Requests exceeding the maximum amount configured in `CX_DEF_MEM_WR_LEN_SUPP` are trashed and a completion with CA status is returned.

3.29.1.1 DMWr Routing - Egress Blocked Support

When DMWr Egress Blocking is enabled through the Link Control 3 Register in an egress port, and a DMWr request is scheduled to or goes out that port, that port must:

- Handle the request as an DMWr Egress Blocked error by setting the appropriate bit in the Uncorrectable Error Status Register.
- Return a completion with a completion status of UR.

A possible way to do this (using a Switch as an example) is:

- Set DMWr Egress Blocking bit in the Link Control 3 Register.
- Design a DMWr Detector in your application logic that:
  a. Detects a DMWr forwarded by the Ingress/Downstream Port.
  b. Discards (blocks) that DMWr.
  c. Generates downstream completion with status of UR, for the blocked DMWr.

d. Sets the DMWr Egress Blocked error status bit in the egress port using the `app_err_bus[14]`.

3.29.1.2 Limitations

- DMWr support is limited to Gen3 configurations and above (`CX_GEN3_SPEED >3`)
- DMWr requests addressing TRGT0 are not supported.
- Outbound DMWr TLP type cannot be nullified.
- TPH is not supported on DMWr requests.
- Device 3 extended capability is currently supported only on PFs.

3.29.2 AXI Bridge Deferrable MemWr Support

The AXI Bridge supports optional DMWr requests on the subordinate and master interfaces.

3.29.2.1 DMWr Requests on the AXI subordinate Interface

DMWr requests are accepted on the AXI subordinate when the TLP Type field of `slv_awmisc_info` is set to 5'b1_1011.

- The AXI bridge uses separate queues for Posted and Non-Posted requests. To enforce "NP must not pass a previously-issued P" ordering rule, the application master must wait for the response to the last posted request to be returned before issuing the DMWr request.

  For more information on ordering rules, see "AXI Bridge Ordering" on page 408.
- DMWr maximum request size is determined by the parameter `CX_DEF_MEM_WR_LEN_SUPP` value. Any DMWr request exceeding this value is dropped internally by the AXI bridge and a UR response is returned; the UR response can be mapped to SLVERR, DECERR, or OKAY on the AXI subordinate. In all cases UR is signaled on the `slv_bmisc_info` CPL status field.
- Although DMWr WRAP requests are supported on the AXI subordinate they should be avoided as the request is split into two requests before it is issued on the wire.
- Use of drop to poison (`slv_wmisc_info_ep`) or drop a request (`slv_wmisc_info_silent`) is not supported.
- Decomposition is not required since the DMWr request size is limited to 64 or 128 bytes.

3.29.2.2 DMWr Requests on the AXI Manager Interface

The AXI manager issues DMWr requests received by the wire.

- Decomposition is supported for DMWr requests; decomposition rules for DMWr requests are identical to MWr requests, that is, a DMWr request is decomposed when it exceeds the AXI manager maximum transfer unit (`CC_MSTR_BURST_LEN * MASTER_BUS_DATA_WIDTH`) or crosses an AXI manager page boundary (`2^(CC_MSTR_PAGE_BOUNDARY_PW-1)`)
- RRS responses can be signaled on the AXI manager B channel by returning an OKAY response and setting the corresponding mapping on the `mstr_bmisc_info_cpl_stat` (3'b010); SLVERR and DECERR responses on the AXI subordinate are mapped to CA or UR according to the pre-existent rules for inbound Non-Posted requests.

- The error response is accumulated and RRS is returned if either of the decomposed requests receives that response on the AXI manager.
- To avoid decomposition of inbound DMWr requests set the AXI manager page boundary to 4K and ensure the AXI manager MTU is greater than 128 bytes.

3.30 Enhanced Configuration Access Mechanism (ECAM)

The controller supports the optional Enhanced Configuration Access Mechanism (ECAM) feature of a Downstream Port as defined in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. To enable the ECAM feature you must set the parameter `CX_SBSA_ECAM` to '1'.

AXI Bridge ECAM Support

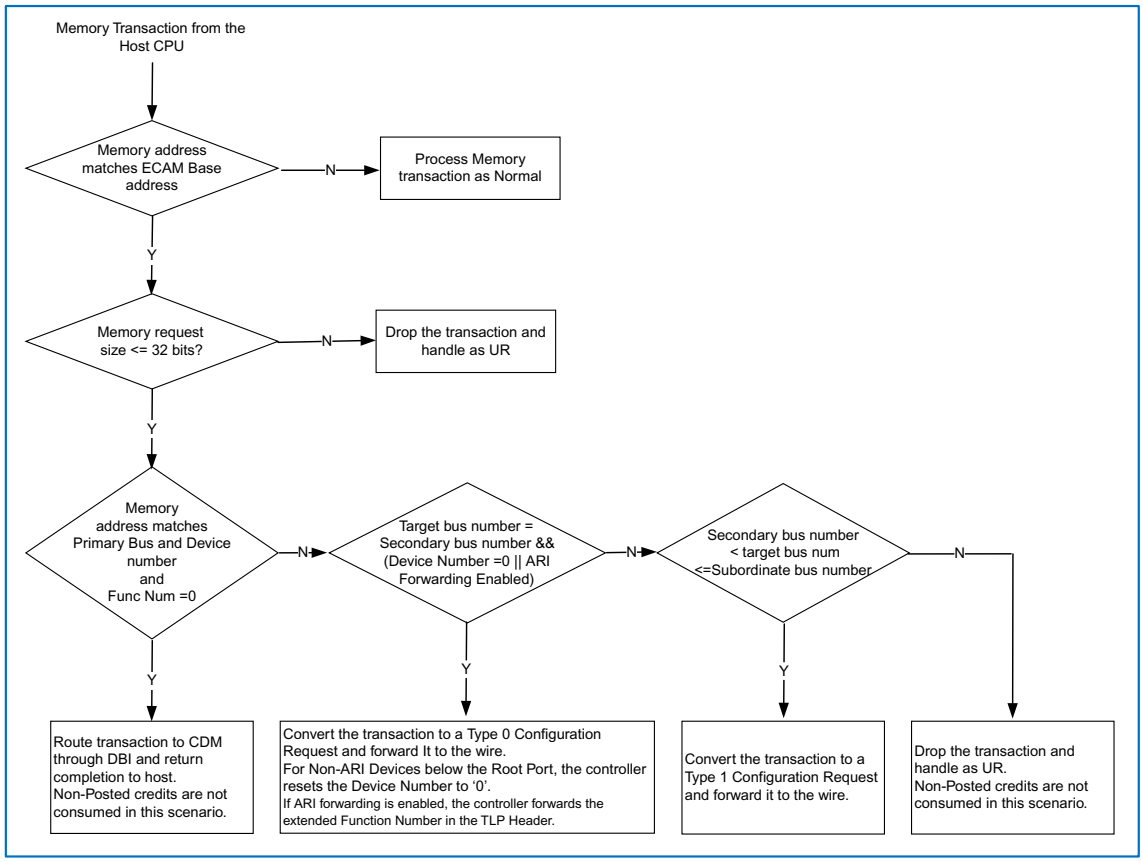
In AXI Bridge configuration (`AMBA_INTERFACE >1`), in addition to setting `CX_SBSA_ECAM` to '1', you must set the `SHARED_DBI_ENABLED` parameter to '1' and `ECAM_CTRL_DSP_ECAM_EN` field of the `ECAM_CTRL_OFF` register to '1' to enable the ECAM feature.

To allow standard enumeration software to work with the controller, the Root Complex driver must program the size and base address of the ECAM region in the in the ECAM Base Address register, that is, `ECAM_BASE_ADDR_[LWR/UPPER]`. You can program this register through the DBI.

To facilitate device enumeration, the controller,

- Converts memory transactions from the host CPU into Configuration Requests
- Forwards the Configuration Requests not targeted to the Primary Bus Number to the PCIe fabric, based on the target address
- Serializes the Configuration Requests

Figure 3-137 AXI Bridge Memory Transaction Handling



Note When ECAM Base Address matches the memory transaction address, the controller considers all the 28 bits of the LSB. It handles the transaction as UR when the bits are not within primary bus to subordinate bus range.

By default, the controller performs Type 1 Configuration Request to Type 0 conversion. To prevent Type 1 Configuration Request to Type 0 conversion, you can set the `ECAM_CTRL_TYPE1_TO_TYPE0_DISABLE` field of the `ECAM_CTRL_OFF` register to '1'.

Native PCIe Controller ECAM Support

In native PCIe controller configuration, by default, the controller performs Type 1 Configuration Request to Type 0 conversion. The controller compares the Bus Number of each Type 1 Configuration Request with the Secondary Bus Number of the port. If it matches, the controller converts the Type 1 Configuration request to Type 0.

To prevent Type 1 Configuration Request to Type 0 conversion, you can set the `ECAM_CTRL_TYPE1_TO_TYPE0_DISABLE` field of the `ECAM_CTRL_OFF` register to '1'.

3.31 Native PCIe Enclosure Management (NPEM)

The controller supports Native PCIe Enclosure Management feature as described in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. NPEM defines mechanisms for simple storage enclosure management for PCIe SSDs. To enable NPEM set the parameter `NPEM_ENABLE` to '1'.

Features and Limitations

- All non-optional and optional NPEM states are supported
- The controller implements all the bits in the Enclosure-specific Controls as RW when the NPEM Enclosure-specific capability register has a non-zero value.
- The controller implements the `NPEM_CMD_COMPLETE` register field as RW1C and `NPEM_ENCLOSURE_SPECIFIC_STATUS` register field as RO.

3.32 Flit Performance Measurement Extended Capability

The controller supports the optional Flit Performance Measurement Extended Capability according to the *PCI Express Base Specification, Revision 6.0, Version 1.0*. You can enable this by setting the `FLIT_PERF_MEASUREMENT_ENABLE` parameter to '1'.

The controller supports only LTSSM Performance Measurement Status Register 1. The controller does not support optional LTSSM Performance Measurement Status Register 2 to Register 5.

Note The controller hardwires FLIT_PERFORM_LTSSM_REG_COUNT field in the FLIT_PERFORM_CAPABILITIES_REG is to '1'. This is read only field.

3.32.1 Flit Latency Tracking

Table 3-99 shows the Flit latency measurement criteria.

Table 3-99 Measurement Tracking for Flit Latencies

Latency	Start Time at Layer 1 (XMLH)	End Time at Receive Logic (RPLH)
Flit to ACK	XMLH accepts last symbol of the transmit Flit.	Receive logic (RPLH) indicates ACK to the transmit logic (XPLH) for the sequence number being tracked.
Flit to NAK		RPLH indicates NAK to the transmit logic (XPLH) for the sequence number being tracked.
NAK to Retry	XMLH accepts last symbol of a Flit carrying a NAK.	RPLH indicates ACK to the transmit logic (XPLH) for the sequence number being replayed.

Note The controller sends an ACK/NAK with a sequence number greater than that of the transmitted NAK to acknowledge the successful receiving of the requested retry flit.

When software sets the `FLIT_PERFORM_LATENCY_MEASURE_EN` field of the `FLIT_PERFORM_CNTRL_REG` register to '1', the controller sets the `FLIT_PERFORM_LATENCY_TRACK_STAT` field of the `FLIT_PERFORM_STAT_REG` to '01' and starts latency tracking.

Table 3-100 shows the action taken by the controller based on software programmed values of the `FLIT_PERFORM_CNTRL_REG` register when `FLIT_PERFORM_LATENCY_TRACK_STAT =1`.

Table 3-100 Flit Latency Tracking when FLIT_PERFORM_LATENCY_TRACK_STAT =01

FLIT_PERFORM_TYPE_TO_MEASURE	FLIT_PERFORM_RESPONS_TYPE	Action
xx	000	No latency tracking
11	xxx	No latency tracking

FLIT_PERFORM_TYPE_TO_MEASURE	FLIT_PERFORM_RESPONS_TYPE	Action
01/10	100	Track NAK to Retry latency for Payload Flits
	010	Track Flit to NAK latency for Payload Flits
	001	Track Flit to ACK latency for Payload Flits
	011	Track Flit to ACK or NAK latency for Payload Flits
	111	Track Flit to ACK or NAK latency for Payload Flits Track NAK to Retry latency for Payload Flits
00	100	No latency tracking. IDLE/NOP Flits do not carry unique sequence numbers and are not tracked for latency.
	010	
	001	
	011	
	111	

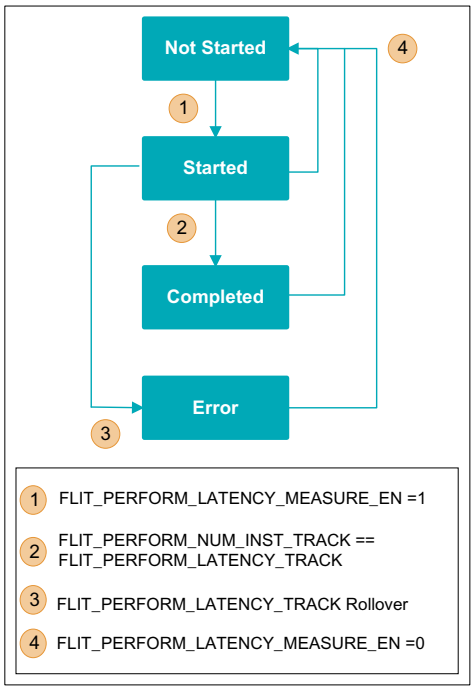
When `FLIT_PERFORM_LATENCY_TRACK_STAT =00, or 10, or 11` there is no latency tracking.

Only one Flit is tracked at any one time for Flit to ACK/NAK latency and NAK to Replay.

Flit Latency Tracking Sequence

Figure 3-138 shows the hardware update method for the `FLIT_PERFORM_LATENCY_TRACK_STAT` field in the `FLIT_PERFORM_STAT_REG` register.

Figure 3-138 FLIT_PERFORM_LATENCY_TRACK_STAT Update Method



The following sequence shows the Flit latency tracking method:

1. Software enables tracking by programming the `FLIT_PERFORM_CNTRL_REG` as mentioned in Table 3-100
2. Layer2 logic, resets the tracker counter and waits for valid transmit Flit to begin tracking latency
   - Flit to ACK/NAK latency: Any payload Flit
   - NAK to Retry Latency: Any Flit transmitting NAK
3. Tracker counter is incremented, with 8ns granularity, until the ending latency event is observed
   - Flit to ACK/NAK latency: ACK/NAK received for the sequence number being tracked
   - NAK to Retry Latency: Requested Retry Flit is received and can be ACK'ed
4. Tracker counter value is reported to the CDM for recording in the `FLIT_PERFORM_STAT_REG` register
   - `FLIT_PERFORM_LTSSM_STAT_TRACK_COUNTER` field is updated with the latency count
   - `FLIT_PERFORM_INTRPT_TRIGGER_EVENT` field is updated (if conditions met)
   - `FLIT_PERFORM_LATENCY_TRACK` field is incremented
   - `FLIT_PERFORM_LATENCY_TRACK_STAT` field is updated (if conditions met)
5. If `FLIT_PERFORM_LATENCY_TRACK =1`, repeat from step 2.

When the `FLIT_PERFORM_CNTRL_REG.FLIT_PERFORM_NUM_INST_TRACK =0`, the `FLIT_PERFORM_STAT_REG.FLIT_PERFORM_LTSSM_STAT_TRACK_COUNTER` contains the highest latency value observed since tracking was started.

When the `FLIT_PERFORM_CNTRL_REG.FLIT_PERFORM_NUM_INST_TRACK !=0`, the `FLIT_PERFORM_STAT_REG.FLIT_PERFORM_LTSSM_STAT_TRACK_COUNTER` contains the accumulated latency for all flits that have been tracked.

The controller sets the `FLIT_PERFORM_STAT_REG.FLIT_PERFORM_INTRPT_TRIGGER_EVENT` field to '1' if any single Flit latency value, exceeds the value in `FLIT_PERFORM_CNTRL_REG.FLIT_PERFORM_INTRPT_EXCEED_ENCODED_VAL`.

3.32.2 LTSSM Latency Tracking

This section details the measurement tracking for LTSSM latencies. Level signal is generated for each event from Layer1 to CDM.

- L0 to Recovery due to a Framing Error /software directed while in L0:
  - Time starts when a Framing Error/software directed is occurred in L0
  - Time ends when Layer1 start to transmit OS from Data stream
- L0p – Electrical Idle to start of Data Stream on Lane on an upconfig:
  - Time starts when TxElecidle is de-asserted (based on a flag in xmlh)
  - Time ends when SDS OS is transmitted (based on a flag in xmlh)
- L1.0 to L0:
  - Time starts when LTSSM exits from L1.0
  - Time ends when LTSSM transit to L0
- L1.1 to L0:

- Time starts when LTSSM exits from L1.1
- Time ends when LTSSM transit to L0

- L1.2 to L0:
  - Time starts when LTSSM exits from L1.2
  - Time ends when LTSSM transit to L0

Table 3-101 shows the action taken by the controller when the `FLIT_PERFORM_CNTRL_REG` register is programmed by software. When the software sets the `FLIT_PERFORM_LATENCY_MEASURE_EN` field to '1', the controller sets the `FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1` field in the `FLIT_PERFORM_STAT_1_REG` register '01' and starts latency tracking.

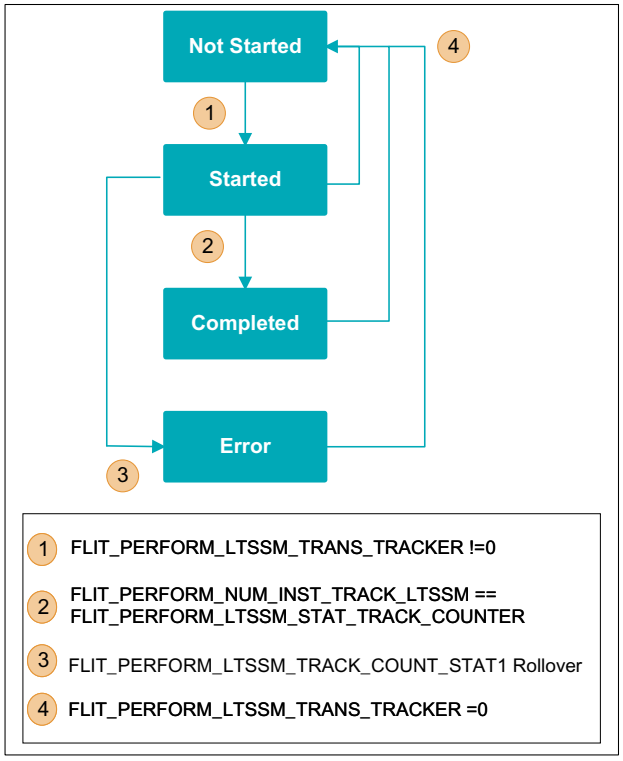
Table 3-101 Flit Latency Tracking When FLIT_PERFORM_STAT_1_REG =1

FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1	FLIT_PERFORM_LTSSM_TRANS_TRACKER	Action
00/10/11	xxxxx	No latency tracking
01	xxxx1	Track L0 to Recovery latency
01	xxx10	Track L0p - Electrical Idle to start of Data Stream on Lane upconfig latency
01	xx100	Track L1.0 to L0 latency
01	x1000	Track L1.1 to L0 latency
10	10000	Track L1.2 to L0 latency

Note Only the LTSSM Performance Measurement Status 1 Register (FLIT_PERFORM_STAT_1_REG) is implemented in the controller, the FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1 field is treated as one-hot with the lowest bit taking priority.

Figure 3-139 shows the hardware update method for `FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1` field.

Figure 3-139 FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1 Update Method



The following sequence describes the Flit latencies tracking method:

1. Software enables tracking by programming the `FLIT_PERFORM_CNTRL_REG` register
2. Layer1 logic, resets the tracker counter and waits for a valid event to begin tracking latency
3. Tracker counter is incremented, with 64us granularity, until the latency event is observed
4. Tracker counter value is reported to the CDM for recording in the `FLIT_PERFORM_STAT_1_REG` register
   a. `FLIT_PERFORM_LTSSM_TRACK_COUNT_STAT1` field is updated with the latency count
   b. `FLIT_PERFORM_LTSSM_TRANS_TRACK_NUM1` field is incremented
   c. `FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1` field is updated (if conditions met)
   d. `FLIT_PERFORM_INTRUPT_EVENT_STAT1` field is updated (if conditions met)
5. If `FLIT_PERFORM_LTSSM_TRANS_TRACK_STAT1 =1`, repeat from step 2.

When the `FLIT_PERFORM_CNTRL_REG.FLIT_PERFORM_NUM_INST_TRACK_LTSSM =0`, the `FLIT_PERFORM_STAT_REG.FLIT_PERFORM_LTSSM_STAT_TRACK_COUNTER`, contains the highest latency value observed since tracking was started.

When the `FLIT_PERFORM_CNTRL_REG.FLIT_PERFORM_NUM_INST_TRACK_LTSSM !=0`, the `FLIT_PERFORM_STAT_REG.FLIT_PERFORM_LTSSM_STAT_TRACK_COUNTER`, contains the accumulated latency for all LTSSM state transitions that have been tracked.

The controller sets the `FLIT_PERFORM_STAT_1_REG.FLIT_PERFORM_INTRUPT_EVENT_STAT1` field to '1' if any single LTSSM transition latency value, exceeds the value in the `FLIT_PERFORM_NUM_INTRPT_BIT_18_17` field or `FLIT_PERFORM_NUM_INTRPT_BIT_16_15_14` field of the `FLIT_PERFORM_CNTRL_REG` register.

Note

LTSSM latency is reported with a granularity of 64us. The controller rounds-up the latency value. The FLIT_PERFORM_LTSSM_STAT_TRACK_COUNTER reports as follows:

- 0x1: 0-64us range
- 0x2: 65-128us range

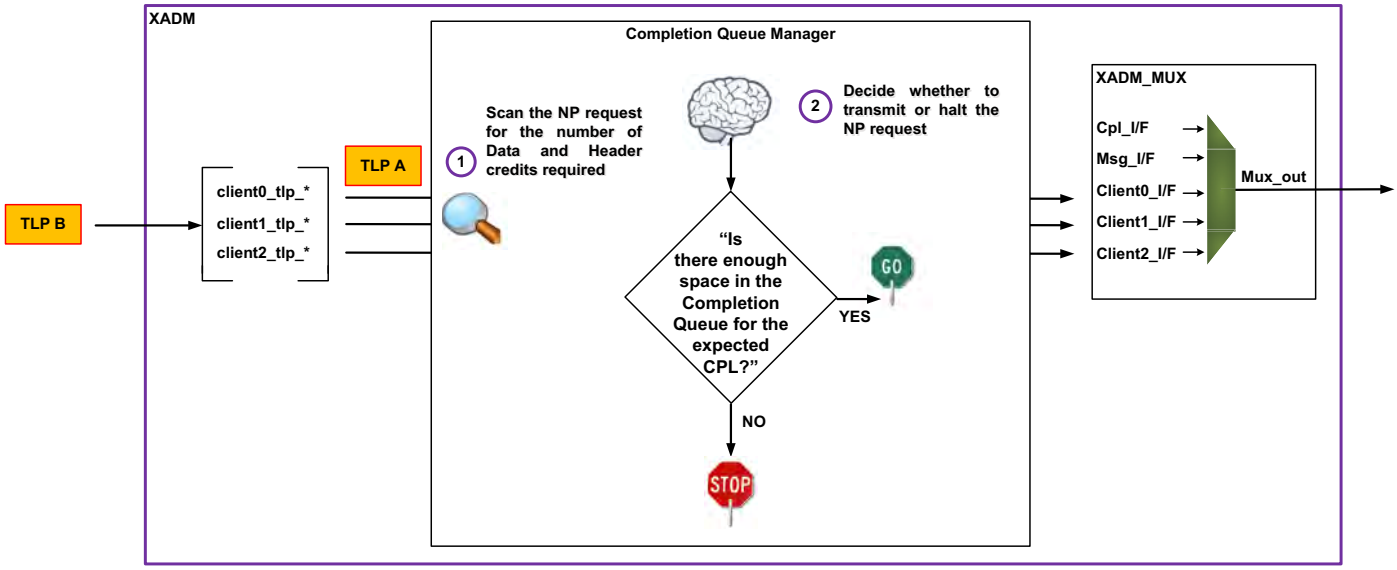
3.33 Completion Queue Management

The controller provides the Completion Queue Management feature to allow operation with completion queues sized appropriately for the expected round trip latency and available bandwidth. Typically this results in smaller completion queues than those which are required when this feature is disabled. This feature works as a high-level flow control mechanism limiting the number of outbound Non-Posted requests when enough space to receive the associated completions is not available.

To enable Completion Queue Management feature set the parameter `CX_CPLQ_MANAGEMENT_ENABLE` to '1'. The feature can be disabled at run-time by setting the `CPLQ_MANAGEMENT_ENABLE` field of `MISC_CONTROL_1_OFF` register to '0'. If the feature is disabled, your application must implement logic to prevent completion queue overflow.

Figure 3-140 describes the high level overview of Completion Queue Management operation.

Figure 3-140 Completion Queue Management Operation



The completion queue manager decides to transmit or halt the NP request transmission on a particular transmit client interface (XALI 0/1/2) based on the number of split completions that can be expected in the worst case, and free slots available in RADM header and data completion queues.

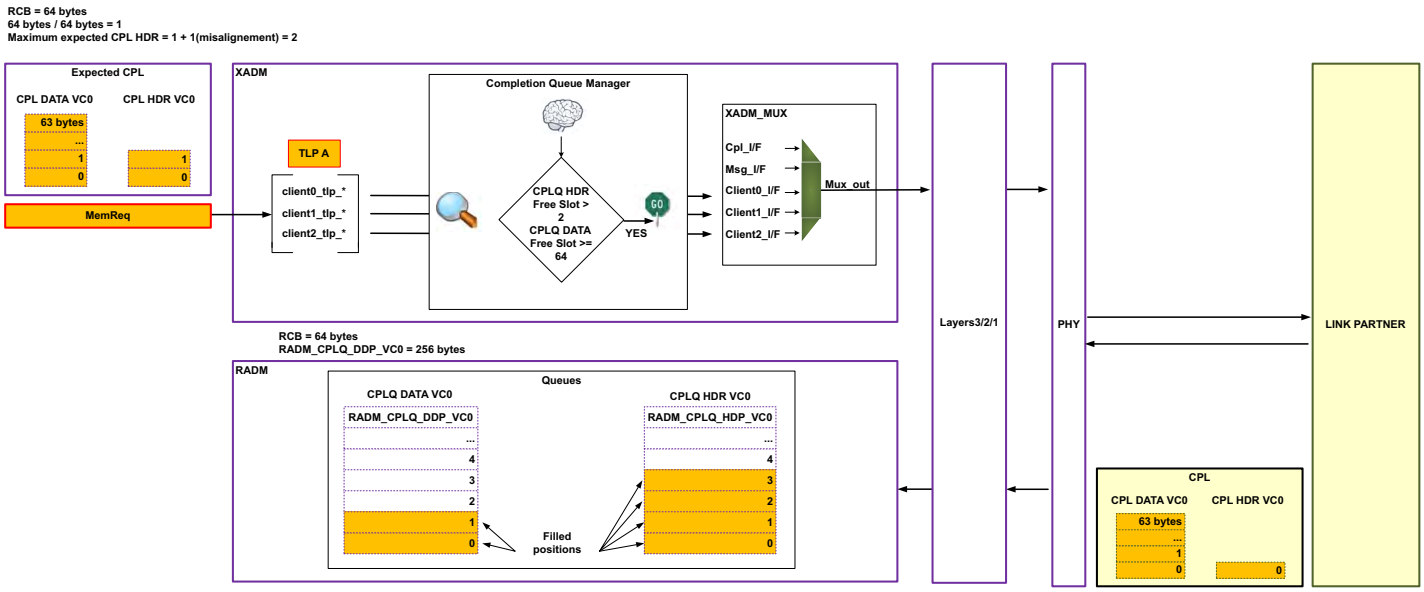
A few example scenarios that depict the completion queue operation are as follows:

Scenario 1: A 64 byte MemRd request at XALI0 complying with RCB rules can be completed two ways:

- One Completion with 64bytes of data
- Two Completions with Completion A data + Completion B data =64bytes(worst case scenario)

Considering the worst case, in this scenario the completion queue manager allows transmission of MemRd request because the completion header queue has 2 free slots and the completion data queue has more than 64 bytes of space available, as described in Figure 3-141.

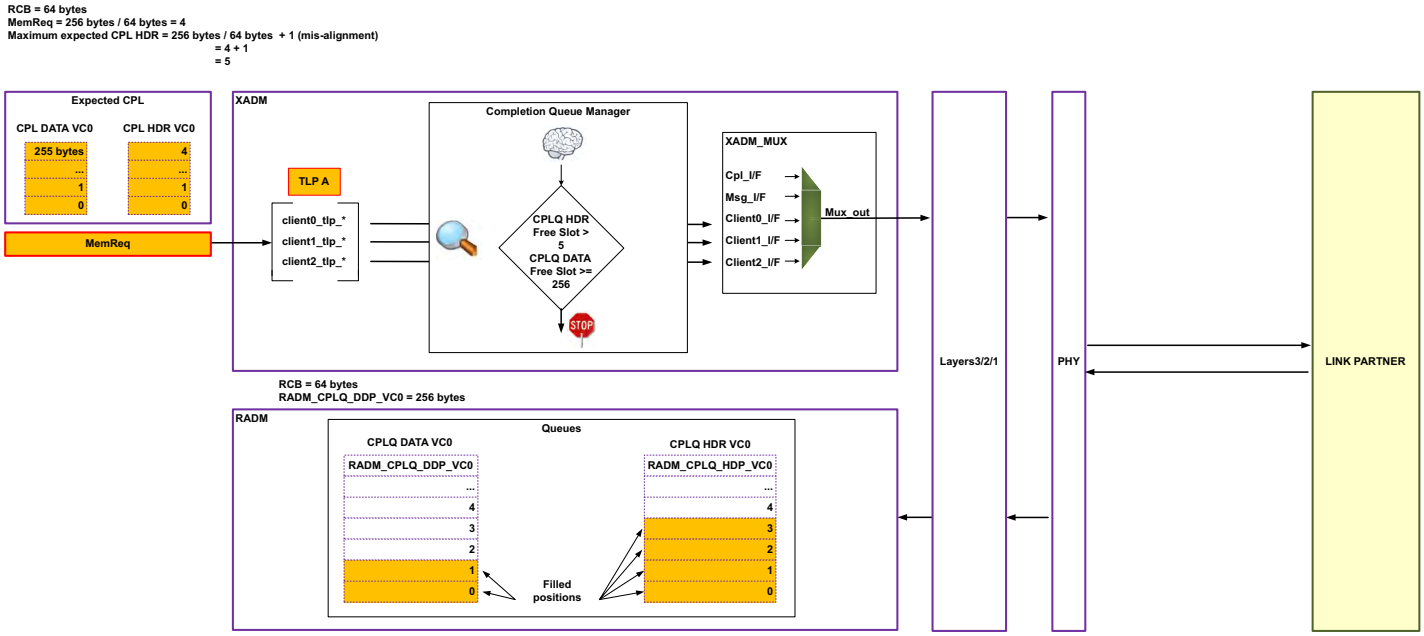
Figure 3-141 Scenario 1: 64 Byte MemRd Request at XALI0



Scenario 2: A 256 bytes MemRd request at XALI0 complying with RCB rules can be completed several ways, the worst case would be five Completions with Completion A data + Completion B data + Completion C data + Completion D data + Completion E data =256 bytes.

In this scenario, the completion queue manager does not allow transmission of MemRd request because even though the completion header queue has 5 free slots, the completion data queue has space available for 248 bytes (62 DW) only, as described in Figure 3-142.

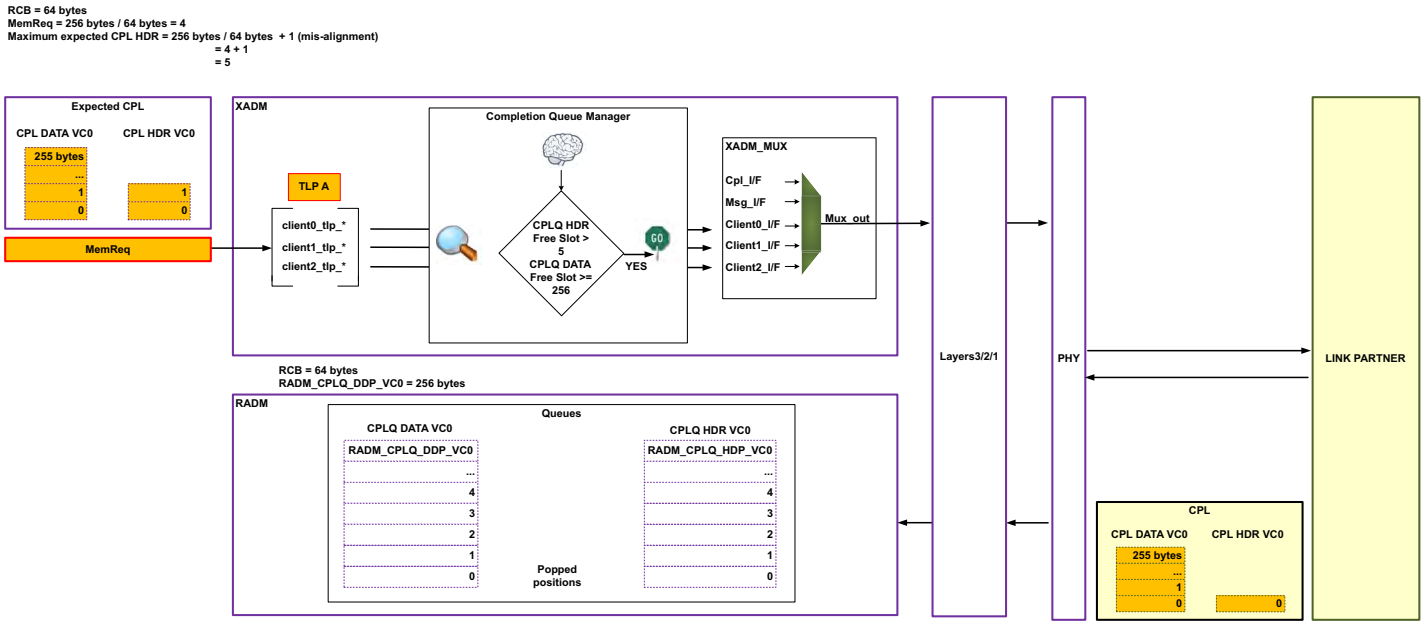
Figure 3-142 Scenario 2: 256 Byte MemRd Request at XALI0



Scenario 3: In scenario 2, let us suppose that after some time the controller pops the header and data completion queues. Now, enough space has been created in header and data completion queues to

accommodate the halted MemRd completions, so the completion queue manager transmits the halted MemRd request, as described in Figure 3-143.

Figure 3-143 Scenario 3: Transmission of Halted 256 Byte MemRd Request at XALI0



Limitations

- This feature is supported only for PCIe devices. It is not supported for PCI-X devices.
- This feature is supported only when ECRC stripping is enabled (`CX_ECRC_STRIP_ENABLE =1`).
- For correct operation of the completion queue management feature the following filter rules must not be changed from the default value of 0.
  - `CX_FLT_MASK_CPL_LEN_MATCH`
  - `CX_FLT_MASK_CPL_TC_MATCH`
  - `CX_FLT_MASK_CPL_TAGERR_MATCH`
  - `CX_FLT_MASK_CPL_ECRC_DISCARD`
  - `CX_FLT_MASK_DABORT_4UCPL`

Completion Timeout

In the event of a Completion Timeout it is recommended that your application perform the following steps:

- Disable completion queue management feature by setting the `CPLQ_MANAGEMENT_ENABLE` field of `MISC_CONTROL_1_OFF` register to 0. This resets the completion queue counters to the initial value.
- Wait until the completion queue is empty and there is no NP request pending in the LUT.
- Enable completion queue management feature by setting the `CPLQ_MANAGEMENT_ENABLE` field of `MISC_CONTROL_1_OFF` register to 1.

3.33.1 Header and Data Completion Queue Size Calculation

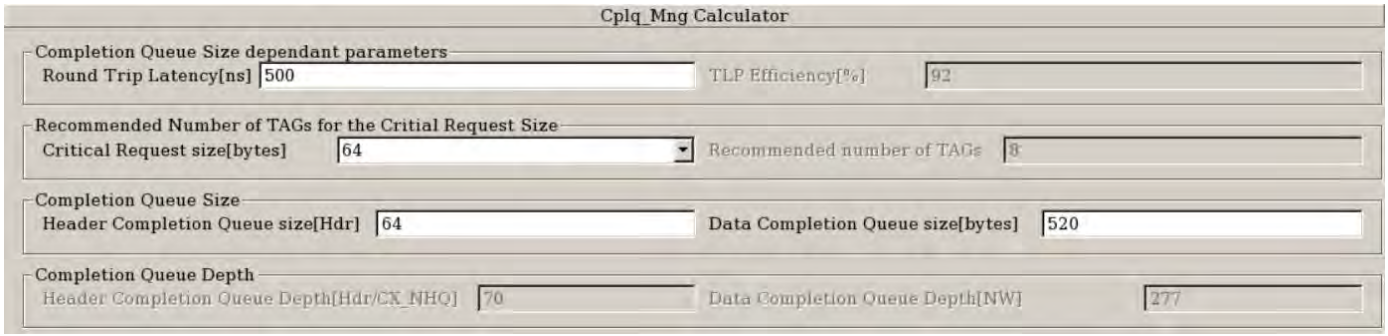
When Completion Queue Management feature is enabled, the controller automatically calculates the completion queue sizes for each VC based on the value of the following parameters entered in coreConsultant:

- *Round Trip Latency [ns]* which should take into account:
  - Local Controller and PHY latencies
  - Remote Controller and PHY latencies
  - Retimers
  - Response time of the remote side
- *Critical Request size [bytes]*, for which full bandwidth usage must be retained (commonly this is 64 bytes due to cache line size).

Using the values of these parameters the controller automatically calculates:

- *Recommended number of TAGs* , to keep the maximum bandwidth of the configuration for the Minimum Request Size.
- Header Completion Queue size [Hdr]. This is the number of entries required in the Completion Header Queue to cover the expected latency given the available bandwidth. The controller considers only two split completions per TAG when calculating this value, other possible split completions scenarios are not taken into account. You can change this value to cover the worst case scenarios as per your requirements.
- Data Completion Queue size [bytes]. This is the size of the Completion Data Queue in bytes to cover the expected latency given then available bandwidth.
- *Header/Data Completion Queue Depth,* needed for the queue control logic. The controller sets the parameters `RADM_CPLQ_HDP_VCn` and `RADM_CPLQ_DDP_VCn` to these values.

Figure 3-144 Completion Queue Size Parameters in coreConsultant GUI



Legacy Method of Header and Data Completion Queue Size Calculation

It is generally mandatory to advertise infinite credits for completion queues according to the *PCI Express Base Specification, Revision 6.0, Version 1.0* to avoid deadlocks or ordering rule violations in the PCIe hierarchy. When completion queue credits are finite the completion queue sizes are determined by the credits. When completion credits are infinite and completion queue management is disabled, the completion queue size per VC is as big as the maximum possible request size times the maximum number

of requests that your application can make. The completion header queue depth (CPLQ_HDP) and completion data queue depth (CPLQ_DDP) are calculated as follows:

```
CPLQ_HDP = (CX_MAX_TAG + 1) * [CX_APP_RD_REQ_SIZE / RCB + 1]CPLQ_DDP = [(CX_MAX_TAG
+ 1) * CX_APP_RD_REQ_SIZE / 4]/CX_NW
```

Where,

- CX_MAX_TAG: Specifies the maximum number of simultaneous outbound PCIe non-posted requests in total for all functions.
- CX_APP_RD_REQ_SIZE: This parameter is used to set the depth of the receive completion data queue (CX_CPLQ_DDP_VC*) when completions are in store-and-forward or cut-through modes, and completion credits are infinite. It is the maximum individual MRd size that your application (or AXI bridge subordinate) makes.
- RCB: Specifies the minimum Read Completion Boundary (RCB) of all CPLs with which the controller inter-operates. For a root port this can be set to 128 bytes, all other components should assume an RCB of 64 bytes unless they exist entirely within closed systems where the minimum RCB is known. This parameter is used only to calculate the required size of the completion header queues when completion credits are infinite and the completion queues are not bypassed.

  RCB (RP) = 128/64 bytes

  RCB (no RP) = 64 bytes
- CX_NW: Specifies the width of the datapath in dwords.

For example, for a configuration with CX_MAX_TAG = 255, CX_APP_RD_REQ_SIZE = 4096 bytes, RCB = 128 bytes, CX_NW = 256 bits, the completion queue header and completion queue data size per VC are calculated as follows:

```
CPLQ_HDP = (255 + 1) * [4096 / 128 + 1] = 8448 (128 bits) = 135168 bytesCPLQ_DDP =
[(255 + 1) * 4096 / 4] / 8 = 32768 (256 bits) = 1048576 bytes
```

There is a relation between the number of tags and the latency of the PCIe link to optimize the design based on the bandwidth expected. The higher the latency the higher the number of TAGs that can be used.

4

Signal Interfaces

For descriptions of each I/O, see "Signal Descriptions" chapter of the *Reference Manual*.

Attention	When you configure the controller in coreConsultant, you can access the I/O descriptions for your actual configuration at *workspace*/report/IO.html using the process described in the "Creating Optional Views and Reports" section in the *User Guide*. This report comes from the exact same source as the *Databook* but removes all the signals that are not in your actual configuration.

RX_TLP is the number of TLPs processed per clock cycle and is also the number of receive queues per TLP type per VC. It is 2 for 256-bit datapath configurations, 4 for 512-bit datapath configurations, and 1 for all other configurations.

This section describes any relevant timing protocols for the following signal interfaces.

- "Transmit Interfaces (XALI0/1/2)" on page 356
- "FC-Transmit Interfaces (FC-XALI)" on page 365
- "CCIX Transmit Interface (XALI_CCIX)" on page 366
- "Receive Bypass Interface (RBYP)" on page 369
- "Receive Request Interface (TRGT1)" on page 370
- "FC-Receive Request Interface (FC-TRGT1)" on page 373
- "CCIX Receive Request Interface (TRGT1_CCIX)" on page 374
- "Data Bus Interface (DBI)" on page 377
- "Vendor Message Interface (VMI)" on page 381
- "System Information Interface (SII)" on page 382
- "PIPE Interface" on page 383
- "PHY Register Bus Interface (PRBI)" on page 389

4.1 Transmit Interfaces (XALI0/1/2)

XALI0 and XALI1 are two similar client interfaces connected to two clients at your application side. Each client can be a requester or a completer. The optional XALI2 interface is present only when `CLIENT2_POPULATED` is defined. The XALI2 interface is identical to XALI0 and XALI1. Requests on XALI0, XALI1, and XALI2 can be asserted in the same clock cycle. The XADM automatically arbitrates requests received from the XALI0, XALI1, and XALI2 interfaces.

Attention In the timing diagrams, the actual clock used to clock the interface might vary in low-power mode. In this case, you should use a different clock than the one indicated in the timing diagram. For more information, see "Synchronous To Attribute" on page 71.

Hint For more information on how to select these client interfaces, see the "Client Interfaces Configuration Guide (without AXI Bridge)" appendix of the *User Guide*.

4.1.1 Transmit Client Interface Protocol Rules

The transmit client provides information on the header bus and payload data on the data bus. The controller does not have internal queues for transmit TLPs. The protocol rules for each of the transmit client interfaces (using the `client0*` signals as the example) are:

- After your application starts a transaction transfer request, it must continue to stream data unless the controller halts the client interface.
- For transactions without an associated payload (for example MRd), the data valid (`client0_tlp_dv`) signals must never be asserted. For more information, see Figure 4-1.
- For transactions with an associated payload (for example MWr), the header valid (`client0_tlp_hv`) and data valid (`client0_tlp_dv`) signals must be asserted in the same cycle for the header phase. However, during the payload phase, `client0_tlp_dv` can be asserted without `client0_tlp_hv` having to be asserted. For more information, see Figure 4-3.
- The end of transaction indication signal (`client0_tlp_eot`) must be asserted during the last cycle of data valid (`client0_tlp_dv`) or of header valid (`client0_tlp_hv`) when no payload is present. You must not assert it at any other time.
- When the `xadm_client0_halt` signal is asserted, your application must maintain the information on the header bus and data bus.
- When idle, the transmit client interface maintains `xadm_client0_halt =1` by default. At the end of a packet when client arbitration for the next packet is underway `xadm_client0_halt` may toggle. The behavior of the `xadm_client0_halt` signal is dependent on the datapath width of the controller.
- 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, where suitable for your application.
- To send a message through the transmit client interface, your application asserts the message code on `client0_tlp_byte_en`, and the last two DWORDs of the header through `client0_tlp_addr[63:0]`, where `client0_tlp_addr[63:32]` is the third DWORD (bytes 8-11) of the header. When there is data to send with the message, your application sends the data through

`client0_tlp_data`, as with other TLP data. For more information, see "Message Generation" on page 194.

Payload and Read Request Length rules

- The controller does not check for TLP errors; instead it sends the TLP as presented on the XALI interface.
- The native PCIe controller does not check that the TLP payload size is less than the Maximum Payload Size limit.
- The AXI bridge module (if present) guarantees that the TLP payload size is less than the Maximum Payload Size limit.
- The native PCIe controller does not check that the TLP payload size is less than `CX_MAX_MTU` limit. Exceeding this limit overflows the retry buffer, resulting in data corruption.
- When address alignment is enabled (`GLOB_ADDR_ALIGN_EN =1`), your application logic must make sure that the length of the TLP is less than the maximum payload supported. For more information, see "Transmit Address Alignment" on page 361.

Hint You can use the "Target Completion Lookup Table" on page 363 to simplify the generation of outbound completions. This feature reduces the amount of information that your application has to process/store about an incoming request on `TRGT1`.

4.1.2 Transmit Client Transactions

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

- Figure 4-1 depicts three memory read (MRd) request TLPs. The client0_tlp_eot and client0_tlp_hv signals remain asserted for two back-to-back MRd TLPs. The TLP header inputs remain valid until the controller de-asserts xadm_client0_halt. The controller asserts the xadm_client0_halt signal when it is not ready to process any transactions.
- Figure 4-2 shows a memory write (MWr) request TLP. The TLP header and payload inputs remain unchanged when xadm_client0_halt is asserted.
- Figure 4-3 shows back-to-back MRd and MWr transactions. The client0_tlp_eot and client0_tlp_hv signals remain asserted across multiple TLPs.
- Figure 4-4 shows a completion TLP requested by client0.

Figure 4-1 Client0 Transaction: MRd Request TLPs

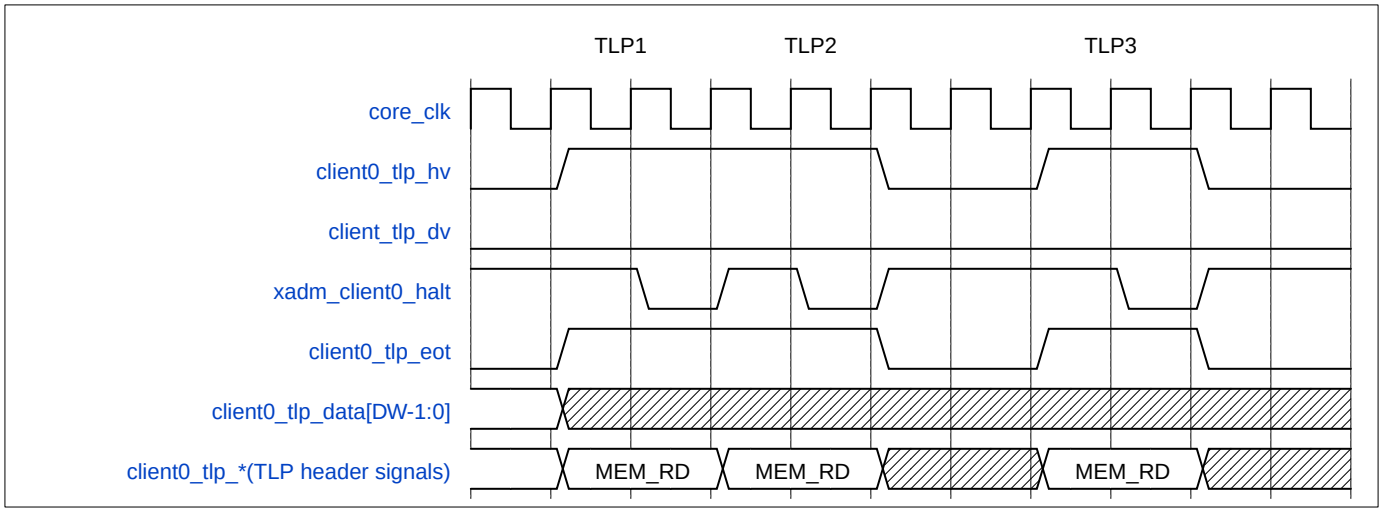


Figure 4-2 Client0 Transaction: MWr Request TLPs

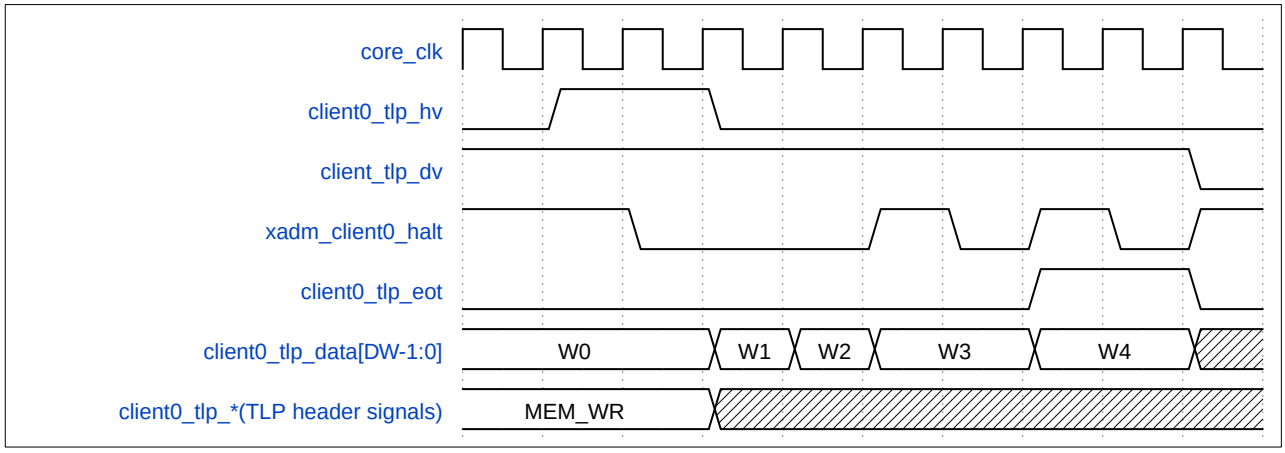


Figure 4-3 Client0 Transaction: Back-to-Back MRd and MWr TLPs

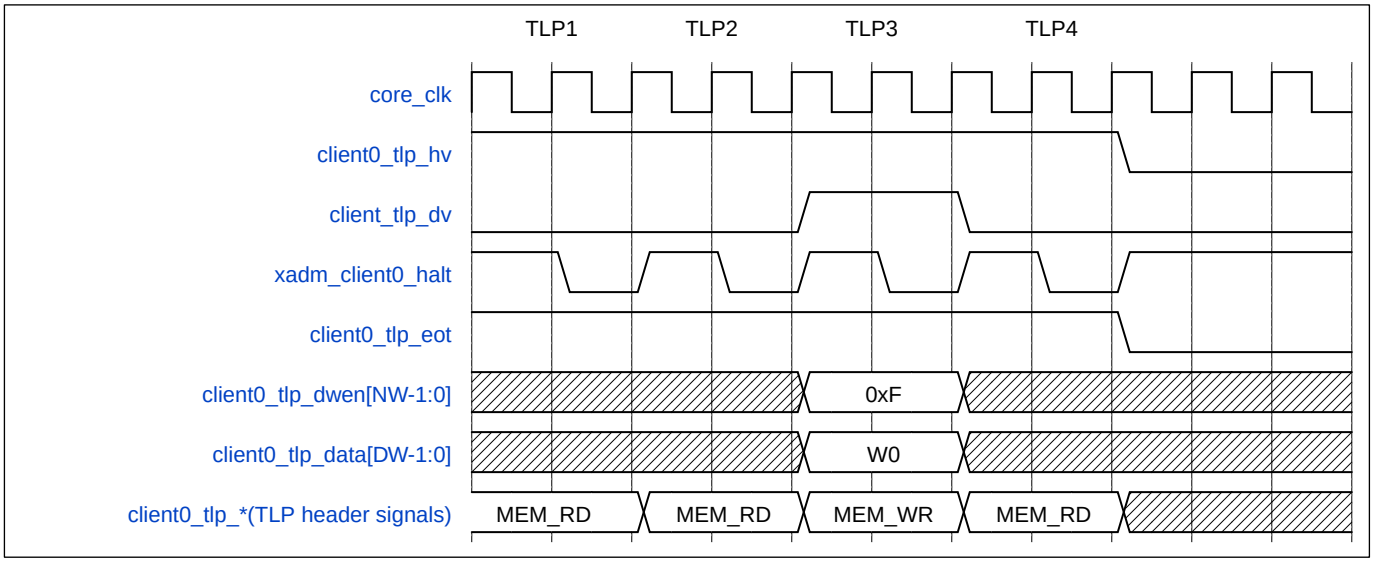
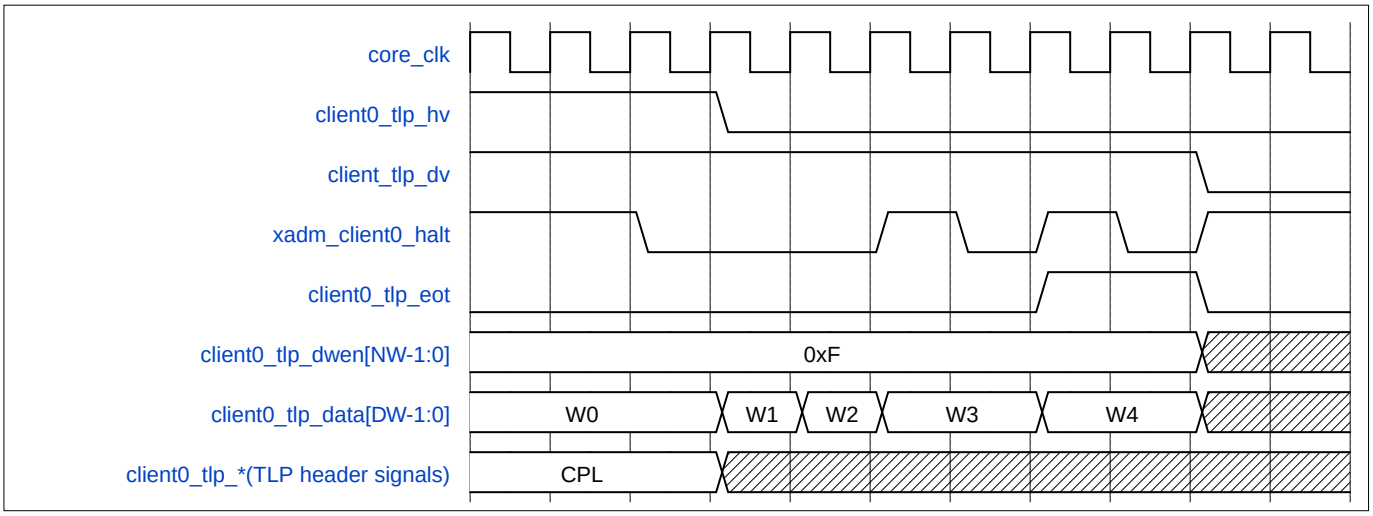


Figure 4-4 Client0 Transaction: CPL TLP



The following diagrams illustrate example for MultiStream Transmission (`CX_MULTISTREAM =1`):

- Figure 4-5 depicts three memory read (MRd) request TLPs for VC0. The client1_tlp_eot[0] and client1_tlp_hv[0] signals remain asserted for three back-to-back MRd TLPs. When the xadm_client1_halt[0] signal is asserted the data is not sampled.
- Figure 4-6 shows a memory write (MWr) request TLP. The TLP header and payload inputs remain unchanged when client2_tlp_halt[0] is set. This behavior is correct, as W1 and W3 are sampled only once each.
- Figure 4-7 shows a completion TLP requested by VC0.
- Figure 4-8 shows simultaneous MRd and MWr.

Figure 4-5 VC0 Transaction: Client1 MRd Request TLPs

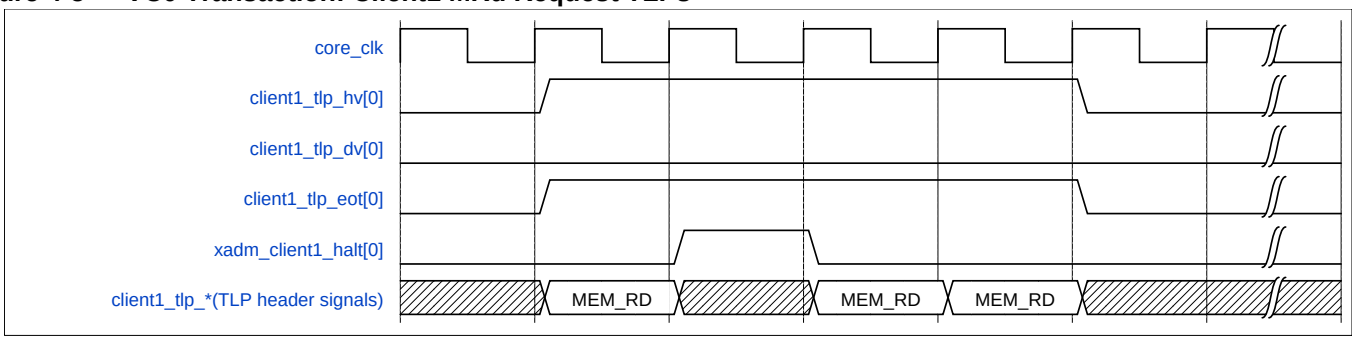


Figure 4-6 VC0 Transaction: Client2 MWr Request TLPs

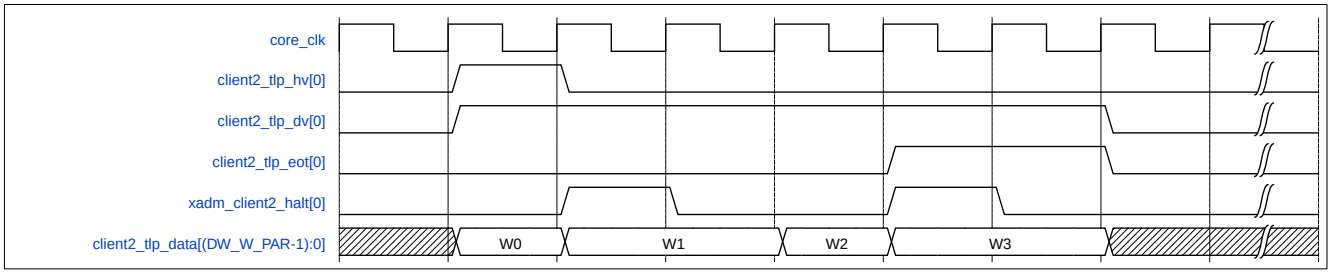


Figure 4-7 VC0 Transaction: Client0 Completion

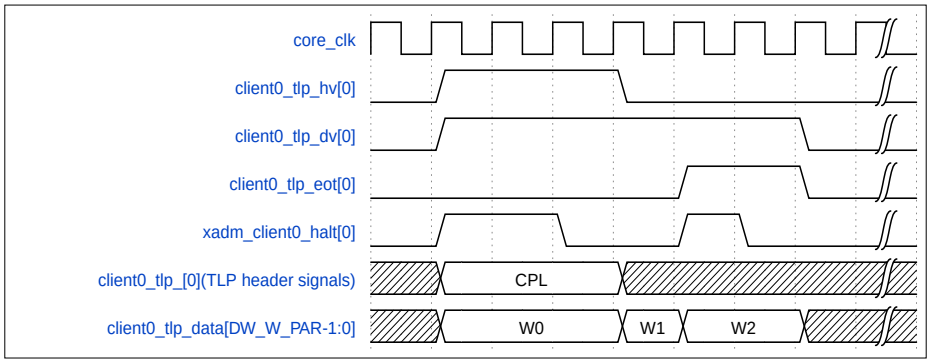
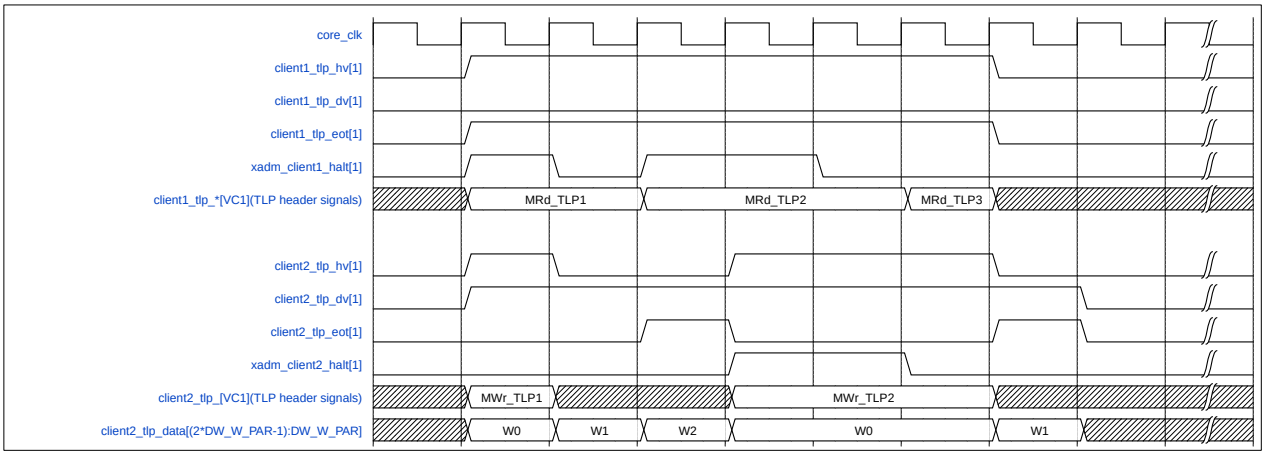


Figure 4-8 VC1 Transaction: Back-to-Back Client1 MRd and Client2 MWr TLPs



4.1.3 Transmit Address Alignment

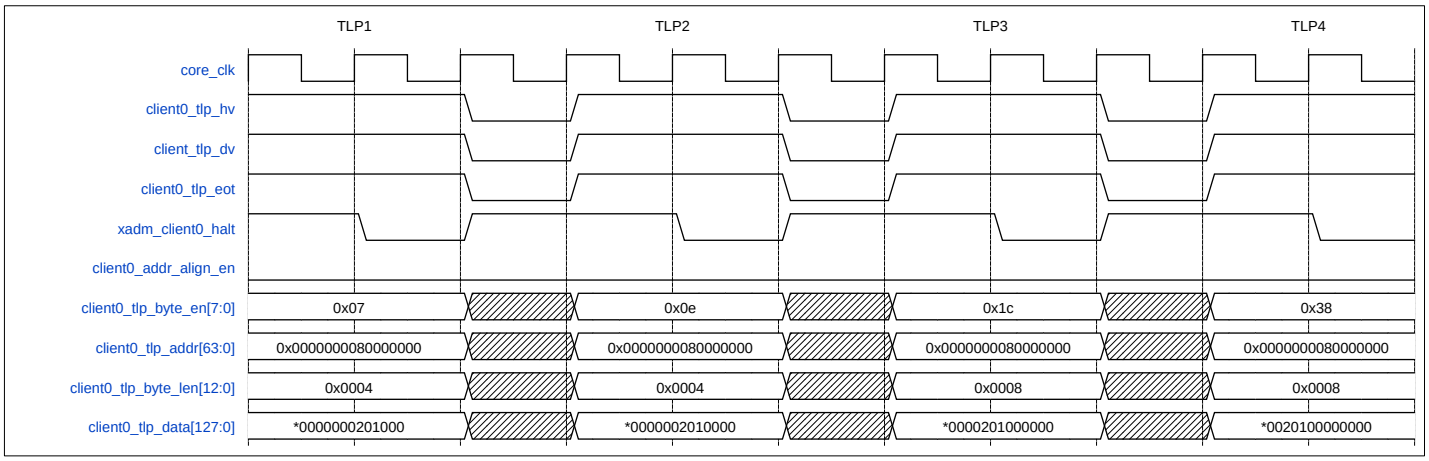
For outbound memory and I/O requests (but not for completions), the controller provides an address alignment feature using the `clienti_addr_align_en` signal so that your application can have per-TLP control of how the address is aligned on the transmit client interface. To enable this feature, set `GLOB_ADDR_ALIGN_EN`. When address alignment is off[1] for a packet (`clienti_addr_align_en =0`), your application is responsible for setting the appropriate first and last byte enable (FBE, LBE) values and presenting the payload in the correct format. The controller transmits it without realignment (if required). When address alignment is on for a packet (`clienti_addr_align_en =1`), the controller uses the lower two bits of the `clienti_tlp_addr` bus to determine the appropriate byte enable values and the alignment of any data included for the transmitted TLP.

4.1.3.1 Address Alignment Off

When address alignment is off, the transmit client interface essentially becomes a DWORD interface for that packet. In this case, your 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`: Bits [1:0] must be 00, that is, the address must be DWORD-aligned.
- `client0_tlp_byte_en`: The byte enables are placed directly into the transmitted TLP. Bits [3:0] form the FBE; bits [7:4] form the LBE. Non-contiguous byte enables can be transmitted in this way.
- `client0_tlp_byte_len`: Bits [1:0] must be set to 00. The byte length value must be rounded up to the next DWORD boundary. For example, when the request is five bytes long, the value of bits [12:0] must be eight.
- `client0_tlp_data`: The controller transmits the data just as it is presented on `client0_tlp_data`. Your application is responsible for shifting the data up according to the byte address.

Figure 4-9 Client0 Transaction: Address Alignment Off



For read requests, the `client0_tlp_data` bus has no requirements. Figure 4-9 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 4-9), the byte length goes directly from four to eight.

1. Should always be off for completions.

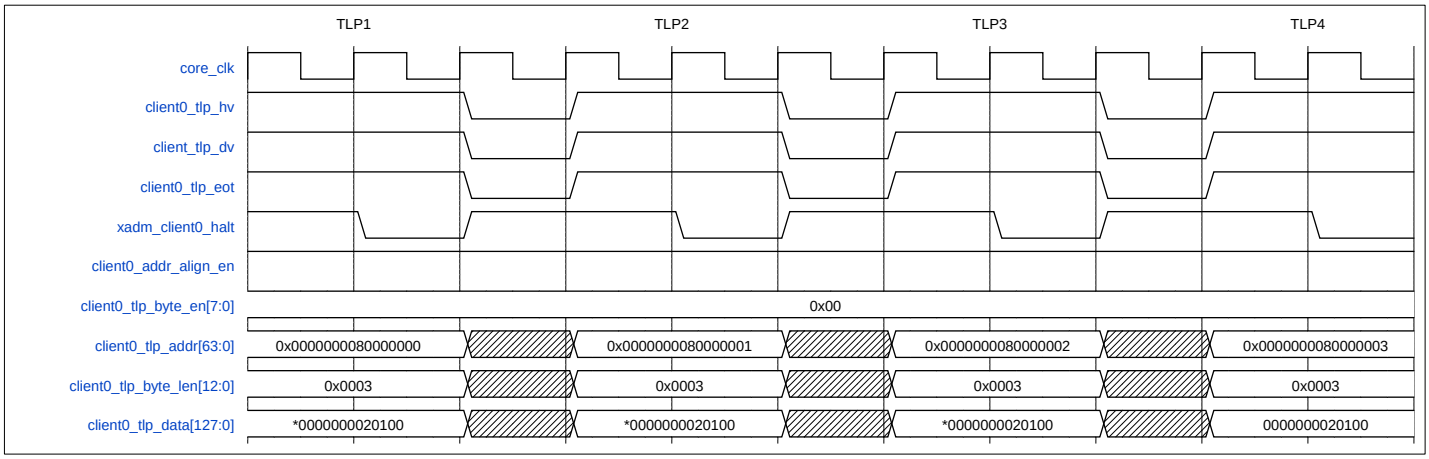
Note You must not use address alignment for completions and must set `clienti_addr_align_en =0`. Your application must drive the correct byte address on `clienti_tlp_addr` so that the completion lower address bits [6:0] are derived from the DWORD-aligned address and FBE of the original request.

4.1.3.2 Address Alignment On

When address alignment is on for a packet, the client interface becomes more of a byte interface. The formatting information that your 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 0x0, that is, not used for a packet with address alignment on.
- `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 controller up-shifts the data in the transmitted TLP based on the lower two bits of the byte address.

Figure 4-10 Client0 Transaction: Address Alignment On



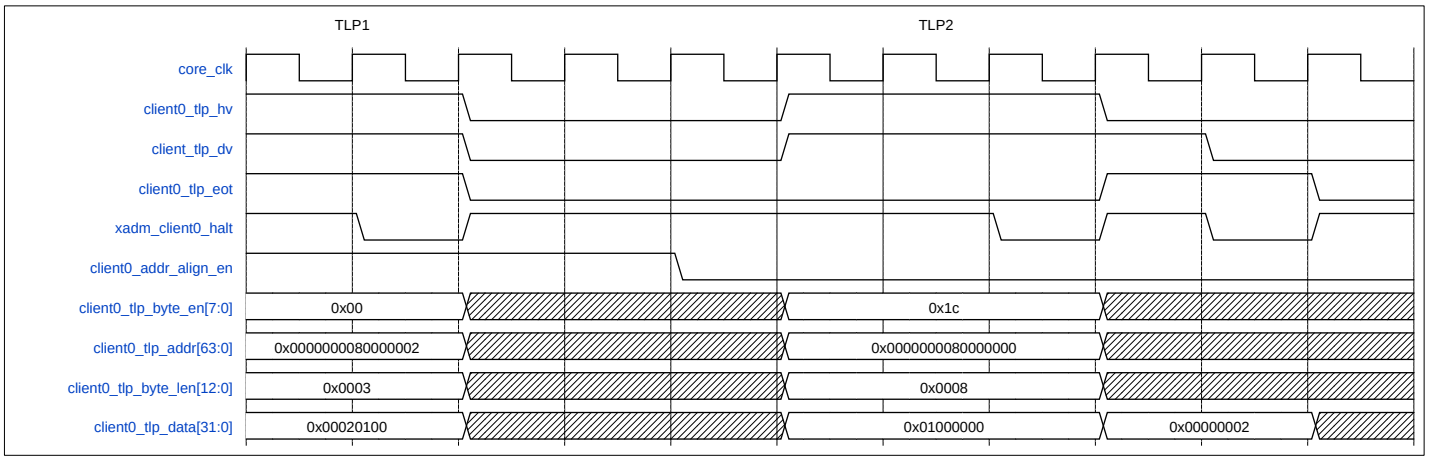
Based on the address and byte length, the controller determines the FBE and LBE fields of the TLP. For read requests, the `client0_tlp_data` signal has no requirements. For write requests, the controller determines how many DWORDs of data payload is transmitted in the associated TLP. Figure 4-10 shows the same sequence of memory write requests as Figure 4-9, except that address alignment is on. The data vector does not change based on the byte address. The controller never generates a TLP with non-contiguous LBE/FBE.

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, when maximum payload is set to 128 bytes (32 DWORDs) and your application sends a request of 128 bytes but with a non-DWORD-aligned address the controller sends the request with a DWORD-aligned address and with a length of 33 DWORDs. This is detected as a malformed TLP at the destination.

4.1.3.3 Address Alignment and Data Bus Cycles

For a write request, the number of clock cycles on your application data bus can be greater when address alignment is off. This occurs when the data payload of the write request crosses the boundary of the data vector size. Figure 4-11 shows two identical memory write requests on a 32-bit transmit data bus. The byte address is 0x80000002 and the byte length is three. The first request has address alignment on and requires only one cycle on the transmit client interface. The second request has address alignment off 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 controller (first request) or by your application (second request).

Figure 4-11 Client0 Transaction: Address Alignment and Data Bus Cycles



4.1.3.4 Address Alignment and ECRC

When your application transmits requests across the client interface with ECRC included, the `clienti_addr_align_en` signal must be zero for those TLPs that contain ECRC (TD =1). This is because there is no way for the controller to know the value of the bytes below the first enabled bytes. When the ECRC is computed over these unknown bytes, the validity of the ECRC in the transmitted TLP cannot be guaranteed when `clienti_addr_align_en` is high for that TLP.

4.1.4 Target Completion Lookup Table

When you set the `TRGT_CPL_LUT_EN` configuration option, the target completion lookup table (LUT) is connected internally between the TRGT1 and XALI0/1/2 interfaces. This feature provides two functions:

- The application does not need to provide certain TLP header information when transmitting completions on XALI0/1/2 because the controller stores and tracks the information in an internal LUT. The LUT uses a table to ease the generation of completions for inbound requests by your application logic. The attributes of all outstanding inbound non-posted requests are stored in the table indexed by a lookup ID. When your application wishes to send a completion, it supplies the lookup ID and the LUT fills in several fields in the completion header.

  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 `client0/1/2_cpl_lookup_id` when transmitting the completion. When using this feature, the application does not need to supply the following header information for completions:

- client0/1/2_remote_req_id
- client0/1/2_cpl_byte_cnt
- client0/1/2_tlp_attr
- client0/1/2_tlp_addr
- client0/1/2_tlp_tid
- client0/1/2_tlp_tc

- When your application logic has not generated a completion for an incoming request within the required time interval, the LUT indicates that a timeout has occurred. Information about the timed-out completion is available on the `trgt_timeout_*` outputs.

The number of entries in the LUT is set by the "Max Forwarded NP Requests" (`CX_REMOTE_MAX_TAG`) configuration parameter. When you have used all the entries in the LUT, the RADM automatically halts the TRGT1 interface.

Note

- In the AXI bridge configuration, the target completion lookup table is always present. Only trgt_-lookup_id output signal is available, client0/1/2_cpl_lookup_id are not available as they become internal signals in the AXI bridge configuration.
- The target completion LUT (and target completion timeout event) should not be confused with the receive completion LUT (and completion timeout event). The target completion LUT is watching for received **application** completions (on XALI0/1/2) corresponding to previously **received** non-posted requests from the PCIe wire. The receive completion LUT is watching for received **PCIe** completions on the wire, corresponding to previously **transmitted** non-posted requests.

In a Gen6 Multistream Architecture configured controller with `CX_FLIT_MODE_ENABLE =1`, the Target Completion Lookup Table also stores the Requester Segment from the received NP request. This is populated in the outgoing completion as the destination segment if this has been deemed to be valid.

4.2 FC-Transmit Interfaces (FC-XALI)

For a Gen6 Multistream Architecture controller, the client interface is specific to an FC (Posted, Non-Posted, or Completion) and VC (Virtual Channel (0-7)). For x16 lanes, the controller can scale the XALI interface to 1024-bit.

The controller can be configured for one or two sets of FC-XALI interfaces for each VC. There must be a minimum of three XALI Interfaces, one for each FC type. Maximum number of FC-XALI interfaces is 24 (3 * FC + 8 * VC).

All the ports in the "Transmit Interfaces (XALI0/1/2)" on page 356 are present in the FC-XALI interfaces.

Table 4-1 FC-XALI Interfaces for Single VC

FC Type	One Set of FC-XALI Interfaces	Two Sets of FC-XALI Interfaces
Posted	client_vc0_p0_<xali_name_suffix>	client_vc0_p0_<xali_name_suffix> client_vc0_p1_<xali_name_suffix>
Non-Posted	client_vc0_np0_<xali_name_suffix>	client_vc0_np0_<xali_name_suffix> client_vc0_np1_<xali_name_suffix>
Completion	client_vc0_cpl0_<xali_name_suffix>	client_vc0_cpl0_<xali_name_suffix> client_vc0_cpl1_<xali_name_suffix>

In Flit Mode (`CX_FLIT_MODE_ENABLE =1`), the controller has additional input signals (`client_vc[0-7]_[p|np|cpl][]0-1]_client_tlp_segment` and `client_vc[0-7]_[p|np|cpl][]0-1]_client_tlp_dst_segment`) on each FC-XALI interface to provide Flit Mode destination, requester, or completer segment information for a given channel.

The TLP header encoding and position is different for Flit Mode and Non-Flit Mode format TLPs. This difference in TLP format is not reflected in the FC-XALI interfaces. For both Flit Mode and Non-Flit Mode, the TLP information is specified in the same way on the XALI interface.

For example, PASID in a Flit Mode TLP is positioned in OHC-A1. In Non-Flit Mode, PASID is positioned in E2E prefix. In both Flit Mode and Non-Flit Mode, the PASID on XALI interface is specified on the `client_vc[0-7]_[p|np|cpl][0-1]_tlp_prfx` signal.

For more information on Address alignment, see "Address Alignment and ECRC" on page 363.

4.2.1 Target Completion Lookup Table

In Gen6 configured controller with Flit Mode (`CX_FLIT_MODE_ENABLE =1`), the target completion LUT records the requester Segment ID from the Non Posted request on the TRGT interface. This is used to populate the destination Segment ID in the outbound Completion TLP. The Non-Posted Requester Segment is now also stored and as a part of a completion is populated as the Destination Segment if valid.

4.3 CCIX Transmit Interface (XALI_CCIX)

The CCIX specification relies on the use of a dedicated high priority Virtual Channel (VC) to transfer data between devices. A dedicated XALI_CCIX interface for transmitting CCIX TLPs provides the following advantages:

- Minimize latency of CCIX traffic on the transmit path
- Provide CCIX support without affecting the standard PCIe functionality
- Minimize collision between standard PCIe traffic and CCIX traffic

XALI_CCIX interface implements a simple handshake mechanism between the controller and your application.

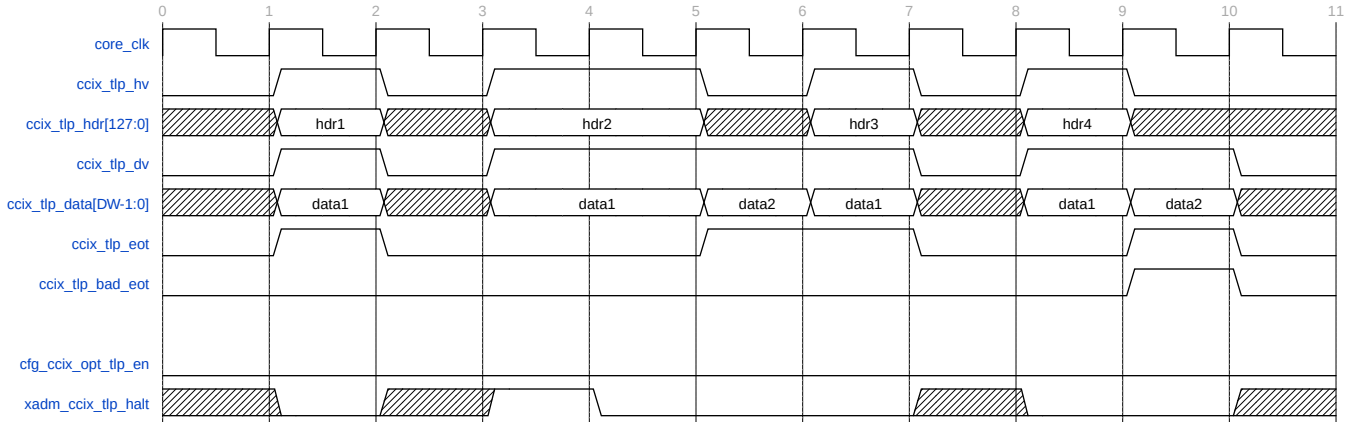
4.3.1 CCIX Transmit Client Transactions

The following diagrams illustrate example transactions on the CCIX transmit client interface:

- Figure 4-12 depicts an example of generic operation of XALI_CCIX interface when CCIX TLPs in PCIe compatible TLP format are transmitted on XALI_CCIX interface.
- Figure 4-13 depicts an example of generic operation of XALI_CCIX interface when CCIX TLPs in optimized CCIX TLP format are transmitted on XALI_CCIX interface.
- Figure 4-14 depicts an example scenario for a 128-bit controller when the amount of data transmitted in one clock cycle is not enough to fill the complete ccix_tlp_data bus.

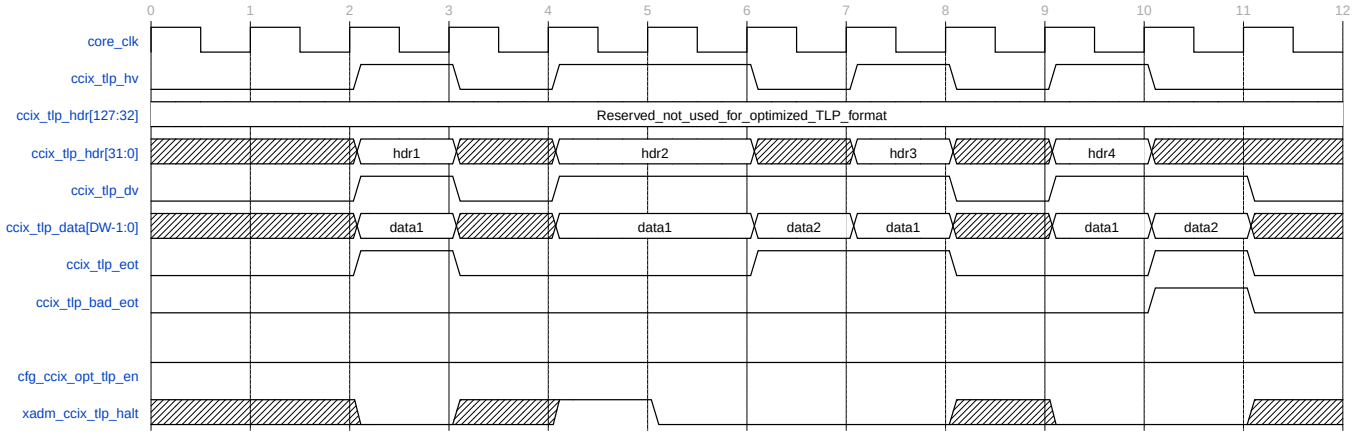
In the timing diagrams, hdr*i* (where *i* is 1,2,3 and so on) represents the header for different TLPs, `data1` represents the first data cycle, `data2` represents the second data cycle, and so on.

Figure 4-12 XALI_CCIX PCIe Compatible TLP Format



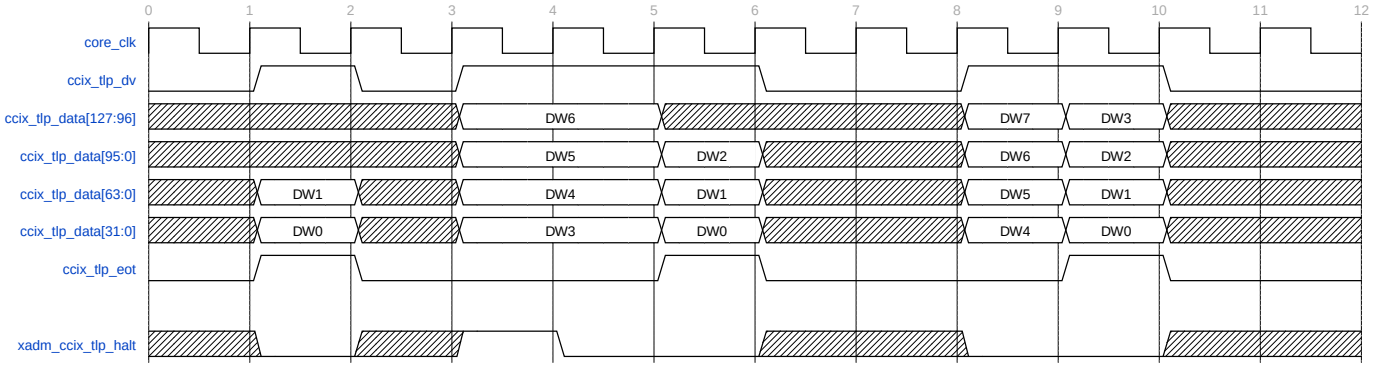
For PCIe compatible TLP format the complete header bus is used (`ccix_tlp_hdr[127:0]`) to transmit the header information.

Figure 4-13 XALI_CCIX Optimized CCIX TLP Format



For optimized CCIX TLP format only 32 bits of the header bus are used (`ccix_tlp_hdr[31:0]`) to transmit the header information. The remaining header bits are reserved.

Figure 4-14 XALI_CCIX Data Bus Usage Example for 128-bit Datapath



Clock cycle one and clock cycle five in Figure 4-14 illustrate how the `ccix_tlp_data` bus is expected to be used when the amount of data to transmit in one clock cycle is not enough to fill the complete `ccix_tlp_data` bus. For both PCIe compatible TLP format and optimized CCIX TLP format the same behavior applies.

4.3.2 CCIX TLP Header Fields and ccix_tlp_hdr Mapping

Table 4-2 illustrates the mapping between CCIX TLP header fields and the bits in the transmit CCIX interface header ccix_tlp_hdr.

Table 4-2 Mapping Between CCIX TLP Header Fields and ccix_tlp_hdr

CCIX TLP Field		
PCIe Compatible TLP Format	**Optimized CCIX TLP Format**	**ccix_tlp_hdr bits**
TLP DW3	NOT USED	96 - 127
Vendor ID		95 - 80
Device Number		79 - 75
Function Number		74 - 72
Bus Number		71 - 64
Message Code		63 - 56
TAG		55 - 48
Requester ID		47 - 32
Length[7:0]	CCIX DW0 Content	31
	Length[6:0]	30 - 24
TD	CCIX DW0 Content	23
EP		22
Attr		21 -20
AT		19 -18
Length[9]		17
Length[8]		16
R		15
TC	TC	14 - 12
R	CCIX DW0 Content	11
Attr[2]		10
R		9
TH		8
FMT	0	7
	Type	6
	CCIX DW0 Content	5
Type		4 - 0

4.4 Receive Bypass Interface (RBYP)

The RBYP interface is used for any queue that you have configured in bypass mode. For more information, see "Receive Queues" on page 100.

Attention When you configure the controller with an AXI bridge, then this signal interface is no longer visible as it now becomes an internal connection between the native PCIe controller and the AXI bridge.

4.4.1 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 cycles 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` can both be asserted in the same cycle.
- `radm_bypass_tlp_abort` and `radm_bypass_dllp_abort` are asserted only when `radm_bypass_eot` is asserted, and when the queue mode is not store-and-forward.

4.4.2 Receive Bypass Interface for Gen6 Configuration

For Gen6 configured controller, Receive Bypass Interface is replaced with a dedicated Receive Request Interface (TRGT1) for Completion TLPs. For more information, see "FC-Receive Request Interface (FC-TRGT1)" on page 373.

4.5 Receive Request Interface (TRGT1)

The controller uses TRGT1 to pass incoming TLPs to your application after they have been filtered. For more information, see "Receive Queues" on page 100.

Attention

In the timing diagrams, the actual clock used to clock the interface might vary in low-power mode. In this case, you should use a different clock than the one indicated in the timing diagram. For more information, see "Synchronous To Attribute" on page 71.

When you configure the controller with an AXI bridge then this signal interface is no longer visible as it now becomes an internal connection between the native PCIe controller and the AXI bridge.

4.5.1 TRGT1 Protocol Rules

- All header information is valid in the cycle when `radm_trgt1_hv` is asserted.
- In cut-through queue mode, the controller might intermittently de-assert the `radm_trgt1_dv` output. In store-and-forward queue mode, the controller de-asserts `radm_trgt1_dv` either when the TLP data transfer is complete or if `trgt1_radm_halt` is asserted (after certain clock cycles depending on the pipelines).
- The controller uses the `radm_cpl_trgt1_abort` and `radm_trgt1_dllp_abort` outputs in bypass and cut-through modes only. For more information, see Advanced Information: Advanced Error Handling for Received TLPs.

4.5.2 TRGT1 Protocol Transactions

- Figure 4-15 shows an example TRGT1 transaction in cut-through queue mode where the controller might intermittently de-assert the radm_trgt1_dv output. Your application should be able to absorb the gaps.
- Figure 4-16 shows an example when a memory write (MWr) TLP from upstream is received at the TRGT1 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 4-17 shows an example TRGT1 transaction in which your application inserts wait states by asserting trgt1_radm_halt. The controller 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 TRGT1 outputs at their current values until your application de-asserts trgt1_radm_halt. After sampling trgt1_radm_halt de-asserted, the controller continues the 5-DWORD transaction to completion.

Figure 4-15 TRGT1 TLP with Gaps (Cut-Through Mode Only)

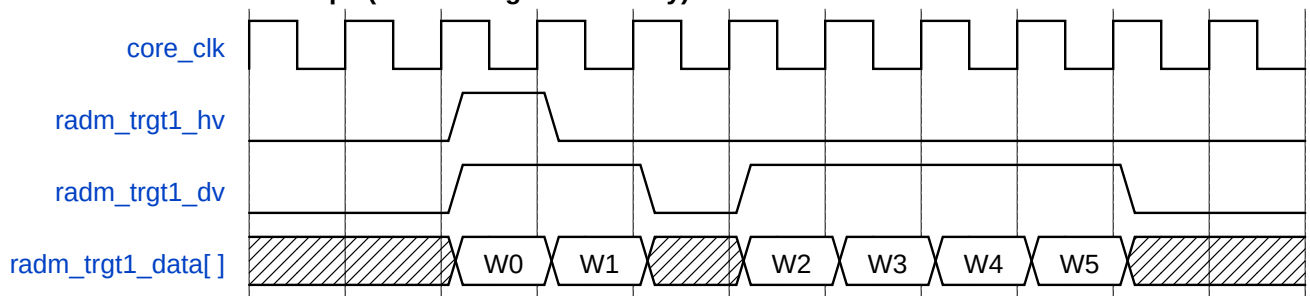


Figure 4-16 TRGT1 Transaction: MWr Request TLP

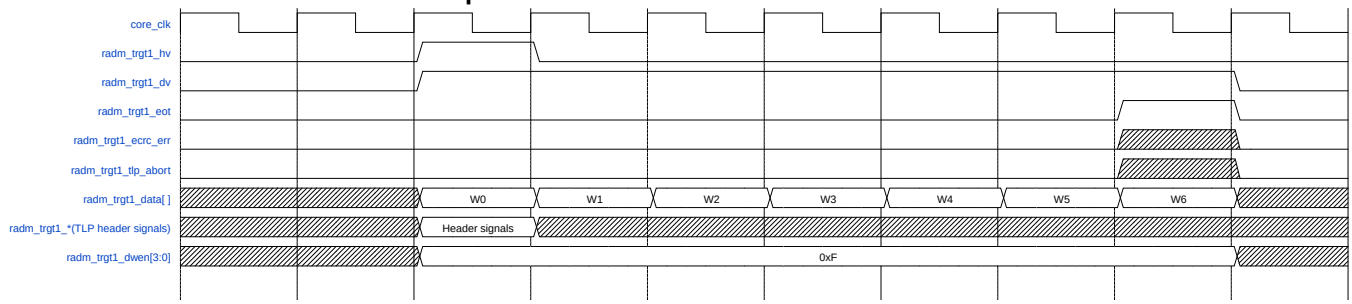


Figure 4-17 TRGT1 Transaction (Scenario 1): MWr Request TLP with Wait States

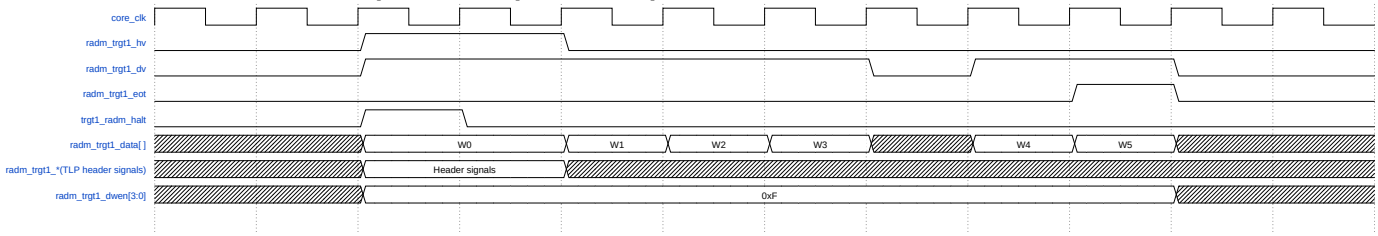
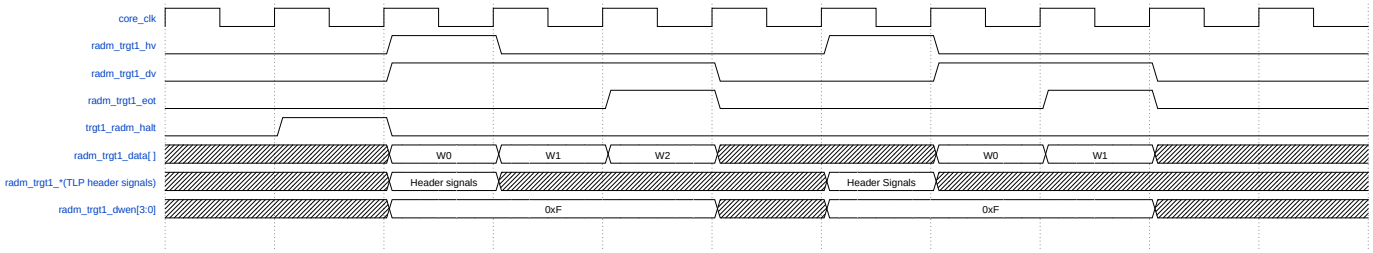


Figure 4-18 TRGT1 Transaction (Scenario 2): MWr Request TLP with Wait States



Note Figure 4-17 and Figure 4-18 are examples to show radm_trgt1_dv is de-asserted when trgt1_radm_halt is asserted only after a certain number of clock cycles depending on the pipeline delays.

4.5.3 TRGT1 Packet Grant and Halt

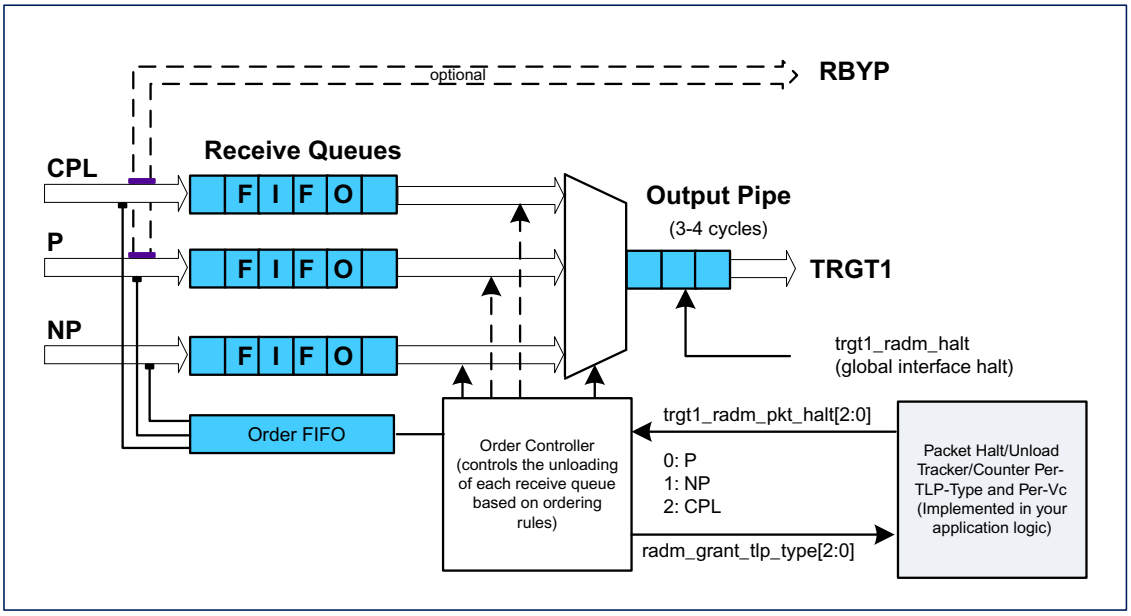
Note For AXI, you do not need to consider these signals because the bridge uses them internally.

In Figure 4-19, note that the ordering controller operates between the receive queues and the output pipe. If a packet enters the output pipe and becomes blocked by the global TRGT1 halt signal `trgt1_radm_halt` or by a too-late assertion of `trgt1_radm_pkt_halt[2:0]`, it blocks the delivery of all other packets and lead to a potential deadlock scenario when a non-posted TLP blocks a posted TLP.

To avoid this you must design a “Non-Posted TLP Tracker” and use the packet halt inputs (`trgt1_radm_pkt_halt[2:0]`) to notify the controller in advance that your application cannot accept any more of a particular packet type. This causes the order controller (in Figure 4-19) to block that particular packet type in the receive queue and not unload it into the output pipe. You must implement a counter/tracker in your application logic for each TLP type using the `radm_grant_tlp_type[2:0]`

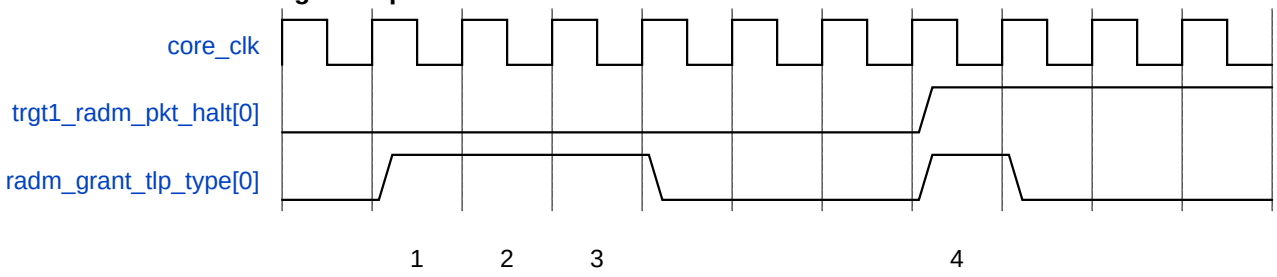
outputs. For example, design a counter that increments on `radm_grant_tlp_type[2:0]`. When `radm_trgt1_hv` of the correct type of transaction is asserted, decrement the counter. When this counter reaches the number of free TLP locations in your applications buffer, assert `trgt1_radm_pkt_halt`.

Figure 4-19 Receive Queue and Output Queue Halting (Single-VC/ non-AXI non-DMA Design)



The `radm_grant_tlp_type[2:0]` signals are pulse outputs indicating that a transaction has been granted to dequeue from the receive queue. The level input signal `trgt1_radm_pkt_halt[2:0]` acts like a flow control On/Off. It alone does not control the exact number of transactions that the controller delivers, because of the pipelines inside the output pipe. The signal `radm_grant_tlp_type[2:0]` is a clocked output. Therefore you must combinatorially assert the signal`trgt1_radm_pkt_halt[2:0]` to stop extra transactions from being dequeued from the receive-queue. There is a minimum delay of one clock cycle from the de-assertion of `trgt1_radm_pkt_halt[2:0]` to the assertion of `radm_grant_tlp_type[2:0]`.

Figure 4-20 Grant/Halt Unloading Example



When you have used all the entries in the Target LUT, the RADM automatically halts the TRGT1 interface.

4.6 FC-Receive Request Interface (FC-TRGT1)

For Gen6 configured controller, the TRGT1 interface is specific to a FC (Posted, Non-Posted, Completion) with a minimum of three and maximum of six TRGT1 interfaces.

Table 4-3 FC-TRGT1 Interfaces

TRGT1 Interface	One Set of FC Interfaces	Two Sets of FC Interfaces
Posted	radm_trgt1_p_0_<trgt1_name_suffix>	radm_trgt1_p0_<trgt1_name_suffix> radm_trgt1_p1_<trgt1_name_suffix>
Non-Posted	radm_trgt1_np_0_<trgt1_name_suffix>	radm_trgt1_np0_<trgt1_name_suffix> radm_trgt1_np1_<trgt1_name_suffix>
Completion	radm_trgt1_cpl_0_<trgt1_name_suffix>	radm_trgt1_cpl0_<trgt1_name_suffix> radm_trgt1_cpl1_<trgt1_name_suffix>

All the ports in the "Receive Request Interface (TRGT1)" on page 370 are present in the FC-TRGT1 interfaces.

For x16 lanes, the controller can scale the TRGT1 interface to 1024 bit.

When `CX_FLIT_MODE_ENABLE =1`, the controller has additional input signals (`radm_trgt1_<fc>_tlp_segment` and `radm_trgt1_<fc>_dst_segment`) on each TRGT1 interface to provide Flit Mode Format.

The TLP header encoding and position is different for Flit Mode and Non-Flit Mode format TLPs. This difference in the TLP format is not reflected in the TRGT1 interfaces. For both Flit Mode and Non-Flit Mode, the TLP information is specified in the same way on the TRGT1 interface.

For example, PASID in a Flit Mode TLP is positioned in OHC-A1. In Non-Flit Mode, PASID is positioned in E2E prefix. In both Flit Mode and Non-Flit Mode, the PASID on TRGT1 interface is specified in the `ctrgt1_<fc>_tlp_prfx` signal.

4.6.1 TRGT1 Interface Halting

The TRGT1 interfaces can be halted (unless in bypass) by asserting the `trgt1_radm_intf_<fc>_0_halt` input port or by using the Packet Grant and Halt mechanism as described in "TRGT1 Packet Grant and Halt" on page 371.

4.6.2 TRGT1 TLP Ordering Information

The output signals `radm_trgt1_p_X_pkt_order_no`, `radm_trgt1_np_X_pkt_order_no`, and `radm_trgt1_cpl_X_pkt_order_no` provide TRGT1 TLP ordering information.

Use these signals together with RO, IDO, and RID information manages ordering of TLPs in the application logic.

4.7 CCIX Receive Request Interface (TRGT1_CCIX)

The CCIX specification relies on the use of a dedicated high priority Virtual Channel (VC) to transfer data between devices. A dedicated TRGT1_CCIX interface to pass incoming CCIX TLPs to your application provides the following advantages:

- Minimizes the latency of CCIX traffic on the receive path
- Provides CCIX support without affecting the standard PCIe functionality
- Minimizes collision between standard PCIe traffic and CCIX traffic

TRGT1_CCIX interface implements a simple handshake mechanism between the controller and your application.

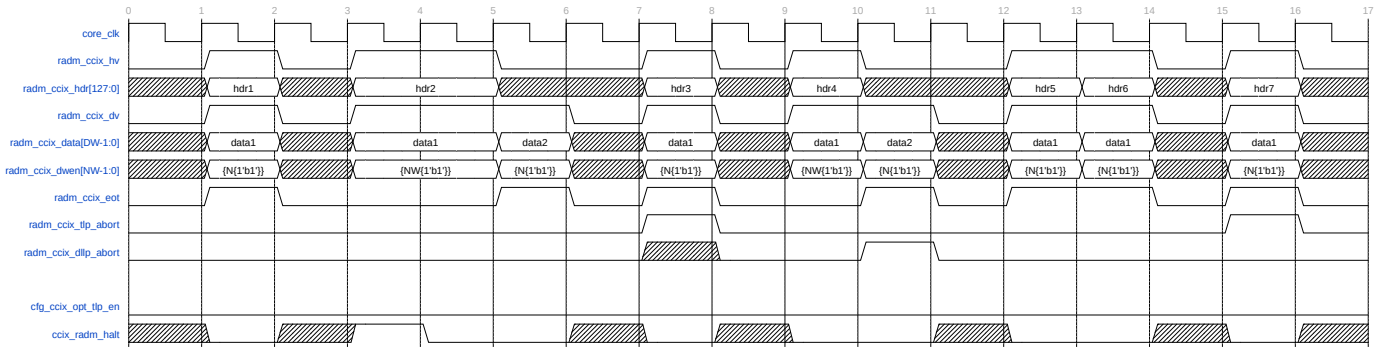
4.7.1 CCIX TRGT1 Protocol Transactions

The following diagrams illustrate example transactions on the CCIX receive client interface:

- Figure 4-21 depicts an example of generic operation of TRGT1_CCIX interface when CCIX TLPs in PCIe compatible TLP format are received at TRGT1_CCIX interface.
- Figure 4-22 depicts an example of generic operation of TRGT1_CCIX interface when CCIX TLPs in optimized CCIX TLP format are received at TRGT1_CCIX interface.
- Figure 4-23 depicts an example scenario for a 128-bit controller when the amount of data received in one clock cycle is not enough to fill the complete radm_ccix_data bus.

In the timing diagrams, hdr*i* (where *i* is 1,2,3 and so on) represents the header for different TLPs, `data1` represents the first data cycle, `data2` represents the second data cycle, and so on.

Figure 4-21 CCIX_TRGT1 PCIe Compatible TLP Format[1]



1. For radm_ccix_dwen, 1 <= N <= NW.

Figure 4-22 CCIX_TRGT1 Optimized CCIX TLP Format[1]

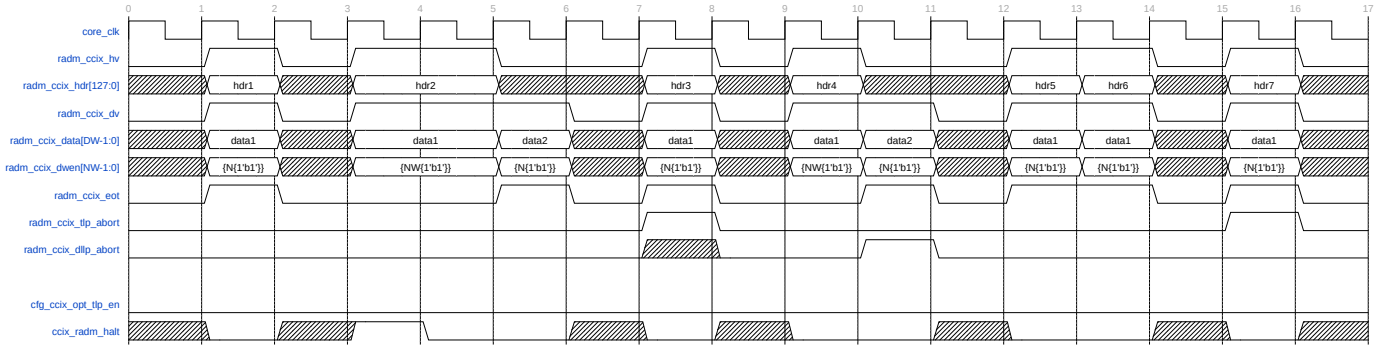
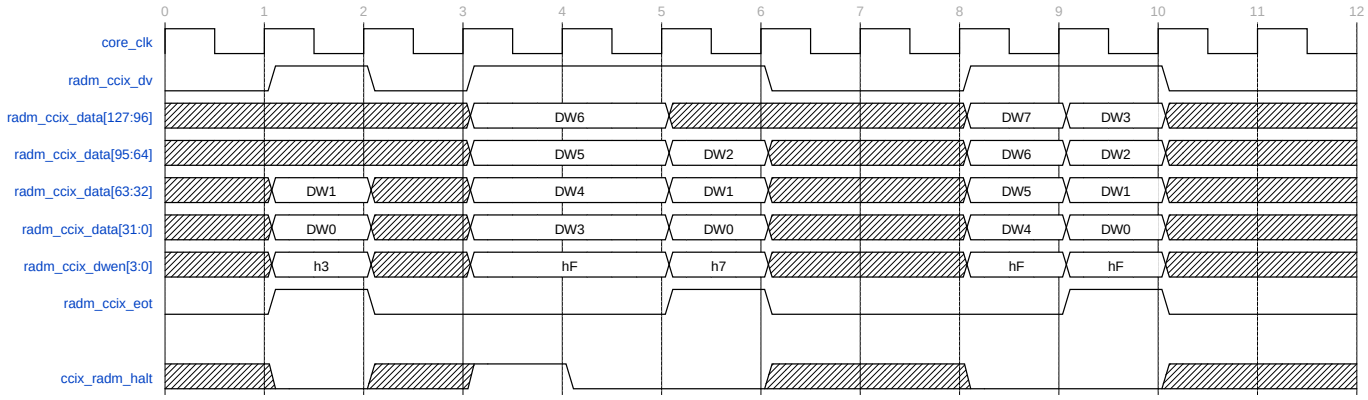


Figure 4-23 CCIX_TRGT1 Data bus Usage Example for 128-bit Datapath



Clock cycle one and clock cycle five in Figure 4-23 illustrate how the `radm_ccix_data` bus is expected to be used when the amount of data received in one clock cycle is not enough to fill the complete `radm_ccix_data` bus. For both PCIe compatible TLP format and optimized CCIX TLP format the same behavior applies.

4.7.2 CCIX TLP Header Fields and radm_ccix_hdr Mapping

Table 4-2 illustrates the mapping between CCIX TLP header fields and the bits in the receive CCIX interface header radm_ccix_hdr.

1. For radm_ccix_dwen, 1 <=N <=NW.

Table 4-4 Mapping Between CCIX TLP Header Fields and radm_ccix_hdr

CCIX TLP Field		radm_ccix_hdr bits
PCIe Compatible TLP Format	**Optimized CCIX TLP Format**	
TLP DW3	NOT USED	96 - 127
Vendor ID		95 - 80
Device Number		79 - 75
Function Number		74 - 72
Bus Number		71 - 64
Message Code		63 - 56
TAG		55 - 48
Requester ID		47 - 32
Length[7:0]	CCIX DW0 Content	31
	Length[6:0]	30 - 24
TD	CCIX DW0 Content	23
EP		22
Attr		21 -20
AT		19 -18
Length[9]		17
Length[8]		16
R		15
TC	TC	14 - 12
R	CCIX DW0 Content	11
Attr[2]		10
R		9
TH		8
FMT	0	7
	Type	6
	CCIX DW0 Content	5
Type		4 - 0

4.8 Data Bus Interface (DBI)

Attention In the timing diagrams, the actual clock used to clock the interface might vary in low-power mode. In this case, you should use a different clock than the one indicated in the timing diagram. For more information, see "Synchronous To Attribute" on page 71.

For more information, see "Data Bus Interface (DBI) Access" on page 146. Limitations are detailed in "Access Limitations" on page 150.

4.8.1 DBI Protocol Rules

The DBI protocol rules are:

- Assertion of `dbi_cs` indicates an active CPU cycle.
- `The dbi_wr` input indicates the byte enables of a write access. All bits zero indicates a read access.
- The `dbi_cs` and `lbc_dbi_ack` signals form a synchronous handshake. The signal `dbi_cs` must remain asserted until the controller asserts `lbc_dbi_ack`.
- The controller asserts `lbc_dbi_ack` for two clock cycles.
- When `dbi_addr[0] =0`, the access is to an internal configuration register in the CDM.
- The DBI includes a second chip select signal (`dbi_cs2`) to access:
  - shadow registers
  - iATU registers
  - DMA registers

  For more information, see "Register Types" on page 143.

4.8.2 DBI Protocol Transactions

The following diagrams illustrate example transactions on the DBI:

- Figure 4-24 depicts an example of a CPU performing write accesses to internal configuration space registers through the DBI.
- Figure 4-25 depicts an example of a CPU performing read accesses to configuration space registers through the DBI.

Figure 4-24 DBI Transaction: Register Write

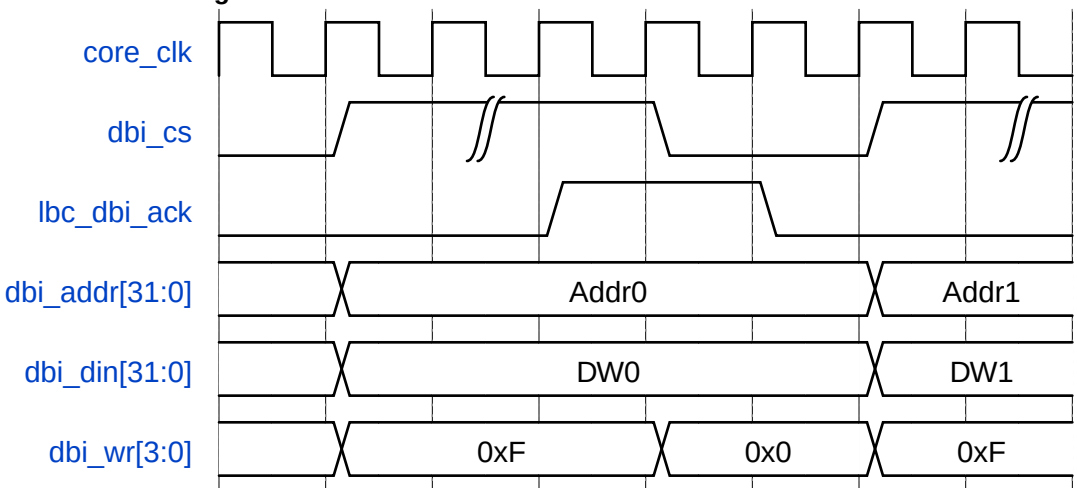
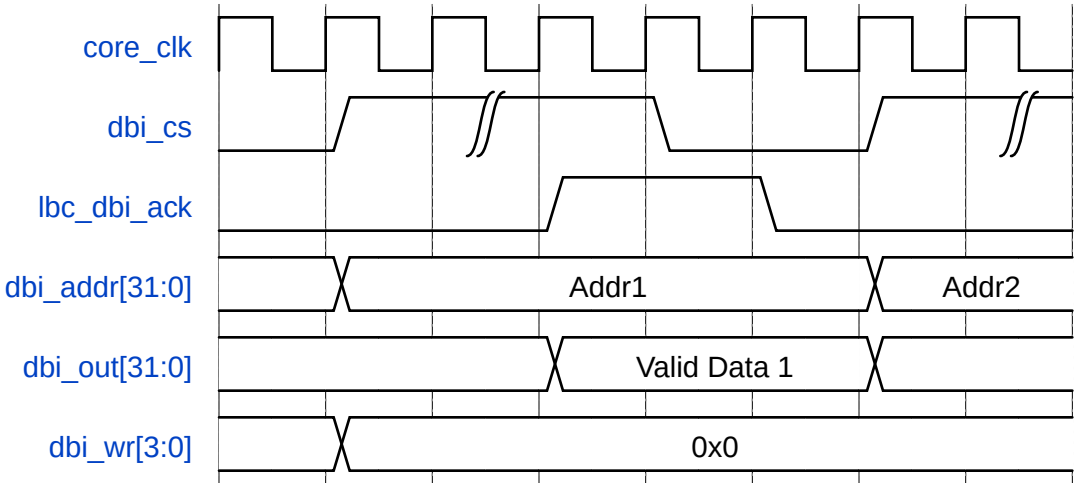


Figure 4-25 DBI Transaction: Register Read



4.9 External Local Bus Interface (ELBI)

For more information, see "Register Module, LBC, and DBI" on page 141. Limitations are detailed at "Access Limitations" on page 150.

4.9.1 ELBI Protocol Rules

- Assertion of `lbc_ext_cs` indicates an active request cycle.
- The `lbc_ext_wr` output indicates the byte enable of a write access. For a read access, the controller de-asserts all bits of this signal.
- Refer to the ELBI Access Restrictions in "Access Limitations" on page 150.
- It is expected that your application only decodes the valid address bits to the BAR limit specified.
- The `lbc_ext_dout` output is only valid when `lbc_ext_cs` is asserted.
- The `lbc_ext_cs` and `ext_lbc_ack` signals form a synchronous handshake. The controller keeps `lbc_ext_cs asserted` until your application asserts `ext_lbc_ack`. The wait time between `lbc_ext_cs` and `ext_lbc_ack` is unlimited.
  - Your application must return an ack, otherwise, the transaction hangs. It is suggested that your application designs an error access detection circuit. When mis-access is encountered, return an ack with null data.
  - The `ext_lbc_ack` signal should never be hardwired to 1b1, even if the ELBI is not being used. It should only be high when `lbc_ext_cs` is 1b1. That is, it should be inactive when `lbc_ext_cs` is inactive.
  - Your application should assert `ext_lbc_ack` signal for two clock cycles for successful register accesses. In case of an erroneous access (that is, accessing a non-existent register or writing to a read-only register) the application can assert the `ext_lbc_ack` signal for one clock cycle. This error response is then sent to the `lbc_dbi_ack` signal for a DBI request. This error response does not affect the completion status for the accesses from the wire.
- The controller sets the `lbc_ext_bar_num` output to 3'b111 for all DBI accesses.

4.9.2 ELBI Protocol Transactions

The following diagrams illustrate example transactions on the ELBI:

- Figure 4-26 depicts an example of the LBC performing a 32-bit write access to an external application register through the ELBI.
- Figure 4-27 depicts an example of the LBC performing a 32-bit read access to an external application register through the ELBI.

Figure 4-26 ELBI Transaction: 32-bit Write Access to External Registers

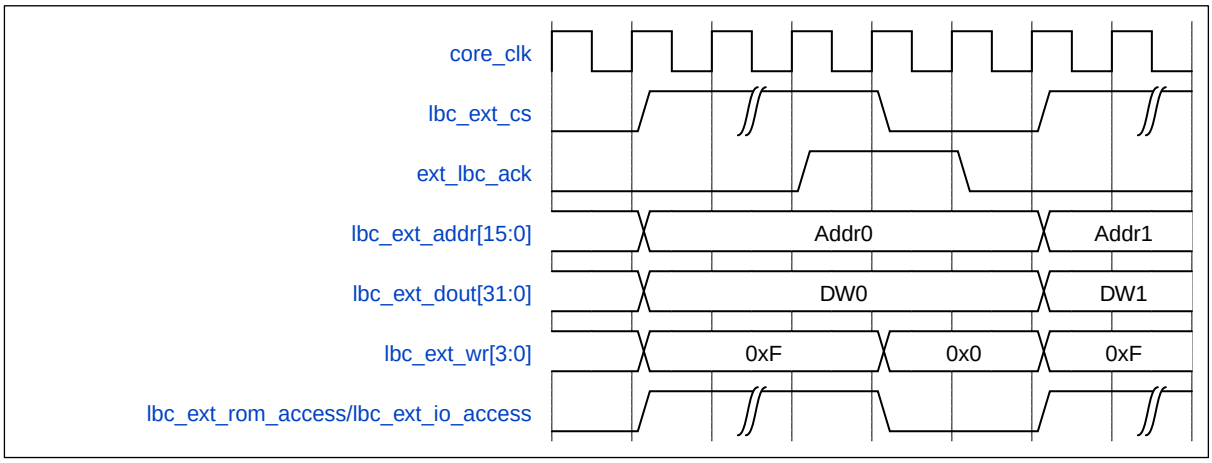
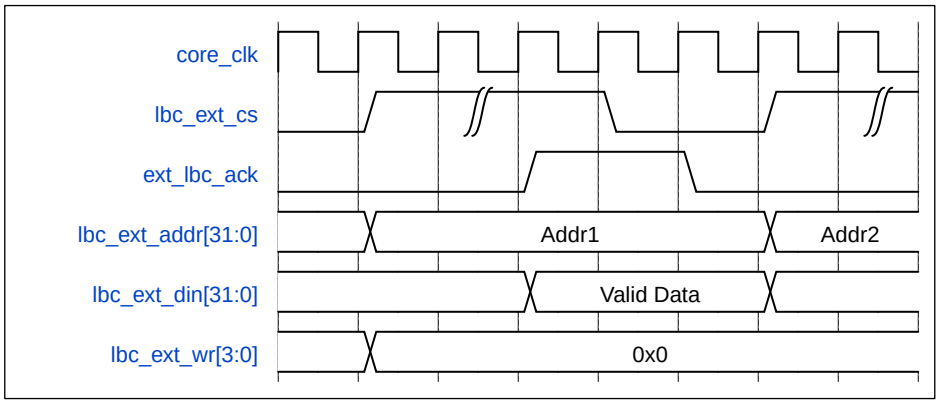


Figure 4-27 ELBI Transaction: 32-bit Read Access to External Registers



4.10 Vendor Message Interface (VMI)

The VMI is an interface for sending vendor-defined messages. The VMI protocol is a simple synchronous request/acknowledge handshake. The VMI can only be used to send a header-only (no payload) vendor-defined message (Msg). MsgD is not supported. To send a vendor-defined MsgD, you must use one of the transmit client interfaces XALI0/1/2 (or AXI bridge).

Attention In the timing diagrams, the actual clock used to clock the interface might vary in low-power mode. In this case, you should use a different clock than the one indicated in the timing diagram. For more information, see "Synchronous To Attribute" on page 71.

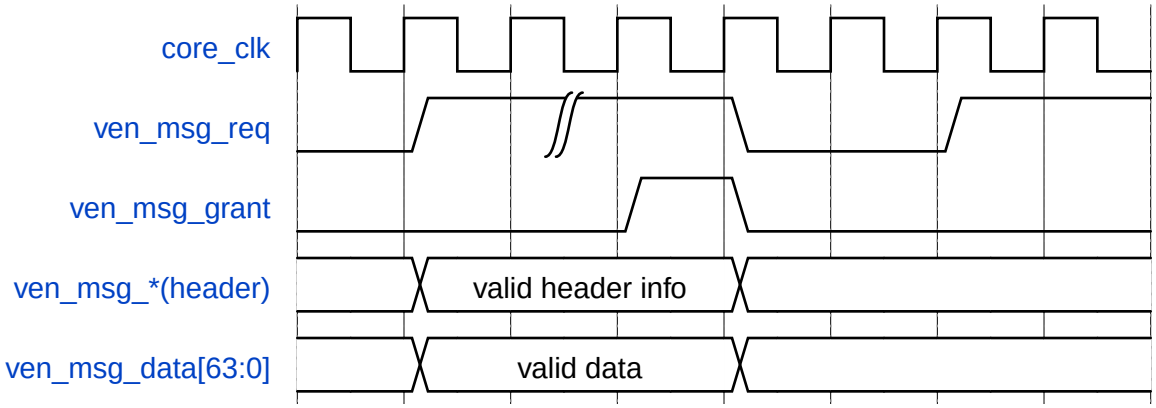
4.10.1 VMI Protocol Rules

- The `ven_msg_req` pin must be tied to 0 if the VMI is not used. For more information, see "Message Generation" on page 194.
- When `ven_msg_req` is asserted, it must remain asserted until the controller asserts `ven_msg_grant`.
- The time between `ven_msg_req` and `ven_msg_grant` is undefined.
- Your application must de-assert `ven_msg_req` on the next clock cycle after sampling `ven_msg_grant` high.

4.10.2 VMI Protocol Transactions

- Figure 4-28 depicts an example where your application client requests the controller to send a vendor-defined message upstream. The ven_msg_req signal stays asserted until the controller asserts ven_msg_grant.

Figure 4-28 Vendor Message Request



4.11 System Information Interface (SII)

The SII exchanges various system-related information between the controller and your application. Most of the SII signals are provided for flexibility. Your application is not required to use all of the SII signals. Your application logic is expected to drive and monitor the signals that it needs to function correctly. SII inputs that your application does not require, must be driven to 0.

4.12 PIPE Interface

The PHY is instantiated outside of the controller, and there is an external PIPE interface for connection to the external PHY. For more information, see the Integrating the controller with the PHY section in the *User Guide*. In the multi-lane case, the per-lane signals are concatenated to form a bus, with the low-order bits corresponding to lane0. The PIPE is compliant with the *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1*.

Note

The controller does not use the following PIPE signals:

- PHY Mode[1:0] output. You must externally drive the corresponding PHY input with the value '00'
- DataBusWidth[1:0] input
- Max PCLK input

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

When the automatically derived `CX_FREQ_STEP_DOWN_EN` parameter is '1', then a module called `freq_step` (`workspace/src/common/freq_step.sv`) is placed between the PHYs PIPE and the controllers PIPE. This module steps up/down the signals to/from the PIPE interface. For more information, see "Frequency Step Module (CX_FREQ_STEP_EN =1)" on page 49.

4.12.1 64-bit PIPE

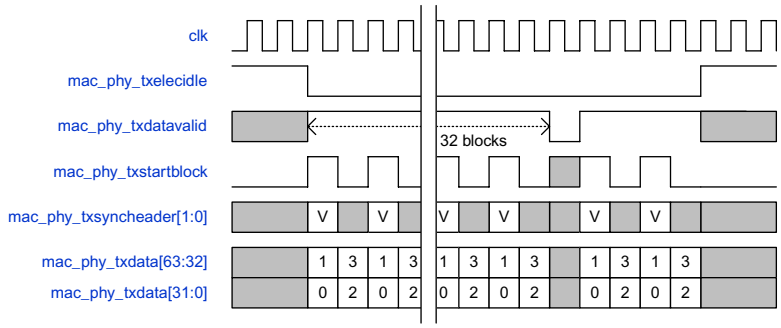
The controller supports 64-bit PIPE. This section describes the 64-bit PIPE transmit and receive operation.

4.12.1.1 64-bit PIPE Transmit Operation

Figure 4-29 describes the 64-bit PIPE transmit operation.

- De-assert `txdatavalid` for one cycle on every 32 blocks
- Block length is always 128-bit aligned

Figure 4-29 64-bit PIPE Transmit Operation

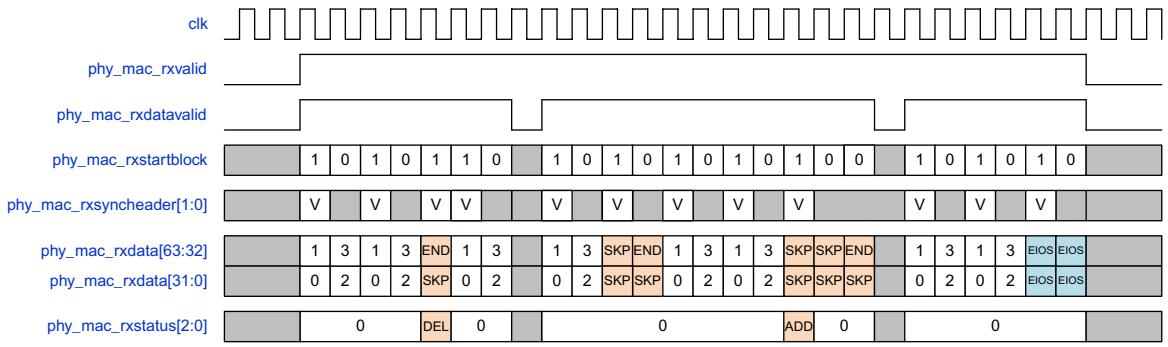


4.12.1.2 64-bit PIPE Receive Operation (CX_PIPERX_MULTI_BLOCK =0)

Figure 4-30 describes the 64-bit PIPE transmit operation when CX_PIPERX_MULTI_BLOCK =0.

- The controller only accepts 64-bit-aligned SKP OS
- `rxdatavalid` can be de-asserted at anytime
- Possible SKP OS length: 64/128/192 bit

Figure 4-30 64-bit PIPE Receive Operation (CX_PIPERX_MULTI_BLOCK =0)

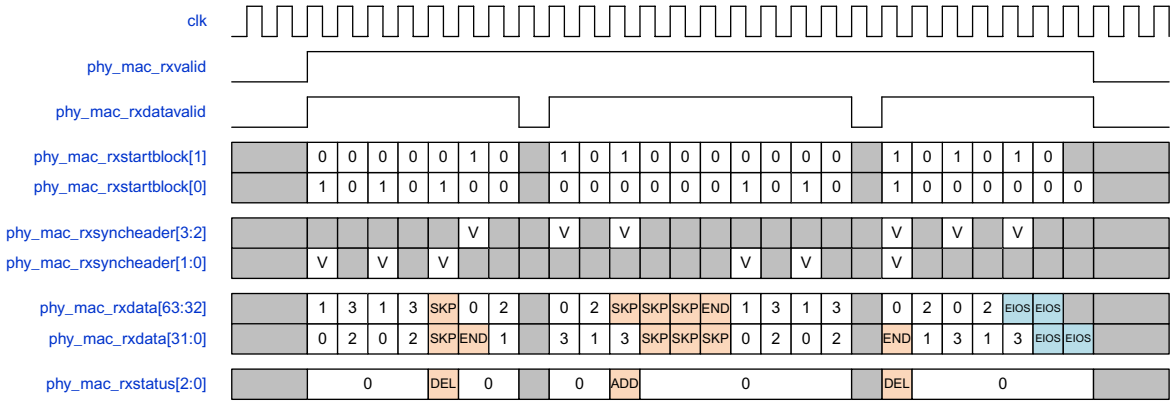


4.12.1.3 64-Bit PIPE Receive Operation (CX_PIPERX_MULTI_BLOCK =1)

Figure 4-31 describes the 64-bit PIPE transmit operation when CX_PIPERX_MULTI_BLOCK =1.

- Extend bit width of `rxstartblock` and `rxsyncheader`
  - `phy_mac_rxstartblock` 2-bit (per-lane)
  - `phy_mac_rxsyncheader` 4-bit (per-lane)
- `rxdatavalid` can be de-asserted at anytime
- Possible SKP OS length: 32/64/96/128/160/192/224 bit

Figure 4-31 64-bit PIPE Receive Operation (CX_PIPERX_MULTI_BLOCK =1)



4.12.2 PIPE Interfaces

The controller implements Low Pin Count Interface and SerDes PIPE architecture features for PIPE 5.1.1, PIPE 6.0 and PIPE 6.1 according to PIPE Specification for PCI Express, Version 5.1.1. The topics discussed in this section are:

- "Low Pin Count Interface"
- "SerDes Architecture" on page 387
- "Asynchronous Deep Power Management (Optional)" on page 388

4.12.2.1 Low Pin Count Interface

The controller implements Low Pin Count Interface with the following features and limitations:

Features

- Low Pin Count Interface removes obsolete Legacy Pin Interface signals.
- The controller implements PHY/MAC Message Bus registers.
- The controller supports Gen1, Gen2, Gen3, Gen4, Gen5, and Gen6 speed (when `CX_PIPE_VER >=4`)

Table 4-5 Limitations

Limitations (when CX_PIPE_VER >=3)
The following PHY Message Bus registers are not supported: ■ Elastic Buffer Control (2h)/Elastic Buffer Depth Control ■ PHY RX Control4 (8h)/Elastic Buffer Reset Control
The following MAC Message Bus registers are not supported: ■ Elastic Buffer Status (3h)/Elastic Buffer Status ■ Elastic Buffer Location (4h)/Elastic Buffer Location
The controller does not issue any Message Bus command to the unsupported PHY registers. Likewise, any Message Bus commands to the unsupported MAC registers does not affect to the behavior of the controller.

Limitations (when CX_PIPE_VER >=4)	
The following PHY Message Bus registers are not supported:	
PHY RX Control1 (4h)	■ Bit[1]: IORecal (Optional) ■ Bit[0]: ExEqTraining (Optional)

The following MAC Message Bus registers are not supported:	
Rx Status0 (5h)	■ Bit[1]: IORecalDone (Optional) ■ Bit[0]: RxEqTraining (Optional)
Rx Control0 (6h)	■ Bit[0]: PHYIORecalRequest (Optional)

The following encoding for the LinkEvalFeedbackDirectionChange registers are not supported. These encoding are not used in PCIe Gen6 base specification. ■ If the register is set to “Increment by 2” or “Increment by 4”, the register is treated as “Increment by 1” ■ If the register is set to “Decrement by 2” or “Decrement by 4”, the register is treated as “Decrement by 1” Encoding: ❑ 000 -- No change (Supported) ❑ 001 -- Increment by 1 (Supported) ❑ 010 -- Decrement by 1 (Supported) ❑ 011 – Increment by 2 (Not supported) ❑ 100 – Decrement by 2 (Not supported) ❑ 101 – Increment by 4 (Not supported) ❑ 110 – Decrement by 4 (Not supported) ❑ 111 – Reserved	
RX Link Evaluation Status2 (Eh)	■ Bit[6:4]: LinkEvalFeedbackDirectionChange_Cminus1[2:0] ■ Bit[2:0]: LinkEvalFeedbackDirectionChange_Cminus2[2:0]
RX Link Evaluation Status3 (Fh)	■ Bit[6:4]: LinkEvalFeedbackDirectionChange_Cplus1[2:0] ■ Bit[2:0]: LinkEvalFeedbackDirectionChange_Czero[2:0]

The following bits for LocalTxPresetCoefficients and LocalG6 registers are ignored.These bits are not used in PCIe Gen6 base specification.	
Tx Status0 (400h)	■ Bit[7:0]: LocalTxPresetCoefficients_Cminus1[7:0] ❑ Bit[7:4] is ignored at gen6 rate ❑ If the value exceeds 16, the value is treated as 15 (8’b0000_1111)
Tx Status1 (401h)	■ Bit[7:0]: LocalTxPresetCoefficients_Czero[7:0] ❑ Bit[7:6] is ignored at gen6 rate ❑ If the value exceeds 64, the value is treated as 63 (8’b00_111111)
Tx Status2 (402h)	■ Bit[7:0]: LocalTxPresetCoefficients_Cplus1[7:0] ❑ Bit[7:5] is ignored at gen6 rate ❑ If the value exceeds 32, the value is treated as 31 (8’b000_11111)
Tx Status7 (407h)	■ Bit[7:0]: LocalTxPresetCoefficients_Cminus2[7:0] ❑ Bit[7:3] is ignored at gen6 rate ❑ If the value exceeds 8, the value is treated as 7 (8’b00000_111)
Tx Status8 (408h)	■ Bit[7:0]: LocalG6FS[7:0] ❑ Bit[7:6] is ignored at gen6 rate ❑ If the value exceeds 64, the value is treated as 63 (8’b00_11111)
Tx Status9 (409h)	■ Bit[7:0]: LocalG6LF[7:0] ❑ Bit[7:6] is ignored at gen6 rate ❑ If the value exceeds 64, the value is treated as 63 (8’b00_11111)

The controller does not issue any Message Bus command to the unsupported PHY registers. Likewise, any Message Bus commands to the unsupported MAC registers does not affect to the behavior of the controller.

4.12.2.2 SerDes Architecture

The controller implements SerDes architecture (`CX_PIPE_SERDES_ARCH =1`) with the following features and limitation:

Features

- 8b/10b encoding/decoding and error detection
- 128b/130b encoding/decoding and error detection
- Symbol lock
- Block alignment
- Elastic buffer
  - Support Nominal Half Full Buffer mode/Nominal Empty Buffer mode
  - Support SRIS(5600 ppm)/SRNS(600 ppm)
- Polarity inversion
- Remote digital loopback
- The controller supports Gen3, Gen4, and Gen5 speed
- The controller supports the PIPE width as defined in the "Controller-PHY Compatibility" on page 44.
- 8 SKIP symbols for 128b/130b encoding (when `CX_PIPE_VER >=3`)
- 1b/1b encoding/decoding and error detection (when `CX_PIPE_VER >=4`)

Lanes Turn off

The controller supports lane 'turn off' for PIPE SerDes architecture, when the parameter `CX_SERDES_ARCH_LANES_TURNOFF` is set to '1'. This is a Synopsys defined feature and provides the same functionality as lane 'turn off' method described in *PIPE Specification for PCI Express Version 5.1.1*. When this feature is enabled, a new output signal `serdes_pipe_turnoff_lanes` is available. The PHY must 'turn off' the lane for the corresponding bit set in `serdes_pipe_turnoff_lanes`. The PHY behavior corresponding to the 'turned off' lanes must be the same as in the scenario where the lanes are 'turned off' using TxElecIdle and TxCompliance signals (as described in *PIPE Specification for PCI Express Version 5.1.1)*.

Gate Count Costs

When you configure the controller to support SerDes architecture, an additional logic for SerDes PIPE is incurred. Table 4-6 shows an example SerDes PIPE area. The actual numbers are configuration specific.

Table 4-6 Example of SerDes PIPE Area

Product			x1	x2	x4	x8	x16
Area (Kgates)[a]							
	Speed Mode	SRIS Support					

Product			x1	x2	x4	x8	x16
Gen4 Configurations	g1_2s g2_2s_df g3_2s_df g4_2s_df	✗	18.03	39.42	77.92	154.49	288.76
		✓	25.86	51.85	110.21	216.30	413.89
Gen5 Configurations	g1_4s g2_4s_df g3_4s_df g4_4s_df g5_4s_df	✗	35.71	71.52	142.56	285.09	572.90
		✓	45.58	91.24	182.74	365.34	729.42
	g1_4s g2_4s_df g3_8s_dw g4_8s_df g5_8s_df	✗	91.57	183.20	365.81	735.03	NA
		✓	109.98	220.92	440.45	881.90	NA

a. These example area estimations are done with 1 GHz clock frequency. The values might be different in your environment.

4.12.2.3 Asynchronous Deep Power Management (Optional)

This release of the controller does not support the Asynchronous Deep Power Management feature.

4.13 PHY Register Bus Interface (PRBI)

PHY Register Bus Interface (`CX_PHY_VIEWPORT_ENABLE =1`) is an optional interface between the controller and Synopsys PHYs which communicate with the PHY's Control Register (CR) Parallel Interface. It facilitates in direct access of PHY control registers through DBI or CFG requests to the controller. PRBI is a synchronous (maximum frequency 100MHz), 16-bit data/address parallel interface that follows a simple synchronous request/acknowledge handshake protocol. The interface also implements timeouts to avoid deadlocks.

The `PHY_VIEWPORT_CTLSTS_OFF` port logic register, controls the access of PHY through CFG or DBI requests. For more information, see "PHY Port Logic Registers" on page 152. For more information on cold reset sequence with PHY initialization, see Figure 2-26. For more information on PHY initialization after cold reset, see Figure 2-27.

Note

- The PHY can be controlled through PRBI only when PHY's Control Register (CR) Parallel Interface Select signal (`cr_para_sel`) is set to 1.
- This feature does not work during L1.CPM and L1 substate, if `CX_PHY_ENHANCED_PM_EN =1`.
- This feature does not work during L2.

Attention

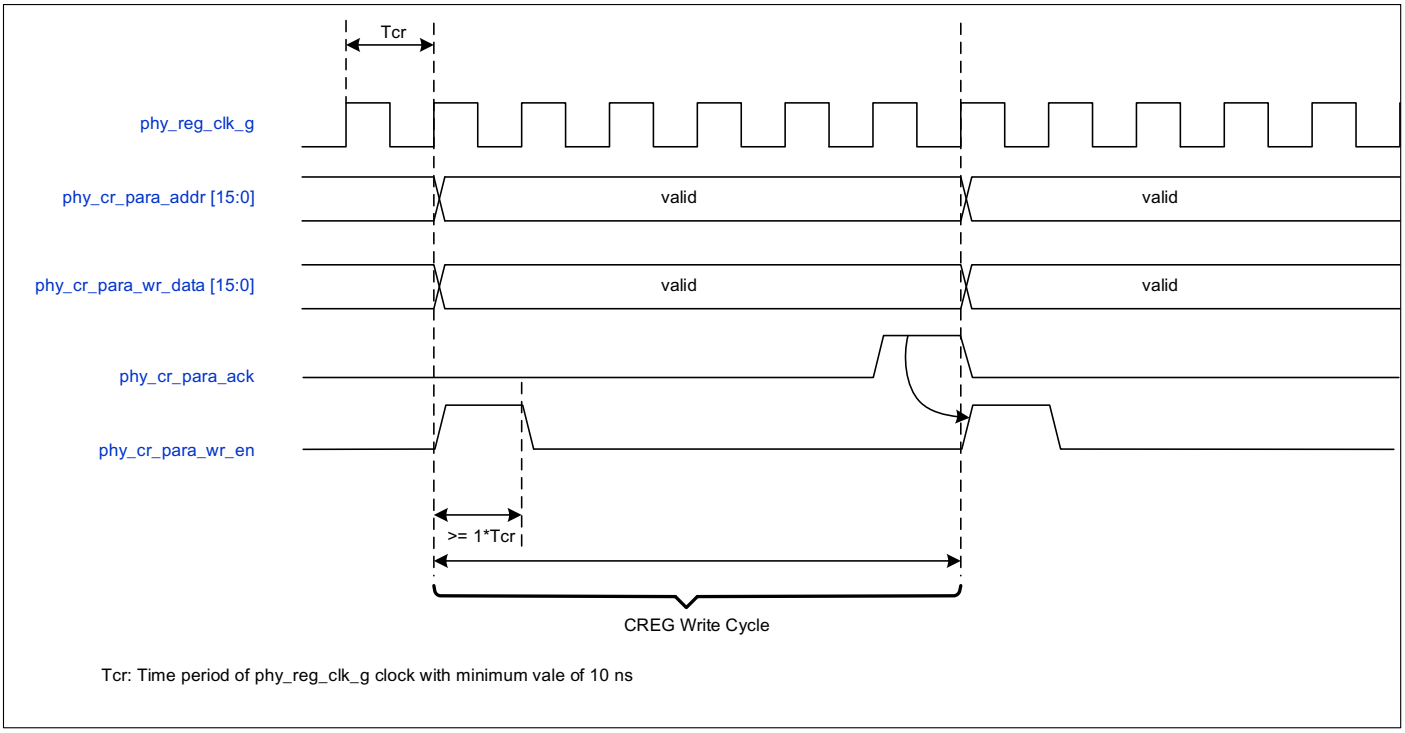
The controller's PHY Viewport (PRBI) to PHY registers can have high access latency. If your application need slow latency to initialize Synopsys PHY registers or SRAM, then PRBI should not be used. Instead, you should design a direct interface to the PHY's SRAM or CR Bus. PRBI is recommended only for PHY debug.

4.13.1 PRBI Protocol Transactions

- Figure 4-32 depicts an example of the controller performing a write operation on a PHY containing a single sub-block.
- Figure 4-33 depicts an example of the controller performing a read operation on a PHY containing a single sub-block.
- Figure 4-34 depicts an example of the controller performing a write operation on PHY0 sub-block of a PHY containing four sub-blocks.
- Figure 4-35 depicts an example of the controller performing a read operation on PHY0 sub-block of a PHY containing four sub-blocks.
- Figure 4-36 depicts an example of the controller performing a write operation on all the sub-blocks of a PHY containing a four sub-blocks.

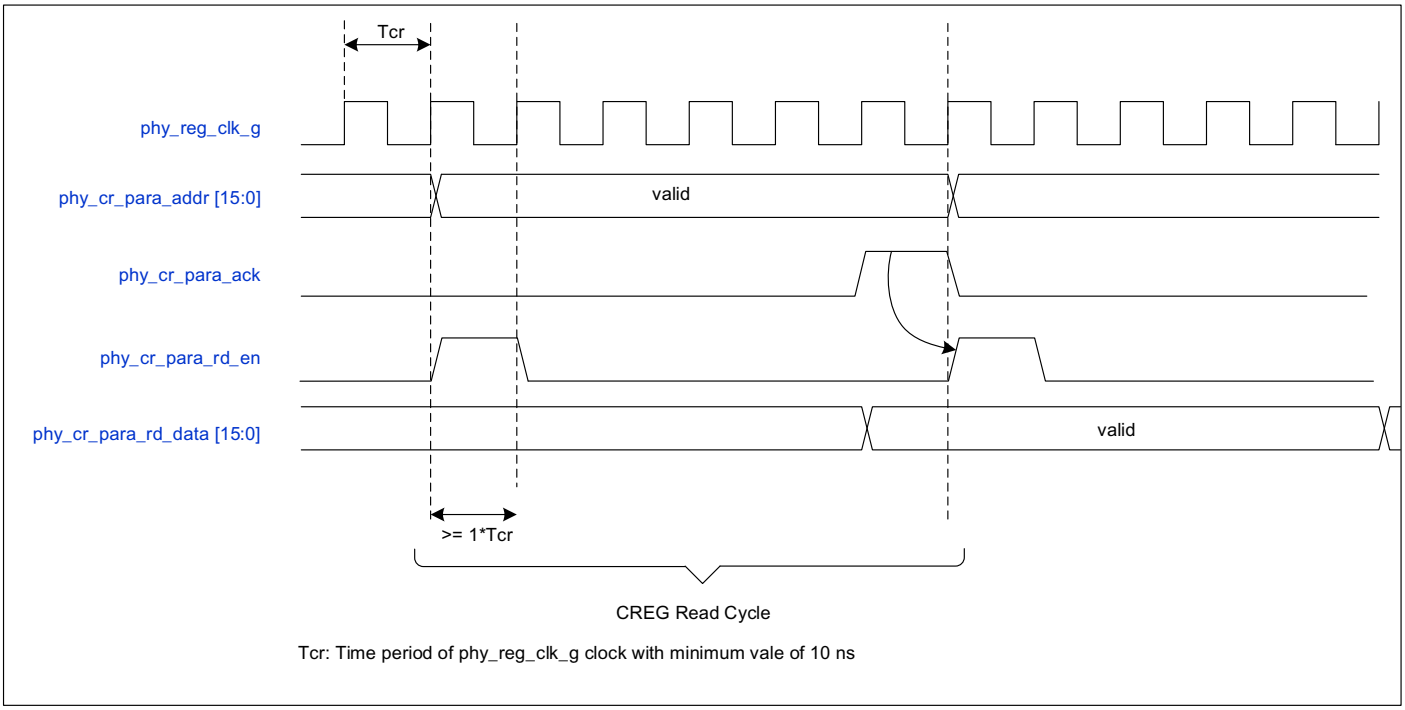
Note For accurate timing information, see *PHY Databook*.

Figure 4-32 Write Operation: PHY Containing Single PHY Sub-block



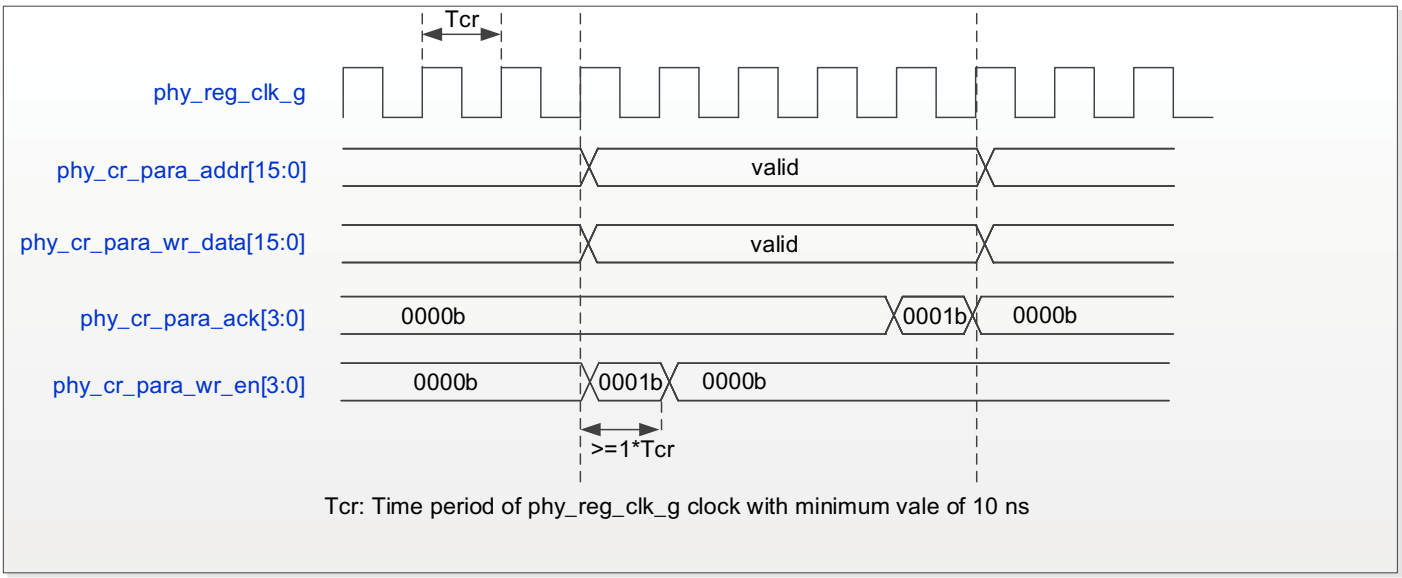
`phy_cr_para_wr_en` is asserted for a single cycle. When asserted, the data on `phy_cr_para_wr_data[15:0]` is written to the address on `phy_cr_para_addr[15:0]`. Assertion of `phy_cr_para_ack` indicates completion write access.

Figure 4-33 Read Operation: PHY Containing Single PHY Sub-block



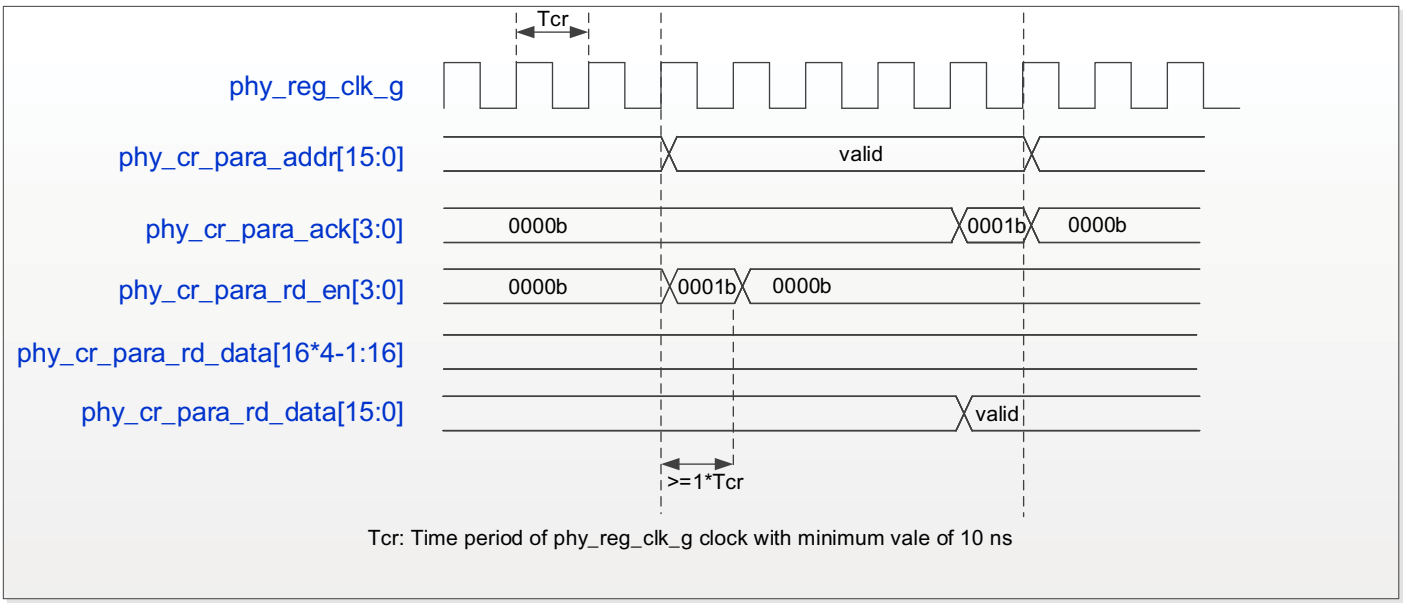
`phy_cr_para_rd_en` is asserted for a single cycle. When asserted, the data at address `phy_cr_para_addr[15:0]` is read and provided on `phy_cr_para_rd_data[15:0]`. Assertion of `phy_cr_para_ack` indicates completion of read access.

Figure 4-34 Write Operation: PHY0 Sub-block of a PHY Containing 4 Sub-blocks



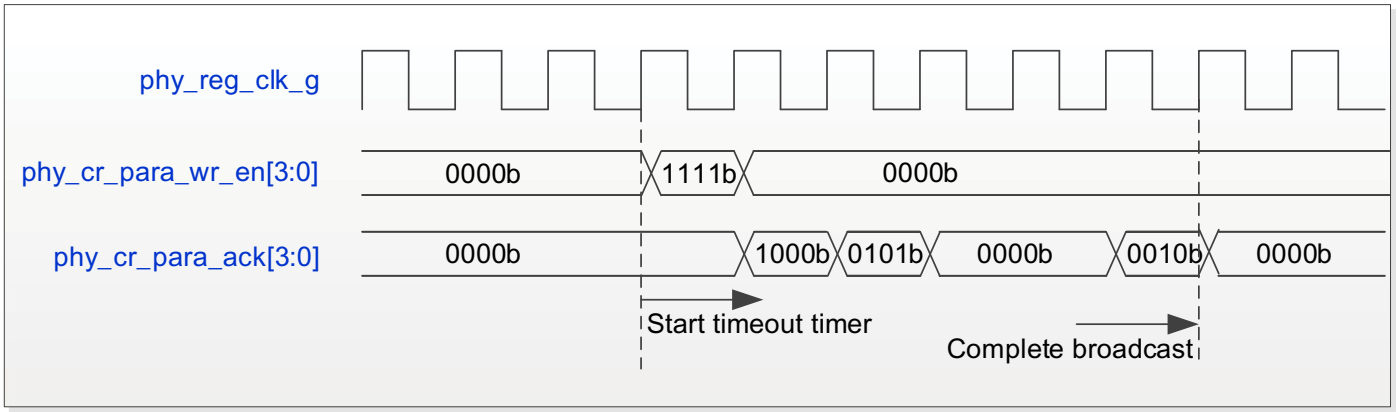
`phy_cr_para_wr_en[3:0]` is asserted for a single cycle. The value 0001b indicates write access to PHY0 sub-block of the PHY. The controller waits for response from PHY0 sub-block.

Figure 4-35 Read Operation: PHY0 Sub-block of a PHY Containing 4 Sub-blocks



`phy_cr_para_rd_en[3:0]` is asserted for a single cycle. The value 0001b indicates read access to PHY0 sub-block of the PHY. Data at address `phy_cr_para_addr[15:0]` is read and provided on `phy_cr_para_rd_data[15:0]`. The controller waits for response from PHY0 sub-block.

Figure 4-36 Write Operation: Broadcast to all Sub-blocks of a PHY Containing 4 Sub-blocks



`phy_cr_para_wr_en[3:0]` is asserted for a single cycle. Value 1111b indicates write access to all the four sub-blocks of the PHY. The controller waits for response from all the four sub-blocks.

5

AMBA

The PCIe controller with AMBA bridge supports AXI3, AXI4, and AXI5 (AMBA AXI protocols), ACE-Lite (AMBA ACE protocol), and Distributed Translation Interface (DTI) Master (AMBA DTI-ATS protocol).

The following topics are discussed in this section:

- "AXI3/AXI4/AXI5 Support in PCIe Controller with AMBA Bridge" on page 394
- "ACE-Lite Support in PCIe Controller with AMBA Bridge" on page 445
- "DTIM Support in PCIe Controller with AMBA Bridge" on page 447
- "LTIM, Order Buffer, and MSI-GIC Interface" on page 456

5.1 AXI3/AXI4/AXI5 Support in PCIe Controller with AMBA Bridge

This section describes the AXI bridge for the PCIe controller. Unless otherwise specified, the term AXI bridge is used to refer the AXI3, AXI4, and AXI5 functionality supported by the PCIe controller. The following topics are discussed in this section:

- "AXI Bridge Overview" on page 394
- "AXI Bridge Architecture" on page 400
  - "Bridge Architecture Overview" on page 400
  - "AXI Bridge Ordering" on page 408
  - "AXI Bridge Initialization, Clocking and Reset" on page 414
- "AXI Bus Operations" on page 414
  - "AXI Sideband (Misc. Bus) Signals" on page 414
  - "Supported AXI Burst Operations" on page 418
  - "Supported AXI Transfer Sizes" on page 419
  - "Non Contiguous Byte Enable (NCBE) Support" on page 425
  - "Zero-Byte Transfers Over the AXI Bridge (Flush Semantics)" on page 427
  - "I/O and CFG Transaction Handling" on page 427
  - "Error Handling" on page 431
- "Advanced AXI Reference Material" on page 435
  - "AXI Decomposition Rules" on page 435
  - "Outbound Bridge Tag Management" on page 439
  - "Inbound Bridge Tag Management" on page 441

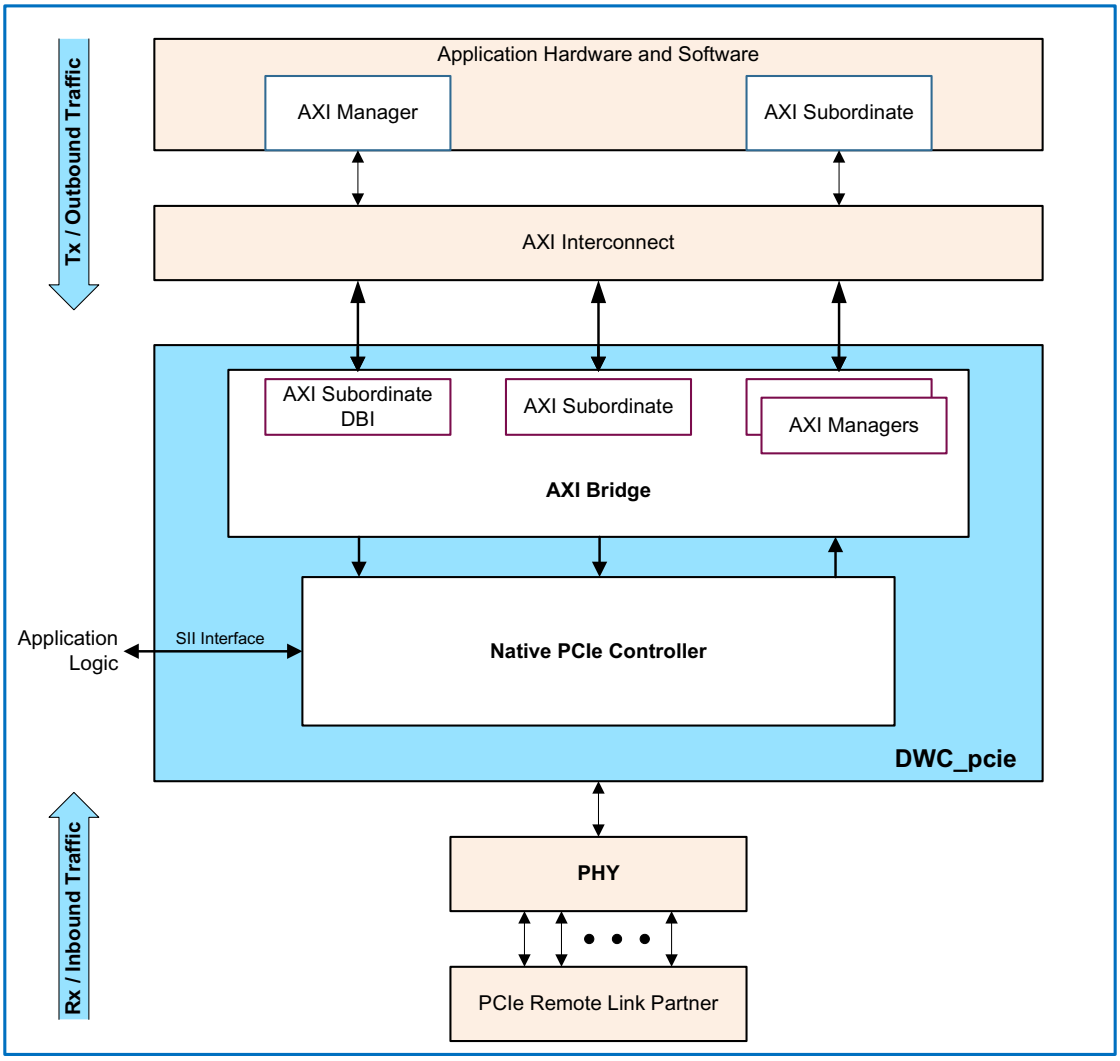
5.1.1 AXI Bridge Overview

The controller provides an optional AXI3, or AXI4, or AXI5 bridging capability for directly adding a PCI Express link to an AXI system fabric. The AXI bridge module acts as a bridge between the standard AXI interfaces and the PCIe controller native interfaces. The bridge interconnects the AXI interfaces within an AXI-embedded system with a remote PCIe link, as either a Root Port or as an Endpoint. The bridge supports up to four AXI interfaces:

- one or two managers
- one link subordinate
- one DBI subordinate

The AXI manager interfaces enable a remote PCIe device to read and write to an AXI subordinate connected to the AXI bridge. The AXI link subordinate interface enables an AXI manager to read and write through the AXI bridge to a remote PCIe device. The DBI slave (see "Data Bus Interface (DBI) Access" on page 146) enables an AXI manager to access the controller's registers.

Figure 5-1 System-Level View of the DWC PCIe AXI controller



5.1.1.1 AXI Features and Limitations

The AXI bridge module supports the following features:

Table 5-1 AXI Bridge Features

AXI Bridge Feature	Supported
AXI3 Protocol Support	Yes
AXI4 Protocol Support	Yes
AXI5 Protocol Support[a]	Yes
ACE Protocol Support	Yes (Lite)
Quality of Service (QOS) for Multi-VC Configurations Using Multiple Masters	Yes[b]

AXI Bridge Feature	Supported
Address Bus Width	32, 64
Databus Width	32, 64,128, 256,512
Independent Internal, Master and Subordinate Databus Widths	Yes
INCR Burst	Yes
WRAP Burst	Yes
FIXED Burst	No
Selectable Maximum Allowable Master Burst Length: 8,16, 32, 64, 128, 256 Beats	Yes
Selectable Maximum Allowable Subordinate Burst Length: 8,16, 32, 64, 128, 256 Beats	Yes
Subordinate Write Interleaving with Reads	Yes
Subordinate Write Interleaving with Writes of Different AWIDs	No
Subordinate Read Data Interleaving	No
Master Write Interleaving with Reads	Yes
Master Read Response Interleaving	Yes
Master Read Response with Write Response Interleaving	Yes
Outbound Zero-Byte Transfer	Write Only
Inbound Zero-Byte Transfer[c]	Yes
Subordinate Write Poisoning by Application[d]	Yes
Graceful Flush and Reset When Link Is Down	Yes
Programmable ID Bus Size	Yes
Non-aligned Starting Address	Yes
Exclusive Access Support	No
Locked Access Support	No
AXI Power Modes	No
Synopsys-specific low-power Modes	Yes
Support for Gating AXI Clocks	Yes
NCBE supported for Inbound Read/Write TLPs	Yes
NCBE supported for Outbound Read/Write TLPs	Write Only
External/Internal RAM	Yes
Programmable Buffer Depths	Yes

AXI Bridge Feature	Supported
CFG and I/O Transactions Support (Sideband Signaling)	Yes
CFG and I/O Transactions (iATU)	Yes
Separate (32-bit) or Shared Subordinate to access the Controller Configuration Space (DBI)	Yes
Independent Master, Subordinate and Controller Clocks	Yes
Programmable Number of Outstanding Read Requests (Either Direction)	Yes
Endianess	Little only
Programmable Master Page Boundary for Inbound Decomposition	Yes
Automatic Decomposition of Inbound and Outbound Requests to Support Independent PCIe and AXI Maximum Read Request and Transfer Sizes[e]	Yes
Datapath and RAM Protection For Limitations, see "RAS Data Protection (RAS DP)" on page 164.	Yes
Bridge RAM Read Enable for Low-power Optimization	Yes
Programmable Integrated MSI Reception Module to Terminate Rx MSI TLPs	Yes
AER Support for Bridge/Bus Errors	No
Bridge Sends Completion for Error on Inbound Non-Posted Request	Yes
Full CDC	Yes
Method of Signaling TLP Header Information	Sideband
Internal ATU Support	Yes
Automatic Graceful Flushing of Bridge Before Reset When Link Goes Down	Yes
Automatic Flushing of Subordinate when Link is Not Up	Yes
PCIe Ordering Rule Compliant	Yes
Virtual Channels (VC)	Up to eight channels
Latency Tolerance Reporting (LTR) Support	Outbound / Tx only
OBFF Support	Yes
Address Translation Services (ATS) Support	Yes
ID Based Ordering (IDO) Signaling Support[f]	Yes
TLP Processing Hints (TPH) Support	Yes
TLP Prefix Support (Including PASID)	Yes
Lightweight Notifications (LN) Support	Yes

AXI Bridge Feature	Supported
Readiness Notifications (RN) Support	Yes
Peer-to-peer Support Across Local AXI Bus (Posted Can Always Make Progress)	Yes
Inbound Completion Re-ordering for Subordinate Read Requests With Same AXI ID	Yes
AXI5 AtomicOp Support	Yes
AXI5 Unique ID Indicator Support	Yes
AXI5 Read Data Chunking Support[g]	Yes
AXI5 Poison Signaling[h]	Yes[i]
AXI5 Interface Protection[j]	Yes[k]

a. AXI5 features not specified in this table are not supported.
b. For more information on QOS limitations, see Table 5-2 on page 398.
c. For more information, see "Zero-Byte Transfers Over the AXI Bridge (Flush Semantics)" on page 427.
d. For more information, see "AXI Subordinate Interface Sideband Signals" on page 416.
e. Except for MsgD (message with data).
f. DMA identifies and decodes IDO bit from the TLP bit stream and passes on the information to the Application through native client interfaces or AXI Bridge. But, IDO bit of the TLP Header cannot be explicitly set by the DMA.
g. For more information, see "AXI5 Read Data Chunking" on page 443.
h. For more information, see "AXI Sideband (Misc. Bus) Signals" on page 414.
i. This functionality is supported through a Synopsys-Specific implementation. It is not AXI5 compliant.
j. For more information, see "RAS Data Protection (RAS DP)" on page 164.
k. This functionality is supported through a Synopsys-Specific implementation. It is not AXI5 compliant.

Table 5-2 AXI Bridge Limitations

Normal Operation	Note
Outbound write burst with zero or partial (but contiguous) write strobes set.	There can be any number of beats with writestrobes set to 0 both at the beginning and end of the burst. Forthe first beat of the burst with non-zero write strobe, the following combinations of write strobes are allowed (using a 32-bit data bus example): 1000 / 1100 / 1110 / 1111. For the last beat of the burst with non-zero write strobe, the following combinations are allowed 0001/ 0011 / 0111 / 1111. All of the remaining intervening beats must have all of the write strobes set.
Vendor messages must not be decomposed.	For more information, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.
PCIe Advanced Error Reporting (AER) is not supported in the AXI bridge.	AER is only supported with respect to the native PCIe controller. Errors detected by the bridge are not reported as part of AER. For more information, see "Master AXI-to-PCIe Error Mapping" on page 433.
Forwarding of ECRC error signals in the received P/NP/CPL TLPs is not supported in the AXI Bridge.	The parameter ECRC_ERR_PASS_THROUGH is disabled for AXI bridge and the default value is 0.
Partial peer-to-peer support	You must implement completion processing for P2P traffic in your application logic. For more information, see "Peer-to-Peer Support (P2P)" on page 251.

Normal Operation	Note
ACE-Lite Limitations	See "ACE-Lite Support in PCIe Controller with AMBA Bridge" on page 445.
MsgD Decomposition not supported	The bridge does not decompose (or reassemble) messages that have a payload (MsgD). Inbound MsgD requests bigger than CC_MSTR_MTU cause data corruption. Outbound MsgD requests bigger than the programmed MTU value cause data corruption.
QOS Temporary Limitations	■ DMA: Embedded DMA controller not supported in dual-master configurations. ■ QoS Transaction Latency is guaranteed in the High Priority Master only if all the conditions in the following list are met: ❑ Outbound CPLs have infinite Header and Data credits. If this condition is not met, a low priority VC CPL can stall the high priority master. ❑ Only one outstanding NCBE is permitted at a time. Having a second or more outstanding NCBE requests stalls the AXI manager read channel. ❑ No CFG/IO transactions. These Non-Posted transactions can stall the AXI bridge master write channel.
Limited AXI subordinate Read Burst Length during 5-bit tag operation	If your configuration has CX_MAX_TAG >31 AND your application logic disables Extended Tag Support, ensure your application logic limits slv_arlen to the value given by the parameter CC_SLV_BURST_LEN_5BIT.

5.1.1.2 Supported Configurations

Table 5-3 provides supported controller configurations for the AXI bridge. All supported lanes, frequencies, and databus widths for the DWC PCIe controller are also supported for the AXI bridge. For the supported controller frequencies and datapath widths, see "Frequency, Speed, and Width Support" on page 36. Note that the PCIe controller datapath width is automatically derived and is not selectable.

Table 5-3 AXI Bridge Module - PCIe Controller Configurations Supported

	32-bit PCIe controller Internal databus width	64-bit PCIe controller Internal databus width	128-bit PCIe controller Internal databus width	256-bit PCIe controller Internal databus width	512-bit PCIe controller Internal databus width
AXI subordinate databus width	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit

	32-bit PCIe controller Internal databus width	64-bit PCIe controller Internal databus width	128-bit PCIe controller Internal databus width	256-bit PCIe controller Internal databus width	512-bit PCIe controller Internal databus width
AXI manager databus width	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit	32-bit 64-bit 128-bit 256-bit 512-bit
AXI dedicated DBI subordinate databus width	32-bit	32-bit	32-bit	32-bit	32-bit
AXI shared DBI databus width	supported[a]	supported[b]	supported[b]	supported[b]	supported[b]

a. For more information on Dedicated versus Shared DBI, see "Data Bus Interface (DBI) Access" on page 146.
b. Only the lower 32-bits of the databus is used by DBI. For more information, see "Access Limitations" on page 150.

For more information on supported DBI data/address sizes, see "Access Limitations" on page 150.

5.1.2 AXI Bridge Architecture

The following topics are discussed in this section:

- "Bridge Architecture Overview"
- "AXI Bridge Ordering" on page 408
- "AXI Bridge Initialization, Clocking and Reset" on page 414

5.1.2.1 Bridge Architecture Overview

This module contains AXI manager and subordinate protocol handlers, packet decomposer, and packet composers. The packet decomposers and composers support the segmentation and reassembly (subordinate only) of a PCIe transaction. The bridge module provides flexible buffering and tag management to facilitate seamless bridging between the controller and your application logic. It has a second optional subordinate (DBI) for accessing internal or local registers. For more information, see "Data Bus Interface (DBI) Access" on page 146.

Your application master request (at the AXI subordinate interface) never requests more than `CC_SLV_BURST_LEN*SLAVE_BUS_DATA_WIDTH/8` bytes and does not issue a request that crosses a 4K page boundary. Therefore, the corresponding response (CPL) composed from one or more CPLs that arrive in the PCIe wire is AXI compliant and does not need to be processed in a decomposer to enforce AXI page boundary and maximum packet size rules.

5.1.2.1.1 Decomposition

The bridge manages the differences between PCIe and AXI transfer sizes. The maximum payload is configured independently for AXI and PCIe systems. The differences between PCIe and AXI transfer size causes a split of a request and composition of a response. Automatic segmentation and reassembly of outbound (application to PCIe wire) packets is performed to satisfy PCI Express Max_Read_Request_Size

and Max_Payload_Size rules. When a decomposition rule is triggered, the outbound TLP is broken up into two or more smaller TLPs. When the TLP type is non-posted then additional PCIe TAGs are used from the tag pool. Additional entries in the outbound header buffer are also used. In a similar manner, automatic segmentation of inbound packets is performed.

5.1.2.1.2 Bridge Buffering

The bridge has additional buffering in addition to the "Receive Queues" on page 100 in the native controller. Table 5-4 identifies all the RAMs that are used in the AXI bridge. For more information on RAM requirements, see "RAM Requirements" on page 64.

Table 5-4 AXI Bridge RAMs

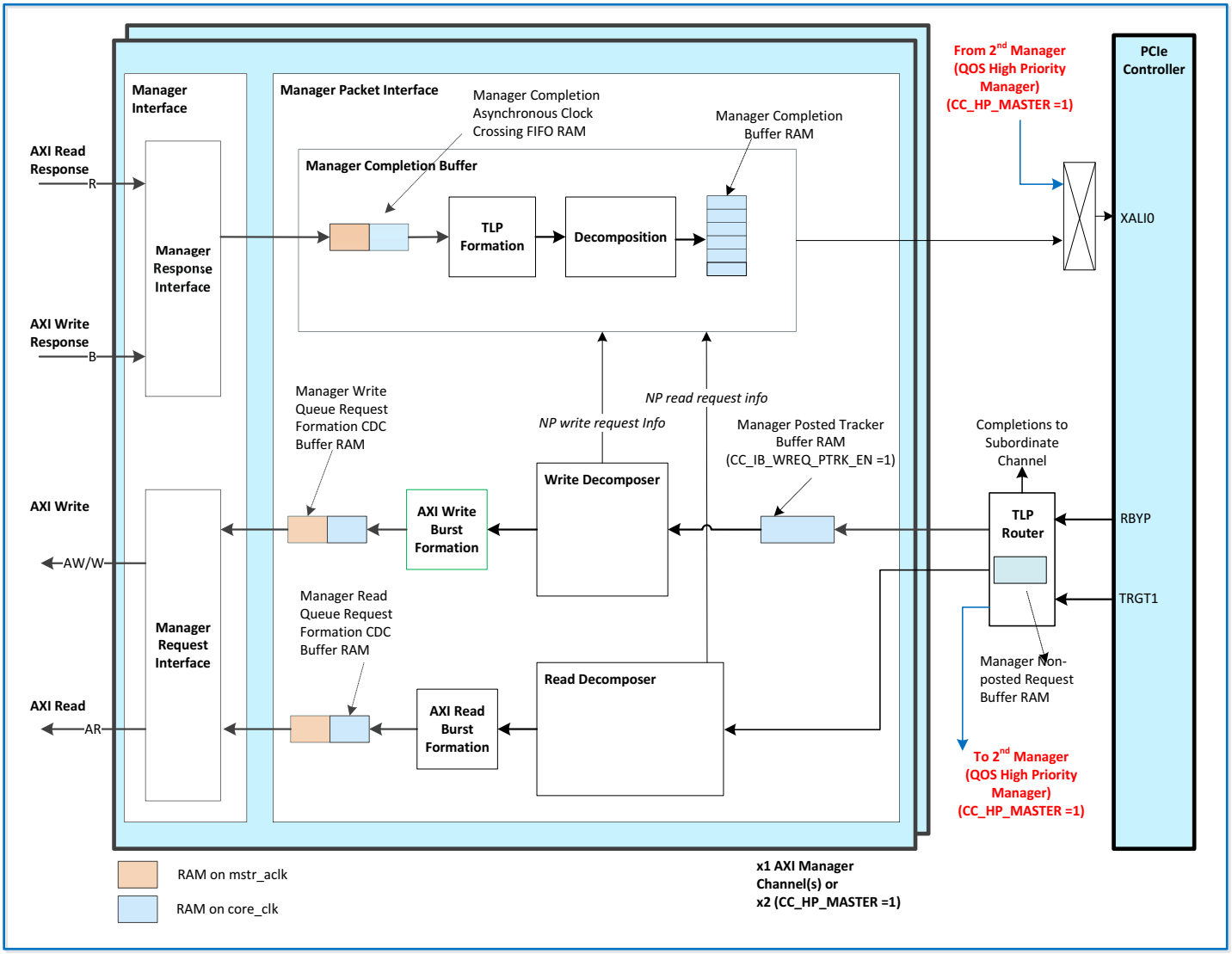
RAM Name	Clocks[a]	Signal Prefix
Master #1 RAMs		
Master Non-posted Request Buffer RAM[b]	Port A: core_clk Port B: core_clk	ib_rreq_ordr
Master Read Queue Request Formation CDC Buffer RAM ■ Exists when MSTR_CLK_DIFF_ENABLE =1	Port A: core_clk Port B: mstr_aclk	ib_rreq_c2a_cdc
Master Write Queue Request Formation CDC Buffer RAM ■ Exists when MSTR_CLK_DIFF_ENABLE =1	Port A: core_clk Port B: mstr_aclk	ib_wreq_c2a_cdc
Master Completion Asynchronous Clock Crossing FIFO RAM ■ Exists when MSTR_CLK_DIFF_ENABLE =1	Port A: mstr_aclk Port B: core_clk	ib_mcpl_a2c_cdc
Master Completion Buffer RAM ■ Buffers the master response before transmission. When it has collected Max_Payload_Size bytes, it transmits a completion. It does not wait until it receives the complete response. ■ Implemented as a store-and-forward segmented buffer with CX_REMOTE_MAX_TAG+1 segments of depth CX_MAX_MTU each. It is not cut-through.	Port A: core_clk Port B: core_clk	ib_mcpl_sb
Master Write Queue Posted Tracker Header RAM[c] ■ Buffers/offloads the posted write request data to avoid blocking the native controller Rx queue when the AXI fabric has not accepted enough of the write data from the bridge master. ■ Single-VC Configurations: Facilitates CPL/NP passing of blocked P when appropriate RO bit is 1. ■ Multi-VC Configurations: Facilitates CPL/NP passing of blocked P in another VC. ■ Multi-Master Configurations: Facilitates P passing of blocked P of another VC-group. ■ Exists when CC_IB_WREQ_PTRK_EN =1	Port A: core_clk Port B: core_clk	ib_wreq_ptrk_hdr
Master Write Queue Posted Tracker Data RAM[d] ■ Exists when CC_IB_WREQ_PTRK_EN =1 **Note:** See notes for Master Write Queue Poster Tracker Header RAM.	Port A: core_clk Port B: core_clk	ib_wreq_ptrk_data

RAM Name	Clocks[a]	Signal Prefix
Master Posted Scheduler Header RAM ■ Buffers Posted request header ■ Exists when CC_IB_SCHEDULER_EN	Port A: core_clk Port B: core_clk	ib_scheduler_p_hdr_ram
Master Posted Scheduler Header RAM ■ Buffers Posted request data ■ Exists when CC_IB_SCHEDULER_EN	Port A: core_clk Port B: core_clk	ib_scheduler_p_data_ram
Master Posted Scheduler Header RAM ■ Buffers Non-Posted request header ■ Exists when CC_IB_SCHEDULER_EN	Port A: core_clk Port B: core_clk	ib_scheduler_np_hdr_ram
Master Posted Scheduler Header RAM ■ Buffers Non-Posted request data ■ Exists when CC_IB_SCHEDULER_EN	Port A: core_clk Port B: core_clk	ib_scheduler_np_data_ram
Master #2 (High Priority) RAMs (CC_HP_MASTER =1)[e]		
Master #2 Read Queue Request Formation CDC Buffer RAM ■ Exists when MSTR_CLK_DIFF_ENABLE =1	Port A: core_clk Port B: mstr_aclk	ibh_rreq_c2a_cdc
Master #2 Write Queue Request Formation CDC Buffer RAM ■ Exists when MSTR_CLK_DIFF_ENABLE =1	Port A: core_clk Port B: mstr_aclk	ibh_wreq_c2a_cdc
Master #2 Completion Asynchronous Clock Crossing FIFO RAM ■ Exists when MSTR_CLK_DIFF_ENABLE =1	Port A: mstr_aclk Port B: core_clk	ibh_mcpl_a2c_cdc
Master #2 Completion Buffer RAM	Port A: core_clk Port B: core_clk	ibh_mcpl_sb
Master #2 Write Queue Posted Tracker Header RAM ■ Buffers/offloads the posted write request data to avoid blocking the native controller Rx queue when the AXI fabric has not accepted enough of the write data from the bridge master. ■ Facilitates CPL/NP passing of blocked P on same VC when appropriate RO bit is 1. ■ Facilitates CPL/NP/P passing of blocked P in another VC.	Port A: core_clk Port B: core_clk	ibh_wreq_ptrk_hdr
Master #2 Write Queue Posted Tracker Data RAM **Note:** See notes for Master #2 Write Queue Poster Tracker Header RAM.	Port A: core_clk Port B: core_clk	ibh_wreq_ptrk_data
Subordinate RAM		
Subordinate Non-Posted Write Set-Aside Buffer RAM ■ For peer-to-peer RP ordering rules and legacy IO transactions. ■ Enables offload of non-posted write requests from the subordinate AXI write channel. For more information, see Table 5-6.	Port A: slv_aclk Port B: slv_aclk	slv_npw_sab

RAM Name	Clocks[a]	Signal Prefix
Subordinate Wrap Response Re-order Buffer RAM ■ Exists when CC_SLV_WRAP_ENABLE =1	Port A: slv_aclk Port B: slv_aclk	slv_wrp_rsp_rob
Subordinate Non-Posted TLP Decomposer Header/Data FIFO RAM	Port A: slv_aclk Port B: core_clk	ob_npdcmp
Subordinate Posted TLP Decomposer Header FIFO RAM	Port A: slv_aclk Port B: core_clk	ob_pdcmp_hdr
Subordinate Posted TLP Decomposer Data FIFO RAM For PCIe link efficiency the AXI bridge must burst all its transmit data without gaps after the TX arbiter of the controller grants it access to the link. Therefore, there is some buffering (2*CC_SLV_MTU) in the AXI bridge for that purpose.	Port A: slv_aclk Port B: core_clk	ob_pdcmp_data
Subordinate Completion Composer Segmented Buffer RAM ■ Used for buffering of inbound responses before presentation to the bridge subordinate interface. ■ Performs packet reassembly caused by: ❑ Bridge-initiated packet segmentation on the original outbound request. ❑ Packet segmentation caused by the remote completer. ■ The subordinate response composer is implemented as a store-and-forward segmented buffer with CX_MAX_TAG +1 segments of depth CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN each. It is not cut-through. ■ When the original outbound request is decomposed into multiple request TLPs, then all of the individual CPLs must be received it forwards any data to the AXI subordinate interface. ■ Each of the CX_MAX_TAG +1 segments is used to reassemble the multiple CPLs for each decomposed outbound request, in the case where there are multiple outstanding requests from the AXI application with the same AXI ID.	Port A: core_clk Port B: slv_aclk	ob_ccmp_data

a. When the RAMs are external to the controller, you should use the RAM instantiation template in workspace/examples/pcie_iip_device.sv. This has the correct clock, reset, address, data, and control connections.
b. Not supported for Gen6 configured controller
c. Not supported for Gen6 configured controller
d. Not supported for Gen6 configured controller
e. All Master #2 RAMs are not supported for Gen6 configured controller

Figure 5-2 Single/Dual Master Channel Architecture (Gen5 and Below)



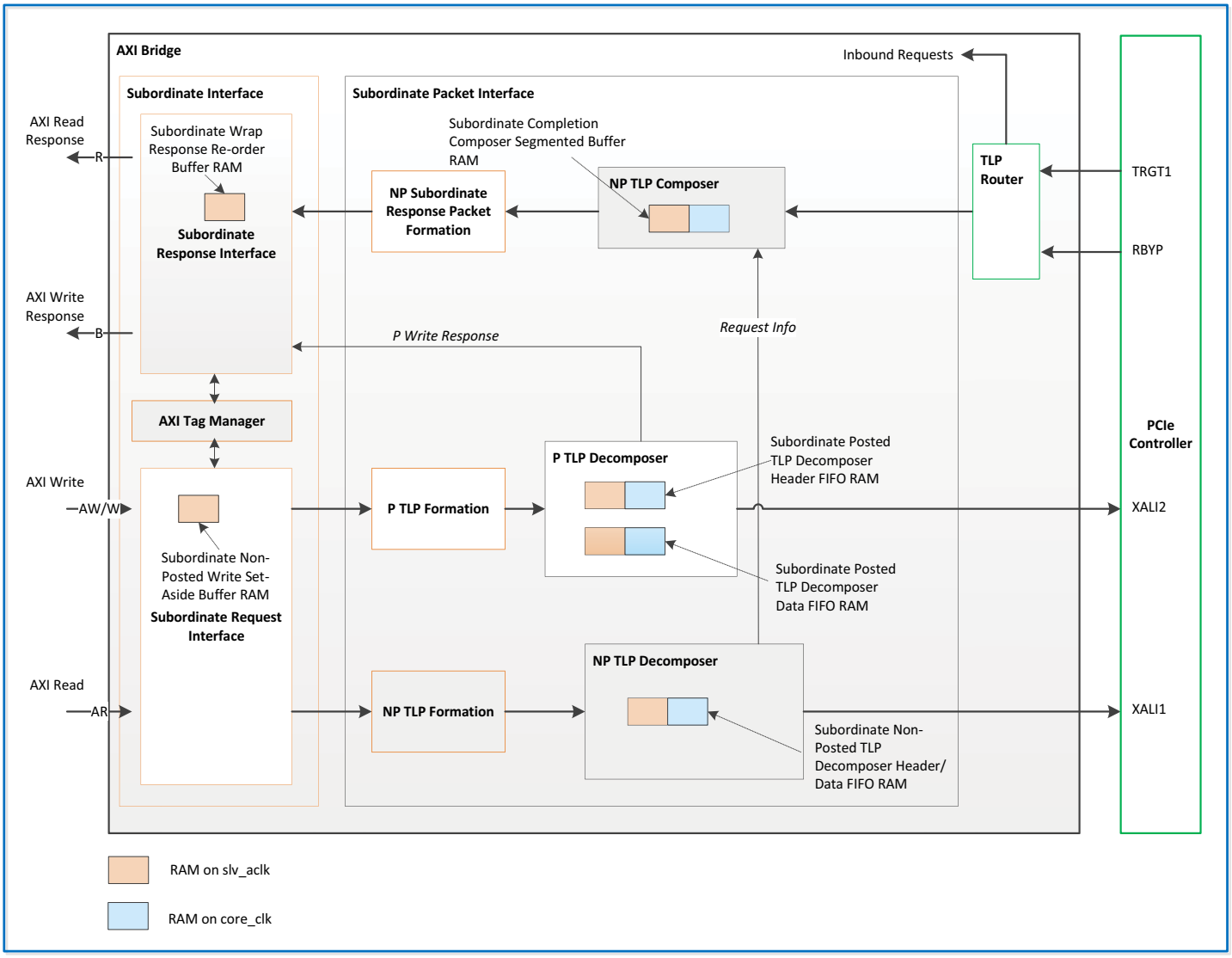
Note

QoS Dual-Master Architecture (CC_HP_MASTER =1, CX_NVC >1)[a]

- To support QoS for inbound traffic, the controller implements two AXI manager traffic paths; a high priority path (Master #2) and a low priority path (Master #1). The highest numbered VC (#CX_NVC-1) is assigned to the high priority master path. The remaining VCs are assigned to the low priority path. No TLP ordering dependency is enforced between the two paths. A known maximum latency and a known minimum bandwidth is guaranteed for high priority inbound traffic. A CPL with RO is always allowed to pass a Posted TLP on the same VC-mapped master.
- The code in this release is not for unapproved tape-out. Contact Synopsys support through SolvNetPlus for access to an unlocking key for this feature.

a. This Note is not applicable for Gen6 configured controller

Figure 5-3 Subordinate Channel Architecture (Gen5 and Below)



Note

The AXI Bridge is Not Double Buffering the RX Data

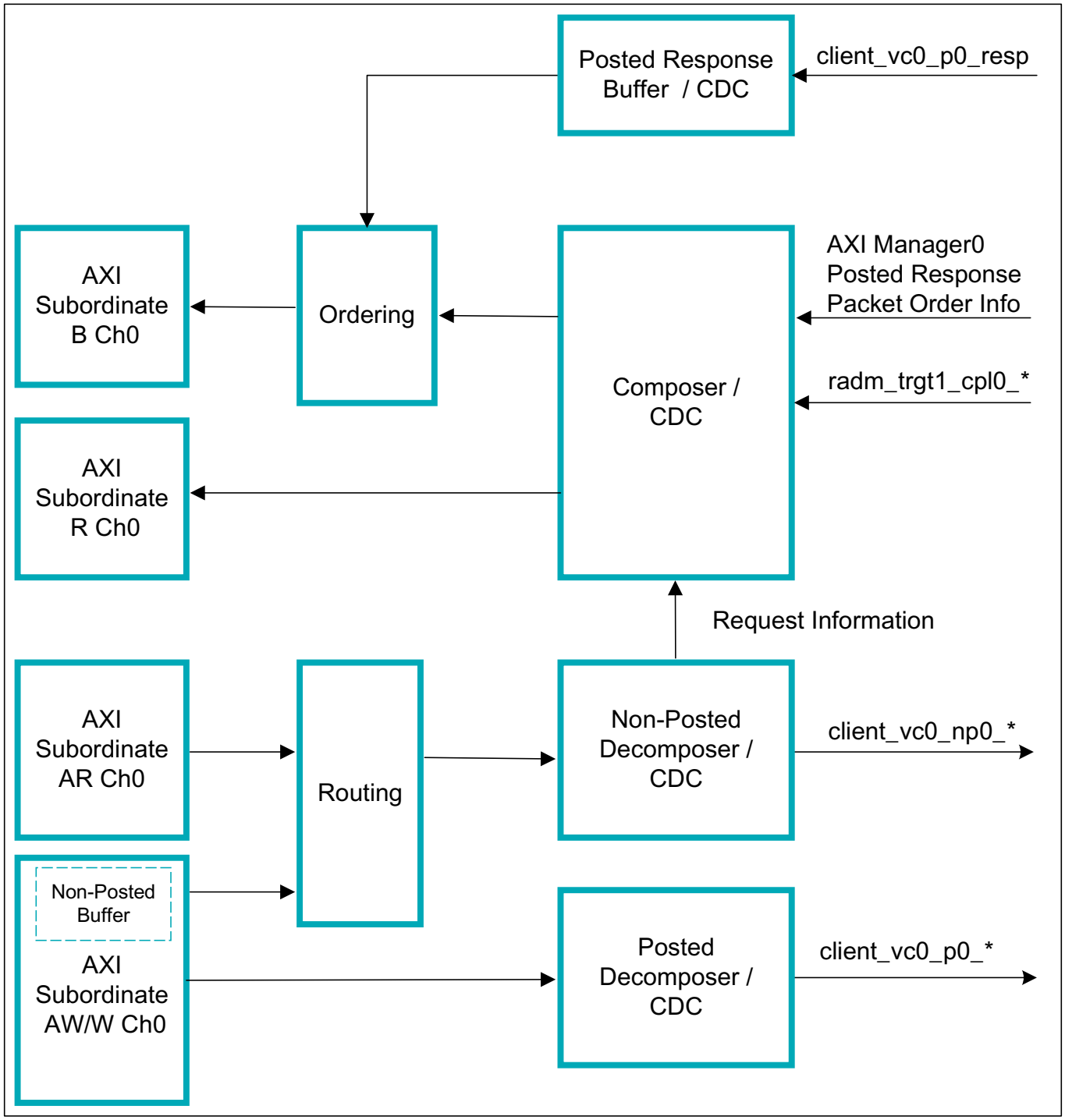
- The completion queue in the native controller is normally in bypass mode, so there is no issue of double buffering. The exception to this is for configurations with a 256-bit[a] internal native controller datapath or when you want to obey CPL must not pass P producer-consumer ordering.
- The received non-posted requests are pre-charged into a small buffer for performance reasons.
- The received posted requests are not buffered in the bridge. If the through-put on the AXI manager is larger than the native controller, idle cycles might be inserted in the AXI burst.

The AXI Bridge is Not Double Buffering the TX Data

- There is no TX buffering in the native controller, so there is no issue of double buffering.
- For PCIe link efficiency the AXI bridge must burst all its transmit data without gaps after the TX arbiter of the controller grants it access to the link. Therefore, there is some buffering (2x CC_SLV_MTU) in the AXI bridge for that purpose.

a. The controller has only one RX completion interface. Therefore, RX buffering is required in the controller for serializing TLPs that arrive in the same clock cycle which can happen with small TLPs in the 256-bit controller. It is not possible to remove the subordinate response composer buffer because it is still needed for completion re-ordering, when completions corresponding to outbound requests with the same AXI ID come back out of order. For more information, see "Completion Re-ordering (Subordinate)" on page 411.

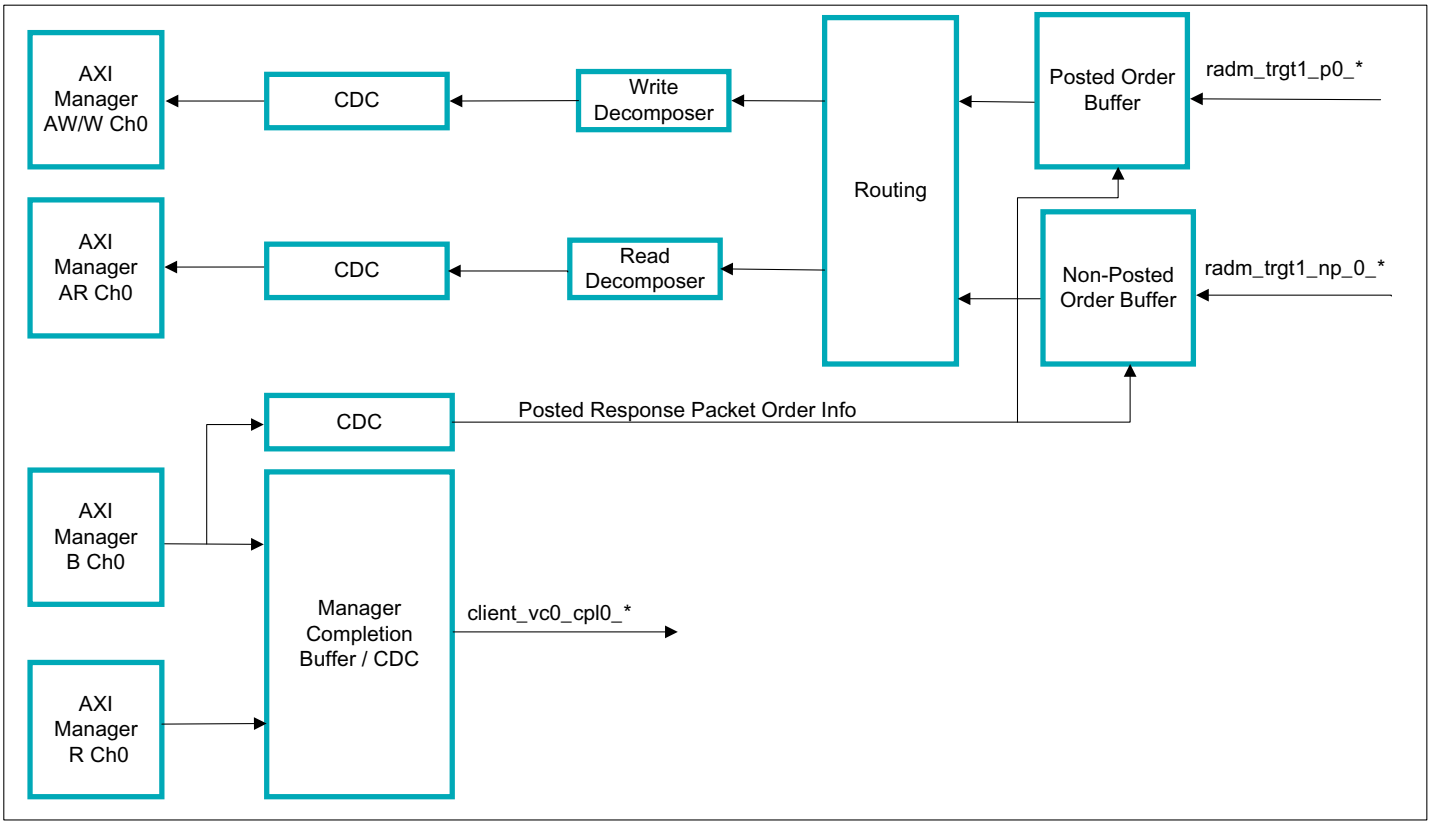
Figure 5-4 AXI Subordinate (Single Channel) for Gen6 Configurations



- In the wire to application direction, Posted and Non-Posted TLPs are routed to independent scheduler modules. The Posted scheduler enables Posted TLPs to overtake Posted TLPs if the preceding TLP cannot make forward progress and subsequent TLPs are weakly ordered or belong to a different VC or IDO stream. The Non-Posted scheduler enforces the PCIe strongly ordered rule "Non-Posted TLPs must not pass Posted TLPs".
  - TLPs read from the respective schedulers are routed to AXI Manager Write Channels or AXI Manager Read Channels depending on ordering requirements and the FC-type.

    For example, Non-Posted TLPs with data must be routed to the an AXI Write Channel, relaxed order P/NP TLPs can be routed to any AXI Manager Channel. Strongly ordered Posted/Non-Posted TLPs must be routed to the same AXI Write/Read Channel.
  - Prior to Posted and Non-Posted TLPs being sent to the AXI Manager ports, TLPs are decomposed depending on the configured maximum Manager burst size. The AXI Manager Write and Read Channel interface modules handle the TLP to AXI Write/Read protocol conversion.
- In the application to wire direction, the AXI Read Channel and AXI Write Channel Responses to AXI Non-Posted Write Requests are accumulated in the Manager Completion Buffer and sent on the wire in chunks <= *MaxPayloadSize*. Write Response Order Information is returned from the AXI Manager Response Channel to the Posted and Non-Posted schedulers to facilitate ordering of TLPs.

Figure 5-5 AXI Manager Architecture for Gen6 Configuration



- In the application to wire direction, AXI Subordinate write and read requests are converted to Posted and Non-Posted TLPs and decomposed before being sent on the wire. Non-Posted AXI Writes and AXI Reads are routed to the Non-Posted decomposer and decomposed depending on *MaxReadRequestSize*. AXI Writes are routed directly to the Posted Decomposer and decomposed on *MaxPayLoadSize*.

- For Non-Posted TLPs, the associated request information is sent to the completion Composer to enable the Composer to reserve buffering for the returning completions and to re-order completions associated with same AXI ID read requests. In addition, the AXI Manager interfaces return Posted order information to enable the Composer to enforce the PCIe strongly ordered rule "Completion TLPs must not pass Posted TLPs".
- If dual AXI Channels are required to improve throughput and efficiency, then the single channel infrastructure is duplicated. To optimize the composer buffer sizes the number of non-posted outstanding read requests associated with each Composer can be reduced depending on the traffic profile.
- To ensure an AXI Read cannot pass an AXI Write, AXI B Channel responses are generated at the point of serialization, that is, when an AXI Posted TLP is sent on the wire and cannot be overtaken by a Non-Posted TLP. The returning posted write responses and the responses to Non-Posted Write Request, if associated with the same ID, are ordered before being sent on the AXI B Response Channel.

5.1.2.2 AXI Bridge Ordering

In this section, the following ordering topics are discussed:

- "Overview"
- "Completion Re-ordering (Subordinate)" on page 411
- "Decomposed Read Request Throttling (Subordinate)" on page 412
- "Write Response Re-ordering (Subordinate)" on page 412
- "Optional Serialization of AXI subordinate Non-posted Requests" on page 413

5.1.2.2.1 Overview

You should first familiarize yourself with "PCIe Ordering Rules" on page 704. For this discussion, it is assumed that the native controller's queue architecture is configured for PCI Express rules, and that the completion (CPL) queue is in store-and-forward mode. Table 5-5 and Table 5-6 detail the AXI bridges compliance with the PCIe ordering rules.

Table 5-5 AXI Inbound Adherence to PCIe Ordering Rules

No.	Description (Per VC)	Obeyed	Reason/Notes
1	P must be allowed to pass a previously-issued NP	Yes	Separate inbound P and NP request paths in bridge. ■ This condition is true for all NP requests in Gen6 configured controller ■ This condition is not true for NP Wr requests in Gen5 (and below) configured controller
2	P must not pass a previously-issued P, unless the later P's RO bit is 1[a]	Yes[b]	Single inbound P request path in bridge[c].
3	P can optionally pass a previously-issued CPL.	Yes	Ordering between subordinate (CPLs) and master (P requests) only enforced for rule (5).
4	CPL must be allowed to pass a previously-issued NP	Yes	No ordering enforced between subordinate (CPLs) and master (NP requests). Separate CPL and request paths in bridge.

No.	Description (Per VC)	Obeyed	Reason/Notes
5	CPL must not pass a previously-issued P, unless the CPL's RO bit is 1	Yes[d]	CPL ordering is Per-VC in RADM; To enable a CPL to bypass a previously issued P from different VC in the AXI bridge, you must enable the Posted Tracker (CC_IB_WREQ_PTRK_EN) or the Order buffer (CC_ORDRB_EN). To enable a CPL with RO =1 to bypass[e] a previously issued P in the same VC, you must enable the Posted Tracker (CC_IB_WREQ_PTRK_EN) or the Order buffer (CC_ORDRB_EN). **Note**: When you place the CPL queue in bypass mode, this rule is violated.
6	CPL can optionally pass a previously-issued CPL, when the transaction[f] IDs are different	Yes	Passing not supported (single inbound CPL path in bridge), but this is optional, so rule is not violated. When Order Buffer feature is enabled, IDO CPL passing is supported.
7	NP can optionally pass a previously-issued NP	Yes	Passing not supported (single inbound NP request path in bridge), but this is optional, so rule is not violated. Order Buffer allows this, when the LTIM imposes out of order NP address translation. When AXI Inbound Posted ID Management is enabled, Read requests can pass Read requests of different VCs.
8	NP can optionally pass a previously-issued CPL	Yes	No ordering enforced between subordinate (CPLs) and master (NP requests). Separate inbound CPL and inbound request paths in bridge.
9	NP must not pass a previously-issued P, unless the NP's RO bit is 1 (in the case of NP writes only)	Yes	The AXI protocol does not support ordering between channels, and NP reads can pass P on your AXI bus fabric. This can result in a violation of rule (9) when the read overtakes a P that is going to the same address. Therefore, the AXI bridge does not issue any NP (with RO bit set to zero) request until all outstanding P writes reach their destination. It does this by waiting for the write responses on the B channel. The Order Buffer allows the NP writes with RO bit set to one to pass Posted. The Order Buffer also allows NP to pass a Posted if both transactions belong to different VC (Virtual Channel) or to do different IDO (ID based Ordering).Enabling the CC_ORDRB_EN parameter disables the AX_MSTR_ORDR_P_EVENT_SEL field in the AMBA_ORDERING_CTRL_OFF register, disabling the choice to select AXI manager AW/W channels as an ordering point. When CC_IB_P_ID_MNG_EN =1, the AXI bridge issues NP Writes if there are outstanding Posted Writes for different VCs.

a. A non-MSI P request can pass previously-received MSI P request by the bridge. When CC_MSIGIC_EN =1, the MSIG_PCIE_ORDR register defines the ordering.
b. In general, the AXI bridge does not consider the RO and IDO bits. It does consider the RO bit for NP writes at the master. When the Order buffer is enabled, the AXI bridge allows Posted to pass Posted of different IDO, and the VC or RO attribute is set. When CC_IB_P_ID_MNG_EN =1, the AXI bridge allows Posted to pass Posted when their VC/IDO/DMA/RO status allows.
c. When CX_MSI_CTRL_EN =1, the "iMSI-RX: Integrated MSI Receiver [AXI Bridge]" on page 211, upon detection of an MSI MWr generates the msi_ctrl_int interrupt, after the B'last event of an earlier Posted transaction. This guarantees P ordering. This is valid for MSI-GIC also.
d. Not obeyed when completions are in bypass.
e. This partial support refers to the data path in the AXI bridge. RADM allows a RO CPL to pass a Posted on the same VC on some conditions.
f. Combination of requester ID and tag.

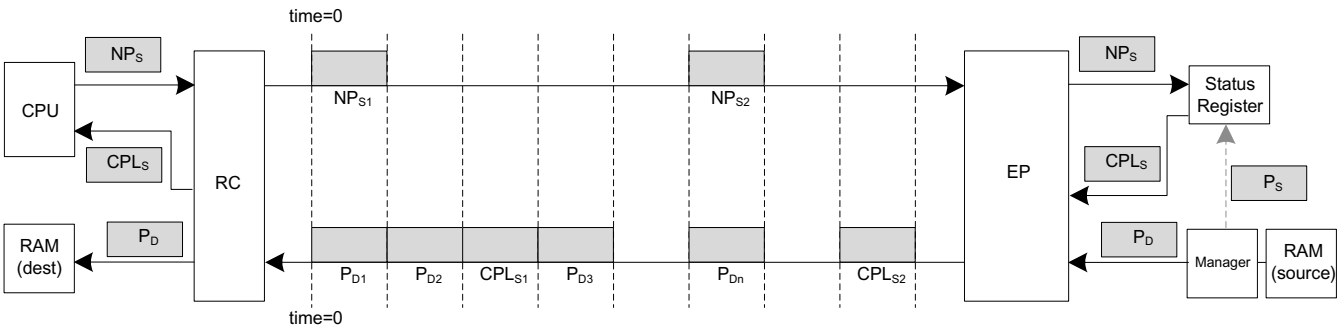
Table 5-6 AXI Outbound Adherence to PCIe Ordering Rules

No.	Description (Per VC)	Obeyed	Reason/Notes
1	P must be allowed to pass a previously-issued NP[a]	Yes	Separate outbound P and NP request paths in bridge. **NP Writes**: To enable P writes to pass NP writes (and avoid deadlocks), a Subordinate Non-Posted Write Set-Aside Buffer is used to offload the NP writes from the AXI Write Channel when NP reads are stalled. Offloading NP writes thereby avoids blocking of P on the AXI Write Channel. You configure the maximum number of offloaded NP writes using the `CC_SLV_NUM_OUTSTND_CPU_WR_REQ` parameter.
2	P must not pass a previously-issued P, unless the later P's RO bit is 1	Yes[b]	Single outbound P request path in bridge.
3	P can optionally pass a previously-issued CPL	Yes	Ordering between subordinate (CPLs) and master (P requests) only enforced for rule (5)
4	CPL must be allowed to pass a previously-issued NP	Yes	No ordering enforced between subordinate (CPLs) and master (NP). Separate bridge queues used.
5	CPL must not pass a previously-issued P, unless the CPL's RO bit is 1	Yes	Separate bridge queues are used; there is no ordering enforced between master (CPLs) and subordinate (P requests). However, the AXI subordinate interface delays the generation of the P response until the native controller has serialized and scheduled the P for transmission. This enforces producer-consumer ordering rules because the CPL for a polling-read from the local producer flag does not pass out the last consumer data write. For more information, see "Producer-Consumer Example" on page 411.
6	CPL can optionally pass a previously-issued CPL, when the transaction[c] IDs are different	Yes	Passing not supported (single outbound CPL path in bridge), but this is optional, so rule is obeyed.
7	NP can optionally pass a previously-issued NP	Yes	Passing not supported (single outbound NP request path in bridge), but this is optional, so rule is obeyed.
8	NP can optionally pass a previously-issued CPL	Yes	No ordering enforced between master (CPLs) and subordinate (NP requests). Separate outbound CPL and outbound request paths in bridge.

No.	Description (Per VC)	Obeyed	Reason/Notes
9	NP must not pass a previously-issued P, unless the NP's RO bit is 1or (in the case of NP writes only)	Yes	Separate bridge queues are used; there is no ordering enforced between NP and P requests. However, the AXI subordinate interface delays the generation of the P response until the native controller has serialized and scheduled the P for transmission. This enforces rule (9) for your AXI manager which does not generate an NP request until the P request has been transmitted.

a. NP read requests include MRd and DB (SHARED_DBI_ENABLED=1) read requests. NP writes requests include IO, Cfg and DB (SHARED_DBI_ENABLED=1) write requests.
b. The AXI bridge does not consider the RO and IDO bits.
c. Combination of requester ID and tag.

Figure 5-6 Producer-Consumer Example



Producer: The manager in the EP is taking data from the local source RAM and writing it into the destination RAM in the RC. This posted data is indicated as P_D. The manager writes to the 'finished' flag in the status register after it has written the last P_D.

Consumer: The CPU in the RC is polling the status register in the EP, checking if the manager is finished producing the data. These polling/read requests are indicated as NP_S. The corresponding completion is indicated as CPL_S.

Producer-Consumer Model: The CPL_S must not pass any P_D, or else the CPU starts to consume the data before it is fully produced. Therefore the EP, any intermediate SW, and RC queues must obey this rule.
The rule does not need to be applied to P TLPs and CPLs that are not part of this Producer-Consumer sequence, for example, RC CPU CFG reads to unrelated functions in the EP. You should set the relaxed ordering (RO) bit for these requests to '1'.

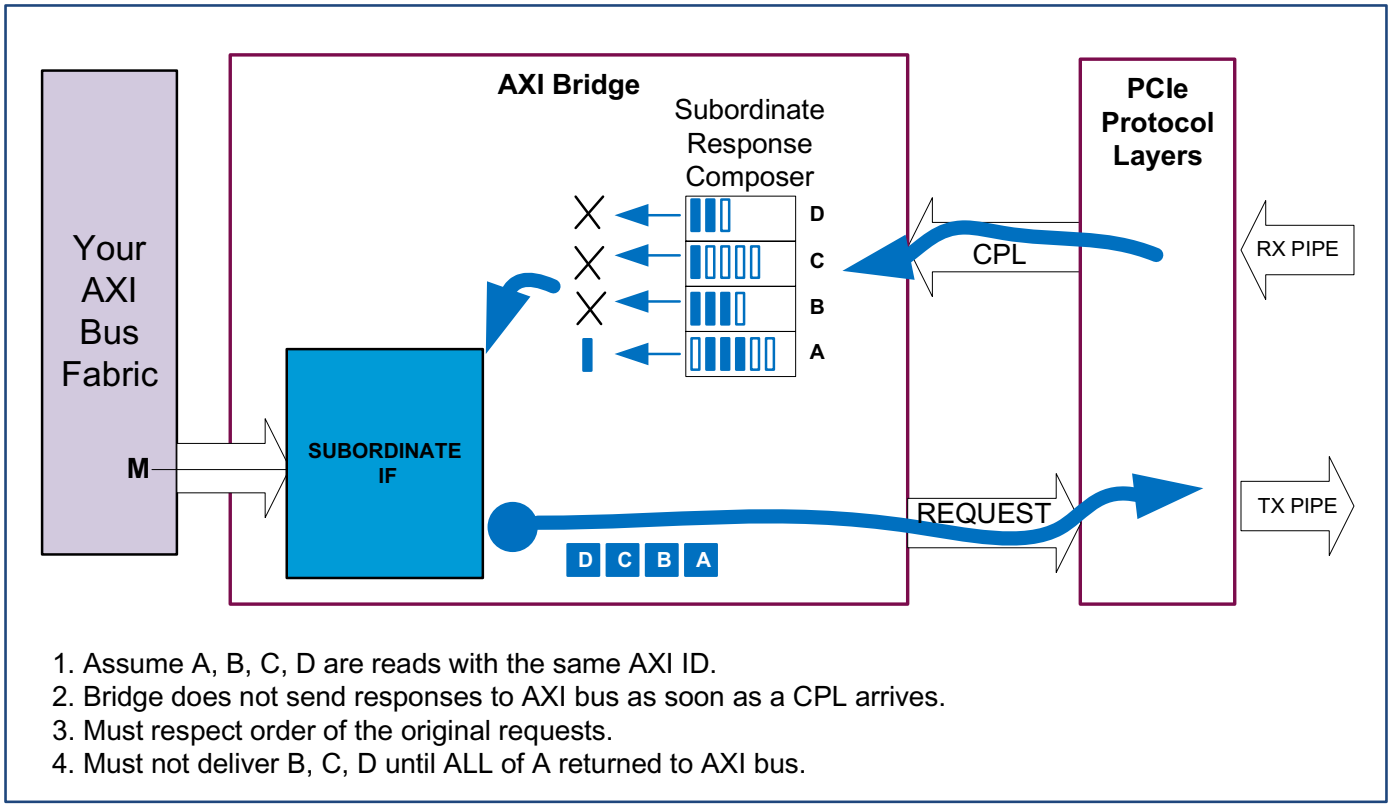
If there is an element in your system that is not enforcing the "CPL must not pass P" rule (for example, if the CPL receive queue in the RC is in bypass mode), you must use an alternative method for communicating status between a producer and a consumer; use interrupts or place the data being read-from/written-to in a different location in your PCIe system.

5.1.2.2.2 Completion Re-ordering (Subordinate)

When your application sends[1] multiple NP requests with the same AXI ID to the PCIe wire, the PCI Express protocol permits completions associated with the different request TLPs to pass each other and be returned out-of-order. When these completions (corresponding to NP requests with the same AXI ID) are received from the PCIe wire out-of-order, the AXI bridge subordinate response composer re-orders them according to the order of the originating NP requests.

1. Without decomposition.

Figure 5-7 Completion Re-ordering (AXI)



5.1.2.2.3 Decomposed Read Request Throttling (Subordinate)

An outbound read request at the AXI bridge subordinate interface is sometimes decomposed (see "Outbound Decomposition" on page 435) into multiple PCIe TLP requests (these are called subTLPs), each with a different PCIe tag. The PCIe protocol allows CPLs associated with the different TLPs to pass each other and be returned out-of-order. The AXI bridge subordinate response composer supports re-ordering of the decomposed CPLs.

If any of the decomposed CPLs signal an error condition (CPL timeout, Completer Abort, and so on), the subordinate response matches error given by the first chronological errored CPL received from the remote device.

5.1.2.2.4 Write Response Re-ordering (Subordinate)

For write requests with the same ID, the controller returns AXI writes responses in order. The PCIe AXI subordinate re-orders out-of-order P and NP write responses. The AXI subordinate allocates resources to buffer P write responses while NP write responses are outstanding.

Note For more information on ordering, see "Advanced Information: Advanced Ordering Information" on page 703 and "DMA Ordering" on page 505.

5.1.2.2.5 Optional Serialization of AXI subordinate Non-posted Requests

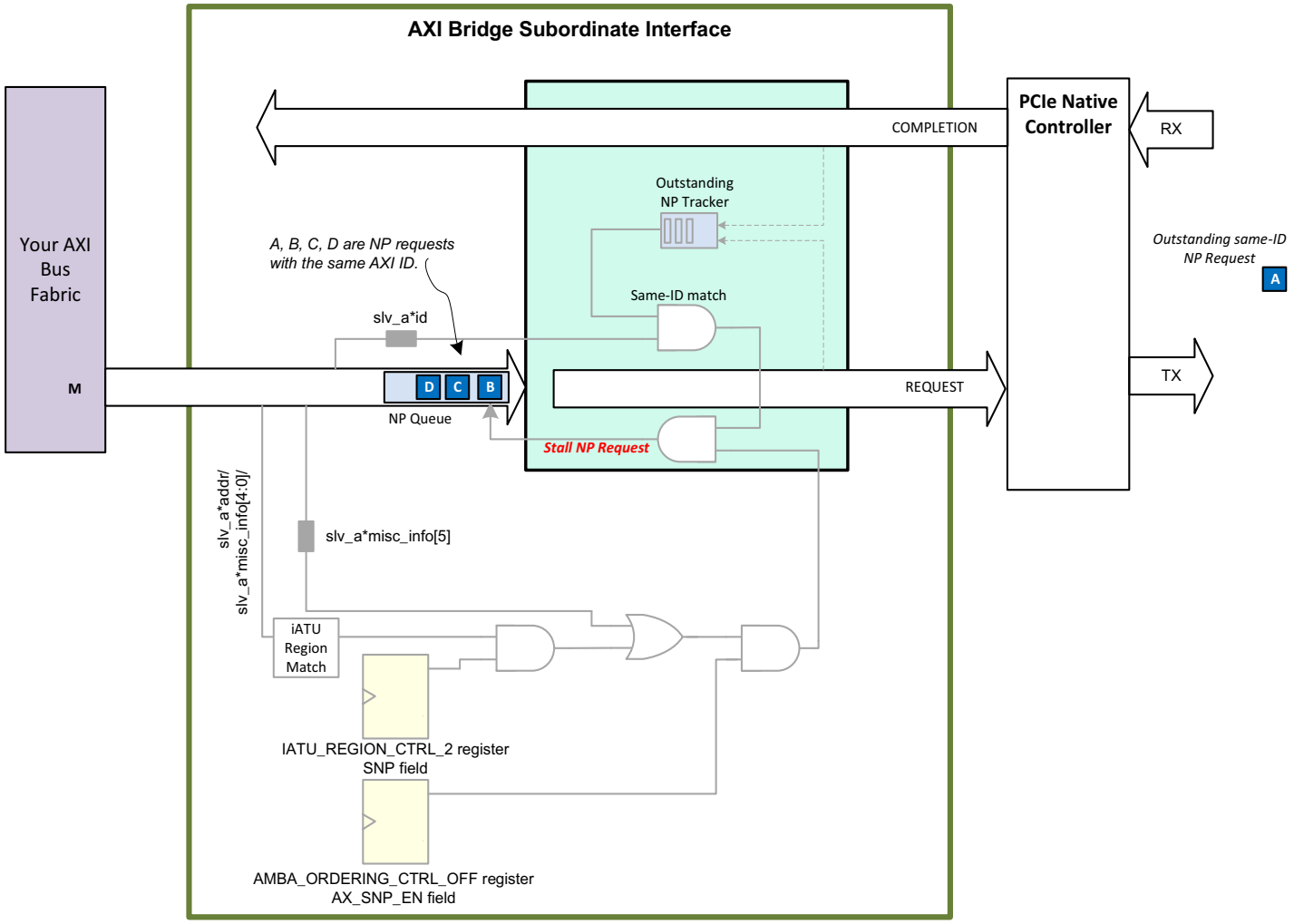
PCIe NP requests (MEM/CFG/IO) are weakly ordered; NP can pass NP. To guarantee the arrival order of same-ID NP requests over PCIe, the controller optionally serializes NP requests by ensuring that there is only one same-ID NP request outstanding on the wire. This is not a PCIe requirement; but a requirement for CFG and IO requests in some AXI systems. When you enable serialization, the bridge tracks pending same-ID NP requests on the wire. If the serialization control is asserted for a new same-ID NP request at the AXI, the controller stalls that request if there is an outstanding same-ID request on the wire. You can use either of the following methods to control serialization as indicated in Figure 5-6 on page 411.

- AXI subordinate Sideband Bus Signaling.

  Assert the serialization bit `slv_awmisc_info[5]/slv_armisc_info[5]` on a per-transaction basis.
- "Internal Address Translation Unit (iATU)" on page 220.

  Programming an outbound iATU region to map a specified address range for NP requests. When the address falls within the iATU address range and you have set the `SNP` field in `IATU_REGION_CTRL_2`, the subordinate serializes the NP request. The bridge ignores `slv_awmisc_info[5]/ slv_armisc_info[5]`when the iATU is enabled.

Figure 5-8 Serialization of NP Subordinate Requests



5.1.2.3 AXI Bridge Initialization, Clocking and Reset

This section indicates AXI aspects of initialization, clocking, and reset that you must be aware of when you are integrating the controller with the AXI bridge into your SoC.

5.1.2.3.1 Initialization

However, if your application AXI manager issues outbound requests to the AXI bridge subordinate interface before the PCIe link is operational, the controller starts a flush timer. The timeout value of this timer is set in the `AMBA_LINK_TIMEOUT_OFF` register. If the timer times out before the PCIe link is operational, the bridge TX request queues are flushed. During this flush process, pending NP subordinate requests are answered with a CPL timeout error response which is determined by the `AMBA_ERROR_RESPONSE_GLOBAL` field in `AXI_ERROR_RESPONSE_DEFAULT_OFF`; the controller also sets the CPL timeout bit `slv_rmisc_info[0]/slv_bmisc_info[0]`. The flush process is similar to the process described in "Hot Reset" on page 97, except that the bridge is only flushed and not reset.

5.1.2.3.2 Low-Power Behavior

The AXI bridge module provides clock gating support as described in "Clock Gating" on page 75. Even if you do not gate the AXI bridge clocks, the bridge stops forwarding and receiving any data when your clock generation logic has turned off `core_clk`.

5.1.2.3.3 Reset

Each of the four clock signals previously mentioned has a corresponding reset signal — `mstr_aresetn`, `slv_aresetn`, `dbislv_aresetn` and `core_rst_n`.

Caution It is recommended that your application asserts the controller, AXI bridge, and application logic reset signals according to the procedure outlined in "Hot Reset" on page 97. This ensures that there are no orphaned requests or completions within the system, that might cause the AXI bus to hang.

Each bridge reset signal can be asserted asynchronously by your application but must be de-asserted synchronously relative to its corresponding clock. Only one reset cycle is required to reset the logic associated with the corresponding clock. For more information, see "Reset Requirements" on page 90.

5.1.2.3.4 Clocking

For more information on clocking, see "AXI Bridge Clocking" on page 74.

5.1.3 AXI Bus Operations

5.1.3.1 AXI Sideband (Misc. Bus) Signals

An AXI subsystem can only convey data, address and read/write information. It has no way to propagate PCIe concepts such as TLP type, posted/non-posted, function number, TLP attributes and so on. Therefore, some method is required to map these PCIe concepts between AXI and PCIe. Two options are available:

- **Software option**

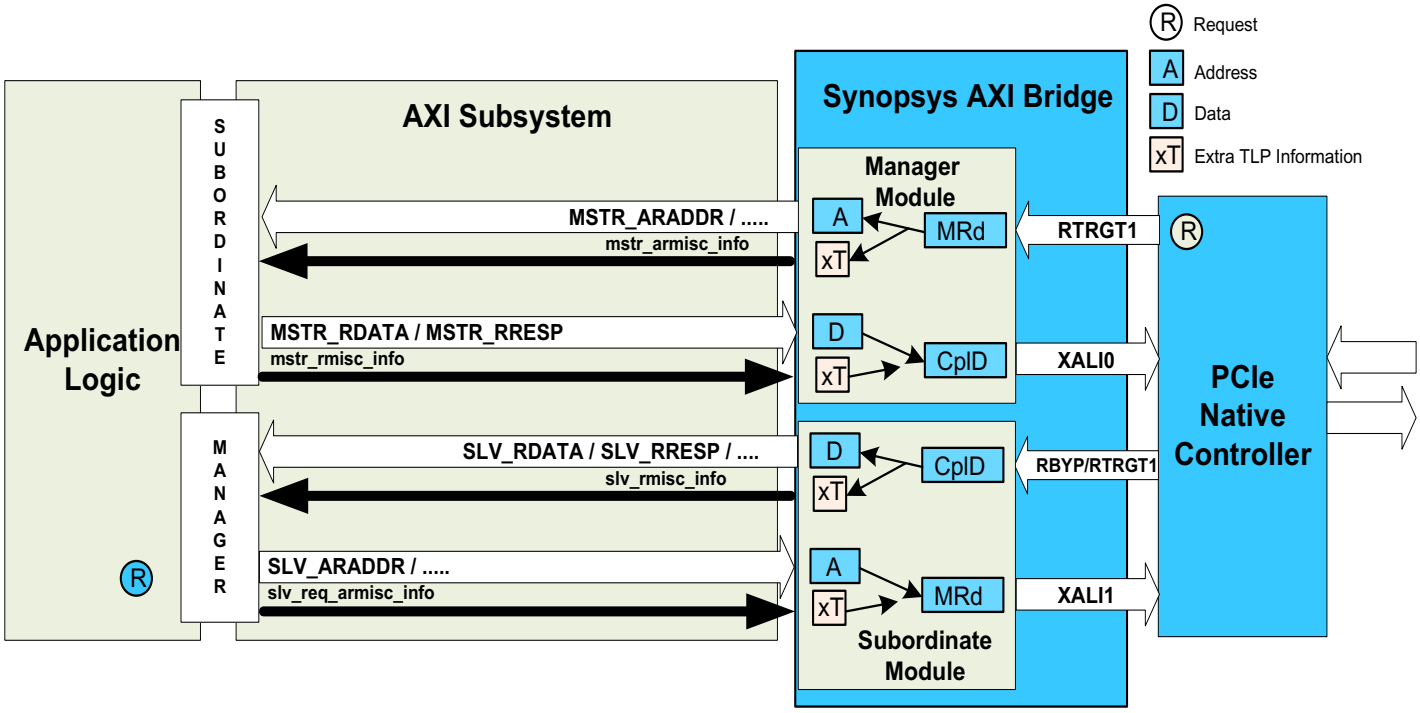
  Use the "Internal Address Translation Unit (iATU)" on page 220 to map outbound AXI transactions from different AXI address regions to particular PCIe address regions and TLP types. For example, map all AXI transactions in the region 0x0010FD00 - 0x00FFFFFF to CFG TLPs in the PCIe address region 0xD010FD00 - 0xD0FFFFFF

- **Hardware option**

  Extend your AXI fabric controller with sideband buses. Extra AXI control signals that are not part of the AXI signal specification are required by the PCIe AXI bridge to support the signaling of information (other than the TLP address and data payload) to and from your application. Figure 5-9 and Figure 5-10 show these `*misc_info`[1] signals for read and write transactions respectively. These sideband buses are also called the misc buses. For more information on each sideband signal, see "Signal Interfaces" on page 355.

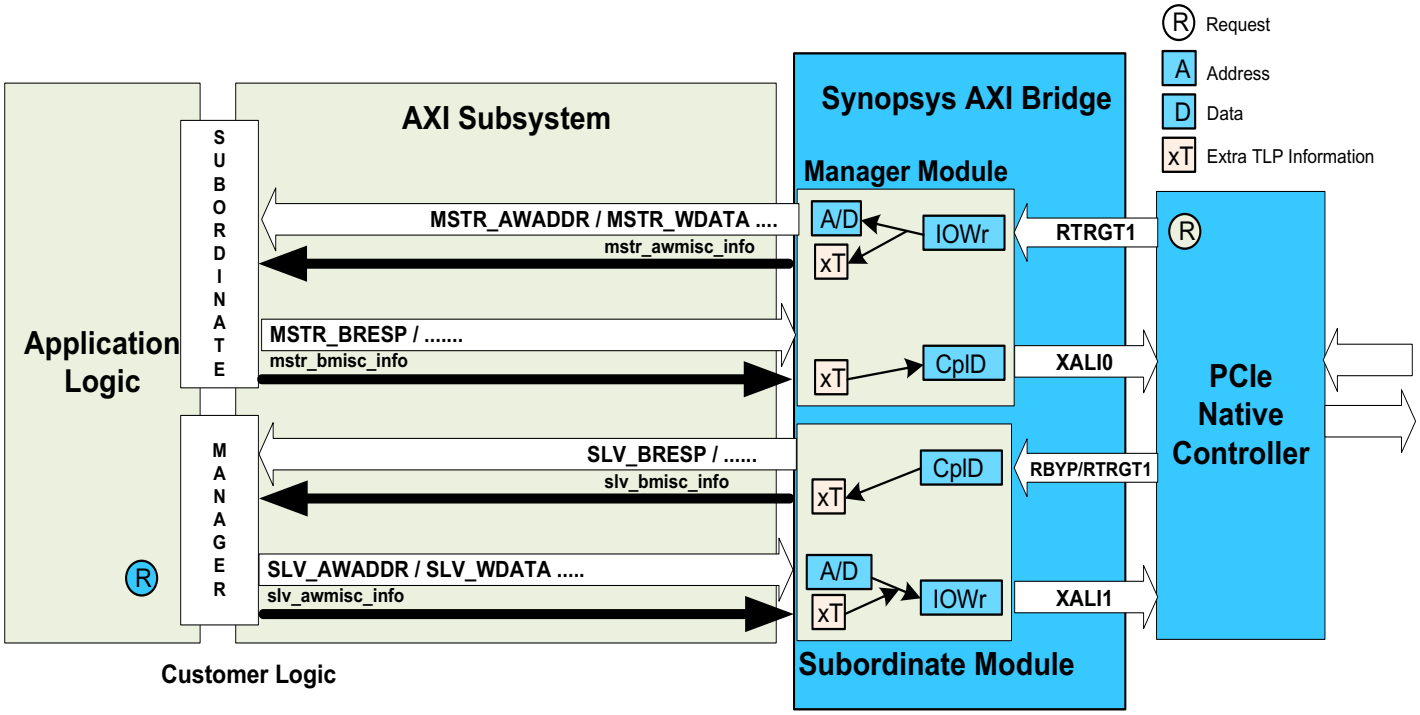
Attention The timing of the driving/sampling of the sideband (misc. bus) signals by your application logic must correspond to AXI bus timing. For example, `slv_awmisc_info` should be thought of as an extension of the AXI write address channel (AW) address/control buses in regard to its timing, and `mstr_rmisc_info` should as an extension of the read response channel (R).

Figure 5-9 AXI Sideband Signal Usage (MRd Example)



1. Renamed as *user in AXI4

Figure 5-10 AXI Sideband Signal Usage (IOWr Example)



5.1.3.1.1 AXI Subordinate Interface Sideband Signals

Your application master drives the subordinate request sideband input bus (`slv_armisc_info`[1], `slv_awmisc_info`, or `slv_wmisc_info_ep`, or `slv_wmisc_info_silentDrop`), and the bridge drives the subordinate response sideband output bus (`slv_rmisc_info` or `slv_bmisc_info`).

Subordinate request sideband input bus use-cases (`slv_armisc_info`, `slv_awmisc_info`, `slv_wmisc_info_ep`, `slv_wmisc_info_silentDrop`):

- An example of subordinate request sideband input information is the TLP header field information TYPE/IDO/EP[2]/RO/NS/TC/AT/MSG code/ function number.
- Another example is the flag indicating when the transaction is a DBI access (DBI slave shared mode only).
- Sideband input signal `slv_wmisc_info_ep` can be used for poisoning write requests that have been started on the AXI subordinate interface but should be discarded by the link partner. When `slv_wmisc_info_ep` is asserted during a data burst, the poison bit (EP) is set on the corresponding TLP header. If a data burst results in multiple decomposed TLPs, TLPs that have been completely processed before `slv_wmisc_info_ep` is asserted, are not marked as poisoned.

  If `slv_wmisc_info_ep` is asserted on a leading zero-byte beat of write request, the TLP with the trailing data has the EP bit set. Asserting `slv_wmisc_info_ep` on a trailing zero-byte beat does not set the EP bit in the TLP with the preceding data because the AXI bridge drops zero-byte write requests.

1. Renamed as slv_aruser and slv_awuser in AXI4
2. If slv_awmisc_info[6] is '1', the poison bit of all the TLPs of the transfer is set. To invalidate a part of the transfer during data burst, assert slv_wmisc_info_ep.

- Sidband input `slv_wmisc_info_silentDrop` can be used to drop write requests that have been started on the AXI subordinate interface (that is, prevent these write requests from being transferred to the link partner). When `slv_wmisc_info_silentDrop` is asserted during a data burst, the corresponding TLP header is dropped. If a data burst results in multiple decomposed TLPs, TLPs that have been completely processed before `slv_wmisc_info_silentDrop` is asserted, are not discarded.If `slv_wmisc_info_silentDrop` is asserted on a leading zero-byte beat of a write request, no data is sent on the wire. Asserting `slv_wmisc_info_silentDrop` on a trailing zero-byte beat does not discard the TLP with the preceding data because the AXI bridge drops zero-byte write requests.

Some of these inputs are optional, so you can set them to 0.

Subordinate request sideband output bus use-cases (`slv_rmisc_info`, `slv_bmisc_info`):

- An example of subordinate response sideband output information is the TLP header field information EP/ATTR/TC/AT/ function number/ PCIe CPL status.

Some of these outputs are optional, so you can ignore them.

Example of Simple Outbound MWr Request at Subordinate Interface[1]

1. Application sets `slv_awmisc_info` =22h000000 when `slv_awvalid` =1b1.
2. Bridge responds with `slv_bmisc_info` =11h000 when `slv_bvalid` =1b1.

Example of Sending Simple Outbound MRd Request at Subordinate Interface

1. Application sets `slv_armisc_info` =22h000000 when `slv_arvalid` =1b1.
2. Bridge responds with `slv_rmisc_info` =11h400 when `slv_rvalid` =1b1.

Example of Sending Simple Outbound CfgRd Request at Subordinate Interface

1. Application sets `slv_armisc_info` =22h000004 when `slv_arvalid` =1b1.
2. Bridge responds with `slv_rmisc_info` =11h400 when `slv_rvalid` =1b1.

Caution

All these simple examples assume that following (normal) PCIe system setup:

- Single function/traffic class.
- TLP header fields BCM, ATTR (NS, RO) and AT are cleared.
- All PCIe CPLs complete successfully with no completion errors.
- Address Translation (if enabled) is not performing TYPE translation.

The sideband signals require special attention by your application during the following operations:

- Address or TYPE translation. For more information, see "Internal Address Translation Unit (iATU)" on page 220.
- Sending I/O or CFG transactions. For more information, see "I/O and CFG Transaction Handling" on page 427
- Sending messages. For more information, see "Message Generation" on page 194.

1. Assuming all the other fields are '0' for this simple example, then the main field of importance becomes the TYPE.

Attention You should use the "Internal Address Translation Unit (iATU)" on page 220 instead of the subordinate request sideband bus to perform the previous three types of operations.

- Multi-function operation.
- Address Translation Services (ATS) operation. For more information, see "Address Translation Services (ATS)" on page 248.

5.1.3.1.2 AXI Manager Interface Sideband Signals

The bridge drives the master request sideband output bus (`mstr_armisc_info`[1] or `mstr_awmisc_info`), and your application subordinate drives the master response sideband input bus (`mstr_rmisc_info`).

Master response sideband output bus use-cases (`mstr_armisc_info`, `mstr_awmisc_info`):

- An example of master request sideband output information is the TLP header field information TYPE/RO/NS/TC/AT/ function number, BAR number of posted TLP.
- Transaction flag indicating ROM or I/O access.
- Transaction flags indicating DMA traffic type.

Some of these outputs are optional, so you can ignore them.

Master response sideband input bus use-cases (`mstr_rmisc_info`):

- The only master response sideband information required is the TLP EP bit.

Example of Receiving Simple Inbound MRd Request at Master Interface

1. Bridge sets `mstr_armisc_info` when `mstr_arvalid` =1b1.
2. Application responds with `mstr_rmisc_info` when `mstr_rvalid` =1b1.

5.1.3.2 Supported AXI Burst Operations

The AXI bridge module is compliant with the AMBA 3.0 and 4.0 AXI Specifications.

5.1.3.2.1 Outbound Transfers

- The AXI bridge subordinate supports the incremental burst type (INCR) which is used in conjunction with `ARLEN` and `AWLEN` to define any length of burst.
- The AXI bridge subordinate does not support the FIXED burst type. When your application issues this burst type to the AXI bridge subordinate, a `SLVERR` or `DECERR` response is not returned by the bridge subordinate. In this scenario, the PCIe controller exhibits undefined behavior.
- When leading zero-byte beats are present, the subordinate aligns the address with the DW that contains the first valid byte. When trailing zero-byte beats are present, the subordinate discards them and calculates the new transfer size to match the amount of data effectively/actually received.

Outbound Wrapping Transfers

- To enable support for WRAP bursts, you must set the `CC_SLV_WRAP_ENABLE` parameter.

1. Renamed as mstr_aruser and mstr_awuser in AXI4

- The bridge subordinate converts all WRAP requests into INCR requests.
- Posted write requests are converted into two sequential INCR bursts when the low start address does not match the low wrap boundary. Realignment of the burst would change the order in which the addresses are written, that is, the bytes prior to the wrap boundary being written first. This would violate the ordering rule Posted must not pass posted. The AXI bridge only sends the TLPs with data on the wire, TLPs without data are dropped. If a write request has all the write strobes set to zero, the AXI bridge sends the second TLP only, it drops the first TLP.
- Non-posted read requests with a starting address not aligned to the wrap boundary, are re-aligned to the wrap boundary. This removes the need to create two INCR bursts which would consume two PCIe tags. The controller uses the Subordinate Wrap Response Re-order Buffer to re-align the read completion data before sending it to the subordinate response channel.
- CFG and IO requests are not supported.
- DBI requests are not supported.

5.1.3.2.2 Inbound Transfers

- The AXI bridge master uses the incremental burst type (INCR) which is used in conjunction with `ARLEN` and `WLEN` to define any length of burst.
- The parameter `CC_MSTR_BURST_LEN` controls the maximum burst length generated by the master interface. The controller supports the use of AXI Bus IP that handle burst lengths exceeding the maximum (16) as defined by the AXI Specification. The Synopsys AXI Bus IP supports burst lengths up to 256.
- The AXI bridge master does not use the WRAP and FIXED burst types.

5.1.3.3 Supported AXI Transfer Sizes

All AWSIZE/ARSIZE values are supported for both the subordinate and the master. The AXI bridge supports all non-aligned starting addresses for both inbound and outbound transfers.

5.1.3.3.1 Outbound Transfers

The AXI bridge supports all aligned and non-aligned starting addresses for outbound transfers with the exception of those outlined in "Non Contiguous Byte Enable (NCBE) Support" on page 425. Some examples of typical read and writes are given in Table 5-7 and Table 5-8.

Table 5-7 Examples of Supported Outbound Writes (PCIe TLP length =1 DW)

slv_awaddr	slv_awsize	slv_awlen	slv_wstrb	slv_wdata	PCIe TLP address	PCIe TLP {Last_First} Byte Enables
0x00	2	0	0b0001	0xXXXX_XXaa	0x00	0b0000_0001
0x01	2	0	0b0010	0xXXXX_aaXX	0x00	0b0000_0010

Table 5-8 Examples of Supported Outbound Reads (PCIe TLP length =1 DW)

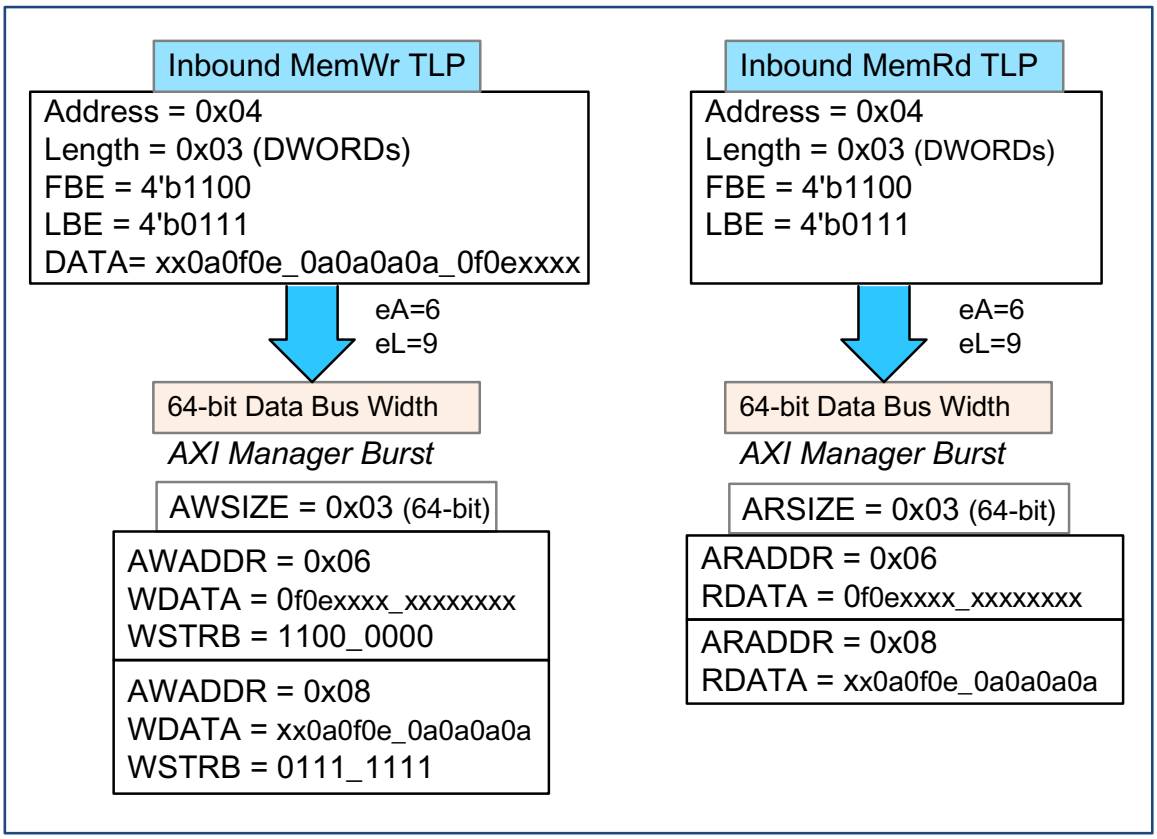
slv_araddr	slv_arsize	slv_arlen	PCIe TLP address	PCIe TLP Last and First Byte Enables	slv_rdata
0x00	0	0	0x00	0b0000_0001	0x0000_00aa

slv_araddr	slv_arsize	slv_arlen	PCIe TLP address	PCIe TLP Last and First Byte Enables	slv_rdata
0x01	0	0	0x00	0b0000_0010	0x0000_aa00
0x01	1	0	0x00	0b0000_0010	0x0000_aa00
0x02	2	0	0x00	0b0000_1100	0xaaaa_0000
0x00	1	0	0x00	0b0000_0011	0x0000_aaaa
0x02	1	0	0x00	0b0000_1100	0xaaaa_0000
0x04	2	0	0x04	0b0000_1111	0xaaaa_aaaa

5.1.3.3.2 Inbound Transfers

When the controllers AXI manager wants to perform a transfer (whose length is greater than or equal to the databus width) it always uses beats of width equal to the databus size that is, `mstr_awsize/mstr_arsize` matches the width of the data bus. Any unwanted data that is read is discarded. Therefore, only prefetchable[1] memory is supported.

Figure 5-11 Examples of Inbound AXI Read and Write Transfers (eA=effective Address; eL=effective Length)



Notes on Figure 5-11.

1. For example, RAM where reading excess data does not prevent you from re-reading that data at a later time. Non-prefetchable memory would be a FIFO where data reads are destructive.

- **Effective Length (eL)**: Effective PCIe TLP (byte) length excluding byte enable holes as per the first and last byte enable (FBE/LBE)
- **Effective Address (eA)**: Effective PCIe TLP (byte) address after resolution of FBE/LBE and DWORD-aligned TLP address. Becomes the AXI address (`araddr` or `awaddr`)

Inbound Narrow Reads

When the controllers AXI manager executes a narrow (total number of read bytes < width of AXI databus) read transfer, it automatically selects one of three methods.

Table 5-9 Methods for Executing Narrow Inbound Reads at AXI Manager Interface

Method and Description	Automatically Selected When
Approximate Full Size (Default Operation for Some Configurations) *Use for prefetchable memory.* Use an approximate byte count and perform one or more full-width accesses. With this method, the master uses the entire bus width (mstr_arsize is set to match the full width of the bus). The controllers master reads in more bytes than the intended narrow transfer. Excess data is discarded and not forwarded to the link partner.	(Single Beat Exact Size eA/eL criterion not satisfied || ! MSTR_SUB_BUS_SIZE_READ_ENABLE) && MSTR_FIXED_SIZE_ENABLE[a]
Multiple Beat Exact Byte Size (Default Operation for Some Configurations) *Use for non-prefetchable memory (for example, FIFO).* Use an exact byte count and perform a burst of byte-wide accesses. To enable a sufficiently-long burst, the following relationship is enforced. `CC_MSTR_BURST_LEN >= MASTER_BUS_DATA_WIDTH/8 whenMSTR_SUB_BUS_SIZE_READ_ENABLE=0 && MSTR_FIXED_SIZE_ENABLE=0` If it is not possible to use Single Beat Exact Size, then this method must be used rather than Approximate Full Size method when the memory is non-prefetchable and [b]read coherency is not an issue. This method provides precision, but it sacrifices performance. To access n bytes, the master uses an n-beat byte-wise burst transfer.	(Single Beat Exact Size eA/eL criterion not satisfied || ! MSTR_SUB_BUS_SIZE_READ_ENABLE) && ! MSTR_FIXED_SIZE_ENABLE
Single Beat Exact Size (Non-Default Operation) *Use for non-prefetchable memory (for example, FIFO).* Use an exact byte count and perform a single AXI access of width (mstr_arsize) equal to the PCIe effective length (eL). A design complexity decision is taken to only support scenarios where the complete request can be completed within a single AXI beat.	(Single Beat Exact Size eA/eL criterion[c] is satisfied && MSTR_SUB_BUS_SIZE_READ_ENABLE[d])

Method and Description	Automatically Selected When
Notes ■ Effective Length (eL): ❑ Effective PCIe TLP (byte) length excluding byte enable holes as per the first and last byte enable (FBE/LBE) ■ Effective Address (eA): ❑ Effective PCIe TLP (byte) address after resolution of FBE/LBE and DWORD-aligned TLP address. ❑ Becomes the AXI address (mstr_araddr) ■ Number of beats in the burst is eL / AXI Transfer Size (in bytes)	

a. Default is [(CC_MSTR_BURST_LEN=8 && MASTER_BUS_DATA_WIDTH=64) || (CC_MSTR_BURST_LEN < MASTER_BUS_-DATA_WIDTH/8)] ? 1:0.
b. For example, reading a 24-bit wide status register where all 24-bits must be read together; and where there is no coherency protection provided in the status register itself. Reading one byte at a time produces erroneous results if your application updates status register during the read process.
c. The Single Beat Exact Size eA/eL criterion is: eL=2n and (eA is aligned to mstr_arsize).
d. Default is 0.

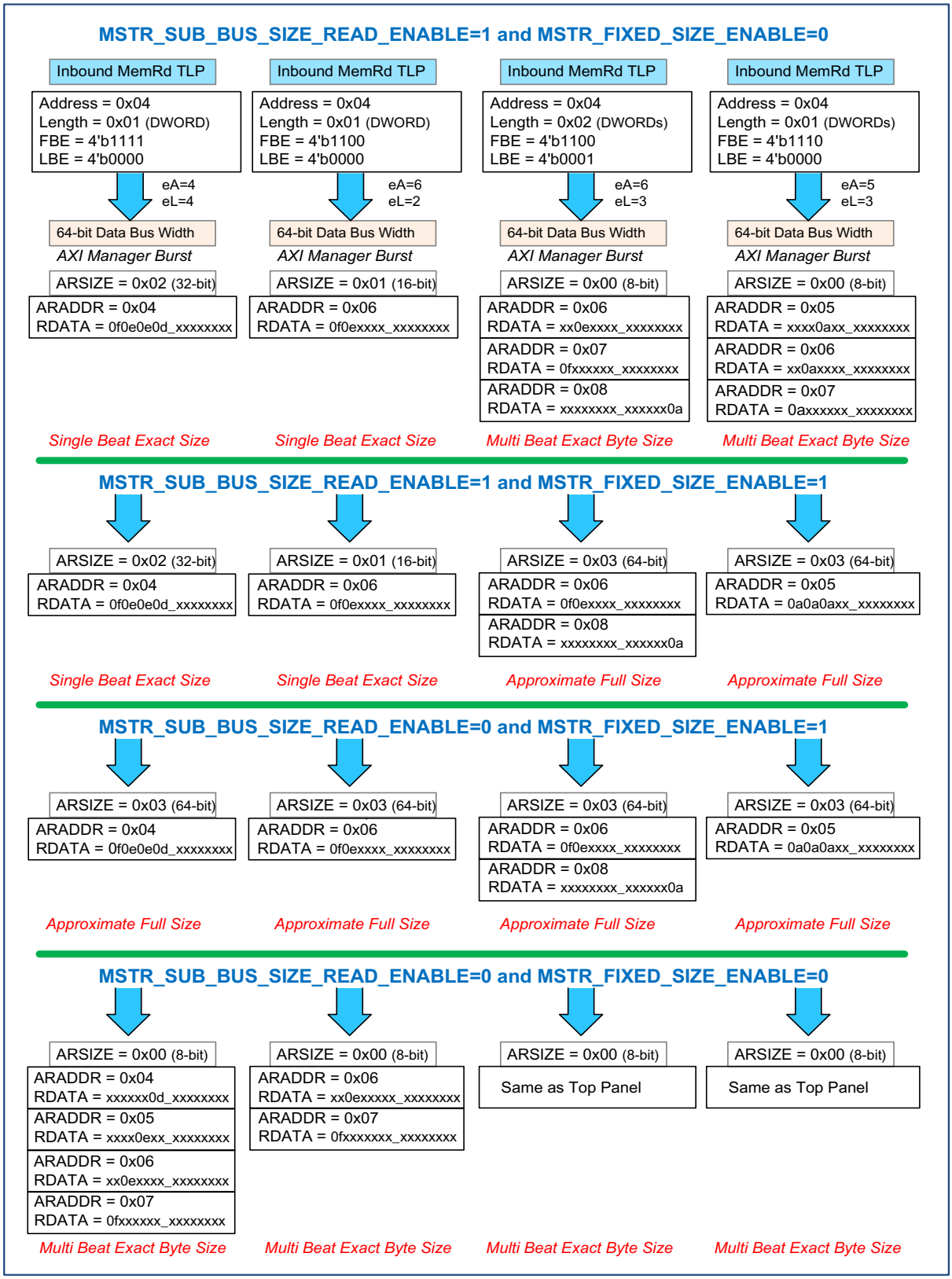
Table 5-10 Automatic Selection of Narrow Read Methods by AXI Manager

MSTR_SUB_BUS_SIZE_READ_ENABLE,MSTR_FIXED_SIZE_ENABLE	Single Beat Exact Size eA/eL Criterion is Satisfied	Method Used
0 0	No	Multiple Beat Exact Byte Size
0 0	Yes	Multiple Beat Exact Byte Size
0 1	No	Approximate Full Size
0 1	Yes	Approximate Full Size
1 0	No	Multiple Beat Exact Byte Size
1 0	Yes	Single Beat Exact Size
1 1	No	Approximate Full Size
1 1	Yes	Single Beat Exact Size

Note: 2n indicates a non-zero integer power of 2.

Note See also “Non Contiguous Byte Enable (NCBE) Support” on page 425.

Figure 5-12 Examples of Inbound AXI Narrow Read Transfers (eA=effective Address; eL=effective Length)



Inbound Narrow Writes

When the controllers AXI manager wants to perform a narrow WRITE transfer when the total number of bytes to be transferred is less than the width of the AXI data bus it can choose one of two methods. The selection of which method to use is automatically selected by the master within constraints defined by the `MSTR_SUB_BUS_SIZE_WRITE_ENABLE` configuration parameter.

Table 5-11 Methods for Executing Narrow Inbound Writes at AXI Manager Interface

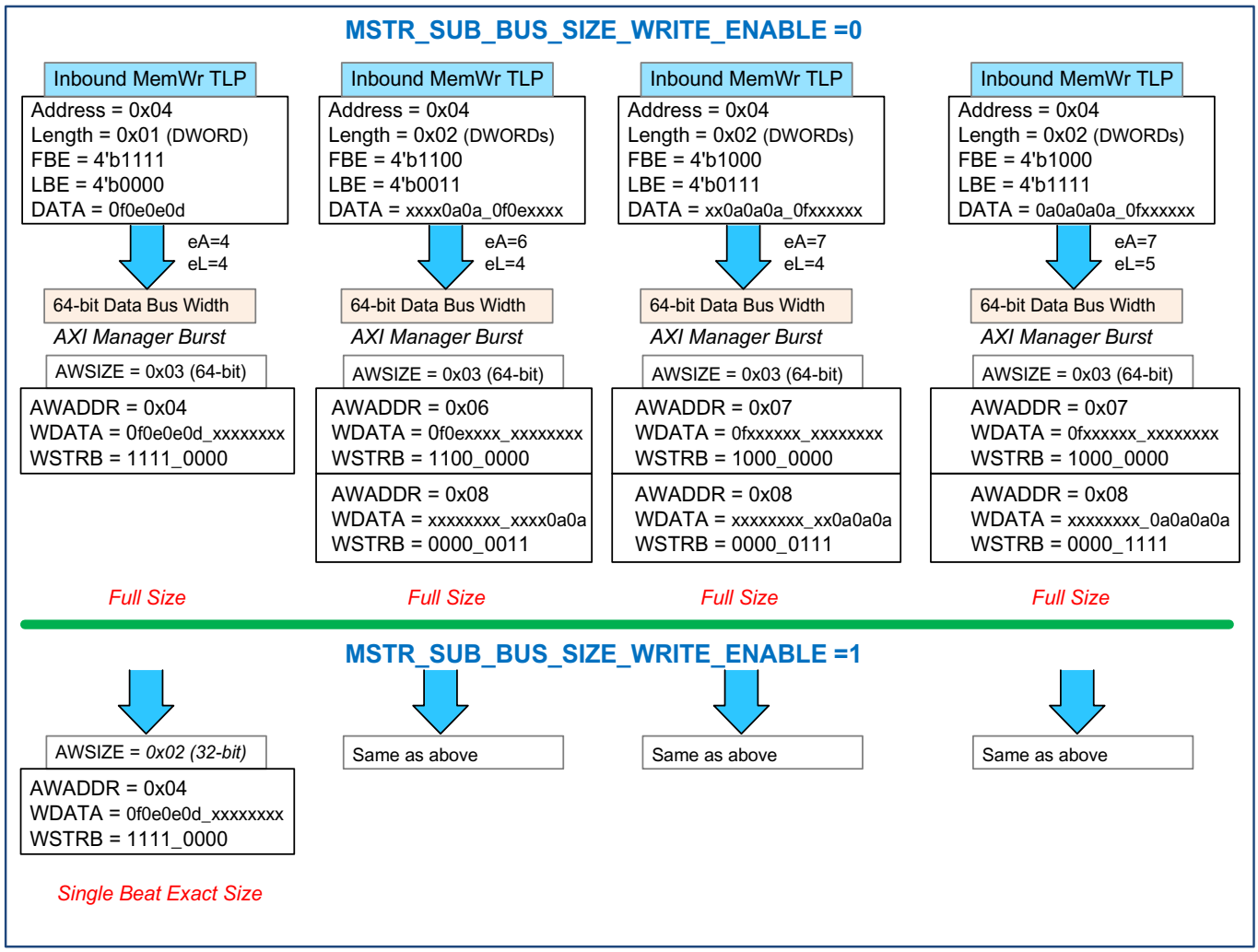
Method and Description	Automatically selected when
Single Beat Exact Size Perform a single AXI access of width (mstr_awsize) equal to the PCIe effective length (eL). A design complexity decision is taken to only support scenarios where the complete request can be completed within a single AXI beat.	eL =2n[a] && eA (`mstr_awaddr`) is aligned to `mstr_awsize` && `MSTR_SUB_BUS_SIZE_WRITE_ENABLE =1`
Full Size Perform one or two accesses of width equal to AXI databus width. With this method, the controller master uses the entire bus width (mstr_awsize is set to match the full width of the bus). The write strobes ensure a precise write to the intended write request address location.	Previous condition not met.
Notes ■ Effective Length (eL): ❑ Effective PCIe TLP (byte) length excluding byte enable holes as per the first and last byte enable (FBE/LBE) ■ Effective Address (eA): ❑ Effective PCIe TLP (byte) address after resolution of FBE/LBE and DWORD-aligned TLP address. ❑ Becomes the AXI address (`mstr_awaddr`) ■ Number of beats in the burst is Effective Length (as defined in this section) / AXI Transfer Size (in bytes)	

a. 2n indicates a non-zero integer power of 2.

Table 5-12 Selection of Narrow Write Methods

MSTR_SUB_BUS_SIZE_WRITE_ENABLE	eL =2n && eA (mstr_araddr) is aligned to mstr_arsize	Method Used
0	No	Full Size
0	Yes	Full Size
1	No	Full Size
1	Yes	Single Beat Exact Size

Figure 5-13 Examples of Inbound AXI Narrow Write Transfers (eA=effective Address; eL=effective Length)



5.1.3.4 Non Contiguous Byte Enable (NCBE) Support

The *PCI Express Base Specification, Revision 6.0, Version 1.0* supports memory reads and writes with holes only when the read and write requests are one or two DWORDs long. The PCIe transaction header first and last byte enable fields are used to signal which bytes are valid. A hole means that it is possible to have non-contiguous byte enables such as 4b0101. Non-contiguous byte enables occur when enabled bytes are separated by non-enabled bytes, for example, 0101b. The rules for byte enables are given in Section 2.2.5 "First/Last DW Byte Enables Rules" of the *PCI Express Base Specification, Revision 6.0, Version 1.0* Byte Enables are included with memory, I/O and configuration requests. The AXI protocol itself can support holes in write data as there are AXI write strobes.

Table 5-13 AXI Inbound NCBE Support for Requests

	MemWr	IOWr	CfgWr	MemRd	IORd	CfgRd	DMWr
NCBE Supported	Yes	Yes	Yes	Yes	Yes	Yes	No
Size Restrictions	1,2 DWORD	1 DWORD	1 DWORD	1,2 DWORD	1 DWORD	1 DWORD	-

Table 5-14 Method and Description

Method and Description	
READS	**Decompose Method (default)** The original PCIe request is first decomposed[a] into a series of internal single byte request TLPs. These appear on the master interface according to the rules in Table 5-9. Likewise, the responses to the individual decomposed requests are composed into a single (or double) DWORD of response data from that address. This method should be used when the memory is non-prefetchable and [b]read coherency is not an issue. This method provides precision, but it sacrifices performance. To access n bytes, the master uses an n-beat byte-wise burst transfer. **Non-Decompose Method (CC_AXI_NCBE_NODECOMP =1)** The original PCIe request is not decomposed as in the previous method. It appears on the master interface according to the rules in Table 5-9. This method is automatically[c] selected when MSTR_FIXED_SIZE_ENABLE=1 && MSTR_SUB_BUS_SIZE_READ_ENABLE=0.
WRITES	**Full Size** AXI write strobes (`mstr_wstrb`) are used. Perform accesses of width equal to AXI databus width. With this method, the controller master uses the entire bus width (`mstr_awsize` is set to match the full width of the bus). The write strobes ensure a precise write to the intended write request address location.

a. For more information, see "Inbound Bridge Tag Management" on page 441 regarding AXI ID usage for NP requests.
b. For example, reading a 24-bit wide status register where all 24-bits must be read together; and where there is no coherency protection provided in the status register itself. Reading one byte at a time produces erroneous results if your application updates status register during the read process.
c. This results in a performance improvement when Approximate Full-Size transfers are enforced at the master interface. (For more information, see "Inbound Transfers" on page 420) because it makes no sense to decompose NCBE inbound transfers when it only results in multiple prefetches of the same address location.

Note Narrow read/writes and 2-DWORD NCBE read/writes are treated independently, but a 1-DWORD NCB narrow transfer. This might constitute a conflict when you want NCBE transfers to be non-prefetchable prefetchable.

Table 5-15 AXI Outbound NCBE Support for Requests

	MemWr	IOWr	CfgWr	MemRd	IORd	CfgRd	DMWr
NCBE Supported	Yes	Yes	Yes	No	No	No	No
Size Restrictions	-	1 DWORD		-	-	-	-

When the AXI subordinate detects null write strobes in midst of a transfer, it decomposes the request into multiple requests. These requests are two DWORD or QWORD-aligned, except for the first one. The first request might be a bigger one if multiple contiguous requests can be joined, without any discontinuous bytes in between.

Note Messages and Deferrable Memory Write do not support NCBE.

5.1.3.5 Zero-Byte Transfers Over the AXI Bridge (Flush Semantics)

The *PCI Express Base Specification, Revision 6.0, Version 1.0* supports zero-byte transactions as a means of flushing out previous transactions.

5.1.3.5.1 Outbound Zero-Byte Transfers

- **Tx Write**: The AXI specification (and the bridge) supports Tx zero byte writes by setting all write strobes (`slv_wstrb`) to 0. This behavior is a specific use case of leading/trailing zero byte behavior described in "Outbound Transfers" on page 418.
- **Tx Read**: The AXI specification (and the bridge) provides NO support mechanism for Tx zero-byte reads.

5.1.3.5.2 Inbound Zero-Byte Transfers

- Rx Read/Write: Option 1 (`DEFAULT_FILTER_MASK_2[3] =1`)

  The native PCIe controller 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 is 0) in the Filter Mask register of the native controller to enable this feature. If the native PCIe controller is configured to support termination of zero-byte transactions, it generates CPLs for zero-byte reads and return them automatically to the requester without sending the transaction to the AXI bridge. It also drops zero-byte writes. Transactions arriving at the PCIe receive queues are terminated with the flush function.

  Configuring the native PCIe controller 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 bridges queue buffer. In other words, the flush semantic terminates at the native PCIe controller's receive queues. For more information, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.

- Rx Read/Write: Option 2 (`DEFAULT_FILTER_MASK_2[3] =0`) - RECCOMENDED

  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 controller and present it to the AXI interface. The AXI specification (and the bridge) supports zero byte writes by setting all write strobes (`mstr_wstrb`) to 0. An inbound zero-byte read is handled as follows:

  - If, `AMBA_ORDERING_CTRL_OFF.AX_MSTR_ZeroLRead_FW =0`, an inbound zero-byte read is terminated at the AXI bridge master when any prior Posted transaction finishes, implementing the PCIe flush semantics. When the zero-byte read terminates, it synthesizes a CPL transaction back to the requester.
  - If, `AMBA_ORDERING_CTRL_OFF.AX_MSTR_ZeroLRead_FW =1`, an inbound zero-byte read is translated into a 1 byte read (from the original request's address) on the AXI manager and forwarded to the application. Zero-byte reads are not supported by the AXI protocol; sideband signal `mstr_armisc_info_zeroread` is used to signal the zero-byte read your application. The resulting AXI response triggers the respective CPL back to the requester.

    Note: This functionality can be useful, for example, to implement zero-length Lightweight Notification (LN) read transactions targeted at the LN completer logic implemented by the application.

5.1.3.6 I/O and CFG Transaction Handling

I/O and CFG-type outbound transfers are fully supported by the AXI bridge. I/O and CFG transactions always have the following characteristics:

- A payload of length equal to one DWORD (four bytes, 32 bits).
- Last byte enable (LBE) is always 0000.
- First byte enable (FBE) can be any value including 0000 (0000 possible through "Zero-Byte Transfers Over the AXI Bridge (Flush Semantics)" on page 427).

5.1.3.6.1 Outbound I/O and CFG Transaction Handling

The following conditions must be satisfied for all outbound CFG and I/O requests presented by your application to the bridge subordinate interface:

- The outbound I/O or CFG transactions should not cross DWORD boundary.
- The outbound AXI-to-PCIe transfer must complete in a single AXI transfer without the need for a bus burst or a series of bus transfers.
- The maximum burst length for outbound I/O and CFG transfers is one, corresponding to one DWORD.
- Non contiguous byte enable (NCBE "Non Contiguous Byte Enable (NCBE) Support" on page 425) behavior is supported for writes but not reads.

To signal the PCIe controller that your application wants to send an I/O or CFG transfer, you can use:

- **Address Translation Method of Sending an Outbound I/O or CFG Transfer (Recommended)**

  The optional internal address translation unit (iATU) in the native controller can change the TYPE of an outbound request from MEM to I/O or CFG by matching the address of that request to a configured address range as set by your application. For more information, see "Internal Address Translation Unit (iATU)" on page 220.

- **Sideband Bus Method of Sending an Outbound I/O or CFG Transfer**

  I/O and CFG outbound transfers are signaled to AXI bridge through the `slv_awmisc_info`[1] or `slv_armisc_info` buses using the Type field in this bus. For more information on bit location details, see "Signal Interfaces" on page 355. The 5-bit Type field encoding is compliant with the *PCI Express Base 3.1 Specification, revision 1.0*. The AXI sideband (misc. bus) bus signals must be driven and sampled properly during I/O and CFG transactions. For more information, see "AXI Sideband (Misc. Bus) Signals" on page 414 for more information and examples.

5.1.3.7 AXI Inbound Posted ID Management ID Mapping

When `CC_IB_P_ID_MNG_EN` parameter is set to '1', the AXI Bridge assigns AXI IDs to Inbound Posted Requests. The ID is selected based on the VC, IDO, RO and DMA status of the request. This enables your application to return AXI B responses out-of-order.

For example: This feature allows the following ordering cases to occur in your application:

- Posted can pass Posted from another VC
- Within the same VC, RO Posted can pass Posted
- Within the same VC, IDO Posted can pass Posted from another IDO ordering stream
- No ordering relationships between DMA and PCIe Posted
- Non-posted request can pass Posted request from another VC

For configurations without Order Buffer[2], Reads and Non-Posted Writes are ordered within their VC:

1. Renamed as slv_awuser in AXI4

- Inbound AXI Read Requests are ordered per VC instead of globally
- Inbound AXI NP Write Requests are blocked ("NP must not pass P" rule) per VC instead of globally
- "NP pass P" configuration, within the bridge, is evaluated per VC instead of globally

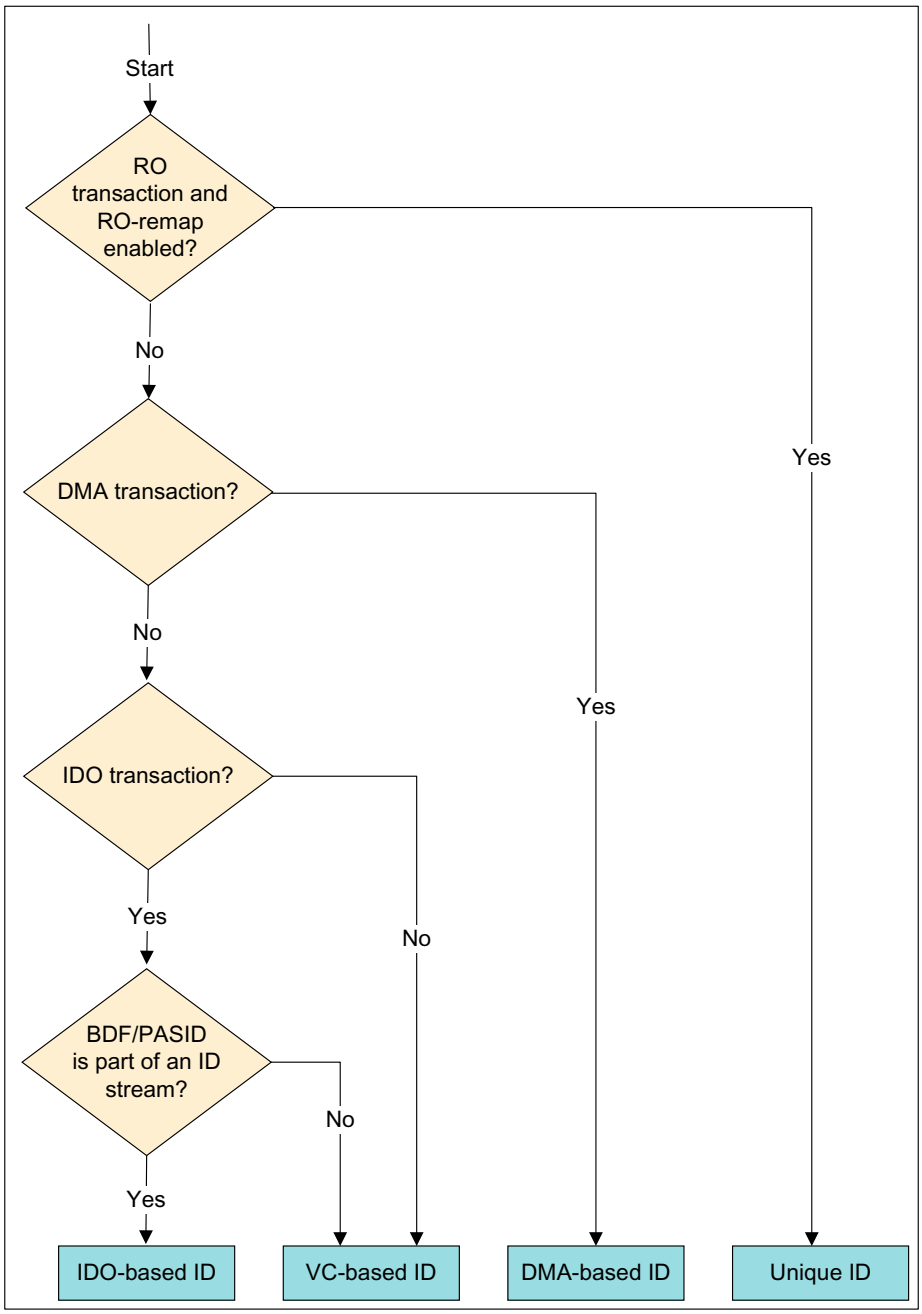
Note The controller re-orders the inbound write requests per VC only when Order Buffer is enabled. The re-ordering is regardless of the relaxation on the application side through the assignment of distinct IDs.

5.1.3.7.1 AXI Inbound Posted ID Management ID mapping

The Posted AXI ID is chosen to allow maximum performance while still adhering to PCIe ordering rules. For DMA, a single AXI ID is shared for all DMA Posted. The ID is chosen according to Figure 5-14.

2. Configurations with Order Buffer takes precedence. For more information, see "Order Buffer" on page 459.

Figure 5-14 AXI Posted ID Management Decision Tree



Note

- IDO-based IDs and Unique IDs for ROs are only supported when the Order Buffer is enabled.
- RO-remap is enabled by default. It can be controlled per VC with the IB_P_ID_MNG_CTRL register in the Port Logic space.

The AXI IDs are mapped sequentially according to Figure 5-15.

Figure 5-15 AXI Posted ID Management ID Mapping

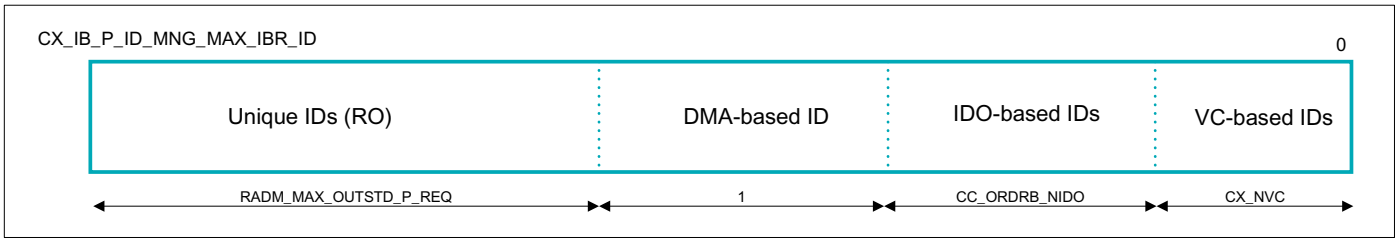


Table 5-16 Example: Configurations with CX_NVC = 2, CC_ORDRB_NIDO = 4, CC_DMA_ENABLE = 1, RADM_MAX_OUTSTD_P_REQ = 64

Basis	Minimum AXI ID	Maximum AXI ID
VC	0 (VC 0)	1 (VC 1)
IDO	2 (IDO 0)	5 (IDO 3)
DMA	6	6
Unique ID (ROs)	7	70

5.1.3.8 Error Handling

This section discusses how the controller maps AXI and PCIe errors. The following topics are discussed:

- "Subordinate AXI-to-PCIe Error Mapping"
- "Poisoned TLP Handling (TX Path)" on page 433
- "Master AXI-to-PCIe Error Mapping" on page 433
- "LCRC Handling (RX Path)" on page 434
- "ECRC Handling (RX Path)" on page 434
- "Poisoned TLP Handling (RX Path)" on page 435

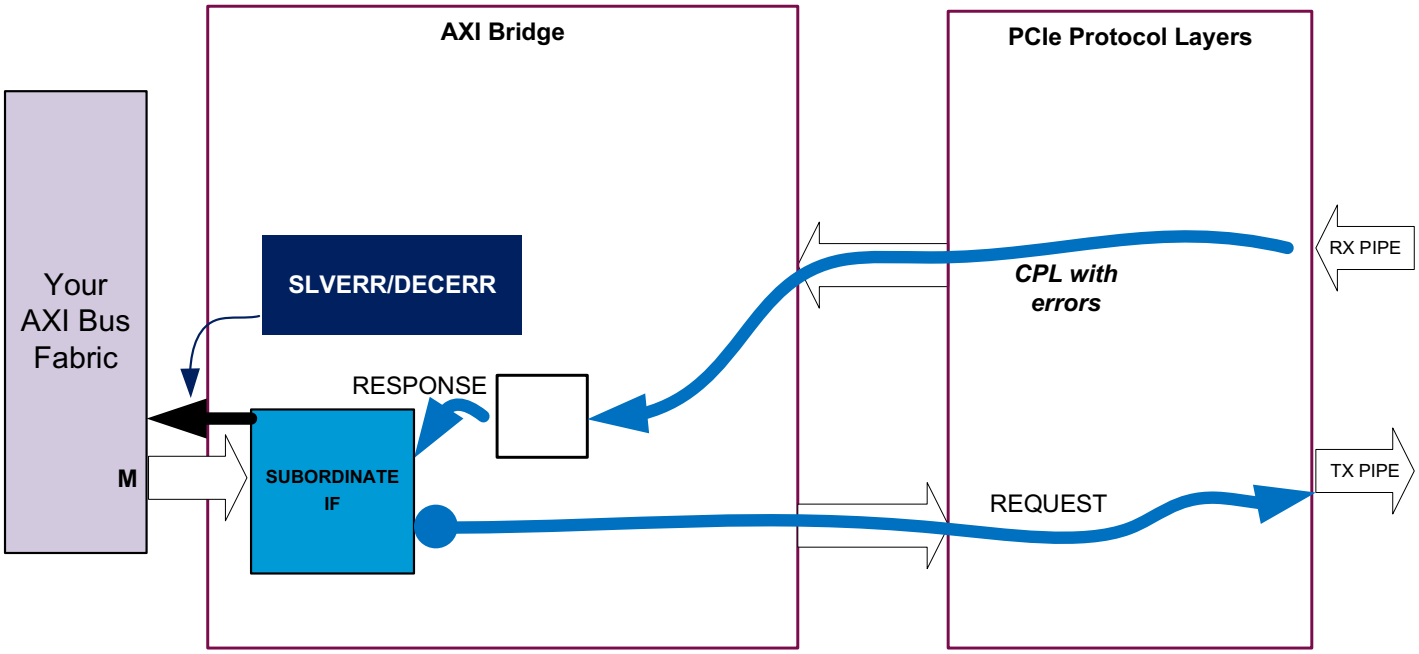
5.1.3.8.1 Subordinate AXI-to-PCIe Error Mapping

Subordinate error mapping refers to what the controller signals back to your application AXI manager on `slv_*resp` when it receives completion errors (CA, UR, CRS, and CPL-Timeout).

Table 5-17 AXI Subordinate Response for Erroneous Requests

Access Type	Transaction	Error Response
Wire	Read (Non-Posted) Write (Non-Posted)	OKAY (default) ■ The payload is set to 0xFFFF for reads ■ slv_*misc_info can be used to determine the type of completion error SLVERR/DECERR ■ If your application sets the AMBA_ERROR_RESPONSE_GLOBAL field to '1', and configures the AMBA_ERROR_RESPONSE_MAP field in the AMBA_ERROR_RESPONSE_DEFAULT_OFF register
	Posted	OKAY
DBI	Read Write	OKAY (default) SLVERR ■ Read or write request with length greater than 1 beat ■ Read or write request crossing DWORD boundary ■ Write request with all null write strobes

Figure 5-16 PCIe-to-AXI Subordinate Error Mapping



CRS CPL Error

You can choose between different ways to handle a CRS completion error by programming `AMBA_ERROR_RESPONSE_CRS` field in the `AMBA_ERROR_RESPONSE_DEFAULT_OFF` register. The following options are available:

- 00: The AXI bridge always returns OKAY with non-FFFF data when a CRS completion is received. This is useful when you are monitoring the completions status field of `slv_bmisc_info/slv_rmisc_info` bus to determine the type of completion error.

- 01: The AXI bridge always returns OKAY with all-FFFF data when a CRS completion is received. This is useful when you do not want to monitor the completion status field of `slv_bmisc_info/slv_rmisc_info` bus to determine the type of completion error; but you still want to detect that a completion is received with error.
- 10: The AXI bridge always returns OKAY with FFFF0001 data when a CRS completion is received for a Vendor ID read request, and OKAY with all-FFFF data for all other CRS completions. This is useful when you do not want to monitor the completion status field of `slv_bmisc_info/slv_rmisc_info` bus to determine the type of completion error; but you still want to distinguish between CRS, and UR completions to Vendor ID read requests.
- 11: The AXI bridge always returns SLVERR or DECERR for all CRS completions, depending on the bit 1 value of `AMBA_ERROR_RESPONSE_MAP` field in the `AMBA_ERROR_RESPONSE_DEFAULT_OFF` register.

Reading Non-existent Vendor ID Register

The PCIe specification states that 0xFFFF is a reserved value for the Vendor ID register, to indicate that the function (containing Vendor ID register) is non-existent. An outbound CfgRd at the AXI bridge subordinate interface to a non-existent register (at the remote link partner) normally results in a UR response back from the PCIe wire without data. The AXI bridge subordinate automatically converts the completion with the UR response, to an OKAY response on the AXI subordinate interface, and also sets the completion data payload to 0xFFFF. This feature is supported for all CfgRd accesses (excluding byte-size accesses) to the Vendor ID register. You can modify this behavior using the `AMBA_ERROR_RESPONSE_VENDORID` bit of `AMBA_ERROR_RESPONSE_DEFAULT_OFF`.

5.1.3.8.2 Poisoned TLP Handling (TX Path)

The handling of poisoned TLPs (when EP bit is set; valid only for TLPs that include Data Payload).

- **Requests**: You can poison output write TLPs (if you want them discarded by remote link partner), for more information, see "AXI Subordinate Interface Sideband Signals" on page 416.

5.1.3.8.3 Master AXI-to-PCIe Error Mapping

Master error mapping refers to the AXI bus responses (for non-posted requests), and what the controller signals back to the remote link partner.

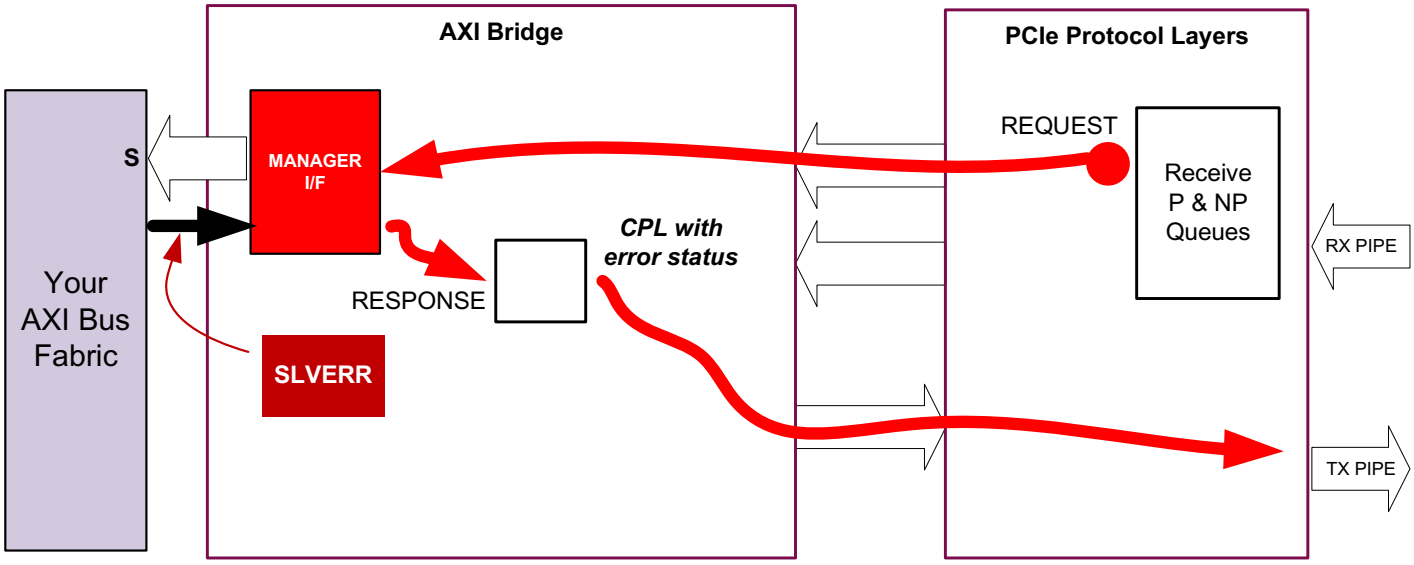
- Error-free responses (OKAY) are normally mapped to a CPL with Successful Completion (SC) status.
  - You can change this to UR or CA using `mstr_bmisc_info_cpl_stat/ mstr_rmisc_info_cpl_stat` if for example, you are tunneling downstream error responses.
- Errored responses (`SLVERR/DECERR`) are always mapped to a CPL with CA status.
  - If data payload exceeds PCIe Maximum Transfer Unit size, split completions are sent on the wire. The split completions have completion status as per `mstr_bmisc_info_cpl_stat/ mstr_rmisc_info_cpl_stat`.The moment the first errored response appears, all the subsequent completions have completion status as CA.

For non-posted requests an errored master response results in the AXI bridge terminating the transmission of completions followed by a CA errored response completion. The AXI bridge master takes the following actions for each outstanding non-posted request when an errored master response data beat is detected:

1. Accepts the remaining master response beats without back pressuring the master response interface or canceling any of the remaining burst beats.
2. Sends a completion with CA status. No data is included in the completion.

3. Continues to re-format the master response packets until the last master response data beat for the last decomposed packet is detected.

Figure 5-17 AXI-to-PCIe Master Error Mapping



The AXI bridge master does not generate an error message (to the remote link partner), or set the error bits in the PCI Compatible Status or Device Status registers, in response to a master error[1]. When your application detects an error in a TLP that is forwarded by the AXI bridge master, it could perform the following actions to execute a complete error reporting mechanism:

- Return AXI error response to the AXI bridge master interface. In response to this, for non-posted requests, the controller transmits a completion with the appropriate completion status.
- Send an error message through the subordinate interface and log the error status (by using the "Application Error Reporting Interface" on page 724). This is particularly important for posted requests, as there is no corresponding completion.

5.1.3.8.4 LCRC Handling (RX Path)

When an LCRC error occurs on an inbound packet (request or completion), the controller automatically discards that packet and waits for the remote device to replay the packet. The AXI bridge does not signal an error to your application but waits for the replayed packet to arrive.

5.1.3.8.5 ECRC Handling (RX Path)

The handling of ECRC errors is different for completions and requests.

- **Requests**: When the controller receives a request TLP with ECRC errors, it does not transmit it to your application through the AXI manager[2]. When you set the `DEFAULT_TARGET` (USP only) and `CX_MASK_UR_CA_4_TRGT1` fields in the `MISC_CONTROL_1_OFF` register to '1', then a request TLP with ECRC errors is not dropped and is forwarded to your application on the AXI bridge master. For more information, see "Routing of Request TLPs with Errors (AMBA Bridge Configuration)" on page 719.

1. When you map the master error response to CA, then Signaled Target Abort is set in the PCI Compatible Status register.
2. If (Segmented-buffer && CX_FLT_MASK_ECRC_DISCARD && !ECRC_ERR_PASS_THROUGH), then forwarded to your application logic.

- **Completions**: When the controller receives a completion with ECRC errors, it does not transmit it to your application through the AXI subordinate[1]. Instead, the bridge waits for the subsequent CPL timeout generated by the controller. It is this timeout that causes the bridge to issue an ERROR response to your application on each beat of the burst corresponding to the original AXI request.

5.1.3.8.6 Poisoned TLP Handling (RX Path)

The handling of poisoned TLPs (when EP bit is set) is different for completions and requests.

- **Requests**: By default, the controller treats poisoned requests as UR and drops them. When you set the `CX_FLT_MASK_UR_POIS` filter rule to '1', then the AXI bridge master passes poisoned requests to your application by asserting `mstr_awmisc_info_ep`.
- **Completions**: The AXI bridge subordinate always passes poisoned completions to your application and notifies your application by setting the EP bit `slv_rmisc_info[1]/slv_bmisc_info[1]`. The AXI bridge subordinate sets `slv_rresp/slv_bresp` to OKAY in this case. The `CX_FLT_MASK_UR_POIS` filter rule does not affect this behavior.

5.1.4 Advanced AXI Reference Material

This section provides advanced information regarding the buffering in the AXI bridge. The following topics are covered:

- "AXI Decomposition Rules"
- "Outbound Bridge Tag Management" on page 439
- "Inbound Bridge Tag Management" on page 441
- "AXI Decomposition Rules" on page 435

5.1.4.1 AXI Decomposition Rules

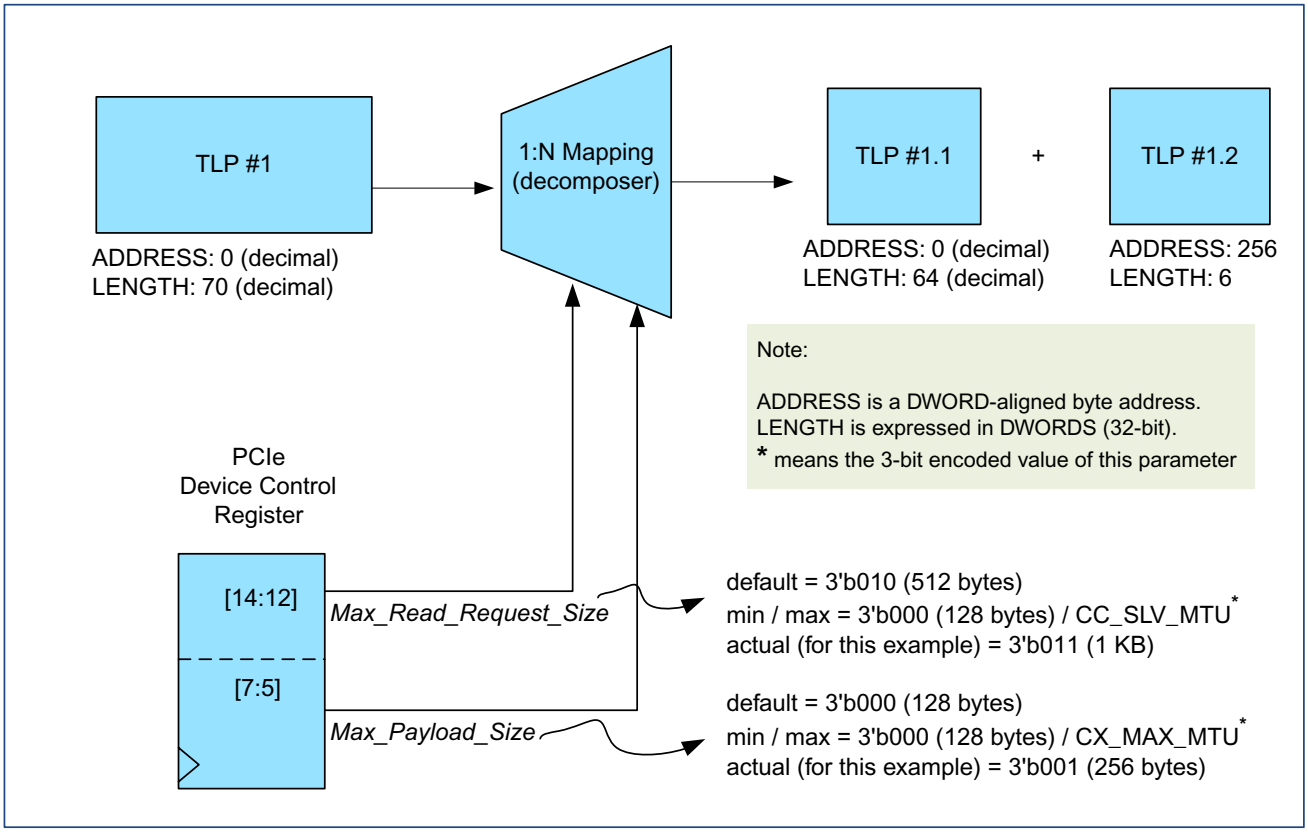
The host software can program the PCIe device to support certain maximum write transfer sizes and maximum read request sizes. The native PCIe controllers configuration module contains the devices Max_Payload_Size and Max_Read_Request size information, which are defined by your application software. The AXI bridge supports mismatches that occur when the AXI maximum transfer length is different than the Max_Payload_Size and Max_Read_Request size. For example, an inbound read transfer has an associated response buffer that is dependent on the remote PCIe devices Max_Read_Request size. An inbound write transfer has a master write buffer that is dependent on the remote PCIe devices Max_Payload_Size.

5.1.4.1.1 Outbound Decomposition

Automatic segmentation and reassembly of outbound (application to PCIe wire) packets is performed to satisfy PCI Express Max_Read_Request_Size and Max_Payload_Size rules. When a decomposition rule is triggered, the outbound TLP is broken up into two or more smaller TLPs. When the TLP type is non-posted then additional PCIe tags are used from the tag pool. Additional entries in the outbound header buffer are also used.

1. If (Segmented-buffer && CX_FLT_MASK_CPL_ECRC_DISCARD && !ECRC_ERR_PASS_THROUGH), then forwarded to your application logic.

Figure 5-18 Outbound Decomposition Example of MWr with Max Payload Size set to 256 bytes (64 DWORDs)



The subordinate request decomposer module uses the rules in Table 5-18 to determine when decomposing takes place.

Table 5-18 Outbound Decomposition Rules

TLP request Type	Decompose When
AXI Burst Size = 1 Byte	
Read	AXI Burst Size * AXI Burst Length > PCIe Max_Read_Request_Size - Address[1:0]
Write	AXI Burst Size * AXI Burst Length > PCIe Max_Payload_Size - Address[1:0]
AXI Burst Size = 2 Byte	
Read	AXI Burst Size * AXI Burst Length > PCIe Max_Read_Request_Size- Address[1]
Write	AXI Burst Size * AXI Burst Length > PCIe Max_Payload_Size - Address[1]
AXI Burst Size >= 4 Byte	
Read	AXI Burst Size * AXI Burst Length > min(PCIe Max_Read_Request_Size, CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN)
Write	AXI Burst Size * AXI Burst Length > PCIe Max_Payload_Size

Note

CX_MAX_MTU (Max_Payload_Size_Supported)

- The largest packet payload that the device supports. Also referred to as PCIe Maximum Transfer Unit (MTU) or Max_Payload_Size_Supported.
- Must be between 128 and 4096. Defaults to 256. Specified in bytes and not DWORDs.
- This is distinct from the maximum operating payload (Max_Payload_Size) set by software.

CC_SLV_MTU (=CC_SLV_BURST_LEN * (SLAVE_BUS_DATA_WIDTH / 8))

- Subordinate Maximum Transfer Unit. The application master (at the AXI Subordinate Interface) must not request or write more than CC_SLV_MTU bytes in a single AXI burst.
- CC_SLV_BURST_LEN is the maximum burst in beats that your application ever issues to the bridge subordinate interface. Must be between 4 and 256. Defaults to 16

Caution

The AXI protocol demands the AXI page boundary to be 4KB. Your application master (at the PCIe controller AXI subordinate interface) should not issue a request crossing the 4KB PCIe page boundary. The requests crossing 4KB boundaries are considered invalid by the controller.

Decomposition Side-Effects

Decomposition degrades the PCIe link performance because it increases the amount of TLP header overhead and uses up extra PCIe tags. Decomposition uses up extra header FIFO locations.

Reducing Outbound Decomposition

To reduce decomposition on the outbound path, observe the following guidelines. For write requests:

- Ensure that your application master does not generate bursts of size greater than or equal to Max_Payload_Size. In this context burst size refers to the total number of bytes requested, or
- Program your PCIe system with a larger value of Max_Payload_Size without exceeding `CX_MAX_MTU`.

For read requests:

- Ensure that your application master does not generate bursts of size greater than or equal to Max_Read_Request_Size, or
- Program your PCIe system with a larger value of Max_Read_Request without exceeding `CC_SLV_MTU`.
- Ensure that your application master does not generate bursts of size greater than `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN`, or increase the parameter value.

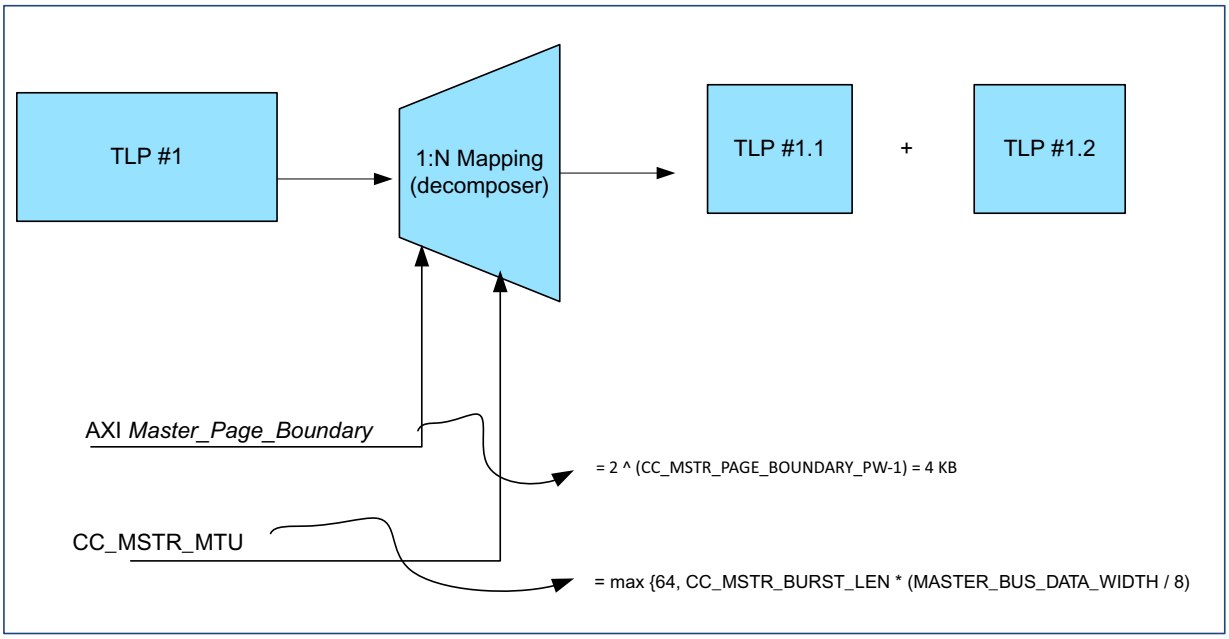
Attention

- The bridge does not handle decomposition of messages that have a payload (MsgD).
- Outbound MsgD requests bigger than Max_Payload_Size cause data corruption.

5.1.4.1.2 Inbound Decomposition

In a similar manner, automatic segmentation of inbound packets is performed. Figure 5-19 illustrates the rules that are used in the master request decomposer module to determine when decomposing takes place.

Figure 5-19 Inbound Decomposition



The following rules are used in the master request decomposer module to determine when decomposing takes place.

Table 5-19 Inbound Decomposition Rules

TLP Request Type	Decompose When
Read	TLP length (bytes) > CC_MSTR_MTU (see Figure 5-19)
Write	TLP length (bytes) > CC_MSTR_MTU (see Figure 5-19)
Read or Write	TLP address + TLP length crosses an a AXI Master_Page_Boundary (see Figure 5-19)

Note

CC_MSTR_MTU (=max {64, CC_MSTR_BURST_LEN * (MSTR_BUS_DATA_WIDTH / 8)})

- Master Maximum Transfer Unit. The bridge master does not request or write more than CC_MSTR_MTU bytes in a single AXI burst.
- CC_MSTR_BURST_LEN is the maximum burst in beats that the bridge master interface issues. It must be between 8 and 256. It defaults to 16.

The AXI master page boundary specifies an address page boundary of either 128, 256, 512, 1K, 2K or 4K. The AXI bridge ensures that the inbound request does not cross the selected page boundary when driven onto AXI manager interface. The default of the AXI master page boundary is 4KB. It can be modified through the `CC_MSTR_PAGE_BOUNDARY_PW` configuration parameter.

Reducing Inbound Decomposition

To reduce decomposition on the inbound path, observe the following guidelines.

For read requests:

- Ensure that the remote link partner does not issue a read request TLP with a requested length greater than the `CC_MSTR_MTU`, or
- Design your application (and configure the bridge) to handle longer AXI bursts. That is, increase the value of the configuration parameter `CC_MSTR_BURST_LEN` or increase the AXI manager data bus width.

For write requests:

- Ensure that the remote link partner does not issue a write request TLP with a payload greater than the `CC_MSTR_MTU`, or
- Design your application (and configure the bridge) to handle longer AXI bursts. That is, increase the value of the configuration parameter `CC_MSTR_BURST_LEN` or increase the AXI manager data bus width.
- In summary, ensure TLP length <= `CC_MSTR_MTU` which means you must set `CC_MSTR_BURST_LEN` >= TLP length / (`MSTR_BUS_DATA_WIDTH` / 8) +1[1]

For all requests:

- Program or design your PCI Express system so that a TLP address plus the TLP length does not cross the AXI Master_Page_Boundary.

Note

- For more information on how decomposition affects performance, see "Areas for Maximizing AXI Bridge Performance" on page 736.
- The master interface asserts the `mstr_a*misc_info_last_dcmp_tlp` output on the last beat of a decomposed request. It is continuously asserted for a TLP that is not decomposed.

Attention

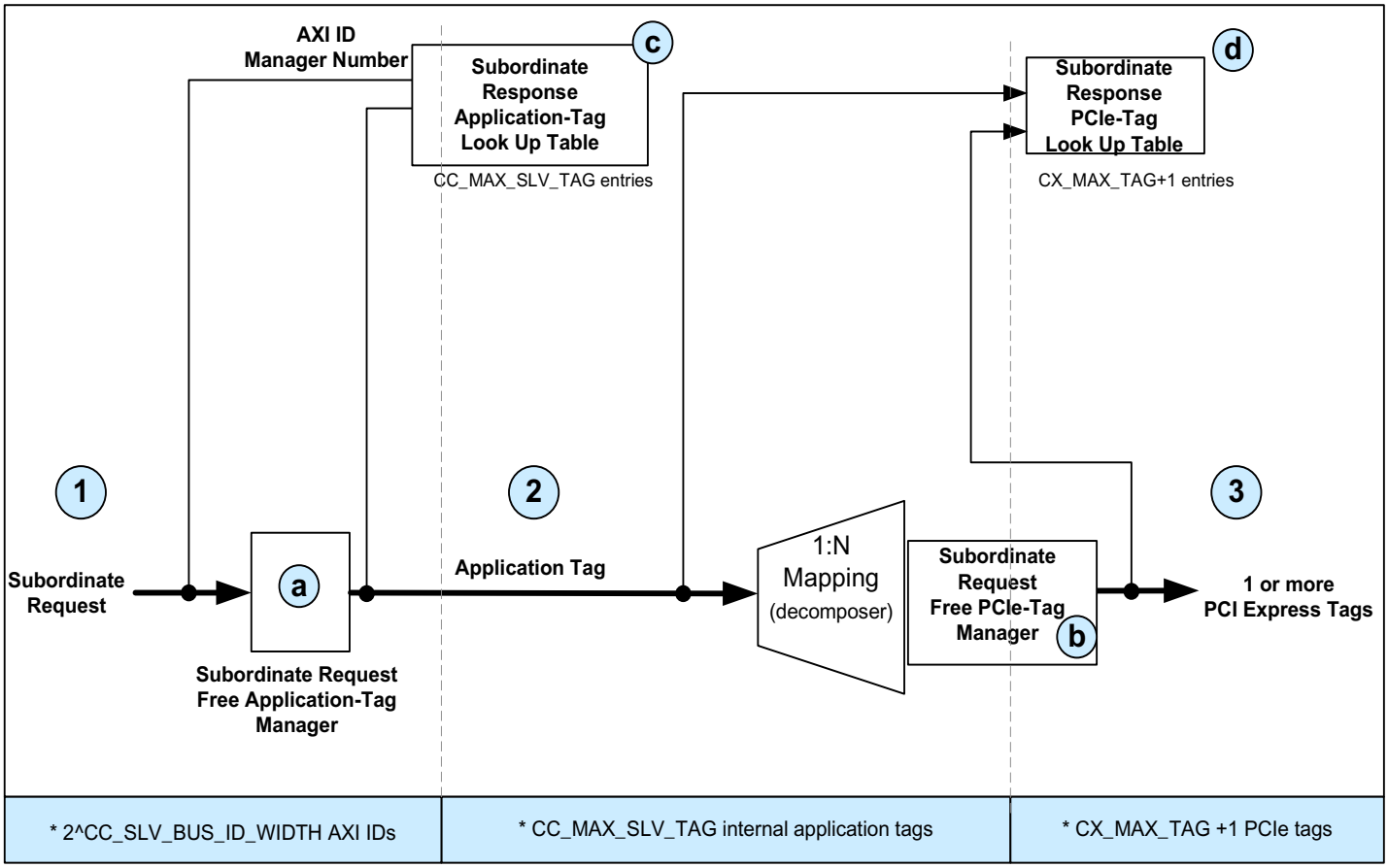
- The bridge does not handle decomposition of messages that have a payload (MsgD).
- Inbound MsgD requests bigger than CC_MSTR_MTU cause data corruption.

5.1.4.2 Outbound Bridge Tag Management

The bridge transfers IDs/tags between PCIe and AXI. Transactions on PCIe and AXI bus have separate IDs/tags and as the bridge transfers the data, it maps and keeps track of the IDs/tags on each side. The next diagram gives an overview of the outbound tag management system for non-posted requests.

1. This extra term is added to cover the scenario where the starting address is misaligned to AXI the datawidth.

Figure 5-20 Outbound Tag Management Architecture for Non-Posted Requests[12]



5.1.4.2.1 PCIe Tag Mapping

Posted requests use the tag number supplied on `slv_awmisc_info_p_tag[7:0]` which you should normally set to 0 when not relevant. When the subordinate interface converts an AXI outbound non-posted request to a PCIe request TLP, it:

- Selects a PCIe tag from the free pool of currently available PCIe tags. This is performed using a tag manager (see module b in Figure 5-20). When the pool is exhausted, no more PCIe request TLPs can be transmitted until the pool is replenished by tags from inbound PCIe CPL TLPs. For more information, see steps 2 and 3 in Figure 5-20.
- Stores the mapping from AXI request to PCIe tag, so that when the corresponding PCIe completion TLP arrives back in the future, it can be correctly associated with the originating request. This is achieved using a tag lookup table (LUT) see module d in Figure 5-20.

The process of mapping AXI manager numbers to PCIe tags is a two-step process (steps 1,2, and 3 in Figure 5-20). It consists of application tag mapping and PCIe tag mapping (as previously described). application tag mapping is an intermediate step which has to occur first.

1. *CX_MAX_TAG +1 PCIe tags: If you configure the PCIe controller with the DMA module, the pool of available tags is reduced by CC_NUM_DMA_RD_TAG.
2. *CX_MAX_SLV_TAG internal application tags: If you configure the PCIe controller with the DMA module, the default of this parameter is reduced by 50%.

5.1.4.2.2 Application Tag Mapping

In the AXI protocol, it is possible to have several AXI requests with the same AXI ID. Therefore, it is necessary for the bridge to differentiate between individual AXI requests to enable it to track them and their completions throughout the system. This is necessary to aid in reassembling[1] (recomposing) the multiple inbound completions that are returning for an original single AXI outbound non-posted request. Multiple completions can occur due to decomposition (see "Decomposition" on page 400) of the original outbound request or due to the remote link partner choosing to complete a request using multiple completions. Application tag mapping occurs through the following process:

- The bridge maps the AXI ID to an internal application tag (see module a in Figure 5-20). This application tag identifies the reserved space in the subordinate response composer for the expected returning completions.
- The bridge stores the mapping from the AXI ID to the internal application tag, so that when the corresponding completion arrives back in the future, it can be correctly associated with the originating request. This is achieved using a tag lookup table (LUT) see module c in Figure 5-20.
- When decomposition occurs, each PCIe TLP is assigned a unique PCIe tag from the pool of `CX_MAX_TAG`[2] tags (see module b in Figure 5-20). Each of these PCIe tags is associated with the same internal application tag and this information is stored in a LUT (see module c in Figure 5-20).

If at any time, all PCIe tags or internal application tags are consumed, then the AXI bridge subordinate interface does not accept any new transactions and applies back-pressure to the AXI fabric.

5.1.4.2.3 Increasing Size of AXI Subordinate ID Bus

The Slave Request Free Application-TAG Manager (see module a in Figure 5-20) memory has `CC_MAX_SLV_TAG` entries, each of width `CC_SLV_BUS_ID_WIDTH`. You can increase or decrease `CC_SLV_BUS_ID_WIDTH` and this changes the width of each entry in this memory, but it does not change the number of available application tags. However, if you increase `CC_SLV_BUS_ID_WIDTH` then your application master can issue more than `CC_MAX_SLV_TAG` outstanding outbound AXI transactions, but the AXI bridge subordinate interface does not accept these extra transactions and applies back-pressure to the AXI fabric.

For example, if the PCI Express controller AXI subordinate is configured to support eight outstanding outbound AXI transactions (`CC_MAX_SLV_TAG` =8) then `CC_SLV_BUS_ID_WIDTH` defaults to log2(8) which is 3. Now, suppose you change `CC_SLV_BUS_ID_WIDTH` to five bits. This does not mean that you can now support 32 (2^5) outstanding outbound AXI transactions. The controller AXI subordinate can still only support 8 outstanding outbound AXI transactions.

5.1.4.3 Inbound Bridge Tag Management

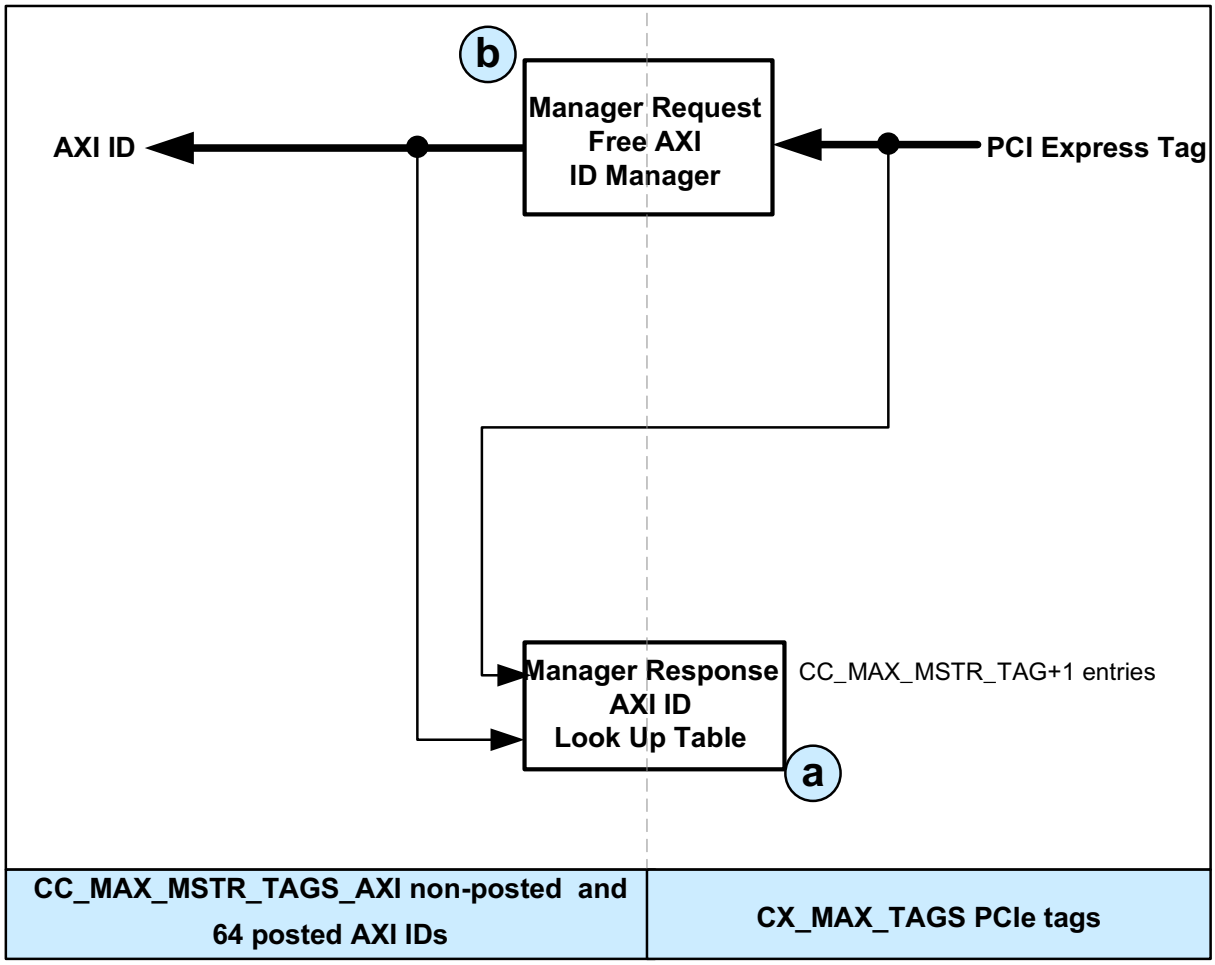
The bridge transfers IDs/tags between PCIe and AXI. Transactions on PCIe and AXI bus have separate IDs/tags and as the bridge transfers the data, it maps and keeps track of the IDs/tags on each side. When an inbound PCIe request TLP is converted to an AXI request at the master interface, it is necessary to:

- Select an AXI ID from the free pool of currently available AXI IDs.
  - For non-posted requests, this is done using an ID manager see module b in Figure 5-21.

1. But not necessarily re-ordering. For more information, see "Advanced Information: Advanced Ordering Information" on page 703.
2. If you configure the PCIe controller with the DMA module, the pool of available tags is reduced by CC_NUM_DMA_RD_TAG.

  - Posted requests always use ID 0. Non-posted requests also use this ID.
- When the pool is exhausted, no more requests can be launched onto the AXI fabric until the pool is replenished by IDs. An ID is released to the pool when the completion has arrived back from your application AXI manager. In the case where the inbound PCIe TLP request is decomposed, then the ID is released when all the individual completions have arrived back.
- Store the mapping from PCIe tag to AXI request ID to, so that when the corresponding AXI response arrives back in the future, it can be correctly associated with the originating request. This is achieved using an ID lookup table (LUT) see module a in Figure 5-21.

Figure 5-21 Inbound Tag Management Architecture Decomposition



"Decomposition" on page 400 does not affect AXI ID usage. When decomposition is occurring, then the same ID is used for each AXI request that is generated by decomposition. Each response is identified by the AXI bridge using its response ID. The AXI bridge master interface issues all:

- Posted requests with an ID of '0'
- Non-posted read requests with an ID from 0 to `CC_MAX_MSTR_TAGS_AXI-1`
- Non-posted write requests with an ID from `CC_MAX_MSTR_TAGS_AXI` to `2 * CC_MAX_MSTR_TAGS_AXI-1`

It is possible to have `CC_MAX_MSTR_TAGS_AXI` outstanding non-posted requests and 32/64/128[1] posted requests at the same time.

Note

- When you enable the DMA (CC_DMA_ENABLE =1); the allocation of master IDs changes. For more information, see "DMA Ordering" on page 505.
- When you enable the second master (CC_HP_MASTER =1); the allocation of master IDs changes.
  - It is possible to have CC_MAX_MSTRH_TAGS_AXI simultaneous NP transactions on the high priority path (master #2) and CC_MAX_MSTR_TAGS_AXI - CC_MAX_MSTRH_TAGS_AXI simultaneous NP transactions on the low priority path (master #1).
  - The maximum number of outstanding Posted transactions allowed on each path is RADM_-MAX_OUTSTD_P_REQ/2; when this threshold is reached the progression of P transactions is halted.

5.1.4.4 AXI5 Read Data Chunking

The AXI5 Read Data Chunking is an optional feature that allows the application subordinate to return Read Data in unordered 16 byte chunks optimizing its buffering logic. You can enable this feature by setting the CX_AXI_RCHUNK_ENABLE parameter to '1'.

When you set the parameters MSTR_FIXED_SIZE_ENABLE =1 and MSTR_SUB_BUS_SIZE_READ_ENABLE =0, the bridge realigns mstr_araddr to multiples of 16 for inbound MemRd requests. This allows more responses to be chunked, but modifies the original PCIe request address by driving the lowest 4 bits to 1'b0. You can disable this behavior by setting the parameter CX_AXI_RCHUNK_NO_PREFETCH to '1'.

Upon reset, Read Data Chunking is enabled by default (the controller drives mstr_archunken whenever a chunked response is possible). However, you can disable Read Data Chunking by setting the AXIG_CTRL_RCHUNK_DISABLE field of the AXIG_CTRL register to '1'.

Table 5-20 Requirements for Chunking AXI Transactions

Signal	Condition
mstr_arsnoop	The transaction type must be ReadNoSnoop, or ReadOnce, or ReadOnceCleanInvalid, or ReadOnceMakeInvalid. The AXI bridge always does this.
mstr_arbust	The burst type must be INCR or WRAP. The AXI bridge always does this.
mstr_araddr	■ The address must be aligned to 16 bytes. ■ In some configurations, the AXI-G Bridge realigns the address for inbound MemRd. ■ CfgRd and IORd requests are never realigned.
mstr_arsize, mstr_arlen	■ The ARSIZE must be equal to data bus width, or ARLEN =1 beat and ARSIZE >=128 bits. The AXI bridge always does this when MSTR_FIXED_SIZE_ENABLE =1 and MSTR_SUB_BUS_SIZE_READ_ENABLE =0. ■ When MSTR_FIXED_SIZE_ENABLE !=1 or MSTR_SUB_BUS_SIZE_READ_ENABLE !=0, the AXI bridge does not enforce this condition. It is still possible to follow this condition if PCIe Inbound Requests are formatted accordingly.
mstr_arid	Must be unique among the outstanding read transactions. The AXI-G bridge always does this.

1. 32 for 32-bit controller data width; 64 for 64-bit and 128-bit controller data widths; 128 for controller data width greater than 128.

5.1.4.4.1 Limitations

You must consider the configuration-level limitations mentioned in Table 5-21 to use Read Data Chunking feature.

Table 5-21 Read Data Chunking Configuration Limitations

Limitation	Affected Configuration	Description
NCBE Decomposition	CC_AXI_NCBE_NODECOMP ==0	Not supported. You must enable CC_AXI_NCBE_NODECOMP for Read Data Chunking support.
Low Master Data Bus Width	MASTER_BUS_DATA_WIDTH <=64	Not supported. You must set the MASTER_BUS_DATA_WIDTH to 128, or 256, or 512 for Read Data Chunking support.
Interface parity/ECC protection	CX_AXI_IF_TYPE_OF_PROT !=0	mstr_archunken, mstr_rchunkv, mstr_rchunknum, and mstr_rchunkstrb interface signals are not protected.

Table 5-22 Read Data Chunking Performance Caveats

Limitation	Description
Performance limitations in TRGT1 request handling	■ When you enable the CX_AXI_RCHUNK_ENABLE parameter, the AXI Bridge Master generates unique AXI IDs for all inbound non-posted requests (AR and AW channels). ■ When the AXI Bridge exhausts all AXI IDs, it stops accepting non-posted requests from RTRGT/RBYP. To ensure that the AXI Bridge does not run out of AXI IDs before it runs out of Master Tags, avoid inbound decomposition. For more information, see "AXI Decomposition Rules" on page 435 ■ You can set the maximum number of outstanding AXI IDs using the CC_MAX_M-STR_TAGS_AXI parameter. However, the AXI-G Bridge does not ensure timely request forwarding when above (CC_MAX_MSTR_TAGS_AXI - 3) outstanding AXI IDs. Furthermore, it is not possible to reach above (CC_MAX_MSTR_TAGS_AXI - 1) outstanding requests of any one type (writes or reads). ■ This limitation does not affect Posted requests.
Unchunkable requests	Some requests cannot be chunked (mstr_archunken =0), and are returned in-order (mstr_rchunkv =0) by the application subordinate. This applies to AXI requests with addresses not aligned to 16 bytes. For more information, see "AXI Manager Interface Signals" in "Signal Descriptions" chapter of the *Reference Manual*.
Increased Bridge Inbound CPL latency	When you enable CX_AXI_RCHUNK_ENABLE, the AXI Bridge Inbound Master Completion Buffer uses Store-and-Forward buffering instead of Cut-Through buffering. This results in increased latency for CPLs with large amounts of data.

5.2 ACE-Lite Support in PCIe Controller with AMBA Bridge

The controller supports the AXI4 protocol with additional support for the ACE-Lite protocol. To enable these features, you must set `AMBA_INTERFACE =3 or 4` and `CC_ACELITE_ENABLE =1`.

ACE-Lite Features and Limitations

- The AXI bridge subordinate does not implement ACE; it ignores the value of the DOMAIN, BAR and SNOOP fields; barrier transactions are not supported.
- The AXI bridge master interface generates requests that are dependent upon the NS and TH bits as shown in Table 5-23.

Table 5-23 ACE-Lite Master Interface Request Types and Controls

Cache Coherency	Request Type	Peripheral Type	Memory Type[a] 0: Peripheral 1: Main Memory	NS	TH	AxSNOOP	AxBAR	AxDOMAIN	Shareability Domain	Memory Type
Non-coherent	Message	Peripheral	X	X	X	0000	00	11	System	Device, Non- bufferable
Non-coherent	Read or Write	Peripheral	0	X	X	0000	00	11	System	Device, Non- bufferable
Non-coherent	Read or Write	Memory	1	1	X	0000	00	11	System	Normal Non- cacheable Non- bufferable
Coherent	Read	Memory	1	0	0	0000	00	10	Outer Shareable	Write back, No- allocate
Coherent	Write	Memory	1	0	0	0000	00	10	Outer Shareable	Write back, No- allocate
Coherent	Read	Memory	1	0	1	0000	00	10	Outer Shareable	Write back, Read- allocate
Coherent	Write	Memory	1	0	1	0000	00	10	Outer Shareable	Write back, Write- allocate

a. Set by the CFG_MEMTYPE_VALUE field of the COHERENCY_CONTROL_1_OFF register.

- The master interface always sets `AxBAR` and `AxSNOOP` to ‘0’ so that requests are restricted to ReadNoSnoop, WriteNoSnoop, ReadOnce, and WriteUnique.
- You can override the value of the `DOMAIN` field, except for message requests where it is always 11.
- The master does not support barriers.
- Cache coherency is not optional, and the master always uses the NS bit, unless you override the `DOMAIN` signals.

5.3 DTIM Support in PCIe Controller with AMBA Bridge

The DTI Master (DTIM) is used to interface to an ARM SMMU which provides address translation functions for memory accessing IO coherent masters. The DTIM complies with the DTI-ATS protocols to send page and translation request messages to the SMMU and to receive invalidation request messages from the SMMU. The DTIM uses bi-directional AXI4 Streaming interfaces to transport these messages.

The AXI Bridge DTI Master of the PCIe controller complies with the A*RM AMBA Distributed Translation Interface (DTI) Protocol Specification, Edition 2*. You can use the `CC_DTIM_ENABLE` parameter to enable DTIM support in the PCIe controller.

5.3.1 Features

- Adheres to ARM's DTI-ATS protocol to request and invalidate address translations between the PCIe Root Complex and a System Memory Management Unit (SMMU).
- Transports the DTI protocol messages using the AXI4-Stream Interface; messages sent from the DTI Master (DTIM) to the DTI Slave (DTIS) and vice-versa.
  - The PCIe RP is the DTIM, the SMMU is the DTIS.
  - The DTIM contains an AXI4-Stream Master interface and an AXI4-Stream Subordinate interface.
- The PCIe RP implements a single DTI-ATS interface to the TCU for one or more ACE-Lite AXI4 Masters. Each AXI4 Master interfaces to a SMMU TBU component.
- To enable the SMMU to provide translations the following ATS related sideband signals are available on the local PCIe AXI ACE-Lite Master interface:
  - Requester ID (Stream ID)
  - PASID (Substream ID)
  - AT

  Note: The sideband signals provided on both the AR and AW channels.
- All DTI-ATS protocol messages are supported.
- Connect/Disconnect Handshake
  - The DTIM initiates the DTI-ATS interface connection process if a page or translation request is received. The connection message includes a request for a number of translation request flow control tokens and the number of invalidate request tokens supported by the DTIM. The number of invalidate and translation request flow control tokens is configured by you at build time.
  - Your application initiates the disconnection process when all ATCs have been invalidated and disabled. The DTIM sends the Disconnect request message to the DTIS.
- Address Translations
  - The DTIM converts the PCIe ATS translation request from the ATC and forwards the request to the SMMU through the DTI-ATS interface. Translation requests may or may not include the PASID(TLP Prefix supported).
  - The translation response is sent by the SMMU for each translation request. If the translation is granted the DTI_ATS_TRANS_RESP message is converted to a PCIe Translation Completion of type CplD. If the translation request is denied the DTI_ATS_TRANS_FAULT message is converted to a PCIe Translation Completion of type Cpl.
  - The DTIM generates translation tags or IDs to correlate requests and responses. The DTIM and the AXI Manager share a common pool of IDs.

- There are no ordering guarantees between the translation requests and responses.

- Invalidations
  - The DTIM sinks invalidation requests. The resources to buffer up these requests is defined at build time. The DTIM supports upto 16 requests.
  - The DTIM ensures no more than 32 invalidation requests are outstanding per ATC; Note: Posted PCIe invalidation requests are non-blocking.
  - The DTIM filters the ATS Invalidation Requests by Root Port ID and Bus Number. The upper 16 bits represent the Root Port ID and the lower 16 bits represents the BDF. Bus identification is on the upper 8 bits of the BDF. Port ID, Secondary Bus Number and Subordinate Bus Number fields are passed to the DTIM at build time. To deem an ATS Invalidation Request as valid:
    - The request Root Port ID must match with the Root Port ID of the PCIe component attached to the DTIM.
    - The request Bus Number must be within the range of Secondary Bus Number and the Subordinate Bus Number.
  - The DTIM generates ITAGs to correlate PCIe requests with completions; Note: there are no ordering guarantees between the PCIe requests and the completions.
  - More than one PCIe translation completion can be received per invalidation request. In addition, translation completions can coalesce responses for different ITAGs. The DTIM keeps track of the translation completions per transaction ID (BDF, ITAG}. A invalidation response or SYNC acknowledge is only sent to the SMMU when the SYNC request message, DTI_ATS_SYNC_REQ, is received.
  - The DTI-ATS protocol limits the number of outstanding SYNC requests to a single request. When the DTI_ATS_SYNC_REQ is received the DTIM returns the DTI_ATS_SYNC_ACK after all outstanding PCIe invalidation completions prior to SYNC request have been received and all outstanding memory writes associated with the invalidated memory ranges have completed. The ATC guarantees all memory reads have completed before sending the invalidate completions.
- Page Request Interface
  - The DTIM supports PCIe PRI page requests and responses.PRI enables the EP to use unpinned memory for direct memory access, that is it allows the EP to used shared virtual memory.
- Error Handling
  - The DTIM has its own specific mechanism to notify the system that an error has occurred. Both pin and MSI interrupt mechanisms are supported.
- By default, ATS and Page Request Services are enabled when the DTIM is enabled. You must enable TLP Prefixing to support PASID.
- RAS Data Protection
  - Even and odd data parity on the data and address paths are supported.
  - ECC detection and correction on the DTI Master AXI4-Stream Subordinate Invalidate Request Queue RAM is supported.

5.3.2 Limitations

- The AXI Manager/Subordinate address must be 64-bits.
- The AXI Manager/Subordinate data width must be >= 64-bits.
- The data width of the DTI-ATS interface is fixed at 160 bits.All messages complete in a single cycle.

- ATC Requests for multiple translations are not supported. Ifsuch a request is received the DTIM sends the first translation request only.
- The DTIM and the AXI4 Master share the same PCIe Non-Posted memory transaction ID pool. The maximum number of PCIe memory reads offloaded from the Rx Queues is fixed at 256, that is the maximum number of outstanding DTI-ATS translations requests that can be sent to the SMMU.

5.3.3 AXI4-Stream Interface

The DTI Master uses the AXI4-Stream as the transport medium. An AXI4-Stream link for DTI consists of two AXI4-Stream interfaces, one for each direction. The mapping of AXI4-Stream signals to the DTI protocol is described in the *ARM AMBA Distributed Translation Interface (DTI) Protocol Specification, Edition 2*. For more information on AXI4-Stream interface signal descriptions, see "Signal Descriptions" chapter of the *Reference Manual*.

5.3.4 RAMs

Table 5-24 identifies the additional RAMs included in the AXI Bridge when DTI is enabled.

Table 5-24 AXI Bridge DTI RAMs

RAM Name	Clock	Signal Prefix
DTI Master AXI4-Stream Subordinate Invalidate Request Queue RAM	Port A: mstr_aclk	dtim_reqq_*
	Port B: mstr_aclk	

For more information on the RAM interface signals, see "Signal Descriptions" chapter of the *Reference Manual*.

5.3.5 Programming Model

For a detailed description of the DTI control and status registers, see "Port Logic Descriptions" section in the "Register Descriptions" chapter of the *Reference Manual*.

5.3.6 Error Handling

The following errors are detected by the DTIM:

- **ATC Invalidation Timeout:** This error occurs when an Endpoint function (ATC) does not return an invalidate completion to an outstanding invalidation request within the designated timeout period. The DTIM implements a programmable timer per outstanding invalidate request to detect timeouts. If a function fails to return an invalidation completion to an outstanding invalidation request within the timeout period, the DTIM is deemed to be deadlocked, that is, the sync acknowledge to a sync request issued after the outstanding invalidate request, is not be returned. The DTIM implements a programmable timer per outstanding invalidate request to detect timeouts, so that DTIM deadlock can be avoided. The timer can be programmed using the `DTIM_CTRL5_IREQ_TMR_TRGT_TO` field of the `DTIM_CTRL5_OFF` register.
- **ATC Invalidation Error:** This error occurs when an Endpoint function (ATC) returns an ERR_NON-FATAL error message. This can happen in several scenarios where,
  - An invalidation request is sent to a function with ATS disabled (functions that do not support ATS treats an Invalidate Request as UR), or
  - The request is sent with an out-of-range PASID value

- **Unexpected Invalidate Completion:** This error occurs when an unexpected invalidation completion is received.
- **Translation ID Mismatch:** This error occurs when the TRANSLATION_ID field in the DTIM translation response DTI_ATS_TRANS_RESP, does not correspond to an outstanding DTIM translation request.
- **Illegal Invalidate Requests:** This error occurs when the OPERATION field of the DTIM invalidation request DTI_ATS_INV_REQ, is illegal.
- **Output Address Size (OAS) mismatch:** The Output Address Size (OAS) is the maximum address size permitted for translated addresses. The translation response returns the range of addresses or region size for which the translation is valid. If the following rules are violated, an OAS mismatch error occurs.
  - For normal translations (BYPASS =0) the range of address must not exceed the OAS
  - If translation is bypassed (BYPASS =1) the range of addresses must match the OAS
  - The translated address must not exceed the OAS
- **Connect Deny:** This error occurs if connection is denied when a page or translation request is received.
- **PRI Not Supported:** This error occurs if a page request is received when PRI is not supported by the SMMU. The `DTIM_DIAG_SUP_PRI` field of the `DTIM_DIAG_OFF` register indicates whether SMMU supports DTI-ATS Page Request/Responses or not.
- **PCI Specific ATS Errors**
  - The Length field of the PCIe translation request, PCIE_ATS_TRANS_REQ, is
    - Not an even value, or less than 2 DWs, or greater than RCB/4 DWs
    - The native controller discards the PCIe translation request
  - The native controller discards the PCIe page request if the page request message is not received in traffic class 0.

Table 5-25 DTIM Error Handling Mechanism

Error	Controller Action
ATC Invalidation Error	Your application must handle the ERR_NONFATAL error as described in "ATC Invalidate Error Handling" on page 451.
Unexpected Invalidate Completions	■ The BDF and the ITAG associated with the first unexpected invalidation completion is recorded in `DTIM_ERR_LOG0_OFF` register. ■ Unexpected invalidation completions are discarded. ■ The error event is recorded in `DTIM_INT_STATUS_OFF` register; the wire interrupt or MSI, if enabled, is sent to your application. For more information, see "Interrupt Handling" on page 453.
Translation ID Mismatch	■ The first illegal translation ID mismatch is recorded in `DTIM_ERR_LOG2_OFF` register. ■ The translation response is discarded. ■ The first error event is recorded in `DTIM_INT_STATUS_OFF`; the wire interrupt or MSI, if enabled, is sent to your application. For more information, see "Interrupt Handling" on page 453. *Note:* If the original request does not receive a response then the RP timeouts (TRGT_CPL_LUT timeout) and the EP timeouts the completion and send an error message to the host. However, it is expected that the DTIS must always return a response, failure to do so requires the controller to be reset.

Error	Controller Action
Illegal Invalidate Requests	■ The first illegal OPERATION code is recorded in DTIM_ERR_LOG1_OFF register. ■ The invalidation requests are discarded and acknowledge returned. ■ The error event is recorded in `DTIM_INT_STATUS_OFF` register; the wire interrupt or MSI, if enabled, is sent to your application. For more information, see "Interrupt Handling" on page 453.
OAS Size Mismatch	■ The error event is recorded in `DTIM_INT_STATUS_OFF` register; the wire interrupt or MSI, if enabled, is sent to your application.Formore information, see "Interrupt Handling" on page 453. ■ A PCI Translation Completion is returned with "Completer Abort" status. *Note:* If reserved OAS values are received in the translation response, the DTIM defaults to the minimum value of 4GB; an error is not flagged.
Connect Deny	■ The DTI-ATS interface transitions from REQ_CONNECTED state to DISCONNECTED state. ■ If the connect request is triggered by a page request, the DTIM ❑ Returns a page response with "Response Failure" status, and ❑ Discards the page request if the request is the last page request associated with the page group index; If it is not, then only the page request is discarded, page response is not sent. *Note:* Page responses do not include a PASID ■ If a page request prior to the last page request in the group is denied by the SMMU, but the last page request accepted, then all translations may not be available. In this case the Endpoint function can re-send the page requests if subsequent translation requests fail. ■ If the connect request is triggered by a translation request, the DTIM returns a translation completion with "Completer Abort" status and discards the translation request. ■ Each translation or page request is processed in the following sequence: a. Multiple translation or page requests received b. Connect request message sent to SMMU c. Wait for SMMU to respond d. If connection denied, discard translation request, return "Completer Abort" status e. Process next pending translation request, repeat steps b to d f. When all pending translation requests are denied, process page requests *Note:* There is no guarantee on the order in which the translation requests, page requests, and error responses are processed. Page requests (Posted) can overtake translation requests (Non-Posted).
PRI Not Supported	■ Returns a page response with "Response Failure" status, and ■ Discards the page request if the request is the last page request associated with the page group index; If it is not, then only the page request is discarded, page response is not sent. *Note:* Page responses do not include a PASID
PCI Specific ATS Errors	■ These errors are handled by the Root Port's PCI Express Advanced Error Reporting Capability.

5.3.6.1 ATC Invalidate Error Handling

If an Endpoint function (ATC) returns an ERR_NONFATAL error message while invalidate requests to the same Endpoint function are outstanding,

1. Your application must handle the ERR_NONFATAL error and flush the outstanding requests to the Endpoint function from the DTIM invalidation request queue. The DTIM has no knowledge that the Endpoint function has returned an ERR_NONFATAL response, it automatically timeouts the

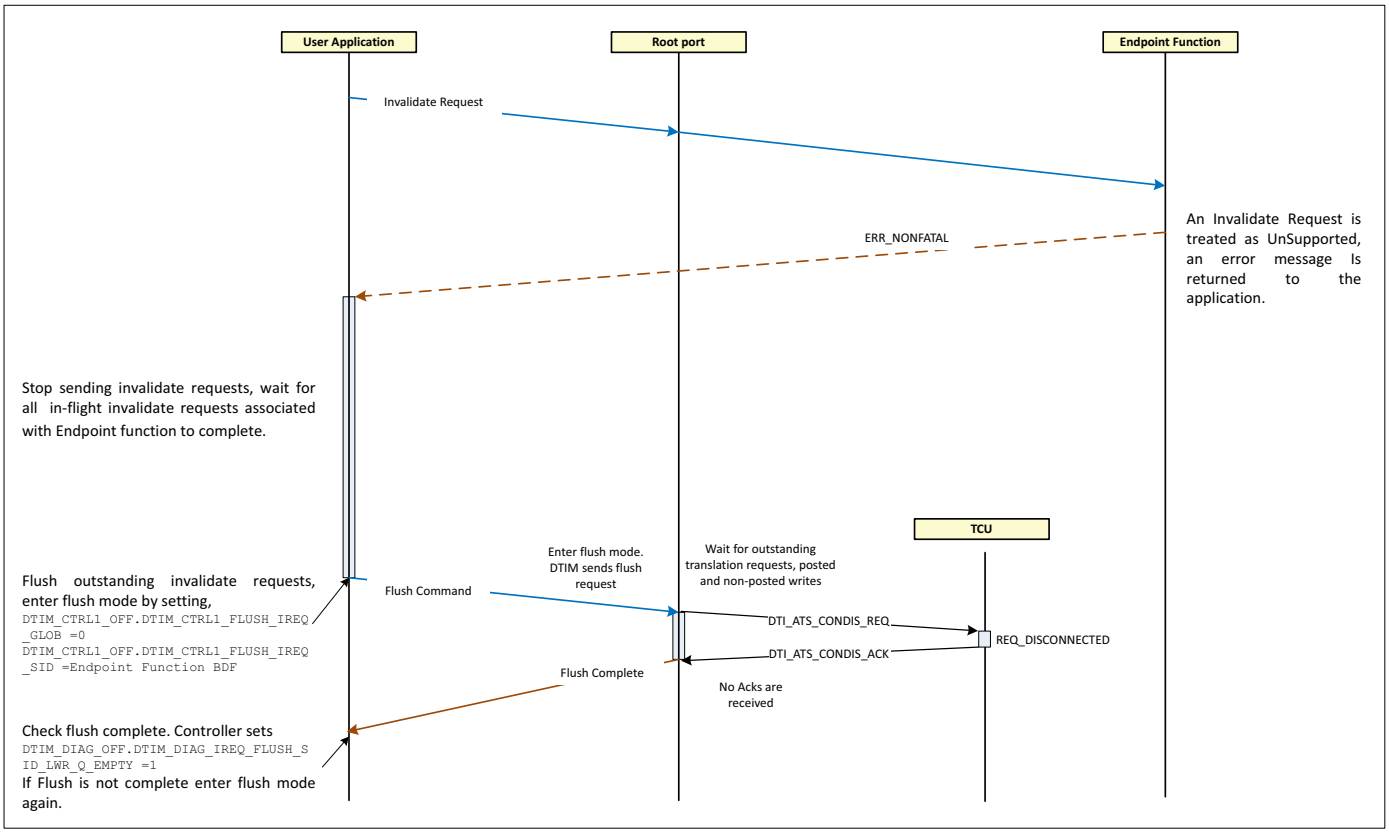
outstanding invalidate requests. DTIM sends a sync error message to the SMMU, if a sync request is pending, and the invalidate requests are timed out by the controller.

2. Your application can flush the invalidate requests before it is timed out by the controller to prevent the DTIM from sending a sync error to the SMMU. To flush invalidate requests associated with a particular Endpoint function your application must set the `DTIM_CTRL0_IREQ_SW_FLUSH_EN` field of the `DTIM_CTRL0_OFF` register to '1' and then trigger the flush by setting,
   a. `DTIM_CTRL1_OFF.DTIM_CTRL1_FLUSH_IREQ_SID =Endpoint Function BDF`, and
   b. `DTIM_CTRL1_OFF.DTIM_CTRL1_FLUSH_IREQ_GLOB =0`
3. Prior to flushing the invalidate request queue your application must stop sending new requests to the Endpoint function and wait for all in-flight invalidate requests to propagate to the DTIM. The controller iterates through all entries in the queue, flushing all invalidate requests that match `DTIM_CTRL1-FLUSH_IREQ_SID`. The flushing of the invalidate request queue is asynchronous to the reception of new invalidate requests from the SMMU. To ensure all in-flight invalidate requests are flushed, your application may need to trigger multiple flush sequences if in-flight requests are accepted by the DTIM after the flush sequence completes. Figure 5-22 describes an example of an invalidate request queue flush sequence triggered by your application.
4. The DTI-ATS master (RP) initiates the disconnect request asynchronous of outstanding and in-flight invalidate or sync requests. It knows when the outstanding translations are complete. You must re-initialize every device with an ATC to manually flush all translations from the system. The RP flushing works as follows:
   a. After the link is lost, the device enters the flush mode (AXI manager asserts flush request). Wait for the following requests to complete:
      i. All outstanding DTI translation completions
      ii. All AXI manager posted or non-posted requests
   b. AXI bridge discards any pending requests, asserts the subordinate flush request, waits for the flush Ack.
   c. DTIM sends DTI_ATS_CONDIS_REQ to the DTIS. DTI-ATS interface enters REQ_DISCONNECT state and waits for the DTI_ATS_CONDIS_ACK message.
      i. After the DTIM sends the DTI_ATS_CONDIS_REQ, the DTIM must not return any Acks to outstanding invalidate or sync requests (any requests accepted by the DTIM prior to sending DTI_ATS_CONDIS_REQ).
      ii. If the DTIM receives any in-flight invalidate or sync requests it discards these requests and does not return an Ack (DTIM drops the requests through the trap door).
   d. The DTIS then sends the DTI_ATS_CONDIS_ACK and the DTI-ATS interface transitions to the DISCONNECTED state. At this point the DTIS cannot send any more traffic and the DTIM can now be reset. The DTIM returns the flush Ack and the reset proceeds.

Note When the DTIM enters flush mode the t_{ready} is not de-asserted, the DTIS continues to accept messages. When the interface moves to the REQ_DISCONNECTED state the DTIM just accepts and discards any new requests.

5. The following diagnostics are supported by the controller:
   a. The controller sets the flag `DTIM_DIAG_OFF.DTIM_DIAG_IREQ_FLUSH_DONE =1` when the flush completes. `DTIM_DIAG_IREQ_FLUSH_DONE` is cleared by the controller when your application triggers the flush sequence.
   b. When the invalidate request queue flush sequence targeting an Endpoint function is triggered `DTIM_DIAG_OFF.DTIM_DIAG_IREQ_FLUSH_SID_LWR_Q_EMPTY` is set to '1'. If subsequent in-flight invalidate requests targeting the same Endpoint function are received after the flush sequence completes, `DTIM_DIAG_IREQ_FLUSH_SID_LWR_Q_EMPTY` is cleared.
   c. When all outstanding invalidate requests are flushed `DTIM_DIAG_OFF.DTIM_DIAG_IREQ_ORDRQ_EMPTY` is set to '1'. If subsequent in-flight invalidate requests are received, `DTIM_DIAG_IREQ_ORDRQ_EMPTY` is cleared.

Figure 5-22 Invalidate Request Queue Flush Sequence



5.3.6.2 Interrupt Handling

- The advanced error reporting capability associated with the Root Port not used to report DTIM specific errors to your application.
- The DTIM supports wired and MSI interrupt generation models. MSI-X interrupts are not supported.
- The DTIM supports level triggered wired interrupts. There is one interrupt per source and a common interrupt pin `dtim_int`.

- If the interrupt source is enabled, then by default the wired interrupt is asserted. If the MSI address field is non-zero, then an MSI is sent to your application through the AXI Manager.
- There is an interrupt mask per interrupt source, supporting an MSI interrupt only model if the MSI is configured, or a polling model if the MSI is not configured. Both wired and MSI interrupts can be enabled simultaneously.
- Each interrupt source sets a corresponding bit in the interrupt status register. It is your application's responsibility to clear the status by writing the corresponding interrupt clear bit.
- By default, the MSI is disabled and wire interrupts are enabled.
- The MSI generation is local to the DTIM and is configured through the DBI. The MSIs are sent through the AXI Manager.
- The MSI contains the,
  - Target memory address (64-bit DW aligned address supported), and
  - Data value to be written to that address (32-bit data pattern)
- The MSI address or Interrupt Memory Write Address register serves as the target address for the MSI memory write. The MSI data or Interrupt Memory Write Data register contains the MSI payload.

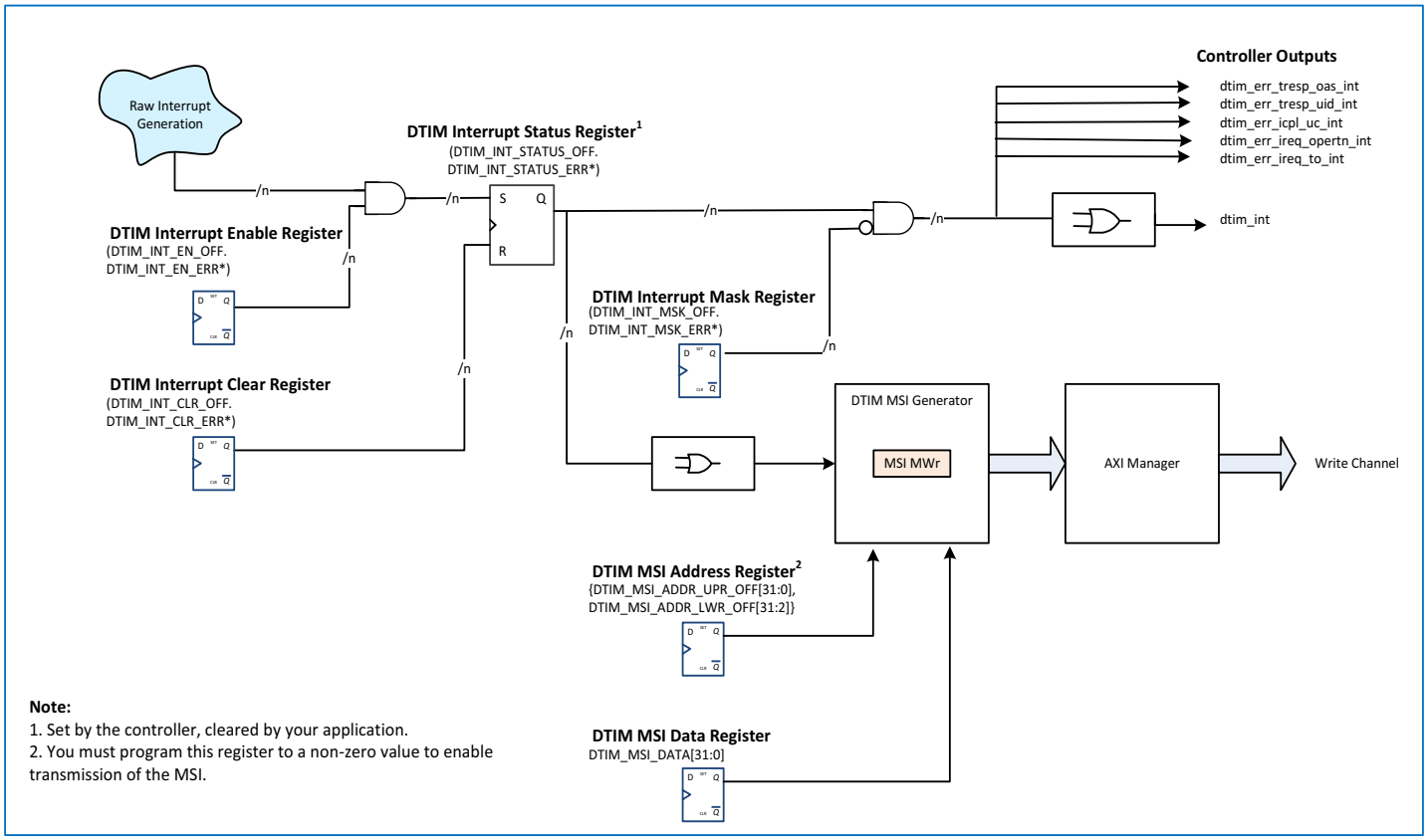
Figure 5-23 DTIM Message Signal Interrupt Handling

				3									2							1								0				Byte
MWr	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
31	FMT=010			TYPE=00000					RSVD	TC=0			RSVD	IDO=0	RSVD		TD=0	EP=0	RO=0	NS=0	AT=0		Length = 0x01										0
63	REQUESTER_ID																TAG=0								LBE=0x0				FBE=0xF				32
95	DTIM_IMWR_ADDR_UPR																																64
127	DTIM_IMWR_ADDR_LWR																														2'b0		96
159	DTIM_IMWR_DATA																																128

- The headers fields of the MSI Write conforms to the following rules:
  - Header fields "No Snoop" and "Relaxed Ordering" field areset to 0
  - The Requester ID is the ID of the Root Port
  - The Length field indicates a payload size of 1 DW
  - The first byte enable must be all ones
  - The last byte enables must be all zeros

  Figure 5-24 describes the interrupt mechanism used to handle DTIM errors.

Figure 5-24 DTIM Interrupt Handling Mechanism



5.4 LTIM, Order Buffer, and MSI-GIC Interface

In this section, the following topics are discussed:

- "Overview and Supported Features"
- "Local Translation Interface Master (LTIM)" on page 458
- "Order Buffer" on page 459
- "MSI-GIC" on page 460

5.4.1 Overview and Supported Features

Local Translation Interface (LTI) is a point-to-point protocol and defines the communication between an IO device and Translation Buffer Unit (TBU). The IO device includes the LTIM module and TBU includes the Local Translation Interface Slave (LTIS) module.

The MSI-GIC interface is a direct AMBA AXI4 stream-based connection between the Root Port and the GIC (Generic Interrupt Controller). It provides a low latency interface to deliver downstream devices message signaled interrupts (MSI) or Root Port generated MSI.

Figure 5-25 shows the overview of LTIM and MSI-GIC interface.

Figure 5-25 Overview of LTIM, Order Buffer, and MSI-GIC Interface

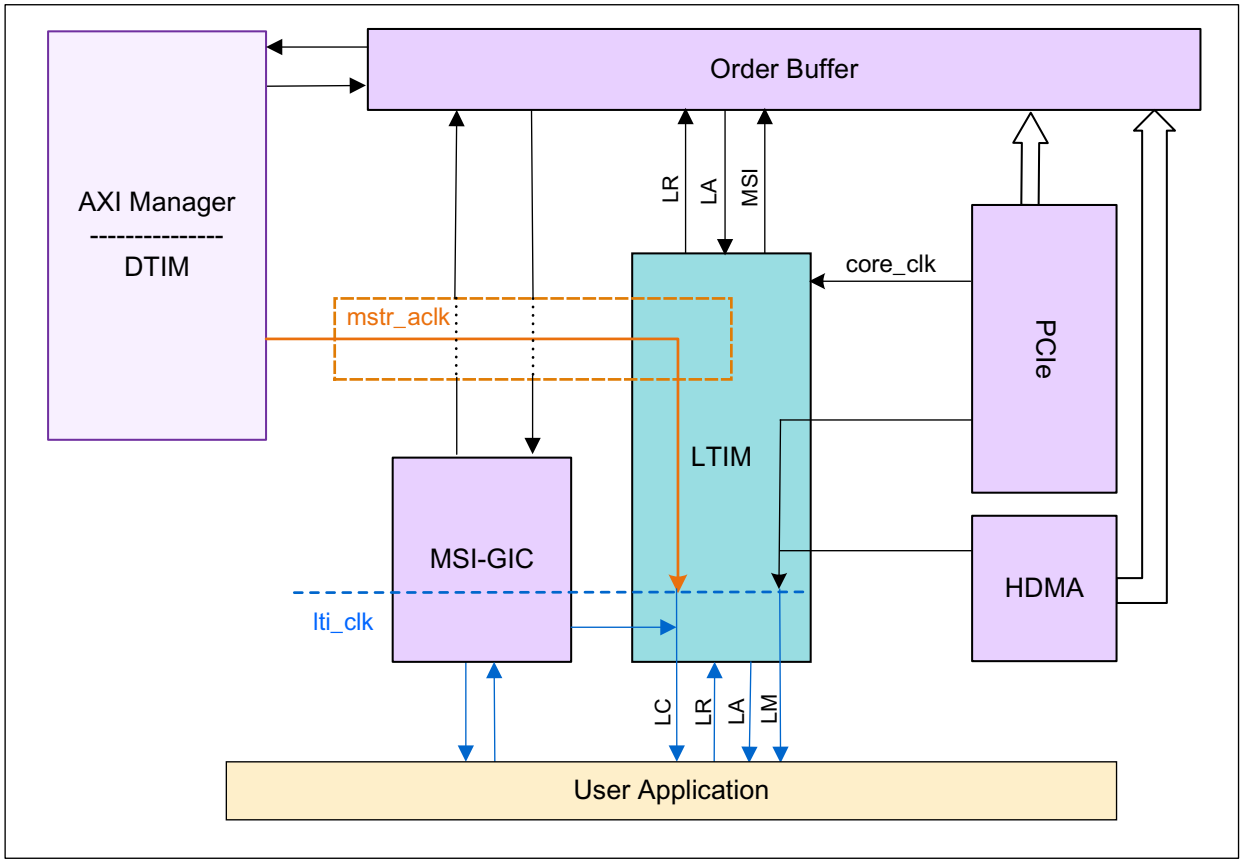


Table 5-26 shows the LTIM, Order Buffer, and MSI-GIC supported features.

Table 5-26 LTIM, Order Buffer, and MSI-GIC Interface Features

Feature	Supported	Notes
Dual Mode (CC_DEVICE_TYPE =2)	Yes	Applicable only in host mode
DW PCIe EP	No	
HDMA	Yes	
eDMA	No	
Native interface	No	
AXI5 ACE-Lite	Yes	
AXI5 AtomicOps	Yes	
AXI5 Unique ID	Yes	
AXI5 Read Data Chunking	Yes	
Multiple AXI Managers	No	
CCIX	No	
CXL	Yes	
DTI Master	Yes	
SMMU In-line Integration	Yes	Classical PCIe RC to SMMU topology
SMMU Look-Aside Integration	Yes	This is the subject of the LTI module
SMMU Cached Integration	Yes	The DTI-ATS translation provides this functionality
Supported Translation Flows: ■ Stall ■ No Stall ■ ATST ■ PRI	 No Yes Yes No	 Not supported by the PCIe protocol Default for PCIe untranslated traffic Default for PCIe translated traffic This allows the ATS PRI to load pages before a known traffic, such as DMA, minimizing translation latency. A device using this flow uses DTI-ATS for PRI requests only and does not make any ATS requests. This refers only to LTI PRI flow, not to DTI-ATS flow.
LTI requests issued before the ordering point	Yes[a]	Minimizes latency and avoids deadlock scenarios
Translation latency buffering	Yes	Buffer depth is a function of the LTI latency. Supports the worst-case scenario of 1us for all transactions
Out of order translation response	Yes	Re-orders transactions in the latency buffering
LTI VC mapped to PCIe TC	No	Because re-order transactions are based on the LTI
LTI Translation Error map to PCIe errors	Yes	

Feature	Supported	Notes
LTI AER reporting	Yes	AER reports translation faults on PCIe Posted TLP
PASID	Yes	PASID is part of the address resolution, together with the address and requester ID information
PASID up to 20-bit support Execute and Privilege bits	Yes	
Supported LR address width: 32,36,40,42,44,48,52 bits	Yes	
MSI-GIC Delivery Interface	Yes	Supports PCIe in-band MSI TLP (no support for RP generated MSI, except LTI generated). This feature deprecates “Integrated MSI reception module to Terminate RX MSI” based on interrupt pins.
ID Based Ordering	Yes	The LTI re-order buffer can re-order TLP based on IDO attributes. The Order buffer assumes that if a traffic source uses IDO attribute, then all TLP from that source must have IDO attribute set. The AXI ID is always 0x0 for all traffic.
Relaxed Ordering	Yes	The LTI re-order buffer can re-order TLP based on RO attributes. The AXI ID is always 0x0 for all traffic.
RASDP	Yes	RASDP on the internal data-path and in all interfaces, exception on LTI and MSI-GIC interfaces
Architectural Clock Gating	Yes	

a. After LTI response, the Order buffer re-evaluates ordering based on the LTI translation availability.

5.4.2 Local Translation Interface Master (LTIM)

LTI is compliant with *AMBA LTI Protocol Specification* provides an alternative to interface the SMMU. You can enable the LTI feature by setting the `CC_LTIM_ENABLE` parameter to ‘1’. This enables re-order buffer (`CC_ORDRB_EN`) and the MSI-GIC interface (`CC_MSIGIC_EN`). The translation starts as soon as a TLP arrives. At this point the transaction diverges into two paths:

- Through the PCIe ordering rules and AMBA bridge transformation rules inside the controller
- Through the SMMU

The SMMU path is asynchronous with the controller. The controller must wait for the result of the translation before committing the transaction to the application. The translation results either in a new swap in address or on an error where the transaction must drop.

On the event of a translation page miss, the SMMU must load the translation table from the main memory and return the translation response in hundreds of clock cycles. This delay is much higher than the controller latency, creating a potential performance problem if the transactions are “strongly ordered” or if the controller does not re-order VC or relax-ordered transactions received.

Limitations

- The following TLPs do not translate through the LTI interface:
  - IO Read
  - IO Write
  - CfgRd Type 0 and Type 1
  - CfgWr Type 0 and Type 1
- Message TLP does not have an address in its definition, so by default PCIe messages do not translate through the LTI interface. However, using the ATU (`CX_INTERNAL_ATU_ENABLE =1`) to append an address to the message, allows messages to be LTI translated. After ATU processing, use `MSG_REQ` field of the `LTIM_TRANSLATION_CTRL_OFF` register to enable or disable PCIe message LTI translation.
- By default, DTI traffic is not LTI translated. Use `LTIM_TRANSLATION_CTRL_OFF` registers to enable translation. DTI message requests can be optionally LTI translated if the ATU supplied an address beforehand. However, for LTI LCCTAG transactions, the CTAG order is not related with the DTI transaction state, that is, it can return before the transactions are "globally observed".
- For Posted TLP that end with LTI translations the header information is captured in the AER. However, it is not possible to save the original TLP address that originated the translation fault.

5.4.3 Order Buffer

LTI introduces latency and returns translations out of order. While translations are outstanding, this buffer allows a continuous and concurrent stream of received packets. After the translation is available, the buffer re-orders the received TLP using PCIe RO (relaxed ordered), PCIe VC (Virtual Channel), and PCIe IDO (ID based ordering), minimizing overall latency and unnecessary stalling while receiving the traffic stream.

Note You can enable Order Buffering by setting CC_ORDRB_EN parameter to '1' without enabling LTIM (CC_LTIM_ENABLE =1).

IDO ordering has the following limitations:

- If a BDF needs IDO, then all the TLP it generates has IDO bit (Attr[2]) set and belongs to the same PCIe Traffic Class
- If a BDF needs IDO and uses PASID, then TLP it generates has IDO bit set and has a PASID prefix

For more information on IDO, see "ID-Based Ordering Usage" section in the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

You must set `ORDRB_IDO_CTRL_LOW_i.IDO_EN_i` to '1' before setting `ORDRB_IDO_CTRL_HIGH_i.PASID_MATCH_EN_i` to '1'.

Buffer Capabilities and Receiving Back-pressure

Header and Data RAM defines the Order buffer buffering capabilities:

- `CC_ORDRB_AVG_MTU * 2^(CC_ORDRB_TLP_ID_WIDTH-1)` defines the Data RAM size in bytes. Posted and Non-Posted TLP width data share this RAM.
- `2^CC_ORDRB_TLP_ID_WIDTH` gives the total number of TLP that the re-order buffer can track. This defines the depth of the Header RAM.

When the Header RAM or the Data RAM gets full, the RADM packet halt mechanism back-pressures all the VC or DMA. The packet halt mechanism only applies to the Posted and Non-Posted queues.

The Header and Data RAM space gets free when the TLP streams out of the order buffer. However, some types of packets only release the Header RAM resources when the packet is globally observed. They require an extra clock from the re-order buffer when the packet is globally observed[1]. It applies to the following types of Posted TLP when the `P_PASS_P` field of the `ORDB_CONTROL_PNP` register is not zero:

- AXI Manager RO Writes
- DTI Messages
- MSI-GIC TLP

The RADM tracks the number of outstanding Posted TLPs to inform the PM (Power Management) module that it is active. The parameter `RADM_MAX_OUTSTD_P_REQ` sizes this tracker, defining a constraint on the maximum number of Posted TLPs sent to the re-order buffer and into the application. By default, a high value is set to account for the LTI latency plus the latency to the AXI Manager B Response. Therefore, the `RADM_MAX_OUTSTD_P_REQ` parameter must be higher than the re-order buffer capability.

LTI LA channels back-pressures (LA interface runs out of credits) using the packet halt mechanism as well.

LA back-pressure must be always a transient state, permanent back pressure affects multi-VC QoS applications and PCIe ordering.

When the `CC_LTIM_ENABLE =1`, the TLP waits in the Order Buffer for an LTI response. For correct controller operation an LTI response must always return. If this condition is not satisfied, the flush feature in Order Buffer allows the waiting TLP transactions to progress. It simulates a translation fault and notifies the host software and the endpoints about the unsuccessful TLPs. When the LTIM LM, LA, LR, or LC interfaces time-out, it generates a MSI to the host application. In turn the host application can use register `ORDRB_LTI_LR_FLUSH_OFF` to flush all TLP inside the Order Buffer. The application must reset the controller after the flush. For more information, see "Reset Requirements" on page 90.

5.4.4 MSI-GIC

After SMMU memory translation, the MSI (Message Signal Interrupt) routes to the GIC (Generic Interrupt Controller) considering the PCIe ordering rules and reduces the latency of the interrupt generation. This interface handles only the PCIe inbound received interrupts and the local LTIM generated interrupts. The interface between the PCIe and the GIC is based on AMBA AXI4-Stream protocol.

Note You can enable MSI-GIC module without LTIM by setting CC_MSIGIC_EN parameter to '1' when CC_ORDRB_EN =1.

5.4.5 LTIM Mapping to AXI Manager

Table 5-27 shows the signal mapping between LTI and AXI manager for debugging purpose.

Table 5-28 shows the LTI LR Fault mapping to AXI, ACE-Lite, Posted and Non-Posted Transactions.

Table 5-29 shows the LRATTR Mapping to AxDOMAIN and AxCACHE.

1. This imposes a performance limitation when the traffic is RO intensive.

Table 5-27 LTIM Mapping to AXI Manager

LTI Signal	Width	AXI AW Signal	AXI AR Signal
ltim_lr_valid	1	mstr_awmisc_info_lti_valid	mstr_armisc_info_lti_valid
ltim_lr_ctag	1	mstr_awmisc_info_lti_ctag	mstr_armisc_info_lti_ctag
ltim_lr_attr	4	ACE Lite Signals[a]	ACE Lite Signals
ltim_lr_resp	2	mstr_awmisc_info_lti_resp	mstr_armisc_info_lti_resp
NA[b]	CC_AMBA_TLP_SEQ_WD	mstr_awmisc_info_lti_seqNum	mstr_armisc_info_lti_seqNum

a. For more information, see Table 5-29.
b. Synopsys implementation specific information.

Table 5-28 LTI LR Fault mapping to AXI, ACE-Lite, Posted and Non-Posted Transactions

Transaction Type	LTI Faults are Mapped to	AXI Response	FaultAbort and FaultRAZWI
Posted	AER	Faulted transactions are not presented to the AXI manager	LTIM_LRRESP_PCIE_MAP[3-0]
Non-Posted	CPL Errors		LTIM_LRRESP_PCIE_MAP[6-5]

Note An LTI Fault on a non-Posted transaction triggers the AXI bridge to synthesize a CPL without any prefix. It is irrespective of the controller configuration and corresponding attribute of the NP request prefix.

Table 5-29 LRATTR Mapping to AxDOMAIN and AxCACHE

LRATTR Encoding	ARM Memory Type	ACE Shareability Domain	AXI Memory Type	AxDOMAIN	AR CACHE	AW CACHE
0	Device-nGnRnE	System	Device non-Bufferable	0x3	0x0	0x0
1	Device-nGnRE	System	Device Bufferable	0x3	0x1	0x1
2	Device-nGRE					
3	Device-GRE					
4	Normal Non-cacheable	System	Normal Non-cacheable Bufferable	0x3	0x3	0x3
5	Normal Inner Non-cacheable Outer Cacheable					
6	Normal Write-back No-allocate	Outer-shareable	Write-back No-allocate	0x2	0xB	0x7
7	Normal Write-back Allocate Outer-shareable	Outer-shareable	Write-back Read and Write-allocate	0x2	0xF	0xF

LRATTR Encoding	ARM Memory Type	ACE Shareability Domain	AXI Memory Type	AxDOMAIN	AR CACHE	AW CACHE
14	Normal Write-back No-allocate	Non-shareable	Write-back No-allocate	0x0	0xB	0x7
15	Normal Write-back Allocate Non-shareable	Non-shareable	Write-back Read and Write-allocate	0x0	0xF	0xF
AxBAR and **AxSNOOP** are always 0x0.						

6

Legacy DMA Controller

This section describes the optional embedded multichannel legacy DMA controller. You enable the DMA controller by setting the `CC_DMA_ENABLE` configuration parameter. DMA VSEC registers (`PF0_VSECDMA_CAP`) help in identifying the DMA implementation, register location, and layout. The topics for this section are:

- "DMA Overview" on page 464
- "DMA Architecture" on page 468
- "Using the DMA" on page 483
- "Advanced DMA Information and Operation" on page 501

Note A new DMA architecture with advanced features is also available. For more information, see "HDMA Controller" on page 523.

6.1 DMA Overview

The RC system CPU, or the EP application CPU, can offload the transferring of large blocks of data to the embedded DMA controller[1], leaving the CPU free to perform other tasks. You can configure the DMA to have one to eight read channels and one to eight write channels. It can simultaneously perform the following types of memory transactions (as shown in Figure 6-1):

- **DMA write**

  Transfer (copy) of a block of data from local (application) memory to remote (link partner) memory.
- **DMA read**

  Transfer (copy) of a block of data from remote (link partner) memory to local (application) memory.

Therefore the DMA supports full duplex operation, processing read and write transfers at the same time, and in parallel with normal (non-DMA) traffic. Upon completion of a DMA transfer or an error, the DMA optionally interrupts the local CPU or sends an interrupt MWr (IMWr) to the remote CPU. The DMA is highly configurable and you can program it using the local DBI or over the PCIe wire[2].

In linked list mode, the DMA fetches the transfer control information (called channel context) for each transfer (block), from a list of DMA elements that you have constructed in local memory.

The DMA supports "Native Controller Operation (without AXI Bridge)" on page 481.

You can also use the DMA with the AXI bridge, in which case the DMA is located between the native PCIe controller and the AXI bridge module.

1. For brevity, the term DMA is used instead of embedded legacy DMA controller, throughout the remainder of this chapter.
2. RP can program the EP DMA using CfgWr or MWr (when ENABLE_MEM_MAP_PL_REG =1), but the EP cannot program the RP DMA over the wire.

6.1.1 Limitations

Table 6-1 identifies the limitations when using the DMA module.

Table 6-1 DMA Module Limitations

Limitation	Note
Memory DMA Traffic Only	I/O and Type 0 or Type 1 configuration DMA transfers are not supported
Performance limitations in a few configurations[a]	In the following configurations DMA may not achieve full link utilization. ■ 512-bit controller data width and 256/128 bytes MTU ((CX_NW = 16) && (CX_MAX_MTU <= 256)) ■ 256-bit controller data width and 128 bytes MTU ((CX_NW = 8) && (CX_MAX_MTU<=128)) ■ AXI manager configurations with data-path wider than 64 bits do not naturally align DMA descriptor reads. These configurations must have MSTR_FIXED_SIZE_ENABLE to avoid byte accesses that degrade performance. ((AMBA_INTERFACE > 1) && (MSTR_FIXED_SIZE_ENABLE==0) && (MASTER_BUS_DATA_WIDTH > 64)) ■ Single channel DMA ((CC_NUM_DMA_WR_CHAN =1) || (CC_NUM_DMA_RD_CHAN =1)) ■ Insufficient number of outstanding NP requests for the system latency (Insufficient CC_NUM_DMA_RD_TAG and/or CX_REMOTE_MAX_TAG)
Idle interval in the transfer in a few use cases[a]	DMA can load only one descriptor at a time. So, the previous descriptor transactions must end before loading the next one, that is, ■ All completions must be received, transformed into MWR and the write transaction must be acknowledged ■ If enabled, all interrupts must be generated In the following use cases, waiting for all the transactions to end before loading the next descriptor creates an idle interval in the transfer. ■ Low channel count ■ Small descriptor transfer size DMA early fetch feature, (CC_DMA_EFETCH_EN) supported only in AXI bridge configurations, minimizes the load time of the next descriptor, but does not eliminate the idle interval completely.
No support for Inbound Posted Tracker Enable feature or AXI High Priority Master	DMA does not support the following features: ■ Inbound Posted Tracker Enable (CC_IB_WREQ_PTRK_EN =1) ■ AXI High Priority Master topology (CC_HP_MASTER =1)

a. You must execute the DMA performance test cases provided with the controller to assess the DMA performance in configurations involving the listed use cases.

6.1.2 Feature List

The DMA module supports the following features:

Table 6-2 DMA Module Features

DMA Module Feature	Supported
Number of Read Channels	1-8
Number of Write Channels	1-8
AXI Support	Yes
Native Controller Support (without AXI)	Yes
Linked List Mode	Yes
Linked List Descriptor Alignment	DWORD
Scatter Gather Support (by using Linked Lists)	Yes
Full Duplex Read/Write Channel Operation	Yes
Separate Tag Pool For DMA Traffic	Yes
SAR/DAR Address Alignment	BYTE
Transfer Size	1 byte - 4 GB
AXI Address Bus Width	32, 64
AXI Databus Width	32, 64, 128, 256, 512
Independent Internal, Master and Subordinate Databus Widths	Yes
Independent Controller and AXI Clocks	Yes
Latency Tolerance Reporting (LTR) Support	Yes
OBFF Support	Yes
Address Translation Services (ATS) Support	Yes
AtomicOps Support	-
TLP Processing Hints (TPH) Support	Yes
TLP Prefix Support (Including PASID)	-
Programmable Interrupt Generation	Yes
Reordering of DMA Read Transfer Completions	Yes
Reordering of DMA AXI R Channel Transactions (AXI Configurations)	Yes
ID Based Ordering (IDO) Signaling Support[a]	Yes
Datapath and RAM Protection For Limitations, see “RAS Data Protection (RAS DP)” on page 164.	Yes

DMA Module Feature	Supported
RAM Read Enable for Low-Power Optimization	Yes
Automatic Flushing/Reset of Bridge (AXI Configurations) When Link Goes Down	Yes
Automatic Flushing of Subordinate (AXI Configurations) When Link is Not Up	Yes

a. DMA identifies and decodes IDO bit from the TLP bit stream and passes on the information to the Application through native client interfaces or AXI Bridge. But, IDO bit of the TLP Header cannot be explicitly set by the DMA.

6.2 DMA Architecture

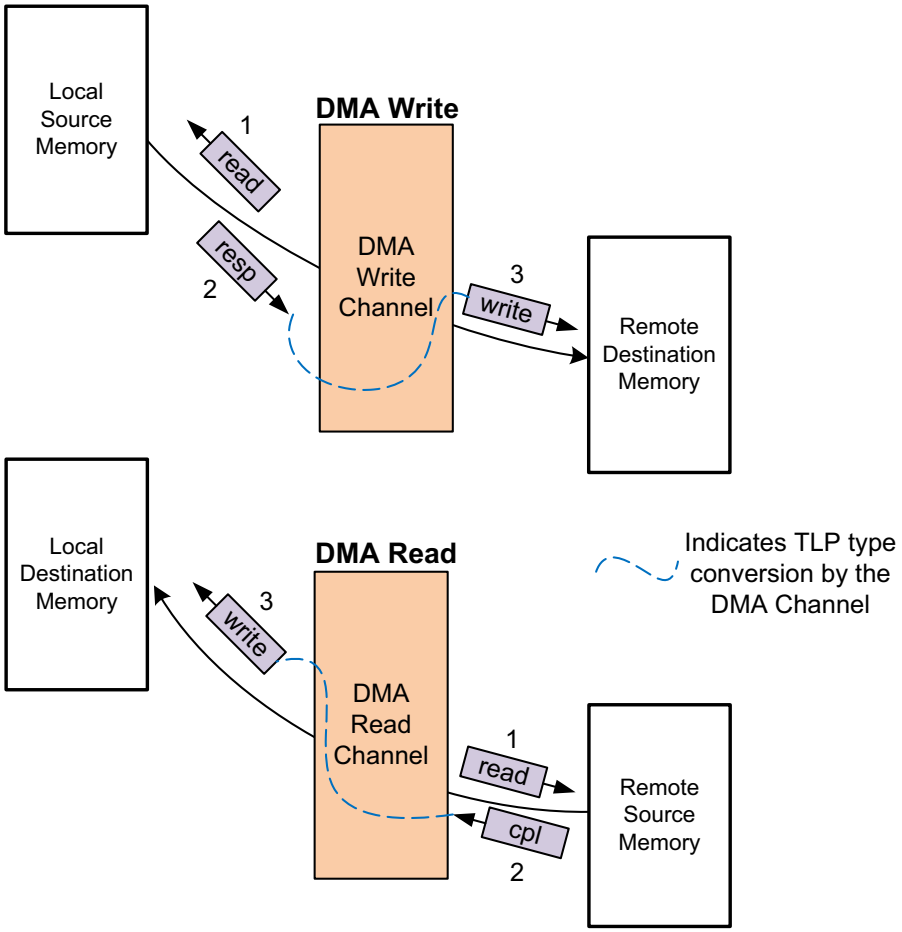
This section describes the architecture of the DMA. The topics for this section are:

- "Architecture Overview"
- "Interrupts and Error Handling" on page 469
- "Linked List Mode" on page 475
- "Native Controller Operation (without AXI Bridge)" on page 481

6.2.1 Architecture Overview

A DMA write and read channel operate independently to maximize the performance of the DMA read and write data transfers over the PCIe link. When you configure the DMA with multiple read channels, then it uses a Weighted Round Robin (WRR) arbitration scheme to select the next read channel to be serviced. The same applies when you have multiple write channels.

Figure 6-1 System Level View of DMA



6.2.1.1 DMA Write Transfer

The DMA injects multiple MRd requests of size less than or equal to Max_Payload_Size into the inbound request path, directed toward the local application. The DMA converts the read responses into MWr TLPs, that it then transmits to the remote link partner. When the DMA data transfer is complete, the CPU is notified. For more information, see "Interrupts and Error Handling" on page 469. For a write transfer, the SAR is the address of the local memory, and the DAR is the address of the remote memory.

6.2.1.2 DMA Read Transfer

The DMA injects multiple MRd requests of size less than or equal to the minimum of {Max_Read_Request_Size, Max_Payload_Size_Supported[1]} into the outbound request path, directed toward the remote link partner. The DMA converts the read responses into write requests, that it then transmits to the local application. When the DMA data transfer is complete, the CPU is notified. For more information, see "Interrupts and Error Handling" on page 469. For a read transfer, the SAR is the address of the remote memory, and the DAR is the address of the local memory.

6.2.1.3 RAM Requirements

The DMA requires two RAMs. For more information, see "RAM Requirements" on page 64.

6.2.1.4 Registers and Context Memory

You program each DMA transfer (channel) using the registers described in "DMA Software Register Map" on page 487. The registers that are specific to each channel are called the channel context registers. As the transfer progresses, the DMA updates these registers[2]. You program the channel context information and other DMA control registers directly with the local CPU through the DBI or remotely over the PCIe link. Some of the registers are implemented in a high speed wide interface context RAM. If you implement your RAMs external to the PCIe controller, you must connect this RAM through the external RAM interfaces.

6.2.2 Interrupts and Error Handling

The registers referenced in Figure 6-2 and Figure 6-3 that are related to error and interrupt handling are described in "DMA Software Register Map" on page 487.

The DMA generates two interrupts per channel:

- **Done**: The DMA successfully completes the transfer.
- **Abort**: The DMA fails to complete the transfer, or an error occurs during the transfer.

The interrupts are signaled to the software on your CPU, using one or both of the following mechanisms:

- Locally through the `edma_int[CC_NUM_DMA_RD_CHAN+CC_NUM_DMA_WR_CHAN-1:0]` bus.
- Remotely using a posted memory write (IMWr), which can be interpreted as an MSI or MSI-X when directed toward the RP.

1. This limit is set by the DMA read buffer segment size. For more information, see "Read Buffer" on page 506.
2. The design sometimes caches the context to increase performance. Therefore these registers might contain a delayed snapshot of the actual DMA channel state.

- There are two programmable IMWr addresses per channel, one each for the done and abort interrupts. For MSI, you must program all IMWr address registers with the same MSI address, as PCIe only supports a single MSI address per function.
- A single IMWr data register is used for both types of interrupts, so you must read `DMA_READ_INT_STATUS_OFF` (or `DMA_WRITE_INT_STATUS_OFF`) to identify the interrupt type.

Note

- The IMWr address and data do not come from the MSI PCIe capability registers but from the "DMA Software Register Map" on page 487.
- For an Upstream Port, your software must program these registers with the same information that it is using to program the MSI/MSI-X capability for the function that is associated by your software with that channel in a multifunction setup.
- You only need to setup these registers at initialization, and not for every DMA transfer.
- For more information on interrupt handling, see the descriptions of the DMA interrupt registers and "Programming Examples" on page 490.

- The relaxed ordering (RO) bit is automatically set to zero in the TLP header of all IMWr's so that subsequent MWr TLPs do not arrive at the remote device before the IMWr.

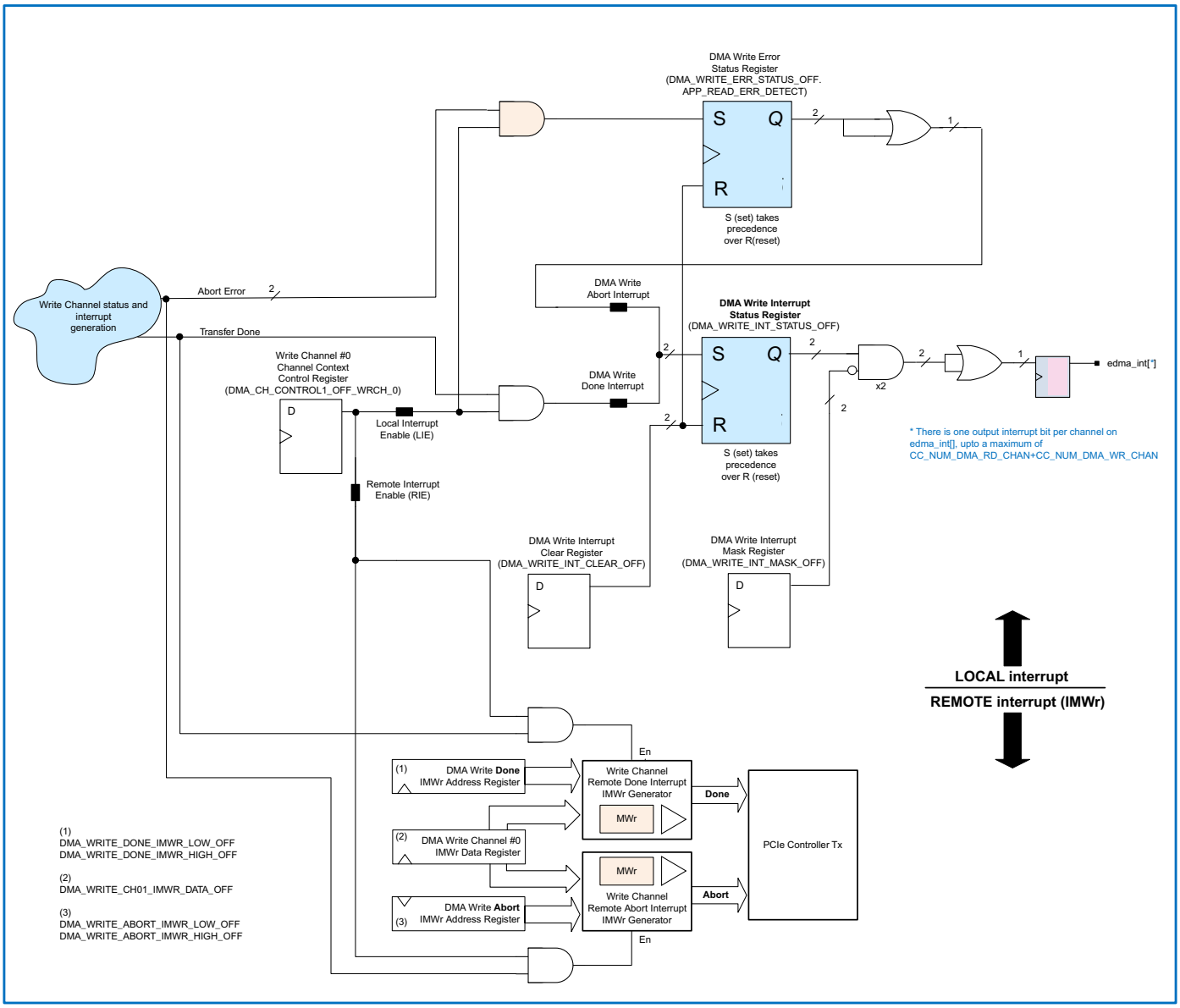
The interrupt handling mechanism is different for linked list (LL) mode (than non LL mode), and there are also some differences between the read and write channels.

6.2.2.1 Non Linked List Mode Interrupt Handling

You enable the local and remote interrupts through the local and remote interrupt enable (LIE and RIE) bits: `DMA_CH_CONTROL1_OFF_WRCH_0.lie` and `DMA_CH_CONTROL1_OFF_WRCH_0.rie`.

In the write channel, there is only one error condition that results in an abort interrupt. For more information, see "Linked List Mode" on page 475. You mask, clear, and read the status of each of the two interrupts (done and abort) through the DMA interrupt registers as indicated in Figure 6-2. You can read the status of the abort error (that contribute to the abort interrupt) through the `APP_READ_ERR_DETECT` field of the `DMA_WRITE_ERR_STATUS_OFF` register.

Figure 6-2 Write Interrupt Generation - Non Linked List Mode - Shown For Write Channel #0



Hint

If you want a remote interrupt and not a local interrupt then:

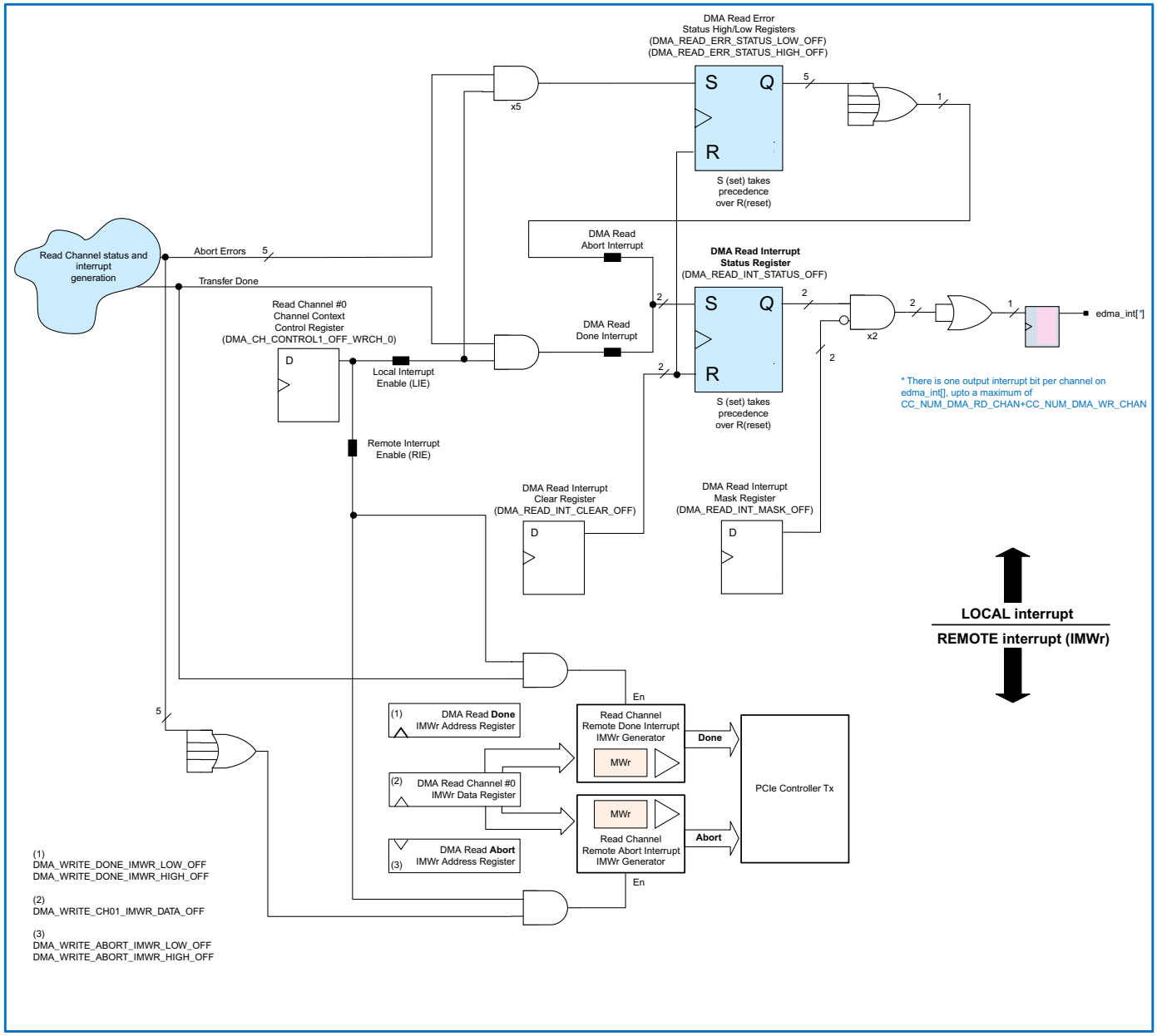
- Set LIE and RIE
- Mask the local interrupt using the mask register

This allows you to poll the status register to distinguish a DONE from an ABORT

In the read channel, there are five error conditions that results in an abort interrupt. For more information, see "Linked List Mode" on page 475. You mask and clear each of the two interrupts (done and abort) through the DMA interrupt registers as indicated in Figure 6-3. However, you can read the status of each of

the five abort errors (that contribute to the abort interrupt) through `DMA_READ_ERR_STATUS_LOW_OFF` and `DMA_READ_ERR_STATUS_HIGH_OFF`.

Figure 6-3 Read Interrupt Generation - Non Linked List Mode - Shown For Read Channel #0

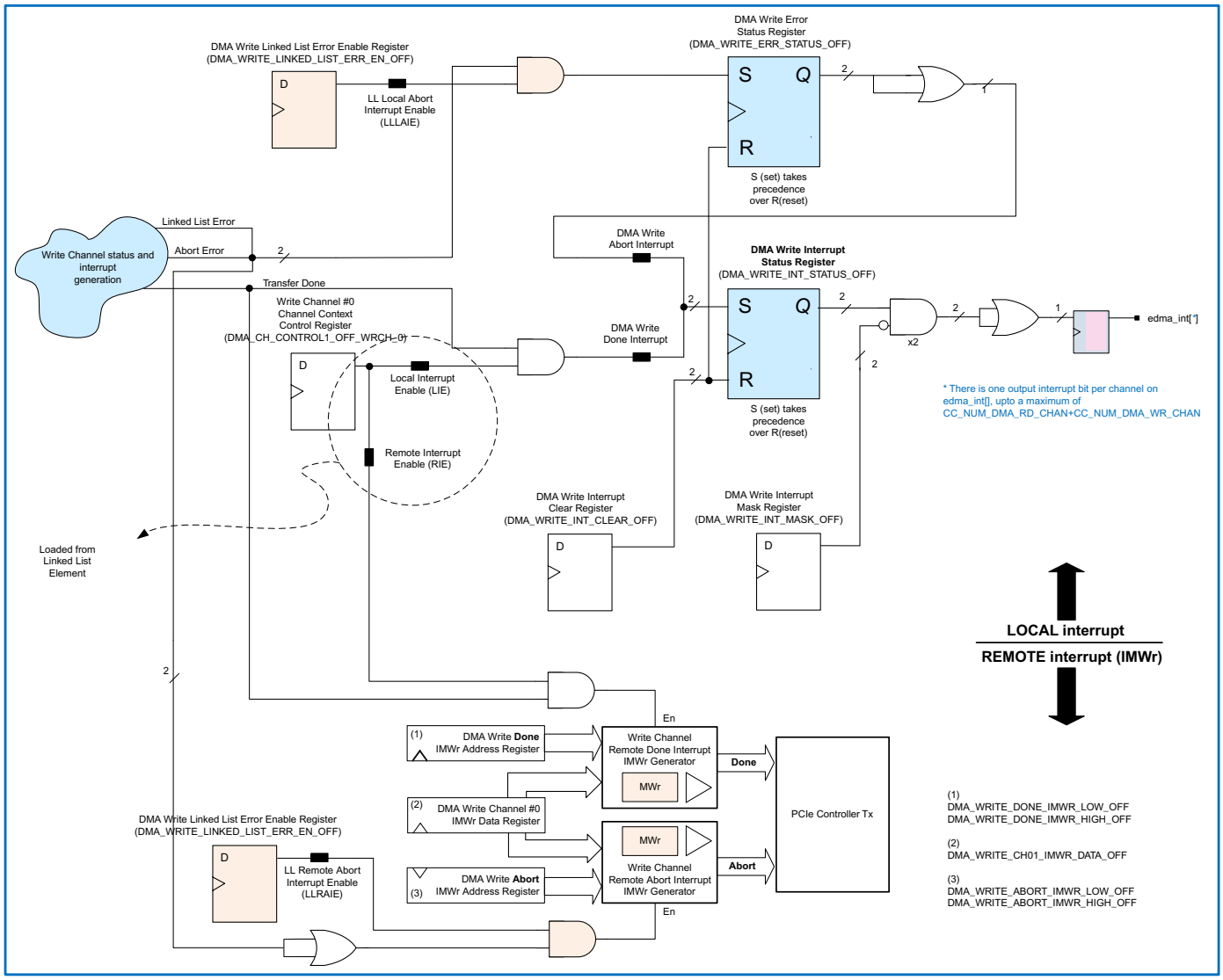


6.2.2.2 Linked List Mode Interrupt Handling

The LIE and RIE bits in the LL element enable the channel done interrupts (local and remote). The `LLLAIE` and `LLRAIE` bits of the `DMA_WRITE_LINKED_LIST_ERR_EN_OFF/DMA_READ_LINKED_LIST_ERR_EN_OFF` registers enable the channel abort interrupts (local and remote). In the write channel, there are two error conditions that results in an abort interrupt. For more information, see "Linked List Mode" on page 475.

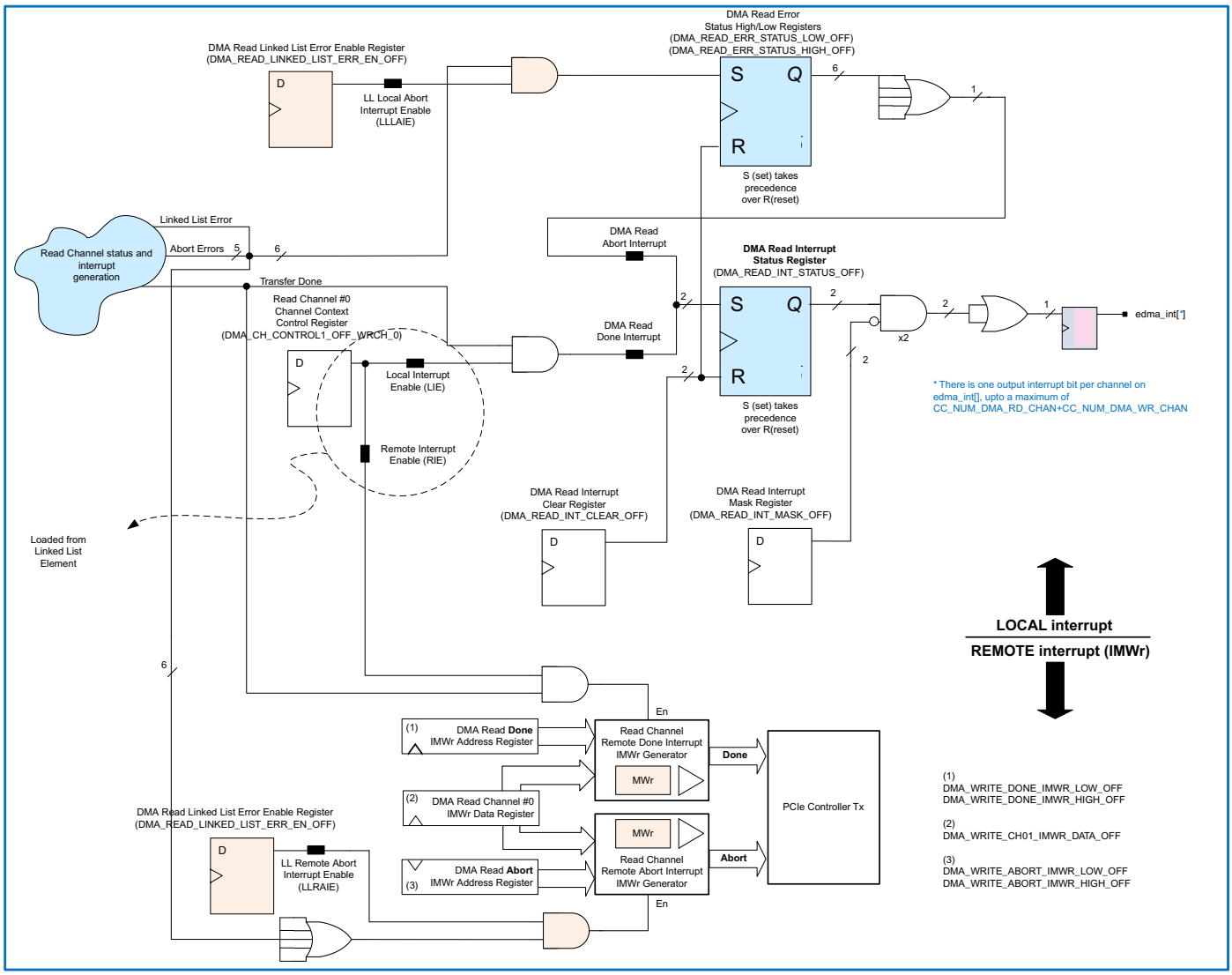
You mask and clear each of the two interrupts (done and abort) through the DMA interrupt registers as indicated in Figure 6-4. You can read the status of each of the two abort errors (that contribute to the abort interrupt) through the `DMA_WRITE_ERR_STATUS_OFF` register.

Figure 6-4 Write Interrupt Generation - Linked List Mode - Shown For Write Channel #0



In the read channel, there are six error conditions that results in an abort interrupt. For more information, see "Linked List Mode" on page 475. You mask and clear each of the two interrupts (done and abort) through the DMA interrupt registers as indicated in Figure 6-5. You can read the status of each of the six abort errors (that contribute to the abort interrupt) through the `DMA_READ_ERR_STATUS_LOW_OFF` and `DMA_READ_ERR_STATUS_HIGH_OFF` registers.

Figure 6-5 Read Interrupt Generation - Linked List Mode - Shown For Read Channel #0



For more information on done interrupts in linked list mode, see "Using Interrupts for Linked List Producer-Consumer Synchronization" on page 516.

Note In non-linked list mode, LIE acts as a global switch. However when in linked list mode, LIE is just local to the current linked list element and the global switch is LLLAIE.

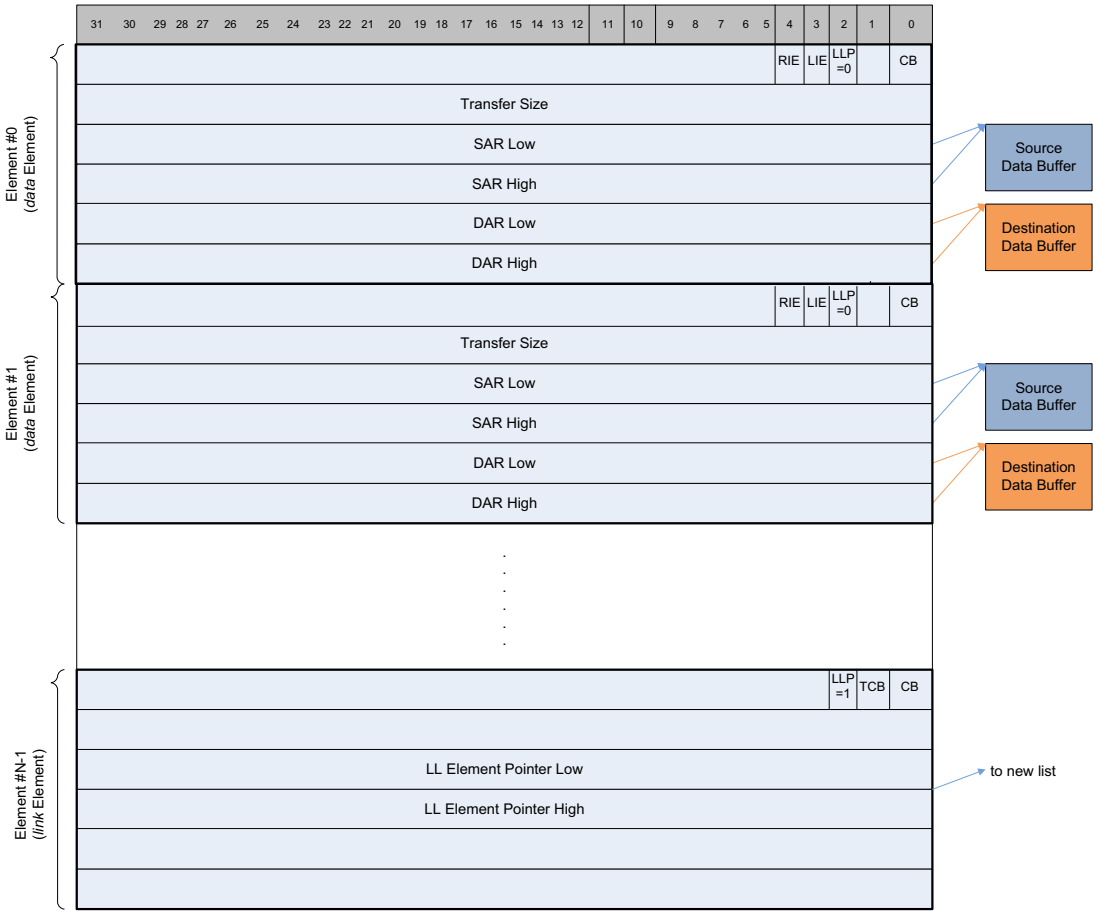
Attention **Software Programming of an Endpoints LIE and RIE Bits for Linked List Transfers.**
If the DMA driver is running on the host and the interrupt service routine is reading local interrupts to determine if the transfer is successful, then you must set LIE and RIE in the same element and you should mask or ignore the local interrupt pin. Setting RIE and LIE in element A followed by RIE (only) in element B is not a verified usage scenario.

6.2.3 Linked List Mode

The DMA provides a linked list (LL) mode to efficiently move data from source to destination with minimal intervention from the local CPU. This mode provides an alternative to programming the DMA multiple times to transfer multiple blocks of data. The programming information (address, size, and so on) for each block of memory is pre-programmed by your software into a LL element (also known as a descriptor) in local memory. Each element (called a data element) in the LL structure (called a transfer list) can transfer up to 4 GB of data.

You enable LL operation for a channel, by setting the LLE field of the `DMA_CH_CONTROL1_OFF_WRCH_i`[1] register to 1. You can enable LL mode independently for each channel. When you enable LL for more than one channel, then you must have a separate LL structure in local memory for each channel. Your application must produce the LL element structure in local memory as shown in Figure 6-6. Normally, all of the elements are contiguous (one after the other) in memory, and each element has six DWORDs containing the information about the block of data to be transferred. You program the channel context registers (`DMA_LLP_LOW_OFF_WRCH_i` and `DMA_LLP_HIGH_OFF_WRCH_i`) with the location of where you have placed the LL element structure in local memory.

Figure 6-6 Linked List Element/Descriptor Structure in Local Memory with N Elements



1. Replace WRCH with RDCH for a read channel.

When you start the DMA transfer (by writing to the DMA Write Doorbell Register `DMA_WRITE_DOORBELL_OFF` or DMA Read Doorbell Register `DMA_READ_DOORBELL_OFF`), the DMA reads (consumes) each element from local memory, and loads the information (SAR, DAR, size, and so on) from that element into the channel context registers in the DMA. These channel context registers determine the operation of the channel that the DMA controller is currently servicing. The DMA then proceeds to transfer the block of data (as defined by the element), and when it is finished, reads the next element from local memory. Normally, all of the elements are contiguous (one after the other) in memory, with the starting address defined in the channel context DMA Linked List Pointer Low Register `DMA_LLP_LOW_OFF_WRCH_0`. When you want to jump in local memory to another element list (or recycle the consumed elements), then you set the LLP bit in the element (for example, link element #N-1 in Figure 6-6), specify the location of the next element structure using the LL Element Pointer DWORDs (as indicated in Figure 6-6), and, set TCB to 1 (for recycling) or to 0 (to jump to another list).

Relationship Between Element DWORDs and Channel Context Registers

Notice the similarity between a data element and the DMA Channel Context registers for each channel. Each element has six DWORDs as in Figure 6-6. There are eight channel context registers (DWORDs) for a channel. The DMA loads the six element DWORDs into the following channel context:

- CB, LLP, LIE, and RIE fields of the DMA Channel Control 1 register.
- DMA Transfer Size
- DMA SAR Low and DMA SAR High
- DMA DAR Low and DMA DAR High

The definitions of the element DWORD bit fields are the same as the DMA Channel Context registers described in the "DMA Software Register Map" on page 487, with the exception of the LIE and RIE bits. The LIE and RIE bits in a LL element, only enable the done interrupt. In non-LL mode, the RIE and LIE bits (in the channel context registers) enable the done and abort interrupts.

When the LLP field of the first DWORD in the element is set to 1, the element is a link element. The DMA only loads the following information into the channel context registers:

- CB, TCB, and LLP bits of the first DWORD. The LIE and RIE bits are not defined in a link element.
- LL Element Pointer (3rd and 4th DWORDs) into the DMA Linked List Pointer registers (`DMA_LLP_LOW_OFF_WRCH_0` and `DMA_LLP_HIGH_OFF_WRCH_0`).

For more information on LL operation, see "Linked List Operation" on page 513.

Caution

Linked List Operation

- When you enable linked list operation (DMA_CH_CONTROL1_OFF_WRCH_0.lle =1b1), the DMA overwrites the following DMA channel context registers with information from linked list data elements:
  - The CB, LLP, LIE, and RIE fields of the DMA Channel Control 1 register.
  - DMA Transfer Size
  - DMA SAR Low & High
  - DMA DAR Low & High
- The structure of a link element is different to that of the data element. A data element has no TCB field. A link element has no LIE or RIE fields. It has no SAR, DAR, or Transfer Size DWORDs, but has LL Element Pointer DWORDs instead of the SAR DWORDs.
- The LIE and RIE bits in a LL element, only enable the done interrupt. In non-LL mode, the RIE and LIE bits enable the done and abort interrupts. For more information, see "Interrupts and Error Handling" on page 469.

Descriptor Early Fetch (AXI Bridge Configuration)

In AXI bridge configuration, the DMA serializes the next descriptor fetch with the previous descriptor's last data fetch. This allows the next descriptor to be loaded into DMA's context memory just in time. Early descriptor fetch reduces the number of idle cycles between subsequent descriptor fetches introduced because of high round-trip read-to-completion latencies in scenarios where the descriptor's transfer size is small.

The descriptor is fetched and loaded into AXI Bridge's master completion buffer. While the previous descriptor's CPL and MWr transactions are in progress, DMA back-pressures the descriptor loaded in the master completion buffer. It is recommended to have the configuration respecting the following relationship to avoid any potential performance issues:

```
CC_MAX_MSTR_TAGS_AXI > (CC_NUM_DMA_RD_CHAN + CC_NUM_DMA_WR_CHAN)
```

The early fetched descriptor completes concurrently with previous descriptor's data completion. If previous descriptor transfer aborts due to an error in any data completion, the early-fetched descriptor is flushed.

Figure 6-7 and Figure 6-8 describe Read/Write channel operation example for a scenario where each descriptor transfer size is 1 TLP.

Note: You can revert to legacy descriptor fetch by setting the hidden parameter `CC_DMA_EFETCH_EN` to 0.

Figure 6-7 Read Channel Operation Example: Before and After Early Fetch Optimization

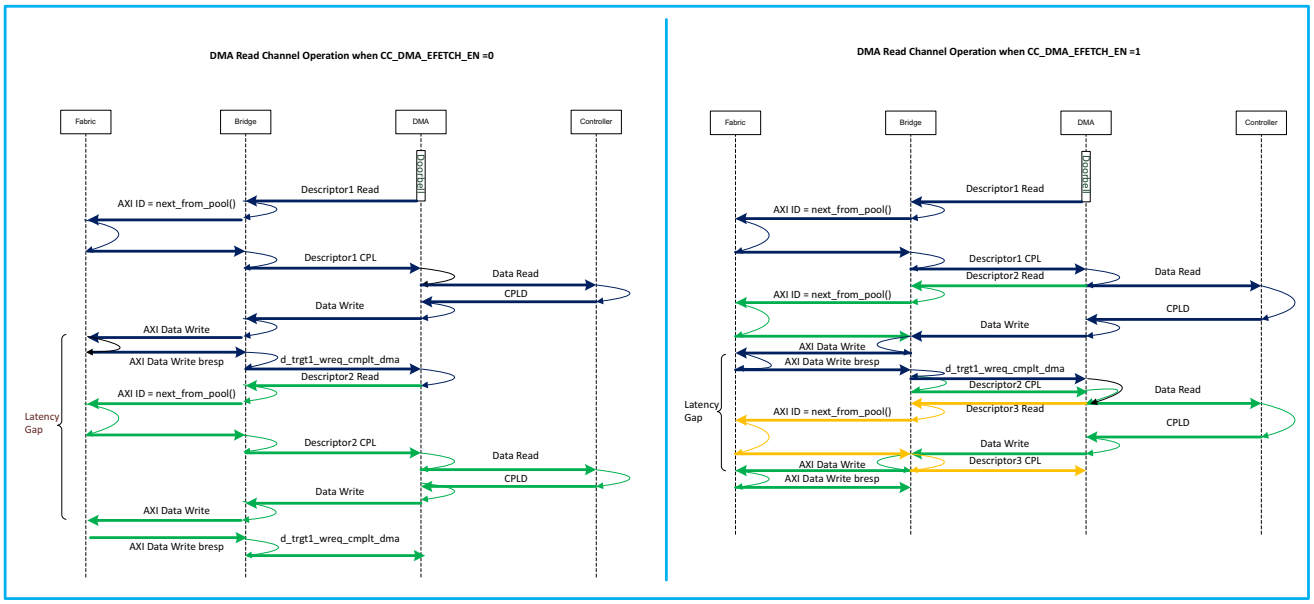
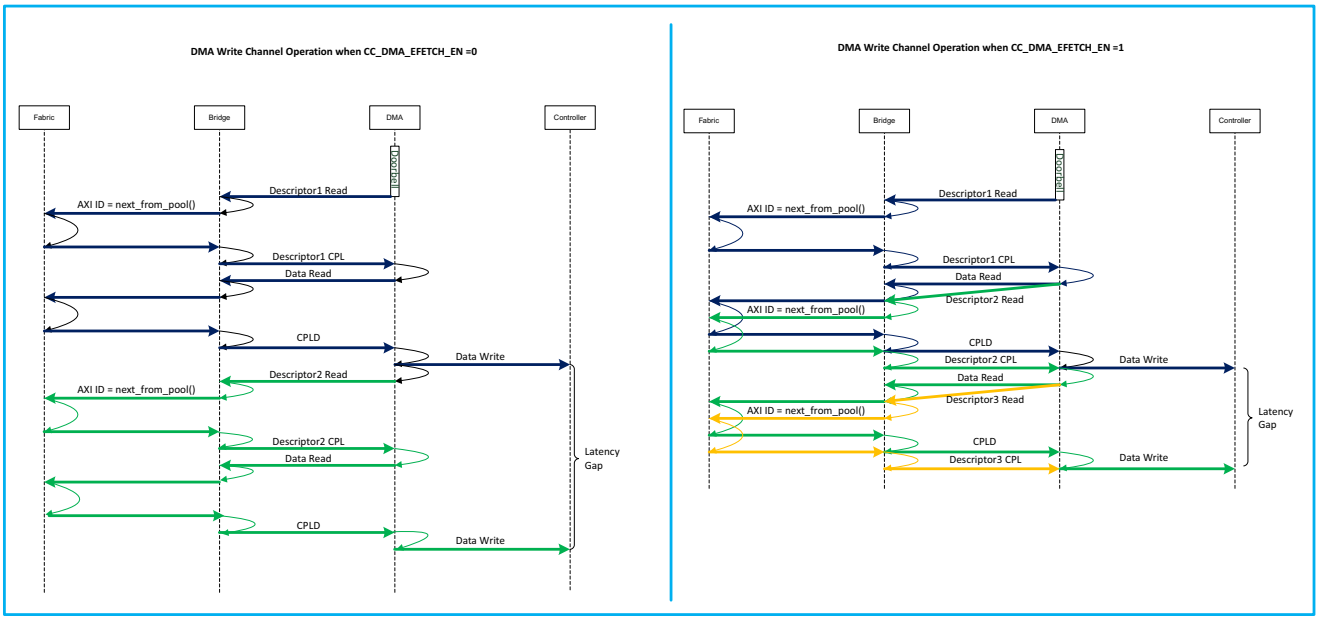


Figure 6-8 Write Channel Operation Example: Before and After Early Fetch Optimization



6.2.4 Hardware Flow Control (CC_DMA_HSHAKE =1)

This optional feature is available only for those channels which are configured to operate in linked list (LL) mode. It enables your application hardware to flow control the DMA controller, that is, your hardware determines when data block transfer starts.

In normal mode, as soon as the DMA is doorbelled, the DMA reads the LL descriptor, and starts the data block transfer. When this feature is enabled, the DMA reads the descriptor, but starts the data transfer only when your application logic toggles the `dma_[w|r]dxfer_go_togg[]` input.

This feature is helpful in scenarios where the data block generation/consumption is hardware controlled. After your application software configures the DMA channel in LL mode and enables the handshake feature, your application hardware can directly notify the DMA of the data block availability or its ability to receive the data block, thereby freeing your application software of additional overheads of tracking the hardware states, maintaining the memory mapped linked lists, and managing the DMA.

The DMA handshake mechanism can be turned on or off per channel using `DMA_[WRITE/READ]_ENGINE_EN_OFF` register. The handshake mechanism cannot be enabled/disabled when the channel status is active, that is, when `DMA_CH_CONTROL1_OFF_[WR|RD]CH_i.CS =01`.

DMA Handshake Operation

The handshake between the DMA and your application hardware is done using `dma_[w|r]dxfer_go_togg[]` and `dma_[w|r]dxfer_done_togg[]`. In case of a DMA Write, your application should toggle the `dma_wdxfer_go_togg[]` signal to indicate data block availability. In case of a DMA Read, your application should toggle the `dma_rdxfer_go_togg[]` signal to indicate that your application hardware is ready to receive a data block.

To keep a track of `dma_[w|r]dxfer_go_togg[]` signal toggles, the DMA implements a 5-bit handshake counter for each write/read channel. When your application toggles `dma_[w|r]dxfer_go_togg[]` signal, the handshake counter is incremented. The handshake counter value is taken into account by the DMA before performing the data transfer for each descriptor. The data transfer happens only when the handshake counter value is non-zero. When the data transfer is complete, the DMA decrements the handshake counter, and toggles `dma_[w|r]dxfer_done_togg[]` signal to indicate completion of data transfer to your application.

Caution

- This feature is not supported for LLelements with zero-byte transfer size.

dma_[w|r]dxfer_done_togg

- The signal `dma_[w|r]dxfer_done_togg` is asynchronous to the `core_clk`. The time period between toggle edges must be wider than the (1 + `CX_DMA_HSHAKE_SYNC_DEPTH`) `core_clk` period.

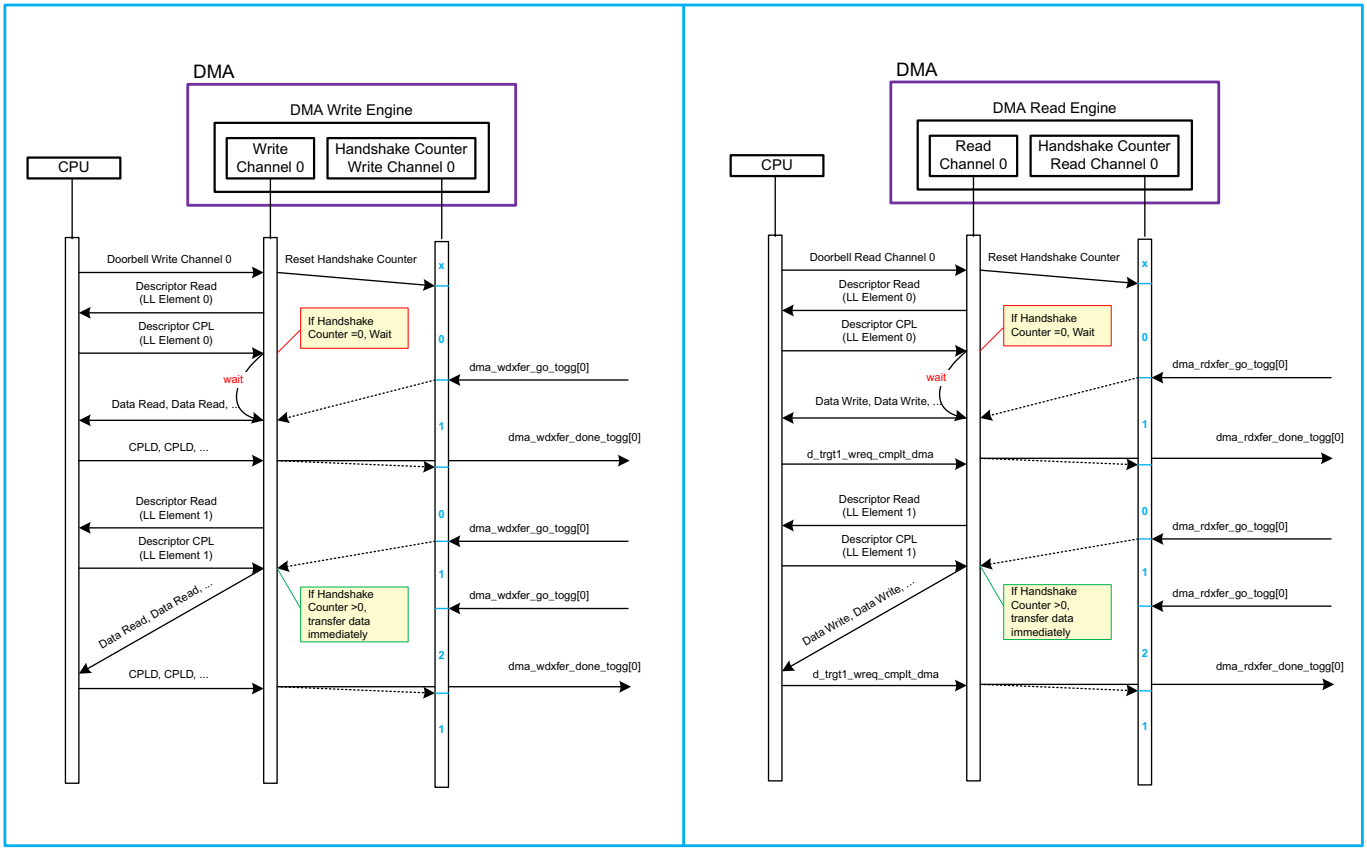
  Note: The default value of `CX_DMA_HSHAKE_SYNC_DEPTH` is 2.
- The signal `dma_[w|r]dxfer_go_togg` can toggle only when `edma_xfer_pending` is asserted, after the channel is doorbelled.
- There is no CDC handshake to acknowledge `dma_[w|r]dxfer_done_togg` toggle.

DMA Handshake Counter

- The handshake counter is 5-bit wide, so only 32 outstanding `dma_[w|r]dxfer_go_togg` requests can be handled by the DMA for each channel. The DMA does not implement an overflow protection or overflow error indication mechanism for the handshake counter. It is the responsibility of your application to keep the number of `dma_[w|r]dxfer_go_togg` toggles under check.
- The handshake counter is read only, your application must use it for debug purpose only.

The operation of the DMA when the handshake feature is enabled for a write/read channel is as follows:

Figure 6-9 DMA Handshake Operation (Write/Read Channel 0)



1. The CPU doorbells channel 0.
2. The DMA resets the handshake counter for channel 0.
3. The DMA reads LL element 0 descriptor. After the DMA receives descriptor read completions, the DMA checks the value of the handshake counter.
   - If handshake counter =0, DMA waits until handshake counter >0 before transferring the data block.
   - If handshake counter >0, DMA transfers the data block immediately.

   Note: The handshake counter is checked every time a new descriptor block is loaded and DMA is ready to start a new descriptor data transfer.
4. After the data transfer is complete, the DMA
   - Decrements the handshake counter, and
   - Toggles `dma_[w|r]dxfer_done_togg[0]`.
5. Steps 3-4 are repeated for all the remaining elements of the linked list.

6.2.5 Native Controller Operation (without AXI Bridge)

When you configure the native PCIe controller (that is, without AXI bridge) with the DMA, then special consideration must be given to the proper use of the Tx and Rx interfaces.

Caution

To Avoid Deadlock:

- Use XALI0 to return PCIe and DMA write channel completions[a] to the controller.
- Use XALI1 for supplying non-posted requests to the controller.
- Use XALI2 for supplying posted requests to the controller.
- Your application must return all DMA write channel completions in the same order[b] as requested, and must not split completions into multiple smaller responses.

To Avoid Data Corruption:

- As described in "PCIe TAGs" on page 507, the DMA uses a fixed number of the PCIe tags in the transmit direction. Therefore your application should not use those tags when generating outbound non-posted requests.

a. DMA write channel completions are completions (to MRd requests generated by the DMA write channel) that your application has supplied on the XALI0 interface and that the DMA write channel converts to posted writes for transmission to the remote link partner.
b. The ordering requirement is within a single channel number; inter-channel ordering is not required.

When you enable the DMA (`CC_DMA_ENABLE =1`), the "Target Completion Lookup Table" on page 363 is included. The maximum number of outstanding DMA write channel MRd requests (to the local application) is fixed at `CX_REMOTE_MAX_TAG`+1, and the maximum number of outstanding DMA read channel MWr transactions (to the local application) is fixed at 256. You must correctly drive the signals in Table 6-3.

Table 6-3 Additional I/O Added for Native Controller DMA

Name	I/O	Function
d_trgt1_req_dma[2:0]	O	Specifies the source of the request on the TRGT1 Interface.
d_trgt1_req_dma_channel[2:0]	O	DMA channel number.
d_trgt1_wreq_cmplt_dma	I	Your application must pulse this signal every time a DMA read transfer MWr is successfully written to local memory. **Note**: If you do not pulse this signal, the DMA becomes stuck.
d_trgt1_wreq_cmplt_dma_channel[2:0]	I	You must set this input to the DMA channel number.
d_trgt1_wreq_err_dma	I	Specifies if an error occurred in your application during a DMA read transfer MWr to local memory
You must use these flow control signals to control the unloading of DMA non-posted requests from the receive interface on a cycle-per-cycle basis in conjunction with the `trgt1_radm_pkt_halt` and `radm_grant_tlp_type` signals. So, after receiving a grant, if your application is unable to process further grants, the `trgt1_radm_pkt_halt` must be asserted so that more grants are not issued. For more information, see "TRGT1 Packet Grant and Halt" on page 371.		
radm_trgt1_dma_grant	O	Indicates that a particular non-DMA transaction has been granted to output. This grant signal is a pulse.
trgt1_radm_dma_pkt_halt[a]	I	Halts the transfer of non-posted packets from DMA.

a. There is no equivalent signal to stop MWr packets from leaving DMA.

A few additional constraints described in Table 6-4 need to be considered when DMA is used in native PCIe controller configuration (without AMBA bridge).

Table 6-4 Additional Constraints for Native Controller DMA

Name	Function
Write channel priority	Non-DMA inbound MemRd has priority over DMA Wr Channel Rd request. In a scenario where all TAGs are consumed, when a TAG is released, the DMA logic first serves the non-DMA traffic; this can cause the DMA to starve.
Error handling	For DMA Write channel, all DMA completions with completion status different to SUCESSFUL must be without data.
Byte alignment	If byte alignment is enabled, DMA can send a MemRd request to your application with a max size of CX_MAX_MTU+ 4. To prevent this, the transaction need to be aligned to DW boundary.
4K Boundary	Your application may receive a Memory Write that crosses 4K boundary.

6.3 Using the DMA

This section describes how to use the DMA. The topics for this section are:

- "Source and Destination Address Registers (SAR, DAR)" on page 483
- "DMA Transfer Size Register" on page 485
- "Starting the DMA Transfer" on page 485
- "Detecting the End of the DMA Transfer" on page 485
- "Stopping the DMA Transfer (Software Stop)" on page 486
- "TLP Generator Header Control" on page 487
- "DMA Software Register Map" on page 487
- "Programming Examples" on page 490

You can configure the DMA to have from one to eight read channels and one to eight write channels. It can simultaneously perform the following types of memory transactions:

- DMA write

  Transfer (copy) of a block of data from local memory to remote memory.
- DMA read

  Transfer (copy) of a block of data from remote memory to local memory.

The DMA supports full duplex operation, processing read and write transfers at the same time, and in parallel with normal (non-DMA) traffic. Upon completion of a DMA transfer or an error, the DMA optionally interrupts the local CPU or sends an interrupt MWr (IMWr) to the remote CPU. The DMA is highly configurable and you can program it using the local DBI, or over the PCIe wire.

The channels are named read channel 7..0 and write channel 7..0. Each channel is programmed using a number of registers. For more information, see "DMA Software Register Map" on page 487.

Note

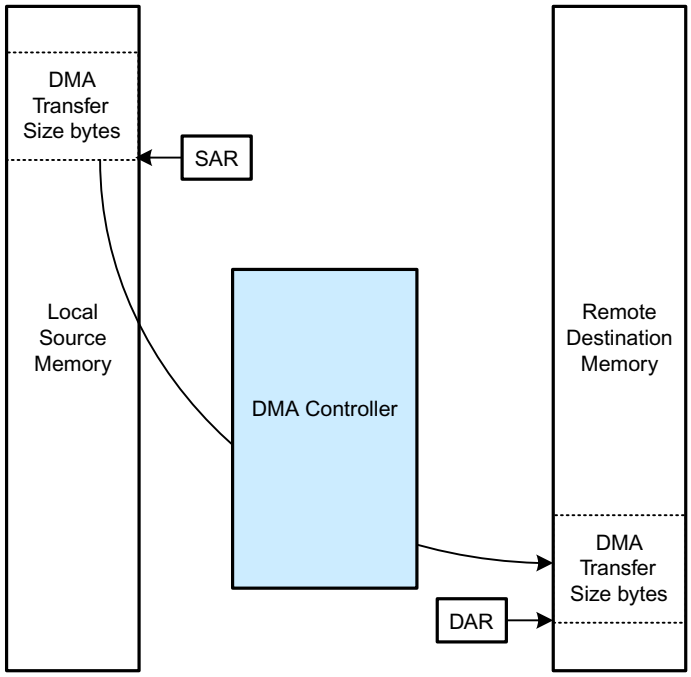
- Synopsys provides an example library of register access, and task level functions, that illustrate how to program the various features of the embedded DMA controller. To obtain this library contact Synopsys support through SolvNetPlus.
- The library is provided as an example, and is not subject to warranty, support, or maintenance.

6.3.1 Source and Destination Address Registers (SAR, DAR)

The DMA channel context SAR and DAR registers (`DMA_SAR_LOW_OFF_WRCH_0`[1] / `DMA_SAR_HIGH_OFF_WRCH_0` / `DMA_DAR_LOW_OFF_WRCH_0` / `DMA_DAR_HIGH_OFF_WRCH_0`) provide support for remote-to-local, and local-to-remote PCIe address mapping. You program the start of the local and remote data buffers using these registers, and the DMA increments the SAR and DAR as the DMA transfer progresses. For a write transfer, the SAR is the address of the local memory, and the DAR is the address of the remote memory, as shown in Figure 6-10.

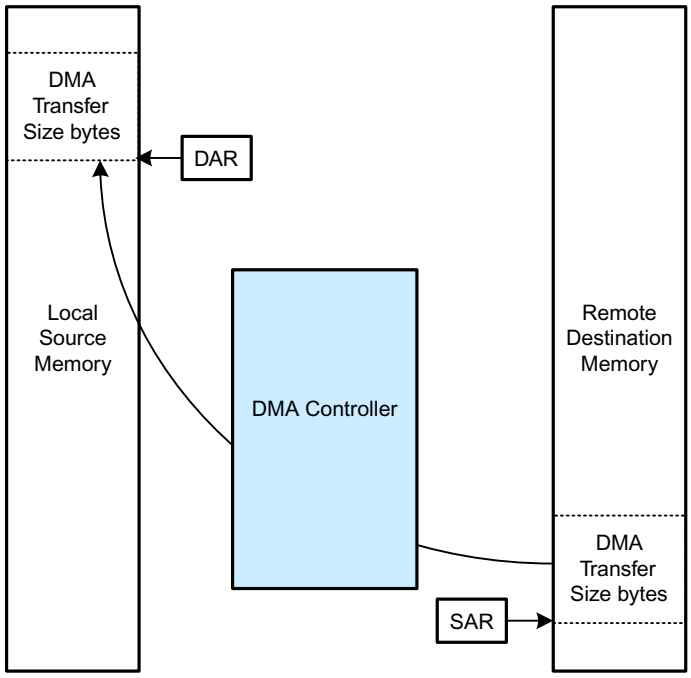
1. Replace WRCH with RDCH for a read channel.

Figure 6-10 Write Transfer: SAR and DAR for Write Channel 0



For a read transfer, the SAR is the address of the remote memory, and the DAR is the address of the local memory, as shown in Figure 6-11.

Figure 6-11 Read Transfer: SAR and DAR for Read Channel 0



The presence of the DMA controller does not affect:

- Normal filtering rules for inbound TLPs (for example, BAR checking in an Endpoint, as described in "Internal Address Translation Unit (iATU)" on page 220)
- The operation of any internal or external address translation, as described in "Receive Filtering" on page 134

Caution

When you do not want the iATU to translate outbound requests that are generated by the internal DMA module, then you must implement one of the following approaches:

- Ensure that the combination of DMA channel programming and iATU control register programming, causes no translation of DMA traffic to be done in the iATU.
- Activate the DMA bypass mode to allow request TLPs which are initiated by the DMA controller to pass through the iATU untranslated. You can activate DMA bypass mode by setting the DMA Bypass field of the iATU Control 2 Register (IATU_REGION_C-TRL_OFF_2_OUTBOUND_0).

6.3.2 DMA Transfer Size Register

You program the DMA transfer size using the DMA Transfer Size Register (`DMA_TRANSFER_SIZE_OFF_WRCH_i` or `DMA_TRANSFER_SIZE_OFF_RDCH_i`). The maximum DMA transfer size is 4GB, and the minimum transfer size is one byte (0x1). The DMA decrements the value in this register as the DMA transfer progresses. When all bytes are successfully transferred, the value in this register is zero. In LL mode, the DMA overwrites this register with the corresponding dword of the LL element.You can read this register to monitor the transfer progress. However, in some scenarios there is a delay before the controller updates this register. For example, when less than three channels are doorbelled, this register is only updated after a descriptor finishes (LL mode), or the transfer ends (non-LL mode).

6.3.3 Starting the DMA Transfer

After you program the DMA controller registers (including writing to the DMA Read Engine Enable or DMA Write Engine Enable register), you start a DMA transfer by writing the channel number to the Doorbell Number field of the DMA Write Doorbell Register (`DMA_WRITE_DOORBELL_OFF`) or DMA Read Doorbell Register (`DMA_READ_DOORBELL_OFF`). You can program and start both a read and a write transfer at the same time. The DMA supports full duplex operation, processing read and write transfers at the same time and in parallel with normal (non-DMA) traffic. You can program and start any number of channels in the DMA controller sequentially or simultaneously.

- **Sequentially**: configure and start a channel, and then configure and start another channel
- **Simultaneously**: configure multiple channels, and then start multiple channels

Caution

You must not write to any of the context registers for a particular channel after you start the channel by writing the channel number to the Doorbell Number field of the DMA Write Doorbell Register (DMA_WRITE_DOORBELL_OFF) or DMA Read Doorbell Register (DMA_READ_DOORBELL_OFF).

6.3.4 Detecting the End of the DMA Transfer

If a DMA transfer proceeds without any errors, it stops automatically when finished.

Detecting End of Transfer without Errors

The normal end of a DMA transfer is detected by any of the following methods:

- Done local interrupt (pin) asserted.
- Done remote interrupt (IMWr) received.
- Channel status field of the Channel Control 1 register is Stopped, and the DMA Transfer Size register is 0x0.
- Polling of the DMA Write Interrupt Status Register (`DMA_WRITE_INT_STATUS_OFF`) or DMA Read Interrupt Status Register (`DMA_READ_INT_STATUS_OFF`).

Detecting End of Transfer with Errors

The abnormal end of a DMA transfer is detected by any of the following methods:

- Abort local interrupt (pin) asserted.
- Abort remote interrupt (IMWr) received.
- Polling of the DMA Write Interrupt Status Register (`DMA_WRITE_INT_STATUS_OFF`) or DMA Read Interrupt Status Register (`DMA_READ_INT_STATUS_OFF`).
- Channel Status field of the Channel Control 1 register is Halted. When the DMA controller detects an error, it forces the DMA to stop issuing requests for the channel. It also sets the channel status field of the Channel Control 1 register to Halted, generates an abort interrupt (if enabled), and sends an abort IMWr (if enabled). The DMA Transfer Size register indicates the remaining number bytes to be transferred, except when there is an AXI write error during a DMA read transfer. For more information, see Table 6-14.
- Channel Status field of the Channel Control 1 register is Stopped, and the DMA Transfer Size register is not 0x0. You have prematurely stopped this channel as described in "Stopping the DMA Transfer (Software Stop)" on page 486.

6.3.5 Stopping the DMA Transfer (Software Stop)

You can manually abort (stop) the DMA transfer by writing the channel number to the Doorbell Number field and writing 1 to the Stop field in `DMA_WRITE_DOORBELL_OFF`. This causes the DMA to:

- Place the channel in a Stopped state.

  The channel Status field in `DMA_CH_CONTROL1_OFF_WRCH_0` is Stopped and the value in `DMA_TRANSFER_SIZE_OFF_WRCH_0` is not 0x0.
- Wait for all outstanding pending transactions.
- Assert the abort interrupt (if it is enabled) in `DMA_WRITE_INT_STATUS_OFF`.

You might do this as part of error handling which is described in "Error Handling" on page 510, which is only necessary during software development if you incorrectly program the DMA write channel DAR.

Caution

Use This Feature With Caution

You should not use the STOP feature in operational mode. It is only a debug feature and has associated hazards.

- Before setting the Stop bit,
  - You must read the channel status field (CS) of the DMA Channel Control 1 register to ensure that the corresponding channel is Running (transferring data).
  - To eliminate the possibility of a race condition between these two actions (read and write), you should confirm the presence of the abort bit in the DMA Write Interrupt Status Register (DMA_WRITE_INT_STATUS_OFF) or DMA Read Interrupt Status Register (DMA_READ_INT_STATUS_OFF) and check the status of any fatal error if some other event has occurred.
- After a Stop event, you cannot seamlessly resume the transfer again because the DMA does not resume operation exactly from the point it stopped. Therefore, you must setup and start the complete channel context again, including the
  - linked list pointer
  - linked list structure in memory
  - related PCS and CCS values
  - When operating in non-linked list mode, setup channel context registers

6.3.6 TLP Generator Header Control

You must program bits [31:12] of the DMA Channel Control 1 Register (`DMA_CH_CONTROL1_REG_WRCH_0/DMA_CH_CONTROL1_OFF_RDCH_0`), so that the DMA TLP generator correctly sets the TC, RO, NS, AT, and FN TLP header fields for any TLPs that it generates.

6.3.7 DMA Software Register Map

This section discusses the grouping and access methods of the DMA registers. For specific information of individual DMA registers, see "Register Configuration Space Overview" on page 142. The following registers are present when the DMA is enabled.

- DMA Global Registers: these registers control global settings.

Table 6-5 DMA Global Registers

Registers	Description
B[a] +0x000	DMA Arbitration Scheme for TRGT1 Interface
B + 0x008	Number of DMA Channels
B + 0xC	DMA Write Engine Enable
B + 0x010	DMA Write Doorbell
B + 0x018	DMA Write Engine Channel Arbitration Weight Low
B + 0x01C	DMA Write Engine Channel Arbitration Weight High
B + 0x02C	DMA Read Engine Enable
B + 0x030	DMA Read Doorbell

Registers	Description
B + 0x034	DMA Global Read Control
B + 0x038	DMA Read Engine Channel Arbitration Weight Low
B + 0x03C	DMA Read Engine Channel Arbitration Weight High

a. The Base address is dependent on the configuration and is decided by the position of CS2 and CDM/ELBI bit. For more information, see "AXI DBI Access Details" on page 148.

- DMA Interrupt Registers: these registers control the generation of local interrupts and remote IMWr. The IMWr addresses and data are stored in RAM.

Table 6-6 DMA Interrupt Registers

Byte Offset	Description
B[a] + 0x4C	DMA Write Interrupt Status
B + 0x50	Not used (RsvdP).
B + 0x54	DMA Write Interrupt Mask
B + 0x58	DMA Write Interrupt Clear
B + 0x5C	DMA Write Error Status
B + 0x60	DMA Write Done IMWr Address Low
B + 0x64	DMA Write Done IMWr Address High
B + 0x68	DMA Write Abort IMWr Address Low
B + 0x6C	DMA Write Abort IMWr Address High
B + 0x70	DMA Write Channels 1 and 0 IMWr Data
B + 0x84	DMA Write Channels 3 and 2 IMWr Data
B + 0x88	DMA Write Channels 5 and 4 IMWr Data
B + 0x8C	DMA Write Channels 7 and 6 IMWr Data
B + 0x90	DMA Write Linked List Error Enable
up to +0x9C	Not used (RsvdP).
B + 0xA0	DMA Read Interrupt Status
B + 0xA4	Not used (RsvdP).
B + 0xA8	DMA Read Interrupt Mask
B + 0xAC	DMA Read Interrupt Clear
B + 0xB0	Not used (RsvdP).
B + 0xB4	DMA Read Error Status Low

Byte Offset	Description
B + 0xB8	DMA Read Error Status High
up to +0xC0	Not used (RsvdP).
B + 0xC4	DMA Read Linked List Error Enable
B + 0xC8	Not used (RsvdP).
B + 0xCC	DMA Read Done IMWr Address Low
B + 0xD0	DMA Read Done IMWr Address High
B + 0xD4	DMA Read Abort IMWr Address Low
B + 0xD8	DMA Read Abort IMWr Address High
B + 0xDC	DMA Read Channels 1 and 0 IMWr Data
B + 0xE0	DMA Read Channels 3 and 2 IMWr Data
B + 0xE4	DMA Read Channels 5 and 4 IMWr Data
B + 0xE8	DMA Read Channels 7 and 6 IMWr Data

a. The Base address is dependent on the configuration and is decided by the position of CS2 and CDM/ELBI bit. For more information, see "AXI DBI Access Details" on page 148.

- DMA Channel Context Registers: these registers (which are stored in RAM) are for the control and status information that is specific to each channel. There is a set of registers for each channel. The number of channels in the DMA is configurable in each direction (write and read) from 1 to 8 using the `CC_NUM_DMA_WR_CHAN` and `CC_NUM_DMA_RD_CHAN` configuration parameters.

Table 6-7 DMA Channel Context Registers for Each Channel

Byte Offset	Description
DMA Channel Context	
B[a]+ 0x200*(i+1)[b] + 0x100*(WR|RD)[c]	DMA Channel Control 1 Register
B + 0x374	Not used (RsvdP)
B + 0x008 + 0x200*(i+1) + 0x100*(WR|RD)	DMA Transfer Size
B + 0x00C + 0x200*(i+1) + 0x100*(WR|RD)	DMA SAR Low
B + 0x010 + 0x200*(i+1) + 0x100*(WR|RD)	DMA SAR High
B + 0x014 + 0x200*(i+1) + 0x100*(WR|RD)	DMA DAR Low
B + 0x018 + 0x200*(i+1) + 0x100*(WR|RD)	DMA DAR High

a. The Base address is dependent on the configuration and is decided by the position of CS2 and CDM/ELBI bit. For more information, see "AXI DBI Access Details" on page 148.
b. Where i is the channel number; it can have values from 0 to 7.
c. WR=0 | RD=1

Attention

For releases previous to 4.80a, the DMA registers has been programmed through an indirect addressing scheme using an index register (DMA_VIEWPORT_OFF), to reduce the address footprint in the PCI Express extended configuration space. To use the legacy method of accessing DMA registers, contact Synopsys support through SolvNetPlus.

Caution

Implementation of Channel Context Registers

- The channel context registers are stored in RAM whose contents are automatically initialized after power-on.
- You must not write to any of the context registers for a particular channel after you start the channel by writing the channel number to the Doorbell Number field of the DMA Write Doorbell Register (DMA_WRITE_DOORBELL_OFF) or DMA Read Doorbell Register (DMA_READ_DOORBELL_OFF).

Linked List Operation

When you enable linked list operation (Linked List Enable field of DMA Channel Control 1 Register (DMA_CH_CONTROL1_OFF_WRCH_0) t o 1b1), the DMA overwrites the following DMA channel context registers with the following information from linked list data elements:

- The CB, LLP, LIE, and RIE fields of the DMA Channel Control 1 register.
- DMA Transfer Size
- DMA SAR Low
- DMA SAR High
- DMA DAR Low
- DMA DAR High

Note:

- The structure of a link element is different to that of the data element. A data element has no TCB field. A link element has no LIE or RIE fields. It has no SAR, DAR, or Transfer Size DWORDs, but has LL Element Pointer DWORDs instead of the SAR DWORDs. For more information, see Relationship Between Element DWORDs and Channel Context Registers.
- The LIE and RIE bits in a LL element, only enable the Done interrupt. In non-LL mode, the RIE and LIE bits enable the Done and Abort interrupts. For more information, see Interrupts and Error Handling.

6.3.8 Programming Examples

This section provides a programming and operation flow for each of the following scenarios:

- "1 MB Write Transfer On Write Channel #0 Initiated by Local CPU (Non LL mode)" on page 491
- "1 MB Read Transfer On Read Channel #0 Initiated by Local CPU (Non LL mode)" on page 493
- "1 MB Write Transfer On Write Channel #5 Initiated by Remote CPU (Non LL mode)" on page 494
- "1 MB Read Transfer On Read Channel #5 Initiated by Remote CPU (Non LL mode)" on page 495
- "Linked List Mode Programming Example (Write Channel #0)" on page 496

Note

The order in which the registers in the following examples are programmed is not important, except for the Doorbell register, which initiates the DMA transfer. Therefore, to start both a "1 MB Write Transfer On Write Channel #0 Initiated by Local CPU (Non LL mode)" on page 491 and a "1 MB Read Transfer On Read Channel #0 Initiated by Local CPU (Non LL mode)" on page 493, just execute the commands in Table 6-8 and then execute the commands in Table 6-9. If, as mentioned in "Starting the DMA Transfer" on page 485, you want both channels to start at (approximately) the same time, then do not execute the last command (at 0x280) in Table 6-8 until after you have executed all the commands in Table 6-9.

Attention

You must enable the DMA before you start a DMA channel. Therefore you must write '1' to DMA Read Engine Enable or DMA Write Engine Enable register before you write '1' to the DMA Read Doorbell or DMA Write Doorbell register.

Hint

You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html` for more programming examples. For more information on VTB, see the "Integrating the Controller and VIP using VTB section" in the "Integration" chapter of the *User Guide*.

6.3.8.1 1 MB Write Transfer On Write Channel #0 Initiated by Local CPU (Non LL mode)

In this example, the IMWr generation is disabled, as the local CPU initiates the DMA transfer. The local CPU is interrupted using the `edma_int` bus. The SAR is the address of the local memory, and the DAR is the address of the remote memory, as shown in Figure 6-12. Table 6-8 provides the programming information for this example transfer.

Figure 6-12 1 MB Write DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0

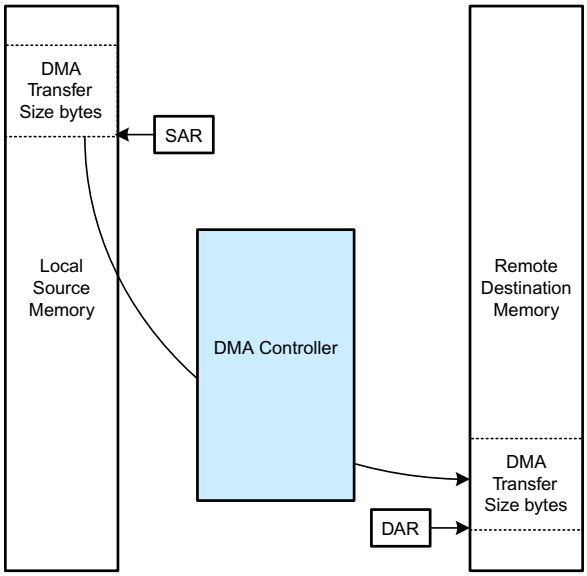


Table 6-8 1 MB Write DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0

Address (0x8_0000+)	Name	Value
0xC	DMA Write Engine Enable You must not write 0 to this register. Even temporarily writing 0 to this register resets the DMA logic. For more information, see DMA Write Engine Enable Register (`DMA_WRITE_ENGINE_EN_OFF`).	0x1
0x054	DMA Write Interrupt Mask	0x0
0x200	DMA Channel Control 1 register ■ Local Interrupt Enable (LIE) =1 ■ Remote Interrupt Enable (RIE) =0 ■ AT, RO, NS, TC, Function Number =0	0x00000008
0x208	DMA Transfer Size	0x00100000
0x20C	DMA SAR Low	0xBEEF_BEE0
0x210	DMA SAR High	0x0000_0000
0x214	DMA DAR Low	0xCAFE_CAF0
0x218	DMA DAR High	0x0000_0000
0x010	DMA Write Doorbell	0x0

6.3.8.2 1 MB Read Transfer On Read Channel #0 Initiated by Local CPU (Non LL mode)

In this example, the IMWr generation is disabled, as the local CPU initiates the DMA transfer. The local CPU is interrupted using the `edma_int` bus. The SAR is the address of the remote memory, and the DAR is the address of the local memory, as shown in Figure 6-13. Table 6-9 provides the programming information for this example transfer.

Figure 6-13 1 MB Read DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0

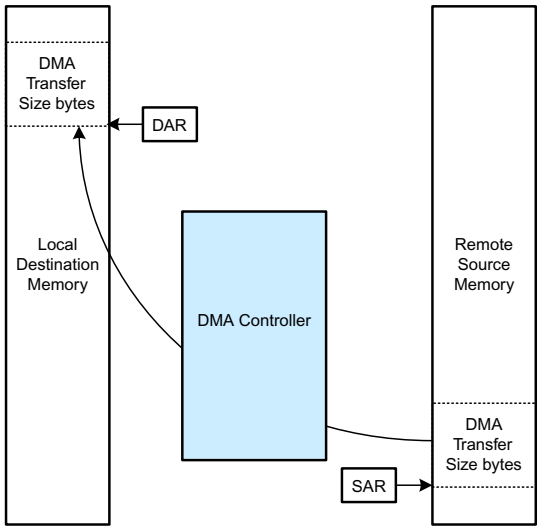


Table 6-9 DMA Register Setup

Address (0x8_0000+)	Name	Value
0x02C	DMA Read Engine Enable You must not write 0 to this register. Even temporarily writing 0 to this register resets the DMA logic. For more information, see DMA Read Engine Enable Register (`DMA_READ_ENGINE_EN_OFF`).	0x1
0x0A8	DMA Read Interrupt Mask	0x0
0x300	DMA Channel Control 1 register ■ Local Interrupt Enable (LIE) =1 ■ Remote Interrupt Enable (RIE) =0 ■ AT, RO, NS, TC, Function Number =0	0x00000008
0x308	DMA Transfer Size	0x00100000
0x30C	DMA SAR Low	0xBEEF_BEE0
0x310	DMA SAR High	0x0000_0000
0x314	DMA DAR Low	0xCAFE_CAF0
0x318	DMA DAR High	0x0000_0000

Address (0x8_0000+)	Name	Value
0x030	DMA Read Doorbell	0x0

6.3.8.3 1 MB Write Transfer On Write Channel #5 Initiated by Remote CPU (Non LL mode)

In this example, the local interrupt generation is disabled, as the remote CPU initiates the DMA transfer. The remote CPU is interrupted using an IMWr. The SAR is the address of the local memory, and the DAR is the address of the remote memory, as shown in Figure 6-14. Table 6-10 provides the programming information for this example transfer.

Figure 6-14 1 MB Write DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0

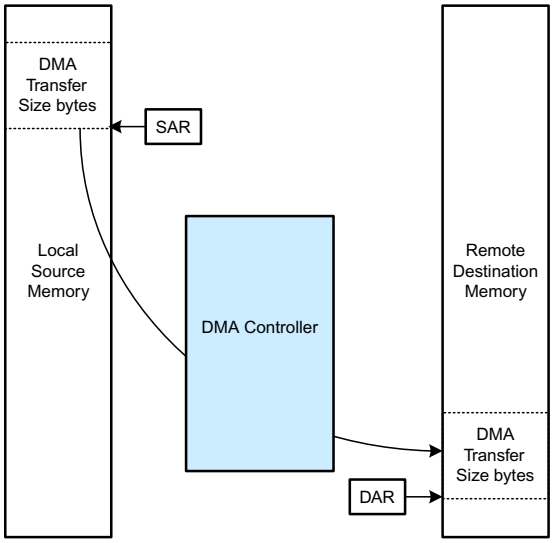


Table 6-10 DMA Register Setup

Address (0x8_0000+)	Name	Value
0xC	DMA Write Engine Enable You must not write 0 to this register. Even temporarily writing 0 to this register resets the DMA logic. For more information, see DMA Write Engine Enable Register (`DMA_WRITE_ENGINE_EN_OFF`).	0x1
0x060 / 0x064	DMA Write Done IMWr Address Low and High	your IMWr Address #1
0x068 / 0x06C	DMA Write Abort IMWr Address Low and High	your IMWr Address #2
0x088	DMA Write Channel 5 IMWr Data	your IMWr Data

Address (0x8_0000+)	Name	Value
0xC00	DMA Channel Control 1 register ■ Local Interrupt Enable (LIE) =0 ■ Remote Interrupt Enable (RIE) =1 ■ AT, RO, NS, TC, Function Number =0	0x00000010
0xC08	DMA Transfer Size	0x00100000
0xC0C	DMA SAR Low	0xBEEF_BEE0
0xC10	DMA SAR High	0x0000_0000
0xC14	DMA DAR Low	0xCAFE_CAF0
0xC18	DMA DAR High	0x0000_0000
0x010	DMA Write Doorbell	0x5

6.3.8.4 1 MB Read Transfer On Read Channel #5 Initiated by Remote CPU (Non LL mode)

In this example, the local interrupt generation is disabled, as the remote CPU initiates the DMA transfer. The remote CPU is interrupted using an IMWr. The SAR is the address of the remote memory, and the DAR is the address of the local memory, as shown in Figure 6-15. Table 6-11 provides the programming information for this example transfer.

Figure 6-15 1 MB Read DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0

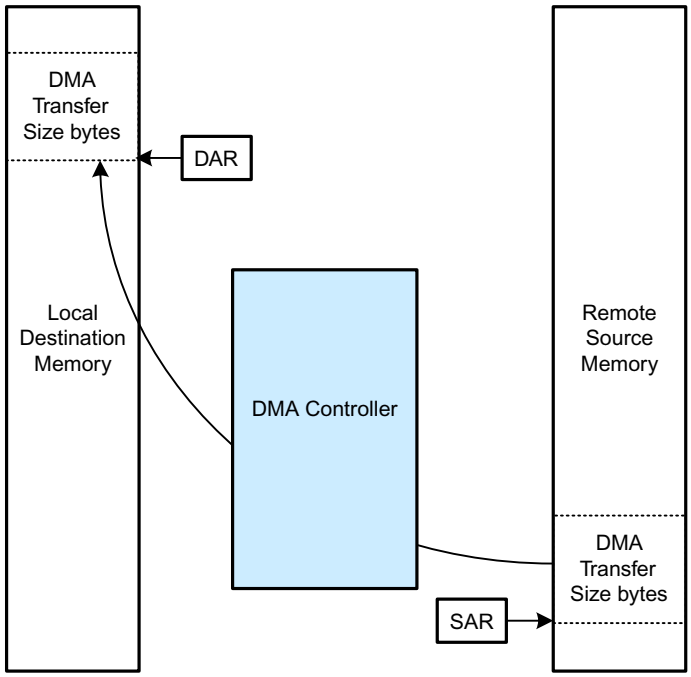


Table 6-11 DMA Register Setup

Address (0x8_0000+)	Name	Value
0x02C	DMA Read Engine Enable You must not write 0 to this register. Even temporarily writing 0 to this register resets the DMA logic. For more information, see DMA Read Engine Enable Register (`DMA_READ_ENGINE_EN_OFF`).	0x1
0x0CC / 0x0D0	DMA Read Done IMWr Address Low and high	your IMWr Address #1
0x0D4 / 0x0D8	DMA Read Abort IMWr Address Low and High	your IMWr Address #2
0x0E4	DMA Read Channel 5 IMWr Data	your IMWr Data
0xD00	DMA Channel Control 1 register ■ Local Interrupt Enable (LIE) =0 ■ Remote Interrupt Enable (RIE) =1 ■ AT, RO, NS, TC, Function Number =0	0x00000010
0xB08	DMA Transfer Size	0x00100000
0xD0C	DMA SAR Low	0xBEEF_BEE0
0xD10	DMA SAR High	0x0000_0000
0xD14	DMA DAR Low	0xCAFE_CAF0
0xD18	DMA DAR High	0x0000_0000
0x030	DMA Read Doorbell	0x5

6.3.8.5 Linked List Mode Programming Example (Write Channel #0)

This write channel example corresponds to Figure 6-24 on page 518, with the following features:

- Recycled linked list (LL) of nine data elements[1]and one link element.
- Transfer is initiated by local CPU.
- Each data element corresponds to a 1 MB (0x0010_0000) write transfer.
- The SARs for each element are separated by 2 MB (0x0020_0000). The DAR for each element are separated by 16 MB (0x0100_0000). The SAR is the address of the local memory, and the DAR is the address of the remote memory, as shown in Figure 6-16.
- The LL transfer list is located at address 0x0000_0200 in local memory.
- No remote interrupt is generated so the IMWr generation is disabled, as the local CPU initiates the DMA transfer. The local CPU is interrupted using the `edma_int` bus.
- There is a Watermark interrupt at element #5, and an Empty interrupt at element #8. Both of these interrupts are done interrupts (as opposed to abort interrupts).

1. Also known as descriptors.

You should familiarize yourself with the Linked List Flow for Producer and Consumer as shown in Figure 6-23. These are the main steps for this process in this example:

Step 1: Create TL in local memory

- Create the 10 elements as per Table 6-12.
  - For elements 0, 1, 2, 3, 4, 6, 7: RIE, LIE, LLP, CB =0,0,0,1
  - For element 5: RIE, LIE, LLP, CB =0,1,0,1
  - For element 8: RIE, LIE, LLP, CB =0,1,0,1
  - For element 9: LLP, TCB, CB =1,1,0
- For more information, see "PCS-CCS-CB-TCB Producer-Consumer Synchronization" on page 517 and "Linked List Mode" on page 475.

Step 2: Program and Start DMA

- Configure and start the DMA registers as per Table 6-13.

Step 4: Wait for done interrupt and recycle TL

- For more information, see "Using Interrupts for Linked List Producer-Consumer Synchronization" on page 516.
- When recycling elements, consider "Element Recycling" on page 520.

Attention

Software Programming of an Endpoints LIE and RIE Bits for Linked List Transfers.

If the DMA driver is running on the host and the interrupt service routine is reading local interrupts to determine if the transfer is successful, then you must set LIE and RIE in the same element and you should mask or ignore the local interrupt pin. Setting RIE and LIE in element A followed by RIE (only) in element B is not a verified usage scenario.

Figure 6-16 1 MB Write DMA Transfer from SAR to DAR

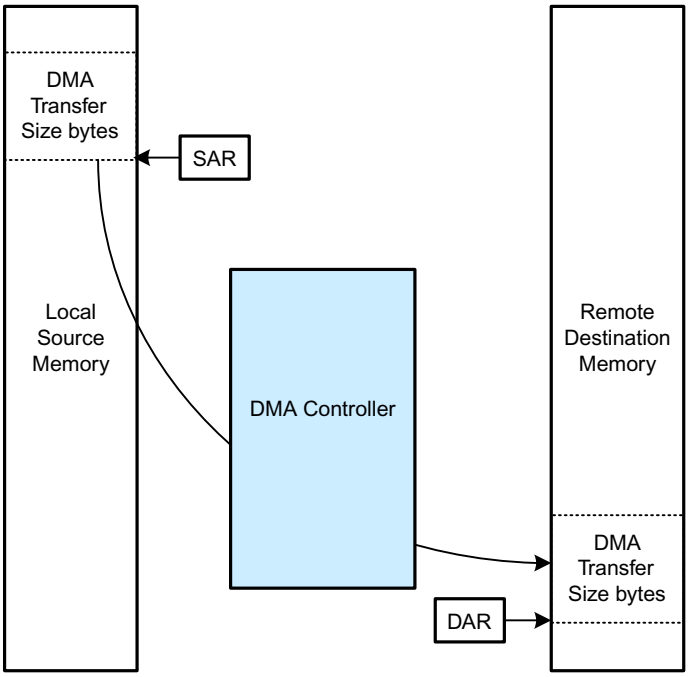


Table 6-12 Linked List Element Initial Setup

Element #	Address	Data	Label
0	0x0000_0200	0x00000001	Channel Control
0	0x0000_0204	0x00100000	Transfer Size
0	0x0000_0208	0xBEEF_BEE0	SAR Low
0	0x0000_020C	0x0	SAR High
0	0x0000_0210	0xCAFE_CAF0	DAR Low
0	0x0000_0214	0x0	DAR High
1	0x0000_0218	0x00000001	Channel Control
1	0x0000_021C	0x00100000	Transfer Size
1	0x0000_0220	0xBF0F_BEE0	SAR Low
1	0x0000_0224	0x0	SAR High
1	0x0000_0228	0xCBFE_CAF0	DAR Low
1	0x0000_022C	0x0	DAR High
2	0x0000_0230	0x00000001	Channel Control
2	0x0000_0234	0x00100000	Transfer Size

Element #	Address	Data	Label
2	0x0000_0238	0xBF2F_BEE0	SAR Low
2	0x0000_023C	0x0	SAR High
2	0x0000_0240	0xCCFE_CAF0	DAR Low
2	0x0000_0244	0x0	DAR High
3	0x0000_0248	0x00000001	Channel Control
3	0x0000_024C	0x00100000	Transfer Size
3	0x0000_0250	0xBF4F_BEE0	SAR Low
3	0x0000_0254	0x0	SAR High
3	0x0000_0258	0xCDFE_CAF0	DAR Low
3	0x0000_025C	0x0	DAR High
4	0x0000_0260	0x00000001	Channel Control
4	0x0000_0264	0x00100000	Transfer Size
4	0x0000_0268	0xBF6F_BEE0	SAR Low
4	0x0000_026C	0x0	SAR High
4	0x0000_0270	0xCEFE_CAF0	DAR Low
4	0x0000_0274	0x0	DAR High
5	0x0000_0278	0x00000009	Channel Control
5	0x0000_027C	0x00100000	Transfer Size
5	0x0000_0280	0xBF8F_BEE0	SAR Low
5	0x0000_0284	0x0	SAR High
5	0x0000_0288	0xCFFE_CAF0	DAR Low
5	0x0000_028C	0x0	DAR High
6	0x0000_0290	0x00000001	Channel Control
6	0x0000_0294	0x00100000	Transfer Size
6	0x0000_0298	0xBFAF_BEE0	SAR Low
6	0x0000_029C	0x0	SAR High
6	0x0000_0290	0xD0FE_CAF0	DAR Low
6	0x0000_0294	0x0	DAR High
7	0x0000_0298	0x00000001	Channel Control

Element #	Address	Data	Label
7	0x0000_029C	0x00100000	Transfer Size
7	0x0000_02A0	0xBFCF_BEE0	SAR Low
7	0x0000_02A4	0x0	SAR High
7	0x0000_02A8	0xD1FE_CAF0	DAR Low
7	0x0000_02AC	0x0	DAR High
8	0x0000_02B0	0x00000009	Channel Control
8	0x0000_02B4	0x00100000	Transfer Size
8	0x0000_02B8	0xBFEF_BEE0	SAR Low
8	0x0000_02BC	0x0	SAR High
8	0x0000_02C0	0xD2FE_CAF0	DAR Low
8	0x0000_02C4	0x0	DAR High
9	0x0000_02C8	0x00000006	Channel Control
9	0x0000_02CC	0x0	Reserved
9	0x0000_02D0	0x0000_0200	Linked List Element Pointer Low
9	0x0000_02D4	0x0	Linked List Element Pointer High

Table 6-13 DMA Register Setup

Address (0x8_0000+)	Name	Value
0xC	DMA Write Engine Enable You must not write 0 to this register. Even temporarily writing 0 to this register resets the DMA logic. For more information, see DMA Write Engine Enable Register (`DMA_WRITE_ENGINE_EN_OFF`).	0x1
0x054	DMA Write Interrupt Mask	0x0
0x090	DMA Write Linked List Error Enable ■ LLLAIE =1	0x1
0x200	DMA Channel Control 1 register ■ Linked List Enable (LLE) =1 ■ Consumer Cycle Status (CCS) =PCS =1 ■ AT, RO, NS, TC, Function Number =0	0x00000300
0x21C	DMA Linked List Pointer Low	0x0000_0200
0x220	DMA Linked List Pointer High	0x0000_0000
0x010	DMA Write Doorbell	0x0

6.4 Advanced DMA Information and Operation

This section discusses advanced features and operation of the DMA. This following topics are covered in this section:

- "Overview" on page 501
- "Internal Architecture" on page 504
- "DMA Ordering" on page 505
- "Read Buffer" on page 506
- "PCIe TAGs" on page 507
- "Error Handling" on page 510
- "DMA Debug" on page 512
- "Linked List Operation" on page 513
- "Multichannel Arbitration" on page 520

6.4.1 Overview

The DMA write and read channels operate independently to maximize the performance of the DMA read and write data transfers over the PCIe link.

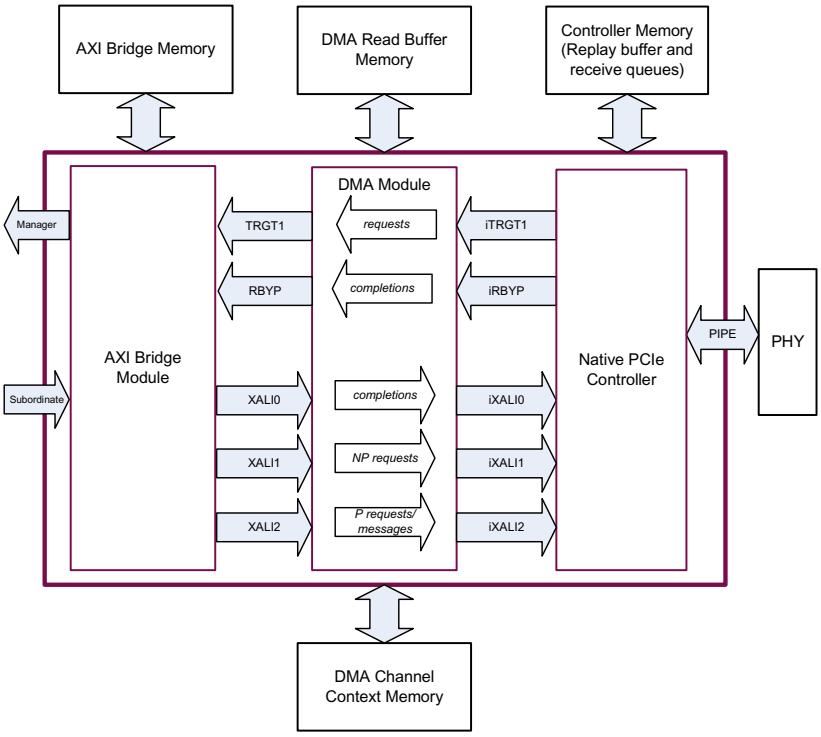
The DMA can simultaneously perform the following types of memory transactions:

- DMA write: transfer (copy) of a block of data from local memory to remote memory
- DMA read: transfer (copy) of a block of data from remote memory to local memory

After you have programmed and started a DMA transfer (see "Using the DMA" on page 483), the DMA transfers the data as described in "DMA Write Transfer" on page 469 and "DMA Read Transfer" on page 469.

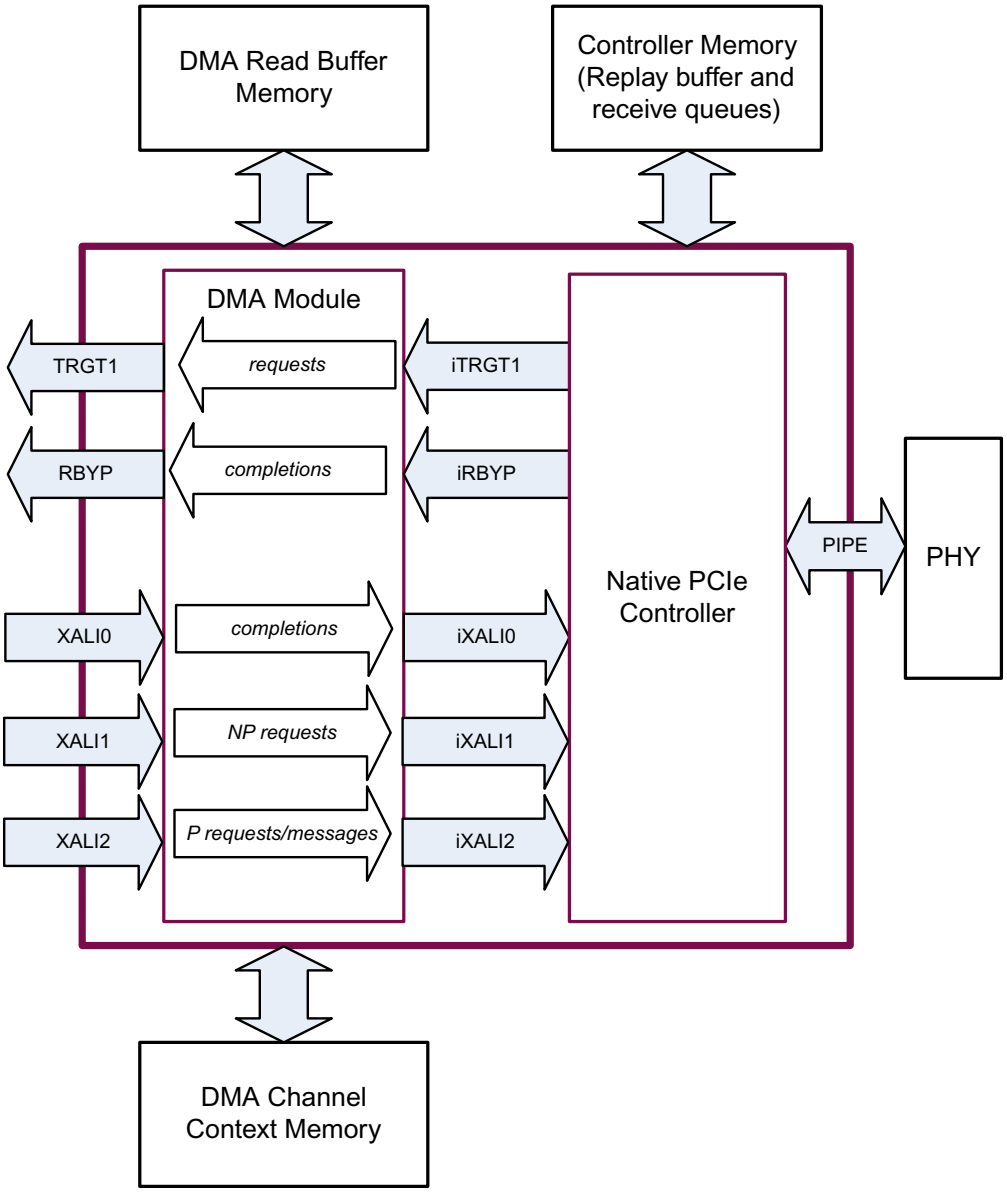
When the PCIe controller is configured to use the AXI bridge, then the DMA is located between the native PCIe controller and the AXI bridge module as shown in Figure 6-17. It must share, with your normal non-DMA traffic, access to the AXI bridge, and PCIe controller and link. This sharing process is described in "Internal Architecture" on page 504. The internal TRGT1, RBYP, and XALI0/1 interfaces have been renamed to iTRGT1, iRBYP, and iXALI0/1.

Figure 6-17 Location of the DMA Controller when the PCIe includes the AXI Bridge



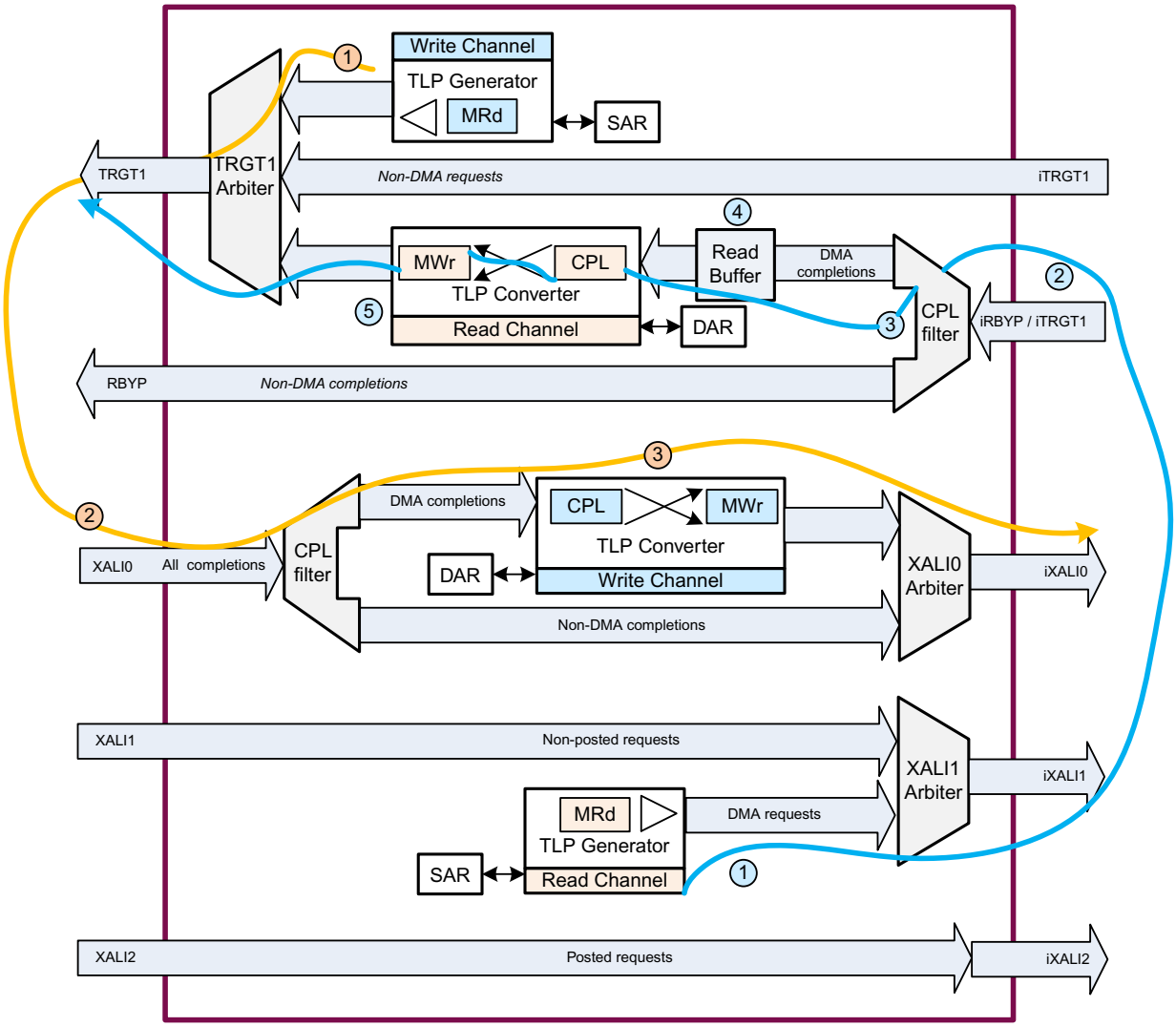
For definitions of acronyms used for block and interface names, see "Terms and Abbreviations" on page 22.

Figure 6-18 Location of the DMA Controller when the PCIe does not include the AXI Bridge



6.4.2 Internal Architecture

Figure 6-19 Operation of DMA Read (blue) and Write (orange) Channels



DMA Write Transfer

The DMA injects multiple MRd requests of size less than or equal to Max_Payload_Size into the inbound request path. The TLP addresses are derived from the "Source and Destination Address Registers (SAR, DAR)" on page 483. When the DMA data transfer is complete, the CPU is notified. For more information, see "Interrupts and Error Handling" on page 469. Referring to the orange paths in Figure 6-19:

1. The DMA generates and sends READ requests to your local application. Because the TRGT_CPL_LUT is always enabled, it uses one of the shared DMA/non-DMA TRGT_CPL_LUT IDs instead of a PCIe TAG.
2. Your application returns data on the XALI0 interface (or AXI bridge master) in response to these requests. Data must be returned in strict order within a single channel.

3. The DMA converts the CPL TLPs to WRITE requests and sends these to the PCIe native controller for transmission to the remote link partner.

DMA Read Transfer

The DMA injects multiple MRd requests of size less than or equal to the minimum of {Max_Read_Request_Size, Max_Payload_Size_Supported} into the outbound request path. The TLP addresses are derived from the "Source and Destination Address Registers (SAR, DAR)" on page 483. When the DMA data transfer is complete, the CPU is notified. For more information, see "Interrupts and Error Handling" on page 469. Referring to the blue paths Figure 6-19:

1. The DMA generates and sends READ requests to the remote link partner. It uses a DMA tag.
2. The link partner returns data in response to these requests on the iRBYP or iTRGT1 interface.
3. The DMA module filters this data from non-DMA traffic on iRBYP/iTRGT1 and sends it to the Read Buffer. The DMA module identifies DMA traffic by looking for PCIe completion TLPs that have a DMA tag.
4. The Read Buffer reorders completions that have been returned out-of-order. This is to ensure that the DMA writes data to the application in address-order.
5. The DMA converts the completion TLPs to WRITE requests and sends these to your application on TRGT1 (or AXI bridge master).

In Figure 6-19, the XALI1 arbitration priority is Read Channel MSI (highest), Write Channel MSI, Non-DMA requests, Read Channel MRd. The TRGT1 arbitration priority is programmable (using the `DMA_CTRL_DATA_ARB_PRIOR_OFF` register) and defaults to Non-DMA requests (highest), Write Channel MRd (DMA data requests and LL element/descriptor access), Read Channel MRd (LL element/descriptor access), Read Channel MWr.

6.4.3 DMA Ordering

The DMA assumes that all DMA read channel data must be written to the local application memory in strict address order. For a DMA read transfer, each original outgoing request has a unique PCIe tag. The remote link partner or intermediate switch does not have to preserve the order of CPLs corresponding to such individual requests. When the DMA receives the CPLs out of order, it reorders that data in the "Read Buffer" on page 506 as described in "DMA Read Transfer" on page 469. DMA write transfer data is sourced from the local application memory and sent to the remote memory in address order.

Note For more information on ordering, see "Advanced Information: Advanced Ordering Information" on page 703.

AXI Ordering (DMA)

IDs '0' to (`CC_MAX_MSTR_TAGS_AXI -1`) are used for non-DMA and DMA non-posted requests (including linked list element access). The write channel uses a random AXI ID (per channel) for all DMA non-posted requests sent to the AXI manager. The AXI bridge maintains the order of CPLs returned to DMA within each channel. You can observe the DMA master sideband signals `*misc_info_dma` to distinguish a DMA non-posted request from a non-DMA non-posted request with same ID. ID 0 is always used for non-DMA and DMA posted requests.

AXI Manager CPL-P Ordering Impact on Performance

The AXI ordering manager ensures that the rule "CPL must not pass a previously-issued P" is obeyed for each virtual channel (VC). It does this by halting the unloading of completions from the native controller's receive queues by the AXI bridge when there are outstanding posted transactions on the AXI manager interface. In a system with a single VC, this means that strongly ordered non-DMA completions at the head of the completion queue can block weakly ordered non-DMA completions and impact performance.

DMA outbound read completions are weakly ordered and should be mapped to a different VC (in a multi-VC system), enabling the controller to send the completions to the DMA read buffer when strong and weakly ordered read completions are interleaved on the wire. In a single-VC system, the DMA can still share the VC without impact from these enforced ordering rules if the RO bit (relaxed ordering) is set on DMA read engine transactions.

Note For more information on AXI ordering, see "AXI Bridge Ordering" on page 408.

Native Controller Write Channel Deadlock Avoidance

Special consideration must be given to the proper use of the three client interfaces and you must implement a non-posted tracker in your application logic as described in "TRGT1 Packet Grant and Halt" on page 371. For more information, see "Native Controller Operation (without AXI Bridge)" on page 481.

6.4.4 Read Buffer

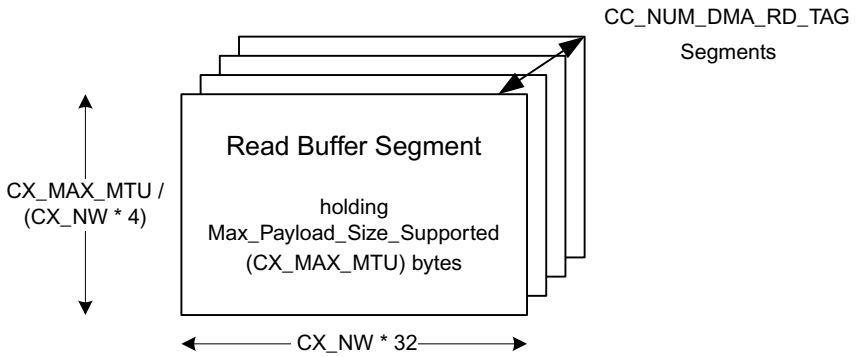
The DMA read buffer fulfills two functions:

- Reordering of CPLs that are returned out-of-order as described in "PCIe TAGs" on page 507.
- Buffering between the DMA and the AXI bridge master (or application if bridge not used). This is needed for buffering CPLs when your application applies back pressure (stalling/halting), as the controller advertises infinite CPL credits.

The DMA read buffer size is automatically configured at build time and is logically arranged as a segmented-buffer, as shown in Figure 6-20. It has `CC_NUM_DMA_RD_TAG` segments. For more information on configuration parameter `CC_NUM_DMA_RD_TAG`, see "PCIe TAGs" on page 507. The size of each segment is `CX_MAX_MTU`[1]. This is the value of the *Max_Payload_Size_Supported* field of the Device Capability register in the PCIe Express Capability Structure. At enumeration time, the RP can program the *Max_Payload_Size* field in the Device Control register in the PCIe Express Capability Structure with any value less than or equal to this.

1. The only other value that could have been picked during the design is 4 KB, which is the maximum amount of data a MRd request can ever request in a PCIe system.

Figure 6-20 DMA Read Buffer



The DMA read buffer size is as follows:

- **Width**: Same as the controller datapath width =`CX_NW`*32
- **Depth**: `CC_DMA_SEG_BUF_DEPTH` =`CC_NUM_DMA_RD_TAG` * `CX_MAX_MTU` / (`CX_NW`*4) +2[1]

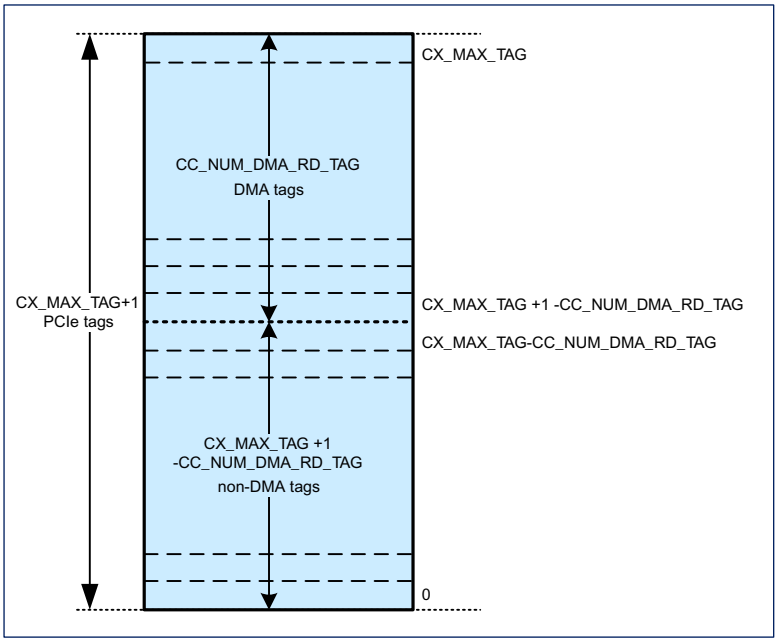
The default segment size is 256 bytes, and the default number of DMA tags is 16. Therefore the default read buffer size is 16 * 256 =4 KB.

6.4.5 PCIe TAGs

The combined number of PCIe tags assigned to the DMA read channel and your application (or AXI bridge) is always equal to `CX_MAX_TAG`+1. You set the `CC_NUM_DMA_RD_TAG` configuration parameter to specify how many of the `CX_MAX_TAG`+1 PCIe tags are reserved for DMA MRd request TLP generation. The remaining tags are assigned to non-DMA transfers generated by your application on XALI0/1/2 (or the AXI bridge subordinate). Figure 6-21 shows the division of PCIe tags between the DMA controller and the rest of the PCIe controller. It also shows the value range of each pool of tags.

1. This extra two entries only apply for cores with 64-bit and 128-bit datapaths.

Figure 6-21 PCIe Tag Allocation For DMA/Non-DMA MRd Traffic Using CC_NUM_DMA_RD_TAG

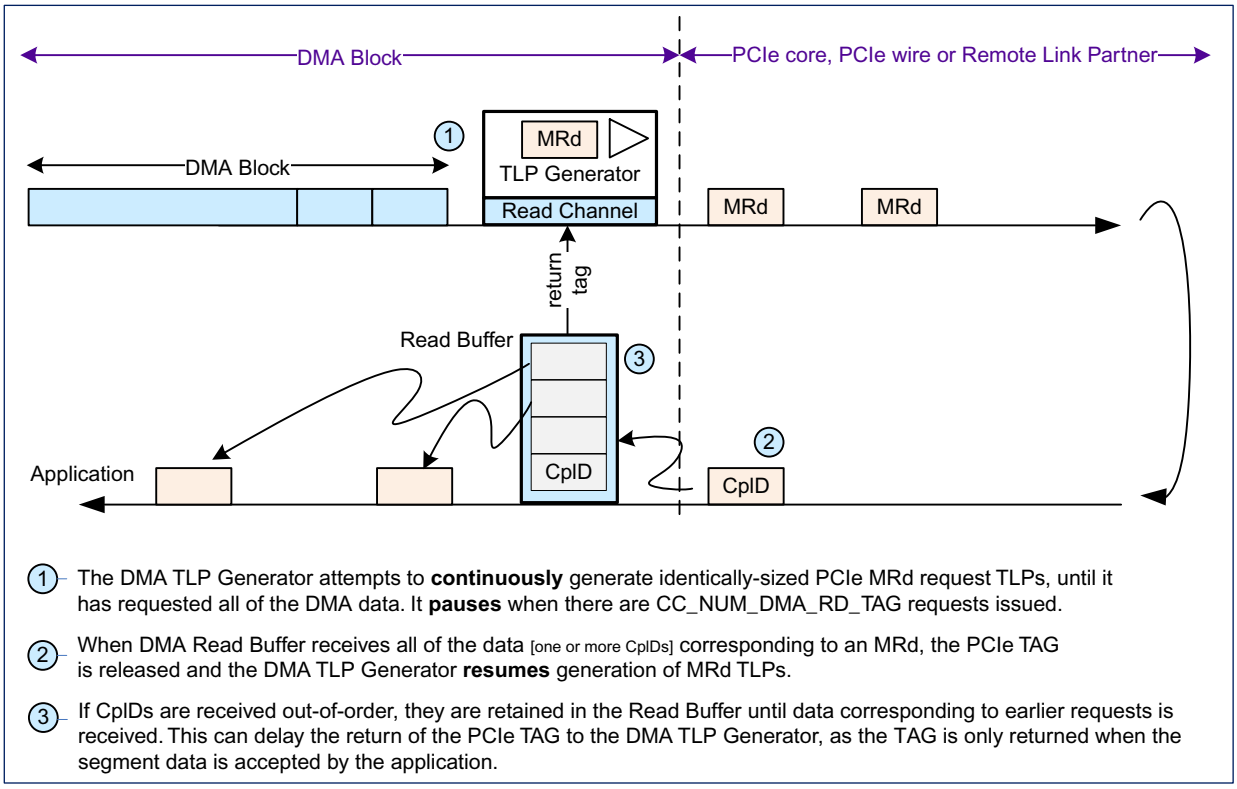


6.4.6 Example Transfer

The DMA never has more than `CC_NUM_DMA_RD_TAG` simultaneous MRd requests outstanding on the PCIe link. To demonstrate this, consider a DMA read transfer as follows. Figure 6-22 shows a scenario where there are four DMA tags (`CC_NUM_DMA_RD_TAG=4`), and the remote link partner is returning out-of-order CPLs. An example of out-of-order CPL delivery would see the CPL (or CPLs) for the first MRd request arriving after the CPL for the second MRd request.

- There are two MRd requests in transit on the PCIe link.
- There is one out-of-order CplD in the DMA read buffer.
- There is one CplD in transit on the PCIe link.

Figure 6-22 DMA Read Transfer Decomposition into Multiple MRd Requests with CC_NUM_DMA_RD_TAG =4



The DMA segments the transfer into a number (up to a maximum of `CC_NUM_DMA_RD_TAG`) of MRd requests. The size of each request is limited to the minimum of {*Max_Read_Request_Size*, *Max_Payload_Size*}.

- When a CplD TLP for an MRd request returns from the PCIe Wire out-of-order:
  - The CplD is stored in the DMA read buffer.
  - The tag is not released back to the DMA tag pool, and the DMA cannot generate another MRd request with this tag number.
  - When the CplD from all of the previous MRd requests have been received, the DMA completes the next steps:
    - The CplD is converted to an MWr and forwarded to your application (or AXI bridge master for writing to the AXI bus).
    - The tag is released back to the DMA tag pool.
    - The DMA generates another MRd request with this tag number.
- When a CplD TLP for an MRd request returns from the PCIe Wire in-order:
  - The CplD is converted to an MWr and forwarded to your application (or the AXI bridge master for writing to the AXI bus).
  - The TAG is released back to the DMA tag pool.
  - The DMA generates another MRd request with this tag number.

6.4.7 Error Handling

The DMA supports full AXI and PCIe error handling in conjunction with the PCIe controller. Table 6-14, and Table 6-15 list the possible sources of an error for DMA read and write transfers. In general,

Table 6-14 Possible Sources of an Error during a DMA Read Transfer

Source	Type	Description
Application Write Error Detected	Fatal	The DMA read channel has received an error response from the AXI bus at the bridge master interface (or TRGT1 interface when the AXI bridge is not used) while writing posted data to it. This error is fatal. ■ You must restart the transfer from the beginning because the channel context is corrupted. ■ The MWr is not rolled back. ■ The AXI bridge master does not report this error to software through error logging or MSG. **Note**: During linked list mode, the Application Write Error only corrupts the current element in the linked list.
Unsupported Request (UR)	Non-fatal	The DMA read channel has received a PCIe UR CPL status from the remote device in response to the MRd request.
Completer Abort (CA)	Non-fatal	The DMA read channel has received a PCIe CA CPL status from the remote device in response to the MRd request.
CPL Time Out	Non-fatal	The DMA read channel has timed-out while waiting for the remote device to respond to the MRd request, or a malformed CplD has been received.
Data Poisoning	Fatal	The DMA read channel has detected data poisoning in the CPL from the remote device in response to the MRd request. The DMA read channel drops the completion and then be halted. The CX_FLT_MASK_UR_POIS filter rule does not affect this behavior.
Linked List Element Fetch Error Detected	Fatal	The DMA read channel has received an error response from the AXI bus (or TRGT1 interface when the AXI bridge is not used) while reading a linked list element from local memory.

Table 6-15 Possible Sources of an Error during a DMA Write Transfer

Source	Type	Description
Application Read Error Detected	Fatal	The DMA write channel has received an error response from the AXI bus at the bridge master interface (or TRGT1 interface when the AXI bridge is not used) while reading data from it. ■ The response TLP is discarded by the DMA and is not converted to an outbound MWr TLP. ■ The UR or CA CPL status TLP generated by the bridge is discarded by the DMA. ■ The AXI bridge master does not report this error to software through error logging or MSG.

Source	Type	Description
Linked List Element Fetch Error Detected	Fatal	The DMA write channel has received an error response from the AXI bus at the bridge master interface (or TRGT1 interface when the AXI bridge is not used) while reading a linked list element from local memory. ■ The UR or CA CPL status TLP generated by the bridge is discarded by the DMA. ■ The AXI bridge master1 does not report this (or for non-DMA traffic) error to software through error logging or MSG.

When the DMA detects an error, it performs the following actions (for the channel with the error):

- Stops issuing new requests to the remote link partner.
- Stops issuing new requests to your application.
  - All data in the "Read Buffer" on page 506, that is received before the error is detected, is valid and is sent to your application. In the case of a non-fatal error, this is reflected in the DMA Transfer Size Register `DMA_TRANSFER_SIZE_OFF_WRCH_0`.
  - All data in the read buffer (for that channel), that is received after the error is detected, is discarded.
- Places the DMA channel in a halted state (channel status field of the DMA Channel Control 1 Register is halted).
- Waits for all outstanding requests and CPLs to complete.
- Generates the abort interrupt or IMWr, if enabled, as outlined in "Interrupts and Error Handling" on page 469.

For a DMA read transfer, you can read the DMA Read Error Status High Register (`DMA_READ_ERR_STATUS_HIGH_OFF`) and DMA Read Error Status Low Register (`DMA_READ_ERR_STATUS_LOW_OFF`) to identify the source of the error. For a DMA write transfer (in linked list mode), you can read the DMA Write Error Status Register (`DMA_WRITE_ERR_STATUS_OFF`) to identify the source of the error. For a DMA write transfer (in non linked list mode), you can read the `Application Read Error Detected` field of the DMA Write Error Status Register (`DMA_WRITE_ERR_STATUS_OFF.APP_READ_ERR_DETECT`) to identify the source of the error. After your software has finished error and interrupt routine handling for Non-Fatal errors, you can request the DMA to continue processing (for the channel with the error) by:

1. Reprogramming the SAR based on the number of bytes remaining in the DMA Transfer Size register
2. Writing the channel number to the Doorbell Number field of the DMA Write Doorbell Register (`DMA_WRITE_DOORBELL_OFF`) or DMA Read Doorbell Register (`DMA_READ_DOORBELL_REG`).

Caution You cannot continue after a Fatal Error is detected, as the channel context is corrupted. The actions in progress are not rolled back. To exit from this condition, you need to Soft Reset the complete DMA controller, as described in DMA Write Engine Enable Register (`DMA_WRITE_ENGINE_EN_OFF`) and DMA Read Engine Enable Register (`DMA_READ_ENGINE_EN_OFF`).

When an error occurs in linked list mode, DMA Linked List Pointer Low Register (`DMA_LLP_LOW_OFF_WRCH_0`) is not incremented by the DMA, but remains pointing to the element that caused the error.

Error Handling Assistance by Remote Software

During a DMA write transfer, the DMA is not aware of MWr errors that are detected in the remote device, because an MWr is a posted request and has no corresponding completion TLP. In these cases, your remote CPU must manually abort the DMA transfer by writing the channel number to the Doorbell Number field, and 1 to the Stop field of the DMA Write Doorbell Register (`DMA_WRITE_DOORBELL_OFF`). This causes the DMA to:

- Place the channel in a Stopped state. The channel Status field of the DMA Channel Control 1 Register (`DMA_CH_CONTROL1_OFF_WRCH_0`) is Stopped and the DMA Transfer Size register is not 0x0.
- Wait for all outstanding pending transactions.
- Assert the abort interrupt if it is enabled

For more information, see "Stopping the DMA Transfer (Software Stop)" on page 486. Alternatively, your remote CPU can return an error Msg to the local CPU. The local CPU can then stop the channel by setting the Stop field. You can re-start the transfer by rolling back the SAR based on the number of bytes remaining in the DMA Transfer Size Register (`DMA_TRANSFER_SIZE_OFF_WRCH_0`) subject to the restrictions outlined in "Stopping the DMA Transfer (Software Stop)" on page 486.

Note This procedure is only necessary during software development; when you incorrectly program the DMA write channel DAR. An MWr error is fatal and you must resolve it by fixing your software. In normal operation MWr errors are not expected to occur.

6.4.8 DMA Debug

To assist you in debugging your hardware, use the following outputs to identify DMA traffic (transfer requests, and linked list element MRds) and non-DMA traffic.

Table 6-16 DMA Debug Outputs

Signal Name	Where Used
armisc_info_dma	AXI manager
awmisc_info_dma	AXI manager
d_trgt1_req_dma	Native controller (without AXI bridge)
d_trgt1_req_dma_channel[2:0]	Native controller (without AXI bridge)

Soft Reset

To assist you in debugging your software, use the DMA write engine enable field of the DMA Write Engine Enable Register (`DMA_WRITE_ENGINE_EN_OFF`) to reset the DMA write logic. This might be necessary during software development, if you incorrectly program and hang the DMA. Similarly, use the DMA read engine enable field of the DMA Read Engine Enable Register (`DMA_READ_ENGINE_EN_OFF`) to reset the DMA read logic. This soft reset does not clear the DMA configuration registers.

Stop Bit

To assist you in debugging your software, use the Stop bit of the DMA Write Doorbell Register (`DMA_WRITE_DOORBELL_OFF`), or of the DMA Read Doorbell Register (`DMA_READ_DOORBELL_REG`), to

prematurely stop a DMA transfer. For more information, see "Stopping the DMA Transfer (Software Stop)" on page 486.

Test Interrupt

You can write to the DMA Write Interrupt Status Register (`DMA_WRITE_INT_STATUS_OFF`), or the DMA Read Interrupt Status Register (`DMA_READ_INT_STATUS_OFF`), to emulate interrupt generation, during software or hardware testing. A write to these registers triggers an interrupt, but the DMA does not set the done or abort bits in these registers. This test interrupt is unaffected by the mask bits. To clear this interrupt, you must write to either of the DMA Write Interrupt Clear or the DMA Read Interrupt Clear registers.

6.4.9 Linked List Operation

This section describes the detailed operation of the DMA in linked list (LL) mode. It also discusses how the software should produce LL elements[1], and how the DMA and software maintain synchronization. The following topics are discussed:

- "LL Operation Overview"
- "Using Interrupts for Linked List Producer-Consumer Synchronization" on page 516
- "PCS-CCS-CB-TCB Producer-Consumer Synchronization" on page 517
- "Element Recycling" on page 520
- "Link Down Recovery" on page 520

6.4.9.1 LL Operation Overview

You should first read "Linked List Mode" on page 475 for an overview of LL mode. In this section, a normal LL operation called recycling[2] is described in detail. The process is described for a write channel, but it is possible to have duplicate processes running in parallel for other write or read channels. In this process interrupts are used to trigger the producer (software) to recycle elements. Typically a Watermark interrupt (positioned near the middle of the transfer list (TL)) and an Empty interrupt (in the last *data* element) are used by setting LIE in these two elements to '1'. The steps in this process are:

1. Software creates a LL element structure called a TL in local memory consisting of N-1 *data* elements and one *link* element.
   - The *link* element s "LL Element Pointer Low/High DWORDs" in Figure 6-6) are programmed to point back to the beginning of the first data element.
   - One of the data elements (near the middle of the TL) is programmed to generate a "Watermark" interrupt.
   - The last data element is programmed to generate an "Empty" interrupt.
2. Software programs the DMA with the location of the TL. This is done by writing to the DMA Linked List Pointer Low Register (`DMA_LLP_LOW_OFF_WRCH_0`) and DMA Linked List Pointer High Register (`DMA_LLP_HIGH_OFF_WRCH_0`).

1. Also known as descriptors.
2. With recycling, the same LL structure is used over and over again. Another mode of operation would be to jump to a new LL element structure when the current one is consumed.

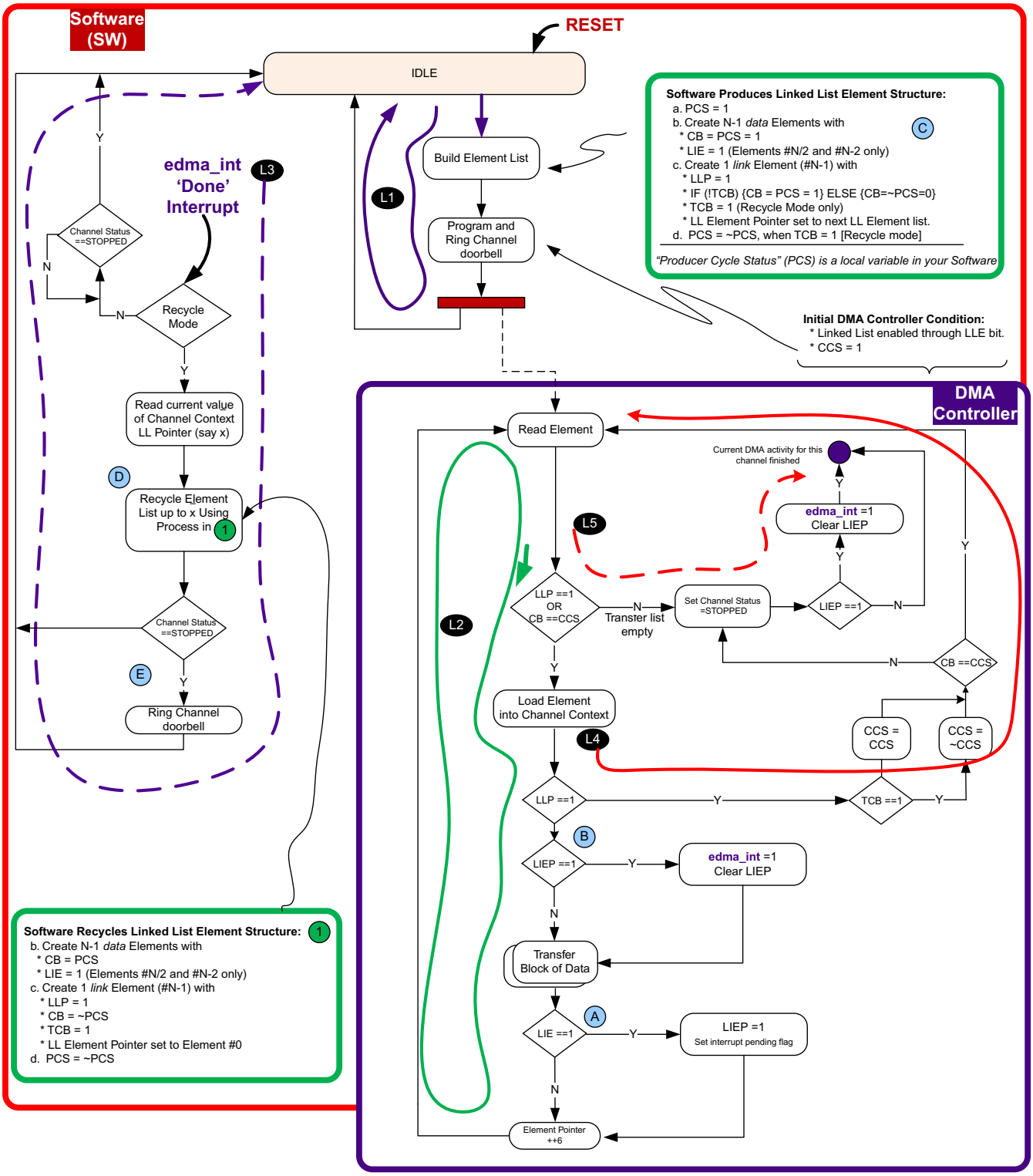
3. Software starts the DMA process by ringing the write channel Doorbell. This is done by writing to the DMA Write Doorbell Register (`DMA_WRITE_DOORBELL_OFF`).

   The **solid purple loop L1** in Figure 6-23 corresponds to steps 1-3.

4. The DMA reads (**consumes**) each *data* element from local memory, and if the CB and CCS bits match, loads the information (SAR, DAR, size, and so on) from that data element into the channel context registers. These context registers determine the operation of the channel that the DMA is currently servicing. The DMA executes the check for the CB/CCS match after it has evaluated TCB and toggled CCS. The DMA always loads link elements into the channel context.

Note If the DMA stops on a link element, then your software must set the DMA Linked List Pointer registers, and restart the DMA process by ringing the doorbell.

Figure 6-23 Linked List Flow for Producer and Consumer



5. The DMA then proceeds to transfer the block of data (as defined by the element), and when it is finished, reads the next element from local memory.

   The **solid green loop L2** in Figure 6-23 corresponds to steps 4-5.

6. The last element in the list (called a *link* element, and indicated by having its LLP field set to 1) is not used to transfer a block of data.

   - It causes the DMA to effectively repeat the task in step 4.
   - Its TCB is typically set to 1. This causes the DMA to toggle its CCS bit, and the software to toggle its PCS bit. The software and DMA use the "PCS-CCS-CB-TCB" producer-consumer synchronization mechanism to ensure that:
     - Software does not recycle elements that have not yet being consumed by the DMA
     - DMA correctly recognizes and consumes recycled elements.

   The **solid red loop L4** in Figure 6-23 corresponds to step 6.

7. Upon reception of the interrupt (mentioned in step 1), the software starts to recycle the TL.

   - Software reprograms each data element with new DMA transfer information.
   - The software and DMA use the "PCS-CCS-CB-TCB" producer-consumer synchronization mechanism described later in this section.

   The **dashed purple loop L3** in Figure 6-23 corresponds to step 7.

8. As some point, the software wants to terminate the complete DMA process, by not recycling any more elements. The DMA recognizes this condition when CB !=CCS, and sets the channel status to stopped

   The **dashed red loop L5** in Figure 6-23 corresponds to step 8.

6.4.9.2 Using Interrupts for Linked List Producer-Consumer Synchronization

When the DMA is finished with an element, the DMA checks the LIE bit, before reading the next LL element. For more information, see circled step A in Figure 6-23. When the LIE is set, then the DMA does not assert the "done" interrupt immediately, but sets an internal interrupt pending flag (LIEP). It asserts the actual done interrupt after the next data element has been read from system memory. For more information, see circled step B in Figure 6-23. This automatic internal process of delaying the interrupt avoids a race condition between the DMA and software when the LIE bit is set in an element.

Your software should set the "Watermark" interrupt early to schedule the recycling of the consumed elements. Placing the Watermark interrupt too far down in the list, combined with a slow element recycling process in your application; ensures that the DMA returns to the start of the LL before your software has recycled it. The DMA channel STOPs, and you have to restart it by writing to its Doorbell. Typically a "Watermark interrupt" (positioned near the middle of the TL) and an Empty interrupt (in the last data element) are used by setting LIE in these two elements to 1b1. Upon reception of the interrupt, the software starts to recycle the TL.

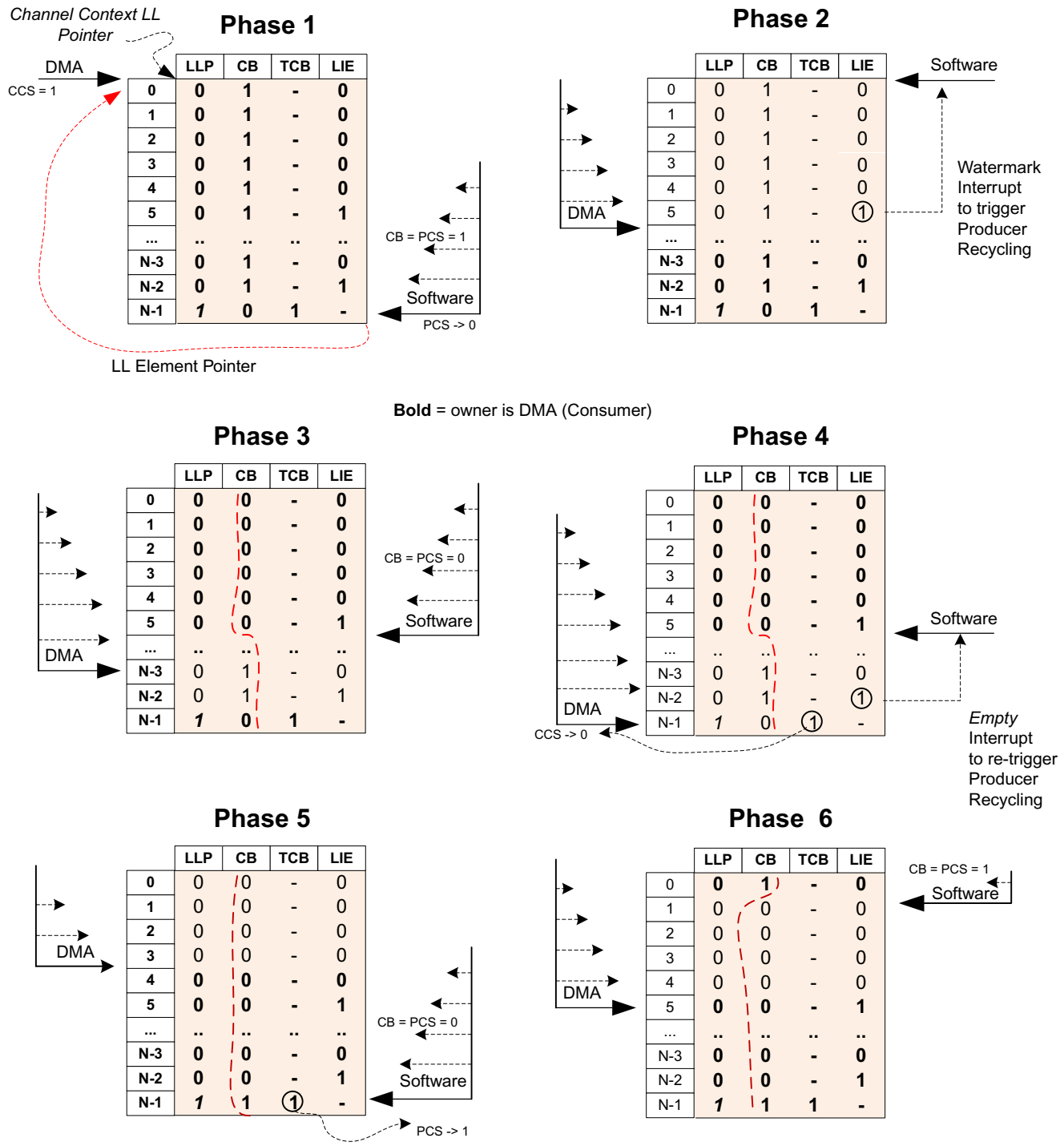
Operational interrupts are always "done" interrupts. For more information, see "Interrupts and Error Handling" on page 469.

Note The DMA processes the remote interrupt enable (RIE) bit in the LL element in the same way as the LIE. Therefore, everywhere that LIE and edma_int are mentioned in this section can be interpreted to mean LIE and edma_int, or RIE and IMWr.

6.4.9.3 PCS-CCS-CB-TCB Producer-Consumer Synchronization

The software and DMA use the "PCS-CCS-CB-TCB" producer-consumer synchronization mechanism to ensure that software does not recycle elements that have not yet being consumed by the DMA, and that the DMA correctly recognizes and consumes recycled elements. This process, which is shared between the DMA and the software, is illustrated through an example producer-consumer flow in Figure 6-24.

Figure 6-24 Example Producer (Software) - Consumer (DMA) Synchronization Flow



Before looking at the example in detail, it is useful to note that the DMA performs the following two tests as part of this process:

- Consumer-Owned Element, or Transfer List (TL) Empty Test

  The **dashed red loop** marked L5in Figure 6-23 corresponds to this test. The Cycle Bit (CB) of DMA Channel Control 1 Register (`DMA_CH_CONTROL1_OFF_WRCH_0`) (which is loaded from the LL element)

is tested against the DMA Consumer Cycle State (CCS) bit of the same register. When CB =CCS, the element is owned by the consumer (DMA) and the data transfer is executed. When CB !=CCS, the TL is empty and the DMA sets channel Status (CS) in DMA Channel Control 1 Register (`DMA_CH_CONTROL1_OFF_WRCH_0`) to "Stopped". When LLP bit of DMA channel Control 1 Register is set, the end of list decision is done in the next decision branch after the TCB bit is loaded from the LL element.

- Toggle Cycle Bit (TCB) Test

  The **solid red loop** marked L4 in Figure 6-23 corresponds to this test. When the Toggle Cycle Bit (TCB) of the DMA Channel Control 1 Register (`DMA_CH_CONTROL1_OFF_WRCH_0`) (which is loaded from the LL element) is set to 1, the DMA toggles the CCS bit of the DMA Channel Control 1 Register (`DMA_CH_CONTROL1_OFF_WRCH_0`) to be ~CB. The producer (software) also performs a similar action on its local variable "Producer Cycle Status" (PCS), as it sets PCS =~PCS. The TCB field is only valid in the last LL element (the *link* element). Toggling the PCS and CCS flags in this manner synchronizes the consumer to the producer.

The example in Figure 6-24 has the following six phases:

1. The software builds the transfer list for the write (or read) channel, according to the steps outlined in the circled step C in Figure 6-23. For each element, CB =PCS =CCS =1. The exception to this is the link element where CB is set to the next iteration value because the link element is always loaded into the channel context.

   a. The *link* element points to the start of the TL and its TCB bit is 1. Software toggles its PCS variable.

   b. Software writes the location of the TL into the channel context LL Pointer register.

   c. Software *rings* the write channel Doorbell.

2. The DMA starts processing elements. After processing element 5 (and fetching the next element), the DMA asserts a *Watermark* "done" interrupt to signal the software to start recycling elements.

3. The **software** is recycling elements. It reads the channel context DMA Linked List Pointer Low Register (`DMA_LLP_LOW_OFF_WRCH_0`) and DMA Linked List Pointer High Register (`DMA_LLP_HIGH_OFF_WRCH_0`), and recycles elements up to this location[1] in the TL. It sets CB for each element to be PCS. The Dashed Purple Loop Marked L3 in Figure 6-23 corresponds to this activity.

   The **DMA** continues processing (consuming) elements as before.

4. The **DMA** then asserts an *Empty* "done" interrupt (from the LIE bit in the last data element at the end of TL that is set in the previous phase) to indicate to the software, to start recycling elements again. The DMA reaches the end of the TL, and as the TCB is set to 1, it toggles its CCS bit to 0 (~CB). For more information, see circled step D in Figure 6-23.

5. The **software** reads the channel context DMA Linked List Pointer Low Register (`DMA_LLP_LOW_OFF_WRCH_0`) and DMA Linked List Pointer High Register (`DMA_LLP_HIGH_OFF_WRCH_0`), and recycles elements up to this location in the TL. It sets CB for each element to be PCS. As it recycles the link element, it detects that TCB is set to 1. It toggles its internal variable PCS to 0, and the initial TL has been fully recycled.

   The **DMA** starts processing the recycled elements.

6. The **DMA** continues processing (consuming) elements as before. Software did not have to ring the Doorbell, as the first element is already recycled before the DMA started processing it.

   The **software** might continue recycling from the top of the list, if the DMA had already returned to the top of the list when the software read the channel context DMA Linked List Pointer Low Register (`DMA_LLP_LOW_OFF_WRCH_0`) and DMA Linked List Pointer High Register (`DMA_LL-`

1. If the consumer has moved on to another element, this is not a problem. These newly consumed elements are recycled in the next recycling phase.

P_HIGH_OFF_WRCH_0), in step 5. If not, it starts recycling again when the DMA processes element 4 and the DMA asserts a *Watermark* "done" interrupt.

6.4.9.4 Element Recycling

When your software is recycling elements, it should program the first DWORD (control bits) in Figure 6-6 only after the other DWORDs have been programmed. In addition, it should set the CB bit (to ~PCS) within this DWORD only after the other bits in this DWORD have been set. This programming sequence avoids the DMA incorrectly fetching an element that software is currently recycling.

When the software is finished recycling it must check the channel context status (CS). When CS is "Stopped", the software must ring the channel Doorbell again. For more information, see circled step E in Figure 6-23. If your element recycling process is fast enough, the DMA seamlessly moves from processing the last old element to the first new element without the need for the software to ring the channel Doorbell again. To eliminate the possibility of a race condition between these two actions (read and write), you should first confirm the presence of the "done" interrupt bit in the DMA Write Interrupt Status Register (DMA_WRITE_INT_STATUS_OFF) or DMA Read Interrupt Status Register (DMA_READ_INT_ STATUS_OFF) and also check the DMA Linked List Pointer Low Register (DMA_LLP_LOW_OFF_WRCH_0) to confirm that it is pointing to the correct location.

6.4.9.5 Link Down Recovery

During normal operation when the link goes down, the controller generates a "link reset" request. A high-to-low transition on link_req_rst_not indicates that the PCIe controller is requesting external logic to reset the PCIe controller because the PHY link is down. For more information, see "Hot Reset" on page 97. To determine the progress of the linked list operation when the link went down, you can read the contents of the channel context DMA Linked List Pointer Low Register (DMA_LLP_LOW_OFF_WRCH_0) and DMA Linked List Pointer High Register (DMA_LLP_HIGH_OFF_WRCH_0).

6.4.10 Multichannel Arbitration

You can configure the DMA to have one to eight read channels and one to eight write channels using the CC_NUM_DMA_WR_CHAN and CC_NUM_DMA_RD_CHAN configuration parameters. A DMA write and read channel operate independently in full duplex mode. This maximizes the performance of the DMA read and write data transfers over the PCIe link. When you configure the DMA with multiple read channels, then it uses a weighted round robin (WRR) arbitration scheme to select the next read channel to be serviced. The same applies when you have multiple write channels. The DMA handles the multiple channels in a time-multiplexed way. From a software point of view every channel operates independently, so each one has its own instance of linked list and context. The bandwidth for all channels is shared in time and is controlled by a weighted round-robin arbiter.

You can set the arbitration weight for each channel using the DMA_WRITE_CHANNEL_ARB_WEIGHT_LOW_OFF, DMA_WRITE_CHANNEL_ARB_WEIGHT_HIGH_OFF), DMA_READ_CHANNEL_ARB_WEIGHT_LOW_REG, and DMA_READ_CHANNEL_ARB_WEIGHT_HIGH_OFF registers. Each register has a 5-bit field (representing a weight from 1 to 32) for up to four channels. You must set the weights before "ringing" the doorbell.

The channel weight specifies the number of TLP requests that the DMA can issue for that channel before it must return to the arbitration routine. When the channel weight count is reached, the WWR arbiter selects the next channel to be processed. For example, consider two DMA write channels (write channel #0 and

write channel #1) where you want to assign the bandwidth for write channel #1 to be two times that of write channel #3. Therefore set:

- write channel #0 weight to 32
- write channel #1 weight to 16

Depending on the transfer size and the max_payload_size, the DMA issues 32 MRd requests for write channel #0, followed by 16 MRd requests for write channel #1, followed by 32 MRd requests for write channel #0 and so on, until the current transfer (or element when using LL mode) is finished. Therefore in this example, the weights reserve 33% bandwidth for write channel #1 and 67% bandwidth for write channel #0.

Note

- CX_REMOTE_MAX_TAG+1 constraints the maximum number of outstanding DMA write engine MRd requests to the application, so the sum of weights should not exceed this value.
- For the DMA read engine, the sum of all weights should not exceed CC_NUM_DMA_RD_TAG.
- If these limits are exceeded, then the respective DMA engine requester thread is back-pressured, defeating purpose of the round-robin bandwidth sharing mechanism.
- When the ratio (transfer-size / TLP-size) is smaller than the round-robin arbiter setting (say, `DMA_WRITE_CHANNEL_ARB_WEIGHT_LOW_OFF`), the DMA issues less requests than the value in `DMA_WRITE_CHANNEL_ARB_WEIGHT_LOW_OFF`. The TLP size is determined by the DMA using each function's Max_Payload_Size and Max_Read_Request_Size rules. This means that the arbitration value in `DMA_WRITE_CHANNEL_ARB_WEIGHT_LOW_OFF` determines the number of TLPs send and not the absolute data bandwidth.

Request, Response, and Descriptor-Fetch Threading in LL Mode For a Read Channel

For each read channel there are two threads (1) the request thread and (2) the response thread which are running in opposite directions. The DMA processes these threads independently. The request thread does not wait for the individual responses to arrive before issuing the next request. The request thread for the current channel i continues to issue requests until one of the following is true:

- The DMA exhausts its pool of available tags and the logic in the "Read Buffer" on page 506 back-pressures the DMA read engine controller.
- The DMA completes its current DMA transfer block for the current channel. It has requested all of the data specified by the currently-loaded LL element[1].
- The current channel has issued Wi MRd requests, where Wi is the round robin weight associated with that channel.

A channel arbitration event now occurs and the DMA loads the context for the winning channel *j*. When the DMA is loading the new context, the response path for the previous channel *i* is still utilizing the wire because the response thread lags the request thread. As soon as the DMA has loaded the context (and LL element if in LL mode) for the current channel *j*, it starts to issue requests. Therefore, in multichannel operation, when the block for channel *i* completes and a new LL element is been fetched, the DMA can be issuing requests belonging to other DMA channels. The currently inactive requester thread for channel *i* needs to synchronize with the response thread (that is, wait for all completions) before fetching its next LL element from local memory.

1. Also known as a descriptor.

7

HDMA Controller

This section describes the optional embedded multichannel Hyper DMA (HDMA) controller. You enable HDMA controller by setting the `CX_DMA_PF_ENABLE` configuration parameter in addition to `CC_DMA_ENABLE` configuration parameter. DMA VSEC registers (`PF0_VSECDMA_CAP`) help in identifying the DMA implementation, register location, and layout. The following topics are discussed in this section:

- "HDMA Overview" on page 524
- "HDMA Architecture" on page 529
- "Using the HDMA" on page 539
- "Advanced HDMA Information and Operation" on page 548

7.1 HDMA Overview

The RC system CPU, or the EP application CPU, can offload the transferring of large blocks of data to the embedded HDMA controller, leaving the CPU free to perform other tasks. You can configure the HDMA to have one to `CC_NUM_DMA_RD_CHAN` read channels and one to `CC_NUM_DMA_WR_CHAN` write channels. It can simultaneously perform the following types of memory transactions (as shown in Figure 7-1):

- **HDMA write**

  Transfer (copy) of a block of data from local (application) memory to remote (link partner) memory.
- **HDMA read**

  Transfer (copy) of a block of data from remote (link partner) memory to local (application) memory.

Therefore, the HDMA supports full duplex operation, processing read and write transfers at the same time, and in parallel with normal (non-HDMA) traffic. Upon completion of a HDMA transfer or an error, the HDMA optionally interrupts the local CPU or sends an interrupt MWr (IMWr) to the remote CPU. The HDMA is highly configurable, and you can program it using the local DBI or over the PCIe wire[1].

The HDMA can operate in non-linked list mode or linked list mode. In linked list mode, the HDMA fetches the transfer control information (called channel context) for each transfer (block), from a list of HDMA elements that you have constructed in local memory. For more information, see "Linked List Mode" on page 536. In non-linked list mode, your application provides the transfer control information (called channel context) for each transfer, at run time.

Note In this release, HDMA is available in the AXI bridge configuration only.

7.1.1 Feature List

Table 7-1 compares the features of legacy DMA with HDMA.

Table 7-1 Legacy DMA and HDMA Features

Module Feature	Legacy DMA	HDMA
Number of Write Channels	1-8	1-64
Number of Read Channels	1-8	1-64
AXI Support	Yes	Yes
Native Controller Support (without AXI)	Yes	No
Linked List Mode	Yes	Yes
Linked List Descriptor Alignment	DWORD	DWORD
Scatter Gather Support (by using Linked Lists)	Yes	Yes
Full Duplex Read/Write Channel Operation	Yes	Yes

1. RP can program the EP HDMA using CfgWr or MWr (when ENABLE_MEM_MAP_PL_REG =1), but the EP cannot program the RP HDMA over the wire.

Module Feature	Legacy DMA	HDMA
Separate Tag Pool for DMA Traffic	Yes	Yes
SAR/DAR Address Alignment	BYTE	BYTE
Transfer Size	1 byte - 4 GB	1 byte - 4 GB
AXI Address Bus Width	32, 64	32, 64
AXI Databus Width	32, 64, 128, 256, 512	32, 64, 128, 256, 512
Independent Internal, Master and Subordinate Databus Widths	Yes	Yes
Independent Controller and AXI Clocks	Yes	Yes
Latency Tolerance Reporting (LTR) Support	Yes	Yes
OBFF Support	Yes	Yes
Address Translation Services (ATS) Support	Yes	Yes
AtomicOps Support	No	Yes
TLP Processing Hints (TPH) Support	Yes	Yes
TLP Prefix Support (Including PASID)	No	No
Programmable Interrupt Generation	Yes	Yes
Reordering of DMA Read Transfer Completions	Yes	Yes
Reordering of DMA AXI R Channel Transactions (AXI Configurations)	Yes	Yes
ID Based Ordering (IDO) Signaling Support[a]	Yes	Yes
Datapath and RAM Protection For Limitations, see “RAS Data Protection (RAS DP)” on page 164.	Yes	Yes
RAM Read Enable for Low-Power Optimization	Yes	Yes
Automatic Flushing/Reset of Bridge (AXI Configurations) When Link Goes Down	Yes	Yes
Automatic Flushing of Subordinate (AXI Configurations) When Link is Not Up	Yes	Yes
Hardware Flow Control	Yes	No
Linked List Descriptor Early Fetch	Yes	NA
Channel Assignment to a VF	Yes	Yes
Round Robin Bandwidth Sharing Between Channels for QoS	No	Yes
Configurable Weighted Round Robin Bandwidth Sharing Between Channels for QoS	Yes	No
Doorbell at Any Time	Yes	Yes

Module Feature	Legacy DMA	HDMA
Synthesis QoR for 1 GHz or Above	No	Yes
Multiple Descriptor Fetch in One Clock Cycle[b]	No	Yes
Programmable Descriptor Prefetch Depth	No	Yes
Full Bandwidth Utilization for Small Transfer Blocks (1TLP per Descriptor)	No	Yes
HDMA Backward Compatibility with Legacy DMA	NA	Yes[c]
Registers DBI Channel Separation for Better SR-IOV Support	No	Yes
Register Wire Channel Separation Per-channel BAR/PF/VF	No	No
Descriptor Size Alignment with DATA Width / Cache Lines	No	No
Non-blocking Multi VC Support	No	No
Per-descriptor PCIe to Memory or Memory to PCIe Support	No	No
RAS-DES Support	No	No
PCIe 10-bit Tag Support	No	No
Linked List Starvation Interrupt	No	No
AXI Burst Transaction Alignment	No	No
PASID Prefixing Support	No	No
Stop Doorbell Support for SR-IOV	No	No
VSEC Feature Set Discovery	Yes	Yes

a. DMA identifies and decodes IDO bit from the TLP bit stream and passes on the information to the Application through native client interfaces or AXI Bridge. But, IDO bit of the TLP Header cannot be explicitly set by the DMA.
b. Possible only if datapath allows it.
c. For more information on legacy DMA and HDMA software compatibility, see the Legacy DMA and HDMA Software Compatibility section in "HDMA Software Register Map" on page 543.

7.1.2 Limitations

Table 7-2 compares the limitations of legacy DMA with HDMA.

Table 7-2 Legacy DMA and HDMA Limitations

Limitation	Legacy DMA	HDMA
Memory DMA Traffic Only	I/O and Type 0 or Type 1 configuration DMA transfers are not supported	

Limitation	Legacy DMA	HDMA
Performance limitations in a few configurations[a]	In the following configurations DMA may not achieve full link utilization. ■ 512-bit controller data width and 256/128 bytes MTU ((CX_NW =16) && (CX_MAX_MTU <=256)) ■ 256-bit controller data width and 128 bytes MTU ((CX_NW = 8) && (CX_MAX_MTU <=128)) ■ AXI manager configurations with data-path wider than 64 bits do not naturally align DMA descriptor reads. These configurations must have MSTR_FIXED_SIZE_ENABLE to avoid byte accesses that degrade performance. ((AMBA_INTERFACE >1) && (MSTR_FIXED_SIZE_ENABLE ==0) && (MASTER_BUS_DATA_WIDTH > 64)) ■ Single channel DMA ((CC_NUM_DMA_WR_CHAN =1) || (CC_NUM_DMA_RD_CHAN =1)) ■ Insufficient number of outstanding NP requests for the system latency (Insufficient CC_NUM_DMA_RD_TAG and/or CX_REMOTE_MAX_TAG)	In the following configurations DMA may not achieve full link utilization. ■ AXI manager configurations with data-path wider than 64 bits do not naturally align DMA descriptor reads. These configurations must have MSTR_FIXED_SIZE_ENABLE to avoid byte accesses that degrade performance. ((AMBA_INTERFACE >1) && (MSTR_FIXED_SIZE_ENABLE==0) && (MASTER_BUS_DATA_WIDTH >64)) ■ Insufficient number of outstanding NP requests for the system latency (Insufficient CC_NUM_DMA_RD_TAG and/or CX_REMOTE_MAX_TAG)
Idle interval in the transfer in a few use cases[a]	DMA can load only one descriptor at a time. So, the previous descriptor transactions must end before loading the next one, that is, ■ All completions must be received, transformed into MWR and the write transaction must be acknowledged ■ If enabled, all interrupts must be generated In the following use cases, waiting for all the transactions to end before loading the next descriptor creates an idle interval in the transfer. ■ Low channel count ■ Small descriptor transfer size DMA early fetch feature, (CC_DMA_EFETCH_EN) supported only in AXI bridge configurations, minimizes the load time of the next descriptor, but does not eliminate the idle interval completely.	NA

Limitation	Legacy DMA	HDMA
No support for Inbound Posted Tracker Enable feature or AXI High Priority Master	HDMA does not support the following features ■ Inbound Posted Tracker Enable (CC_IB_WREQ_PTRK_EN =1) ■ AXI High Priority Master (CC_HP_-MASTER =1)	DMA also does not support the following features ■ Inbound Posted Tracker Enable (CC_IB_WREQ_PTRK_EN =1) ■ AXI High Priority Master (CC_HP_-MASTER =1)

a. You must execute the DMA performance test cases provided with the controller to assess the DMA performance in configurations involving the listed use cases.

7.2 HDMA Architecture

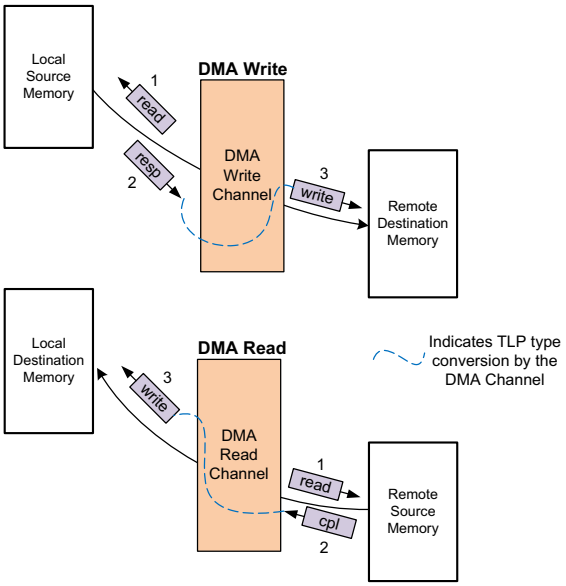
This section describes the architecture of the HDMA. The topics for this section are:

- "Architecture Overview"
- "Interrupts and Error Handling" on page 533
- "Linked List Mode" on page 536

7.2.1 Architecture Overview

HDMA write and read channels operate independently to maximize the performance of the HDMA read and write data transfer over the PCIe link. When you configure the HDMA with multiple read channels, then it uses a round robin (RR) arbitration scheme to select the next read channel to be serviced. The same applies when you have multiple write channels.

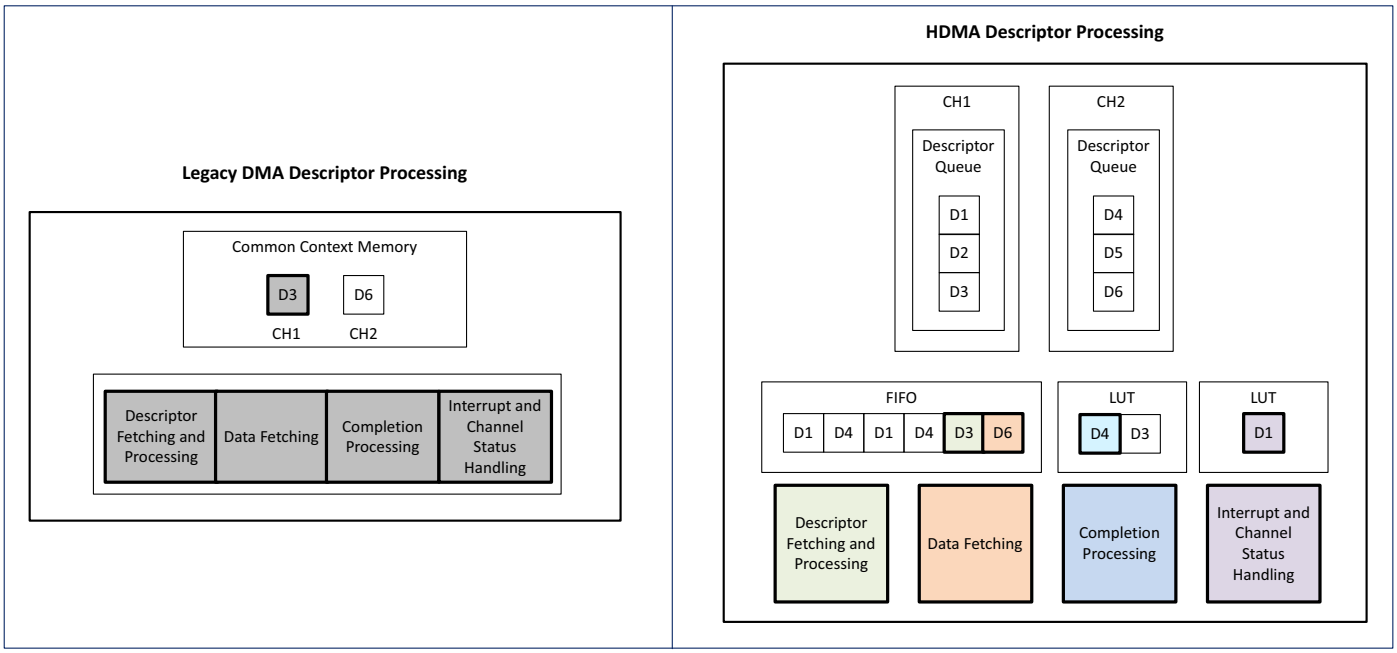
Figure 7-1 System Level View of HDMA



In linked list mode, unlike legacy DMA, HDMA decouples descriptor fetching and processing, data fetching, data completion processing, and channel status and interrupt handling operations. An independent thread is associated with each of these HDMA operations. Instead of a central context memory like legacy DMA, HDMA maintains a per-channel descriptor queue. After data fetch there is no global channel context control, each HDMA operation carries a snapshot of the channel context (channel control information, DAR/SAR, transfer size) that tracks the progression of a transaction. As a result of which, HDMA is able to process multiple descriptors at a time.

Figure 7-2 shows a comparison of legacy DMA descriptor processing (where next descriptor is fetched only after the completion of the current descriptor transactions) and HDMA descriptor processing (where four descriptors are processed by HDMA simultaneously).

Figure 7-2 Comparison of Legacy DMA and HDMA Read/Write Engine Descriptor Processing



7.2.1.1 Descriptor Prefetch

HDMA features an overlay dual port RAM in the write/read engine, where the linked list descriptors are queued. This RAM is logically split into per-channel descriptor queues (FIFO). After doorbell, HDMA reads the main memory (software visible) and copies descriptors into these per-channel descriptor queues, ahead of the data transfer. The depth of the overlay RAM is the product of the number of configured read/write channels and per-channel descriptor queue depth, that is,

```
CC_IF_[WR|RD]CTX_LLQ_OVERLAY_RAM_DP = CC_NUM_DMA_[WR|RD]_CHAN) *
CX_DMA_[WR|RD]_LLQ
```

The per-channel descriptor queue depth (`CX_DMA_[WR|RD]_LLQ`) for write/read engine must be calculated using simulation. It is a function of the descriptor read transaction latency of the write/read engine and your application's latency.

The `PF_DEPTH` of all channels should be set together at the beginning of the programming stage to avoid unexpected behaviors. Changing `PF_DEPTH` after programming other registers creates unexpected behaviors and affects all channels.

Prefetch Features

- For each channel in the doorbell state, HDMA reads descriptors when the channel's descriptor queue reports at least `ELEMENT_PREFETCH` descriptors.Thisvalue is also the maximum number of outstanding descriptors at a given time. `ELEMENT_PREFETCH` value can be programmed at run time using `ELEMENT_PREFETCH` field of the `HDMA_ELEM_PF_OFF_[WR|RD]CH_i` register. It defaults to zero (that is, prefetch one descriptor).

- If your application uses less than CC_NUM_DMA_[WR|RD]_CHAN channels, you can redistribute the unused per-channel overlay RAM among the channels that are in use. This re-configuration is possible only when HDMA is idle, using PF_DEPTH field of the HDMA_QOS_OFF_[WR|RD]CH_i register.

  For example, if

  ```
  CC_NUM_DMA_WR_CHAN =8
  CX_DMA_WR_LLQ =8
  CC_IF_WRCTX_LLQ_OVERLAY_RAM_DP =64 (default)
  ```

  If your application uses one channel, set

  ```
  HDMA_QOS_OFF_[WR|RD]CH_i.PF_DEPTH =63
  ```

  so that the total available per-channel overlay RAM space of 64 can be used by one channel. However, your application must doorbell only channel 0, doorbelling any other channel yields undefined results. For more information see, *"Prefetch Limitations"*.

- When the descriptor read length is greater than the maximum transfer unit size (CX_MAX_MTU), HDMA decomposes the descriptor read request. HDMA ignores the physical function MTU value and requests less than or equal to CX_MAX_MTU bytes in a single descriptor read.

  For example, if

  ```
  MTU =128 bytes (32 DW)
  HDMA_ELEM_PF_OFF_WRCH_i.ELEMENT_PREFETCH =6 (that is, 7 descriptors)
  Descriptor Read Length =7 * 6 DW =42 DW
  ```

  As descriptor read length is greater than the MTU, HDMA issues two descriptor read requests

  ```
  Descriptor Read 1: 5 Descriptors (30 DW)
  Descriptor Read 2: 2 Descriptors (12 DW)
  ```

  HDMA waits till data read for these 7 descriptors is complete, before fetching the next 7 descriptors.

- In linked list mode, HDMA loads the ELEMENT_PREFETCH value from the link element of the transfer list. The ELEMENT_PREFETCH value must match the length of a contiguous set of descriptors in the main memory, that is, from HDMA_LLP_[LOW|HIGH]_[WR|RD]CH_ito the next transfer list pointer. This ensures HDMA does not read descriptors out of bounds.

Prefetch Limitations

- The descriptor queue size (CX_DMA_[WR|RD]_LLQ) is the same for all channels.
- If your application uses less than CC_NUM_DMA_[WR|RD]_CHAN channels, you can redistribute the unused per-channel overlay RAM among the channels that are in use. However, only the channels that are in use can be doorbelled That is, if N channels are used, only 0 to N-1 channels can be doorbelled.

  For example, if

  ```
  CC_NUM_DMA_WR_CHAN =8
  CX_DMA_WR_LLQ =8
  CC_IF_WRCTX_LLQ_OVERLAY_RAM_DP =64 (default)
  ```

  If your application uses two channels, that is,

  ```
  HDMA_QOS_OFF_[WR|RD]CH_i.PF_DEPTH =31
  ```

  so that the total available per-channel overlay RAM space of 64 can be used by two channels.

  In this scenario you can use only two channels, that is, channel 0 and channel 1. Doorbelling any other channel yields undefined results.

- For each channel, you must program the link pointer to respect the `ELEMENT_PREFETCH` block boundaries. You must not program the link list pointer (`HDMA_LLP_[LOW|HIGH]_[WR|RD]CH_i`) to point to a data element in the middle of a contiguous `ELEMENT_PREFETCH` block.

  The HDMA increments the link pointer by 6 DWORDs times the `ELEMENT_PREFETCH` value for data elements in a transfer list. If you program the link list pointer to point to a data element in the middle of a contiguous `ELEMENT_PREFETCH` block, then HDMA reads descriptors out of bounds.
- If HDMA stops in the middle of an `ELEMENT_PREFETCH` block due to an error or end of list detection, you must reset `ELEMENT_PREFETCH` value to prevent HDMA from reading descriptors out of bounds.

7.2.1.2 HDMA Write Transfer

The HDMA injects multiple MRd requests of size less than or equal to Max_Payload_Size into the inbound request path, directed toward the local application. The HDMA converts the read responses into MWr TLPs, aligns the request to the application data bus width for size greater than Max Tranfer Unit (MTU), and transmits to the remote link partner. When the HDMA data transfer is complete, the CPU is notified. For more information, see "Interrupts and Error Handling" on page 533. For a write transfer, the SAR is the address of the local memory, and the DAR is the address of the remote memory.

7.2.1.3 HDMA Read Transfer

The HDMA injects multiple MRd requests of size less than or equal to the minimum of {Max_Read_Request_Size, Max_Payload_Size_Supported} into the outbound request path, directed towards the remote link partner. The HDMA converts the read responses into write requests, requests to the application data bus width when size is bigger than Max Tranfer Unit (MTU) for alignment on MWr (the first request alignment guarantees the subsequent alignment), and transmits to the local application. When the HDMA data transfer is complete, the CPU is notified. For more information, see "Interrupts and Error Handling" on page 533. For a read transfer, the SAR is the address of the remote memory, and the DAR is the address of the local memory.

7.2.1.4 RAM Requirements

For more information on HDMA RAM Requirements, see "RAM Requirements" on page 64.

7.2.1.5 Registers and Context Memory

You program each HDMA transfer (channel) using the registers described in "HDMA Software Register Map" on page 543. Each channel has its own exclusive register address segment. Exclusive, per-channel register address segment allows better support of virtualized systems. As the transfer progresses, the HDMA updates these registers. You program the channel context information and other HDMA control registers directly with the local CPU through the DBI or remotely over the PCIe link.

Per-channel Register Address Segment Separation

The per-channel register address segments are not contiguous. By default, each channel's register address segment is separated by 256. Consecutive read/write channel's register address segment is separated by 512. That is, if write channel 0's address space starts at 0x0, then read channel 0's address space starts at 0x100, write channel 1's address space starts at 0x200, read channel 1's address space starts at 0x300, and so

on. You can configure the per-channel register address segment separation using the internal parameter `CX_DMAREG_CHADDR_SPACE` or `CHANNEL_SEPARATION` field of `VSECDMA_DEVICE_INFORMATION_OFF` register.

The per-channel register address segment separation feature has the following limitations:

- This feature is not supported when HDMA operates in compatibility mode (`CC_LEGACY_DMA_MAP =1`).
- The number of channels configured, and the per-channel register address segment separation defined, must satisfy the following criteria:

  $\log_2$`(max (Number of Read Channels, Number of Write Channels)) + the number of CHANNEL_SEPARATION bits used < 8`

Channel Context Register Update

HDMA processes multiple descriptors simultaneously. So, unlike legacy DMA, HDMA updates the channel context registers that provide the transfer status (`HDMA_LLP_[LOW|HIGH]_OFF_[WR|RD]CH_i`, `HDMA_XFERSIZE_OFF_[WR|RD]CH_i`, and `HDMA_[SAR|DAR]_[LOW|HIGH]_OFF_[WR|RD]CH_i`) only when certain events occur. Table 7-3 describes the scenarios when these registers provide valid values that you can use.

Table 7-3 Channel Context Register Considerations

Register	Channel Status	Scenarios when Register Values are Valid
HDMA_LLP_[LOW|HIGH]_OFF_[WR|RD]CH_i	RUNNING	HDMA updates this register only when a watermark interrupt event occurs. In order to obtain the current transfer location you can use watermark interrupts. You can use this information to trigger descriptor recycling up to and including the Load Link Pointer (LLP) value.
	ABORTED	HDMA updates these registers to point to the next element in the transfer list, after the event.
	STOPPED	
HDMA_XFERSIZE_OFF_[WR|RD]CH_i HDMA_[SAR|DAR]_[LOW|HIGH]_OFF_[WR|RD]CH_i	RUNNING	These registers must be ignored.
	ABORTED	These registers hold valid data except in the scenario when the descriptor fails to load because of a LL error (that is, HDMA_INT_STATUS_OFF_[WR|RD]CH_i is LL_CPL_UR, LL_CPL_CA, or LL_CPL_EP).
	STOPPED	These registers hold valid data.

7.2.2 Interrupts and Error Handling

The registers referenced in Figure 7-3 that are related to error and interrupt handling are described in "HDMA Software Register Map" on page 543.

The HDMA generates the following interrupts per channel:

- **Done**: The HDMA successfully completes the transfer.

- **Watermark**: Interrupt generated at the end of each watermark event (end of linked list element). Generated in LL mode only.
- **Abort**: The HDMA fails to complete the transfer, or an error occurs during the transfer.

The interrupts are signaled to the software on your CPU, using one or both of the following mechanisms:

- Locally through the `edma_int[CC_NUM_DMA_RD_CHAN+CC_NUM_DMA_WR_CHAN-1:0]` bus.
- Remotely using a posted memory write (IMWr), which can be interpreted as an MSI or MSI-X when directed toward the RP.
  - There are three programmable IMWr addresses per channel, one each for the done, watermark, and abort interrupts. For MSI, you must program all IMWr address registers with the same MSI address, as PCIe only supports a single MSI address per function.
  - A single IMWr data register is used for all the interrupts, so you must read `HDMA_INT_STATUS_OFF_[WR|RD]CH_i` to identify the interrupt type.

Note

- The IMWr address and data do not come from the MSI PCIe capability registers but from the "HDMA Software Register Map" on page 543.
- For an Upstream Port, your software must program these registers with the same information that it is using to program the MSI/MSI-X capability for the function that is associated by your software with that channel in a multifunction setup.
- You only need to setup these registers at initialization, and not for every HDMA transfer.

- The relaxed ordering (RO) bit is automatically set to zero in the TLP header of all IMWr's so that subsequent MWr TLPs do not arrive at the remote device before the IMWr.

The interrupt handling mechanism is different for linked list (LL) mode (than non LL mode), and there are also some differences between the read and write channels.

7.2.2.1 Linked List Mode Interrupt Handling (Channel Stopped Interrupt: CSIRQ)

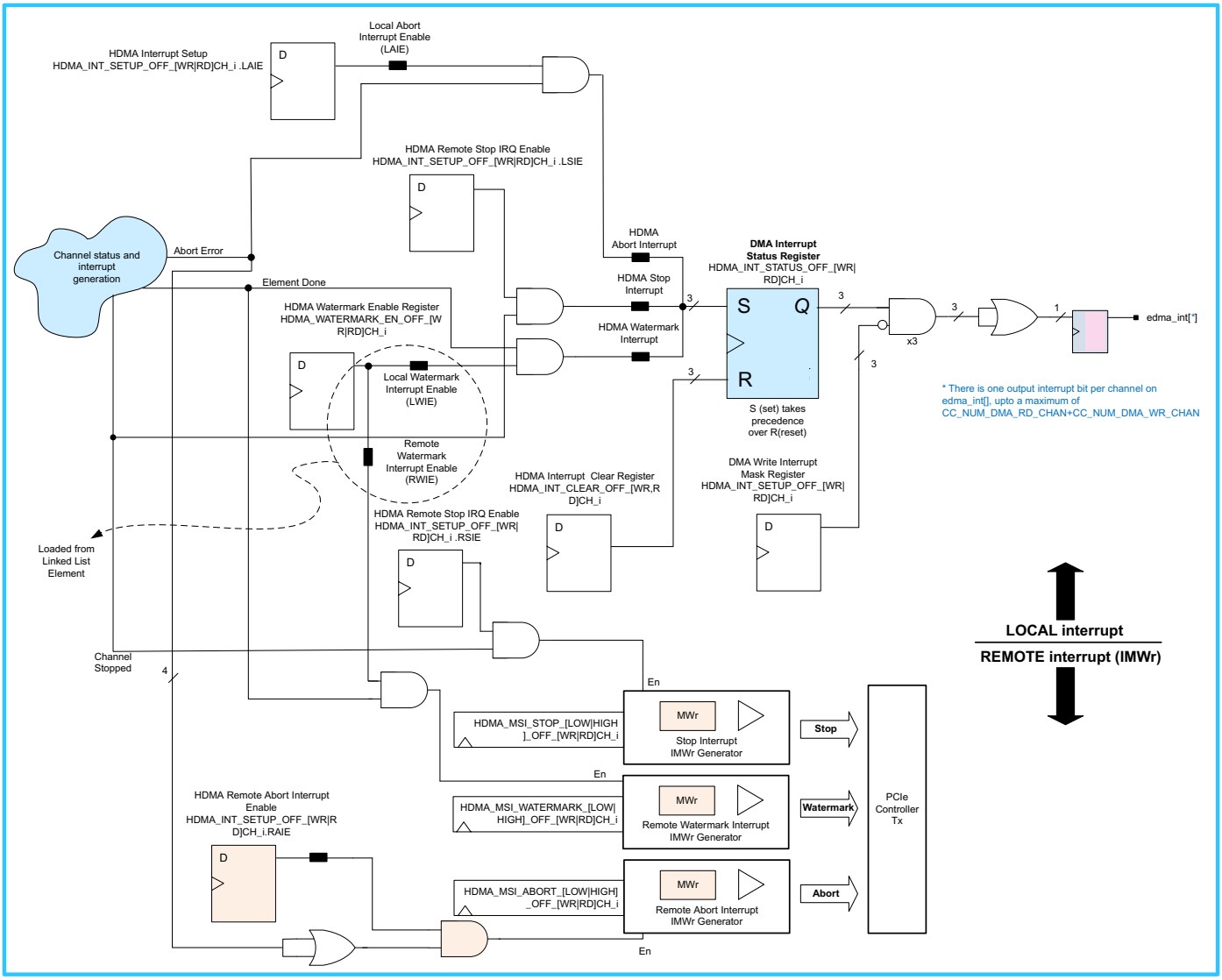
When Local/Remote stop interrupt is enabled (`HDMA_INT_SETUP_OFF_[WR|RD]CH_i.[LSIE|RSIE] =1`) for a particular channel, and any of these conditions are met:

- End of list is detected, that is, descriptor is loaded and CB !=CCS

  (Note: This can be the first descriptor after doorbell, so the end of list interrupt fires with no actual transfer. In this scenario HDMA does not load any data into the channel context registers.)
- Your application stops the HDMA transfer

HDMA generates a local and/or a remote stop interrupt and sets `HDMA_INT_STATUS_OFF_[WR|RD]CH_i.STOP =1`.

Figure 7-3 Write/Read Interrupt Generation - Linked List Mode (CSIRQ)



Note

- When HDMA processes a large amount of small transfer size descriptors, the register / interrupt pin interface might not be fast enough to sense all individual local interrupts. To overcome this limitation, you must track the linked list pointer registers: HDMA_LLP_[LOW|HIGH]_OFF_[WR|RD]CH_i
- HDMA write engine MSIs are transmitted on XALI0, and read engine MSIs are transmitted on XALI1. You can specify the type of TLP you are supplying the XALI0/1 interface using client[0|1]_tlp_dma input signal.
- HDMA does not support Legacy DMA test interrupt feature in this release.

7.2.2.2 Non-Linked List Mode Interrupt Handling

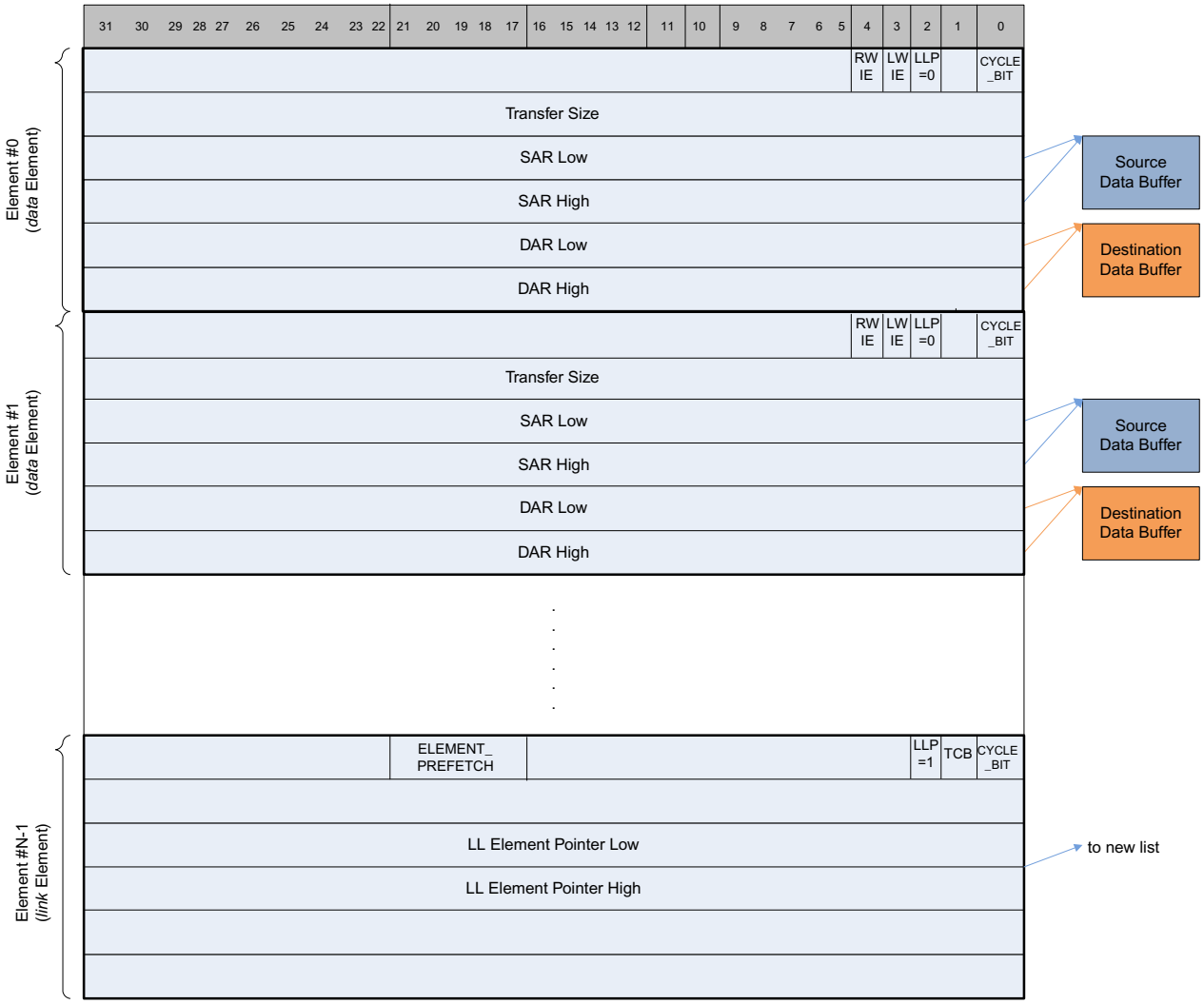
Non-linked list mode has the same type of interrupts as the linked list mode, except for LL related errors and watermark interrupts.

7.2.3 Linked List Mode

The HDMA provides a linked list (LL) mode to efficiently move data from source to destination with minimal intervention from the local CPU. This mode provides an alternative to programming the HDMA multiple times to transfer multiple blocks of data. The programming information (address, size, and so on) for each block of memory is pre-programmed by your software into a LL element (also known as a descriptor) in local memory. Each element (called a data element) in the LL structure (called a transfer list) can transfer up to 4 GB of data.

You enable LL operation for a channel, by setting the `LLEN` field of the `HDMA_CONTROL1_OFF_[WR|RD]CH_i` register to 1. You can enable LL mode independently for each channel. When you enable LL for more than one channel, then you must have a separate LL structure in local memory for each channel. Your application must produce the LL element structure in local memory as shown in Figure 7-4. Normally, all of the elements are contiguous (one after the other) in memory, and each element has six DWORDs containing the information about the block of data to be transferred. You program the channel context registers (`HDMA_LLP_LOW_OFF_[WR|RD]CH_i` and `HDMA_LLP_HIGH_OFF_[WR|RD]CH_i`) with the location of where you have placed the LL element structure in local memory.

Figure 7-4 Linked List Element/Descriptor Structure in Local Memory with N Elements



When you start the HDMA transfer (by writing to the HDMA Write Doorbell Register `HDMA_DOORBELL_OFF_WRCH_i.DB_START =1` or HDMA Read Doorbell Register `HDMA_DOORBELL_OFF_RDCH_i.DB_START =1`), the HDMA reads (consumes) elements continuously from local memory, and loads the information (SAR, DAR, transfer size, and so on) from that element into the channel descriptor queue. HDMA always tries to keep the channel descriptor queue full.

Normally, all of the elements are contiguous (one after the other) in memory, with the starting address defined in the channel context HDMA Linked List Pointer Low/High Register `HDMA_LLP_[LOW|HIGH]_OFF_[WR|RD]CH_i`. When you want to jump in local memory to another element list (or recycle the consumed elements), then you set the LLP bit in the element (for example, link element #N-1 in Figure 7-4), specify the location of the next element structure using the LL Element Pointer DWORDs (as indicated in Figure 7-4), and, set TCB to 1 (for recycling) or to 0 (to jump to another list).

Relationship Between Element DWORDs and HDMA Registers

Each element has six DWORDs as in Figure 7-4. The HDMA loads the six element DWORDs into the following registers:

Table 7-4 Linked List Element and HDMA Register Mapping

Link List Element	HDMA Register
Data Element	
CYCLE_BIT	HDMA_CYCLE_OFF_[WR|RD]CH_i
LLP	Load Link Pointer (LLP). Indicates that this linked list element is a link element, and its LL element pointer DWORDs are pointing to the next (non-contiguous) element. For Data element LLP =0.
LWIE	HDMA_WATERMARK_EN_OFF_[WR|RD]CH_i.LWIE
RWIE	HDMA_WATERMARK_EN_OFF_[WR|RD]CH_i.RWIE
Transfer Size	HDMA_XFERSIZE_OFF_[WR|RD]CH_i.XFERSIZE
SAR Low	HDMA_SAR_LOW_OFF_[WR|RD]CH_i.SAR_PTR_LOW
SAR High	HDMA_SAR_HIGH_OFF_[WR|RD]CH_i.SAR_PTR_HIGH
DAR Low	HDMA_DAR_LOW_OFF_[WR|RD]CH_i.DAR_PTR_LOW
DAR High	HDMA_DAR_HIGH_OFF_[WR|RD]CH_i.DAR_PTR_HIGH
Link Element	
CYCLE_BIT	HDMA_CYCLE_OFF_[WR|RD]CH_i.CYCLE_BIT
TCB	Toggle Cycle Bit (TCB). Indicates to the HDMA to toggle its interpretation of the Cycle Bit. It is used to synchronize the producer (software) and the consumer (HDMA). For more information, see "PCS-CCS-CB-TCB Producer-Consumer Synchronization" on page 517.
LLP	Load Link Pointer (LLP). Indicates that this linked list element is a link element, and its LL element pointer DWORDs are pointing to the next (non-contiguous) element. For link element LLP =1.

Link List Element	HDMA Register
ELEMENT_PREFECTCH	HDMA_ELEM_PF_OFF_[WR|RD]CH_i.ELEMENT_PREFETCH
LL Element Pointer Low	HDMA_LLP_LOW_OFF_[WR|RD]CH_i.ELEMENT_LIST_PTR_LOW
LL Element Pointer High	HDMA_LLP_HIGH_OFF_[WR|RD]CH_i.ELEMENT_LIST_PTR_HIGH

For more information on LL operation, see "Linked List Operation" on page 513.

Caution

Linked List Operation

- When you enable linked list operation (HDMA_CONTROL1_OFF_[WR|RD]CH_i.LLEN =1b1), the HDMA overwrites the HDMA registers in Table 7-4 with information from linked list data/link elements.
- The structure of a link element is different to that of the data element. A data element has no TCB field. A link element has no LWIE or RWIE fields. It has no SAR, DAR, or Transfer Size DWORDs, but has LL Element Pointer DWORDs instead of the SAR DWORDs.
- The LWIE and RWIE bits in a LL element, only enable the watermark interrupt. In non-LL mode, the RWIE and LWIE bits enable the done and abort interrupts. For more information, see "Interrupts and Error Handling" on page 533.
- HDMA_ELEM_PF_[WR|RD]CH_i.ELEMENT_PREFETCH is updated every time a linked list pointer element is loaded.

7.3 Using the HDMA

This section describes how to use the HDMA. The topics for this section are:

- "Source and Destination Address Registers (SAR, DAR)" on page 539
- "HDMA Transfer Size Register" on page 541
- "Starting the HDMA Transfer" on page 541
- "Detecting the End of the HDMA Transfer" on page 541
- "TLP Generator Header Control" on page 542
- "Stopping the HDMA Transfer (Software Stop)" on page 542
- "HDMA Software Register Map" on page 543

You can configure the HDMA to have from one to `CC_NUM_DMA_RD_CHAN` read channels and one to `CC_NUM_DMA_WR_CHAN` write channels. It can simultaneously perform the following types of memory transactions:

- HDMA write

  Transfer (copy) of a block of data from local memory to remote memory.
- HDMA read

  Transfer (copy) of a block of data from remote memory to local memory.

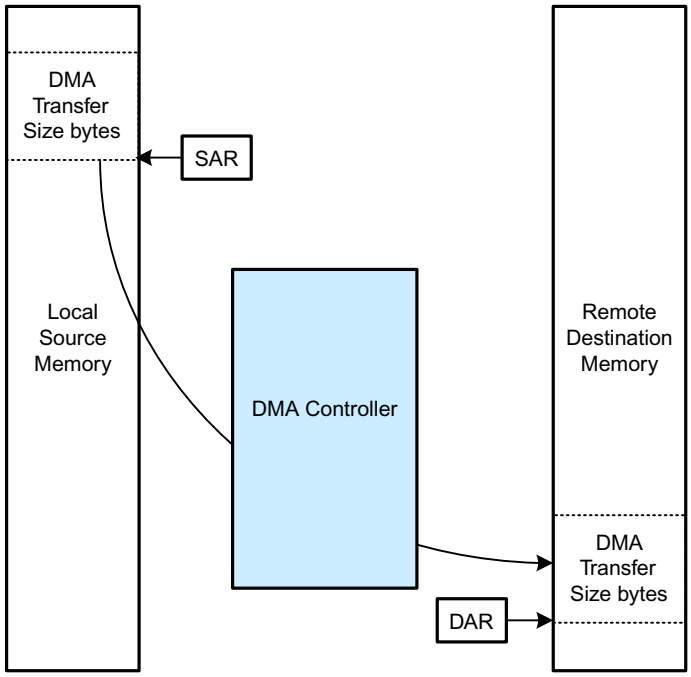
The HDMA supports full duplex operation, processing read and write transfers at the same time, and in parallel with normal (non-HDMA) traffic. Upon completion of a HDMA transfer or an error, the HDMA optionally interrupts the local CPU or sends an interrupt MWr (IMWr) to the remote CPU. The HDMA is highly configurable and you can program it using the local DBI, or over the PCIe wire.

The channels are named read channel `CC_NUM_DMA_RD_CHAN-1`..0 and write channel `CC_NUM_DMA_WR_CHAN-1`..0. Each channel is programmed using a number of registers. For more information, see "HDMA Software Register Map" on page 543.

7.3.1 Source and Destination Address Registers (SAR, DAR)

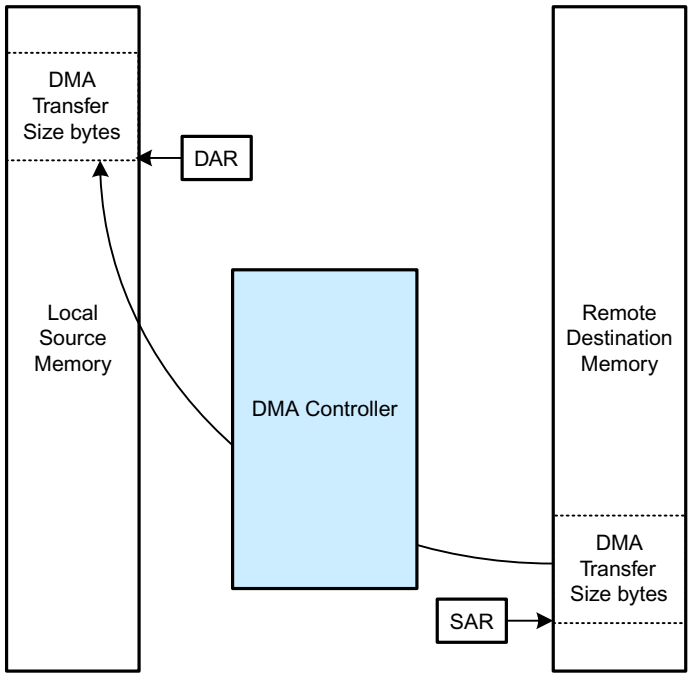
The HDMA channel context SAR and DAR registers (`HDMA_[SAR|DAR]_[LOW|HIGH}_OFF_[WR|RD]CH_i`) provide support for remote-to-local, and local-to-remote PCIe address mapping. You program the start of the local and remote data buffers using these registers, and the HDMA increments the SAR and DAR as the HDMA transfer progresses. For a write transfer, the SAR is the address of the local memory, and the DAR is the address of the remote memory, as shown in Figure 7-5.

Figure 7-5 Write Transfer: SAR and DAR for Write Channel 0



For a read transfer, the SAR is the address of the remote memory, and the DAR is the address of the local memory, as shown in Figure 7-6.

Figure 7-6 Read Transfer: SAR and DAR for Read Channel 0



The presence of the HDMA controller does not affect:

- Normal filtering rules for inbound TLPs (for example, BAR checking in an Endpoint, as described in "Internal Address Translation Unit (iATU)" on page 220)
- The operation of any internal or external address translation, as described in "Receive Filtering" on page 134

Caution

When you do not want the iATU to translate outbound requests that are generated by the internal HDMA module, then you must implement one of the following approaches:

- Ensure that the combination of HDMA channel programming and iATU control register programming, causes no translation of DMA traffic to be done in the iATU.
- Activate the HDMA bypass mode to allow request TLPs which are initiated by the HDMA controller to pass through the iATU untranslated. You can activate HDMA bypass mode by setting the HDMA Bypass field of the iATU Control 2 Register (IATU_REGION_C-TRL_OFF_2_OUTBOUND_0).

7.3.2 HDMA Transfer Size Register

You program the HDMA transfer size using the HDMA Transfer Size Register (`HDMA_XFERSIZE_OFF_[WR|RD]CH_i`). The maximum HDMA transfer size is 4GB, and the minimum transfer size is one byte (0x1). The HDMA decrements the value in this register as the HDMA transfer progresses. When all bytes are successfully transferred, the value in this register is zero. In LL mode, the HDMA overwrites this register with the corresponding DWORD of the LL element.This register holds valid data when HDMA stops the transfer. HDMA operates on multiple descriptors at the same time, so this register may not provide accurate transfer progress information.

7.3.3 Starting the HDMA Transfer

After you program the HDMA controller registers (including writing to the HDMA Read Channel Enable or HDMA Write Channel Enable register), you start a HDMA transfer by setting the `DB_START` field of the HDMA Write/Read Channel Doorbell Register (`HDMA_DOORBELL_OFF_[WR|RD]CH_i`). You can program and start both a read and a write transfer at the same time. The HDMA supports full duplex operation, processing read and write transfers at the same time and in parallel with normal (non-HDMA) traffic. You can program and start any number of channels in the HDMA controller sequentially or simultaneously.

- **Sequentially**: configure and start a channel, and then configure and start another channel
- **Simultaneously**: configure multiple channels, and then start multiple channels

Caution

You must not write to any of the context registers for a particular channel after you start the channel by setting the DB_START field of the HDMA Write/Read Channel Doorbell Register (HDMA_DOORBELL_OFF_[WR|RD]CH_i).

7.3.4 Detecting the End of the HDMA Transfer

If an HDMA transfer proceeds without any errors, it stops automatically when finished.

Detecting End of Transfer without Errors

The normal end of an HDMA transfer is detected by any of the following methods:

- Stop local interrupt (pin) asserted.
- Done remote interrupt (IMWr) received.
- `STATUS` field of the HDMA Write/Read Channel Status register (`HDMA_STATUS_OFF_[WR|RD]CH_i`) is Stopped, and the HDMA Transfer Size register is 0x0.
- Polling of the HDMA Write/Read Interrupt Status register (`HDMA_INT_STATUS_OFF_[WR|RD]CH_i`).

Detecting End of Transfer with Errors

The abnormal end of an HDMA transfer is detected by any of the following methods:

- Abort local interrupt (pin) asserted.
- Abort remote interrupt (IMWr) received.
- Polling of the HDMA Write/Read Interrupt Status Register (`HDMA_INT_STATUS_OFF_[WR|RD]CH_i`).
- `STATUS` field of the HDMA Write/Read Channel Status register is Aborted. When the HDMA controller detects an error, it forces the HDMA to stop issuing requests for the channel. It also sets the channel status field of the HDMA Write/Read Channel Status register to Aborted, generates an abort interrupt (if enabled), and sends an abort IMWr (if enabled). The HDMA Transfer Size register indicates the remaining number bytes to be transferred, except when there is an AXI write error during a HDMA read transfer. For more information, see "Error Handling" on page 510.

7.3.5 TLP Generator Header Control

You must program bits [26:1] of the HDMA Write/Read Channel Control 1 register (`HDMA_CONTROL1_OFF_[WR|RD]CH_i`), so that the HDMA TLP generator correctly sets the TC, RO, NS, AT, and FN TLP header fields for any TLPs that it generates.

7.3.6 Stopping the HDMA Transfer (Software Stop)

You can stop the HDMA by setting the `DB_STOP` field of the `HDMA_DOORBELL_OFF_WRCH_i` register to '1'. This causes the HDMA to:

- Clear the doorbell state and stop the linked list requests
- Flush the descriptors already loaded and not yet processed
- Consider the descriptor at the head of the queue as the stop marker
- Complete the data read request already sent as expected
- Flush the read requests not yet queued
- Change the channel status to Stopped when the stop processing is confirmed. HDMA presents the context of the point where it stopped. If required, it allows to resume from the stop point.

The HDMA doorbell stop is compliant with the PCIe FLR (Function Level Reset). You might also do this as part of a function level reset (FLR). FLR does not directly affect the HDMA transfer so you must manually stop the HDMA transfer before initiating an FLR.

Note Software can Stop HDMA on L0 state only, it is not possible to Stop HDMA in Low Power states.

7.3.7 HDMA Software Register Map

The register map in HDMA is distributed to avoid bottlenecks associated with shared central memory access.

- HDMA Interrupt Registers: these registers control the generation of local interrupts and remote IMWr. The IMWr addresses and data are stored in RAM.

Table 7-5 HDMA Interrupt Registers

Byte Offset	Register	Description
B+ i * 0x200 + 0x080	HDMA_STATUS_OFF_WRCH_i	HDMA Write Channel Status
B+ i * 0x200 + 0x180	HDMA_STATUS_OFF_RDCH_i	HDMA Read Channel Status
B+ i * 0x200 + 0x084	HDMA_INT_STATUS_OFF_WRCH_i	HDMA Write Channel Interrupt Status
B+ i * 0x200 + 0x184	HDMA_INT_STATUS_OFF_RDCH_i	HDMA Read Channel Interrupt Status
B+ i * 0x200 + 0x088	HDMA_INT_SETUP_OFF_WRCH_i	HDMA Write Channel Interrupt Setup
B+ i * 0x200 + 0x188	HDMA_INT_SETUP_OFF_RDCH_i	HDMA Read Channel Interrupt Setup
B+ i * 0x200 + 0x08C	HDMA_INT_CLEAR_OFF_WRCH_i	HDMA Write Channel Interrupt Clear
B+ i * 0x200 + 0x18C	HDMA_INT_CLEAR_OFF_RDCH_i	HDMA Read Channel Interrupt Clear
B+ i * 0x200 + 0x090	HDMA_MSI_STOP_LOW_OFF_WRCH_i	HDMA Write Stop Remote Interrupt Address Low
B+ i * 0x200 + 0x190	HDMA_MSI_STOP_LOW_OFF_RDCH_i	HDMA Read Stop Remote Interrupt Address Low
B+ i * 0x200 + 0x094	HDMA_MSI_STOP_HIGH_OFF_WRCH_i	HDMA Write Stop Remote Interrupt Address High
B+ i * 0x200 + 0x194	HDMA_MSI_STOP_HIGH_OFF_RDCH_i	HDMA Read Stop Remote Interrupt Address High
B+ i * 0x200 + 0x098	HDMA_MSI_ABORT_LOW_OFF_WRCH_i	HDMA Write Abort Remote Interrupt Address Low
B+ i * 0x200 + 0x198	HDMA_MSI_ABORT_LOW_OFF_RDCH_i	HDMA Read Abort Remote Interrupt Address Low
B+ i * 0x200 + 0x09C	HDMA_MSI_ABORT_HIGH_OFF_WRCH_i	DMA Write Abort Remote Interrupt Address High
B+ i * 0x200 + 0x19C	HDMA_MSI_ABORT_HIGH_OFF_RDCH_i	DMA Read Abort Remote Interrupt Address High
B+ i * 0x200 + 0x0A0	HDMA_MSI_WATERMARK_LOW_OFF_WRCH_i	HDMA Write Watermark Remote Interrupt Address Low
B+ i * 0x200 + 0x1A0	HDMA_MSI_ABORT_LOW_OFF_RDCH_i	HDMA Read Watermark Remote Interrupt Address Low

Byte Offset	Register	Description
B+ i * 0x200 + 0x0A4	HDMA_MSI_ABORT_HIGH_OFF_WRCH_i	HDMA Write Watermark Remote Interrupt Address High
B+ i * 0x200 + 0x1A4	HDMA_MSI_ABORT_HIGH_OFF_RDCH_i	HDMA Read Watermark Remote Interrupt Address High
B+ i * 0x200 + 0x0A8	HDMA_MSI_MSGD_OFF_WRCH_i	HDMA Write Channel Remote Interrupt Data
B+ i * 0x200 + 0x1A8	HDMA_MSI_MSGD_OFF_RDCH_i	HDMA Read Channel Remote Interrupt Data

- HDMA Channel Context Registers: these registers are for the control and status information that is specific to each channel. There is a set of registers for each channel.

Table 7-6 HDMA Channel Context Registers for Each Channel

Byte Offset	Register	Description
B+ i * 0x200 + 0x000	HDMA_EN_OFF_WRCH_i	HDMA Write Channel Enable
B+ i * 0x200 + 0x100	HDMA_EN_OFF_RDCH_i	HDMA Read Channel Enable
B+ i * 0x200 + 0x004	HDMA_DOORBELL_OFF_WRCH_i	HDMA Write Channel Doorbell
B+ i * 0x200 + 0x104	HDMA_DOORBELL_OFF_RDCH_i	HDMA Read Channel Doorbell
B+ i * 0x200 + 0x008	HDMA_ELEM_PF_OFF_WRCH_i	HDMA Write Channel Prefetch
B+ i * 0x200 + 0x108	HDMA_ELEM_PF_OFF_RDCH_i	HDMA Read Channel Prefetch
B+ i * 0x200 + 0x00C	HDMA_HANDSHAKE_OFF_WRCH_i	HDMA Write Channel Handshake
B+ i * 0x200 + 0x10C	HDMA_HANDSHAKE_OFF_RDCH_i	HDMA Read Channel Handshake
B+ i * 0x200 + 0x010	HDMA_LLP_LOW_OFF_WRCH_i	HDMA Write Channel Linked List Pointer Low
B+ i * 0x200 + 0x110	HDMA_LLP_LOW_OFF_RDCH_i	HDMA Read Channel Linked List Pointer Low
B+ i * 0x200 + 0x014	HDMA_LLP_HIGH_OFF_WRCH_i	HDMA Write Channel Linked List Pointer High
B+ i * 0x200 + 0x114	HDMA_LLP_HIGH_OFF_RDCH_i	HDMA Read Channel Linked List Pointer High
B+ i * 0x200 + 0x018	HDMA_CYCLE_OFF_WRCH_i	HDMA Write Channel Producer-Consumer Cycle Synchronization
B+ i * 0x200 + 0x118	HDMA_CYCLE_OFF_RDCH_i	HDMA Read Channel Producer-Consumer Cycle Synchronization
B+ i * 0x200 + 0x01C	HDMA_XFERSIZE_OFF_WRCH_i	HDMA Write Channel Transfer Size
B+ i * 0x200 + 0x11C	HDMA_XFERSIZE_OFF_RDCH_i	HDMA Read Channel Transfer Size
B+ i * 0x200 + 0x120	HDMA_SAR_LOW_OFF_RDCH_i	HDMA Read Channel SAR Low
B+ i * 0x200 + 0x124	HDMA_SAR_HIGH_OFF_RDCH_i	HDMA Read Channel SAR High
B+ i * 0x200 + 0x028	HDMA_DAR_LOW_OFF_WRCH_i	HDMA Write Channel DAR Low

Byte Offset	Register	Description
B+ i * 0x200 + 0x128	HDMA_DAR_LOW_OFF_RDCH_i	HDMA Read Channel DAR Low
B+ i * 0x200 + 0x02c	HDMA_DAR_HIGH_OFF_WRCH_i	HDMA Write Channel DAR High
B+ i * 0x200 + 0x12c	HDMA_DAR_HIGH_OFF_RDCH_i	HDMA Read Channel DAR High
B+ i * 0x200 + 0x030	HDMA_WATERMARK_EN_OFF_WRCH_i	HDMA Write Channel Linked-list Watermark Enable
B+ i * 0x200 + 0x130	HDMA_WATERMARK_EN_OFF_RDCH_i	HDMA Read Channel Linked-list Watermark Enable
B+ i * 0x200 + 0x034	HDMA_CONTROL1_OFF_WRCH_i	HDMA Write Channel Control Settings 1
B+ i * 0x200 + 0x134	HDMA_CONTROL1_OFF_RDCH_i	HDMA Write Channel Control Settings 1
B+ i * 0x200 + 0x038	HDMA_FUNC_NUM_OFF_WRCH_i	HDMA Write Channel Function Number
B+ i * 0x200 + 0x138	HDMA_FUNC_NUM_OFF_RDCH_i	HDMA Write Channel Function Number
B+ i * 0x200 + 0x03C	HDMA_QOS_OFF_WRCH_i	HDMA Write Channel QoS Settings
B+ i * 0x200 + 0x13C	HDMA_QOS_OFF_RDCH_i	HDMA Read Channel QoS Settings

Legacy DMA and HDMA Software Compatibility

You can reuse your legacy DMA software to drive HDMA when it operates in compatibility mode (`CC_LEGACY_DMA_MAP =1`). However, you must consider the following exceptions while reusing your legacy DMA software for HDMA.

- HDMA does not implement engine enable, that is `DMA_[WRITE|READ]_ENGINE_EN_OFF.DMA_[WRITE|READ]_ENGINE` field.
- In legacy DMA, when the hidden parameter `CC_DMA_CH_PWR_EN` is set to '1', you can use the `DMA_[WRITE|READ]_CHi_PWR_EN_OFF` register to enable/disable power for a channel. HDMA also supports this legacy DMA feature.
- Legacy DMA releases the watermark interrupt only when it loads the next descriptor. This provides a synchronization mechanism for determining the end of the list, if the last descriptor is watermarked.

  HDMA does not have such a restriction on watermark interrupts. HDMA releases a watermark interrupt on successful transfer of data for that descriptor, while processing other descriptors.

  So, you must keep a track of the number of watermarked descriptors created, and the number of watermark interrupts released, to determine the accurate transfer location.
- Legacy DMA reports poisoned data error for the read engine only.

  HDMA handles poisoned data based on `CC_DMA_ABORT_ON_POISONED` parameter setting for the read engine as well as the write engine.

  You must take the differences in poisoned data handling behavior between the legacy DMA and HDMA into consideration.

Table 7-7 HDMA and Legacy DMA Register Mapping (CC_LEGACY_DMA_MAP =1)

HDMA Register Name	Corresponding Legacy DMA Register
HDMA_DOORBELL_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_DOORBELL_OFF
HDMA_ELEM_PF_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i. DMA_ELEMENT_PREFETCH
HDMA_HANDSHAKE_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_ENGINE_HSHAKE_CNT_LOW_OFF DMA_[WRTE|READ]_ENGINE_HSHAKE_CNT_HIGH_OFF
HDMA_LLP_LOW_OFF_[WR|RD]CH_i	DMA_LLP_LOW_OFF_[WR|RD]CH_i
HDMA_LLP_HIGH_OFF_[WR|RD]CH_i	DMA_LLP_HIGH_OFF_[WR|RD]CH_i
HDMA_CYCLE_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i.CB DMA_CH_CONTROL1_OFF_[WR|RD]CH_i.CCS
HDMA_XFERSIZE_OFF_[WR|RD]CH_i	DMA_TRANSFER_SIZE_OFF_[WR|RD]CH_i
HDMA_SAR_LOW_OFF_[WR|RD]CH_i	DMA_SAR_LOW_OFF_[WR|RD]CH_i
HDMA_SAR_HIGH_OFF_[WR|RD]CH_i	DMA_SAR_HIGH_OFF_[WR|RD]CH_i
HDMA_DAR_LOW_OFF_[WR|RD]CH_i	DMA_DAR_LOW_OFF_[WR|RD]CH_i
HDMA_DAR_HIGH_OFF_[WR|RD]CH_i	DMA_DAR_HIGH_OFF_[WR|RD]CH_i
HDMA_WATERMARK_EN_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i
HDMA_CONTROL1_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i
HDMA_FUNC_NUM_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i
HDMA_QOS_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i. DMA_TC DMA_[WRITE|READ]_CHANNEL_ARB_WEIGHT_LOW_OFF DMA_[WRITE|READ]_CHANNEL_ARB_WEIGHT_HIGH_OFF DMA_[WRITE|READ]_ENGINE_PF_OFF
HDMA_STATUS_OFF_[WR|RD]CH_i	DMA_CH_CONTROL1_OFF_[WR|RD]CH_i
HDMA_INT_STATUS_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_INT_STATUS_OFF DMA_[WRITE|READ]_ERR_STATUS_OFF DMA_[WRITE|READ]_LINKED_LIST_ERR_EN_OFF
HDMA_INT_SETUP_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_INT_MASK_OFF DMA_[WRITE|READ]_LINKED_LIST_ERR_EN_OFF
HDMA_INT_CLEAR_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_INT_CLEAR_OFF

HDMA Register Name	Corresponding Legacy DMA Register
HDMA_MSI_STOP_LOW_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_DONE_IMWR_LOW_OFF DMA_[WRITE|READ]_DONE_IMWR_HIGH_OFF
HDMA_MSI_STOP_HIGH_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_DONE_IMWR_HIGH_OFF
HDMA_MSI_WATERMARK_LOW_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_DONE_IMWR_LOW_OFF
HDMA_MSI_WATERMARK_HIGH_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_DONE_IMWR_HIGH_OFF
HDMA_MSI_ABORT_LOW_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_ABORT_IMWR_LOW_OFF
HDMA_MSI_ABORT_HIGH_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_ABORT_IMWR_HIGH_OFF
HDMA_MSI_MSGD_OFF_[WR|RD]CH_i	DMA_[WRITE|READ]_CH01_IMWR_DATA_OFF DMA_[WRITE|READ]_CH23_IMWR_DATA_OFF DMA_[WRITE|READ]_CH45_IMWR_DATA_OFF DMA_[WRITE|READ]_CH67_IMWR_DATA_OFF

7.4 Advanced HDMA Information and Operation

This section discusses advanced features and operation of the DMA. The following topics are covered in this section:

- "Overview"
- "Error Handling"
- "AXI Ordering (HDMA)" on page 549
- "HDMA Debug" on page 549

7.4.1 Overview

HDMA advanced features work in a manner similar to DMA, the differences in operation are discussed in this section.

7.4.2 Error Handling

HDMA monitors the completion of transactions at XALI0 and TRGT1 interfaces, the possible sources of HDMA errors are the same as legacy DMA. Table 6-14 on page 510 specifies the possible sources of errors during DMA read transfer and Table 6-15 on page 510 specifies the possible sources of errors during DMA write transfer. HDMA error handling is slightly different to DMA error handling. When the HDMA detects an error, it performs the following actions (for the channel with the error):

- HDMA trashes all subsequent transactions at the stage where an error occurred (data fetch, descriptor fetch and processing, completion processing, or interrupt and channel status handling).
- HDMA trashes all completion data stored in the HDMA read buffer, or outstanding at the wire/application, in order.
- HDMA clears the doorbell state for the channel.
- HDMA trashes all the prefetched descriptors.
- When the channel processes all its outstanding transactions, HDMA halts the channel by setting `HDMA_INT_STATUS_OFF_[RD|WR}CH_i.ABORT =1` and `HDMA_STATUS_OFF_[WR|RD]CH_i .STATUS = ABORTED`. The error is specified in `HDMA_INT_STATUS_OFF_[WR|RD]CH_i.ERROR`.

Note HDMA detects more than one error in different HDMA processing stages (data fetch, descriptor fetch and processing, completion processing, or interrupt and channel status handling), but it records only the first error in the transfer stream and trashes all the subsequent errors. For example, if at a time HDMA detects an error on a CPL data, and later, on a write response, since the write response is earlier in the transfer stream, the HDMA registers point to the location of write response error and not the CPL data error.

- HDMA does not require soft reset. After the channel is halted and/or the interrupts are generated, the channel context points to the last successful transaction.

7.4.3 AXI Ordering (HDMA)

HDMA issues unique AXI ID for each of the multiple outstanding descriptors. The HDMA CPL ordering logic in the AXI bridge does the ordering for each of the following channels:

1. HDMA write engine data transfer CPL
2. HDMA write engine descriptor CPL
3. HDMA read engine descriptor CPL

7.4.4 HDMA Debug

To assist you in debugging your hardware, use the following outputs to identify HDMA traffic (transfer requests, and linked list element MRds) and non-HDMA traffic.

Table 7-8 DMA Debug Outputs

Signal Name	Where Used
armisc_info_dma	AXI manager
awmisc_info_dma	AXI manager

Soft Reset

HDMA does not support this legacy DMA feature in this release.

Test Interrupt

HDMA does not support this legacy DMA feature in this release.

8

Cache Coherent Interconnect for Accelerators (CCIX)

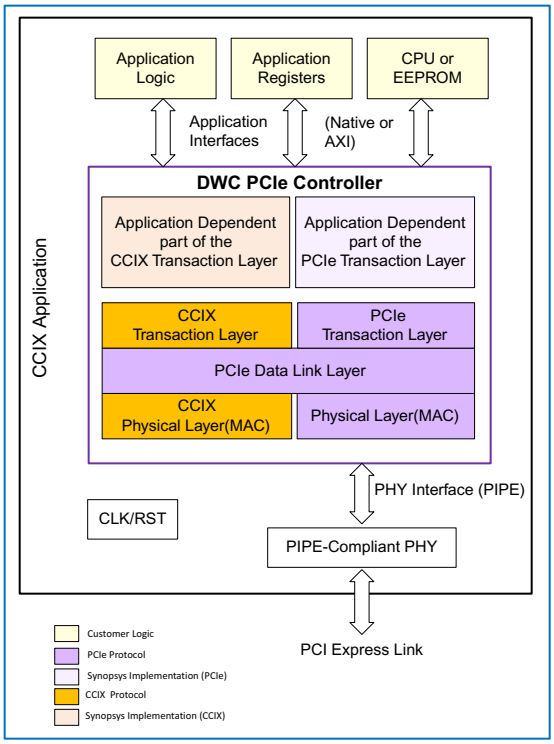
This section discusses the CCIX PCIe controller. The following topics are discussed:

- "CCIX PCIe Controller Overview" on page 552
- "CCIX PCIe Controller Operation" on page 557
- "CXS Interface" on page 565

8.1 CCIX PCIe Controller Overview

The CCIX PCIe controller enables hardware accelerators to use memory shared with multiple processors in a cache coherent manner. CCIX's coherence protocol can be carried across PCIe links. The existing PCIe controller implementation is extended with logic to implement CCIX Transport. CCIX Transport comprises of CCIX and PCIe Transaction Layers, PCIe Data link Layer, and CCIX Physical Layer.

Figure 8-1 CCIX Application



8.1.1 Configuring the CCIX PCIe Controller

The configuration parameters relevant to the CCIX are described in Table 8-1.

Table 8-1 CCIX Configuration Parameters

Parameter Name	Parameter Label	Value Range	Default
CX_CCIX_ENABLE	CCIX Protocol Supported	1, 0	0

Table 8-2 Other Configuration Parameters that Indirectly Affect CCIX Operation

Parameter Name	Description
CX_CCIX_ESM_SUPPORT	ESM Mode Supported
CX_CCIX_INTERFACE_ENABLE	CCIX Interface and optimized TLP supported

8.1.2 CCIX PCIe Controller Features

- All non-optional features of the *CCIX Transport Specification.*
- Optional CCIX Extended Speed Mode (ESM) feature to extend the supported data rates to 20.0GT/s and 25.0 GT/s.
- CCIX Transport DVSEC containing CCIX specific Control and Status Registers (CSRs) for CCIXPhysical, Data Link and Transaction Layers.

8.1.3 CCIX PCIe Controller Limitations

- Native PCIe controller data path width must be greater than or equal to 128 bits.
- AMBA bridge is supported for single master channel architecture (`CC_HP_MASTER =0`) only.
- TX Interface
  - TLPs using the traffic class (TC) dedicated to CCIX cannot be sent on the transmit client interfaces (XALI0/1/2).
  - The CCIX PCIe controller does not check for TLP errors; it forwards the TLP to your application as presented on the CCIX interface. This is consistent with the behavior of the transmit client interfaces (XALI0/1/2).
  - The CCIX PCIe controller does not check whether the TLP payload size is less than `PCIe Max Payload Supported (CX_MAX_MTU)` or not. Exceeding this limit may overflow the retry buffer, resulting in data corruption. This is consistent with the behavior of the transmit client interfaces (XALI0/1/2).
  - ECRC pass through is not supported, so your application must not forward TLPs with ECRC to XALI_CCIX.
- RX Interface
  - Bypass mode is not supported for receive posted CCIX VC queue.
  - The received CCIX VDM1 is forwarded when the function number in the Transaction ID matches the implemented functions.
  - The received CCIX Optimized TLPs are forwarded even if the FLR of any PF/VF is initiated.
  - The received CCIX Compatible TLPs are discarded when the target function is under FLR.
  - ECRC error pass through is not supported, so TLPs with ECRC error are not forwarded to TRGT1_CCIX.
- Lane Margining at Receiver is not supported at 16.0 GT/s of ESM Data Rate0.
- Arbitration between CCIX transmit interface XALI_CCIX and XALI0/1/2 is fixed; XALI_CCIX interface gets priority over XALI0/1/2 interfaces.
- The value of `MISC_CONTROL_1_OFF` register fields `DEFAULT_TARGET` and `UR_CA_MASK_4_TRGT1` affect the filter rules only when `ccix_tlp_disable =1`. When `ccix_tlp_disable =0,` the CCIX PCIe controller considers `DEFAULT_TARGET =0` and `UR_CA_MASK_4_TRGT1 =0,` irrespective of the value set for these fields.
- RASDES
  - Event Counter
    - The result of Group 7 (Layer3 non-error) is incorrect if the counters are enabled while Enable Optimized TLP Generation and Reception bit is 1.

        - The group 0-6 works correctly.
    - Time Based Analysis
        - The analysis must be disabled while the link is in the ESM mode.
        - The result of Group 1 (Throughput) is incorrect if the analysis is enabled while Enable Optimized TLP Generation and Reception bit is 1.
        - The group 0 works correctly.
    - Error Injection
        - Error Injection Control 6 (Packet Error) cannot insert errors into the TLPs transmitted from the XALI_CCIX interface.
- The CCIX PCIe controller does not implement CCIX Protocol DVSEC registers. You must implement them externally.
    - If the device type is RP, they should be connected directly to the application's interconnect.
    - If the device type is not RP, they can be connected to the ELBI.
    - Regardless of the device type, the PCI capability list must be updated through the DBI read only write feature to write the offset of the CCIX Protocol DVSEC. For more information on DBI read only write feature, see "Writing to Read-Only Registers" on page 144).
- The CCIX PCIe controller does not support DMA.

8.1.4 Frequency, Speed, and Width Support

Before the link training is initiated your application must configure:

- `CCIX_VDM_VID` field in the `CCIX_CTRL_OFF` register (configures the Vendor ID field for CCIX VDM1 TLPs)
- `CCIX_DVSEC_VID` field in the `CCIX_TP_HDR1_OFF` register (configures the Vendor ID register for DVSEC when this register is written usingDBI. For more information on writing to Read-Only registers over DBI, see "Writing to Read-Only Registers" on page 144)

A CCIX device supports one of the three PHY types:

- PCI Express 4.0, or
- PCI Express 5.0, or
- Extended Data Rate (EDR)

You can configure PHY type using `CX_CCIX_ESM_SUPPORT` parameter.

Table 8-3 CCIX PHY Types and Operating Modes

PHY Type	PCIe Express Mode	ESM Mode	CCIX Speed Mode
PCI Express 4.0 or PCI Express 5.0 (CX_CCIX_ESM_SUPPORT=0)	2.5 GT/s	Not Applicable	CCIX Speed1
	5.0 GT/s	Not Applicable	CCIX Speed2
	8.0 GT/s	Not Applicable	CCIX Speed3
	16.0 GT/s	Not Applicable	CCIX Speed4
	32.0 GT/s	Not Applicable	GEN5_Speed

PHY Type	PCIe Express Mode	ESM Mode	CCIX Speed Mode
Extended Data Rate (EDR) (CX_CCIX_ESM_SUPPORT=1)	2.5 GT/s	2.5 GT/s	CCIX Speed1
	5.0 GT/s	5.0 GT/s	CCIX Speed2
	8.0 GT/s	ESM Data Rate0 8.0 GT/s or 16.0 GT/s	CCIX Speed3
	16.0 GT/s	ESM Data Rate1 16.0 GT/s or 20.0 GT/s or 25.0 GT/s	CCIX Speed4

Table 8-4 Supported Controller Configurations

core_clk @ CCIX Speed {1,2,3,4} [a]		CCIX Speed1 Mode[b]	CCIX Speed2 Mode	CCIX Speed3 Mode	CCIX Speed4 Mode	Gen5 Speed Mode	Max[c] Link Width (for each controller[d] Datapath width)		
							128-bit	256-bit	512-bit
CCIX ESM 20/25 Configuration	125, 250, (500 or 1000), (1000 or 1250 or 1562.5)	ccix1_2s	ccix2_2s_df	ccix3_2s_df	ccix4_2s_df	NA	x8	x16	NA
	62.5, 125, (250 or 500), (500 or 625 or 781.25)	ccix1_4s	ccix2_4s_df	ccix3_4s_df	ccix4_4s_df	NA	x4	x8	x16
	125, 250, (250 or 500), (500 or 625 or 781.25), 1000	ccix_2s	ccix_2s	ccix_4s	ccix_4s	g5_4s	x4	x8	x16

a. The controller supports PIPE SerDes architecture (CX_PIPE_SERDES_ARCH =1) for the highlighted configurations.
b. The ccix value indicates the CCIX speed mode. The s value indicates the number of 8-bit symbols processed per clock cycle per lane by the CCIX PCIe controller PIPE interface, in the indicated speed mode. df indicates dynamic frequency.
c. The value in each cell indicates the maximum link width supported. You can configure the controller with a link width up to this value, and coreConsultant automatically calculates the datapath width.
d. For license scheme information, see the “Checking License Requirements” section in the DWC PCI Express Controller Installation Guide.

Table 8-5 indicates the supported controller PHY combinations.

Table 8-5 Supported Controller PHY Combinations

	PHY			
	ccix1_2s **ccix2_2s** **ccix3_2s** **ccix4_2s**	**ccix1_4s** **ccix2_4s** **ccix3_4s** **ccix4_4s**	**ccix1_1s** **ccix2_1s** **ccix3_4s** **ccix4_4s**	**ccix1_2s** **ccix2_2s** **ccix3_4s** **ccix4_4s** **g5_4s**
ccix1_2s ccix2_2s ccix3_2s ccix4_2s	✓	✓	✗	✗
ccix1_4s ccix2_4s ccix3_4s ccix4_4s	✓	✓	✓	✗
ccix1_2s ccix2_2s ccix3_4s ccix4_4s g5_4s	✗	✗	✗	✓

8.2 CCIX PCIe Controller Operation

This section describes the CCIX PCI controller operation. The following topics are discussed in this section:

- "CCIX Extended Speed Mode Overview"
- "PIPE Interface" on page 560
- "CCIX Transaction Layer Receiver Operation" on page 563

8.2.1 CCIX Extended Speed Mode Overview

The following topics are discussed in this section:

- "Initial Speed Change Sequence"
- "ESM Equalization (EQ)" on page 559
- "Quick Equalization (EQ) Redo" on page 559
- "Software Equalization (EQ) Redo" on page 560
- "Polling.Compliance Testing" on page 560
- "LTSSM Enhancements to Support ESM Operation" on page 560

8.2.1.1 Initial Speed Change Sequence

The initial speed change sequence to attain ESM data rate is as follows:

1. PCIe compliant phase
   a. Physical Link is fully compliant to *PCI Express Base Specification, Revision 6.0, Version 1.0*.
   b. Initial transition to LTSSM L0 state operating at 16.0 GT/s data rate.
   c. Link-Up to DL_Active state and Flow Control Initialization is complete.
2. Software based discovery and PHY Calibration in L1 st ate
   a. Your application probes the ESM Data Rate Capabilities of the link partner to determine data rates and configures the ESM Data Rate 0/1 (`ESM_DATA_RATE0/ESM_DATA_RATE1`) fields of the ESM Control register (`ESM_CNTL_REG`).
   b. Your application transitions the link to 2.5 GT/s data rate.
   c. Your application probes the ESM Data Rate Capabilities of the link partner to determine the Link Reach Target and configures the Link Reach Target (`LINK_REACH_TARGET`) of the controller. Based on the value of `LINK_REACH_TARGET`, the ESM Extended Equalization Phase3 Timeout (`ESM_EXT_EQ3_DSP_TIMEOUT`)and ESM Extended Equalization Phase2 Timeout (`ESM_EXT_EQ2_USP_TIMEOUT`) fields of the ESM Control register are programmed.
   d. Your application queries ESM Mandatory Data Rate Capabilities and ESM Optional Data Rate Capabilities registers of the link partner, and sets the selected data rates into the Rate Capabilities of the link partner to determine data rates and configures the ESM Data Rate 0/1 (`ESM_DATA_RATE0/ESM_DATA_RATE1`) fields of the ESM Control register (`ESM_CNTL_REG`).
   e. Your application sets ESM Perform Calibration (`ESM_PERFORM_CAL`) field of ESM Control Register for the link partner as well as the controller.
   f. Your application drives link to L1 state using L1-PM mechanism.
   g. MAC requests PHY calibration on all lanes using the M2P_MessageBus.

h. PHY performs calibration and acknowledges the calibration complete on all the non-turn-off lanes to MAC using the P2M_MessageBus.

i. MAC collects the calibration complete on all the non-turn-off lanes. If it is back on all the non-turn-off lanes, the controller sets the Calibration Complete field (`ESM_CALIB_CMPLT`) of ESM status register (`ESM_STAT_REG`) to 1.

j. Your application drives an exit from L1 state, using the L1-PM mechanism, after the predetermined ESM Calibration Time has expired.

k. On exit from L1 state, your application must check the Calibration Complete field (`ESM_CALIB_CMPLT`) of the ESM status register (`ESM_STAT_REG`) on both sides of the link.

   i. If `ESM_CALIB_CMPLT =0`, ESM transition is discontinued and the remaining steps are not performed.

   ii. If `ESM_CALIB_CMPLT =1`, your application must set the ESM Enable (`ESM_ENABLE`) field of the ESM Control register (`ESM_CNTL_REG`) for both the link partners.

3. Link transition to selected ESM Data Rates

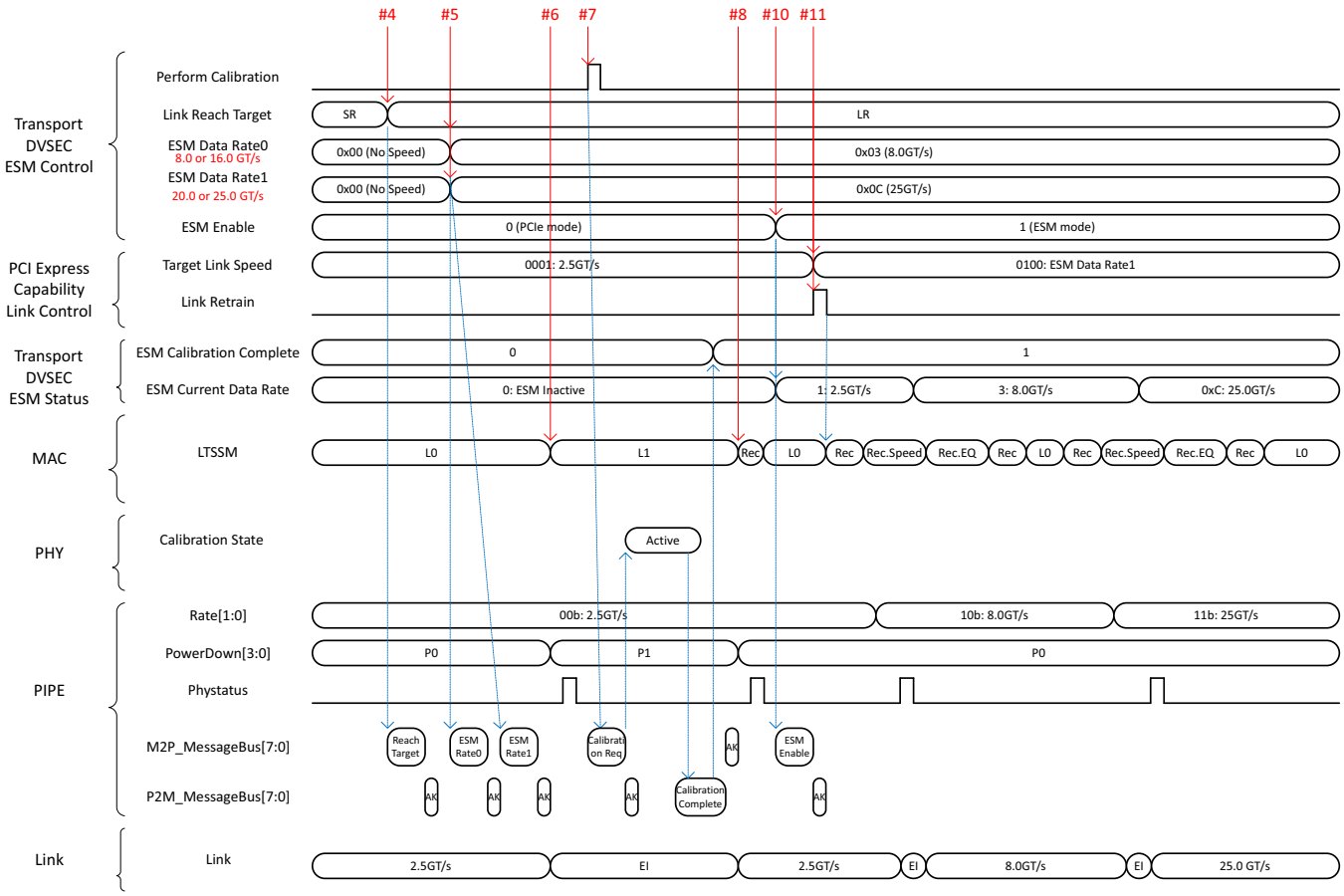
   a. Your application sets Target Link Speed and Link Retrain on DSP.

   b. In this step, Data Rate change occurs based on PCI Express LTSSM.

Note

Only `PCIE_CAP_MAX_LINK_SPEED` field of the Link Capabilities register determines the maximum link speed of the port. The physical layer ESM mode indicated by `ESM_MODE_SUPPORT` field of the `CCIX_TP_CAP_TP_HDR2_OFF` register, or ESM Data Rate set in the ESM Mandatory Data Rate Capability register (`ESM_MNDTRY_RATE_CAP_OFF`), do not determine the maximum link speed of the port. You must set the parameter `CX_MAX_PCIE_SPEED` or `PCIE_CAP_MAX_LINK_SPEED` field of the Link Capabilities register appropriately to support the maximum link speed for both PCIe mode as well as ESM mode.

Figure 8-2 Initial Speed Change Sequence



8.2.1.2 ESM Equalization (EQ)

- If the PHY is Long Reach Capable, that is, `CCIX_TP_CAP_REG.ESM_REACH_LENGTH =01b`, the controller uses the ESM Extended Timeout value (`ESM_EXT_EQ3_DSP_TIMEOUT/ESM_EXT_EQ2_US-P_TIMEOUT`) in the ESM Control Register as the EQ fine tuning time. For all other values of `ESM_REACH_LENGTH`, the controller uses the PCIe specification defined timeout value as the EQ fine tuning time. After timeout, the controller moves to Recovery.Speed LTSSM state.
- Your application must clear the Quick Equalization Timeout Select (`QUICK_EQ_TIMEOUT`) field in the ESM Control register (`ESM_CNTL_REG`) if the ESM Data Rate1 is achieved after Hot Reset.

8.2.1.3 Quick Equalization (EQ) Redo

Your application can drive quick EQ by performing the following steps (the quick EQ redo has higher priority than the link reach target):

- Ensure that the link is in ESM Data Rate1.
- Probe the ESM Quick Equalization value of the link partner.
- Initiate a Hot Reset.

- Set `QUICK_EQ_TIMEOUT` value in `ESM_CNTL_REG` register.

The link performs a quick EQ redo (if the `QUICK_EQ_TIMEOUT` is not 0) by using the quick EQ timeout (`ESM_QUICK_EQ_TIMEOUT`) value in the EQ phase 2/3 at ESM Data Rate1.

8.2.1.4 Software Equalization (EQ) Redo

Your application can drive EQ redo by using Perform EQ (`PERFORM_EQ`) field of PCIe Link Control 3 (`LINK_CONTROL3_REG`) register.

8.2.1.5 Polling.Compliance Testing

Your application can drive Polling.Compliane by performing the following steps:

- linkup to DL_ACTIVE at PCIe Gen1 rate.
- Set Enter Compliance, Enter Modified Compliance, or Compliance Receive for the link partners, depending on test scenario requirements.
- Set ESMRate0, ESMRate1, Target Link Speed for both the link partners.
- Set Link Reach Target for both the link partners.
- Set ESM Compliance and ESM Enable for both the link partners.
- Sets Hot Reset on DSP.
- Link performs calibration in Detect.Quiet state and then moves to Polling.Compliance state after calibration.
- Link performs compliance test after the speed changes to the highest ESM Rate.
- C lear ESM Compliance and ESM Enable fields and set Target Link Speed to Gen1 rate.
- Drive Polling.Compliance exit and linkup to Gen1 rate.
- Clear all Compliance test related settings in both the link partners.

8.2.1.6 LTSSM Enhancements to Support ESM Operation

- Extended EIEOS format used for ESM Data Rate1 (20.0/25.0 GT/s).
- Control SKP OS format used for ESM Data Rate1 (20.0/25.0 GT/s) and ESM Data Rate0 (16.0 GT/s).
- PCI Express 8.0 GT/s LTSSM rules reused for ESM Data Rate0.
- PCI Express 16.0 GT/s LTSSM rules reused for ESM Data Rate1.

8.2.2 PIPE Interface

The following topics are discussed in this section:

- "PIPE Interface Enhancements to Support ESM Operation"
- "Calibration in P1/L1 state" on page 561
- "Calibration in Detect.Quiet State for Polling.Compliance Testing" on page 562
- "PIPE Message Bus Enhancements to Support ESM Data Rate" on page 562

8.2.2.1 PIPE Interface Enhancements to Support ESM Operation

This section describes *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1* enhancements for supporting ESM operation.

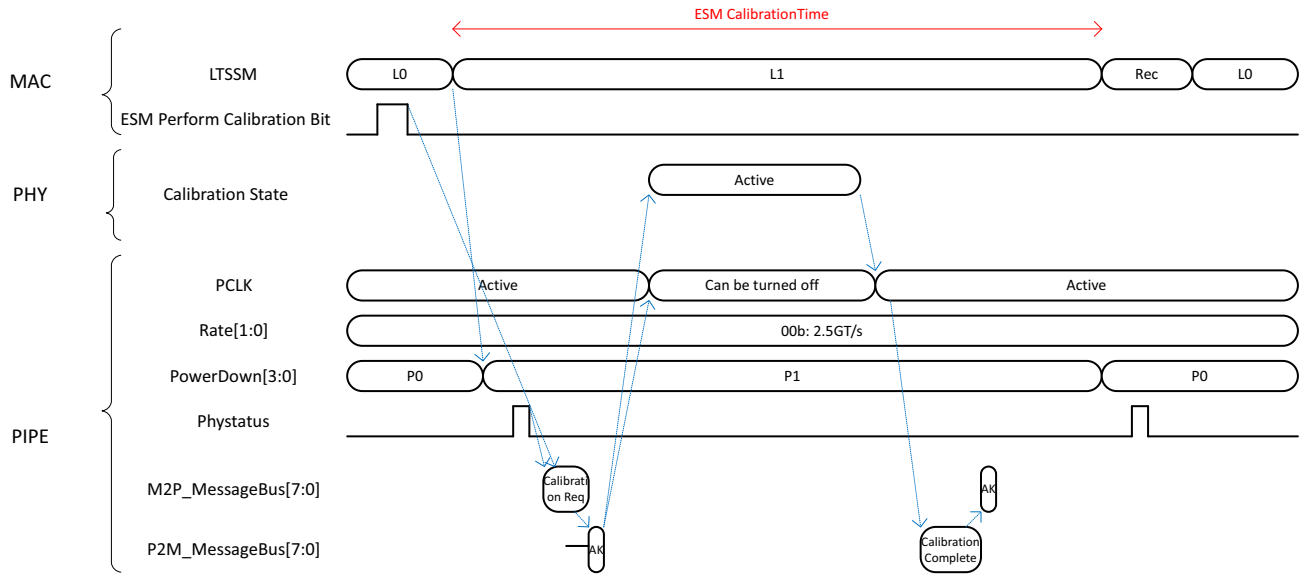
- The controller provides ESM Data Rate information through PIPE message bus.
  - When the value of ESM Data Rate0/1 (`ESM_DATA_RATE0/ESM_DATA_RATE1`) or ESM Enable (`ESM_ENABLE`) fields in the `ESM_CNTL_REG` register changes, the controller generates a write transaction to the PHY to update this information.
  - For more information, see "PIPE Message Bus Enhancements to Support ESM Data Rate" on page 562.
- Calibration handshake in L1 state is done through PIPE message bus. For more information, see "Calibration in P1/L1 state" on page 561.
- Link Reach Target can be Long Reach (LR) or Short Reach (SR).
- PCIe rate change protocol is used for ESM Operation.

8.2.2.2 Calibration in P1/L1 state

Your application can drive Calibration in L1 state by performing the following steps.

1. Disable ASPM and L1 substates on both USP and DSP when link enters Gen1 rate.
2. Read the maximum of ESM Calibration Time values from CCIX Transport Capabilities Register on DSP and USP as the upper limit time for calibration.
3. Set ESM Perform Calibration to initiate calibration.
4. Drive the Link to L1 state using L1-PM mechanism.
5. The controller generates write transaction to set Calibration Request field through PIPE message bus
6. The PHY generates Write Ack command to the controller.
7. The PHY optionally performs calibration for upcoming ESM Data rates.
8. The PHY can turn off PCLK after Step 6.
9. PCLK must be turned on only after the PHY completes the calibration.
10. The PHY generates write transaction to set Calibration Complete through PIPE message bus. If PHY does not need calibration, your application must set Calibration Complete for the PIPE.
11. The controller generates Write Ack command to the PHY.
12. Your application drives the link to exit L1 using L1-PM mechanism after ESM Calibration timeout.

Figure 8-3 Calibration in P1 State



8.2.2.3 Calibration in Detect.Quiet State for Polling.Compliance Testing

1. Initial linkup to DL_Active at PCIe Gen1 rate
2. SW sets ESMRate0, ESMRate1, ESM Compliance, and ESM Enable in DSP and USP
3. SW sets Hot Reset in DSP and the link moves to Detect.Quiet state
4. After Hot Reset and the port is in P1 powerdown mode, MAC writes Calibration Request to PHY through Messgage Bus and PHY performs calibration
5. PHY writes Calibration Complete back to MAC through Message Bus after the calibration completion
   a. If PHY does not write Calibration Complete back to MAC after ESM Calibration Timeout, the MAC exits the calibration autonomously and clears the PHY's calibration bit through Message Bus
6. MAC drives LTSSM transition to the next Detect.Active state

8.2.2.4 PIPE Message Bus Enhancements to Support ESM Data Rate

Table 8-6 describes the PHY register enhancements and Table 8-7 describes the MAC register enhancements to support ESM Data Rate.

Table 8-6 PIPE Message BUS: PHY Registers

Register Address	Register Name	Bit Field	Bit Field Name	Attribute	Description	Allowed Command
0xF00	ESM Rate0	[7]	Reserved	N/A		WC
		[6:0]	ESM Data Rate0	Level	Rate for ESM operation for 1st ESM rate	

Register Address	Register Name	Bit Field	Bit Field Name	Attribute	Description	Allowed Command
0xF01	ESM Rate1	[7]	Reserved	N/A		WC
		[6:0]	ESM Data Rate1	Level	Rate for ESM operation for 2nd ESM rate	
0xF02	ESM Control	[7:2]	Reserved	N/A		WC
		[1]	ESM Calibration request	1-cycle	Controller request to the PHY to calibrate for ESM	
		[0]	ESM Enable	Level	Enable ESM operation	
0xF03	ESM Link Reach Target	[7:6]	Reserved	N/A		WC
		[0]	ESM Long Reach	Level	ESM Extended EQ2/3 timeout values	

Table 8-7 PIPE Message BUS: MAC Registers

Register Address	Register Name	Bit Field	Bit Field Name	Attribute	Description	Allowed Command
0xF00	ESM Status	[7:1]	Reserved	N/A		WC
		[0]	ESM Calibration Complete	1-cycle	Calibration complete indication	

8.2.3 CCIX Transaction Layer Receiver Operation

The following topics are discussed in this section:

- "Error Detection"
- "Rx VC Arbitration" on page 564

8.2.3.1 Error Detection

Different error detection rules apply to TLPs received on the PCIe VC and on CCIX VC. Traffic Class (TC) label on received TLPs is used to steer the traffic either to PCIe VC or CCIX VC. To switch between PCIe compatible TLP format and CCIX optimized TLP format use `CCIX_EN_OPT_TLP_GEN_RECPT` field of the `CCIX_TL_CNTL_REG` register. CCIX TLPs are processed as per the *CCIX Transport Specification*, good CCIX TLPs are passed to the CCIX TRGT1 interface.

Note

Optimized TLPs with Length[6:0] <2

According to the CCIX specification, all TLPs in CCIX optimized TLP format must have Length [6:0] in the range 02H <= Length [6:0] <= 7FH. The controller behavior is undefined when TLPs in CCIX optimized TLP format with Length[6:0] <2 are received.

8.2.3.2 Rx VC Arbitration

The ordering between the CCIX VC and the PCIe VC is achieved using TRGT1 packet Grant/Halt signals (`radm_grant_tlp_type/trgt1_radm_pkt_halt`).

- `radm_grant_tlp_type` indicates the type of TLP received, CCIX TLP (both the PCIe Compatible TLP/CCIX Optimized TLPs) or PCIe TLP. `trgt1_radm_pkt_halt[cfg_ccix_vc_resource*3+:1]` controls the CCIX VC and PCIe VC arbitration. For more information on the usage of the TRGT1 Packet Grant/Halt signals, see "TRGT1 Packet Grant and Halt" on page 371.
- While the CCIX TLPs are being forwarded to your application through the TRGT1_CCIX interface, the RTRGT1 interface does not forward the TLPs to your application. Similarly, TRGT1_CCIX never forwards the TLPs to your application while TLPs are being forwarded to your application through RTRGT1 interface. Therefore, halting of the TRGT1_CCIX interface by `ccix_radm_halt` or RTRGT1 interface by `trgt1_radm_halt` affects the bandwidth of both the interfaces.
- `radm_grant_tlp_type[cfg_ccix_vc_resource*3+1+:2]` is never asserted when the Non-Posted / Completion buffers on the CCIX VC are in the store-and-forward mode.
- If the AMBA bridge is enabled, `trgt1_radm_pkt_halt[cfg_ccix_vc_resource*3+:1]` must be controlled by your application. The other bits of `trgt1_radm_pkt_halt` must be 0.

8.3 CXS Interface

The following topics are discussed in this section:

- "CXS Overview"
- "CXS Interface Clocking" on page 566
- "CXS Interface Reset" on page 567
- "CXS Transmitter Link Activation/Deactivation" on page 567
- "CXS Receiver Link Activation/Deactivation" on page 569
- "CXS Continuous Delivery Mode" on page 571
- "CXS Power Management" on page 572
- "CXS Data Protection" on page 573
- "CXS Error Handling and Reporting" on page 577
- "CXS Error Full Data (CXSERRORFULLDATA)" on page 580
- "CXS Packet Constraints" on page 580

8.3.1 CXS Overview

The CCIX PCIe controller supports CXS (`CX_CXS_ENABLE =1`) as defined in *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0*. The CCIX PCIe controller can be connected to Coherent Multichip Link (CML) based systems such as ARM CoreLink CMN-600 Coherent Mesh Network using the CXS interface. CMN assembles CCIX messages into CCIX packets that can be sent to/received from the controller through the CXS interface. The CXS interface operation is controlled by the CXS controller.

Features

- Synchronous implementation to PCIe `core_clk`
- Asynchronous implementation with dedicated CXS interface clock (`cxs_clk`)
- Support for 128b, 256b, and 512b PCIe controller datapath widths
- CXS data flit width: 256b, 512b, and 1024b supported
- CXS receiver maximum packets per flit: 2, 3, and 4
- CXS interface data protection: None, Parity, and ECC
- CXS transmitter and receiver data path and RAM protection
- CXS receiver and transmitter continuous/non-continuous mode support
- CXS Transmitter error full packet feature support through Receive queue configuration
- Optimized CCIX TLP support
- Low latency operation
- Low power mode support

Limitations

- CXS error full packet feature is not fully supported
- CXS interface signal replication feature is not supported

Table 8-8 shows the supported PCIe data width and CXS data flit width configurations.

Table 8-8 Supported CXS Data Flit and PCIe Datapath Width Configurations

CXS data flit width	PCIe datapath width		
	128	256	512
256	Yes	Yes	Yes
512	Yes	Yes	Yes
1024	Yes	Yes	Yes

8.3.2 CXS Interface Clocking

CXS controller implements asynchronous (`CX_CXS_SYNCHRONOUS =0`) and synchronous (`CX_CXS_SYNCHRONOUS =1`) clocking architectures.

8.3.2.1 Asynchronous Clocking Architecture (CX_CXS_SYNCHRONOUS =0)

In asynchronous clocking architecture, CXS controller operates in two clock domains.

- CXS interface clock (`cxs_clk`)
- PCIe controller clock (`core_clk`)

The CXS controller considers `cxs_clk` to be completely asynchronous to `core_clk`, and implements the appropriate clock crossing synchronization. For more information, see "Advanced Information: Clock and Data Crossing (CDC)" on page 675.

CXS interface clock frequency in normal operational mode is defined by `CX_CXS_FREQUENCY` parameter, and its default value is defined by the ratio between PCIe controller datapath width and CXS data flit width.

```
CX_CXS_FREQUENCY =(CX_NW*32/CX_CXS_DATAFLITWIDTH) * CX_MAX_CORECLK_FREQ
```

Driving the CXS interface clock at a frequency lower than the default value may affect Continuous mode operation and CXS performance. In this case, if Continuous mode is required either on the receive or transmit datapath, program the CXS receiver and/or transmitter control register (`CXS_RX/TX_CTRL_OFF`) to enable Store-Forward mode. For more information on Continuous mode, see "CXS Continuous Delivery Mode" on page 571.

The upper frequency limit of `cxs_clk` is defined by the synthesis technology library that you use. The maximum and minimum simulation-verified clock ratios are 8:1 and 1:2 respectively. The `cxs_clk` can be changed dynamically when the CXS link is not active, without the need of reset.

8.3.2.2 Synchronous Clocking Architecture (CX_CXS_SYNCHRONOUS =1)

In synchronous clocking architecture, the CXS controller is clocked by `core_clk`, and all clock crossing circuitry is eliminated from the design. The CXS Receiver/Transmitter buffers have synchronous FIFOs.

Note In synchronous clocking architecture, CXS data flit width must match PCIe controller datapath width.

8.3.3 CXS Interface Reset

Based on the selected clocking architecture and configuration, CXS controller operates either in single or two clock domains: `core_clk` and `cxs_clk`. Each of these clocks has a corresponding asynchronous low reset signal: `core_rst_n` and `cxs_rst_n`.

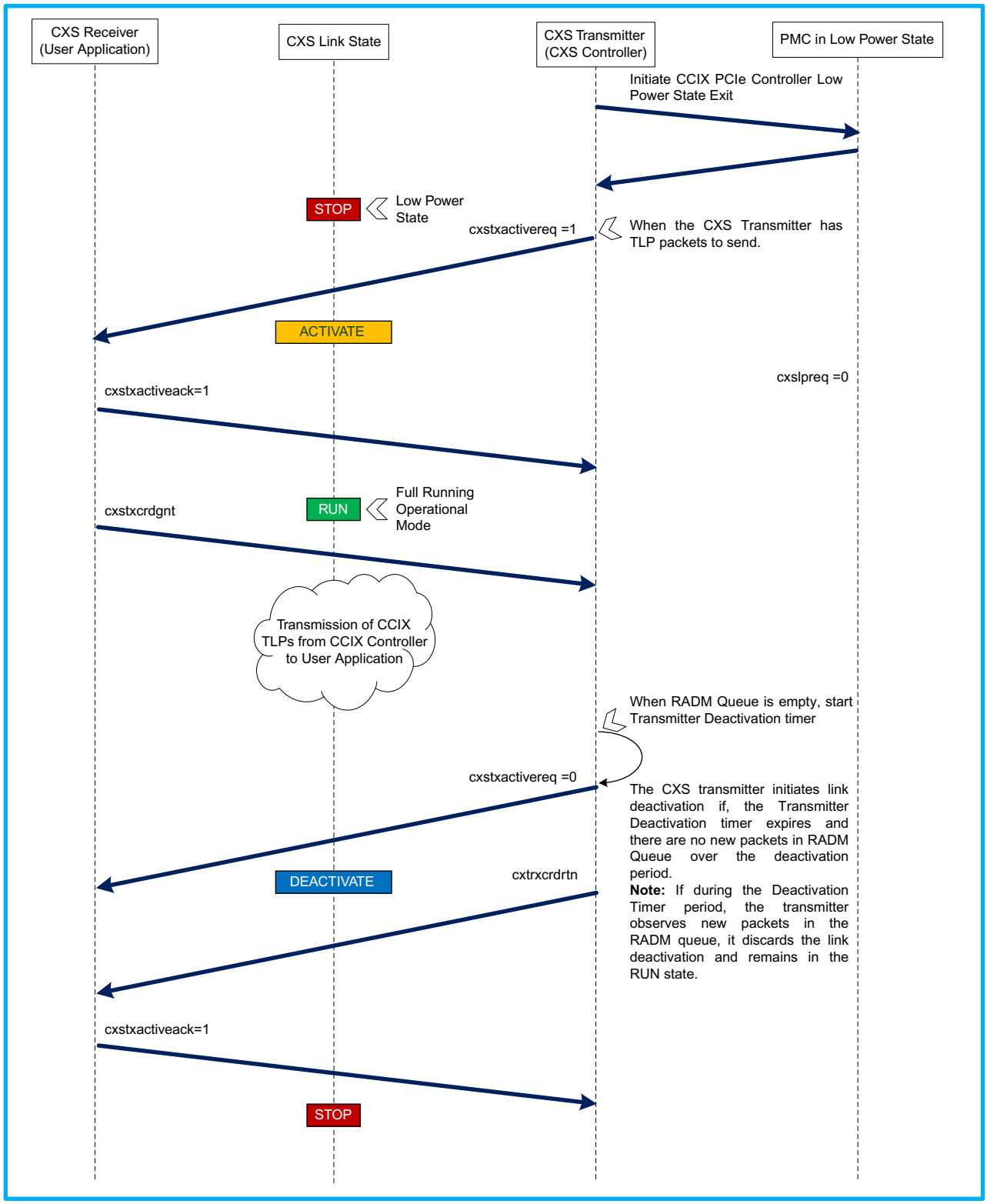
Each reset signal asserted asynchronously by your application must be de-asserted synchronously relative to its corresponding clock. Only one reset cycle is required to reset the logic associated with the corresponding clock. For more information, see "Reset Requirements" on page 90.

8.3.4 CXS Transmitter Link Activation/Deactivation

Figure 8-4 describes a CXS Transmitter Link Activation - TLP Transfer - Link Deactivation scenario where,

- Two credits are granted by the CXS Transmitter
- Transmitter Deactivation Timer is not set to all 1's
- Power Management Controller (PMC) is in low power mode

Figure 8-4 CXS Transmitter Link Activation - CCIX TLP Transfer - Link Deactivation



8.3.4.1 Transmitter Deactivation Timer

If the Transmitter Deactivation Timer is programmed with all 1's, once activated, the link remains active until it is requested to switch from the running operational to low power state. A state change can be requested either by Power Management Controller (PMC) or by User Application's CXS Receiver by asserting the deactivation hint (`cxstxdeacthint`). Irrespective of the Deactivation Timer value, if a state change is requested then the Deactivation Timer value is ignored and the state change occurs immediately.

8.3.4.2 Deactivation Hint

CXS Transmitter implements `cxstxdeacthint` assertion to influence the link deactivation in the following way:

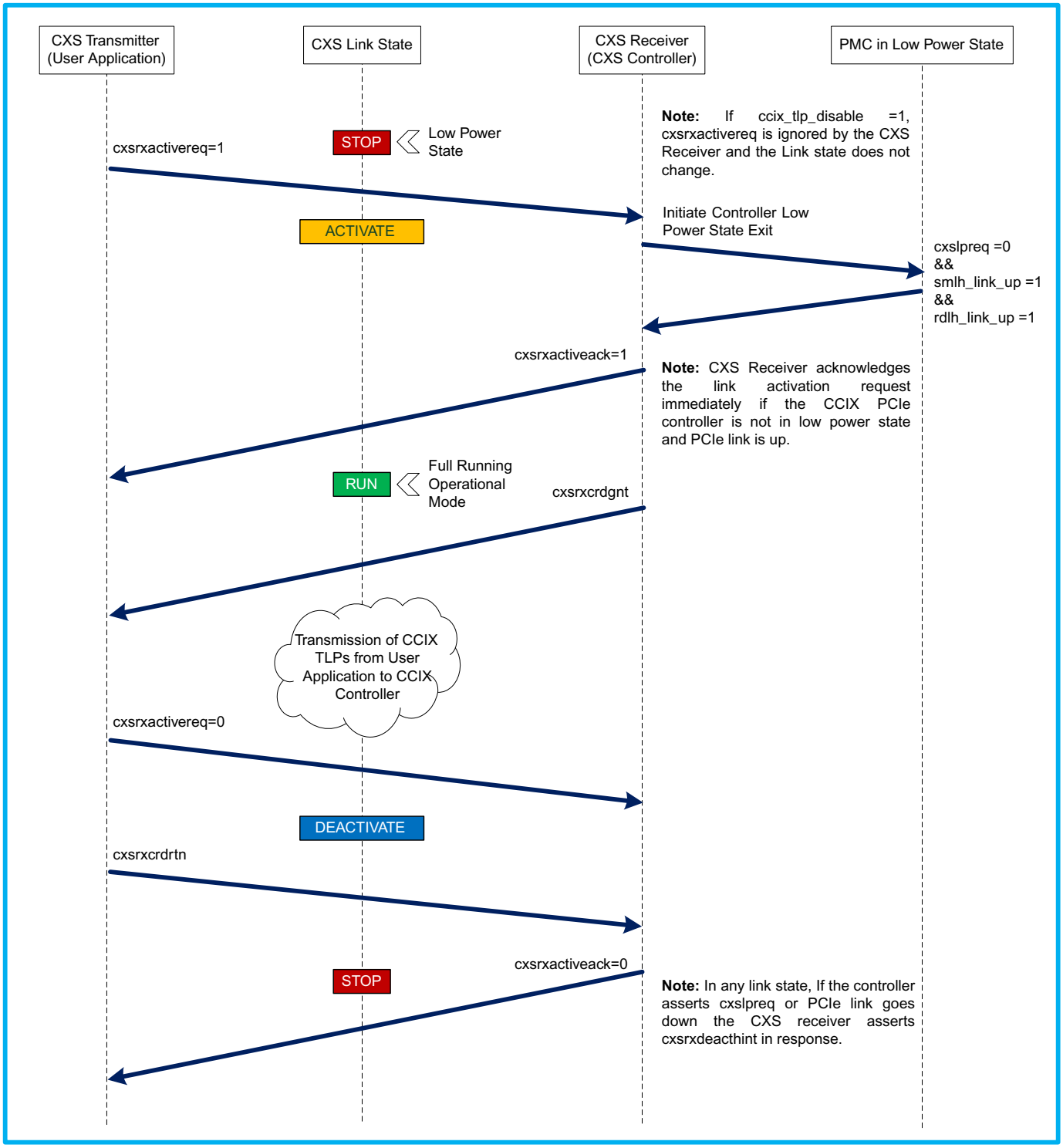
1. It blocks the RADM CCIX queue tracker to stop offloading any new TLP packets into the transmitter buffer
2. Once the offloaded TLP packets are sent over CXS Interface and therefore the transmitter buffer is empty the transmitter de-asserts `cxstxactivereq` and moves to the DEACTIVATE state starting CXS link deactivation.

8.3.5 CXS Receiver Link Activation/Deactivation

Figure 8-5 describes a CXS Receiver Link Activation - TLP Transfer - Link Deactivation scenario where,

- Two credits are granted by the CXS Receiver
- `ccix_tlp_disable` =0
- Power Management Controller (PMC) is in low power mode

Figure 8-5 CXS Receiver Link Activation - CCIX TLP Transfer - Link Deactivation

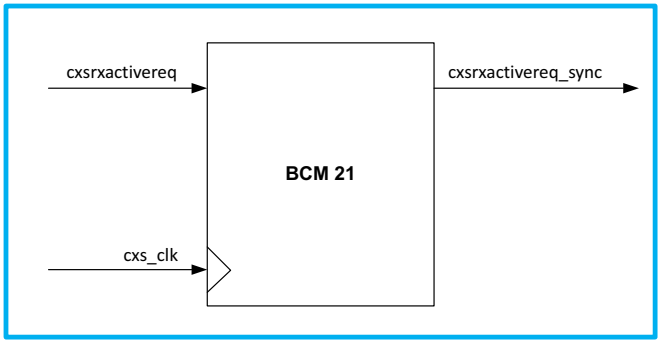


8.3.5.1 Activation Request Synchronization

The CXS link activation is initiated by the transmitter asserting `cxsrxactivereq`. According to the CXS specification, the `cxsrxactivereq` signal may have a purely combinational path (no flip-flops) between transmitter and receiver, and may be a multi-cycle path due to the distance and required frequency of the interface. Therefore, it should be treated by the receiver as an asynchronous signal to the CXS interface clock (`cxs_clk`) and run through appropriate synchronization logic to avoid metastability before use.

BCM 21 module, clocked by the CXS interface clock (`cxs_clk`), is used to synchronize `cxsrxactivereq`. The number of metastability flip-flops used in BCM 21 is configurable by adjusting `CX_CXS_SYNC_STAGES` configuration parameter.

Figure 8-6 Synchronization of cxsrxactivereq



8.3.5.2 Deactivation Hint

When the Power Management Controller (PMC) asserts `cxslpreq`, the CXS receiver asserts `cxsrxdeacthint` to indicate to the transmitter that it is transitioning to a lower power state and therefore the CXS link must be deactivated. In this scenario, all the packets sent between the link deactivation hint and CXS link deactivation are processed normally by PCIe controller.

The same scenario occurs when hot reset is applied or PCIe link goes down, the CXS Receiver asserts `cxsrxdeacthint` and waits for the Transmitter to start CXS link deactivation, that is, de-assert activation request signal and return all the credits, before resetting the controller. In this scenario, all the packets sent between the link deactivation hint and CXS link deactivation are flashed within PCIe controller.

CXS Receiver can change the value of `cxsrxdeacthint` at any time regardless of the CXS link state, and it is required that the transmitter must never ignore deactivation hint assertion and must initiate CXS link deactivation in response in a relatively short period of time.

8.3.6 CXS Continuous Delivery Mode

The *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0* defines CXSCONTINUOUSDATA mode attribute for the Receiver and Transmitter. You can configure the CXS Transmitter and Receiver to operate in continuous mode using `CX_CXS_TX_CONTINUOUSDATA/CX_CXS_RX_CONTINUOUSDATA` parameters.

8.3.6.1 Receiver Continuous Delivery Mode Operation

The CXS Receiver continuous mode operation depends on the value of `CXS_RX_STORE_EN` field in the `CXS_RX_CTRL_OFF` register (`CXS_RX_STORE_EN =0`). The default value of `CXS_RX_STORE_EN` is defined by the inversion of `CX_CXS_RX_CONTINUOUSDATA` configuration parameter.

If `CXS_RX_STORE_EN =0`, CXS Receiver operates in continuous mode, provided the peer transmitter device starts a packet and completes in consecutive cycles when enough credits are available. However, if the peer Transmitter device is not able to continuously deliver the full packet, your application can set `CXS_RX_STORE_EN` field in the `CXS_RX_CTRL_OFF` register to '1', to enable the buffering of the complete packet in CXS Receiver FIFO before driving it on the downstream XADM interface.

Note: CXS Receiver Store-Forward mode capability is enabled only if the controller is generated with `CX_CXS_RX_CONTINUOUSDATA =0`. If the controller is generated with `CX_CXS_RX_CONTINUOUSDATA =1`, CXS Receiver always operates in continuous mode irrespective of the value of `CXS_RX_STORE_EN` field in the `CXS_RX_CTRL_OFF` register.

8.3.6.2 Transmitter Continuous Delivery Mode Operation

The CXS Transmitter continuous mode operation depends the value of `CXS_TX_STORE_EN` field in the `CXS_TX_CTRL_OFF` register (`CXS_TX_STORE_EN =1`). The default value of `CXS_TX_STORE_EN` is defined by `CX_CXS_TX_CONTINUOUSDATA` configuration parameter.

If `CXS_TX_STORE_EN =1`, the CXS Transmitter guarantees that a packet will not be started on the CXS interface until transmitter can deliver all the data of that packet in subsequent cycles with no interruptions if it is given enough credits on the CXS interface, otherwise, if `CXS_TX_STORE_EN =0`, the CXS Transmitter operates in cut-through mode.

8.3.7 CXS Power Management

The CXS interface has a direct impact over the power management, hence the CXS link state is considered by Power Management Controller (PMC) when deciding to enter or exit from the low power state. CXS provides three-signal handshake interface with PCIe PMC: `cxslpreq`, `cxslpack`, and `cxsrxactivereq`. Signals `cxslpreq` and `cxslpack` are generated internally within PCIe controller based on the link state while `cxsrxactivereq` is a standard CXS Interface signal directly routed to PMC.

The following scenarios are considered from power management perspective:

- Low Power mode exit due to assertion of `cxsrxactivereq`

  CXS Receiver is activated when the peer transmitter device asserts activation request. The `cxsrxactivereq` is treated as a wakeup signal by PMC. When cxsrxactivereq is asserted it initiates the exit from the Low Power state to prepare the CXS Receiver for packet reception. If PMC is already in L0 state, it ignores the assertion of `cxsrxactivereq`.

- Low Power mode entry

  If PMC requests to go into Low Power mode, it first observes the status of `cxslpack` signal on CXS output.

  - If `cxslpack` is asserted, it indicates that both CXS Receiver and Transmitter are in idle state and ready to transit to Low Power state.
  - If `cxslpack` is not asserted, PMC asserts `cxslpreq` and waits for both CXS Receiver and Transmitter to acknowledge the request by asserting `cxslpack` before going into the Low Power state.

The assertion of `cxslpreq` initiates CXS link deactivation procedure for both CXS Transmitter and Receiver.

The following sub-sections describe how CXS Transmitter and Receiver react on assertion of Low Power request by PMC.

8.3.7.1 CXS Transmitter Low Power Mode Request Handling

The CXS Transmitter is not expected to directly react to the Low Power mode request assertion from PMC. Before moving to Low Power state, PMC must ensure that Low Power mode conditions for all the other parts of the PCIe controller are met. Once those conditions are met, and there is no ongoing traffic on the wire interface, and hence RADM queue is empty, CXS Transmitter switches to Low Power mode on assertion of `cxslpreq`. In this case, CXS Transmitter moves to DEACTIVATE state, returns all the credits to the peer receiver device, and eventually moves to the IDLE state accepting Low the Power request from the PMC.

Note: PCIe controller must not enter Low Power mode and must remain in L0 state, until both CXS Receiver and Transmitter acknowledge the PMC Low Power mode request.

8.3.7.2 CXS Receiver Low Power Mode Request

In response to CXS receiver low power mode request, the peer transmitter device must de-assert `cxsrxactivereq` once it is ready to acknowledge the link deactivate hint.

Note: According to *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0* the transmitter is not obliged to immediately react on `cxsrxdeacthint` assertion or react on its assertion at all.

In any case, it is expected that after a deterministic period, the peer transmitter device de-asserts `cxsrxactivereq` either reacting to the deactivation hint or running out of packets to be delivered over the interface. Once `cxsrxactivereq` is de-asserted, the Receiver waits for all granted credits to be returned by the transmitter, and thereafter, it eventually transits to STOP state, acknowledging Low Power request from the PMC.

8.3.8 CXS Data Protection

CXS implements following levels of data protection:

- CXS interface data protection
- CXS internal datapath protection
- RADM/XADM interface datapath protection

Figure 8-7 describes the CXS Transmitter/Receiver parity protection architecture. Figure 8-8 describes the CXS Transmitter/Receiver RAM-only protection architecture.

Figure 8-7 CXS Parity Protection (CX_RASDP =2 and CX_CXS_TX/RX_DATACHECK =1)

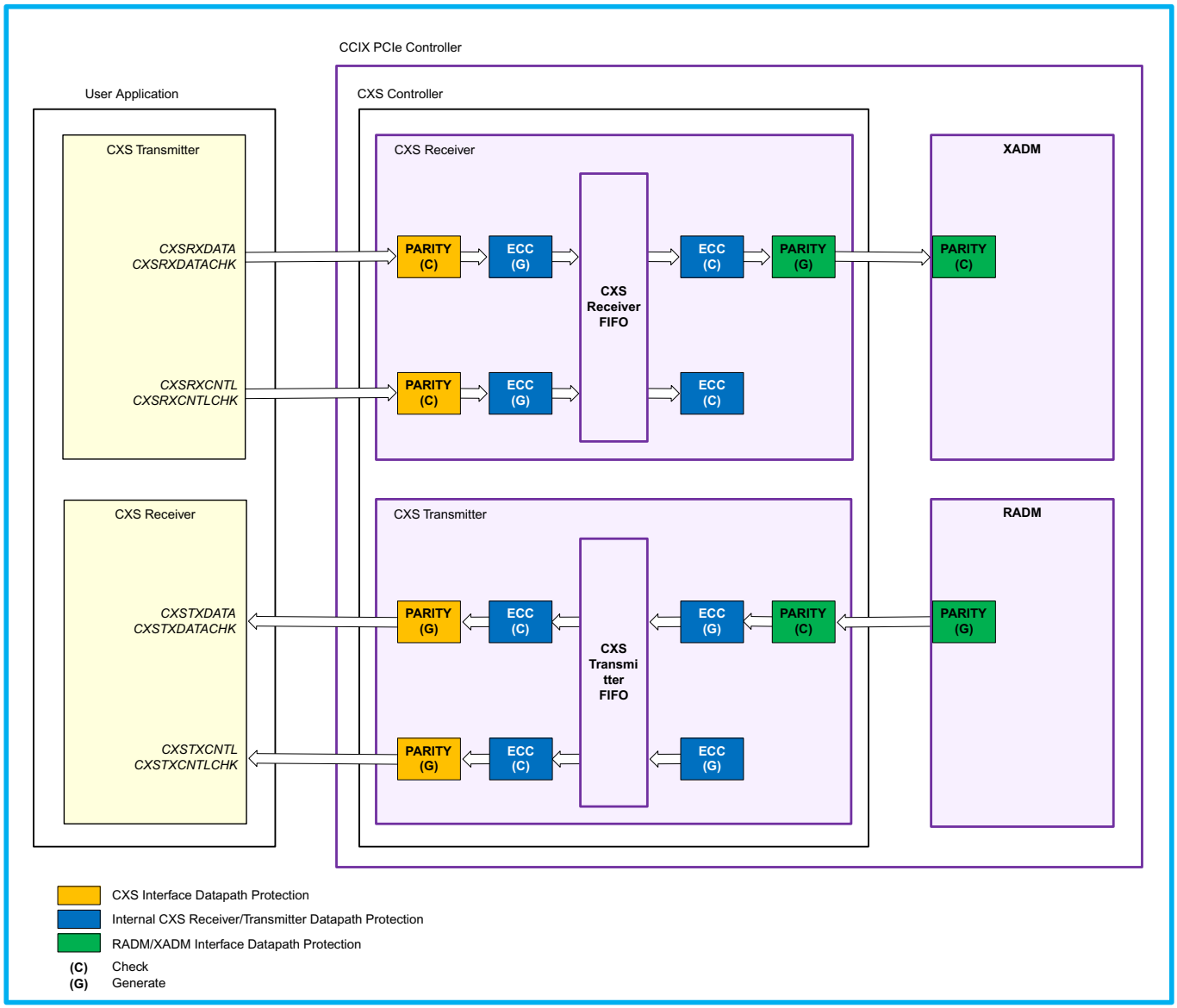
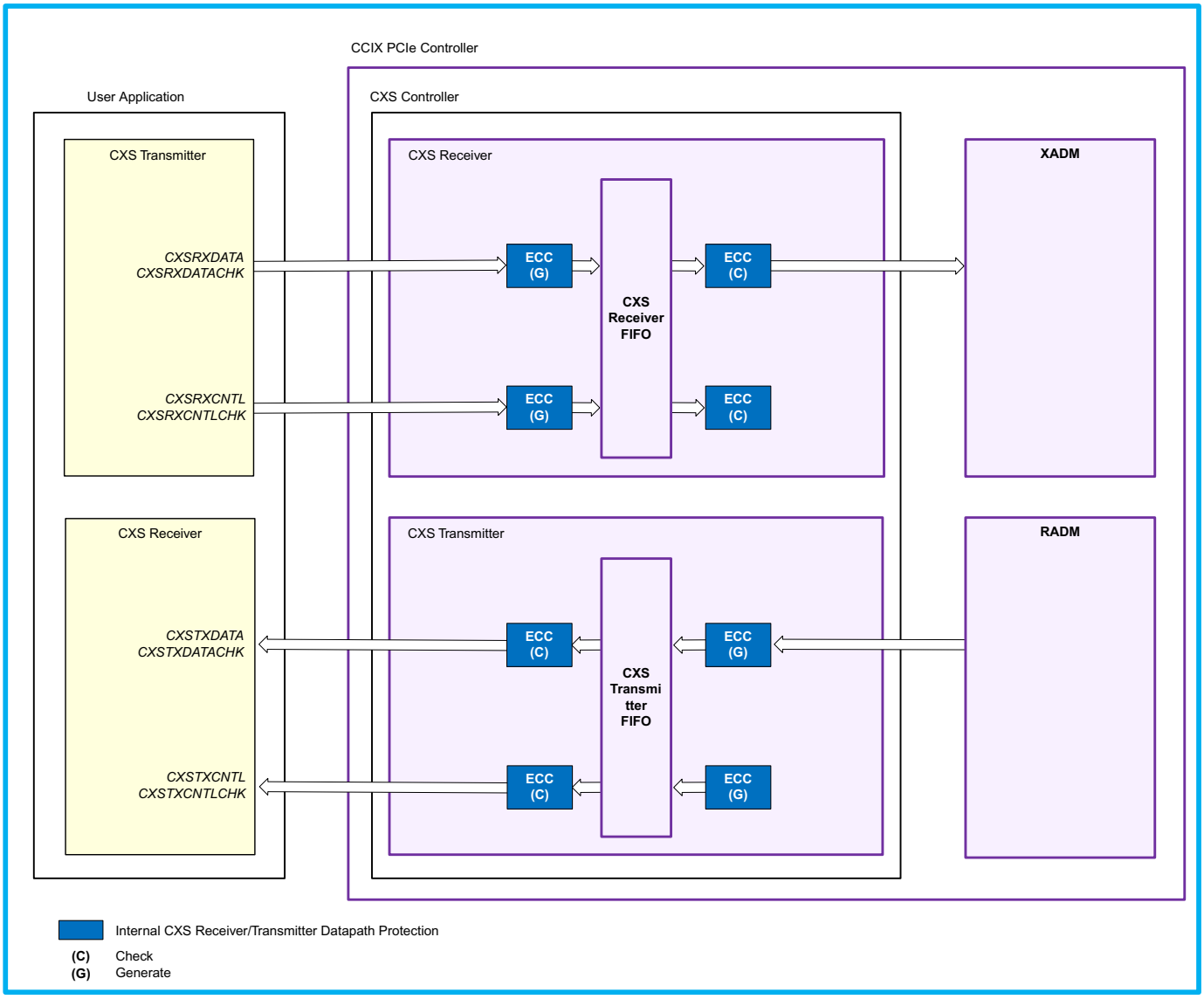


Figure 8-8 CXS RAM-only Protection (CX_RASDP =0 and CX_RASDP_RAM_PROT =1)



8.3.8.1 CXS Data Protection Features and Limitations

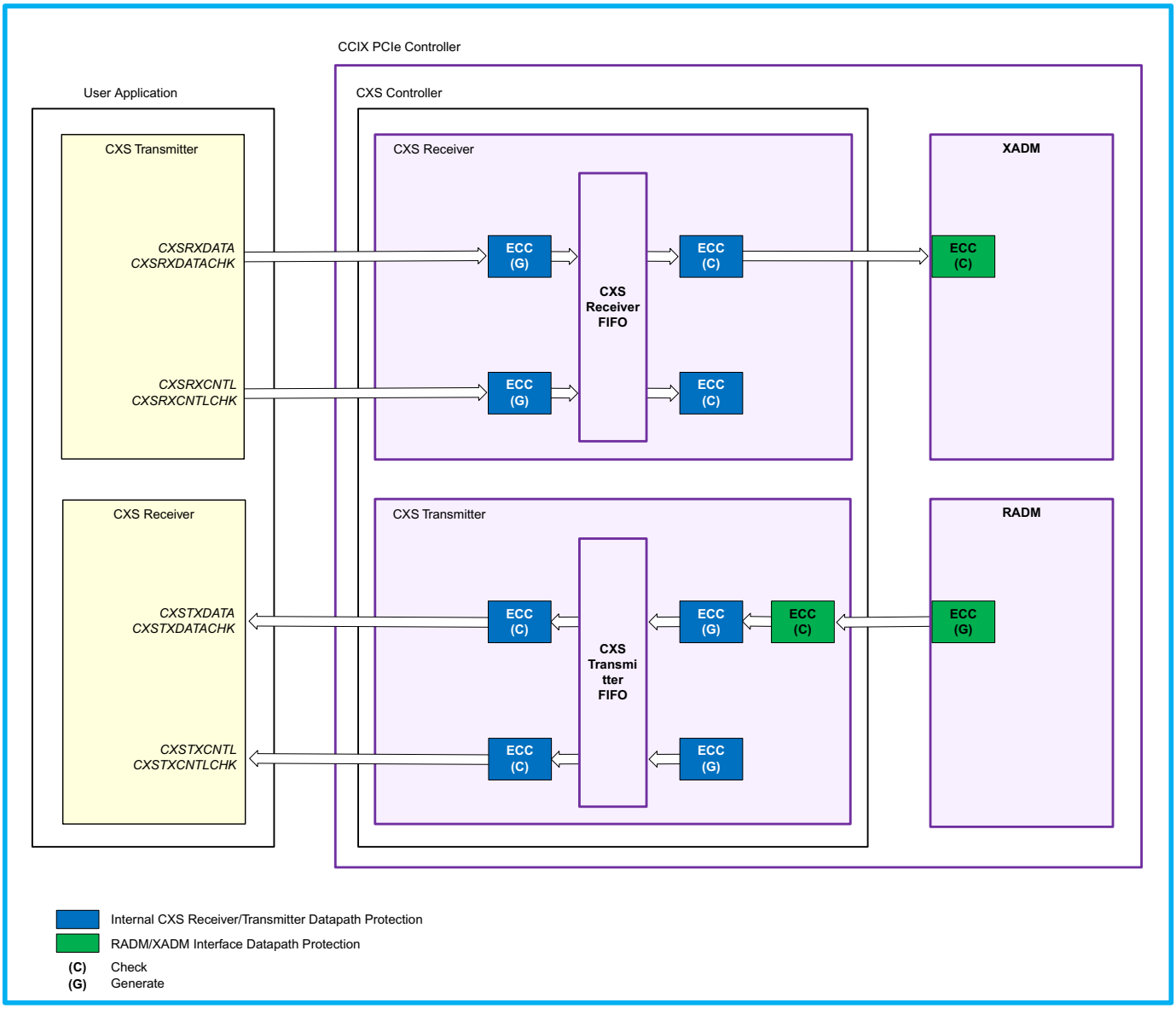
- CXS interface datapath protection
  - Set `CX_CXS_TX_DATACHECK`/`CX_CXS_RX_DATACHECK` >0 to enable CXS Transmitter/Receiver datapath protection.
  - CXS datapath protection level must match the internal datapath protection level defined by `CX_RASDP` setting.
  - Odd Parity or SECDED data check supported.
- Internal CXS Transmitter/Receiver datapath protection

  - CXS Transmitter/Receiver internal datapath and memory protection is enabled if either memory (`CX_RASDP_RAM_PROT`) or datapath (`CX_RASDP_EN`) protection is enabled for CCIX PCIe controller.
  - CXS Transmitter/Receiver internal datapath is protected by ECC code due to the presence of RAM on the datapath which always requires ECC protection.
- RADM/XADM interface datapath protection
  - The input of CXS Transmitter - RADM interface/output of CXS Receiver - XADM interface, is always protected based on the `CX_RASDP` configuration parameter.
  - The default CXS Interface data protection value is derived from the CCIX PCIe controller datapath protection parameter `CX_RASDP`.
  - If CXS interface is enabled, `CX_RASDP` setting is limited to None (0), odd parity (2), or SECDED (3).
- CXS interface control signals replication is not supported in this release.

8.3.8.2 ECC Data Protection

SECDED ECC algorithm is not specified in *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0*, therefore the same ECC algorithm is implemented over CXS Interface as for internal PCIe datapath protection. If ECC protection is enabled on CXS Interface, the same data protection level is applied to PCIe internal datapath to allow data protection code propagation throughout the entire data path and avoid the requirement of protection code re-calculation. Figure 8-9 describes ECC data protection architecture.

Figure 8-9 CXS ECC Protection (CX_RASDP =3 and CX_CXS_TX/RX_DATACHECK =1)



8.3.9 CXS Error Handling and Reporting

The CXS errors can be classified into the following four categories:

- Transport errors between transmitter and receiver on CXS interface

  Information on the CXS signals could be corrupted between the transmitter and receiver. As an example, the corruption could be a bit flipped from an alpha particle or voltage drop. If there is no data protection on the interface signals that are corrupted, this leads to a silent error and can cause unpredictable effects on the receiver or transmitter, including possible corrupt data, performance degradation (if a credit is lost), or the interface hanging.

- Packet errors on CXS interface

  Packet errors may occur on both CXS Receiver and Transmitter side when either a packet marked with an ERROR bit on CXS controls bus is received by CXS Receiver or CXS Transmitter attempts to transmit an errored TLP from RADM queue.
- RAM protection and internal datapath errors
- CXS functional errors

  CXS functional errors occur within CXS device indicating a fault in either Receiver or Transmitter functionality, CXS Interface protocol violations, and more.

Table 8-9 describes the CXS error handling and reporting mechanisms.

Table 8-9 CXS Error Handling and Reporting

Error Type	Cause	Handling	Reporting
Transport Errors Data Protection Enabled (CX_CXS_RX/TX_DATACHECK =1[a])			
Uncorrectable	■ Single bit error on signal protected by parity ■ Duplicated or triplicated signals that are not all identical[b] ■ Double bit error on signal protected by SECDED/ECC	CXS Transmitter	
		■ Not Applicable	
		CXS Receiver	
		Error in CXS data flit ■ Ongoing TLP Nullified	■ PF RAS Datapath Protection Capability Structure (VSEC) Registers
		Error in CXS control bus ■ Ongoing TLP Terminated and nullified ■ All further CXS data flits dropped until an error-free data flit is received starting a new TLP at position 0 within that flit	■ PF RAS Datapath Protection Capability Structure (VSEC) Registers ■ CXS Interface Extended Capability (VSEC) Registers
Correctable	■ Single bit error on signal protected by SECDED/ECC	CXS Transmitter/CXS Receiver	
		■ Error corrected	■ PF RAS Datapath Protection CapabilityStructure (VSEC) Registers
Packet Errors			

Error Type	Cause	Handling	Reporting
Uncorrectable ■ Malformed TLP ■ ECRC error ■ LCRC error	■ CXS Transmitter attempts to transmit an errored TLP	CXS Transmitter	
		■ CXS transmitter asserts error bit ■ Application must drop the received TLP, if either radm_c-cix_tlp_abortor radm_ccix_dll-p_abort is asserted on the RADM interface	■ CXS Interface Extended Capability (VSEC) Registers
	■ CXS Receiver receives a packet marked with an error bit on CXS control bus	CXS Receiver	
		■ CXS Receiver asserts an error flagon the XADM interface	■ Extended Capability (VSEC) Registers
Datapath Errors Data Protection enabled (CX_CXS_RX/TX_DATACHECK =1) && RAM Datapath protection enabled (CX_RASDP_RAM_PROT =1)			
Uncorrectable	■ Error detected in the RADMoutput on the path between RADM and CXS Transmitter, or in CXS transmitter internal data-path	CXS Transmitter	
		■ CXS Transmitter asserts error bit, application must drop the received TLP	■ RAS Datapath Protection CapabilityStructure (VSEC) Registers
	■ Error detected in theCXS Receiver Buffer RAM output on the path between CXS Receiver andXADM, or in CXS Receiver internal data-path	CXS Receiver	
		■ Ongoing TLP Nullified	■ RAS Datapath Protection CapabilityStructure (VSEC) Registers
Functional Error			
Buffer Overflow	■ Overflow of CXS Trans-mitteror Receiver Buffer FIFOs	CXS Transmitter/CXS Receiver	
		■ Unpredictable effect on the transmitter or receive	■ CXS Interface Extended Capability (VSEC)Registers
Credit Overflow	■ Overflow of Transmitter credit counter when more than 15 credits are granted to the CXS Transmitter ■ Overflow of Receiver credit counter when the CXS Receiver gets back more than granted credits	CXS Transmitter/CXS Receiver	
		■ Unpredictable effect on the transmitter or receiver	■ CXS Interface Extended Capability (VSEC)Registers

a. If CX_CXS_[RX|TX]_DATACHECK =0, the CXS receiver/transmitter behavior is unpredictable, including data corruption, performance degradation (if a credit is lost), or the interface hanging.
b. CXS control signals duplication and triplication is not supported in the current release.

8.3.10 CXS Error Full Data (CXSERRORFULLDATA)

CXS Receiver does not support CXSERRORFULLDATA =False attribute as defined in the *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0*. CXS Receiver always requires the length of every packet (including packets ending with EndError) to match the packet length specified in the packet header. Therefore, the parameter corresponding to RX CXSERRORFULLDATA, `CX_CXS_RX_ERRORFULLDATA` is a read-only parameter with a default value of '1'.

CXS Transmitter does not directly support CXSERRORFULLDATA =True attribute as defined in the *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0*. CXS Transmitter does not guarantee the packet length to match the one specified in the header. Therefore, the parameter corresponding to TX CXSERRORFULLDATA, `CX_CXS_TX_ERRORFULLDATA` is a read-only parameter with a default value of '0'.

But, CXS Transmitter indirectly supports CXSERRORFULLDATA =True attribute when the non-posted TLP CCIX VC receive queue is in "Store/Fwd" mode (`RADM_NP_QMODE_VC[N] =1`). When the receive queue is in "Store/Fwd" mode, the receive path ensures that all the malformed packets are dropped in the receive queue itself; only the full packets are delivered to CXS Transmitter. So, even if the CXS Transmitter ends the packet with an EndError indication, it sends the number of bytes specified in the packet header before ending the packet.

8.3.11 CXS Packet Constraints

The CCIX PCIe controller transmits and receives packets that meet the following requirements impacted by the CCIX PCIe controller limitations, *CCIX Transport Specification*, and *ARM® CoreLink™ CMN-600 Coherent Mesh Network, Technical Reference Manual, Version r1p0*s:

- Packet size must be at least 3 DWORD (12 bytes).
- Packet size must not exceed 128 DWORD (512 bytes).
- The first byte of a packet must be placed on an aligned 4-DWORD (16-byte) boundary (byte 0, 16, 32, 48, 64, 80, 96, 112). CXS Transmitter supports only 1 packet per flit, hence all packets must be aligned to byte[0].
- Subsequent bytes of the packet must occupy subsequent bytes of the flit.
- Packets can end on any DWORD (4-byte) aligned boundary.
- Once a packet has been started at a byte position in the current flit, that packet must occupy every subsequent byte in that flit until the packet ends or the flit ends.
- If there are remaining bytes in the packet when the flit ends, that packet must start at byte[0] of the next flit and occupy every subsequent byte position until the packet ends or the flit ends.
- When a packet ends within a flit, the remaining bytes in the flit may be unused (not part of any packet).
- Any packet in a flit must begin at the first available 16-byte boundary relative to the start of the flit or the ending of a previous packet.

Table 8-10 illustrates the packet placement and CXS control bus usage rules for 256-bit controller datapath width.

Table 8-10 Packet Placement and CXS Control Bus Usage Rules for 256-bit Controller Datapath Width

Signal	Field	Cycle		
		1	2	3
CXSDATA [7:0]		TLPA [7:0]	TLPB [135:128]	TLPC [7:0]
CXSDATA [15:8]		TLPA [15:8]	TLPB [143:136]	TLPC [15:8]
CXSDATA [23:16]		TLPA [23:16]	TLPB [151:144]	TLPC [23:16]
CXSDATA [31:24]		TLPA [31:24]	TLPB [159:152]	TLPC [31:24]
CXSDATA [39:32]		TLPA [39:32]	TLPB [167:160]	TLPC [39:32]
CXSDATA [47:40]		TLPA [47:40]	TLPB [175:168]	TLPC [47:40]
CXSDATA [55:48]		TLPA [55:48]	TLPB [183:176]	TLPC [55:48]
CXSDATA [63:56]		TLPA [63:56]	TLPB [191:184]	TLPC [63:56]
CXSDATA [71:64]		TLPA [71:64]		TLPC [71:64]
CXSDATA [79:72]		TLPA [79:72]		TLPC [79:72]
CXSDATA [87:80]		TLPA [87:80]		TLPC [87:80]
CXSDATA [95:88]		TLPA [95:88]		TLPC [95:88]
CXSDATA [103:96]				TLPC [103:96]
CXSDATA [111:104]				TLPC [111:104]
CXSDATA [119:112]				TLPC [119:112]
CXSDATA [127:120]				TLPC [127:120]
CXSDATA [135:128]		TLPB [7:0]		TLPC [135:128]
CXSDATA [143:136]		TLPB [15:8]		TLPC [143:136]
CXSDATA [151:144]		TLPB [23:16]		TLPC [151:144]
CXSDATA [159:152]		TLPB [31:24]		TLPC [159:152]
CXSDATA [167:160]		TLPB [39:32]		
CXSDATA [175:168]		TLPB [47:40]		
CXSDATA [183:176]		TLPB [55:48]		
CXSDATA [191:184]		TLPB [63:56]		
CXSDATA [199:192]		TLPB [71:64]		
CXSDATA [207:200]		TLPB [79:72]		
CXSDATA [215:208]		TLPB [87:80]		

Signal	Field	Cycle		
		1	2	3
CXSDATA [223:216]		TLPB [95:88]		
CXSDATA [231:224]		TLPB [103:96]		
CXSDATA [239:232]		TLPB [111:104]		
CXSDATA [247:240]		TLPB [119:112]		
CXSDATA [255:248]		TLPB [127:120]		
CXSCNTL [1:0]	START [1:0]	0x3	0x0	0x1
CXSCNTL [2]	START0PTR [0]	0x0	0x0	0x0
CXSCNTL [3]	START1PTR [0]	0x1	0x0	0x0
CXSCNTL [5:4]	END [1:0]	0x1	0x1	0x1
CXSCNTL [7:6]	ENDERROR [1:0]	0x0	0x0	0x0
CXSCNTL [10:8]	END0PTR [2:0]	0x2	0x1	0x4
CXSCNTL [13:11]	END1PTR [2:0]	0x0	0x0	0x0

9

PCIe for USB4

This section discusses the PCIe controller for USB4. The following topics are discussed:

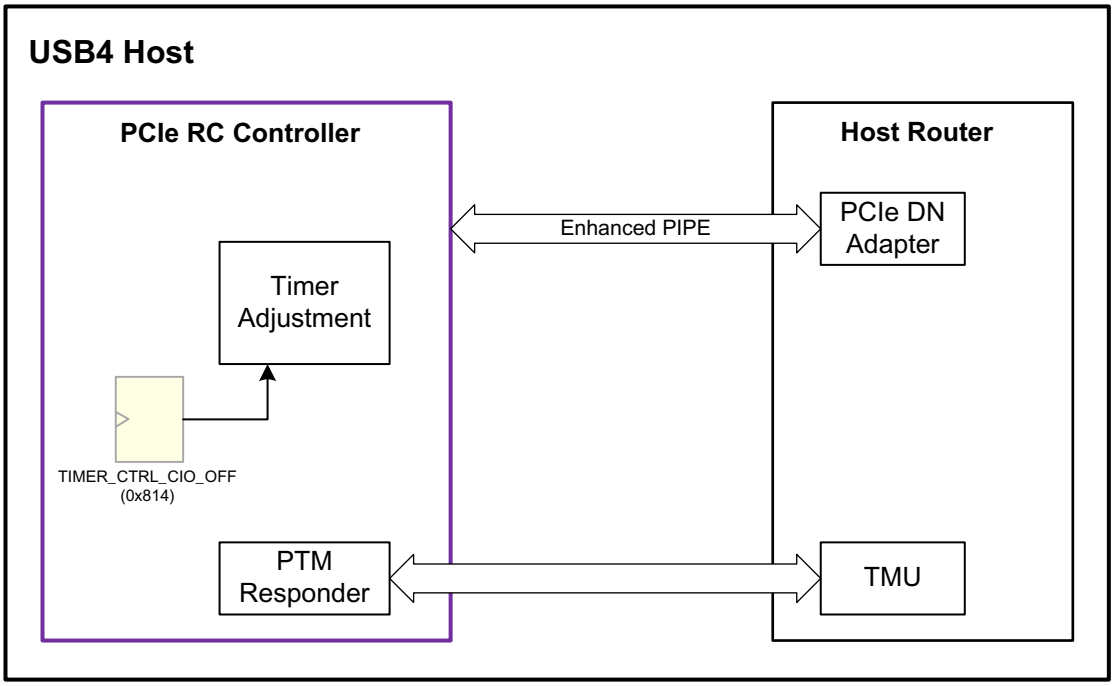
- "PCIe Controller for USB4 Overview" on page 584
- "Precision Time Management (PTM)" on page 587
- "Timer Adjustment" on page 588
- "ASPM L1 Entry" on page 589

9.1 PCIe Controller for USB4 Overview

The PCIe controller for USB4 connects to USB4 PCIe UP/DN adapter through an enhanced PIPE interface. The existing PCIe controller implementation is extended to support USB4 functionality. The changes made to the PCIe controller for USB4 support are as follows:

- PIPE interface extensions for transmitter back-pressure and receiver throttling
- LTSSM changes; removal of states and state arcs
- Programmable timeout control to allow programmable extension of PCIe timers and timeouts
- PTM support for USB4 TMU time conversion

Figure 9-1 USB4 Host: PCIe RootPort and USB4 PCIe DN Adapter Connection



9.1.1 Configuring PCIe Controller for USB4

The relevant configuration parameters are described in Table 9-1.

Table 9-1 Configuration Parameters

Parameter Name	Parameter Label	Value
CC_DEVICE_TYPE	Device Type	EP or RP or DM or SW
CX_PCIE_MODE	PCIe Modes Supported	PCIe over USB4

9.1.2 Features

- The PCIe controller for USB4 runs at the following configuration:

- Max PCIe Speed = GEN1
- Pipe Mode: 8-bit (s-ness 1s) @ 250MHz
- 16 Lanes: 1sx16 Gen1 core, 128-bit datapath
- Max Payload Size: 128-byte
- Lane de-skew disabled
- L1 sub-states disabled
- L0s disabled
- Completion Timeout Ranges enabled

- Optimized Register Configuration Space
  - Some RW PCIe Capability Structure registers are RO in PCIe for USB configuration
  - Some RW Port Logic registers are RO in PCIe for USB configuration
- Programmable modifiers for Layer2 and Layer3 protocol timers
  - Adjustment to larger timeout counts to accommodate USB4 PIPE back-pressure allowed
  - Adjustment to smaller timeout counts to accommodate under-clocking (FPGA) allowed
- Power Management FSM updates to prevent possible deadlock in L1 Entry
- LTSSM changes to align with USB4 specification
  - Current LTSSM state reflected on existing top-level port signals
  - Additional conditions on transition from Recovery.Idle/Config.Idle -> L0 state
    - Reception of STP/SDP as condition for L0 transition added
    - 200us timeout as condition for L0 transition added
    - Maintain reception of LogicalIdles as condition for L0 transition
  - Electrical Idle inference mechanism based on reception of SKP OS removed
  - Setting Scrambling bit in all transmitted TS's in Config.Complete state removed
  - Periodic SKP OS in transmit removed, no expectation of SKP OS in receive
  - Loopback, Polling.Compliance and L0s states removed, all arcs to these states removed
  - Link width negotiation removed (no upsizing/downsizing)
    - Always train to full x16 link only sampling on Lane 0
    - PIPE Rx control signals other than Lane 0 are unused
    - Any condition for "all lanes" is satisfied for Lane 0 only
  - Link speed changes disabled (arcs to Recovery.Speed removed)
  - TS2 Ordered Sets after transition to L0 state are ignored, until a non-TS2 is received (STP/DLLP/other OS)
  - Service of directed transition to Disable LTSSM state if asserted during Configuration.Link-width.Start state delayed
- PIPE extensions
  - New PIPE signals; `mac_phy_txcvalid/phy_mac_txcready`
    - Back pressure the PCIe controller transmit path by asserting `phy_mac_txcready`

- New PIPE signals; `phy_mac_rxcvalid/mac_phy_rxcready`
  - Throttle PCIe controller receive path by de-asserting `phy_mac_rxcvalid`
- Transmitter back-pressure and receiver throttling are independent and are supported in any clock `core_clk` cycle

- Support for TMU A/B parameters in the PTM ResponseD messages
  - Ability to sample TMU USB4 TMU time
  - Encoding/Decoding of TMU_to_PTM_A and TMU_to_PTM_B parameters in PTM ResponseD messages

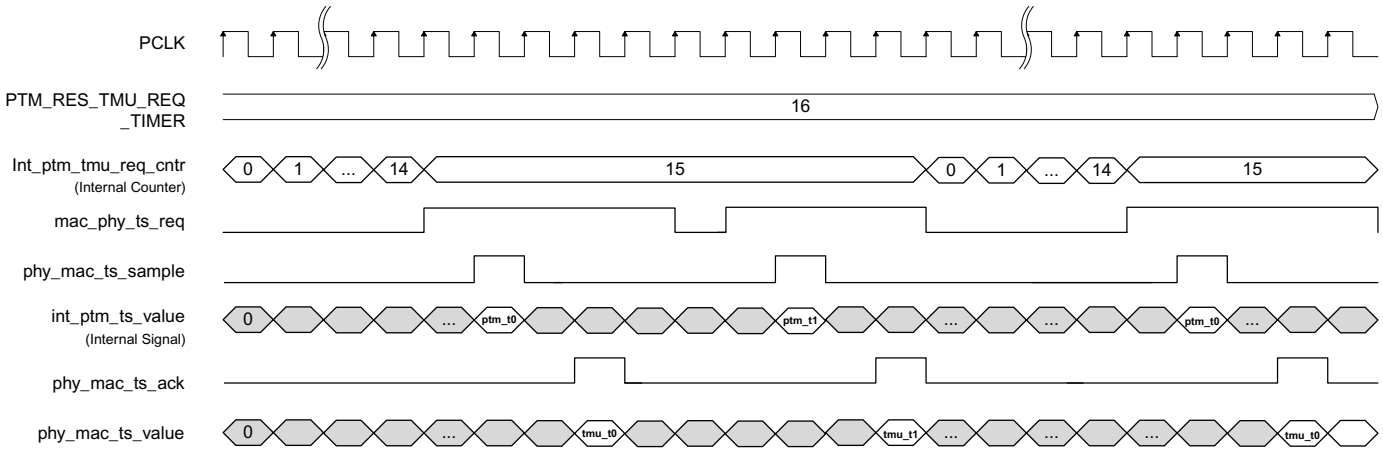
9.2 Precision Time Management (PTM)

Sideband signals between Precision Time Measurement (PTM) module in the controller and Time Measuring Unit (TMU) in the USB4 PCIe Adapter port are used to account for the latencies introduced in the USB4 system. You can switch between the standard PCIe PTM and the USB4 TMU_To_PTM by using `PTM_*_USB4_MODE` field of the `PTM_*_USB4_CIO_CTRL_OFF` register.

PTM Responder (DSP)

Using the sideband signals, the PTM responder (DSP) requests timestamps from the USB4 TMU and samples PTM timestamps as shown in Figure 9-2.

Figure 9-2 Example PTM Responder and USB4 TMU Interaction



The PTM Responder requests timestamps from the USB4 TMU periodically based on the value of `PTM_RES_TMU_REQ_TIMER` field of the `PTM_RES_USB4_CIO_CTRL_OFF` register (configured to 16 in this example). The `ptm_tmu_req_cntr` (internal counter) increments on each clock tick, when it reaches the value configured in `PTM_RES_TMU_REQ_TIMER`, the PTM Responder asserts `mac_phy_ts_req`. The signal `mac_phy_ts_req` remains asserted until the USB4 TMU asserts `phy_mac_ts_sample` and `phy_mac_ts_ack`. The PTM timestamps (ptm_t0, ptm_t1) are qualified by `phy_mac_ts_sample` assertion. The PTM Responder captures successive USB4 TMU samples (tmu_t0, tmu_t1), on each `phy_mac_ts_ack` assertion.

The PTM Responder uses ptm_t0, tmu_t0 and ptm_t1, tmu_t1 to calculate the TMU_to_PTM_A and TMU_to_PTM_B values as follows:

```
TMU_to_PTM_A =(ptm_t1 - ptm_t0)/(tmu_t1 - tmu_t0)
TMU_to_PTM_B =ptm_t1 - (TMU_to_PTM_A * tmu_t1)
```

When `PTM_RES_USB4_MODE =1`, PTM Responder generates PTM ResponseD packet using the calculated TMU_to_PTM_A and TMU_to_PTM_B values.

PTM Requester (USP)

When `PTM_REQ_USB4_MODE =1`, requester (USP) decodes the PTM ResponseD packet to extract TMU_to_PTM_A and TMU_to_PTM_B values. These values are used to reconstruct PTM master time.

9.3 Timer Adjustment

Due to additional latencies incurred when PCIe packets are tunneled in a USB4 system, it may be required to adjust the standard PCIe timeout values. The timer adjustments can also be applied when the PCIe controller for USB4 is over-clocked (that is, running PCLK at 330MHz). Table 9-2 provides a list of PCIe timers that may be adjusted in the PCIe controller for USB4.

Table 9-2 PCIe Controller for USB4 Timer Adjustments

Timer	PCIe Spec Value	Timeout Value for USB4		
		Min	Max	Resolution
LTSSM Timers				Extend timeouts by multiplier factors: x1, x2, x3, x4
L1 Reject Timer	9.5[us]	9.5[us]	9.5[us]+512[us]	8[us]
FC Init1 Timer	32[us]	32[us]	32[us]+120[us]	8[us]
FC Init2 Timer	32[us]	32[us]	32[us]+120[us]	8[us]
Update FC TO Timer	32[us]	32[us]	32[us]+120[us]	8[us]
Retrain Link Timer	248[us]	248[us]	248[us]+1152[us]	128[us]
Ack Latency Timer	192[ns]	192[ns]	192[ns]+256[us]	192[ns] + i where i=2,4,8,16,32,64,128,256[us]
Replay Timer	576[ns]	576[ns]	576[ns]+768[us]	576[ns] + i where i=3,6,12,24,48,96,192,384,768[us]
Update FC Latency Timer	192[ns]	192[ns]	192[ns]+256[us]	192[ns] + i where i=2,4,8,16,32,64,128,256[us]

In the PCIe controller for USB4, the `PHY_CONTROL_OFF` port logic register is re-purposed for timer control (renamed to `TIMER_CTRL_CIO_OFF`). You can use the `TIMER_CTRL_CIO_OFF` register for all timer adjustments.

For an FPGA application, or when the controller is under-clocked at 125MHz or 62.5MHz, all timers in the controller can be run at a faster pace to rate-match the timeout value with the slower clock. The `TIMER_FREQ_MULTIPLIER` field of the `TIMER_CTRL_CIO_OFF` register is used to define the multiplicative factor to run all timers at a faster pace.

9.4 ASPM L1 Entry

In the standard PCIe protocol, the L1 entry negotiation starts when a downstream component transmits a continuous stream of PM_Active_State_Request_L1 DLLPs to an upstream component. If an upstream component is not able to enter L1, it responds with a PM_Active_State_Nak message TLP. The Downstream Port continues to transmit the PM_Active_State_Request_L1 DLLPs until it can respond to PM_Active_State_Nak messages. The upstream component should reject all future PM_Active_State_Request_L1 DLLPs until a 9.5[us] timer expires. This is to ensure that the current negotiation completes before starting a new negotiation.

Due to USB4 back-pressure on the controller's transmit path, the controller may halt some PM_Active_State_Request_L1 DLLPs for a period that exceeds the 9.5[us] timeout. If this happens, the upstream component can interpret a stale PM_Active_State_Request_L1 DLLP as a new negotiation, leading to out of sync LTSSMs.

The controller can be configured to drop a programmable number of PM_Active_State_Request_L1 DLLPs after transmission of the PM_Active_State_Nak message using the `TIMER_CTRL_CIO_OFF` register to prevent the upstream component from receiving stale PM_Active_State_Request_L1 DLLPs.

10

PCIe Integrity and Data Encryption (IDE)

The PCIe controller supports PCIe IDE as defined in the *PCI-SIG ECN Integrity and Data Encyption (IDE) - Revision A*. The PCIe Integrity and Data Encryption (IDE) along with the DesignWare® Integrity and Data Encryption (IDE) Security Module provides confidentiality, integrity, and replay protection for TLPs over PCIe interfaces. You can enable this feature by setting the `CX_IDE_ENABLE` parameter to '1'.

This chapter describes the implementation of the IDE features from the PCIe controller perspective and offers guidelines for the integration with the IDE Security Module.

Each version of the PCIe controller is compatible with only a specific IDE Security Module version. For more information, contact Synopsys support through SolvNetPlus.

This chapter is applicable for the whole PCIe IDE implementation of the PCIe controller and of the IDE Security Module.

The term 'IDE core' refers to the DWC_pcie_ide controller implementing the IDE Security Module, depending on the configuration.

- "PCIe IDE Features and Limitations" on page 592
- "PCIe IDE Architecture Overview" on page 594
- "PCIe IDE Interfaces" on page 601
- "PCIe IDE Error Handling and Reporting" on page 607
- "PCIe and IDE Cores Interoperability" on page 608

10.1 PCIe IDE Features and Limitations

10.1.1 Features

- Separate TX and RX main datapaths to the IDE Security Module to allow encryption of transmitted and decryption of the received TLPs
- Configuration interface to the IDE Security Module to access the IDE Capability Structure from the DBI and the wire.
- Supports ECRC for non-IDE packets
- Supports RAS DP configurations. IDE Security Module reports the Correctable and Uncorrectable errors it detected to the controller.
- Supports discarding of TLPs on the transmit path matching insecure IDE Streams
- Supports IDE disable feature, that allows you to isolate the PCIe controller from the IDE Security Module
- Support for PCRC
- Optional input or output pipelines on IDE related interfaces
- For native configurations (non-AXI and non-DMA), the controller allows to drive the T-bit on a per-TLP basis from the XALI interface.
- Support for Nullified TLPs. For more information on how Nullified TLPs are handled, see the *IDE Security Module Databook*.
- IDE requires the usage of the IDE Prefix. This implies that you must consider to increase the `CX_NPRFX` value by '1' with respect to your non-IDE configuration

10.1.2 Limitations

- The PCIe controller data path width must be greater than or equal to 128 bits
- Malformed IDE TLPs with the Length field mismatching the actual TLP length might result in incorrect Flow Control Credit restoring to the link partner.
- Selective IDE for Configuration Requests is not supported
- CCIX is not supported
- IDE Flow-through is not supported
- For Dual Mode ports and Root ports with Peer-to-peer support (`CX_P2P_ENABLE =1`):
  - IDE traffic is not compatible with peer-to-peer traffic. It is expected that, when IDE is not Disabled, no peer-to-peer packets are either transmitted or received.
- IDE aggregation is not supported
- For 512b configurations (Non-Multistream Architecture) using more than four prefixes (`CX_MULTISTREAM_ARCH=0 && CX_NW=16 && CX_NPRFX > 4`), inbound traffic of IDE TLPs using more than four prefixes might incur in a performance loss between 0% and ~15%, depending on the traffic profile
- The application must implement the DOE Extended Capability Structure on the ELBI interface.
- The IP-XACT view does not support the IDE Capability Structure, as it is implemented outside the PCIe controller, in the IDE Security Module.
- Unexpected IDE Completions that are also Misrouted are handled as Malformed TLPs.

- A Function Level Reset to Function0 affects the IDE Capability Structure, therefore it is expected that IDE Streams are disabled by the application before initiating an FLR to Function0
- For Flit Mode configurations, there are some limitations in the usage of IDE TLPs together with multiple-segments hierarchies. For more information, see the *IDE Security Module Databook*.

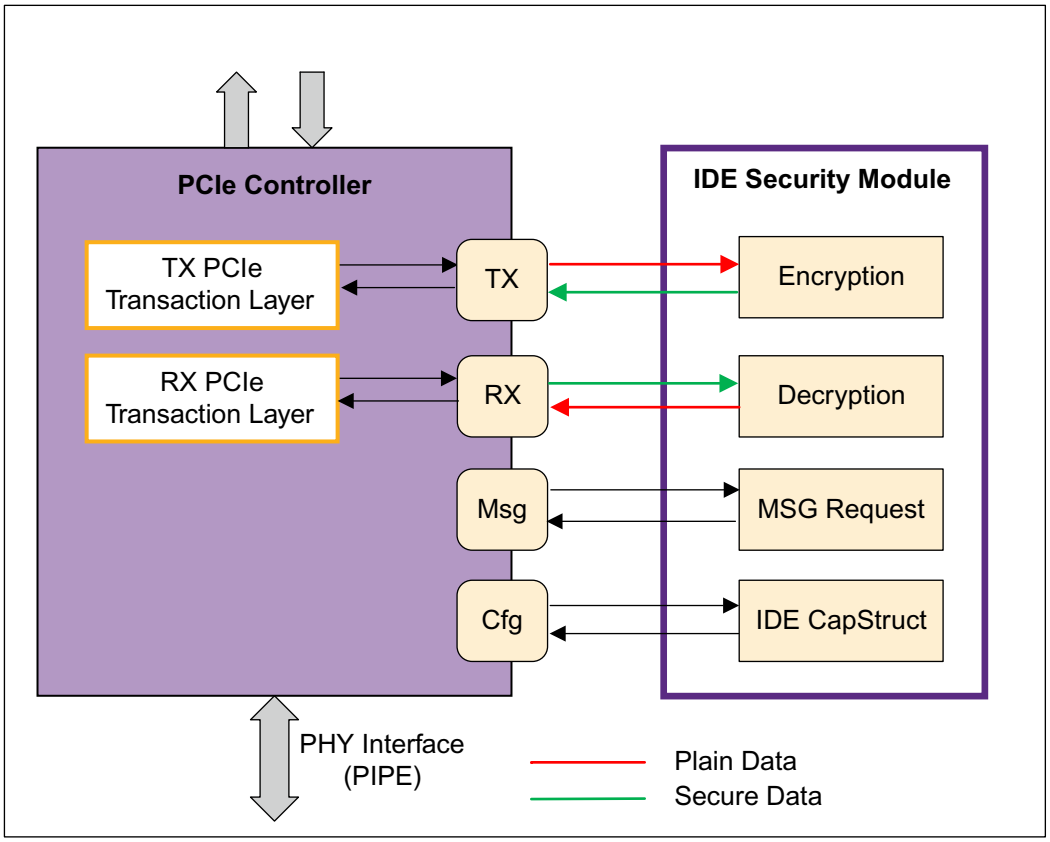
Note The controller assumes that the application monitors the SII signal rdlh_link_up to know when the Link has to exit the DL_Active state and transitions the Streams to Insecure.

Note You can run simulation for PCIe controller and IDE Security Module together using coreConsultant with some limitations. For more information, contact Synopsys support through SolvNetPlus.

10.2 PCIe IDE Architecture Overview

Figure 10-1 shows the overview of dataflow between the PCIe IDE and the IDE Security Module.

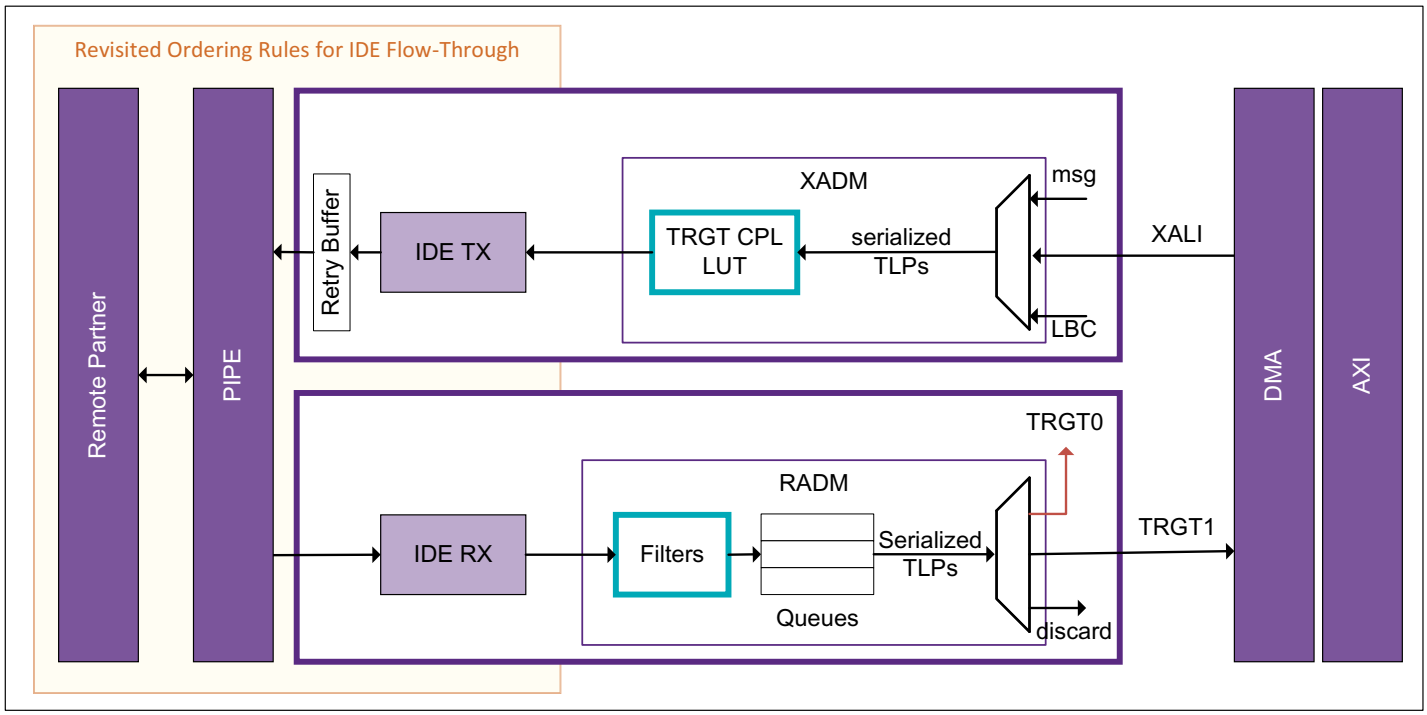
Figure 10-1 PCIe IDE Connected to IDE Security Module



10.2.1 Non-Multistream Architecture (When CX_MULTISTREAM_ARCH =0)

This section describes the Non-Multistream Architecture of the PCIe core when PCIe IDE is enabled for non-Switch ports (`CX_IDE_ENABLE =1 && CX_MULTISTREAM_ARCH =0`).

Figure 10-2 Non-Multistream Architecture (When CX_MULTISTREAM_ARCH =0), with AXI and DMA



10.2.1.1 Transmission

The IDE protocol imposes strict ordering rules within the same Stream. Because the Stream is associated with each TLP by the IDE core itself, it is necessary to know the exact transmission order when the PCIe controller passes the TLPs to the IDE core. Therefore, on the transmit side the IDE core is positioned after the XADM Arbiter. At this point, Flow Control gating has already happened and TLPs from all different sources have already been serialized. This implies that all TLPs, regardless of their source, are encrypted.

In this release the AXI Bridge and the DMA, when present, are not IDE-aware: traffic generated from the AXI Bridge and DMA is handled as any other TLP, that is, serialized and encrypted depending on the Stream matching.

The PCIe controller does not enforce ordering rules on TLPs before the IDE core, as TLPs are considered to be outside of the IDE-protected portion of the link. After encryption, no re-ordering is possible anymore (TLPs are already serialized), so all ordering rules are automatically met.

If the IDE core encrypts a TLP, the MAC and PCRC are appended to it. If the encrypted TLP had the TD bit set, the IDE core clears it before encryption as ECRC is not allowed for IDE TLPs.

In case of a security error, the IDE core flags a TLP to be discarded to the PCIE controller (for example, if hit an Unsecure Stream). In this case the PCIe controller silently discards the TLP before transmission, and the consumed Flow Control credits are restored. For discarded Non-Posted Requests, a Completion Timeout will eventually trigger.

For native PCIe configurations (without the AXI Bridge), it is expected that the application:

- When transmitting Posted and Non-Posted TLPs, PCIe controller never uses the IDE Prefix when driving a XALI Interface (the IDE core adds the IDE Prefix based on the Stream matching). Depending

on the configuration of the IDE core, it might be relevant to drive the T-bit on the XALI interface using the dedicated input port.

Note	The application must never drive more than CX_NPRFX-1 on the XALI interface for Posted and Non-Posted TLPs, to account for the IDE core adding the IDE Prefix.

- When transmitting Completions, the application must prefix the CPL or CPLD with the appropriate IDE Prefix when driving a XALI interface, depending on the received Non-Posted Request (according to the IDE protocol, the Completions must have the same <StreamID, Tbit> of the Non-Posted Requests). The fields that are valid in the IDE Prefix are {Fmt, Type, StreamID, Sub-Stream, Tbit}, IDE core ignores and overwrites the other fields. Depending on the configuration of the IDE core, it might be relevant to drive the T-bit on the XALI interface using the dedicated input port.

Note	The application can drive CX_NPRFX on the XALI interface for Completions, including the IDE Prefix.

Attention	Your application is responsible for tracking inbound Non-Posted Requests and generate the IDE Prefix for outbound Completions.

For AXI bridge configurations, the application must not modify its behavior with respect to a non-IDE configuration. The PCIe controller uses Target Completion LUT to internally store the <StreamID, Tbit> for each received Non-Posted Request and associate them to the corresponding transmitted Completions.

Note	The application must never drive more than CX_NPRFX-1 on the XALI interface for Posted, Non-Posted and Completion TLPs, to account for the IDE core adding the IDE Prefix.

10.2.1.2 Reception

Similarly to the transmit side, PCIe controller performs decryption on the received TLPs in their receiving order. The IDE core is positioned before the RADM Queues, therefore TLPs are passed to it for decryption in the order they have been received from the wire. This also implies that the PCIe controller passes all TLPs to the IDE core, regardless of their destination. TLPs with Data-Link Layer errors that are going to be replayed are passed to the IDE core, but do not affect its internal state.

When the PCIe controller passes an IDE TLP to the IDE core (that is, having the MAC only or the MAC and PCRC), the IDE core decrypts it according to the IDE protocol rules. The PCIe controller strips the MAC and PCRC (if present), therefore they are not passed to the application. The IDE Prefix is not stripped. For more information, see the *IDE Security Module Databook*.

The PCIe controller does not enforce ordering rules on the TLPs after the IDE core, as TLPs after decryption are considered outside of the IDE-protected portion of the link. After decryption, re-ordering and routing to different destinations are possible without causing improper IDE errors.

In this release the AXI Bridge and the DMA, when present, are not IDE-aware: the PCIe controller handles the traffic directed to the AXI Bridge and DMA as any other received TLP.

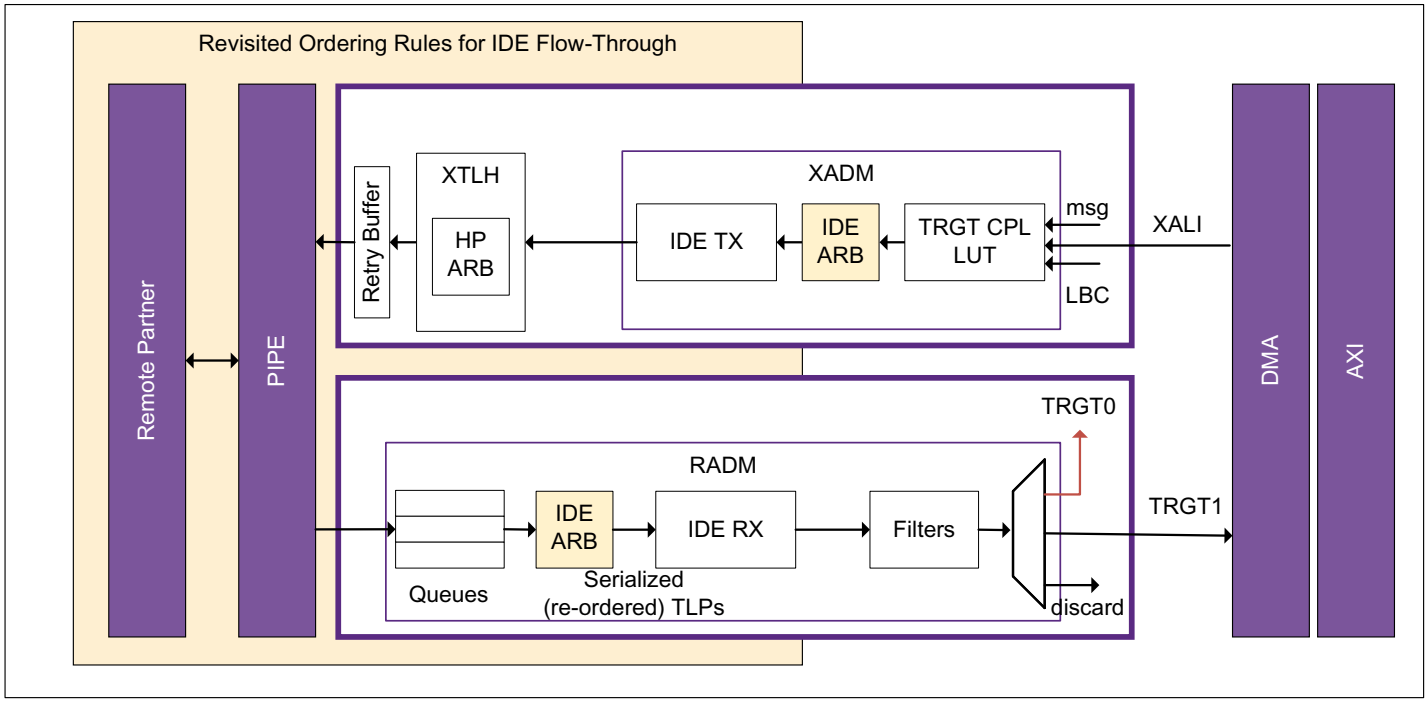
For native PCIe configurations (without AXI Bridge), the application receives the decrypted TLPs with the IDE Prefix as received from the wire and without the MAC and PCRC.

For configurations with the AXI Bridge, the application receives the decrypted TLPs with the IDE Prefix as received from the wire and without the MAC and PCRC, except for Completions, for which the AXI Bridge does not pass the prefixes back to the application. The IDE Prefix is only informative for received TLPs as all processing involving it is internal to the PCIe controller.

10.2.2 Multistream Architecture (When CX_MULTISTREAM_ARCH =1)

This section describes the Multistream Architecture of the PCIe core when PCIe IDE is enabled for non-Switch ports (`CX_IDE_ENABLE =1 && CX_MULTISTREAM_ARCH =1`). Unless specified otherwise, the same rules described for the "Non-Multistream Architecture (When CX_MULTISTREAM_ARCH =0)" on page 594 apply regarding the expected behavior of the application.

Figure 10-3 Multistream Architecture (When CX_MULTISTREAM_ARCH =1) with AXI and DMA



10.2.2.1 Transmission

Similarly to the "Non-Multistream Architecture (When CX_MULTISTREAM_ARCH =0)" on page 594 case, the final transmission order must be known in order to properly encrypt TLPs. The PCIe controller uses an extra IDE Arbiter when IDE is not disabled from `IDE_CTRL_OFF` register. This IDE Arbiter serializes TLPs (N-to-1) across all input interfaces (XALI interfaces, Message Generator and LBC) according to a Round Robin schedule prioritizing MSG over LBC packets and LBC over XALI packets. The PCIe controller uses high-performance arbiter in XTLH when IDE is disabled from the `IDE_CTRL_OFF` register, or for configurations not supporting IDE.

The PCIe controller does not enforce ordering rules on TLPs before the IDE core, as TLPs are considered still outside of the IDE-protected portion of the link. After encryption, no re-ordering is possible anymore (TLPs are already serialized by the extra IDE arbiter), so all ordering rules are automatically met.

For native configurations (without the AXI Bridge), the application is expected to drive on the XALI interface the IDE Prefix with the appropriate {Fmt, Type, StreamID, Tbit} along with Completions (depending on the received Non-Posted Requests) regardless of Flit Mode being enabled or not. The PCIe controller takes care of converting it into the OHC-C in case Flit Mode is enabled. The same rules of "Non-Multistream Architecture (When CX_MULTISTREAM_ARCH =0)" on page 594 applies regarding the number of prefixes that can be driven on the XALI interface.

If Flit Mode is enabled:

- For TLPs requiring an OHC-C because of the RequesterSegment (regardless of IDE):
  - If the TLP matches an IDE Stream, the transmitted OHC-C has all the fields populated and valid.
  - If the TLP does not match an IDE Stream, the transmitted OHC-C has only {RequesterSegment, RSV} fields populated. The Sub-Stream field is set to 4'b0111 and the remaining fields are Reserved.
- For TLPs not requiring an OHC-C because of the RequesterSegment:
  - If the TLP matches an IDE Stream, the transmitted OHC-C has all fields valid and populated if it is a Request, or with {PR_Sent_Counter, Stream_ID, Sub-Stream, K, T} fields populated and RSV and RequesterSegment equal 0 if it is a Completion.
  - If the TLP does not match an IDE Stream, OHC-C is not present.

Attention For native PCIe controller configurations (without AXI), your application is responsible for tracking the inbound Non-Posted Requests to correctly generate the IDE Prefix for outbound Completions.

10.2.2.2 Reception

Differently from the Non-Multistream case, the IDE core is positioned after the RADM Queues.

This implies that the decryption order is not necessarily the order with which TLPs have been received from the wire, and an additional IDE arbiter reads from the RADM Queues and passes TLPs to the IDE core.

This arbiter serializes TLPs (N-to-1) across all RADM Queues interfaces following the IDE Revisited Ordering Rules for Flow-Through IDE Streams as described in the Specification, with the following limitations applying per-Virtual Channel:

- A TLP never passes another TLP belonging to the same FC Type, even if allowed.
- Revisited Ordering Rules are applied between all TLPs (between IDE TLPs belonging to different Streams, between IDE TLPs and non-IDE TLPs and between non-IDE TLPs).

When IDE is disabled from the `IDE_CTRL_OFF` register, or for configurations not supporting IDE, this IDE Arbiter is bypassed and TLPs flow to the TRGT1 interface directly. This is still compliant with the regular PCIe ordering rules. Relaxed order and IDO are not considered by this arbiter.

If Flit Mode is enabled:

- For IDE TLPs, the PCIe controller synthesizes an IDE Prefix with the valid fields {Fmt, Type, Stream_ID, Sub-Stream, K, T}. The other fields are Reserved. If RSV is '1', the PCIe controller send the

RequesterSegment through the dedicated TRGT1 signal. The M and P bits of the IDE Prefix are Reserved and should not be used by the application, as the MAC and PCRC have already been checked and stripped by the IDE core. As for the Non-Multistream case, the IDE Prefix of Completions will not be passed to the Application in case of configurations with the AXI Bridge.

- For Non-IDE TLPs, the PCIe controller does not generate IDE Prefix. If an OHC-C was present in the received TLP because of RSV=1, the PCIe controller passes the Requester Segment to the application through the dedicated TRGT1 signal.

10.2.2.3 IDE Sync and Fail Messages

The IDE core can request the transmission of IDE Sync and IDE Fail messages through the dedicated IDE Message Interface. You must program the IDE core such that the `IDE_SYNC_MSG_REQ_THRESHOLD` field of the `IDE_SYNC_MSG_CFG` register is set to a value:

- Less than or equal to 235 when `CX_MULTISTREAM =0` and `CX_MULTISTREAM_ARCH =0`
- Less than or equal to 225 when `CX_MULTISTREAM =1` or `CX_MULTISTREAM_ARCH =1`

IDE Sync and Fail messages have the highest priority among all messages.

The PCIe controller does not advertise received IDE Sync and Fail messages on the SII interface. The RADM Filtering Rules can be changed to forward them to the TRGT1 interface if needed.

10.2.2.4 IDE Disable

The IDE Disable feature allows you to isolate the PCIe controller from the IDE core.

- When you configure the `IDE_CTRL_DISABLE` field of the `IDE_CTRL_OFF` register to '1', the controller isolates from the IDE Security Module. This includes transmit and receive datapaths, Message Interface, and Configuration Interface. When using the IDE Disable feature, the TLPs on the transmit and receive paths have no extra latency with respect to the controller without enabling the IDE Security Module (`CX_IDE_ENABLE =0`).
  - When `IDE_CTRL_DISABLE =1`, the IDE Capability Structure is not accessible anymore and behaves as non-existing registers. The application must ensure no access to the IDE Capability Structure is performed.
    - The field `IDE_CTRL_DISABLE` is writable only by the DBI. It is the responsibility of the application to ensure that the DBI is secure.
    - The field `IDE_CTRL_DISABLE` must never be toggled during operation, as it might disrupt ongoing TLPs or register accesses.
    - When `IDE_CTRL_DISABLE =1`, the IDE TLPs are not expected to be transmitted or received.

10.2.2.5 T-bit Association

For Native configurations (non-AXI, non-DMA), the IDE core can be programmed to associate the T-bit of outbound TLPs with the T-bit passed by the PCIe controller.

When this is the case:

- The Application can drive the T-bit value of all TLPs transmitted through the XALI interface on a per-TLP basis, including Completions.

- Other outbound TLPs, such as internally generated messages, VMI messages, or MSI-X messages in case of internal MSI-X table, have the T-bit clear. It is Application's responsibility to implement the MSI-X table in the Application space and, more in general, to use the XALI interface to transmit such TLPs if this behavior is not acceptable.

For all (Native and non-Native) configurations, the IDE core can be programmed to generate the T-bit internally, such as:

- For Posted and Non-Posted, the T-bit can be associated on a per-Stream basis
- For Completions, the T-bit can be either:
  - Associated on a per-Stream basis or
    - For AXI configurations, copied from the received Non-Posted Request
    - For non-AXI configurations, indirectly driven by the Application when passing the IDE Prefix along with the Completion.

In all cases (Native or non-Native), internally generated Completions has the T-bit copied from the received Non-Posted Request. If the CIC interface is enabled, the application can overwrite the Tbit of the Completion. You must consider that a wrong usage of this feature can cause the system to not work properly.

10.3 PCIe IDE Interfaces

When the parameter `CX_IDE_ENABLE` is set to ‘1’, the controller sends the TLPs in the main TX and RX datapath to the IDE Security Module for encryption and decryption.

10.3.1 PCIe IDE TX Interface

The controller redirects the TX PCIe TLPs towards the IDE Security Module for encryption. After encryption the packets enter the main datapath of the controller. When the `IDE_CTRL_DISABLE` field of the `IDE_CTRL_OFF` register is set to '1', the TLPs are not passed to the IDE Security Module.

Figure 10-4 shows the timing diagram for the TX Plain data interface between the controller and the IDE Security Module for single cycle packet and a two-cycle packet transmission. The controller uses pcie_ide_tx_dllp_error signal to inform the IDE Security Module of a Nullified packet.

Figure 10-4 PCIe IDE TX Plain Data Interface

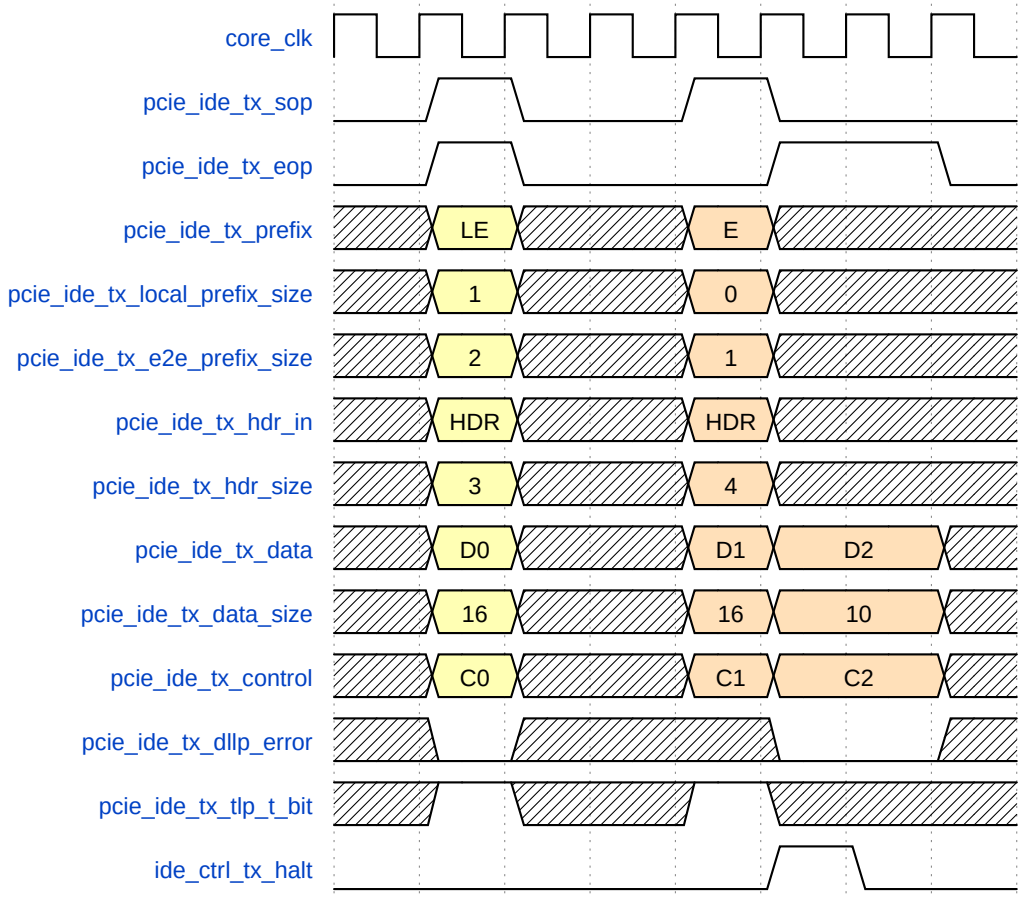
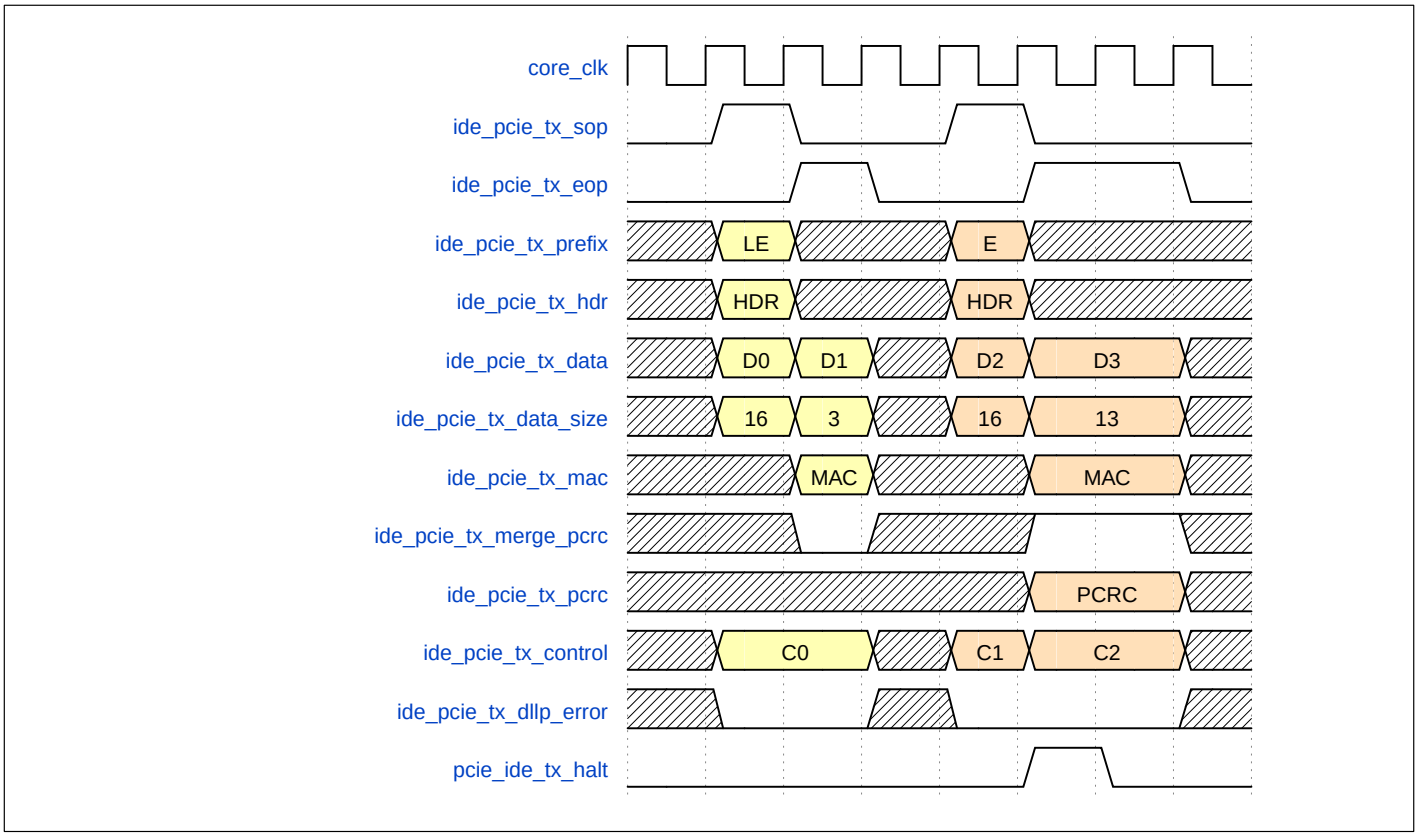


Figure 10-5 shows the timing diagram for the TX Secure data interface between the IDE Security Module and the controller. The timing diagram depicts transmission of two-multi-cycle packets. Both packets have a 3 DW MAC appended. The second packet also includes a PCRC. The IDE Security Module uses ide_pcie_tx_dllp_error signal to inform the controller of a Nullified packet.

Figure 10-5 PCIe IDE TX Secure Data Interface



10.3.2 PCIe IDE RX Interface

The controller redirects the RX PCIe TLPs towards the IDE Security Module for decryption. After decryption the packets return to the main datapath of the controller. When IDE is disabled, the controller does not send the TLPs to the IDE Security Module. No latency is added in this case. For data paths equal to 256b, the controller can send or remove two TLPs to the IDE Security Module.

Figure 10-6 shows the timing diagram for the RX Secure data interface between the controller and the IDE Security Module. The timing diagram depicts the transmission of two-multi-cycle packets to the IDE Security Module for decryption. The controller uses pcie_ide_rx_dllp_error signal to inform the IDE Security Module of a packet with DLLP errors.

The first packet has both its PCRC and MAC passed on the `pcie_ide_rx_pcrc_mac` bus. The second packet has only a MAC passed on the `pcie_ide_rx_mac` bus.

Figure 10-6 RX IDE Security Module Secure Data Interface

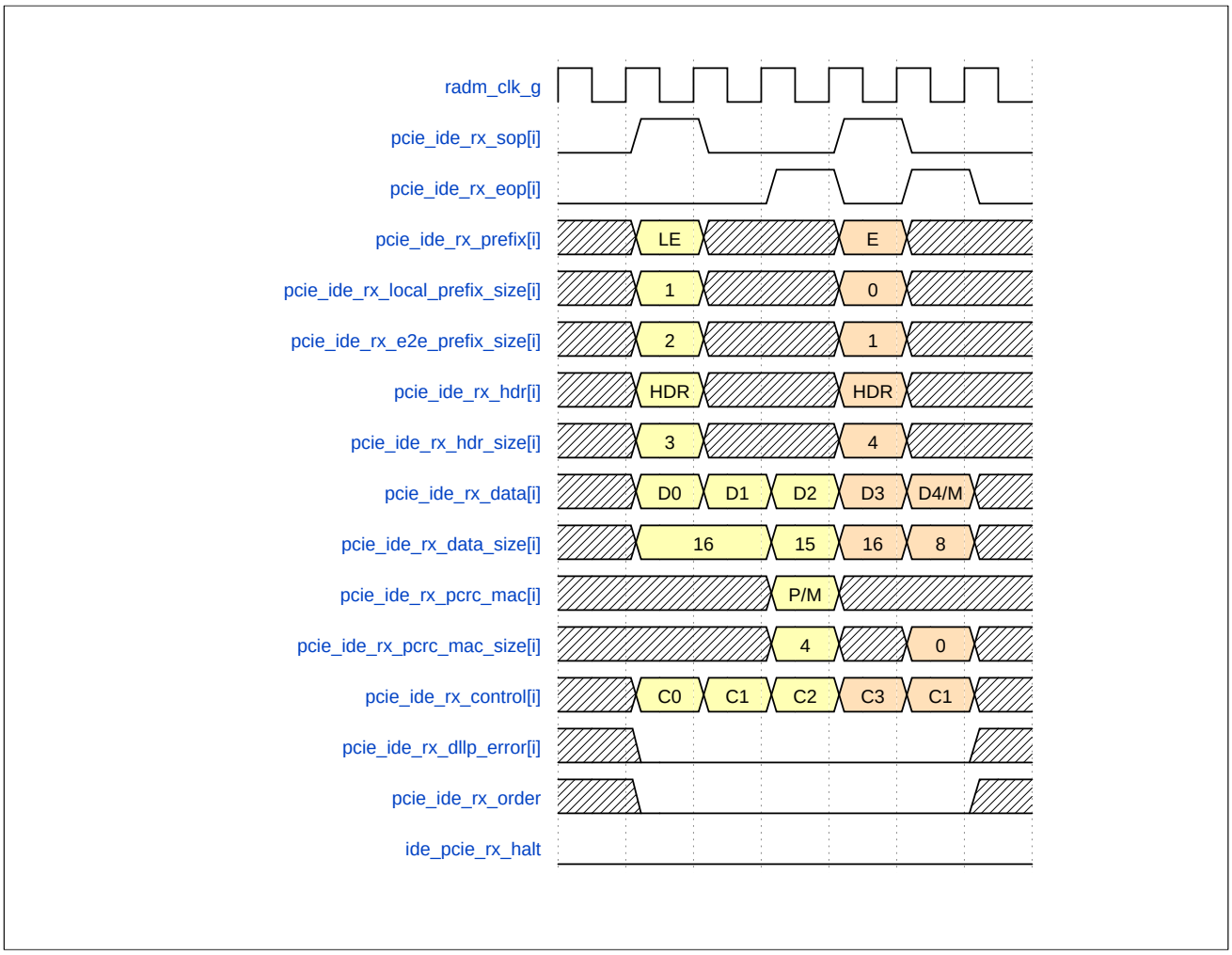
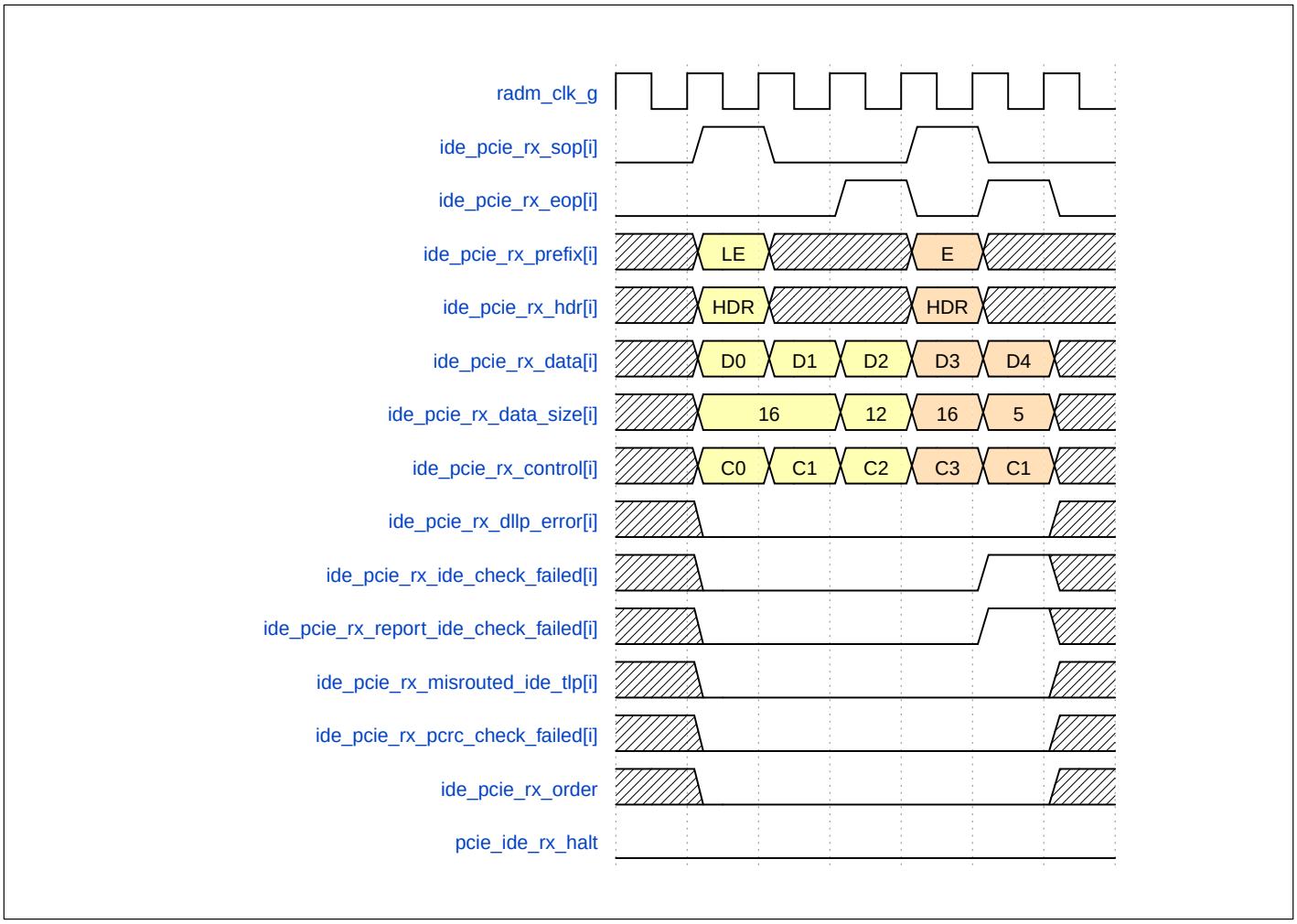


Figure 10-7 shows the timing diagram for the RX Plain data interface between the IDE Security Module and the controller. The timing diagram depicts the reception of two-multi-cycle packets from the IDE Security Module after decryption. These TLPs have their MAC stripped. Second TLP has an IDE MAC Check Fail error.

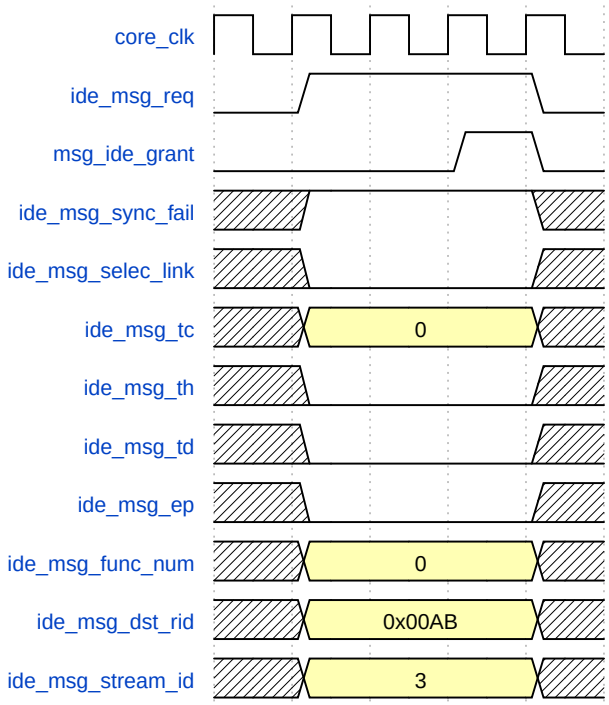
Figure 10-7 PCIe IDE RX Plain Data Interface



10.3.3 PCIe IDE Message Interface

The IDE Security Module uses the IDE Message interface to request the transmission of an IDE Sync or Fail message.

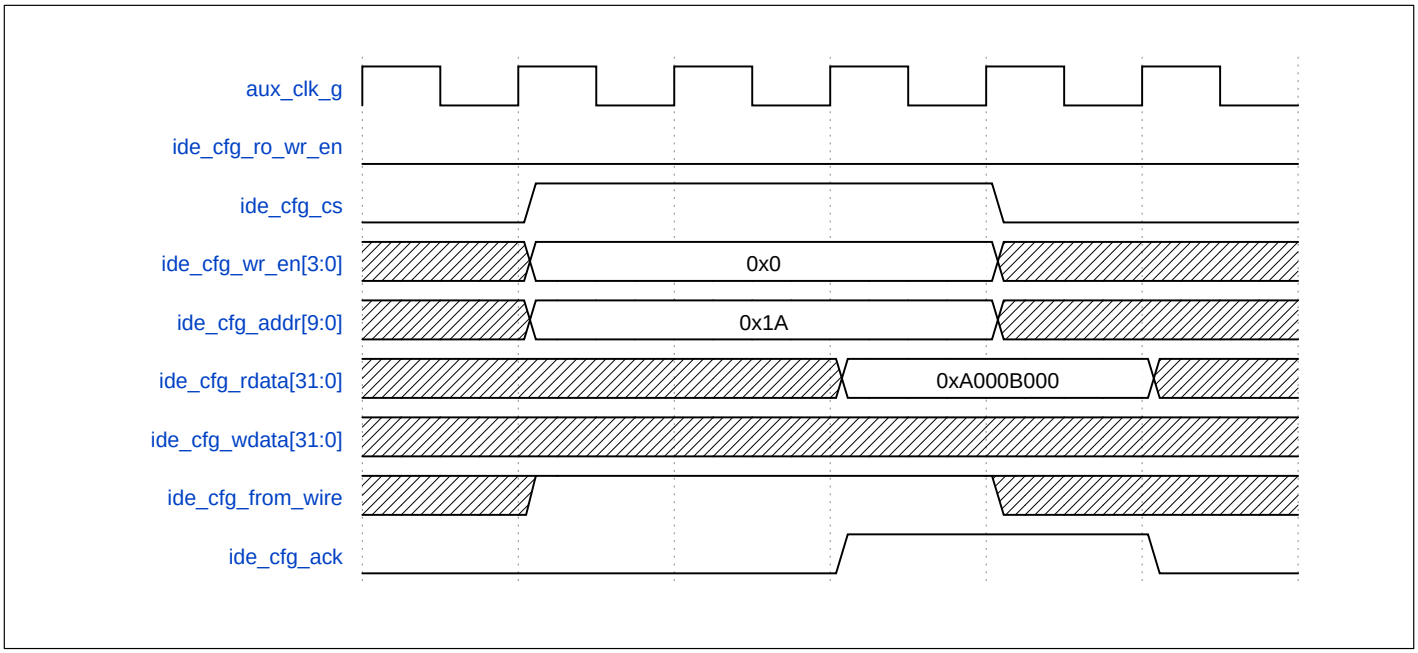
Figure 10-8 IDE Security Module Message Interface Requesting the Transmission of an IDE Message



10.3.4 IDE Security Module Configuration Interface

The controller provides a configuration interface to access the IDE Capability Structure implemented in the IDE Security Module. Although it is implemented outside of the PCIe controller, the IDE Capability Structure resides in the PCIe Configuration Space and is a part of the linked list built by the controller. The controller redirects the DBI and the wire accesses addressing the IDE Capability Structure Configuration space to the IDE Configuration Interface. The interface distinguishes between DBI accesses and wire accesses for security reasons. The controller assumes that the IDE Security Module informs the application of write accesses coming from the wire. It also assumes that the DBI accesses are secure.

Figure 10-9 IDE Security Module Configuration Interface - Example of Read Operation



10.4 PCIe IDE Error Handling and Reporting

The PCIe error handling changes in the following way when the IDE Security Module is enabled:

- TLPs from the IDE Security Module can be associated with three IDE errors, in addition to the existing errors:
  - IDE Check Fail
  - IDE Misrouted
  - IDE PCRC Error

  TLPs with any of the three IDE errors are internally discarded and are not transmitted to your application, after having restored Flow Control credits.

  This includes TLPs with PCRC Error, which are never handled as Advisory Non-Fatal by the controller.

  There are no optional filtering rules to mask these errors and TLPs are always discarded (continuous operation is not supported).
- If RADM queue receives an IDE TLP with an IDE error in cut-through or bypass mode, the TLP can be passed to your application after the controller asserts `radm_trgt1_tlp_abort` or `radm_bypass_tlp_abort` signal.
- The reception of a TLP with an IDE error is signaled, as an optional feature, to the dedicated bits in the `diag_status_bus`. For more information on the `diag_status_bus`, see “Signal Descriptions” chapter in the *Reference Manual*.
- IDE errors are treated as Function specific errors, always associated with Function 0.
- When `APP_RETURN_ERR_EN =1`, the application can force IDE errors using the bits [17-15] of the `app_err_bus`.

10.5 PCIe and IDE Cores Interoperability

This section describes how to configure the two PCIe and IDE cores to achieve full interoperability. Synopsys recommends to always run interoperability tests from coreConsultant to verify the correct configuration and interconnection.

10.5.1 Configuring PCIe IDE

Table 10-1 describes the PCIe configuration parameters related to the PCIe IDE functionality.

Table 10-1 PCIe IDE Configuration Parameters

Parameter Name	Parameter Label	Default Value	Description
CX_IDE_ENABLE	Integration and Data Encryption (IDE)	0	IDE and all the necessary top-level interfaces and internal logic.
IDE_SIZE	Capability Structure Size	12'h58	Size of the IDE Capability Structure implemented by the IDE product, expressed in bytes.
CX_IDE_DEDICATED_BUS_PCRC	PCIe IDE/CXL.io IDE Dedicated TX PCRC Bus	0	When enabled, the IDE Security Module passes PCRC on a dedicated bus on the TX path for PCIe IDE/CXL.io IDE. This parameter must be enabled if DWC_IDE_PCRC_SUPPORTED_GUI in the IDE Security Module is set to '1'.
CX_IDE_CAP_STRUCT_AFTER_PL	Move the IDE Capability Structure after Port Logic	0	When enabled, the IDE CapStruct is moved after the Port Logic (After PL_LIMIT). This can be useful if the remaining space between the last. CapabilityStructure and the PortLogic (0x700) is too small to host the IDE CapStruct. This parameter only affects the packaging script building the linked list. Note that whenever the IDE CapStruct is placed after the PortLogic, it will consume a fixed amount of Configuration space for Function0, effectively reducing the range that can be allocated to ELBI.

You must configure the PCIe and IDE controllers properly to be compatible with each other. This implies that the cC parameters described in this section must be set appropriately on both sides. Some parameters are internally generated when configuring the controller, the process requires reconfiguring the controller twice to calculate the necessary parameters and port them to the other controller.

Table 10-2 lists the parameters that must match between the two controllers.

Table 10-2 Parameters that Must Match Between the IDE and PCIe

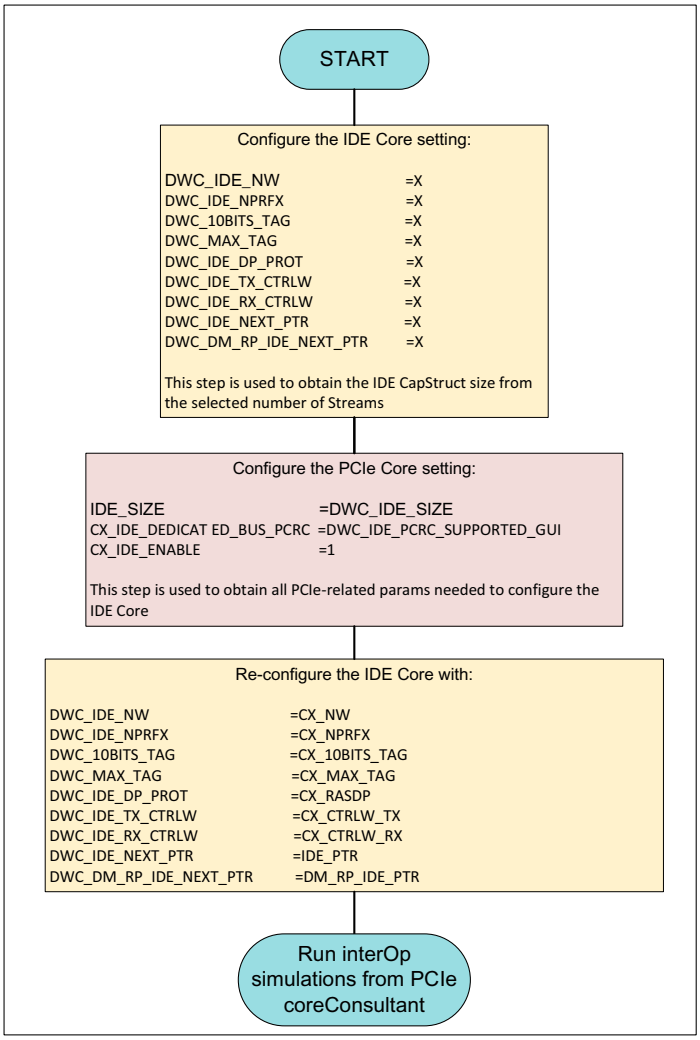
IDE Security Module Parameter	PCIe Parameter
DWC_IDE_NW	CX_NW
DWC_IDE_NPRFX	CX_NPRFX
DWC_10BITS_TAG	CX_10BITS_TAG
DWC_MAX_TAG	CX_MAX_TAG

IDE Security Module Parameter	PCIe Parameter
DWC_IDE_TX_CTRLW	CX_IDE_CTRLW_TX
DWC_IDE_RX_CTRLW	CX_IDE_CTRLW_RX
DWC_IDE_PCRC_SUPPORTED_GUI	CX_IDE_DEDICATED_BUS_PCRC
DWC_IDE_DP_PROT	CX_RASDP
DWC_IDE_SIZE	IDE_SIZE
DWC_IDE_NEXT_PTR	IDE_NEXT_PTR[a]
DWC_DM_RC_IDE_NEXT_PTR	DM_RC_IDE_NEXT_PTR[b]

a. This parameters is not part of the src/DWC_pcie_ctl_cc_constants.svh file and can be found under src/include/cap/port_cfg.svh.
b. This parameters is not part of the src/DWC_pcie_ctl_cc_constants.svh file and can be found under src/include/cap/port_cfg.svh.

A possible flow to follow to configure the two controllers is the following (where a parameter =X means that it is not necessary to set it to the final value, as this is an intermediary step, and any allowed value will work):

Figure 10-10 Configuring the Cores



It is necessary to generate the IDE core configuration in order to calculate `DWC_IDE_SIZE`, which represents the size of the IDE Capability Structure (to be known in order to properly generate the PCIe Core configuration). This step can be skipped if the size is calculated manually (and later checked against the automatically calculated parameter value) using the following formula:

```
IDE_SIZE = 4 * (3 + (NumLinkStreams * 2) + (NumSelectiveStreams * (5 +
NumAddressAssociationRegisterBlocks * 3)))
```

Important:

The size of the IDE Capability Structure rapidly increases with the number of Streams and might exceed the available Configuration Space.

You must ensure that the total size of all the Capability Structures, including the IDE one, does not exceed the address 0x700 (start of the Port Logic registers). Contact Synopsys support through SolvNetPlus for more information.

If the IDE Capability Structure (or one of its subsequent Capability Structures) does not fit below 0x700, you can evaluate the usage of the `CX_IDE_CAP_STRUCT_AFTER_PL` parameter.

When `CX_IDE_CAP_STRUCT_AFTER_PL` is set to '1', the IDE Capability Structure repositions after the Port Logic, that is, its offset is above the Port Logic (usually above 0xC90).

This configuration space is usually empty for Root Complex devices, while it can be allocated to ELBI or TRGT1 for Endpoint devices or Dual Mode devices in Endpoint mode. In the latter cases, the IDE Capability Structure permanently reduces the configuration space you can allocate to ELBI/TRGT1, and it is not possible to change this through firmware or software anymore (except if disabling IDE through the `IDE_CTRL_OFF` register).

For IDE-capable configurations, evaluate the usage of the following parameters for configuring the PCIe controller:

- `CX_DETECT_POISONED_CFGWR`:
  - Provides a top-level interface for monitoring reception of Poisoned CfgWr
  - Useful for the implementation of the DOE Capability Structure by the application
  - Set by default for `CX_IDE_ENABLE`
- `CX_CONFIG_INTERCEPT_ENABLE`:
  - Provides the CIC interface
  - The application uses CIC interface to monitor undesired or suspicious accesses to sensible registers from the wire
  - The CIC interface also allows to overwrite the T-bit of internally generated Completions before they are transmitted
  - Set by default for `CX_IDE_ENABLE`

11

Automotive

This section describes the automotive features supported by the controller and enabled by the `CX_AUTOMOTIVE_ENABLE` parameter when you have the corresponding automotive license. The following topics are discussed:

- "Overview" on page 614
- "Automotive Safety Package" on page 616
- "CDM Register Check" on page 617
- "Functional Safety Interrupt" on page 620

11.1 Overview

In order to make an IP better suited for applications in the automotive space functional safety needs to be considered throughout the IP development process. ISO 26262 addresses functional safety, and requires the hardware components to be analyzed with respect to their ability to detect and/or control the effects of random hardware faults. Random hardware faults can occur at any point in time during the life-cycle of the hardware component. They can take the form of permanent faults, or transient faults, and require safety mechanisms to be in place to ensure that severe consequences do not occur as a result of those faults. The PCIe controller includes a broad range of internal safety mechanisms which assist in handling random hardware faults. These complement the mechanisms already defined in the PCIe protocol (for example, Link recovery, LCRC/ECRC, error checking, and flow control). The list of internal safety mechanisms includes:

- "Reliability, Availability, and Serviceability (RAS)" on page 162
- "RAS Data Protection (RAS DP)" on page 164
- "CDM Register Check" on page 617
- "Tx Interface Progression Detection (CX_INTERFACE_TIMER_EN =1)" on page 128

ASIL B Random Hardware Fault Assessment

The PCIe IP has been analyzed in the scope of ISO 26262 targeting ASIL B for particular reference configurations.

Reference Configurations

Two reference configurations with the following major features: Gen4, x2 (64 bit) and Gen5, x8 (256 bit), covering Dual Mode (DM) controller, AXI interface with DMA, AXI DBI Slave (shared) and Native DBI configuration access, fixed and resizable BAR, iATU, VPD, TPH, CIC, PTM, L1 sub-states, integrated MSI-X request generation and reception (iMSI-X), MSI-IO, ARI, FLR, SR-IOV, SRIS, IDO, LTR, OBFF, L1 sub-states, ECRC, AER, RAS D.E.S., RAS DP datapath parity checking and RAM ECC protection, CDM register check, and Tx interface progression detection have been used for ASIL-B random hardware fault metrics assessment.

FMEDA

A Failure Mode, Effects, and Diagnostic Analysis (FMEDA) analysis has also been performed for the reference configurations. The analysis relies on a per-transistor failure rate derived in accordance to IEC TR 62380. This, in conjunction with area information from design synthesis trials, allows to determine the failure rate for the IP which can then be distributed between the modules of the design according to their relative area.

The list of failure modes and their effects is derived for each module and each of the failure modes is assigned a portion of the failure rate of the module they belong to. If the failure mode has the potential to violate a top level safety requirement (error at a top-level output pin) there is a search for a safety mechanism that can prevent that violation, totally or partially. If a safety mechanism exists, for example ECC protection for RAMs, a diagnostic coverage (DC) value is assigned representing the percentage of the failure rate associated with the specific failure mode that is prevented from violating a top level safety requirement. The process is repeated for all failure modes and finally results are accumulated and IP safety metrics are collected in the FMEDA.

DFMEA

A Design Failure Mode and Effects Analysis has been performed. This analyzes the nominal and safety functions of each of the major blocks in the design and identifies the potential failure modes, their potential

effects impacting safety goals at the top-level and the potential causes at the top-level within each of the blocks. It assigns Severity, Observability, and Detection ratings to each failure mode and identifies controls for prevention and detection of the failure modes, which reduce the risk of the failure mode occurring. Where appropriate, it recommends actions for SoC implementers to reduce risk of design errors during integration of the IP.

DFA

A Dependent Failure Analysis has been performed. The results describe the design risk factors leading to dependent failures and identifies mitigation actions to prevent or control the dependent failures. Identified actions for integration, verification, or validation are defined or referenced in the Assumptions of Use section of the Safety Manual. The DFA document covers:

- Common Cause Failure (CCF) analysis, including mission circuits and their corresponding safety mechanisms, corresponding safety requirements, dependent failure initiators, coupling factors, measures for fault avoidance, measures for fault control, verification method, and verification evidence. Both internal and external safety mechanisms are considered.
- Cascading Failure (CF) analysis, including pairs of parts, subparts or elementary subparts affected by cascading failures, affected safety requirements, analysis scope, cascading failure mechanism, measures for fault avoidance, measures for fault control, verification method, and verification evidence.

11.2 Automotive Safety Package

The automotive safety package for the controller IP contains the functional safety analysis reports, a safety manual with all safety relevant information and documents covering the assessment of suitability for use in safety critical applications. After you have configured the controller in coreConsultant, you can access the automotive documents at `<DESIGNWARE_HOME>/iip_ap/DWC_ac_pcie_ctl/latest`.

11.3 CDM Register Check

The controller supports automatic CDM register checking (`CX_CDM_REG_CHK_EN =1`) to ensure that CDM registers are not corrupted. The CDM register check is done for each PF/VF. The iATU and DMA port logic registers are also checked. All the CDM registers are checked, except for the following:

- Write command registers which do not have a corresponding physical register. For example, Write only over DBI re-train link command, and DMA doorbell (`DMA_WRITE_DOORBELL_OFF` and `DMA_READ_DOORBELL_OFF`).
- RO status register with physical register implemented outside CDM. For example, negotiated link width (`PCIE_CAP_NEGO_LINK_WIDTH`).
- MSI-X tables.
- RAM-mapped DMA registers covered under RAS DP. For example, message interrupts and context registers for DMA write and read channels.

CDM Register Check Features and Limitations

- The feature is supported for all configurations.
- The register check starts when `PL_CHK_REG_CONTROL_STATUS_OFF.CHK_REG_START` is set to 1.
- Continuous or run once (through doorbell) modes.
- Programmable start and end check addresses through `PL_CHK_REG_START_END_OFF` register.
- `CHK_REG_COMPLETE` field of `PL_CHK_REG_CONTROL_STATUS_OFF` indicates check has finished.
- `CHK_REG_COMPARISON_ERROR` field of `PL_CHK_REG_CONTROL_STATUS_OFF` indicates comparison failure.
- Failing address and PF/VF of the first comparison error found is recorded in registers `PL_CHK_REG_ERR_ADDR_OFF` and `PL_CHK_REG_ERR_PF_VF_OFF` respectively.

  Note: Only the address of the first error found is recorded. The checking sequence proceeds until complete, unless software clears the `CHK_REG_START` field of `PL_CHK_REG_CONTROL_STATUS_OFF` register.
- The DBI/wire accesses to the CDM registers takes priority over any CDM register check sequence.
- Built in self-test functionality checks the comparison logic.

  `CHK_REG_LOGIC_ERROR` field of `PL_CHK_REG_CONTROL_STATUS_OFF` register indicates error in comparison logic.
- Test mode (`cdm_reg_chk_test_en =1`) permits forcing of comparison failures at externally writable address locations.
- The controller only reports the error (through AER: Uncorrectable Internal Error); it does not take any action to correct the error.

CDM Register Check Operational Details

CDM register checking is controlled by a FSM which sequences checking of CDM register values against shadow registers, in CDM_B (*Register content check*). The CDM register checker's self-test functionality checks the comparison logic, by injection of an error at one particular address location (*Error Injection*). It also checks address increments during the comparison sequence (*Address Check*).

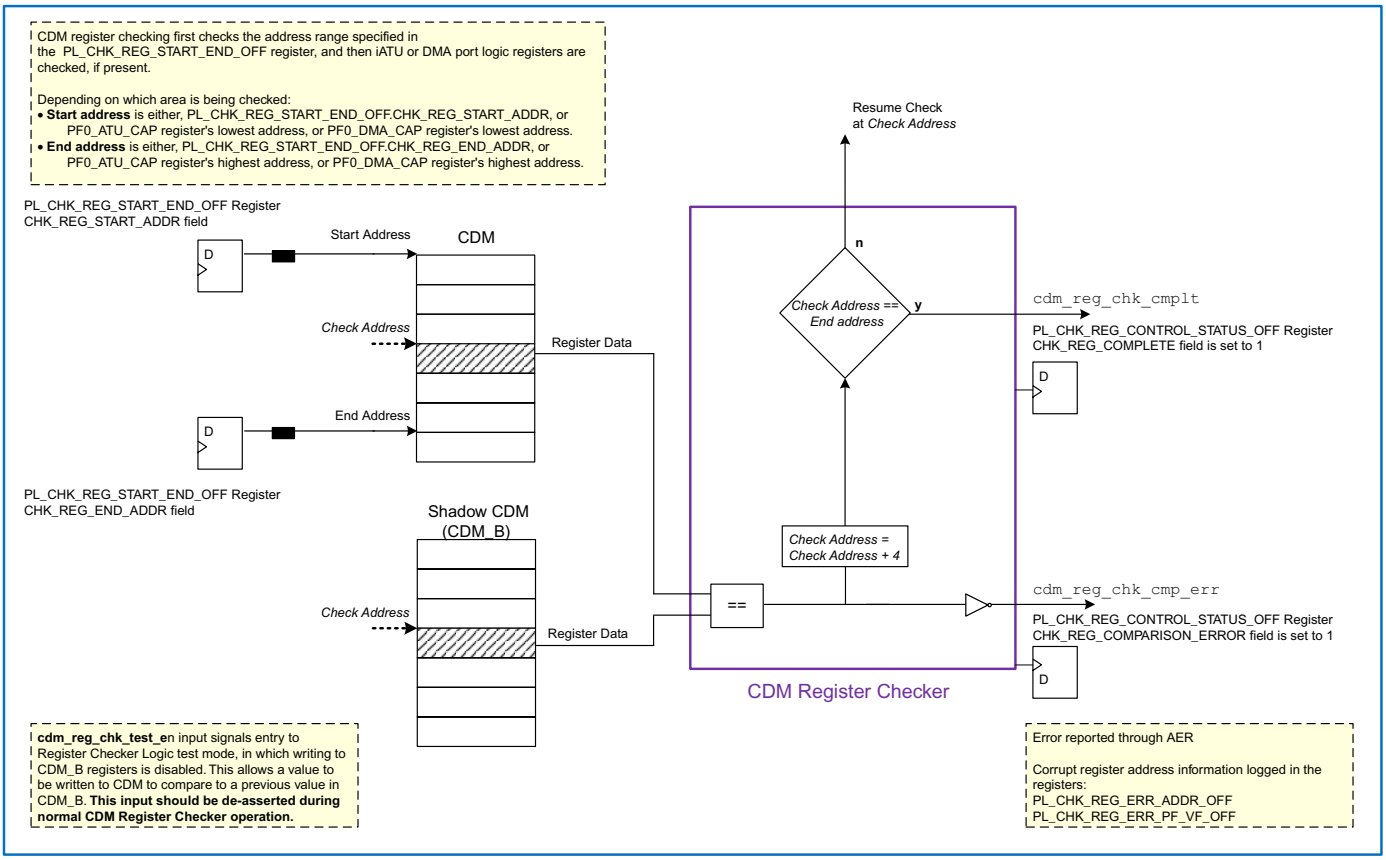
Register Content Check (CDM versus CDM_B): The CDM register checker compares CDM and CDM_B data. If a mismatch is found,

- Signal `cdm_reg_chk_cmp_err` is asserted, and
- `CHK_REG_COMPARISON_ERROR` field of `PL_CHK_REG_CONTROL_STATUS_OFF` register is set to 1, to indicate comparison error.

The checking sequence is as follows:

1. The address range specified in register `PL_CHK_REG_START_END_OFF` is checked for comparison errors. Check is done for each PF/VF also.
2. If accesses to DMA registers are through BAR matched MEM/IO requests, rather than legacy indirect register addressing, DMA port logic registers for all configured channels are checked. For more information on DMA port logic register location, see "Register Configuration Space Overview" on page 142.
3. If accesses to iATU registers are through BAR matched MEM/IO requests, rather than legacy indirect register addressing, iATU port logic registers for all configured regions are checked. For more information on iATU port logic register location, see "Register Configuration Space Overview" on page 142

Figure 11-1 Register Content Check (CDM versus CDM_B)



Error Injection: At the end of *Register Content Check*, the comparison logic is checked by comparing data at address 0x100 in CDM against the inverted value of data in CDM_B at the same address. If the values do not differ:

- Signal `cdm_reg_chk_logic_err` is asserted, and

- `CHK_REG_LOGIC_ERROR` field of `PL_CHK_REG_CONTROL_STATUS_OFF` register is set to 1, to indicate an error in comparison logic.

Note: For automotive configurations, the error is injected at `AER_EXT_CAP_HDR_OFF` register, because AER is always enabled.

Address Check: During *Register Content Check* and *Error Injection*, each time the address is incremented, it is checked. If the increment is not equal to 4 (corresponding to the next 32-bit register address) then:

- Signal `cdm_reg_chk_logic_err` is asserted, and
- `CHK_REG_LOGIC_ERROR` field of `PL_CHK_REG_CONTROL_STATUS_OFF` register is set to 1, to indicate an error in comparison logic.

CDM Register Check Test Mode (cdm_reg_chk_test_en =1)

The test mode is used to verify the *Register Content Check* function. Setting `cdm_reg_chk_test_en` to 1 disables writes to CDM_B. This allows writing a value to CDM to compare with the previous value in CDM_B. If the value written to CDM is different to the value held in CDM_B, comparison error is generated when the address is checked, indicating the *Register Content Check* function is working correctly.

Note: In normal operation mode `cdm_reg_chk_test_en` should be driven to 0.

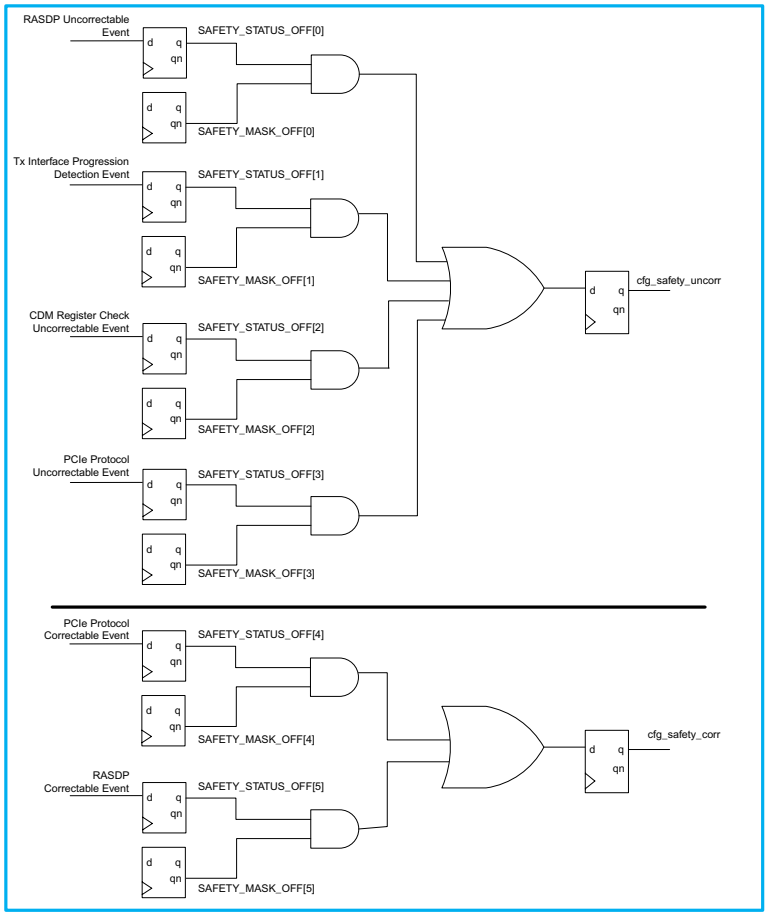
11.4 Functional Safety Interrupt

For configurations with automotive feature enabled (`CX_AUTOMOTIVE_ENABLE =1`) the controller provides dedicated functional safety interrupts (`cfg_safety_uncorr` and `cfg_safety_corr`) to route the safety related indications to your application. These interrupts report events of different severity; `cfg_safety_uncorr` reports uncorrectable errors and `cfg_safety_corr` reports correctable errors. These interrupts have independent status and mask bits in `SAFETY_STATUS_OFF` and `SAFETY_MASK_OFF` registers, respectively. The functional safety interrupt outputs provide indications from two different event categories:

- PCIe protocol defined events
- Internal safety mechanisms events
  - RAS Data Protection
  - CDM Register Check
  - Tx Interface Progression Detection

Events signaled from both the categories are recorded in the status register regardless of the specific event mask setting. The mask controls whether an event causes the safety interrupt (`cfg_safety_uncorr/cfg_safety_corr`) to be asserted or not.

Figure 11-2 Events Reported by cfg_safety_uncorr and cfg_safety_corr



12

Parameter, Signal, Register, and Internal Parameter Descriptions

For Parameter, Signal, Register, and Internal Parameter Descriptions, see *PCIe Controller RP Reference Manual* (DWC_pcie_ctl_rp_reference.pdf).

A

Advanced Information: Gen3/4/5 Equalization Details and Example

This appendix discusses advanced aspects of Gen3 or above equalization operation, and also provides a usage example. The terms MAC and PCIe controller are used interchangeably in this section. The following topics are discussed:

- "Equalization Overview and Synopsys-Specific Features" on page 624
- "Detailed Equalization Procedure" on page 637
- "Usage Example" on page 645

A.1 Equalization Overview and Synopsys-Specific Features

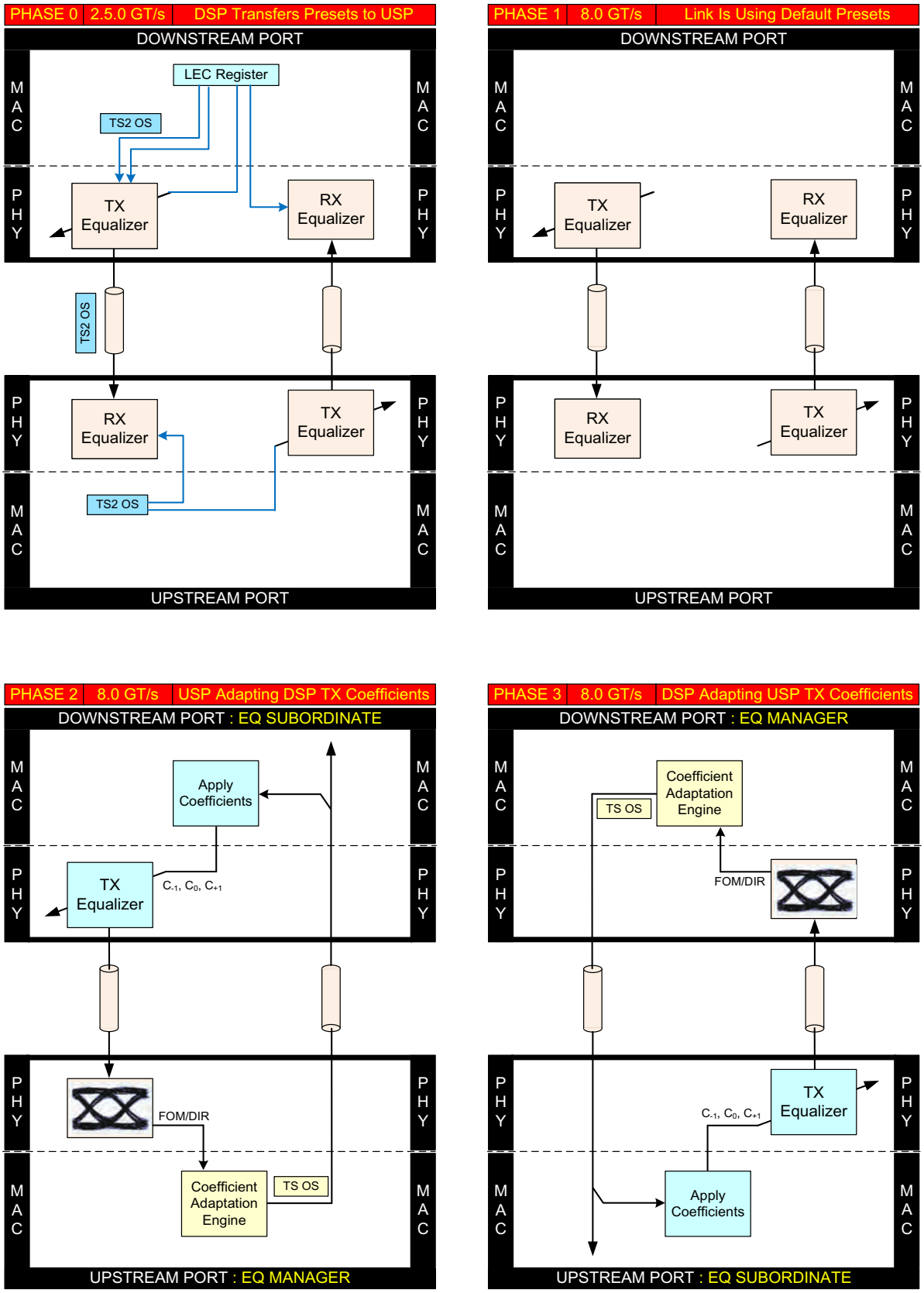
The controller performs link equalization during link training to improve signal quality by adjusting the transmitter and receiver equalization parameters for each lane on each side of the link. The receiver equalizer is a self-contained adaptive equalizer within the PHY. After it receives its preset hints from the Downstream Port (DSP) Lane Equalization Control (LEC) registers for Gen3 and the DSP Lane Equalization Control 2 (LEC2) registers for Gen4, it does not require any more interaction with the MAC or with the remote PHY. The transmitter equalizer is a three tap equalizer whose coefficients are adapted by a coefficient adaptation engine in the remote MAC. This engine uses a quality metric or an error metric from the local PHY to drive its adaptation and convergence algorithms. The controller supports figure of merit (FOM) and direction (DIR) feedback modes. There are four phases in the equalization of the link as indicated in Figure A-1 on page 625. They are briefly described here and are discussed in greater detail later.

- **Phase 0:** When negotiating to Gen3 speed, before the link moves to Gen3 speed, the DSP applies the DSP presets in its LEC to its local transmitter and receiver equalizers, and sends the USP presets to the Upstream Port (USP) over the link using TS2 OS's.

  When negotiating to Gen4 speed, before the link moves to Gen4 speed, the DSP applies the DSP transmitter presets in received TS2 OS's or its LEC2 to its local transmitter and DSP receiver presets in its LEC2 to its local receiver equalizers, and sends the USP presets to the USP over the link using 8GT EQ TS2 OS's. Optionally, the USP can request DSP's transmitter presets in sending TS2 OS's.
- **Phase 1:** The USP and DSP apply the presets and work to operate and maintain a stable link operating at the current Gen3 or above speed. No adaptation of presets/coefficients takes place.
- **Phase 2:** The USP is the equalization master and tunes the DSP transmitter equalizer. Its adaptation engine sends equalization requests to the DSP using TS1 OS's. By using the Use Preset bit in the TS1, it directs the DSP to map and use a preset or to directly use the coefficient supplied in the TS1 as shown in Figure A-1 on page 625.
- **Phase 3:** The DSP is the equalization master and tunes the USP transmitter equalizer.
- **Recovery.RcvrLock:** After Phase 3, both the sides of the link move to Recovery.RcvrLock. The DSP hardware does not initiate equalization redo if any side of the link detects equalization problems (for example, EQ settings mismatch) in the Recovery.RcvrLock state.The DSP generates an interrupt to notify the software of equalization problems, the software must decide whether to redo equalization or not.
- **Recovery.RcvrCfg:** The USP performs the maximum number of EQ redo requests set in the `GEN3_EQ_REQ_NUM` field of the `GEN3_EQ_CONTROL_OFF`, when the `GEN3_SUPPORT_FINITE_EQ_REQUEST` field of the `GEN3_EQ_CONTROL_OFF` register is set to '1'. The USP stops the EQ redo requests for the current data rate to avoid the endless EQ redo requests from the USP when the EQ redo requests are equal or above the number set in the `GEN3_EQ_REQ_NUM` field.

An equalization master is a port that is currently adjusting the remote transmitters settings. This can be an USP in phase 2 or a DSP in phase 3. An equalization slave is a port that is currently receiving and applying coefficients (requested by the remote equalization master) to its local transmitter. This can be a USP in phase 3 or a DSP in phase 2.

Figure A-1 Gen3 Equalization Phases: Simplified Overview



Gen 5 and above

For Gen5 and above data rates the controller supports "Equalization Bypass to Highest Data Rate" and "No Equalization Needed" features.

- **Equalization Bypass to Highest Data Rate:** To enable this feature, set the `EQ_BYPASS_HIGHEST_RATE_SUPPORT` field of the `PL32G_CAPABILITY_REG` register to '1'. When this feature is enabled, and supported by the other side of the link as well, the link skips Gen3 and Gen4 equalizations and directly performs Gen5 equalization. The link can run at Gen1, Gen2, and Gen5 rates.

  When the link is at Gen1 or Gen2 rate:

  - To do EQ for Gen3 and Gen4 rates after performing Gen5 equalization, the DSP controller must set the following register fields:

    i. `PL32G_CONTROL_REG.EQ_BYPASS_HIGHEST_RATE_DISABLE =1`

    ii. `PL32G_CONTROL_REG.NO_EQ_NEEDED_DISABLE =1`

    iii. `LINK_CONTROL3_REG.PERFORM_EQ =1`

    iv. `LINK_CONTROL2_LINK_STATUS2_REG.PCIE_CAP_TARGET_LINK_SPEED >Gen2`

    v. `LINK_CONTROL_LINK_STATUS_REG.PCIE_CAP_RETRAIN_LINK =1`

  - If USP receives EQ TS2s with Gen3 rate in Recovery.RcvrCfg state, the USP uses the presets received from the EQ TS2s as the initial presets for the Gen3 rate equalization when the rate changes to Gen3.
  - If USP receives EQ TS2s with Gen5 rate in Recovery.RcvrCfg state, the USP uses the presets received from the EQ TS2s as the initial presets for the Gen5 rate equalization when the rate changes to Gen5.

- **No Equalization Needed:** To enable this feature, set the `NO_EQ_NEEDED_SUPPORT` field of the `PL32G_CAPABILITY_REG` register to '1'. When this feature is enabled, and supported by the other side of the link as well, the link does not need to perform equalizations for any data rate. The link can reliably run at any data rate from Gen1 to Gen5, if the link is reliable at these data rates.

A.1.1 Preset Mapping Modes

The controller provides a configurable preset-to-coefficient mapping implementation to inter operate with different PHYs. Mapping consists of converting the requested preset to generating a set of three 6-bit coefficients using the PHYs actual LF and FS values, and by referring to the preset encoding table in Section 4.2.3.2. "Encoding of Presets" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. A preset index identifies a set of typical settings for the transmitter filter coefficients. However, these floating point coefficients are normalized values and must be scaled by the actual voltage swings (known as the LF and FS values) used by the PHY to derive the actual coefficients used in the transmitter filter. The controller provides three mapping modes selectable at configuration time using `CX_GEN3_EQ_PSET_COEF_MAP_MODE`:

- **(0) Dynamic PHY**

  The coefficients are dynamically mapped in the PHY. This is the default mode.

Note

- When a Downstream Port is redoing equalization from the Gen3 or above speed L0 state and its PHY mapping time from presets to coefficients is greater than the duration of two TS ordered sets (OS); the controller does not strictly follow the specification requirement: *"The Port must ensure that no more than 2 TS1 OS with EC=00b are transmitted due to being in Recovery.RcvrLock before starting to transmit the TS1 OS required by Recovery.Equalization."* However, this does not impact any functionalities in the controller if the mapping time is less than the duration of 8 TS OS.
- If the PHY's mapping time is greater than the duration of 8 TS OS, the controller cannot prevent transmitting 8 or more TS1 OS with EC=00b, which is interpreted by the remote partner side as a condition to go to Recovery.RcvrCfg instead of Recovery.Equalization. This defeats the intent of redoing equalization. This should never happen because the PHY typically takes less than the duration of two TS OS to complete the mapping.

- **(1) Dynamic MAC**

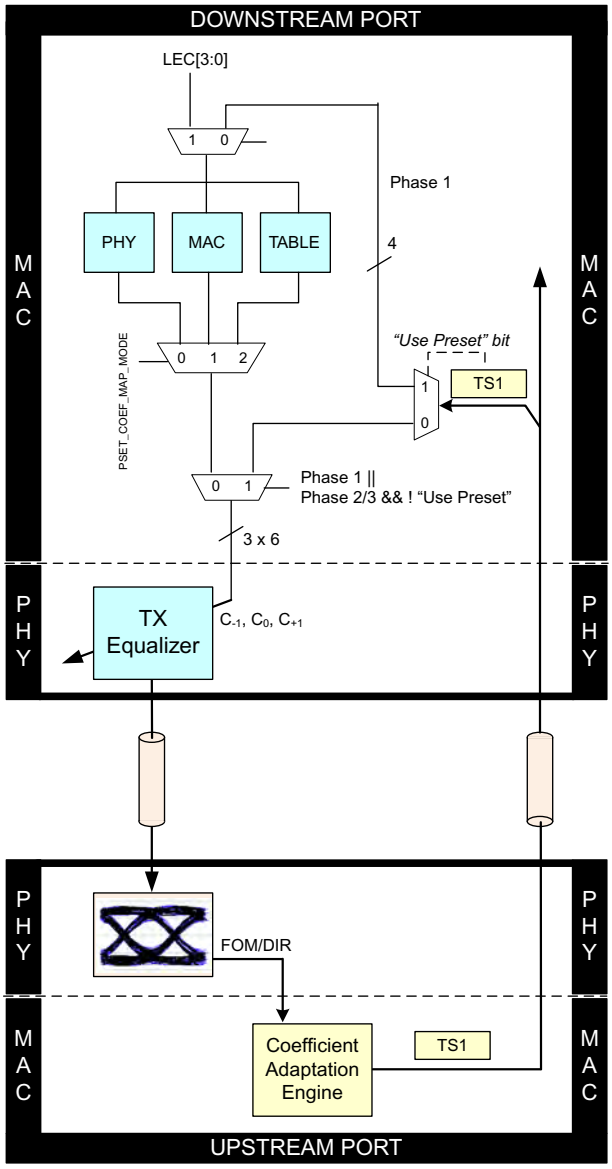
  The coefficients are dynamically mapped in the MAC. The MAC maps a preset to actual coefficients using the normalized coefficients and the FS and LF values provided by the PHY.

- **(2) Programmable Table**

  Your application software programs a preset-to-coefficient mapping LUT called "Gen3 EQ Presets to Coefficients Mapping Register" (`GEN3_EQ_PSET_COEF_MAP__i`) in the port logic register space.

The adaptation engine in the equalization master sends preset/coefficient requests to the equalization slave using TS1 ordered sets. The Use Preset bit in the TS1 ordered set directs the equalization slave to map and use a preset, or to directly use the coefficients supplied in the TS1.

Figure A-2 Preset Mapping Modes (Symbolic Representation; not Actual Implementation)



A.1.2 Coefficient Tuning Feedback Modes

The receiver PHY generates feedback to the transmitter coefficient adaptation engine for determining the optimal coefficients in phases 2 and 3 as shown in Figure A-3 on page 630. This feedback can be in one of two types: figure of merit (FOM) or direction control (DIR). FOM is an absolute number from 0 to 255 indicating the quality of the received signal. DIR is a 2-bit control indicating to the coefficient adaptation engine to increment by 1, decrement by 1, or leave the current coefficient unchanged. Both DIR and FOM could possibly (for example) be derived by the receiver PHY from the eye opening measurement or a value that is inversely proportional to the amount of ISI at the edge of the eye. The FOM procedure in the controller always precedes DIR.

Figure of Merit (FOM)

The coefficient adaptation engine uses the 8-bit feedback from the local PHY on `phy_mac_fomfeedback`. It evaluates the FOM for each of the presets (that you specify to check) and then selects that preset which has the highest FOM. This can be considered coarse or rough tuning as the only possible coefficient sets are those specified in the preset table in the PCIe specification. If your PHY does not support FOM, then it is expected that your PHY returns a constant value on `phy_mac_fomfeedback` for each preset request (if any), and the final preset is used as the starting point for the DIR process.

Direction Change (DIR)

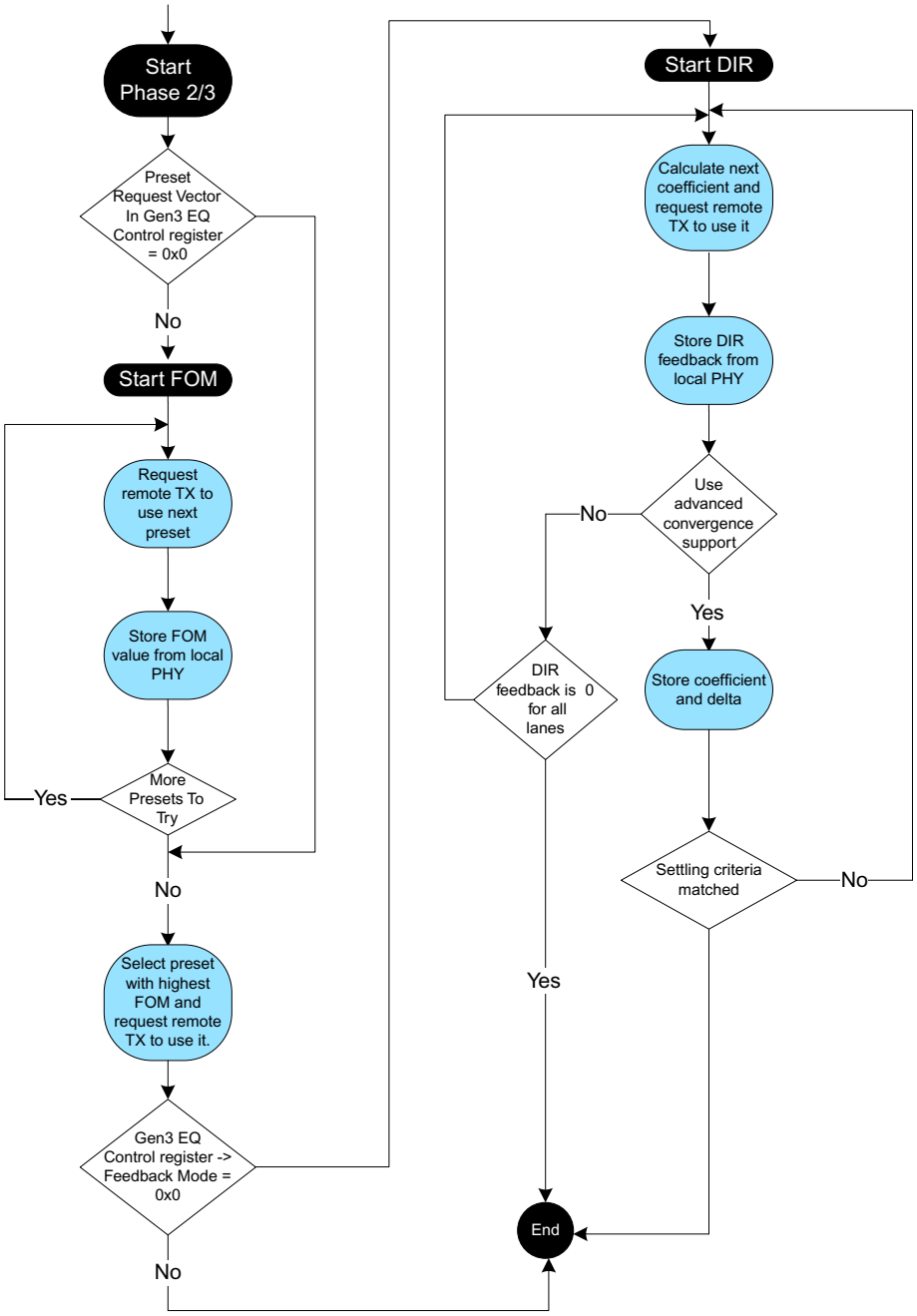
The coefficient adaptation engine in the master uses the 2-bit feedback from the local PHY on phy_mac_dirfeedback to fine tune the coefficients. There are two bits in `phy_mac_dirfeedback` for each of the three coefficients in the transmit filter, indicating whether to increment-by-1, decrement-by-1, or leave unchanged the current coefficient value. This can be considered as fine tuning. If your PHY does not support DIR then you can skip this process by setting the Feedback Mode field in the Gen3 EQ Control Register `GEN3_EQ_CONTROL_OFF` to 1 as indicated in Figure A-3 on page 630.

DIR Convergence Criteria

When you are only using FOM feedback, then the equalization process is finished after you have determined which preset produces the highest figure of merit. When you are using DIR feedback, then the controller normally considers coefficient optimization to be complete when the DIR feedback on all lanes is 0. You can optionally apply settling or windowed filtering to the DIR result. This is an optional mode that allows the controller to use a different decision process to decide when coefficient optimization is complete. It may be useful if you observe oscillations on the DIR feedback as you are waiting for a 0 value on all lanes at the same time. The controller stores metrics corresponding to each coefficient for analysis in a window defined by you. You program the windowing or settling criteria using the port logic Gen3 EQ Direction Change Feedback Mode Control Register (`GEN3_EQ_FB_MODE_DIR_CHANGE_OFF`). You can define the analysis window (number of evaluations) and the delta between the maximum and minimum coefficient values. You must select the DIR convergence mode at controller RTL configuration time using the `CX_GEN3_EQ_COEF_CONV_SUPPORTED` parameter. You cannot change the mode using software that is, it is not programmable. This mode is described in more detail later in this section.

Note It is not advisable to choose convergence support (CX_GEN3_EQ_COEF_CONV_SUPPORTED) if your application cannot program the convergence mode registers through the DBI.

Figure A-3 Representational (Not Actual) Flowchart Showing the Sequence of Major Operations for EQ Master



Master Operation in FOM Mode

The controller fine tunes the link partner using only preset requests. Upon entry into phase 2 (USP) or phase 3 (DSP) in Figure A-6, the EQ master proceeds in consecutive steps as follows:

1. The unmodified settings of the remote transmitter are evaluated.
2. The presets in the Preset Request Vector field are requested and evaluated. Preset requests are made (one for each bit set in the Preset Request Vector in the Gen3 EQ Control Register `GEN3_EQ_CON-`

TROL_OFF) to the link partner. If accepted, an evaluation is made to the local PHY which returns a FOM on the relevant lane of phy_mac_fomfeedback[NL*8-1:0].

3. The final request to the remote partner is the preset with the highest FOM from all the previous evaluations. The preset that returned (through phy_mac_fomfeedback) the highest FOM in step 2 is requested again. When the final preset request is accepted by the remote partner on all lanes, then optimal settings are achieved.

Notes:

- For step 3, it is also possible to choose (using the Include Initial FOM field in the Gen3 EQ Control Register GEN3_EQ_CONTROL_OFF), whether to include or not the FOM feedback from the evaluation performed in step 1, when finding the highest FOM among all preset evaluations.
- Typically this mode is used in conjunction when you select multiple presets in the Preset Request Vector field, such that multiple evaluations are performed in step 2, and a final choice is made in step 3.

Master Operation in DIR Mode

The controller initially tunes the link partner using preset requests. Upon entry into phase 2 (USP) or phase 3 (DSP) in Figure A-6, the EQ master proceeds in consecutive steps as follows:

1. The unmodified settings of the remote transmitter are evaluated.
2. The presets in the Preset Request Vector field are requested and evaluated. Preset requests are made (one for each bit set in the Preset Request Vector in the Gen3 EQ Control Register GEN3_EQ_CONTROL_OFF) to the link partner. If accepted, an evaluation is made to the local PHY which returns a FOM on the relevant lane of phy_mac_fomfeedback[NL*8-1:0]. This step is optional, and can be skipped by programming the Preset Request Vector field to all zeros. If your PHY does not support FOM, then it is expected that your PHY returns a constant value on phy_mac_fomfeedback for each preset request (if any), and the final preset is used as the starting point for the DIR process.
3. The final preset request to the remote partner is the preset with the highest FOM from all the previous evaluations. The preset that returned (through phy_mac_fomfeedback) the highest FOM in step 2 is requested again. This step is optional, and can be skipped by programming the Preset Request Vector field to all zeros.
4. Incremental changes to the remote transmitter coefficients are requested and evaluated, until optimal settings are found, or the 24 ms timeout expires.

The CX_GEN3_EQ_COEF_CONV_SUPPORTED parameter determines if the controller implements additional logic for checking that step 4 is converging toward a stable point.

When you set it to 0, the controller uses a simple procedure to find optimal coefficient settings using the DIR value fed back from the PHY on the relevant lane of phy_mac_dirfeedback[NL*6-1:0].When bits [5:4] and bits [1:0] (pre and post-cursor) on the relevant lane of phy_mac_dirfeedback[NL*6-1:0] are 0x0 for all lanes, then optimal settings are achieved.

When you set it to '1', the controller implements more sophisticated criteria to determine when the current remote transmitter coefficients are optimal. The criteria require that the coefficients of the remote transmitter, requested and evaluated in the last N attempts have a maximum delta that does not exceed a value D. The convergence parameters N and D are programmable and correspond respectively to the Convergence Windows Depth and Convergence Window Aperture fields in the Gen3 EQ Direction Change Feedback Mode Control Register (GEN3_EQ_FB_MODE_DIR_CHANGE_OFF). To prevent premature convergence, the criteria also require that a minimum time T has elapsed from when the EQ Master phase has started. The value of this timer T is programmable in the Gen3 EQ Direction Change Feedback Mode Control Register (GEN3_EQ_FB_MODE_DIR_CHANGE_OFF).

Notes:

- When step 2 and 3 are skipped, the incremental adjustments of the coefficients in step 4 starts from the settings that the remote transmitter has used to enter phase 2 or phase 3. When step 2 and 3 are executed, the incremental adjustments of the coefficients start from the final preset requested in step 3.
- For step 3, it is also possible to choose (using the Include Initial FOM field in the Gen3 EQ Control Register `GEN3_EQ_CONTROL_OFF`), whether to include or not the FOM feedback from the evaluation performed in Step 1, when finding the highest FOM among all Preset evaluations.
- Typically this mode is used in conjunction with zero, or at most one preset selected in the Preset Request Vector field, such that no more than one evaluation is performed in step 2.

A.1.3 EQ Coefficient Request in FOM Mode (FCOEF)

You can enable the EQ Coefficient Request feature in Figure Of Merit (FOM) mode by setting the parameter `CX_EQ_MASTER_COEF_REQ_FOR_FOM_ENABLE` to '1'.

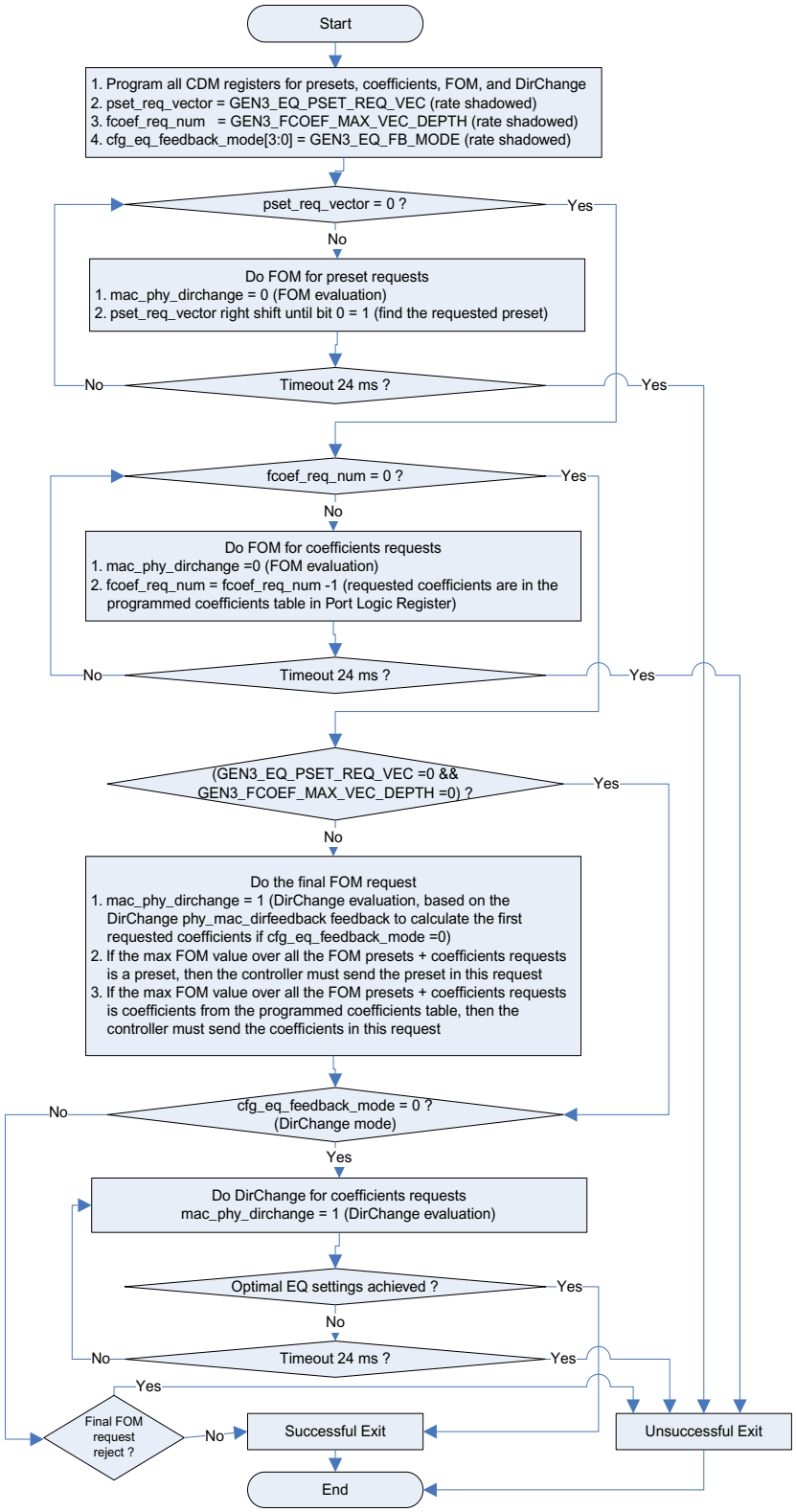
Features

- The EQ master requests for coefficients from the programmed port logic registers `GEN3_FCOEF_LANE*_[PRE|POST]_CURSOR` in Recovery.Equalization Phase2 for USP or Recovery.Equalization Phase 3 for DSP in EQ master phases. For more information, see"Recovery.Equalization Phase 2" on page 639 and "Recovery.Equalization Phase 3 (DSP/USP)" on page 644.
- The EQ master informs the PHY of performing FOM evaluation for the coefficients requested and sends the coefficients through TS1s to the remote partner. The coefficients received by the remote partner are applied to its transmitter.

The FOM coefficient vector length is equal to the value specified in `GEN3_FCOEF_MAX_VEC_DEPTH` field of the `GEN3_FCOEF_VEC_DEPTH_OFF` register. You can set the maximum value for `GEN3_FCOEF_MAX_VEC_DEPTH` field using the hidden parameter `DEFAULT_MAX_COEF_FOM_VECTOR_DEPTH`.

Figure A-4 shows the general flow for EQ master along with the coefficient request feature for FOM.

Figure A-4 General Flow for EQ Master



The value for `GEN3_FCOEF_ACCESS_VAR` register field must be set before you program the registers `GEN3_FCOEF_LANE*_OFF [0: DEFAULT_MAX_COEF_FOM_VECTOR_DEPTH - 1]`. The 12-bits in the `GEN3_FCOEF_LANE*_OFF` shadow register is the Pre-cursor and Post-cursor of the lane.

The cursor is calculated as based on the Full Swing (FS) advertised by the remote partner using the formula:

```
cursor = FS - Pre-cursor - Post-cursor
```

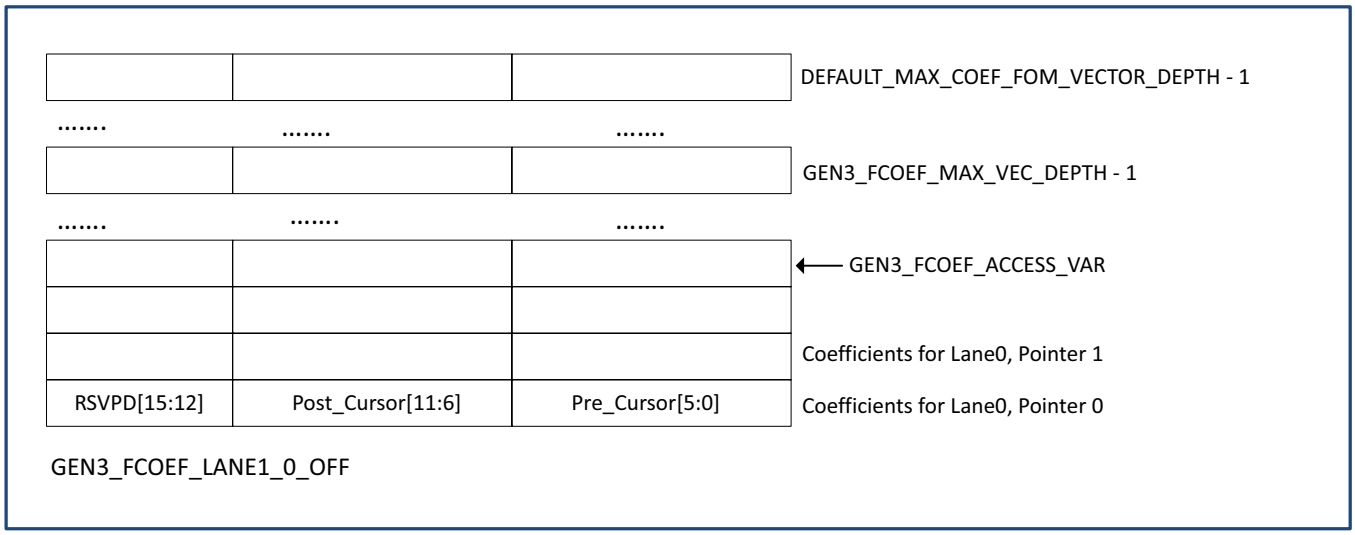
Note: The controller does not check for the programmed coefficient legality. Hence, you must program the correct coefficients without breaking the Rules for Transmitter Coefficient as mentioned in the *PCI Express Base Specification, Revision 6.0, Version 1.0*.

Table A-1 shows a brief impact of GEN3_FCOEF_MAX_VEC_DEPTH register field value to determine the number of coefficient request. Figure A-5 shows the FCOEF coefficients array for lane0. For more information on GEN3_FCOEF_ACCESS_VAR, see "Register Descriptions" chapter of the Reference Manual.

Table A-1 Impact of GEN3_FCOEF_MAX_VEC_DEPTH Field

GEN3_FCOEF_MAX_VEC_DEPTH Value	Description
0	No coefficient request for FOM mode
<= DEFAULT_MAX_COEF_FOM_VECTOR_DEPTH	The number of coefficient request is GEN3_FCOEF_MAX_VEC_DEPTH -1
> DEFAULT_MAX_COEF_FOM_VECTOR_DEPTH	The controller assigns GEN3_FCOEF_MAX_VEC_DEPTH = DEFAULT_MAX_COEF_FOM_VECTOR_DEPTH The number of coefficient requests is 1,2,3,..,DEFAULT_MAX_COEF_FOM_VECTOR_DEPTH

Figure A-5 FCOEF Coefficients Array for Lane0



A.1.4 Other Features and Limitations

Master 24ms Timeout

You can program the controller to the behavior of the EQ master LTSSM state machines, when the coefficient calculation has not converged after the 24ms timeout. Instead of moving back to Recovery.Speed (which is the *PCI Express Base Specification, Revision 6.0, Version 1.0* behavior), the EQ master LTSSM continues as normal (phase 3 if USP, Recovery.RcvrLock if DSP). For more information, see Gen3 EQ Control Register `GEN3_EQ_CONTROL_OFF`.

Limitations

In Recovery.RcvrLock, there is no support for directing the controller to an arbitrary equalization phase.

A.1.5 Programming Registers

The following registers are used for programming the Gen3 or above EQ logic.

Table A-2 Port Logic Registers Used

Bits	Default
+0x190	Gen3 Control Register
+0x194	Gen3 EQ FS and LF Register
+0x198	Gen3 EQ Presets To Coefficients Mapping Register
+0x19C	Gen3 EQ Preset Index Register
+0x1A4	Gen3 EQ Status Register
+0x1A8	Gen3 EQ Control Register
+0x1AC	Gen3 EQ Direction Change Feedback Mode Control Register

The Gen3 EQ Presets to Coefficients Mapping registers are programmed through a register indirect addressing scheme (using an index register) to reduce the address footprint in the PCI Express extended configuration space. This register is only used when you set the Gen3 Equalization Presets to Coefficients Mapping Mode to Programmable Table, by setting the `CX_GEN3_EQ_PSET_COEF_MAP_MODE` configuration parameter to 2. The default value for each register is given in Table A-3.

Table A-3 Default Coefficient Values for Each Preset Index

Preset Index (i)	DEFAULT_GEN3_EQ_POSTCURSOR_PSETi	DEFAULT_GEN3_EQ_PRECURSOR_PSETi
0	round(DEFAULT_GEN3_EQ_LOCAL_FS/4)	0
1	round(DEFAULT_GEN3_EQ_LOCAL_FS/6)	0
2	round(DEFAULT_GEN3_EQ_LOCAL_FS/5)	0
3	round(DEFAULT_GEN3_EQ_LOCAL_FS/8)	0
4	0	0

Preset Index (i)	DEFAULT_GEN3_EQ_POSTCURSOR_PSETi	DEFAULT_GEN3_EQ_PRECURSOR_PSETi
5	0	round(DEFAULT_GEN3_EQ_LOCAL_FS/10)
6	0	round(DEFAULT_GEN3_EQ_LOCAL_FS/8)
7	round(DEFAULT_GEN3_EQ_LOCAL_FS/5)	round(DEFAULT_GEN3_EQ_LOCAL_FS/10)
8	round (DEFAULT_GEN3_EQ_LOCAL_FS/8)	round(DEFAULT_GEN3_EQ_LOCAL_FS/8)
9	0	round(DEFAULT_GEN3_EQ_LOCAL_FS/6)
10	round((DEFAULT_GEN3_EQ_LOCAL_FS - DEFAULT_GEN3_EQ_LOCAL_LF)/2)	0

A.2 Detailed Equalization Procedure

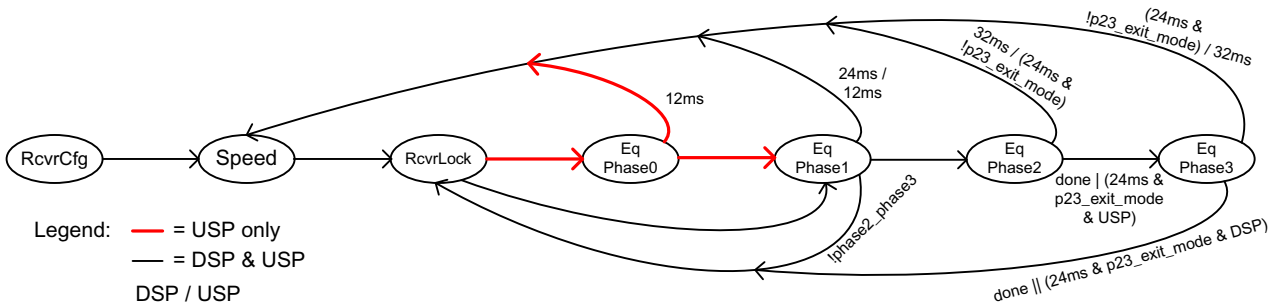
In Section 4.2.3., Link Equalization Procedure for 8.0 GT/s Data Rate (and 4.3.3.5.1., 8.0 GT/s Transmitter Equalization) of the *PCI Express Base Specification, Revision 6.0, Version 1.0*, there is provision for two mechanisms to initiate the equalization procedure: hardware autonomous (strongly recommended) and by software. The following acronyms are used in the following description of the equalization procedure.

- **LF**: Low Frequency.
- **FS**: Full Swing.
- **ecnTS1**: Gen3 or above data rate formatted TS1 with EC[1:0] (Equalization Control field in symbol 6) set to n.
- **eqTS2:** Gen1/2 data rate TS2 with Gen3 equalization presets in symbol 6.
- **8geqTS2**: Gen3 data rate TS2 with Gen4 equalization presets in symbol 7.

A.2.1 Equalization Operations at the PHY Interface

During the equalization procedure the LTSSM moves through a number of different states, as indicated in Figure A-6. The following sections outline the behavior of the interface between the PCIe controller and the PHY in the various equalization states of the LTSSM

Figure A-6 Equalization States in the LTSSM



Programming of Presets

When negotiating to Gen3 speed, before the first speed change to Gen3, the DSP must be programmed with the preset values for its own transmitter and receiver (local presets) and the preset values for the USP (remote presets). The values are independent for each lane and are programmed in the corresponding LEC of the Secondary PCIe Extended capability.

When negotiating to Gen4 speed, before the first speed change to Gen4, the DSP must be programmed with the preset values for its own transmitter and receiver (local presets) and the preset values for the USP (remote presets). The USP can be programmed with the preset values for the transmitter preset values for the DSP (remote presets). The values are independent for each lane and are programmed in the corresponding LEC2 of the Secondary PCIe Extended capability.

A.2.2 Recovery.RcvrCfg

When negotiating to Gen3 speed, before the speed change to Gen3 data rate, each component is required to enter Recovery from L0. In Recovery.RcvrCfg the DSP sends eqTS2s to the USP with the preset fields reflecting the values programmed in the LEC registers. The USP is required to latch the preset fields in received eqTS2s and report the latched values as RO fields in the LEC registers.

When negotiating to Gen4 speed, before the speed change to Gen4 data rate, each component is required to enter Recovery from L0. In Recovery.RcvrCfg the DSP sends 8geqTS2s to the USP with the preset fields reflecting the values programmed in the LEC2 registers. The USP is required to latch the preset fields in received 8geqTS2s and report the latched values as RO fields in the LEC2 registers. If "Upstream Port Send 8GT/s EQ TS2 Disable" (in `GEN3_RELATED_OFF`) for Gen4 is set to 0b, the USP controller sends 8geqTS2s to the DSP with transmitter preset fields reflecting the values programmed in the LEC2 registers. The DSP is required to latch the preset fields in received 8geqTS2s.

A.2.3 Recovery.Speed

In preparation for the speed change to Gen3 or above the PCIe controller asserts `mac_phy_txelecidle` to the PHY and updates `mac_phy_rate[1:0]` to 2'b10(Gen3) or 2'b11(Gen4) to request a data rate change. The request returning `phy_mac_phystatus`. Following `phy_mac_phystatus` the PCIe controller updates the internal data rate variable (used in LTSSM) to match the requested data rate on `mac_phy_rate`.

At this point the PHY asserts valid values on `phy_mac_localfs[5:0]` and `phy_mac_locallf[5:0]`. These values correspond to FS and LF in use by the PHY transmitter. The PCIe controller uses these values to check that the coefficients—corresponding to the preset that must be initially used at Gen3 data rate (obtained from the eqTS2s for USP or from the LEC for DSP) or Gen4 data rate (obtained from the 8geqTS2s for USP/DSP or from the LEC2 for DSP) —are legal according to the rules in chapter 4.2.3.1 of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The same LF and FS values are also advertised through ec1TS1s to the remote link partner during phase 1 of the equalization procedure.

To obtain the coefficients corresponding to the initial preset, the PCIe controller requests the PHY to perform the conversion, by configuring `CX_GEN3_EQ_PSET_COEF_MAP_MODE` to Dynamic PHY. This is done by asserting `mac_phy_getlocal_pset_coef` with `mac_phy_local_pset_index[3:0]` set to the encoding of the preset. When the PHY responds with `phy_mac_local_tx_coef_valid`, the PCIe controller checks that the values on `phy_mac_local_tx_pset_coef[17:0]`are legal, and if legal, it reflects the same values into `mac_phy_txdeemph[17:0]`.

When the values on `phy_mac_local_tx_pset_coef[17:0]`are not legal, the PCIe controller drives `mac_phy_txdeemph[17:0]` with the coefficients that correspond to P4, that is, `{6'b0, phy_mac_localfs[5:0], 6'b0}`. The controller also drives `mac_phy_rxpresethint[2:0]` with the receiver preset Hints obtained from the eqTS2s(for Gen3)/ 8geqTS2s (for Gen4) for USP, or from the LEC (for Gen3) / LEC2(for Gen4) for DSP. The controller uses the same mechanism for "Dynamic MAC" and "Programmable Table", but the `mac_phy_getlocal_pset_coef` / `mac_phy_local_pset_index[3:0]`, and `phy_mac_local_tx_coef_valid` / `phy_mac_local_tx_pset_coef[17:0]` signals are not used. In these modes, the PCIe controller uses internal signals to do the preset-to-coefficients mapping and legality check. The coefficients are encoded as `coefficient[17:0] ={[5:0], [5:0], [5:0]} ={C+1, C0, C-1}` ={post-cursor, cursor, pre-cursor}.

A.2.4 Recovery.RcvrLock

This LTSSM state is entered after Recovery.Speed. Both sides of the link transmit ec0TS1s. When the LTSSM determines that the equalization procedure must be performed, the next state is Recovery.Equalization.

A.2.5 Recovery.Equalization Phase 0

Upstream Port (USP)

The USP transmits ec0TS1s, advertising the coefficients in use by the transmitter, and the preset that are requested through eqTS2s (for Gen3)/ 8gteqTS2s (for Gen4) received from the DSP in Recovery.RcvrCfg at Gen1 or Gen2(for Gen3)/ Gen3(for Gen4) data rate. The remote link partner is in phase 1, and transmits ec1TS1s advertising its own LF and FS. The USP latches LF and FS, and reflects the values into `mac_phy_fs[5:0]`and `mac_phy_lf[5:0]`.

Note: `phy_mac_localfs` and `phy_mac_localllf` represent the LF and FS values of the local transmitter in the USP, but `mac_phy_fs` and `mac_phy_lf` represent the LF and FS values of the remote transmitter.

The PHY can use the remote LF and FS to determine how to adjust the coefficients of the remote transmitter during phase 2. The procedure moves to phase 1 after receiving two ec1TS1s. The PCIe controller asserts the `smlh_ltssm_state_rcvry_eq` signal for the complete equalization state, including EQ phases 0, 1, 2, and 3.

A.2.6 Recovery.Equalization Phase 1

Downstream Port (DSP)

Phase 1 is entered by DSP after Recovery.RcvrLock, when the LTSSM determines that the equalization procedure must be performed. The DSP advertises FS and LF in transmitted ec1TS1s, and (similar to phase 0 for USP) latches LF and FS values from received ec1TS1, and reflects the values into `mac_phy_fs[5:0]` and `mac_phy_lf[5:0]`. The PHY can use these values to determine how to adjust the coefficients of the remote transmitter during phase 3. The procedure moves to phase 2 after receiving two ec1TS1s.

Upstream Port (USP)

Equalization phase 1 is entered by USP after phase 0. During phase 1 the USP advertises FS and LF in transmitted ec1TS1s. The procedure moves to phase 2 when the LTSSM determines that equalization phase 2 and phase 3 must be performed.

A.2.7 Recovery.Equalization Phase 2

Upstream Port (USP)

During phase 2, the USP can request and evaluate a number of different settings in the remote transmitter of the DSP. Phase 2 ends successfully when all lanes reach optimal settings in the remote transmitter within 24ms, otherwise phase 2 ends unsuccessfully. After phase 2 is completed successfully, the next state is:

- Recovery.Speed: if there is a 24ms timeout and the Behavior After 24 ms Timeout field of the Gen3 EQ Control Register `GEN3_EQ_CONTROL_OFF` is 0, else
- Equalization phase 3: Before exiting to Equalization phase 3, the port waits 24ms timeout if current data rate is Gen4 and a Retimer has been detected in Configuration.Complete.

The sequence of operations performed by the PCIe controller as a USP in phase 2 is shown in Figure A-7. The following is a description of the operations in each state of the flowchart performed by the PCIe controller (as USP) in phase 2. The same operations are performed in phase 3 when the PCIe controller is the DSP.

Note

Selectable FOM Gen3 EQ Master Mode

You can select how the controller asserts mac_phy_rxeqeval (to instruct the PHY to do Rx adaptation) using the RXEQ_RGRDLESS_RXTS field in the GEN3_RELATED_OFF register:

- **0 (Legacy Mode)**: `mac_phy_rxeqeval` asserted after 1us and 2 TS1 received from remote partner.
- **1 (Rgardless_RxTS Mode)**: `mac_phy_rxeqeval` asserts after 500ns regardless of TS's received or not.

- RST: this is the default state. Entered when the LTSSM is not in phase 2. The PCIe controller resets to the inactive state (zero) all these phase 2 specific inputs to the PHY: `mac_phy_rxeqeval`, `mac_phy_invalid_req`, `mac_phy_rxeqinprogress`.
- **INIT**: the controller asserts `mac_phy_rxeqinprogress`, latches the preset and coefficients in use on the remote side from the received ec2TS1s, and initializes the internal registers containing the preset and coefficients to be evaluated by the local receiver.If `EQ_EIEOS_CNT` field of `GEN3_RELATED_OFF` Register is set to '1', the next state is EVAL, else the next state is WAIT_REI.
- **Wait_REI**:

  Rgardless_RxTS Mode: The next state is EVAL, as the controller asserts `mac_phy_rxeqeval` regardless of the fact that RX TS1s are detected or not. The REI bit is set to '1' only when any lane asserts `mac_phy_rxeqeval`.

  Legacy Mode: If all active lanes are sending TS1s with REI bit set to '1' or there is a 2ms timeout, the next state is EVAL. The condition that all active lanes are sending TS1s with REI bit set to '1' is dependent on whether all the active lanes achieve Block Alignment after the remote partner applies the new EQ settings to its transmitter or not. This is done to filter out scenarios which fail EQ evaluation.

  For example, if the controller detects TS1s on any lane and sends TS1s with REI bit set to the remote partner, the remote partner immediately sends an EIEOS after 65536 TS1s across all lanes; as a result of this, lanes without Block Alignment may not obtain Block Alignment for the new EQ settings.

  If `EQ_EIEOS_CNT` field of `GEN3_RELATED_OFF` Register is set to '0' during PHY's evaluation of active lanes losing Block Alignment (`phy_mac_rxvalid` =0), the controller transmits TS1s with REI bit reset to '0' on all active lanes so that the PHY can obtain Block Alignment again.
- **EVAL**:

  Rgardless_RxTS Mode: After 500ns timeout since new request the controller asserts `mac_phy_rxeqeval` to instruct the PHY to evaluate the current remote transmitter settings. After all lanes involved have indicated completion of the evaluation cycle with `phy_mac_phystatus`, the controller de-asserts `mac_phy_rxeqeval` immediately. If all lanes involved detected TS1s after 1us timeout since new request or 2ms timeout the next state is FEEDBACK.

  Legacy Mode: the controller immediately asserts `mac_phy_rxeqeval`. After all involved lanes have received `phy_mac_phystatus` the next state is FEEDBACK.

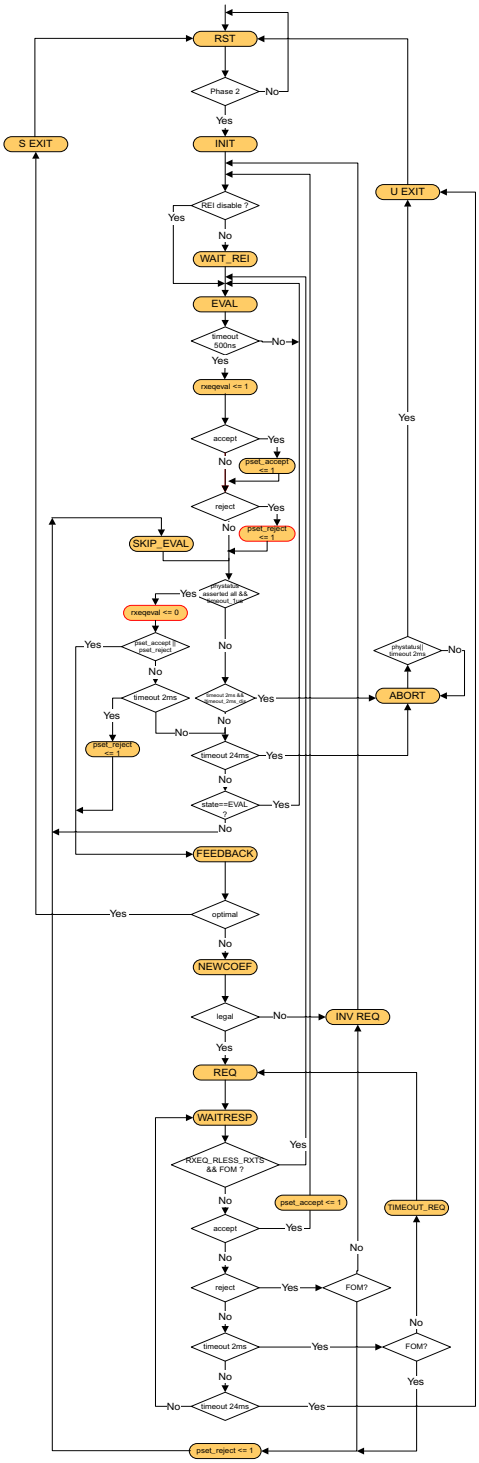
  All modes: during the evaluation cycle if any lane involved does not received `phy_mac_phystatus` after 2ms timeout and `GEN3_EQ_EVAL_2MS_DISABLE` field of `GEN3_EQ_CONTROL_OFF` Register is set to ‘0’, or 24ms timeout, the next state is ABORT.

  Depending upon the cause of transition to EVAL state, the following action should be taken with respect to coefficients:

  - If the EVAL state is entered because of a valid request (accepted by remote partner), then latch the coefficients in use by remote transmitter.

- If the EVAL state is entered because of an invalid request (rejected by remote partner) or 2ms timeout (no reject or accept is received), then do not latch the coefficients. Latching the coefficients causes `phy_mac_dirfeedback[5:0]` to generate new coefficients based on valid coefficients during transition from the final preset request to the first coefficients request.

Figure A-7 EQ Master Flowchart Showing the Sequence of Operations in Phase 2 when PCIe Controller is USP



- **FEEDBACK**: the controller de-asserts `mac_phy_rxeqeval`. When the Preset Request Vector field of the Gen3 EQ Control Register `GEN3_EQ_CONTROL_OFF` has any bits set, the LTSSM moves to NEWCOEFF and executes loops so that each preset in the Preset Request Vector field is requested and evaluated. When all presets are requested and evaluated, then a final preset with the highest FOM from `phy_mac_fomfeedback[7:0]` is requested.
  - If the final preset request is rejected by the remote partner, and if the feedback mode is FOM, then the next state is EXIT (with unsuccessful status), else if the final preset request is accepted by the remote partner, next state is EXIT (with successful status).
  - When the Preset Request Vector field is 0 (no preset request) or the feedback mode is Direction Change after all preset requests, then the controller evaluates the feedback provided by the PHY on `phy_mac_dirfeedback`. It then determines from that feedback, if the current settings of the remote transmitter are optimal. When they are optimal, the next state is EXIT (with successful status), otherwise the next state is NEWCOEF.
- **NEWCOEF**:
  - For preset requests, the requested preset value is determined by right-shifting the 16-bit Preset Request Vector field of the Gen3 EQ Control Register (`GEN3_EQ_CONTROL_OFF`) until bit[0] is 1. This right-shifted number becomes the requested preset value. If the preset is found, the next state is REQ.
  - For coefficients requests, new coefficients for evaluation are calculated by applying the direction change feedback to the current coefficients under evaluation. If the results satisfies the rules from 4.2.3.1 in the *PCI Express Base Specification, Revision 6.0, Version 1.0*, the next state is REQ, else it is INV REQ.
- **REQ:** the PCIe controller transmits ec2TS1s (preset/coefficients request by setting Use Preset bit to 1/0) to request the remote link partner to change the transmitter settings to the new preset or coefficients calculated in the previous state. Next state is WAITRESP.
- **WAITRESP**:

  Rgardless_RxTS: The next state is EVAL where the controller asserts `mac_phy_rxeqeval` after 500ns timeout and start detecting TS1s after 1us timeout since new preset request.

  Legacy Mode: The controller monitors the received ec2TS1s (after 1us timeout since new request), to determine if the remote link partner has accepted the request to change its transmitter settings.
  - If the request is accepted and `EQ_EIEOS_CNT` field of `GEN3_RELATED_OFF` Register is set to '1', the next state is EVAL.
  - If the request is accepted and `EQ_EIEOS_CNT` field of `GEN3_RELATED_OFF` Register is set to '0', the next state is WAIT_REI.
  - If the request is rejected and it is a FOM request, the next state is SKIP_EVAL.
  - If the request is rejected and it is a DIR request, the next state is INV REQ.
  - If request is FOM after a 2ms timeout the next state is SKIP_EVAL.
  - If request is DIR after 2ms timeout the next state is TIMEOUT_REQ.
  - Otherwise after 24ms timeout the next state is EXIT with unsuccessful status.
- **SKIP_EVAL**: this state does not assert `mac_phy_rxeqeval` for the PHY to perform evaluation, as the requested preset is rejected or does not work on the lane. The next state is FEEDBACK to seek the next new preset after all active lanes (excluding the SKIP_EVAL lanes) complete evaluation. Otherwise, after 24ms timeout the next state is EXIT with unsuccessful status.
- **TIMEOUT_REQ**: this state is used to prepare the request needed to restore the last successful coefficients into the remote transmitter. The next state is REQ.

- **INV REQ**: this state is used to assert mac_phy_invalid_req to indicate to the PHY, that the feedback on the latest evaluation generated an invalid set of coefficients for the remote link partner.

  The next state is EVAL if `EQ_EIEOS_CNT` field of `GEN3_RELATED_OFF` Register is set to '1'; Else, the next state is WAIT_REI. The evaluation on the current set of coefficients is repeated (the PCIe controller does not request through ec2TS1s any change to the remote transmitter settings in this case) to give the PHY an opportunity to provide a different feedback on the same set of coefficients.
- **ABORT**: this state is entered when the controller needs to abort an evaluation cycle (because of 2ms[1] or 24ms timeout while `mac_phy_rxeqeval` is asserted, and the PHY has not yet terminated the evaluation with `phy_mac_phystatus`).The controller asserts the `mac_phy_rxeqeval` signal and then waits for the PHY to acknowledge with `phy_mac_phystatus`. The next state is EXIT with unsuccessful status, after the PHY responds on `phy_mac_phystatus` or a timeout.

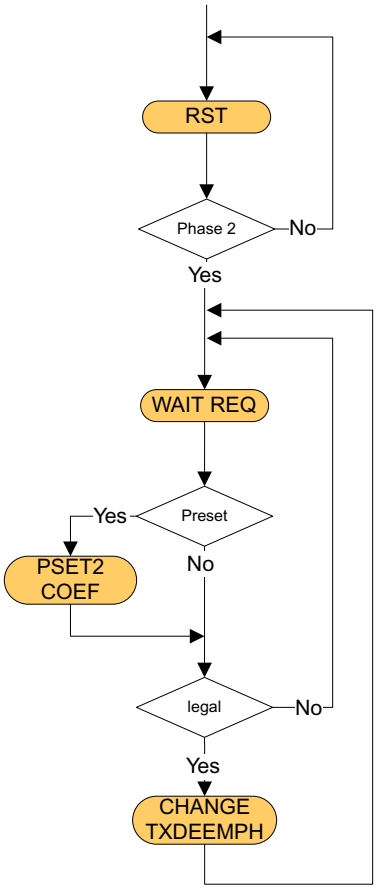
Downstream port (DSP)

During phase 2, the DSP receives requests from the USP to change the settings in its local transmitter. Requests are encoded within the received ec2TS1s in the form of presets or coefficients. To obtain the coefficients corresponding to the requested preset, the controller requests the PHY, MAC, or Programmable Table to perform the conversion, in a way similar to the protocol already described for the conversion of the initial preset in Recovery.Speed. When the coefficients for the request are available, the controller uses the local FS and LF (`from phy_mac_localfs`[5:0] and `phy_mac_locallf[5:0]`) to check that the coefficients are legal according to the rules in "4.2.3.1 Rules for Transmitter Coefficients" section of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. If the coefficients are legal the controller drives `mac_phy_txdeemph[17:0]`accordingly, else `mac_phy_txdeemph` is left unchanged. The process is repeated at every new request. The sequence of operations performed by the controller as a DSP in phase 2 is shown in Figure A-8. The following is a description of the operations in each state of the flowchart performed by the controller. The same operations are performed in phase 3 when the controller is the USP.

- **RST**: This is the default state, entered when the LTSSM is not in phase 2.
- WAIT **REQ**: the PCIe controller monitors received ec2TS1s for a request to change the transmitter settings.
- **PSET2COEF**: For Dynamic PHY mapping mode, the controller asserts `mac_phy_getlocal_pset_coef` with `mac_phy_local_pset_index[3:0]`set to the encoding of the preset, and waits until the PHY responds with `phy_mac_local_tx_coef_valid`. For Dynamic MAC mapping mode, the controller calculates the coefficients from received preset internally. For Programmable Table mapping mode, the controller picks the programmed coefficients mapped to the corresponding preset.
- **CHANGE TXDEEMPH**: controller changes `mac_phy_txdeemph[17:0]` to apply the new settings to PHY

1. GEN3_EQ_EVAL_2MS_DISABLE field of the GEN3_EQ_CONTROL_OFF register is set to '0'; its default value is '1'.

Figure A-8 EQ Slave Flowchart Showing Sequence of Operations in Phase 2 when PCIe Controller is DSP



A.2.8 Recovery.Equalization Phase 3 (DSP/USP)

Phase 3 is similar to phase 2, with the DSP taking the role of the USP, and the USP taking the role of the DSP.

A.3 Usage Example

The section provides an example for setting up Gen3 or above EQ. It also describes the sequence of events that the controller progresses through.

EQ Master

- Hardware Configuration
  - `CX_GEN3_EQ_COEF_CONV_SUPPORTED =1`
- Gen3 EQ Register Programming
  - Gen3 Control Register bit [13] = 1'b0 (Legacy Mode, not *Rgardless_RxTS Mode*)
  - Gen3 EQ Control Register bit [3:0] =4b0000 (Direction Change Feedback Mode)
  - Gen3 EQ Control Register bit [23:8] =16b000000000100100 (Preset Request Vector). Presets 2 and 5 is requested and evaluated in terms of the preset vector.
  - Gen3 EQ Control Register bit [24] =1'b1 (the initial preset evaluation FOM feedback to be included for finding the highest FOM).
  - Gen3 EQ Direction Change Feedback Mode Control Register bit [4:0] =5b01010 (minimum time to check convergence is 10ms)
  - Gen3 EQ Direction Change Feedback Mode Control Register bit [9:5] =5b00011 (number of consecutive coefficients evaluations is 3)
  - Gen3 EQ Direction Change Feedback Mode Control Register bit [13:10] =4b0001 (pre-cursor maximum depth (delta) is 1 within the convergence window)
  - Gen3 EQ Direction Change Feedback Mode Control Register bit [17:14] =4b0001 (post-cursor maximum depth (delta) is 1 within the convergence window)

Note The Gen3 EQ Register Programming is provided as an example, for the controller programming sequence. These do not apply to Synopsys PHY.

- Requests and Evaluations for one Lane

a. Assuming the latched `mac_phy_fs` and `mac_phy_lf` values transmitted by the remote partner in EQ phase1 are 6h2b and 6h12.

b. When the controller enters EQ master, the state machine shown in Figure A-7 moves to INIT. `mac_phy_rxeqinprogress` =1, assuming the latched preset =4b0001 and coefficients =18h0b800 transmitted initially by the remote partner. Next state is EVAL.

c. In EVAL, `mac_phy_rxeqeval` =1 for PHY to do evaluation for the initial preset 4b0001, 2ms timer kicks off, waiting for `phy_mac_phystatus` asserted back. Assuming `phy_mac_phystatus` is asserted back with `phy_mac_fomfeedback[7:0]` =8d12 within 2ms, the controller moves to FEEDBACK.

d. In FEEDBACK, `mac_phy_rxeqeval =0` to terminate the PHYs evaluation. Next state is NEWCOEFF.

e. In NEWCOEFF, latch FOM =8d12 and its preset =4b0001. Right-shifting the preset vector until its bit [0] =1 to find out the first requested preset value 2, and then move to REQ.

f. In REQ, set Use Preset bit to 1 and preset to 2 into the proper fields of the transmitting TS1s. Send TS1s to the remote partner, reset and start the 2ms timer. Next state is WAITRESP.

g. In WAITRESP, assuming accept is received in the response TS1s from the remote partner within 2ms. Next state is EVAL.

h. In EVAL, `mac_phy_rxeqeval =1`for PHY to do evaluation for the requested preset value 2, waiting for `phy_mac_phystatus` asserted back. Assuming `phy_mac_phystatus` is asserted back with `phy_mac_fomfeedback[7:0] =8d188` within 2ms, the controller moves to FEEDBACK and then to NEWCOEFF.

i. In NEWCOEFF, latch FOM =8d188 and its corresponding preset 2, as 188 is larger than 12 (you must find the preset with highest FOM). Then find out the second requested preset 5 and repeat steps f, g, and h. Assuming `phy_mac_fomfeedback[7:0] =8d66` for the last request preset 5. As `phy_mac_fomfeedback =66` is not the highest FOM, you must to do one more preset request, that is, final preset request for the preset 2 again as the preset 2 has highest FOM =8d188. After WAIT-RESP, next state is EVAL.

j. In EVAL, `mac_phy_rxeqeval =1,` latch coefficients received in TS1s from the remote partner that are the coefficients in use by the remote partner. Assuming the latched coefficients= 18h00f80. After receiving `phy_mac_phystatus` from PHY with `phy_mac_dirfeedback =6b010101`, move to FEEDBACK to do coefficients requests. All the preset requests are completed.

k. In FEEDBACK, `mac_phy_rxeqeval =0.` Cannot get optimal settings as the programmed number of consecutive coefficients evaluations is 3. So far in the coefficients evaluation process, it is 0. Next state is NEWCOEFF.

l. In NEWCOEFF, need to calculate new coefficients. The current coefficients are =18h00f80, then C-1 =6b000000, C0 =6b111110, C+1 =6b000000. Because `phy_mac_dirfeedback` =6b010101, so the new coefficients are C-1 =6b000001, C+1 =6b000001, C0 =(`mac_phy_fs` C-1 C+1) =(6h2b 1 1) =6h29. The new coefficients =18h01a41. Now check the new coefficients legality ("4.2.3.1 Rules for Transmitter Coefficients" section of the *PCI Express Base Specification, Revision 6.0, Version 1.0*).

Note: C0 is calculated, so ignore `phy_mac_dirfeedback[3:2]`.

- |C-1| =1 <= Floor (FS/4) =Floor (43/4) =10. So it is legal.
- |C-1| + C0 + |C+1| =43 =FS =43. So it is legal.
- C0 - |C-1| - |C+1| =41 >= LF =6h12 =18. So it is legal

Next state is REQ.

m. In REQ, reset and start 2ms timer. Set Use Preset bit to 0 and place the new coefficients C-1, C0 and C+1 into the correct TS1 fields, and send TS1s to the remote partner. Next state is WAITRESP.

n. In WAITRESP, assuming accept is received in the response TS1s from the remote partner within 2ms. Next state is EVAL.

o. In EVAL, `mac_phy_rxeqeval =1,` the number of evaluation increments by 1 and push C-1 =1 and C+1 =1 into coefficient queues. Waiting for `phy_mac_phystatus` back from PHY. Assuming `phy_mac_phystatus` is asserted back with `phy_mac_dirfeedback =6b010001`within 2ms, then move to FEEDBACK and repeat steps k, l, m, and n.

p. After re-enter EVAL, `mac_phy_rxeqeval =1.` The number of consecutive evaluations is 2 and C-1 =2 and C+1 =2 are pushed into the queues. Waiting for `phy_mac_phystatus` back from PHY. Assuming `phy_mac_phystatus` is asserted back with `phy_mac_dirfeedback =6b011110`within 2ms, then move to FEEDBACK and repeat steps k, l, m, and n.

q. After re-entering EVAL, `mac_phy_rxeqeval =1,` and the number of consecutive evaluations is 3, and C-1 =1 and C+1 =3 are pushed into queues. Waiting for `phy_mac_phystatus` back from PHY. Assuming `phy_mac_phystatus` is asserted back with `phy_mac_dirfeedback =6b101000` within 2ms, then move to FEEDBACK.

r. In FEEDBACK, `mac_phy_rxeqeval =0,` assuming the time spent on the phase in the EQ master so far is 9ms, you must find out if the optimal settings are reached as the number of consecutive evaluations reaches 3 (the programmed number of consecutive coefficients evaluations).

   i. For C-1, three coefficients (1, 2, 1) are pushed into its queue. So the delta for C-1 is (2-1) =1.

   ii. For C+1, three coefficients (1, 2, 3) are pushed into its queue.So the delta for C+1 is (3-1) =2.

   Because the programmed maximum delta for C-1 and C+1 is 1, the delta 2 in the queue for C+1 window does not satisfy the programmed maximum delta. In addition, the time for the convergence check does not satisfy the programmed minimum time 10ms because it is 9ms by now. So the controller can proceed with the next request. Repeat l, m, and n.

s. After re-entering EVAL, mac_phy_rxeqeval =1, the number of consecutive evaluations is 3 and C-1 =1 and C+1 =2 are pushed into queues. Waiting for `phy_mac_phystatus` back from PHY. Assuming `phy_mac_phystatus` is asserted back with `phy_mac_dirfeedback =6b010110`within 2ms, then move to FEEDBACK.

t. In FEEDBACK, `mac_phy_rxeqeval =0`, assuming the time spent on the phase in the EQ master so far is 10.5ms. You must find out if the optimal settings are reached as the number of consecutive evaluations reaches 3.

   i. For C-1, three latest coefficients (2, 1, 1) are pushed into its queue. So the delta for C-1 is (2-1) =1.

   ii. For C+1, three latest coefficients (2, 3, 2) are pushed into its queue. So the delta for C+1 is (3-2) =1.

   Because the programmed maximum delta for C-1 and C+1 is 1, the delta in the both queues for C-1 and C+1 window does satisfy the programmed maximum delta. The time for the convergence check also satisfies the programmed minimum time of 10ms because it is 10.5ms by now. Next state is EXIT.

u. In EXIT, `mac_phy_rxeqinprogress =0,` the successful status bit for the phase is set to 1. The PCIe controller moves to the phase3 for USP or Recovery.RcvrLock for DSP.

EQ Slave

- Hardware Configuration
  - `CX_GEN3_EQ_PSET_COEF_MAP_MODE` =0 (Dynamic PHY)
- Mapping and Applying

a. Assuming `phy_mac_localfs =6'h3e` and `phy_mac_locallf =6'h11` are already set before the port changes to Gen3 or above rate.

b. When the controller enters EQ slave, the state machine shown in Figure A-8 moves to WAITREQ state, transmitter sends TS1s with the preset and coefficients in use. If two consecutive TS1s with EC[1:0] =2b11 for USP or EC[1:0] =2b10 for DSP have been received in the beginning of the phase, or two consecutive TS1s (with EC[1:0] =2b11 for USP or EC[1:0] =2b10 for DSP) with a preset (Use Preset bit =1) different from the last two TS1s with the same EC[1:0] have been received, next state is PSET2COEF.

c. In PSET2COEF, assuming the received request preset =4b0011, the PCIe controller asserts `mac_phy_getlocal_pset_coef` for one cycle with `mac_phy_local_pset_index[3:0] =4b0011` to PHY. Waiting for coefficients mapped back from PHY. A few cycles later, PHY completes mapping and asserts `phy_mac_local_tx_coef_valid` back with `phy_mac_local_tx_pset_coef =18h0a885`(C-1 =6b000101, C0 =6b100010, C+1 =6b001010) for one cycle. Next state is CHANGETXDEEMPH.

d. In CHANGETXDEEMPH, the controller needs to check the coefficients legality ("4.2.3.1 Rules for Transmitter Coefficients" section of the *PCI Express Base Specification, Revision 6.0, Version 1.0*).

- |C-1| =5 <= Floor (FS/4) =Floor (62/4) =15. It is legal.
- |C-1| + C0 + |C+1| =(5 + 34 + 10) =49 !=FS =62. It is illegal.

So `phy_mac_local_tx_pset_coef =18h0a885` is not supplied to `mac_phy_txdeemph[17:0]`because the coefficients are illegal (invalid). `mac_phy_txdeemph[17:0]` is left unchanged. The EQ slave sends EC TS1s with reject and preset =4b0011 in the TS1 fields back to the remote partner. The remote partner receives the reject and can do the next preset request with preset =4b1001. The EQ slave receives two EC TS1s with preset =4b1001 different from the last preset =4b0011, moves to PSET2COEFF.

e. In PSET2COEF, the PCIe controller asserts `mac_phy_getlocal_pset_coef` for one cycle with `mac_phy_local_pset_index[3:0] =4b1001`to PHY. Waiting for coefficients mapped back from PHY. A few cycles later, PHY completes mapping and asserts `phy_mac_local_tx_coef_valid` for one cycle with `phy_mac_local_tx_pset_coef =18h06c08` (C-1 =6b001000, C0 =6b110000, C+1 =6b000110) back. Next state is CHANGETXDEEMPH.

f. In CHANGETXDEEMPH, the controller checks the coefficients legality first.

- |C-1| =8 < =Floor (FS/4) =Floor (62/4) =15. It is legal.
- |C-1| + C0 + |C+1| =(8 + 48 + 6) =62 =FS =62. It is legal.
- C0 - |C-1| - |C+1| =(48 - 8 - 6) =34 >=LF =17. It is legal.

So `phy_mac_local_tx_pset_coef =18h06c08` is supplied to `mac_phy_txdeemph[17:0]`because it satisfies the three rules. PHY detects the change on `mac_phy_txdeemph[17:0]`and applies it to its transmitter. The EQ slave sends EC TS1s with accept and preset =4b1001 in the TS1 fields back to the remote partner. The remote partner receives the accept and starts to do evaluation. After evaluation, the remote partner might not reach optimal settings and can continue the next preset request or coefficient request until reaching optimal settings. If it is preset request, repeat steps b-to-f. If the remote partner reaches optimal settings, it moves to the next EQ phase and send TS1s with `EC[1:0] =2b11` for USP or `EC[1:0] =2b00` for DSP. The EQ slave receives TS1s with `EC[1:0] =2b11` for DSP or `EC[1:0] =2b00` for USP and moves to Recovery state for USP or EQ phase 3 for DSP. The successful status is set to 1 for this phase.

Hint

You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html`for more programming examples. For more information on VTB, see the Integrating the Controller and VIP using VTB section in the "Integration" chapter of the *User Guide*.

B

Advanced Information: Lane Reversal and Broken Lanes

This appendix illustrates how the controller uses the lane reversal and lane flip functions to form links and gives several example using an x8 PCIe controller with an x8 or x4 link partner.

Figure B-1 Reversing and Flipping Architecture

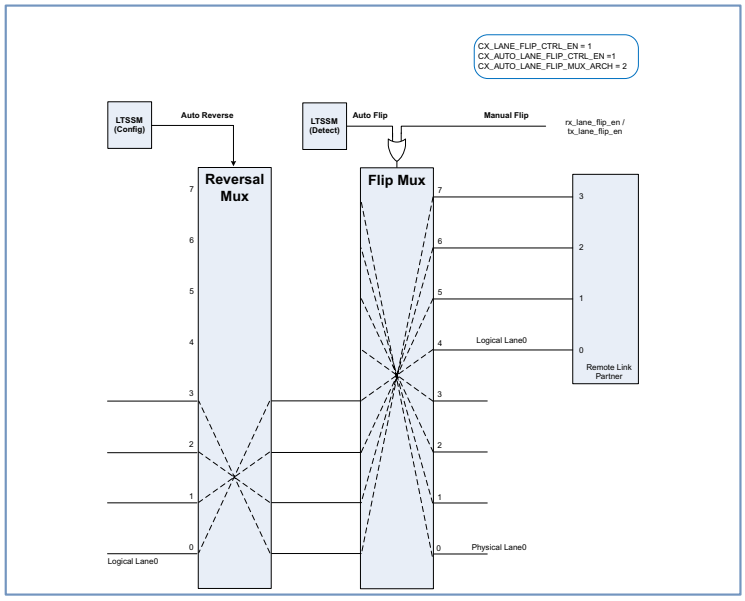


Table B-1 Lane Reversal and Lane Flipping Configuration Options (All Enabled by Default)

Parameter	Muxes Instantiated	Notes
CX_AUTO_LANE_FLIP_CTRL_EN=1 CX_AUTO_LANE_FLIP_MUX_ARCH =0	1	■ Can support a full reversal or a full flip; but not both simultaneously.
CX_AUTO_LANE_FLIP_CTRL_EN=1 CX_AUTO_LANE_FLIP_MUX_ARCH =1	1	■ Can support a full or partial reversal, or a full or partial flip; but not both simultaneously.

Parameter	Muxes Instantiated	Notes
CX_AUTO_LANE_FLIP_CTRL_EN=1 CX_AUTO_LANE_FLIP_MUX_ARCH =2	2	■ Can support a full or partial reversal, and a full or partial flip; but not both simultaneously. ■ This is the default for new configurations.
Note: ■ Lane0 always refers to physical lane0, as determined by its location in the PIPE interface which is [7:0] or [15:0]. ■ Full flip is when Logical Lane0 is connected to Lane NL-1. ■ Partial flip is when Logical Lane0 is connected to either Lane NL/2-1, or Lane NL/4-1, or Lane NL/8-1, and so on.		
Note: To use any of these features you must set the configuration parameter CX_LANE_FLIP_CTRL_EN=1 and the AUTO_LANE_FLIP_CTRL_EN field in GEN2_CTRL_OFF =1.		
Note: Flip: When the LTSSM transitions to DETECT_WAIT, meaning that some lanes are detected in DETECT_ACT, but not all; and lane0 is not detected, the LTSSM autonomously activates a lane flip operation.		
Note: Reversal: When the LTSSM transitions to Configuration state and Logical Lane0 receives TS Ordered Sets with lane number different from '0', the LTSSM automatically performs a lane reversal.		
Note: To use this feature you must set the PRE_DET_LANE field in GEN2_CTRL_OFF to 0. When its value is '0, auto-flip is based on detected lanes and auto-reversal is based on Lane0's lane number in the received TS Ordered Sets. When its value is not '0', auto-lane multiplexer is based on the programmed value (only one instantiated multiplexer).		

Connection to x2 and x1 link partners work similarly. For more information, see "Link Establishment" on page 115. It is assumed that you have set the following configuration parameters and register field:

- `CX_LANE_FLIP_CTRL_EN` =1
- `CX_AUTO_LANE_FLIP_CTRL_EN` =1
- `CX_AUTO_LANE_FLIP_MUX_ARCH =2`
- `AUTO_LANE_FLIP_CTRL_EN` =1 in `GEN2_CTRL_OFF` register
- `PRE_DET_LANE =0` in `GEN2_CTRL_OFF` register

Some of the case examples in this appendix require the programming of the Predetermined Lane for Auto Flip field (`PRE_DET_LANE`) in the `GEN2_CTRL_OFF` port logic register. This field defines which physical lane is connected to logical Lane0 by the flip operation performed in Detect.

When its value is '0, auto-flip is based on detected lanes and auto-reversal is based on lane0's lane number in the received TS Ordered Sets. When its value is not '0', auto-lane multiplexer is based on the programmed value (only one instantiated multiplexer).

Allowed values are:

- 3'b000: Connect logical Lane0 to physical lane 0 or `CX_NL`-1 or `CX_NL/2`-1 or `CX_NL/4`-1 or `CX_NL/8`-1, depending on which lane is detected
- 3'b001: Connect logical Lane0 to physical lane 1
- 3'b010: Connect logical Lane0 to physical lane 3
- 3'b011: Connect logical Lane0 to physical lane 7
- 3'b100: Connect logical Lane0 to physical lane 15

This field is used to restrict the receiver detect procedure to a particular lane when the default detect and polling procedure performed on all lanes cannot be successful. A notable example of when it is useful to program this field to a value different from the default is when a lane is asymmetrically broken, that is, it is detected in Detect LTSSM state but it cannot exit Electrical Idle in Polling LTSSM state. In this case only x1 link can be formed.

Note If CX_AUTO_LANE_FLIP_CTRL_EN =0 or CX_AUTO_LANE_FLIP_MUX_ARCH =0, the cases in this appendix illustrating partial flipping cannot be supported.

- Normal Link Establishment with Non-reversed/Reversed Lanes; and Wide/Narrow Ports
  - "x8 to x8 (Non-reversed Lanes)" on page 652
  - "x8 to x8 (Reversed Lanes)" on page 652
  - "x8 to x4-Bottom (Non-reversed Lanes)" on page 653
  - "x8 to x4-Bottom (Reversed Lanes)" on page 653
  - "x8 to x4-Top (Reversed Lanes)" on page 654
  - "x8 to x4-Top (Reversed Lanes)" on page 654
- Link Down-sizing with Non-reversed/Reversed Lanes; and Wide/Narrow Ports
  - "x8 to x8 (Non-reversed Lanes) with Downsize to x4" on page 655
  - "x8 to x8 (Reversed Lanes) with Downsize to x4" on page 655
  - "x8 to x4-Bottom (Non-reversed Lanes) with Downsize to x2" on page 656
  - "x8 to x4-Bottom (Reversed Lanes) with Downsize to x2" on page 656
  - "x8 to x4-Top (Non-reversed Lanes) with Downsize to x2" on page 657
  - "x8 to x4-Top (Reversed Lanes) with Downsize to x2" on page 657
- Broken Lane Operation
  - "x8 to x4-Bottom (Lane0 Broken in Detect)" on page 658
  - "x8 to x8 (Lane0 Asymmetrically Broken: OK in Detect; Broken in Polling)" on page 658
  - "x8 to x4-Bottom (Lane0 Asymmetrically Broken: OK in Detect; Broken in Polling)" on page 659
  - "x8 to x2 Partial Lane Flip and Partial Lane Reversal" on page 660

Hint You can also inspect the VTB test cases at `<workspace>/doc/html/vtb/testenv/index.html`. For more information on VTB, see "Integrating the Controller and VIP using VTB" section in the "Integration" chapter of the *User Guide*.

B.1 Normal Link Establishment with Non-reversed/Reversed Lanes; and Wide/Narrow Ports

Figure B-2 x8 to x8 (Non-reversed Lanes)

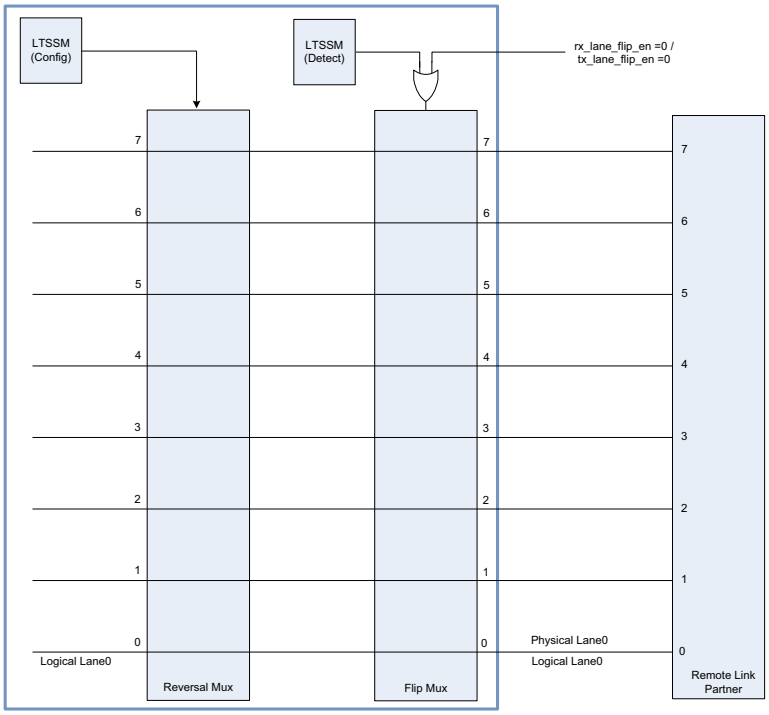


Figure B-3 x8 to x8 (Reversed Lanes)

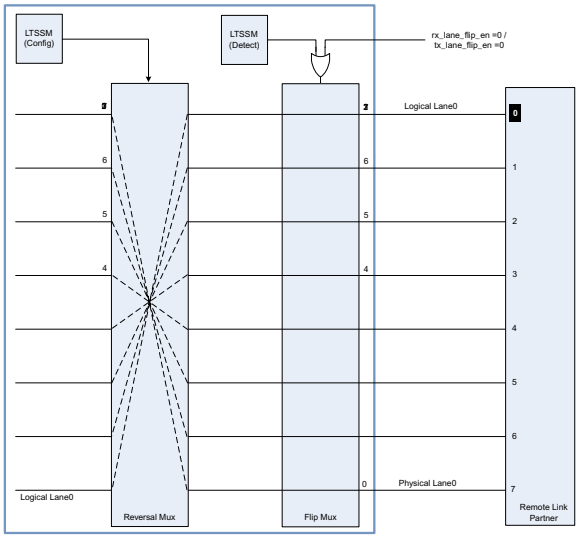


Figure B-4 x8 to x4-Bottom (Non-reversed Lanes)

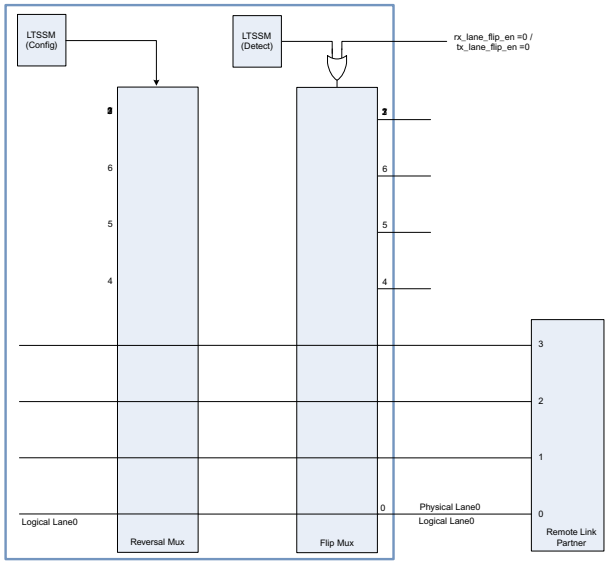


Figure B-5 x8 to x4-Bottom (Reversed Lanes)

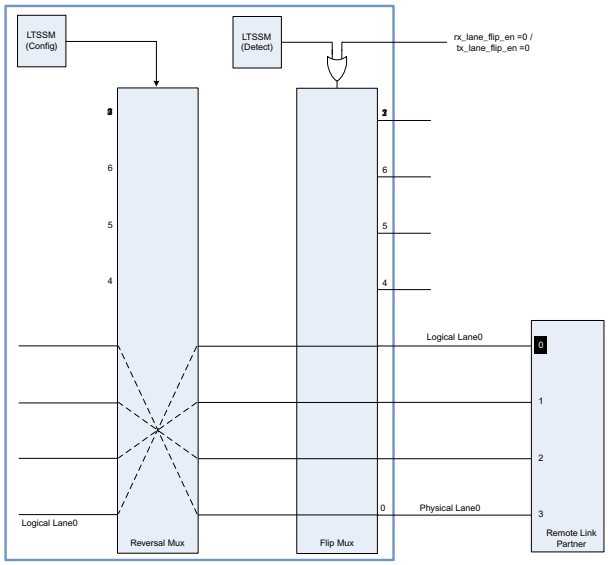


Figure B-6 x8 to x4-Top (Reversed Lanes)

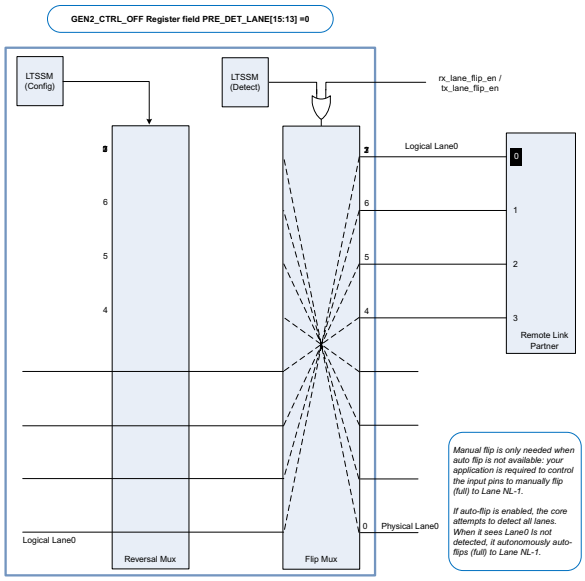
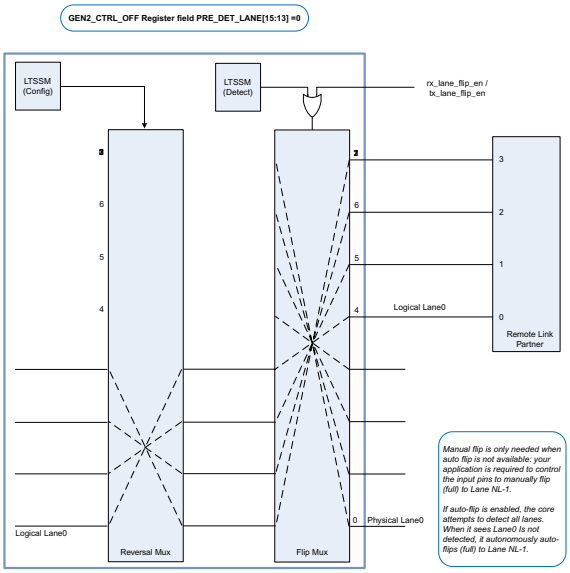


Figure B-7 x8 to x4-Top (Reversed Lanes)



B.2 Link Down-sizing with Non-reversed/Reversed Lanes; and Wide/Narrow Ports

Figure B-8 x8 to x8 (Non-reversed Lanes) with Downsize to x4

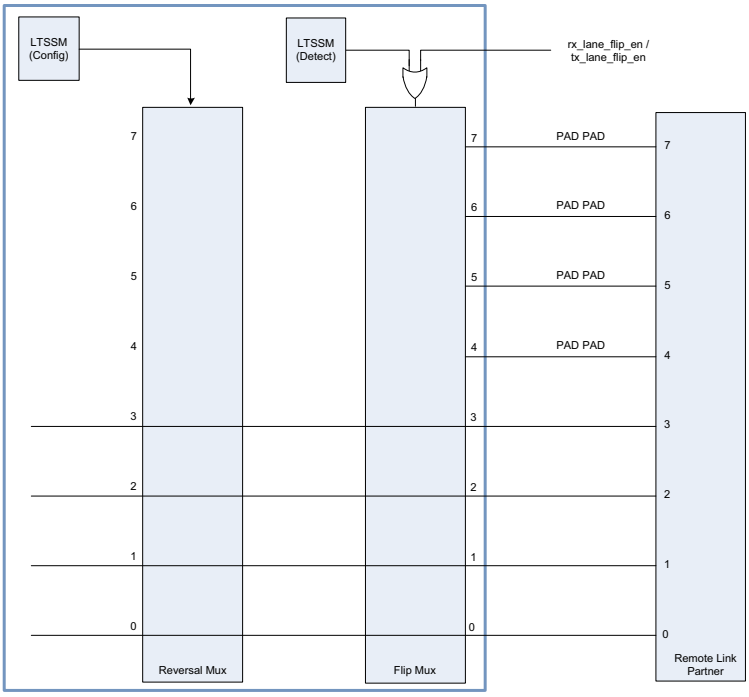


Figure B-9 x8 to x8 (Reversed Lanes) with Downsize to x4

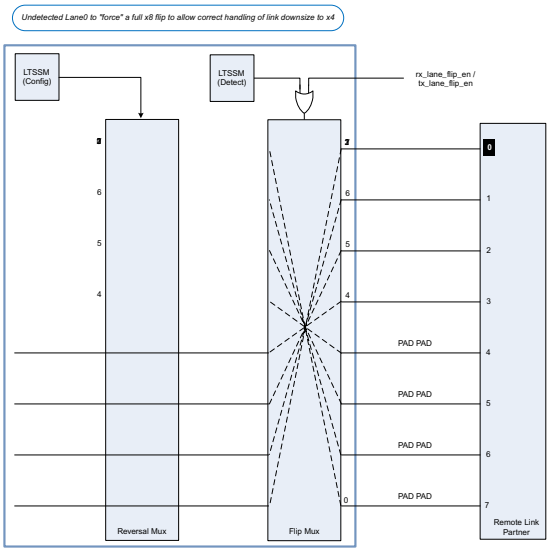


Figure B-10 x8 to x4-Bottom (Non-reversed Lanes) with Downsize to x2

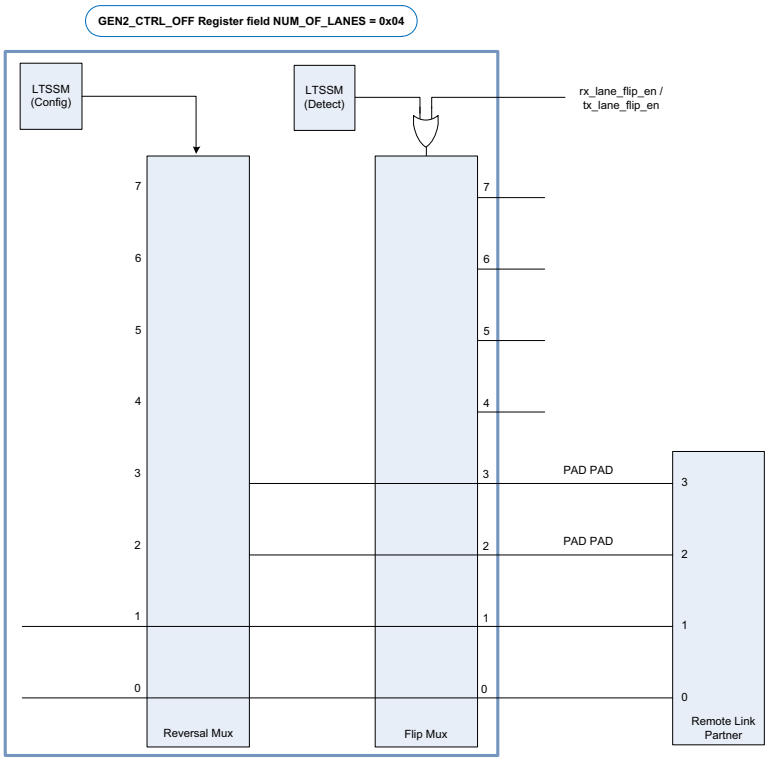


Figure B-11 x8 to x4-Bottom (Reversed Lanes) with Downsize to x2

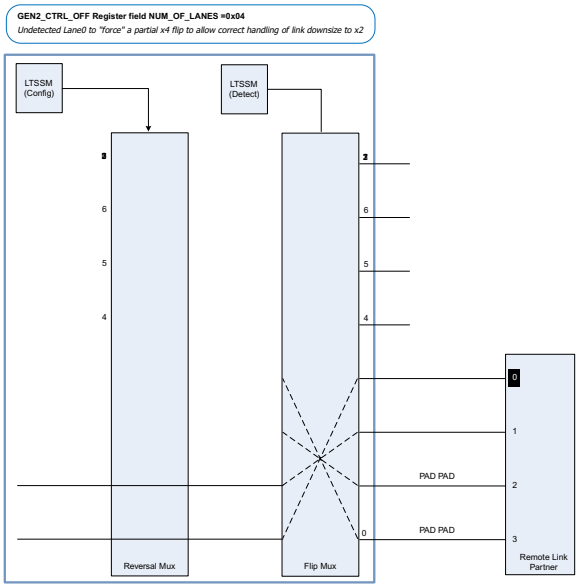


Figure B-12 x8 to x4-Top (Non-reversed Lanes) with Downsize to x2

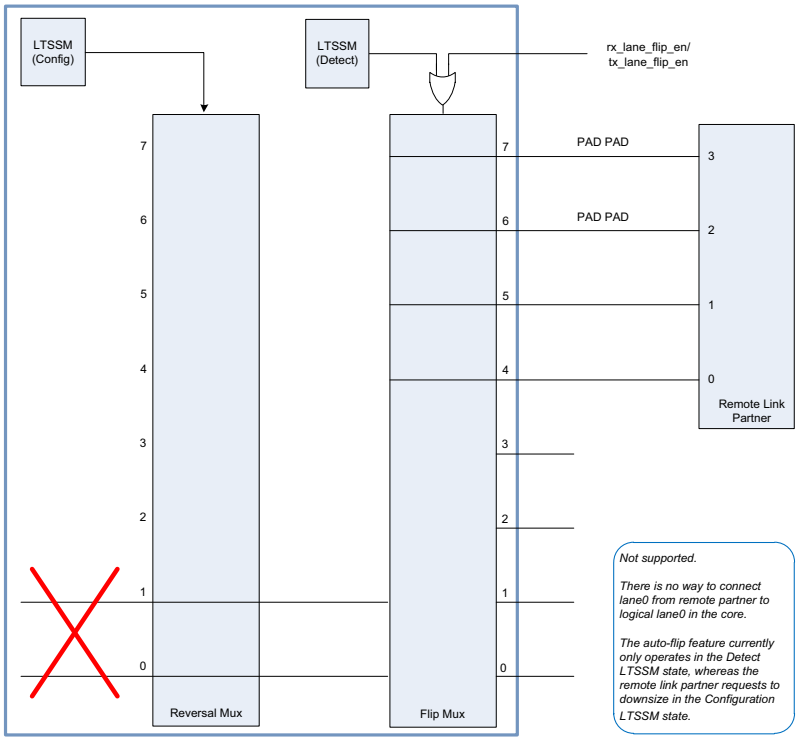
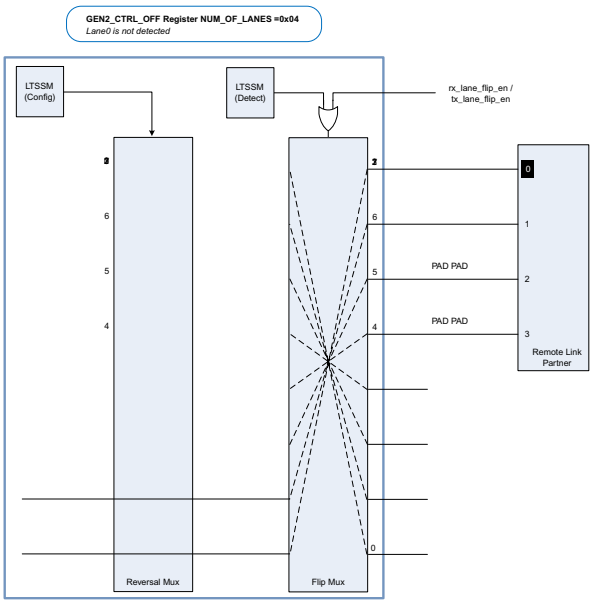


Figure B-13 x8 to x4-Top (Reversed Lanes) with Downsize to x2



B.3 Broken Lane Operation

Figure B-14 x8 to x4-Bottom (Lane0 Broken in Detect)

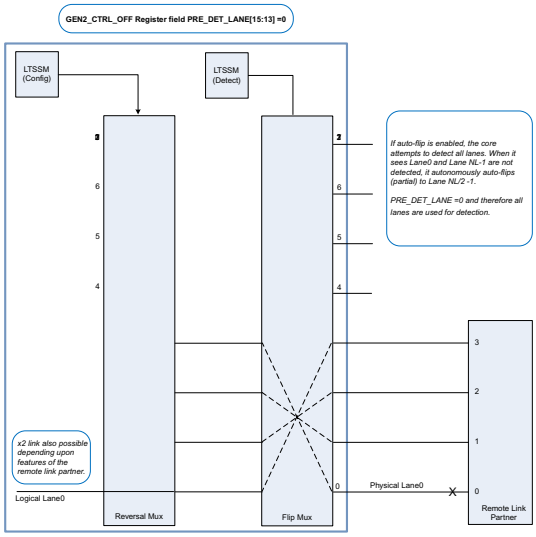


Figure B-15 x8 to x8 (Lane0 Asymmetrically Broken: OK in Detect; Broken in Polling)

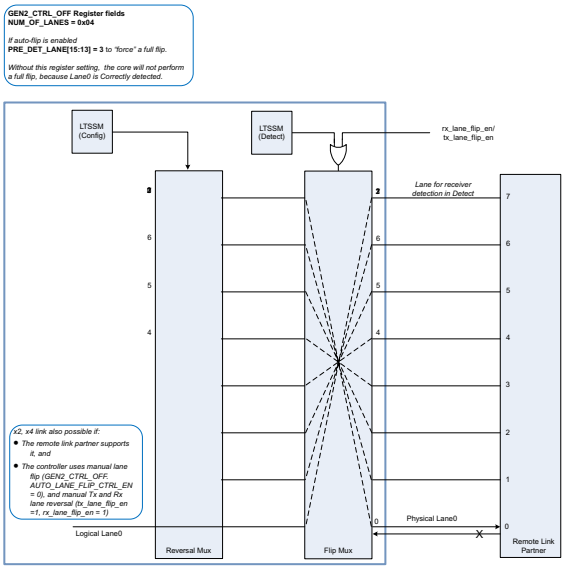


Figure B-16 x8 to x4-Bottom (Lane0 Asymmetrically Broken: OK in Detect; Broken in Polling)

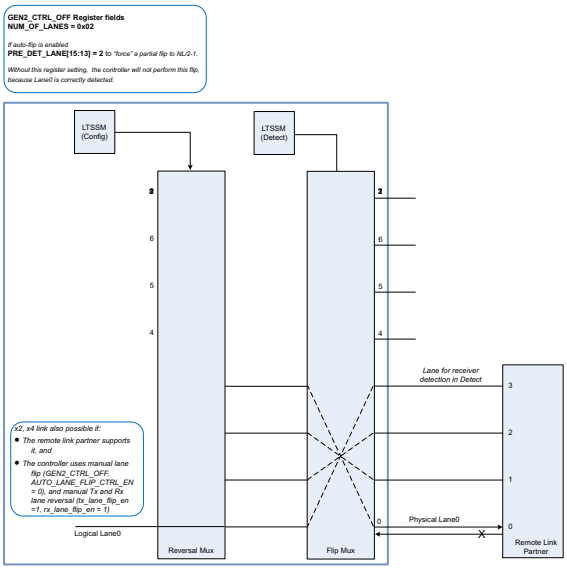


Figure B-17 x8 to x2-Bottom (Lane0 Asymmetrically Broken: OK in Detect; Broken in Polling)

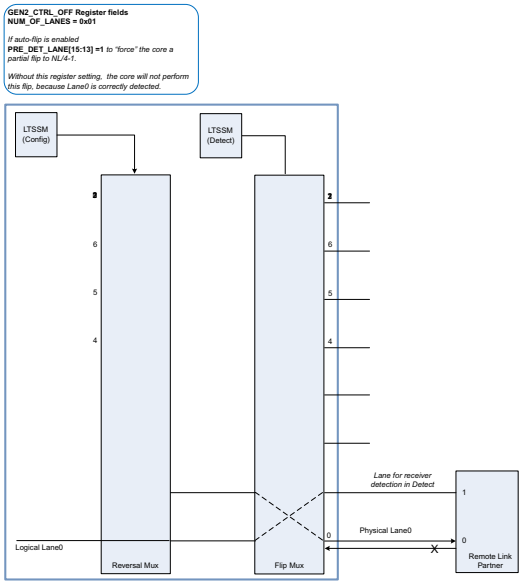


Figure B-18 x8 to x2 Partial Lane Flip and Partial Lane Reversal

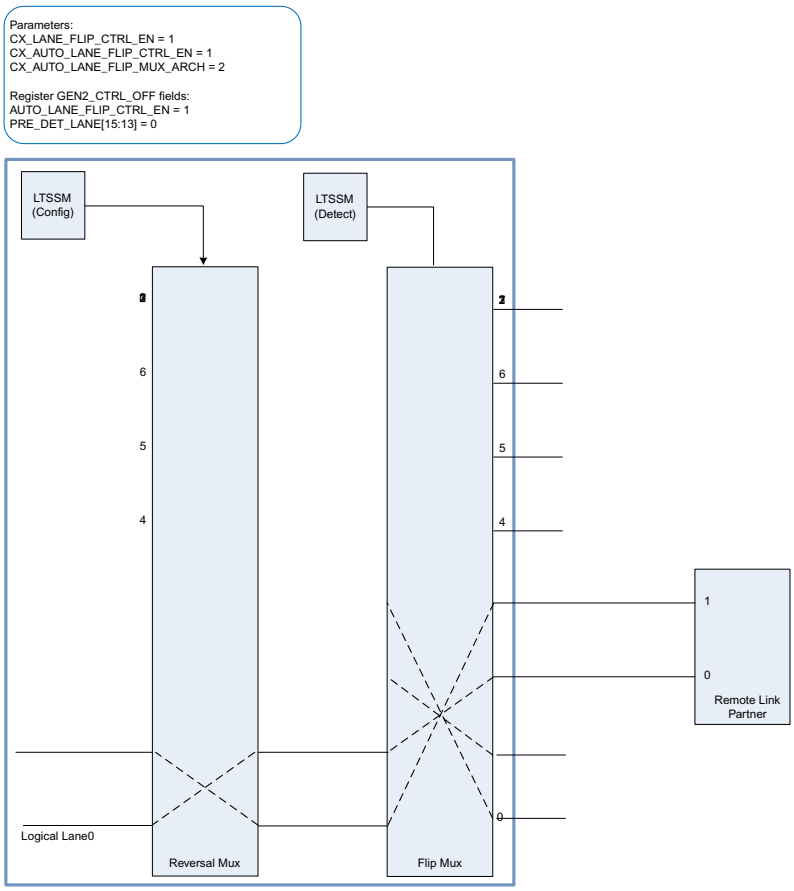
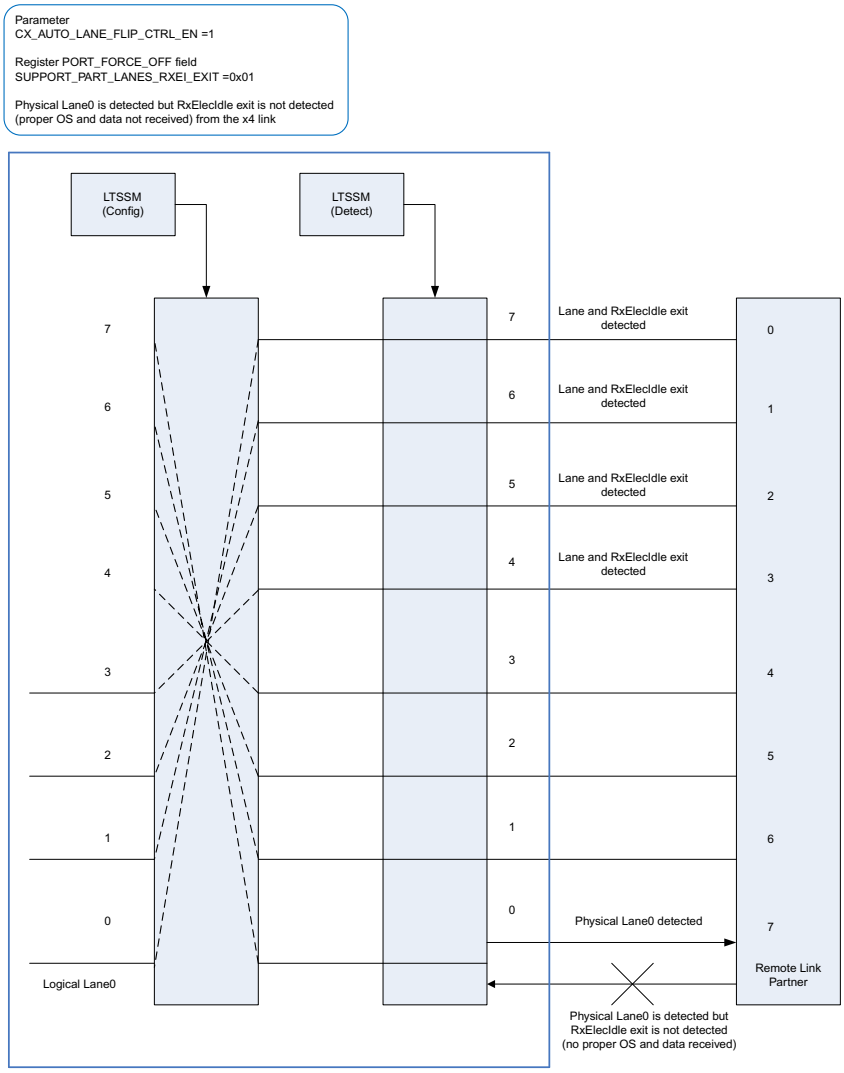


Figure B-19 X8: Autonomous Lane Reversal when Physical Lane 0 is Detected but Rx Electrical Idle is Not Detected



C

Advanced Information: Loopback

The controller supports local and remote digital loopback. The following topics are discussed:

- "Overview"
- "Local Digital Loopback (PIPE)" on page 665
- "Local Analog Loopback" on page 667
- "Remote Digital Loopback" on page 668
- "Equalization for Loopback" on page 670

C.1 Overview

The controller supports local digital and analog loopback modes without the need for a remote link partner, as shown in Figure C-2 and Figure C-2. You can use this mode for chip production test, or in any other scenario where you do not have a remote link partner. It also supports remote digital loopback with a remote link partner acting as the loopback slave, as shown in Figure C-3. You can use this mode for board debug, BER testing, system debug, or in any other scenario where you have a remote link partner.

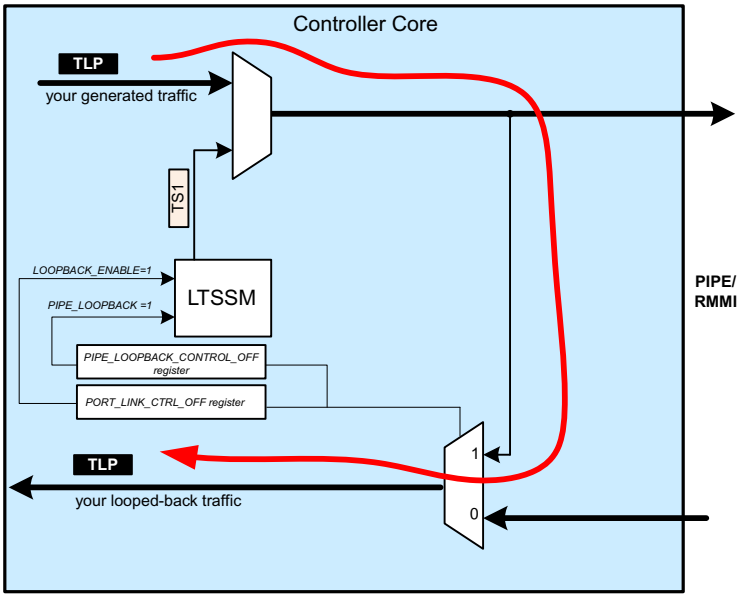
Loopback in the controller operates on a per link basis only. You cannot operate it on a per lane basis. The loopback implementation overrides LinkUp to one (it should be zero according to the specification), and allows the link to initialize as if it is in L0.

For Gen5 with CCIX ESM configurations and the LTSSM transition is from L0 -> Recovery -> Configuration.Linkwidth.Start -> Loopback.Entry. If ESM is enabled on both the sides of the link and the test scenario tries to achieve ESM Data Rate1 in loopback on the link, then before loopback master initiates loopback entry, it must set Max Link Speed and Target Link Speed to Gen4 rate (Loopback Slave Max and Target Link Speed can be Gen5 or Gen4 rate).

C.2 Local Digital Loopback (PIPE)

This section discusses the operation of the controller when entering, and exiting digital local loopback. It also discusses the behavior of the controller when in local loopback.

Figure C-1 Components Involved in Digital Local Loopback Without a Link Partner



C.2.1 Entering Local Loopback

The local loopback[1] connects the PIPE RX signals to the PIPE TX signals, and only operates in the loopback LTSSM state, as the link is not able to train against itself and enter L0. The procedure for entry into local loopback is:

- Set the Gen3 Equalization Disable bit in the Gen3 Control Register `GEN3_RELATED_OFF`. You must do this, as the PHY has no role in PIPE loopback.
- Set the `PIPE_LOOPBACK` bit in the `PIPE_LOOPBACK_CONTROL_OFF` register.
- Set the `LOOPBACK_ENABLE` bit in the `PORT_LINK_CTRL_OFF` register.

Note

ASPM must be disabled prior to enabling LOOPBACKmode.

ASPM is disabled by programming the `PCIE_CAP_ACTIVE_LINK_PM_SUPPORT` field of the `LINK_CAPABILITIES_REG` register to 2'b00.

C.2.2 In PIPE Loopback

The loopback mode allows the link to initialize, as if it is in L0. After entering loopback mode, the controller initializes flow control for VC0. When this completes, you can then send traffic (TLPs) on the controllers XALI0/1/2 client interfaces. As the traffic is looped back, the port must complete its own requests. When in loopback mode, the port initializes flow control against itself as every TLP and DLLP sent is also received. TLPs which are not received correctly as a result of link errors are subject to replay as if the link is in L0

1. This feature is automatically enabled. The CX_PIPE_LOOPBACK_EN hidden configuration parameter is set by default to 1.

state. Transmitted TLPs are subject to credit checks as if the link is fully operational. Received TLPs are subject to error checking and filter checks as it the link is operating in L0 state.

Message Considerations

When the port is an Upstream Port (USP), you must not enable error message generation. When an error occurs on the PCIe link, and you have enabled error message generation, the USP generates an Error message. Traffic generated by the port is looped back to itself, and as an USP does not expect to receive messages, it generates an additional message, and so on.

A DSP must automatically send a Set_Slot_Power_Limit message when it enters L0, so the message is sent and is received by the port itself. A DSP does not expect to receive a Set_Slot_Power_Limit message. Therefore, it is detected as an invalid message, and the unsupported request detected bit in Device Status register is be set.

When the port is a Downstream Port (DSP), then internally generated messages are mixed together with traffic generation by your application on the XALI0/1/2 interfaces (or AXI bridge slave interface).

Enumeration and BAR Setup

You must configure the BARs (USP), memory/IO ranges (DSP), set the memory space enable and bus master enable, so that the receive filter accepts TLPs. Alternatively, you can turn off the filter rules to allow TLPs that would normally be dropped to be accepted (see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715 and "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691).

In loopback mode, you can initialize the BARs by writing from the local CPU through the DBI.

Gen3 Operation

You must set the Gen3 Equalization Disable bit in the Gen3 Control Register `GEN3_RELATED_OFF`, as the PHY has no role in local loopback.

C.2.3 Exiting Local Loopback

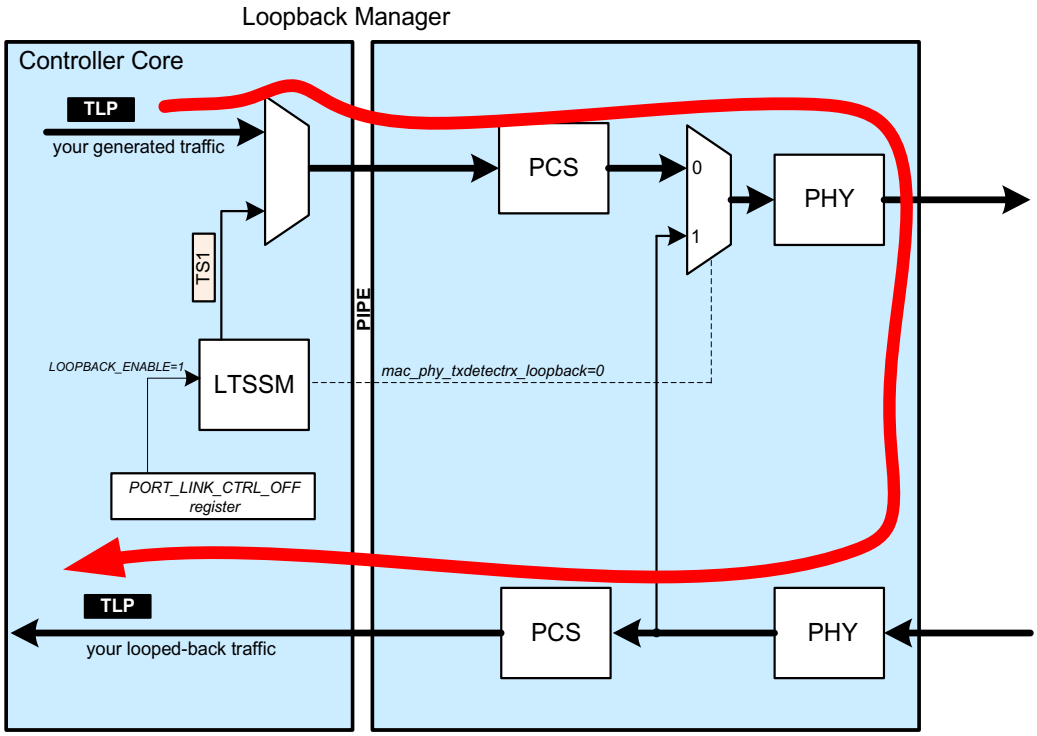
To exit loopback mode:

- Clear the `PIPE_LOOPBACK` bit in the `PIPE_LOOPBACK_CONTROL_OFF` register.
- Clear the `LOOPBACK_ENABLE` bit in the `PORT_LINK_CTRL_OFF` register.

C.3 Local Analog Loopback

This section discusses the operation of the controller when entering, and exiting analog local loopback. It also discusses the behavior of the controller when in local loopback. Entering loopback mode, exiting loopback mode, and looping the TX data back to its RX are the same as "Remote Digital Loopback" on page 668 but without the remote loopback slave (no link partner).

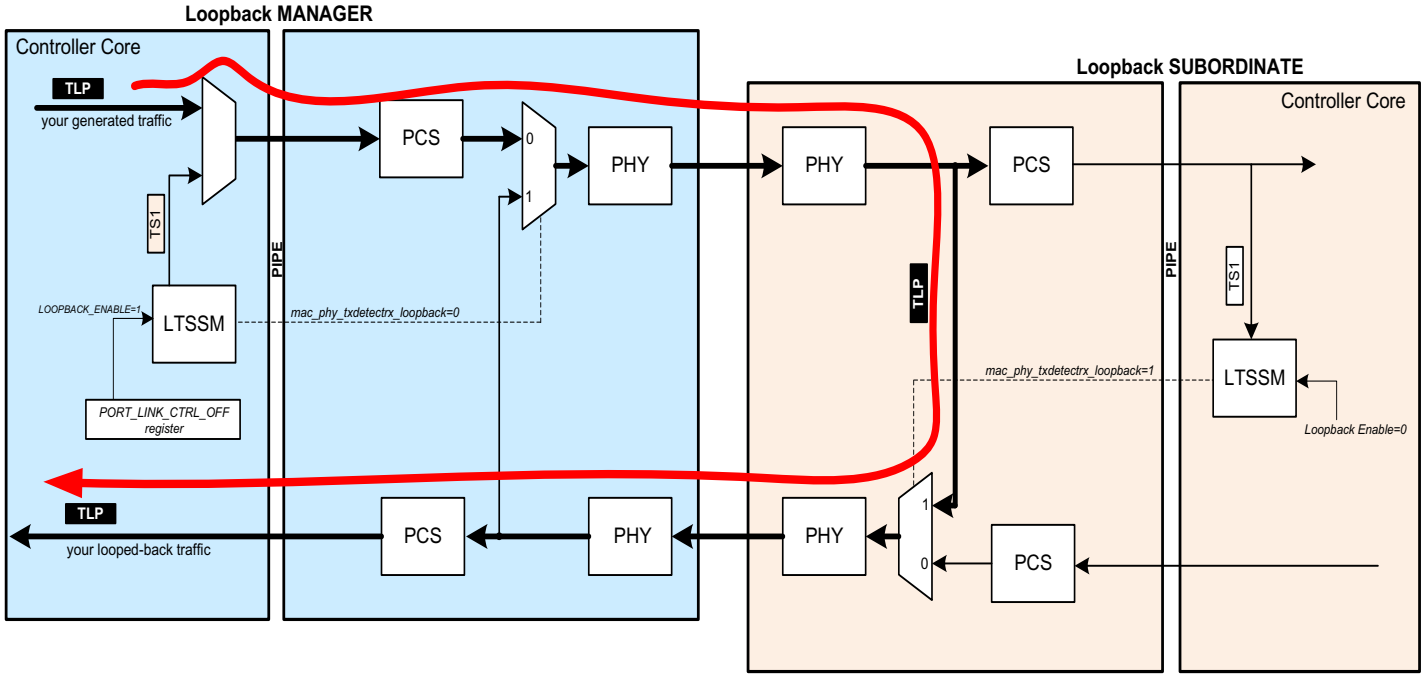
Figure C-2 Components Involved in Analog Local Loopback Without a Link Partner



C.4 Remote Digital Loopback

This section discusses the operation of the controller when entering, and exiting remote digital loopback. It also discusses the behavior of the controller when in remote digital loopback.

Figure C-3 Components Involved in Remote Digital Loopback with a Link Partner



C.4.1 Entering Remote Digital Loopback

A loopback master is the component requesting loopback and transmitting the data, for looping back by the slave, as in Figure C-3. A loopback slave is the component looping back the data. The behavior of a [loopback] master is not defined by the *PCI Express Base Specification, Revision 6.0, Version 1.0*. Only the protocol for negotiating loopback in a slave device is defined.

To direct the master to enter loopback mode, you must set the `LOOPBACK_ENABLE` bit in the `PORT_LINK_CTRL_OFF` register. You do not have to do anything to enable loopback mode in a slave. It enters loopback mode in response to the master initiating loopback mode.

Loopback uses bit 2 (loopback) in the Training Control field which is sent within the TS1 and TS2 Ordered Sets (OS). The slave device enters loopback whenever two consecutive TS1 OS are received with the loopback bit set. The master device enters loopback when it receives back from the slave, the TS1 OS that it sent. For more information, see 4.2.5.10. "Loopback" and 4.2.6.10. "Loopback" sections of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. When the slave LTSSM enters loopback, it asserts the `mac_phy_txdetectrx_loopback` PIPE input of the slave PHY. This activates the slave loopback multiplexer in Figure C-3.

C.4.2 In Remote Loopback

The loopback master mode allows the link to initialize, as if it is in L0. After entering loopback mode, the controller initializes flow control for VC0. When this completes, you can then send traffic (TLPs) on the controllers XALI0/1/2 client interfaces. As the traffic is looped back by the link partner, the master must complete its own requests. When in loopback mode, the master initializes flow control against itself as every

TLP and DLLP sent is also received. TLPs which are not received correctly as a result of link errors are subject to replay as if the link is in L0 state. Transmitted TLPs are subject to credit checks as if the link is fully operational. Received TLPs are subject to error checking and filter checks as it the link is operating in L0 state.

Message Considerations

When the loopback master is an Upstream Port (USP), you must not enable error message generation. When an error occurs on the PCIe link, and you have enabled Error message generation, the USP generates an Error message. Traffic generated by the master is looped back to itself by the slave, and as an USP does not expect to receive messages, it generates an additional message, and so on.

A DSP must automatically send a Set_Slot_Power_Limit message when it enters L0, so the message is sent and is received by the master itself. A DSP does not expect to receive a Set_Slot_Power_Limit message. Therefore, it is detected as an invalid message, and the unsupported request detected bit in Device Status register is be set.

When the loopback master is a Downstream Port (DSP), then internally generated messages are mixed together with traffic generation by your application on the XALI0/1/2 interfaces (or AXI bridge subordinate interface).

Enumeration and BAR Setup

You must configure the BARs (USP), memory/IO ranges (DSP), set the memory space enable and bus master enable, so that the receive filter accepts TLPs. Alternatively, you can turn off the filter rules to allow TLPs that would normally be dropped to be accepted (see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715 and "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691).

Your RP link partner can perform enumeration and BAR setup in L0, before you enable loopback mode. Alternatively, in loopback mode, you can initialize the BARs by writing from the local CPU through the DBI. Any traffic generated by a slave never reaches the PCIe link. Therefore, if the RP is a slave, it cannot do enumeration or BAR setup in loopback mode.

Gen3 Consideration

According to section 4.2.3. Link Equalization Procedure for 8.0 GT/s Data Rate of the *PCI Express Base Specification, Revision 6.0, Version 1.0*, the loopback master is responsible for communicating the Transmitter and receiver settings it wants the slave to use. It does this through the:

- EQ TS1 Ordered Sets it transmits in the 2.5 GT/s or 5.0 GT/s data rate
- Preset or Coefficient TS1 Ordered Sets it transmits in the 8.0 GT/s data rate

The Synopsys implementation of loopback does not support the changing of Preset and Coefficient values in the Loopback.entry LTSSM state.

C.4.3 Exiting Remote Loopback

To cause the [loopback] master to exit loopback mode, you must clear the `LOOPBACK_ENABLE` bit in the `PORT_LINK_CTRL_OFF` register. You do not have to explicitly direct the slave to exit loopback mode. It exits loopback when it detects four consecutive EIOSs, which the master transmits after exiting loopback mode. If the current link speed is 2.5 GT/s, and an EIOS is received, or Electrical Idle is detected or inferred on any Lane, then the slave also exits loopback mode.

C.5 Equalization for Loopback

In the Gen6 Loopback equalization, one port is Loopback Leader and the other port is Loopback Follower. The four EQ phases for DSP and USP are the same as in the Recovery state. The lanes not under test use preset P4 for Gen5 or Q0 for Gen6.

- DSP EQ1
  - If perform_equalization_for_loopback_64GT = 1b, Loopback Follower advertises Gen6 + Flit Mode
  - If perform_equalization_for_loopback = 1b && flit_mode_support =1 Capability Register, EQ2 and EQ3 must be performed
  - 24ms timeout && perform_equalization_for_loopback = 1b
    - Loopback.Eq1 -> Loopback.Entry
- DSP EQ2
  - 64ms timeout && perform_equalization_for_loopback = 1b for Gen6 EQ (32ms timeout for Gen5)
    - Loopback.Eq2 -> Loopback.Entry
- DSP EQ3
  - Eq3 -> Loopback.Entry (for Gen6 EQ with perform_equalization_for_loopback =1b)
    - lane_under_test = optimal && Rx 2 TS1s with Retimer_Eq_Extend = 0b, or 48ms timeout
- USP EQ0
  - Eq0->Loopback.Entry
    - 12ms timeout && perform_equalization_for_loopback =1b
- USP EQ1
  - If perform_equalization_for_loopback_64GT = 1b, Loopback Follower advertises Gen6 + Flit Mode
  - 12ms timeout && perform_equalization_for_loopback = 1b
    - Loopback.Eq1 -> Loopback.Entry
- USP EQ2
  - Eq2 -> Loopback.Entry (for Gen6 EQ)
    - 48ms timeout with perform_equalization_for_loopback = 1b
- USP EQ3
  - Eq3 -> Loopback.Entry (for Gen6 EQ with perform_equalization_for_loopback =1b)
    - lane_under_test receives 2 TS1s with EC = 00b, or 64ms timeout

D

Advanced Information: Lane Deskew

This appendix discusses lane-lane deskew by the RX path of the controller. It is a reference for calculating the maximum theoretical deskew buffer depth; which you can set to a different value based on your PHY and system requirements. You can disable the deskewing mechanism by setting the `CX_DESKEW_DISABLE` configuration parameter. This could be used, for example, when connecting to a multi-lane PHY that implements lane-lane deskewing.

D.1 Conventional PCIe Deskew Requirements

Note

- In releases prior to 4.50a, the deskew buffer depth is calculated from CX_MAX_SKEW_NUM which has two options only (“Shallow” and “Deep”). Therefore, it is not possible to select a deeper depth if required.
- In 4.50a, the CX_MAX_SKEW_NUM parameter is not visible in the coreConsultant GUI. For existing configurations, you can continue to set it in your configuration batch file. For new configurations, you can configure lane deskew capabilities using the new CX_DESKEW_DEPTH_CPCIE parameter.
- From 4.60a onwards:
  - The default of the CX_DESKEW_DEPTH_CPCIE parameter is calculated based on the E12/C8/C10 Synopsys PHY.
  - The coreConsultant GUI displays (as a read-only parameter) the maximum allowable PHY skew based on the value of CX_DESKEW_DEPTH_CPCIE.
  - It is expected that you **confirm this PHY skew value with your PHY vendor** and adjust CX_DESKEW_DEPTH_CPCIE accordingly.
  - The coreConsultant GUI also displays (as read-only parameters) the individual system skew components used in the total skew calculation.

A multi-lane controller configuration removes lane-lane skew using a deskew buffer. The maximum possible amount of PIPE lane-lane skew (as measured at the RX deskew buffer in the controller) is comprised of the following four components:

- **Wire Skew**: This is the maximum skew (LRX-SKEW) allowed by PCIe specification at the PHY RX I/O pads. Added by the channel (physical link) or repeater delay differences.
- **PCS Skew**: Assumes that the PHY skew (measured at PHY RX I/O and PIPE Rx interfaces) is comprised of at least the following components:

  For Original PIPE architecture (`CX_PIPE_SERDES_ARCH =0`)

  - Gen3/4 sync header removal skew

    The sync header removal skew for Gen3/4 configuration accounts for the difference in symbol lock timing between the lanes. `phy_mac_rxdatavalid` de-assertion timing can vary between the lanes.
  - PHY implementation-specific CDC skew
  - PHY implementation-specific de-serializer skew
  - PHY implementation-specific symbol aligner skew

  For SerDes architecture (`CX_PIPE_SERDES_ARCH =1`)

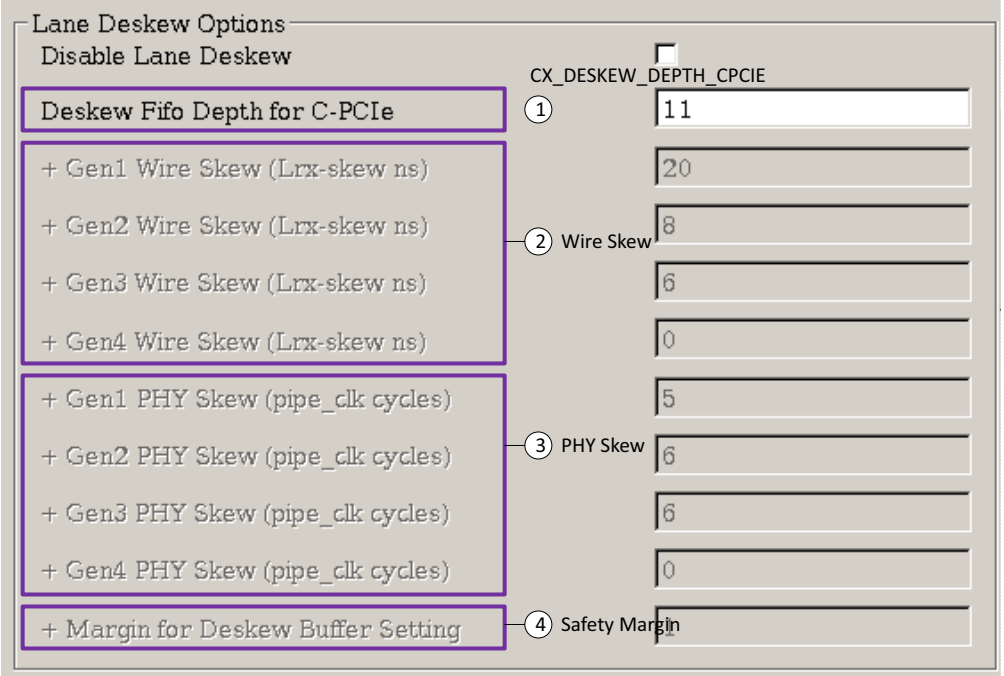
  - PHY implementation-specific CDC skew
  - PHY implementation-specific de-serializer skew
- **MAC Component:**
  - For Original PIPE architecture (`CX_PIPE_SERDES_ARCH =0`), `2*core_clk` cycles for data processing. For some configurations, an additional `core_clk` period is added.
  - For SerDes architecture (`CX_PIPE_SERDES_ARCH =1`), SerDes PIPE skew is added to the Original PIPE architecture skew.
- **Safety Margin**: One `core_clk` period.

Table D-1 Reference Value of Skew Components

Speed	Skew Components				
	Wire Skew (ns)	PHY Skew	MAC Component (core_clk cycles)		Safety Margin (core_clk cycles)
Gen1	20	Consult your PHY vendor	2 or 3[a]	(+ SerDes PIPE Skew)[b]	1
Gen2	8		2 or 3		
Gen3	6		2		
Gen4	5		2		
Gen5	5		2		

a. For on Gen1/2 2s/4s configuration an additional core_clk cycle is added because the COM symbol is not always on byte0.
b. The SerDes PIPE Skew is added only when CX_PIPE_SERDES_ARCH =1

Figure D-1 CX_DESKEW_DEPTH_CPCIE Parameter Setting in coreConsultant



E

Advanced Information: Clock and Data Crossing (CDC)

This appendix discusses aspects of clock and data crossing. The following topics are discussed:

- "CDC Overview"
- "AXI Bridge CDC Details" on page 677
- "SerDes Architecture CDC Details" on page 679
- "Port Clocking and Input Synchonizers" on page 680
- "CDC Reports" on page 681

E.1 CDC Overview

The PCIe controller (in C-PCIe mode, excluding the AXI bridge, CXS interface, and SerDes PIPE) has three clock domains:

- `core_clk / radm_clk_g`
- `aux_clk`
- `pipe_clk`

For more information on clock generation, see "Clock Requirements" on page 68

During normal operation, `core_clk/radm_clk_g` and `aux_clk` must be completely synchronous. During low-power operation, when the controller is operating on `aux_clk` only, the frequency of `aux_clk` can be lowered to achieve additional power reduction. Therefore, there is no CDC synchronization logic implemented between the `aux_clk` and `core_clk/radm_clk_g` domains.

The `core_clk/radm_clk_g` and `pipe_clk` clocks must be quasi-synchronous, that is, having the same phase, but not necessarily having the same frequency. Therefore, there is no CDC synchronization logic implemented between the `pipe_clk` and `core_clk/radm_clk_g` domains.

E.2 AXI Bridge CDC Details

The "AMBA" on page 393 has three[1] bus clocks that are assumed to be asynchronous to `core_clk` and `aux_clk`:

- `slv_aclk`
- `mstr_aclk`
- `dbislv_aclk`

All signals passing between these asynchronous clock domains require CDC synchronization using Synopsys standard DesignWare Base Core Module (BCM) synchronizers. There are no clock paths between the `pipe_clk` and AXI clocks. The AXI bridge module has four clock domains.

- `mstr_aclk`: AXI manager interface clock.
- `slv_aclk`: AXI subordinate interface clock.
- `dbislv_aclk`: AXI DBI slave interface clock when the dedicated DBI slave interface is present.
- `core_clk/radm_clk_g`: PCIe controller clock.

The AXI bridge assumes that `mstr_aclk`, `slv_aclk`, and `dbislv_aclk` are asynchronous to `core_clk/radm_clk_g` and it implements the appropriate clock crossings using pre-existing FIFOs and inserting extra FIFOs when necessary. When your AXI clocks are synchronous to `core_clk/radm_clk_g` then you can eliminate unnecessary clock crossing circuitry by disabling the `SLV_CLK_DIFF_ENABLE` and `MSTR_CLK_DIFF_ENABLE` configuration parameters.

Caution

Gen2/Gen3/Gen4/Gen5 Dynamic Frequency

When you are using a dynamic frequency Gen or Gen3 controller, the frequency of `core_clk` changes across speed modes. In these cases, your AXI clock cannot be synchronous to `core_clk`. You cannot set `SLV_CLK_DIFF_ENABLE` or `MSTR_CLK_DIFF_ENABLE` to 0.

The upper frequency limit for the AXI clocks is set by synthesis technology library that you use. There are no known functional restrictions on the AXI clock to `core_clk/radm_clk_g` clock ratio. However, the maximum and minimum simulation-verified clock ratios are 16:1 and 1:16 respectively. For more information on CDC RAMs supported, see "RAM Requirements" on page 64.

E.2.1 AXI Bridge RAM Architecture

For more information of the AXI bridge RAM clock crossings, see "AXI Bridge Architecture" on page 400.

E.2.2 LTIM and MSI-GIC Interface

When `CC_LTIM_ENABLE` =1, the LTIM module adds the following clock domains:

- `lti_clk`
- `core_clk` or `aux_clk`
- `mstr_aclk`

All the clocks are asynchronous to `core_clk` and `aux_clk`

1. It also has ungated versions of these clocks, slv_aclk_ug, mstr_aclk_ug, and dbislv_aclk_ug.

For MSI-GIC, the CDC synchronization happens between the interface clock and `core_clk` or `aux_clk` based on the `CC_LTIM_ENABLE` parameter.

- When `CC_LTIM_ENABLE =1`, its interface is on `lti_clk`
- When `CC_LTIM_ENABLE =0`, its interface is on `mstr_aclk`

E.3 SerDes Architecture CDC Details

The "SerDes Architecture" on page 387 has `pipe_rx_clk` (per lane recovered clock), that are assumed to be asynchronous to `core_clk_ug` (`CX_FREQ_STEP_EN =0`) or `pipe_clk` (`CX_FREQ_STEP_EN =1`)

The asynchronous clock domains require CDC synchronization using standard DesignWare Base Core Module (BCM) synchronizers or elastic buffer and are implemented as,

- For Gen5 speed, the clock rate difference between these clocks must be maximum 200 ppm with SRNS or 3200 ppm with SRIS.
- For Gen4 and below, the clock rate difference between these clocks must be maximum 600 ppm with SRNS or 5600 ppm with SRIS.
- For common clocks, the clock rate of these clocks must be the same.

E.4 Port Clocking and Input Synchonizers

For more information of the clocks used for input and output clocking, see "Synchronous To Attribute" on page 71. The inputs in Table E-1 have metastability synchronizers.

Table E-1 Inputs With Synchronizers.

Input Name	Synchronizing Clock[a]	Note
phy_clk_req_n	aux_clk	
phy_mac_pclkack_n	aux_clk	
clkreq_in_n	aux_clk	
perst_n	aux_clk	
ack_en_vmain	aux_clk	
phy_mac_rxelecidle	aux_clk	
phy_mac_phystatus	aux_clk	Also clocked normally on core_clk_ug,core_clk,pipe_clk
slv_a*valid	aux_clk	Also clocked normally on slv_aclk.
dbi_a*valid	aux_clk	Also clocked normally on dbi_aclk.

a. These signals are synchronized using aux_clk and can be driven/supplied asynchronously to the controller in certain low-power modes. The presence of aux_clk in the SynchronousTo attribute for these signals identify the synchronizing clock.

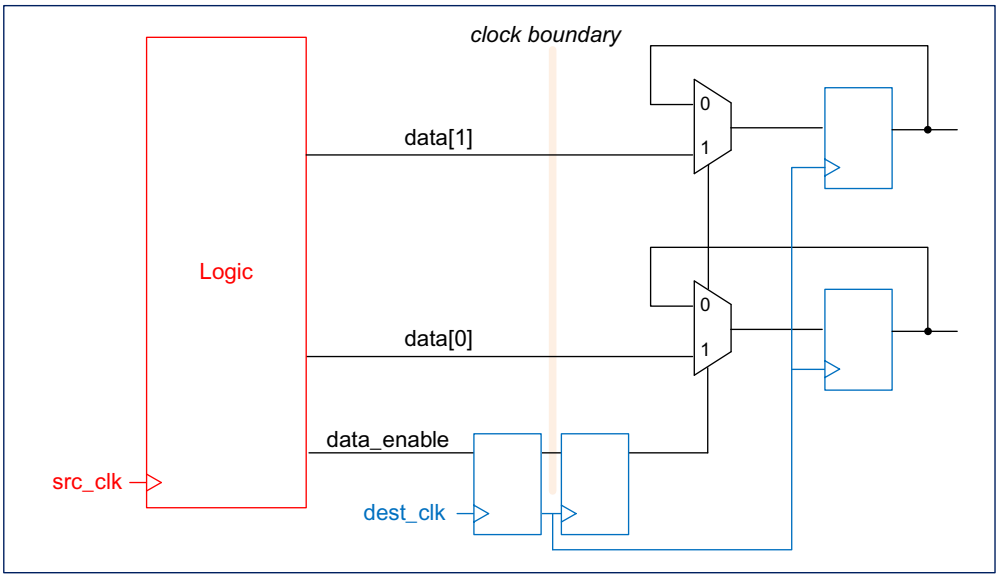
E.5 CDC Reports

Detailed CDC reports and Spyglass waiver information are available at http://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/PCIe_CDC.pdf

You can inspect the Spyglass control and waiver files at the location specified in the *User Guide*.

The Spyglass CDC report confirms that all CDC paths in the design are synchronized between clock domains using standard CDC library parts, with the exception of the paths that are waived. These paths do not need synchronization for the reasons outlined in the CDC reports. The reports refer to the Common multiplexer scheme which is explained in Figure E-1. To create a CDC *path* report, see the "Running STA using PrimeTime" section in the "Configuring the Controller" chapter of the *User Guide*.

Figure E-1 Common Multiplexer (Recirculation) Synchronization Scheme



The controller uses a library of design-specific CDC parts that are mapped to the standard DesignWare library parts in http://www.synopsys.com/dw/buildingblock.php as follows:

Table E-2 PCIe to DesignWare Mapping

PCIe Part Name	DesignWare Library Part Name
DWC_pcie_bcm01.v	DW_minmax
DWC_pcie_bcm05.v	DW_fifoctl_if
DWC_pcie_bcm06.v	DW_fifoctl_s1_df
DWC_pcie_bcm07.v	DW_fifoctl_s2_sf
DWC_pcie_bcm21.v[a]	DW_sync
DWC_pcie_bcm22.v	DW_pulse_sync
DWC_pcie_bcm23.v	DW_pulseack_sync

PCIe Part Name	DesignWare Library Part Name
DWC_pcie_bcm24.v	DW_gray_sync
DWC_pcie_bcm25.v	DW_data_sync (x2 DW_sync plus flipflop)
DWC_pcie_bcm37.v	DW_reset_sync
DWC_pcie_bcm41.v[b]	DW_sync
DWC_pcie_bcm43.v	DW_8b10b_dec
DWC_pcie_bcm44.v	DW_8b10b_enc
DWC_pcie_bcm46_a.v	DW_ecc
DWC_pcie_bcm46_b.v	DW_ecc
DWC_pcie_bcm46_c.v	DW_ecc
DWC_pcie_bcm46_d.v	DW_ecc
DWC_pcie_bcm46_e.v	DW_ecc
DWC_pcie_bcm48.v	DW_crc_p
DWC_pcie_bcm53.v	DW_arb_2t
DWC_pcie_bcm55.v	DW_arb_rr
DWC_pcie_bcm57.v	DW_ram_r_w_s_dff
DWC_pcie_bcm60.v	DW_ram_rw_a_ff
DWC_pcie_bcm62.v	DW_ram__2r_w_a_ff
DWC_pcie_bcm64.v	DW_fifo_s1_df
DWC_pcie_bcm65.v	DW_fifo_s1_sf
DWC_pcie_bcmmod48.v	DW_crc_p (modified)
DWC_pcie_bcmmod57.v	DW_ram_r_w_s_dff
DWC_pcie_bcmmod65.v	DW_fifo_s1_sf

a. It is recommended to replace BCM21 with your custom synchronizer.
b. It is recommended to replace BCM41 with your custom synchronizer. The BCM41 is a Single Clock Data Bus Synchronizer with Configurable Polarity Reset. The reset polarity must verify the instantiation and replace them with your custom synchronizer accordingly.

Metastability Simulation

The CDC synchronizer parts have a behavioral metastability model to simulate metastability mis-sampling across the clock domain. By passing in the `DW_MODEL_MISSAMPLES` parameter, the part randomly mis-samples data transitions across the clock domain, thereby stressing the CDC design. This stressing highlights areas of the design prone to error caused by divergence and re-convergence as the CDC signals is skewed by the mis-sampling algorithm.

F

Advanced Information: Reset Domain Crossing (RDC)

The PCIe controller has a number of primary inputs which are asynchronous resets. Some of these resets are asserted independent of each other, that is, a particular flip-flop can be reset while a flip-flop in its fan-out cone is not being reset. This scenario can lead to metastability in the destination flip-flop due to the fact that the start point is reset asynchronously. This issue can be identified by performing reset domain crossing analysis on the design.

Table F-1 describes the conditions under which the PCIe controller asynchronous resets are asserted.

Table F-1 PCIe Controller Reset Conditions

Reset Name	Cold Reset	Fundamental Reset	L1.2 Power Gating	Hot Reset (Link-down)
pwr_rst_n	Asserted			
sticky_rst_n	Asserted	Asserted	Asserted	
non_sticky_rst_n	Asserted	Asserted	Asserted	Asserted
core_rst_n	Asserted	Asserted	Asserted	Asserted
pipe_rst_n	Asserted	Asserted	Asserted	Asserted
phy_rst_n	Asserted	Asserted		Asserted
phy_reg_rst_n	Asserted	Asserted	Asserted	
ret_sticky_rst_n	Asserted	Asserted		
ret_non_sticky_rst_n	Asserted	Asserted		Asserted
ret_core_rst_n	Asserted	Asserted		Asserted
mstr_aresetn	Asserted	Asserted	Asserted	Asserted
slv_aresetn	Asserted	Asserted	Asserted	Asserted

Reset Name	Cold Reset	Fundamental Reset	L1.2 Power Gating	Hot Reset (Link-down)
dbi_aresetn	Asserted	Asserted	Asserted	Asserted
pipe_msgbus_rst_n	Asserted	Asserted		Asserted
pipe_rx_rst_n	Asserted	Asserted	Asserted	Asserted

The PCIe controller has the following reset conditions:

- **Cold Reset:** It is a full power on reset cycle during which all resets are asserted. Hence there are no reset domain crossing issues.
- **Fundamental Reset:** It is similar to a Cold Reset except that the `pwr_rst_n` is not asserted. Reset domain crossing issues are avoided in fundamental reset, by asserting the controller's resets synchronous to `aux_clk`, since all registers reset by `pwr_rst_n` are clocking by `aux_clk`.
- **Hot Reset:** It may give reset domain crossing issues. The `pwr_rst_n`, `sticky_rst_n`, `phy_reg_rst_n`, and `ret_sticky_rst_n` inputs are not asserted, since there are potential reset domain crossing issues due to the active resets (For example, `core_rst_n` to `pwr_rst_n`) during hot reset.

In the PCIe controller clock-gating during reset assertion is used to ensure that there are no clocks running during the link down reset event.

- All controller clocks are gated off in the clock and reset block, when the controller indicates a pending hot reset, by asserting `link_req_rst_not`.
- The resets are only asserted after the clocks have been gated off.
- The resets are de-asserted after the clocks have been enabled.

G

Advanced Information: VC-Based Arbitration

This appendix discusses VC-Based Arbitration. For an overview of TLP transmit arbitration mechanisms see "Transmit TLP Arbitration" on page 121. The following topics are discussed:

- "VC-Based Arbitration Overview"
- "VC-Based WRR Arbitration Programming Examples" on page 688

G.1 VC-Based Arbitration Overview

VC Based arbitration (`CX_XADM_ARB_MODE =0`) provides VC based programmable weighted round robin arbitration (WRR) using two different arbitration methods for the two groups of VCs:

- Strict Priority for the High-Priority VC (HPVC) group
- RR or WRR for the Low-Priority VC (LPVC) group

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

  - 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. Ties within the HPVC are resolved by client-based Strict Priority arbitration; XALI0 has lowest, XALI1 higher, and XALI2 (if implemented) highest priority.
  - The LPVC group is of lower priority than the HPVC group. Within the LPVC group, priority is determined by RR or WRR arbitration as described later. Ties within the LPVC group are resolved by client-based RR arbitration.

    Note: This scheme operates as described in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The weights are programmed through port logic registers.

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. 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.

Setting Up VC-Based WRR

The WRR arbitration scheme uses the programmed values for each LPVCs weight (programmed by VC id) to determine the percentage of the time that each VC has 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.
- 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 schemes) that the controller is configured to support: 16, 32, 64, or 128-phase).
  - The VC Arbitration Select field in port VC Control Register (selects the arbitration scheme).
- The weight assigned to each VC in the LPVC group, as programmed in VC Transmit Arbitration Register 1 and VC Transmit Arbitration Register 2. 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.

VC Transmit Arbitration Registers 1 and 2

VC Transmit Arbitration Registers 1 and 2 (`VC_TX_ARBI_OFF_1` and `VC_TX_ARBI_OFF_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 eight 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 the register description 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`:

- 1: these registers are RWS.
- 0: these registers are hardwired to a default set by configuration parameters.

G.2 VC-Based WRR Arbitration Programming Examples

The section provides examples for Setting up VC-Based Weighted Round-Robin (WRR) Arbitration.

G.2.1 Setting Up All VCs in the LPVC Group

The following example shows how to set up the controller to meet the following requirements:

- Support four VCs
- All four VCs are in the LPVC group
- VC3 has highest priority ~50%of the time
- VC2 has highest priority ~30%of the time
- VC1 has highest priority ~10-15%of the time
- VC0 has highest priority at least 5%of the time

This can be achieved with 32-phase WRR arbitration with the following weights assigned to VC03:

- VC3 weight =16; priority allocation is 16/32 =50%
- VC2 weight =10; priority allocation is 10/32 =31.3%
- VC1 weight =4; priority allocation is 4/32 =12.5%
- VC0 weight =32-(16+10+4) =2; priority allocation is 2/32 =6.25%

To implement this scheme, set up the controller as follows:

1. Select four VCs during hardware configuration.
2. Choose VC-based arbitration during hardware configuration.
3. Set the Low Priority Extended VC Count field of port VC Capability Register 1 to 3 (3 LPVCs in addition to VC0).
4. Set the VC Arbitration Capability field in port VC Capability Register 2 to 0x02 (32-phase arbitration supported).
5. Set the following weight values in VC Transmit Arbitration Register 1:
   - WRR Weight for VC3 =16
   - WRR Weight for VC2 =10
   - WRR Weight for VC1 =4
   - WRR Weight for VC0 =2
6. Set the VC Arbitration Select field in the port VC Control Register to 010b (32-phase arbitration selected).

Note The register programming in steps 3-5 is a combination of default register value selection during hardware configuration and (optionally) update of the register values at runtime through the DBI. Setting the VC Arbitration Select field (RW) in step 6 is a function of configuration software.

G.2.2 Setting Up 4 VCs in the LPVC Group, 2 VCs in HPVC Group

The following example shows how to set up the controller to meet the following requirements:

- Support 6 VCs

- 4 VCs are in the LPVC group
- 2 VCs are in the HPVC group
- Round robin arbitration is to be used for the 4 VCs in the LPVC group

You can achieve round robin arbitration for the 4 VCs in the LPVC group by assigning them equal weights. As there are 4 VCs, equal weights can be achieved with any of the following setups:

- 16-phase arbitration; the weight of each VC is 4
- 32-phase arbitration; the weight of each VC is 8
- 64-phase arbitration; the weight of each VC is 16
- 128-phase arbitration; the weight of each VC is 32

To implement this scheme, using 32-phase arbitration as an example, set up the controller as follows:

1. Select 6 VCs during hardware configuration.
2. Choose VC-based arbitration during hardware configuration.
3. Set the Low Priority Extended VC Count field of port VC Capability Register 1 to 3 (3 LPVCs in addition to VC0).
4. Set the VC Arbitration Capability field in port VC Capability Register 2 to 0x02 (32-phase arbitration supported).
5. Set the following weight values in VC Transmit Arbitration Register 1:
   - WRR Weight for VC3 =8
   - WRR Weight for VC2 =8
   - WRR Weight for VC1 =8
   - WRR Weight for VC0 =8
6. Set the VC Arbitration Select field in the port VC Control Register to 010b (32-phase arbitration selected).

G.2.3 Setting Up 5 VCs in the LPVC Group, 2 VCs in HPVC Group

The following example shows how to set up the controller to meet the following requirements:

- Support 7 VCs
- 5 VCs are in the LPVC group
- 2 VCs are in the HPVC group
- Round robin arbitration is to be used for the 5 VCs in the LPVC group

You can achieve round robin arbitration for the 5 VCs in the LPVC group by assigning them equal weights. However, because there are 5 VCs, greater granularity and closer to equal weights can be achieved with more arbitration phases. For example, with 32-phase arbitration, equal weights would be 6.4, which could be approximated with the following integer weight values for the 5 VCs:

- 6, 6, 6, 6, 8 (18.75% for 4 VCs; 25% for the remaining VC)
- 7, 7, 7, 7, 4 (21.88% for 4 VCs; 12.5% for the remaining VC)
- 7, 7, 6, 6, 6 (21.88% for 2 VCs; 18.75% for 3 VCs)

64-phase arbitration provides increased granularity. The required equal weights of 12.8 can be more closely achieved with weight assignments of 13, 13, 13, 13, and 12:

- VC4 weight =13; priority allocation is 13/64 =20.3%
- VC3 weight =13; priority allocation is 13/64 =20.3%
- VC2 weight =13; priority allocation is 13/64 =20.3%
- VC1 weight =13; priority allocation is 13/64 =20.3%
- VC0 weight =64 (13 + 13 + 13 + 13) =12; priority allocation is 12/64 =18.75%

To implement the this scheme, set up the controller as follows:

1. Select 7 VCs during hardware configuration.
2. Choose VC-based arbitration during hardware configuration.
3. Set the Low Priority Extended VC Count field of port VC Capability Register 1 to 4 (4 LPVCs in addition to VC0).
4. Set the VC Arbitration Capability field in port VC Capability Register 2 to 0x04 (64-phase arbitration supported).
5. Set the following weight values in VC Transmit Arbitration Register 1 and VC Transmit Arbitration Register 2:
   - WRR Weight for VC4 =13
   - WRR Weight for VC3 =13
   - WRR Weight for VC2 =13
   - WRR Weight for VC1 =13
   - WRR Weight for VC0 =12
6. Set the VC Arbitration Select field in the port VC Control Register to 100b (64-phase arbitration selected).

H

Advanced Information: Advanced Filtering and Routing of TLPs

This appendix discusses advanced features and operation associated with "Receive Filtering" on page 134. You should first read to familiarize yourself with basic information on the filtering and routing of received TLPs. The following topics are discussed:

- "Filtering Rules" on page 692
- "Filtering Algorithm" on page 695
- "Root Port Routing Overview" on page 698
- "Request TLP Routing Rules" on page 699
- "Processing Illegal CFG TLPs and CFG1-CFG0 Conversion in Each PCI Express Port Type" on page 701

Note

- For more information on advanced routing of TLPs with errors, see "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.

H.1 Filtering Rules

These tables describe the filtering rules (based on the PCIe Specification), and the results of the controller's filter. When a received TLP passes all of the filter rules, then it is considered to have no errors, and is routed to the destination that is configured. For more information on routing, see "Receive Routing" on page 135. For more information on what happens to malformed TLPs, see "Error Detection for Received TLPs" on page 136. For more information on what happens for IDE-related errors (IDE Check Fail, Misrouted, PCRC fail), see "PCIe IDE Error Handling and Reporting" on page 607.

Notation of filter results:

- UR: Unsupported Request "CPL Status"[1]
- CA: Completer Abort "CPL Status"[1]
- CRS: Configuration Request Retry Status "CPL Status[1]"
- SU: Successful Completion "CPL Status"[1]
- UC: Unexpected CPL
- MLF: Malformed
- MA: (Received) Master Abort set in "PCI-compatible Status Register"
- TA: (Received) Target Abort set in "PCI-compatible Status Register"
- <blank>: Filtering rule does not apply to TLP type

Table H-1 Result of Filtering Rules Applied to Request TLPs and CPL TLPs: Root Port

	TLP Type							
Filtering Rule	**MRd**	**MWr**	**CFG[a]**	**I/O**	**MSG**	**CPL with UR/CA status**	**CPL with CRS status**	**CPL with SU status**
Address does not satisfy any of the following conditions: 1. Outside of the memory range AND prefetchable memory range as determined by the corresponding Base and Limit fields in the Type-1 header. 2. The filter mask CX_FLT_MASK_UR_OUTSIDE_BAR bit is set, which treats out-of-bar TLPs as supported requests and indicates a special application requirement	UR	UR						
Native controller (without AXI bridge): FLT_Q_ADDR_WIDTH < 64, and any upper address bit (above bit position FLT_Q_ADDR_WIDTH-1) is set to 1	UR	UR		UR				

1. For non-posted TLPs, this filter result also determines the status of the completion that the controller sends back to the requester.

Filtering Rule	TLP Type							
	MRd	MWr	CFG[a]	I/O	MSG	CPL with UR/CA status	CPL with CRS status	CPL with SU status
With AXI bridge: MASTER_BUS_ADDR_WIDTH =32, FLT_Q_ADDR_WIDTH > 32, and any upper address bit (above bit position MASTER_BUS_ADDR_WIDTH-1) is set to 1	UR	UR	UR	UR	UR	UR	UR	UR
TLP header poison bit is set and the filter mask CX_FLT_MASK_UR_POIS bit is not set.	UR	UR	UR	UR	UR			
MRdLk request received and filter mask CX_FLT_MASK_LOCKED_RD_AS_UR bit is set, which indicates that you prefer to filter out the MRdLk.	UR							
CFG request received and the filter mask CX_FLT_MASK_RC_CFG_DISCARD bit is not set			UR					
I/O request received and the filter mask CX_FLT_MASK_RC_IO_DISCARD bit is not set				UR				
Vendor MSG Type0 with filter mask CX_FLT_MASK_VENMSG0_DROP bit not set.					UR			
Not Valid message for RP device					UR/MLF			
TLP with ECRC error detected	CA		CA	CA				
Requester ID mis-match						MA/TA		MLF
Requester Tag mis-match						MA/TA		MLF
Tag error (non-pad zero for reserved Tag bits						MA/TA		MLF
Byte Count Mismatch						MA/TA		MLF
CPL received with status of UR.						MA		
CPL received with status of CA.						TA		
CPL received with CRS status and CPL is not a pending configuration request.							MLF	

a. DM (in RP mode) should not expect to receive a CFG or I/O request.

A complete list of the filtering checks can be referenced in the descriptions of Symbol Timer and Filter Mask 1 register (`SYMBOL_TIMER_FILTER_1_OFF`) and Filter Mask 2 register (`FILTER_MASK_2_OFF`).

Filtering Rules Not Defined in PCIe Specification

When a zero-byte request TLP is received, also called "flush" command, the controller can drop[1] the zero-byte request. This is designed to support some applications that can not process 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 controller is programmed to handle the flush, it is the completer's task to return CPL status.

Note When the controller receives a four DWORD TLP with the LSB of the format field set to 1 and the upper 32 bits set to 0x0, then it processes the TLP as a three DWORD TLP.

1. Service internally but not passing to your application

H.2 Filtering Algorithm

The following algorithm is followed for RX completion filtering.

1. Relevant fields (BCM, Requester ID, Length, Tag, Byte Count, Lower Address, TC, Attr) are extracted from the incoming Completion TLPs.
2. Completion Length field in DW units is converted to byte units considering the lower address offset.

   `byte_offset = 4 - Lower Address[1:0]`

   `CPL Length(bytes) = (Length(DW) - |byte_offset), byte_offset`

   Where, Lower Address and Length are fields of the incoming Completion TLP
3. Bytes pending to be received for that request are stored.

   `Bytes pending = Byte Count – CPL Length`

   Where, Byte Count is a field of the incoming Completion TLP
4. A check for the last Completion TLP is performed.

   A Completion TLP is the last Completion TLP,

   - If the status of the received Completion is different than successful, or
   - If the number of bytes received in the Completion are equal or greater than the Byte Count field received in that Completion

   A Completion TLP is the last ATS Completion TLP (when `FILTER_MASK_2_OFF.MASK_RADM_2[10] =1`),

   - If the Byte Count field of the Completion TLP is four times the Length field
5. Completion filter checks performed (C1 – C9 in Figure H-1).

   - C1: Is there a current outstanding request for that Tag?
   - C2: Does the completion have ECRC error?
   - C3: Does the completion Requester ID match the current outstanding Requester ID for that Tag?
   - C4: Does bus and device number of the Completion Requester ID match the current outstanding Requester ID for that Tag?
   - C5: Does the current outstanding attr for that Tag match the Completion attr field?
   - C6: Does the current outstanding tc for that Tag match the Completion tc field?
   - C7: Is the Completion Tag inside the pool of possible tags?
   - C8: Does the current outstanding Byte Count pending for that Tag match the Completion Byte Count field or the Completion is an ATS Completion (when `FILTER_MASK_2_OFF.MASK_RADM_2[10] =1`)?
   - C9: Does the current outstanding next Lower Address for that Tag match the Completion Lower Address field?

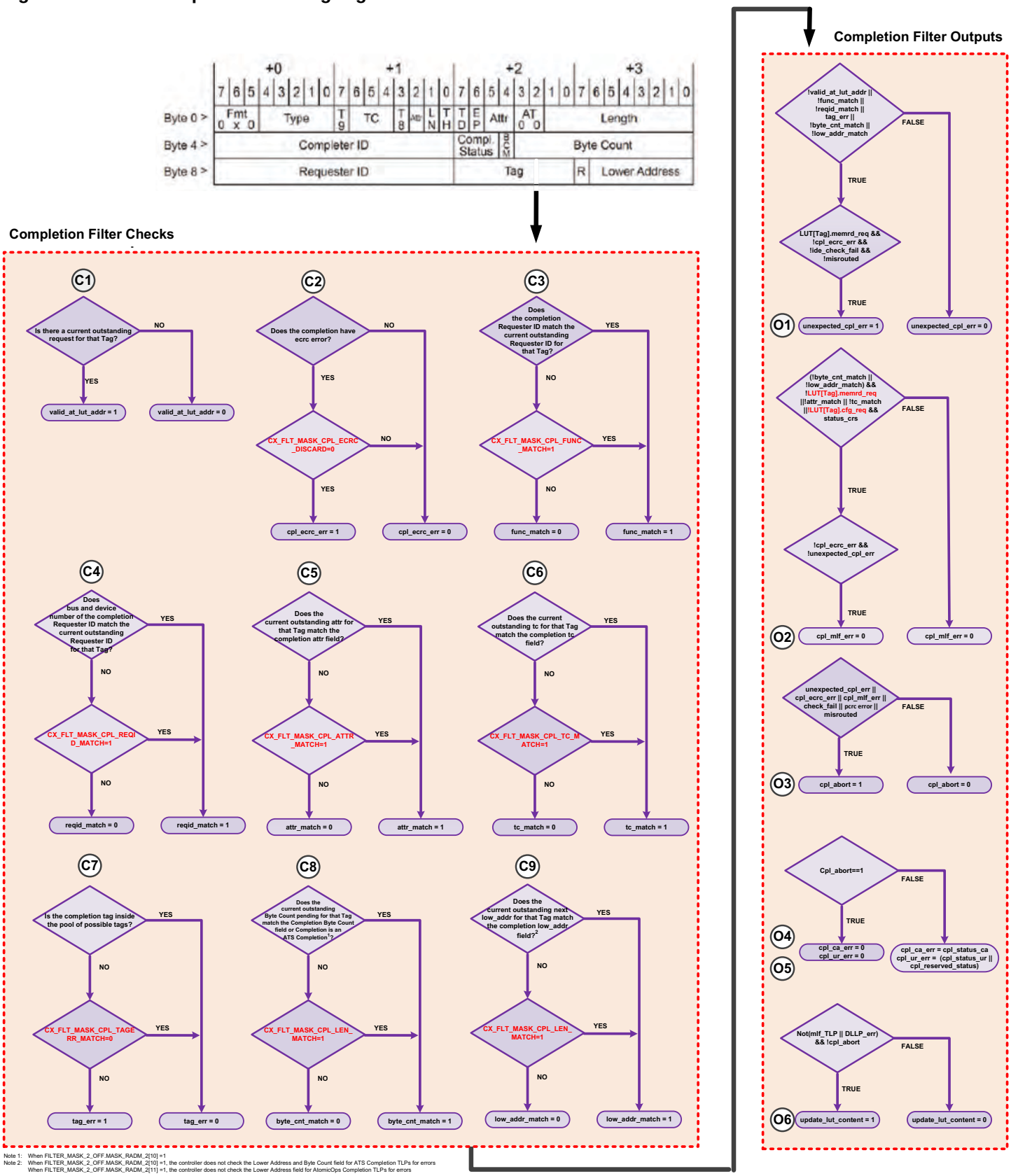
     Note:

     - When `FILTER_MASK_2_OFF.MASK_RADM_2[10] =1`, the controller does not check the Lower Address and Byte Count field for ATS Completion TLPs for errors
     - When `FILTER_MASK_2_OFF.MASK_RADM_2[11] =1`, the controller does not check the Lower Address field for AtomicOps Completion TLPs for errors
6. Output based on the checks is provided (O1 – O6 in Figure H-1)

- O1: `unexpected_cpl_err`
- O2: `cpl_mlf_err`
- O3: `cpl_abort`
- O4: `cpl_ca_err`
- O5: `cpl_ur_err`
- O6: `update_lut_content`

Figure H-1 describes the RX completion filtering algorithm used by the controller.

Figure H-1 RX Completion Filtering Algorithm



Note 1: When FILTER_MASK_2_OFF.MASK_RADM_2[10] =1
Note 2: When FILTER_MASK_2_OFF.MASK_RADM_2[10] =1, the controller does not check the Lower Address and Byte Count field for ATS Completion TLPs for errors
When FILTER_MASK_2_OFF.MASK_RADM_2[11] =1, the controller does not check the Lower Address field for AtomicOps Completion TLPs for errors

H.3 Root Port Routing Overview

For more information, see "Receive Routing" on page 135.

H.4 Request TLP Routing Rules

The tables in this section show the applicability of routing rules for request TLPs. The tables also indicate whether the controller routes the request TLPs to the intended destination or not when the request TLPs meet the conditions of the rule.

For a full analysis of what error conditions contribute toward a UR or CA status, see "Receive Filtering" on page 134 and "Advanced Information: Advanced Error Handling for Received TLPs" on page 715.

Hint

- In many cases, the standard routing rules can be masked or ignored by setting the corresponding bit in the Symbol Timer and Filter Mask 1 register (SYMBOL_TIMER_FILTER_1_OFF) and Filter Mask 2 register (FILTER_MASK_2_OFF).
- For message routing see, "Routing of Received Messages" on page 740.

By default, when the controller detects an error[1] in a received TLP, it normally performs the following:

- Discards the TLP
- Generates a completion (for non-posted requests) with the completion status set to CA or UR
- Sets the status in the PCI-compatible Status register
- Sets the status in the AER registers (when you enable AER)

Notation of routing results

- UR: Unsupported Request
- CA: Completer Abort
- CRS: Configuration Request Retry Status
- SU: Successful
- No: destination is not as specified in rule, even when conditions of rule are met
- Yes: destination is as specified in rule, when conditions of rule are met
- <blank>: Routing rule does not apply to TLP type

Table H-2 Routing Rules for Request TLPs

Routing Rule	MRd	MWr	[a]CFG	[a]IO	Vendor MSG Type0	Vendor MSG Type1	Other MSG
When a request is filtered with SU status, TRGT1 is always the destination.	Yes	Yes	No	Yes	-	-	-
When a request is filtered with UR/CA status, the TLP is dropped. For NP requests, a CPL is also generated.	Yes	Yes	Yes	Yes	Yes	Yes	Yes

1. Excluding TLPs targeted for forwarding (and not for local resources) that have ECRC errors.

Routing Rule	MRd	MWr	[a]CFG	[a]IO	Vendor MSG Type0	Vendor MSG Type1	Other MSG
The TLP is dropped, when the filter mask CX_FLT_MASK_MSG_DROP bit is 0 and the non-Vendor MSG is filtered with SU status.							Yes
The TLP is dropped, when the filter mask CX_FLT_MASK_VENMSG0_DROP bit is 0 and the VEN0 MSG is filtered with SU status.					Yes		
The TLP is dropped, when the filter mask CX_FLT_MASK_VENMSG1_DROP bit is 0 and the VEN1 MSG is filtered with SU status.						Yes	

a. RP should not expect to receive a CFG or I/O request.

H.5 Processing Illegal CFG TLPs and CFG1-CFG0 Conversion in Each PCI Express Port Type

Routing IDs, requester IDs, and completer IDs are 16-bit identifiers traditionally composed of three fields: an 8-bit bus number, a 5-bit device number, and a 3-bit function number.

The controller informs the user application if it has received poisoned CfgWr even when they are discarded, by setting the `radm_rcvd_cfg[0/1]wr_poisoned` signals to 1. You can enable this feature by setting the `CX_DETECT_POISONED_CFGWR` parameter to 1.

Configuration requests always move downstream, and never travel across a peer-to-peer connection. An RP or SW Downstream Port (DSP) never receives configuration requests.

Note You cannot set `radm_rcvd_cfg[0/1]wr_poisoned` signal for a CCIX packet.

H.5.1 RP Mode

Transmit Checking

The controller does not check configuration requests (at your application transmit interface) for valid target Bus.Device.Function numbers before it transmits them. It does not check that the target bus number > secondary bus number and that target bus number <= subordinate bus number. Your application must not provide illegal configuration requests to the controller. It must check all configuration requests from the host software. It can determine the correct range of Bus.Device numbers by snooping host configuration writes to the RP's Secondary Bus and Subordinate Bus registers.

Conversion

The controller never converts CFG1 to CFG0 at your application transmit interface, or in the controller. Your application must do the conversion where necessary.

Bus.Device Number Assignment

The assignment of Bus.Device numbers to the devices within a Root Complex, can be done in an implementation specific way.

Non-existent Devices and Functions

Your host should determine the existence (or absence) of devices and functions by reading (CfgRd) from the vendor ID register. A successful response (SU) with a non-FFFF data value, indicates the existence of the function. When your application logic receives a unsupported request (UR) response (as forwarded by the controller), then it must change the response data to FFFF, change the completion status to SU and pass the completion to the host. The AXI bridge does this automatically.

I

Advanced Information: Advanced Ordering Information

This appendix discusses advanced features and operation of ordering. The following topics are discussed.

- "PCIe Ordering Rules"
- "Inbound (Receive) Order Enforcement" on page 707
- "Outbound (Transmit) Order Enforcement" on page 709

I.1 PCIe Ordering Rules

The PCI Express ordering rules are designed to satisfy the following three requirements:

1. **Requirement: Producer-Consumer Model**

   In this model, (see Figure I-1), the consumer must not start reading a data buffer before a producer is finished writing it. To prevent violation of the model, it is necessary to ensure that no transaction passes a previously issued posted transaction. This is covered by rules 2, 5, and 9 in Table I-1 on page 705. The following performance optimizations are added:

   - **Producer-Consumer Relaxed Ordering (RO) Optimization**

     Completions and writes that are not involved in producer-consumer sequences, are permitted to pass a previously issued posted transaction. This is achieved by setting their RO bit to '1'.

   - **Producer-Consumer ID-Based Ordering (IDO) Optimization**

     A completion or request that originates from a different completer or requester, and that has its IDO bit set to 1, is permitted to pass a blocked posted request. The controller does not implement IDO but provides the information to your application.

   Strong ordering (routing all TLPs in-order without reordering) satisfies the producer-consumer model. The next two requirements seek to modify strong ordering.

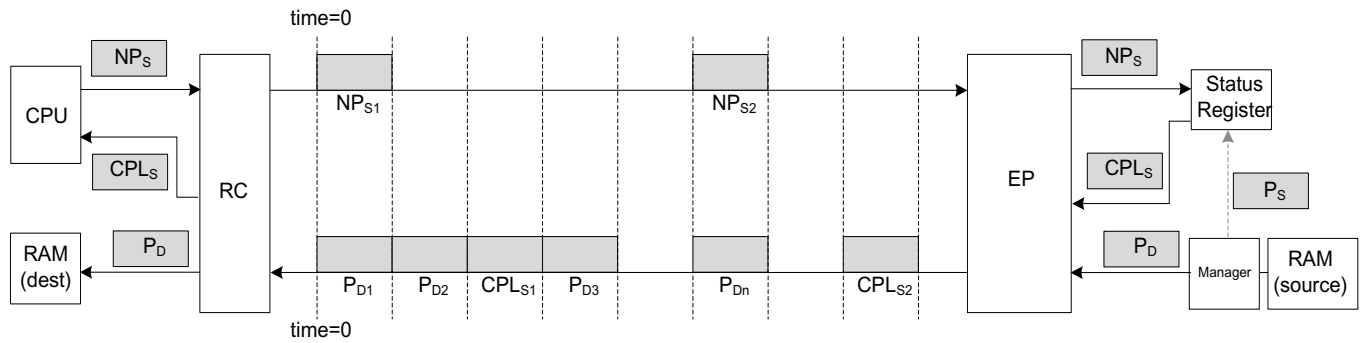
2. **Requirement: Deadlock Avoidance**

   Avoid blocking of completion and posted TLPs, by non-posted requests that are stalled. This is covered by rules 1 and 4 in Table I-1 on page 705.

3. **Requirement: Performance Optimizations**

   TLPs that are not bound by any of the previous requirements can pass or be blocked by each other. This is covered by rules 3, 6, 7, and 8 in Table I-1 on page 705.

Figure I-1 Producer-Consumer Example



Producer: The manager in the EP is taking data from the local source RAM and writing it into the destination RAM in the RC. This posted data is indicated as P_D. The manager writes to the 'finished' flag in the status register after it has written the last P_D.

Consumer: The CPU in the RC is polling the status register in the EP, checking if the manager is finished producing the data. These polling/read requests are indicated as NP_S. The corresponding completion is indicated as CPL_S.

Producer-Consumer Model: The CPL_S must not pass any P_D, or else the CPU starts to consume the data before it is fully produced. Therefore the EP, any intermediate SW, and RC queues must obey this rule.
The rule does not need to be applied to P TLPs and CPLs that are not part of this Producer-Consumer sequence, for example, RC CPU CFG reads to unrelated functions in the EP. You should set the relaxed ordering (RO) bit for these requests to '1'.

If there is an element in your system that is not enforcing the "CPL must not pass P" rule (for example, if the CPL receive queue in the RC is in bypass mode), you must use an alternative method for communicating status between a producer and a consumer; use interrupts or place the data being read-from/written-to in a different location in your PCIe system.

PCIe Ordering Rules (Summarized)

Ordering rules only apply for traffic within a single Traffic Class (TC). For a proper understanding of ordering you should be familiar with Table 2-34 in section 2.4.1, "Transaction Ordering Rules" of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. Table I-1 summarizes the PCI Express ordering rules.

Table I-1 Summarized PCIe Ordering Rules

No.	Description	Underlying System Requirement
1	**P** must be allowed to pass a previously-issued NP.	Deadlock Avoidance
2	**P** must not pass a previously-issued P, unless: ■ the later P's RO bit is 1 or ■ the later P's IDO bit is 1, and the requester IDs are different.	Producer-Consumer Model
3	**P** can optionally pass[a] a previously-issued CPL.	Performance Optimization
4	**CPL** must be allowed to pass a previously-issued NP.	Deadlock Avoidance
5	**CPL** must not pass a previously-issued P, unless: ■ CPL's RO bit is 1 or ■ CPL's IDO bit is 1, and the completer/requester IDs are different.	Producer-Consumer Model
6	**CPL** can optionally pass[a] a previously-issued CPL, when the transaction[b] IDs are different.	Performance Optimization

No.	Description	Underlying System Requirement
7	**NP** can optionally pass[a] a previously-issued NP.	Performance Optimization
8	**NP** can optionally pass[a] a previously-issued CPL.	Performance Optimization
9	**NP** must not pass a previously-issued P, unless: ■ NP's IDO bit is 1, and the requester IDs are different, or (in the case of NP writes only) ■ NP's RO bit is 1.	Producer-Consumer Model

a. Or be blocked by
b. Combination of requester ID and tag

I.2 Inbound (Receive) Order Enforcement

The controller only enforces ordering of TLPs that are delivered to the primary receive application interfaces[1] TRGT1/RBYP. It does not enforce ordering between these TLPs and the TLPs that are delivered to the TRGT0, ELBI, or SII.

Note When you configure the controller with an AXI bridge, then TRGT1/RBYP interfaces are no longer visible as they now become an internal connection between the native PCIe controller and the AXI bridge. Your application receives inbound TLPs on AXI bridge, instead of TRGT1 / RBYP.

The SII is connected to the output of the receive filter and is not subject to receive queuing. Therefore, TLPs that are routed to the SII may overtake the TLPs (not in bypass) that are destined for a primary receive application interface. Typically[2], the controller sends all message TLPs including Assert_INTx/Deassert_INTx messages to the SII.

I.2.1 Receive Ordering Schemes

The following ordering schemes are used for ordering TLPs within a single VC. **For this discussion, it is assumed that all TLP queues are configured in store-and-forward or cut-through mode.** Any TLP type that is configured in bypass mode is not subject to ordering by the controller and can pass any other TLP type. As all TLPs of the same type are queued in a FIFO, no passing within a single queue ever occurs. For example, posted never passes posted.

Note Strict ordering, or round robin ordering, between TLPs of different VCs is used. In strict ordering, the higher numbered VCs are given a higher priority. You can set the VC arbitration scheme during configuration using the CX_RADM_STRICT_VC_PRIORITY parameter or by software by writing to the VC_ORDERING_RX_Q field in the VC0_P_RX_Q_CTRL_OFF register.

PCIe Ordering (CX_RADM_ORDERING_RULES =1):

Summary: This scheme fully obeys the producer-consumer model and deadlock is avoided. All of the PCI Express ordering rules in Table 2-34 ("2.4.1 Transaction Ordering Rules" section of the *PCI Express Base Specification, Revision 6.0, Version 1.0*) are obeyed.

Method: TLPs are filtered into three separate FIFO queues: posted, non-posted, and completion. The ordering controller determines which of the three queues have a TLP that is allowed to proceed, by looking at:

- Packet halt inputs (`trgt1_radm_pkt_halt[]`, "TRGT1 Packet Grant and Halt" on page 371)
- The PCIe ordering rules

When more than one queue has a TLP that is allowed to proceed, it uses a priority arbitration algorithm to select a transaction. This prevents starvation, which occurs when delivery rate for a particular TLP type to your application is unfairly reduced.

Rules: This scheme implements all of the PCI Express ordering rules.

1. And that are not in bypass queue mode.
2. You can configure the controller to send messages to the application receive interface instead of the SII.

Relaxed Ordering (`RELAXED_ORDER_SUPPORT =1`): A completion or a posted transaction with its RO header bit set to 1, can pass a previously issued posted transaction that is blocked. RO is not implemented for non-posted writes.

Strict Priority Delivery (CX_RADM_ORDERING_RULES =0):

Summary: This scheme fully obeys the producer-consumer model and deadlock is avoided. All PCI Express ordering rules in Table 2-34 (section 2.4.1, Transaction Ordering Rules of the *PCI Express Base Specification, Revision 6.0, Version 1.0*) are obeyed.

Method: TLPs are filtered into three separate FIFO queues: posted, non-posted, and completion. They are delivered according to the priority P-CPL-NP regardless of the order in which they are received. Posted has the highest priority. No concept of received-order is used in this scheme. TLPs are delivered according to the following steps:

1. The posted queue is delivered until it is empty, or halted by your application.
2. The completion queue is delivered until it is empty, or halted by your application.
   - When the posted queue is halted, the completion queue becomes blocked unless RO of the completion is 1.
3. The non-posted queue is delivered until it is empty, halted by your application, or blocked as follows:
   - When the posted queue is halted, the non-posted queue becomes blocked.
   - When the completion queue is halted, the non-posted queue becomes blocked.
   - When the completion queue is blocked (by a halted posted queue), the non-posted queue becomes blocked.

Rules: This scheme implements all of the PCI Express ordering rules.

Relaxed Ordering (`RELAXED_ORDER_SUPPORT =1`): A completion or a posted transaction with its RO header bit set to 1, can pass a previously issued posted transaction that is blocked. RO is not implemented for non-posted writes.

Starvation (Fairness Not Comprehended): No concept of received-order is used in this scheme. Therefore, a posted TLP that is received after a non-posted is delivered first, even when the non-posted queue is not halted or blocked. This can lead to possible starvation of the non-posted queue (and completion to a lesser extent) if the influx stream of posted writes from the wire continues indefinitely. When a certain TLP type (completion or non-posted) is being passed too much, then the delivery rate for that TLP type to your application is unfairly starved/reduced, leading to possible permanent blocking of that TLP type. For example when the posted queue is never empty (because of application stalling and the receive posted TLP delivery rate), then the non-posted queue is permanently blocked.

Note Synopsys recommends that you select the PCIe Ordering scheme and not this legacy ordering scheme.

Attention The receive queues use a segmented-buffer architecture. For more information on the legacy single-buffer and multiple-buffer receive architectures, contact Synopsys support through SolvNetPlus. These legacy architectures are not upgraded to support future enhancements such as Gen4 and you should therefore migrate to the segmented-buffer architecture.

I.3 Outbound (Transmit) Order Enforcement

This section discusses reordering in the outbound direction. The following topics are discussed.

- "Ordering Across Outbound Interfaces"
- "Outbound Order Enforcement on XALI0/1/2 Interfaces" on page 709

I.3.1 Ordering Across Outbound Interfaces

The controller does not enforce ordering between the transmit interfaces, DBI, VMI, SII, XALI0, XALI1, XALI2, and so on. Furthermore, it does not enforce ordering between transactions on any of the transmit interfaces, and internally generated messages or completions (for example, for a CFG access to internal registers).

The request arbitration priority (highest first) for transmission of TLPs at the transmit interfaces is:

1. Messages (both internally-generated and messages requested through the VMI)
2. Internally-generated completions for received CFG requests
3. Transmit TLPs from XALI0/1/2 according to the arbitration scheme in "Transmit TLP Arbitration" on page 121.

I.3.2 Outbound Order Enforcement on XALI0/1/2 Interfaces

There are three application outbound transmit interfaces (XALI0, XALI1, and XALI2). All interfaces are served using a round robin arbitration scheme when credit is available, regardless of the type of transaction. For example, when a posted transaction is presented onto XALI1 followed by a completion transaction on XALI0, and when credits permit, then the posted transaction is transmitted onto the wire before the completion. When the credit is not available, then the completion on XALI0 can pass the posted on XALI1 and be sent onto the wire. However, any non-posted or completion TLPs on XALI1 (behind the blocked posted) are blocked by the halted posted queue.

It is the responsibility of your application to make appropriate use of the three interfaces. There is no guarantee that order is preserved among client interfaces. In some cases, you might consider implementing some ordering rules in your application logic, for example, holding off a MRd transaction until the MWr transaction is completed. For example, when your application has one source of outbound read transactions and one source of outbound write transactions, and they are independent sources, then the XALI0 and XALI1 interfaces of the controller should be used to perform the outbound transmission of the transactions. When the read and write are related, then one single interface XALI0 should be used and it is the responsibility of your application to maintain the desired order.

Note

- When you configure the controller with an AXI bridge, then XALI0/1/2 interfaces are no longer visible as they now become an internal connection between the native PCIe controller and the AXI bridge. Your application transmits outbound requests or CPL TLPs on AXI bridge, instead of XALI0/1/2.
- For more information on AXI ordering, see "AXI Bridge Ordering" on page 408.
- For more information on DMA ordering, see "DMA Ordering" on page 505.

J

Advanced Information: How to Turn Off Unused Lanes

When the link width of the PHY is smaller than the link width of the controller, you must turn off unused lanes of the controller's PIPE interface. This appendix describes the procedure to turn off unused lanes of the controller's PIPE interface. In this appendix, the connection between a four-lane controller and one-lane PHY is used as an example to explain the procedure to turn off unused lanes.

Caution

- This appendix only gives general guidelines, and pointers to aspects of unused lanes that you must be aware of.
- Synopsys accepts no responsibility for any PCIe IP controller and PHY integration decisions made by you as a consequence of accessing this appendix.
- The example in this appendix is not simulated or verified by Synopsys.

J.1 Conventional PCIe

Lane0 signals must always be active. For the other lanes (1,2,3) to be made inactive, you must make the connections in Table J-1.

When `CX_PIPE_SERDES_ARCH =1` and the lanes are tied off, the parameter `CX_SERDES_ARCH_LANES_TURNOFF` must be set to '1' so that the signal `serdes_pipe_turnoff_lanes` is available. For more information, see "SerDes Architecture" on page 387.

Table J-1 PIPE Interface Ports Affected by Unused Lanes

PIPE Port Name	Direction	Connection for Unused Lane signals
phy_mac_rxelecidle	Input	1b1
phy_mac_phystatus	Input	connect to phy_mac_phystatus[0]
phy_mac_rxdata	Input	0x0
phy_mac_rxdatak	Input	0x0
phy_mac_rxvalid	Input	0x0
phy_mac_rxstatus	Input	0x0
phy_mac_rxdatavalid	Input	0x0
phy_mac_rxstartblock	Input	0x0
phy_mac_rxsyncheader	Input	0x0
phy_mac_localfs	Input	0x0
phy_mac_locallf	Input	0x0
phy_mac_messagebus	Input	0x50 (ACK opcode)
phy_mac_pclkchangeok	Input	connect to phy_mac_pclkchangeok[0]
phy_mac_rxstandbystatus	Input	0x0
phy_mac_rxstatus	Input	0x0
pipe_rx_clk	Input	0x0
mac_phy_txdata	Output	Unconnected
mac_phy_txdatak	Output	Unconnected
mac_phy_txdetectrx_loopback	Output	Unconnected
mac_phy_txelecidle	Output	Unconnected
mac_phy_txcompliance	Output	Unconnected
mac_phy_rxpolarity	Output	Unconnected
mac_phy_txdatavalid	Output	Unconnected

PIPE Port Name	Direction	Connection for Unused Lane signals
mac_phy_txstartblock	Output	Unconnected
mac_phy_txsyncheader	Output	Unconnected
mac_phy_txdeemph	Output	Unconnected
mac_phy_rxpresethint	Output	Unconnected
mac_phy_rxeqeval	Output	Unconnected
mac_phy_blockaligncontrol	Output	Unconnected
mac_phy_messagebus	Output	Unconnected

Software Configuration

When there are unused lanes in a system, you must reprogram the following registers through the DBI. For this example, where a 4-lane controller is connected to a 1-lane PHY:

- Reprogram `LINK_CAPABLE` field of the `PORT_LINK_CTRL_OFF` register to 6h1 from 6h7.

  This is used by the LTSSM in Detect.
- Reprogram `NUM_OF_LANES[8:0]` field of the `GEN2_CTRL_OFF` register to 9h1 from 9h4.

  This indicates to the LTSSM, the number of lanes to check for exiting from L2.Idle or Polling.Active.
- Reprogram `PCIE_CAP_MAX_LINK_WIDTH` field of the `LINK_CAPABILITIES_REG` register to 6h1 from 6h4.

  This enables the RP to determine the Maximum Link Width for this port.

K

Advanced Information: Advanced Error Handling for Received TLPs

This appendix discusses advanced features and operation associated with "Error Handling" on page 136. You should first read to familiarize yourself with basic information on the filtering and routing of received errored-TLPs. The following advanced topics are discussed in this appendix:

- "Routing of Request TLPs with Errors (Native PCIe Controller)" on page 716
- "Routing of Request TLPs with Errors (AMBA Bridge Configuration)" on page 719
- "Routing of Completions with Errors (Native PCIe Controller)" on page 720
- "Routing of Completions with Errors (AMBA Bridge Configuration)" on page 723
- "Application Error Reporting Interface" on page 724

Note For more information on advanced routing and filtering, see "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.

K.1 Routing of Request TLPs with Errors (Native PCIe Controller)

TLPs appear on the RBYP interface in bypass mode, and on the TRGT1 interface otherwise.

K.1.1 Store-and-Forward Queue Mode

In general when the controller detects an error in a received TLP, it discards the TLP and does not forward it to your application. Exceptions to this are detailed in Figure K-1.

Figure K-1 Overview of Routing For TLPs with Errors in Store-and- Forward Mode

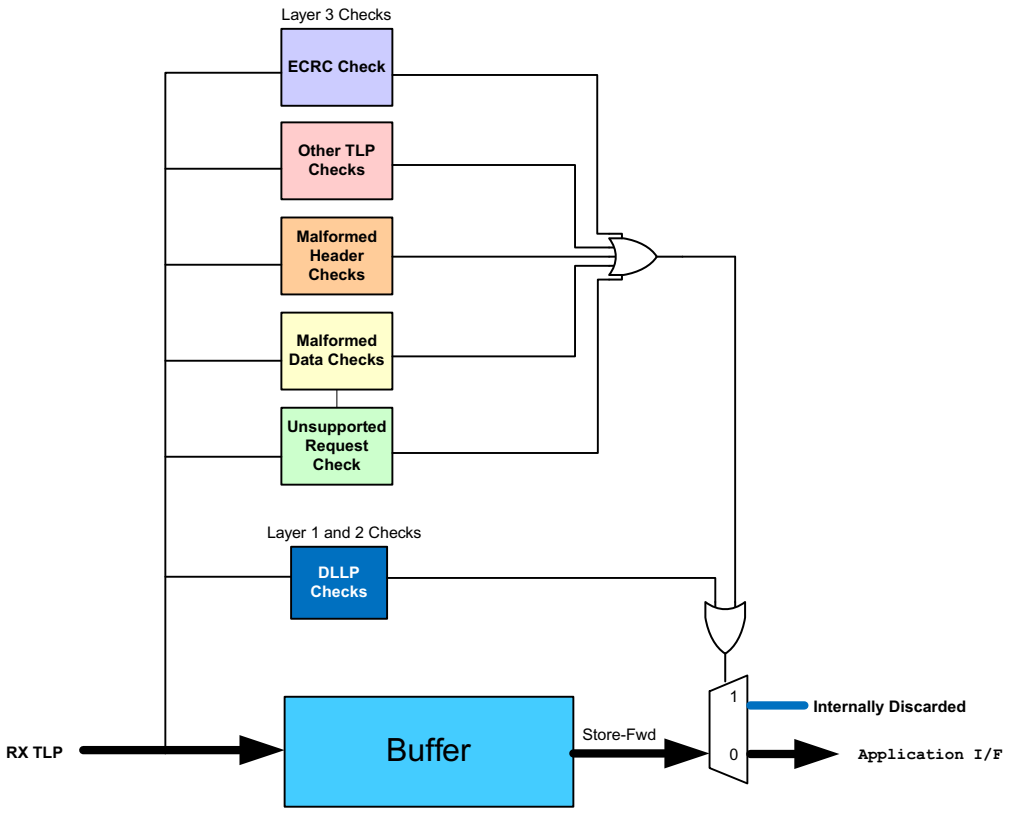


Table K-1 Processing of Received TLPs with Errors in Store-and-Forward Mode

Error Type	TLP Destination	Abort Error Outputs Asserted
Unsupported Request/Request with Malformed Header or Data / Completer Abort / CRS / Data Link Layer Error	Discarded[a]	None
ECRC[b] Error	Discarded	None
IDE errors	Discarded	None

a. Can be redirected to your application logic when the DEFAULT_TARGET (USP only) field in the MISC_CONTROL_1_OFF register is ‘1’.
b. If CX_FLT_MASK_ECRC_DISCARD =1, the TLP is forwarded to your application logic. In addition, radm_trgt1_ecrc_err is asserted if ECRC_ERR_PASS_THROUGH =1. The radm_trgt1_tlp_abort signal is never asserted.

K.1.2 Bypass Queue Mode

In general when the controller detects an error in a received TLP, it continues to forward the TLP to your application, and asserts the `*tlp_abort` or `*dllp_abort` signals to indicate that your application should drop the TLP and rollback its buffer pointers. Exceptions to this are detailed in Table K-2.

Figure K-2 Overview of Routing For TLPs with Errors in Bypass Mode

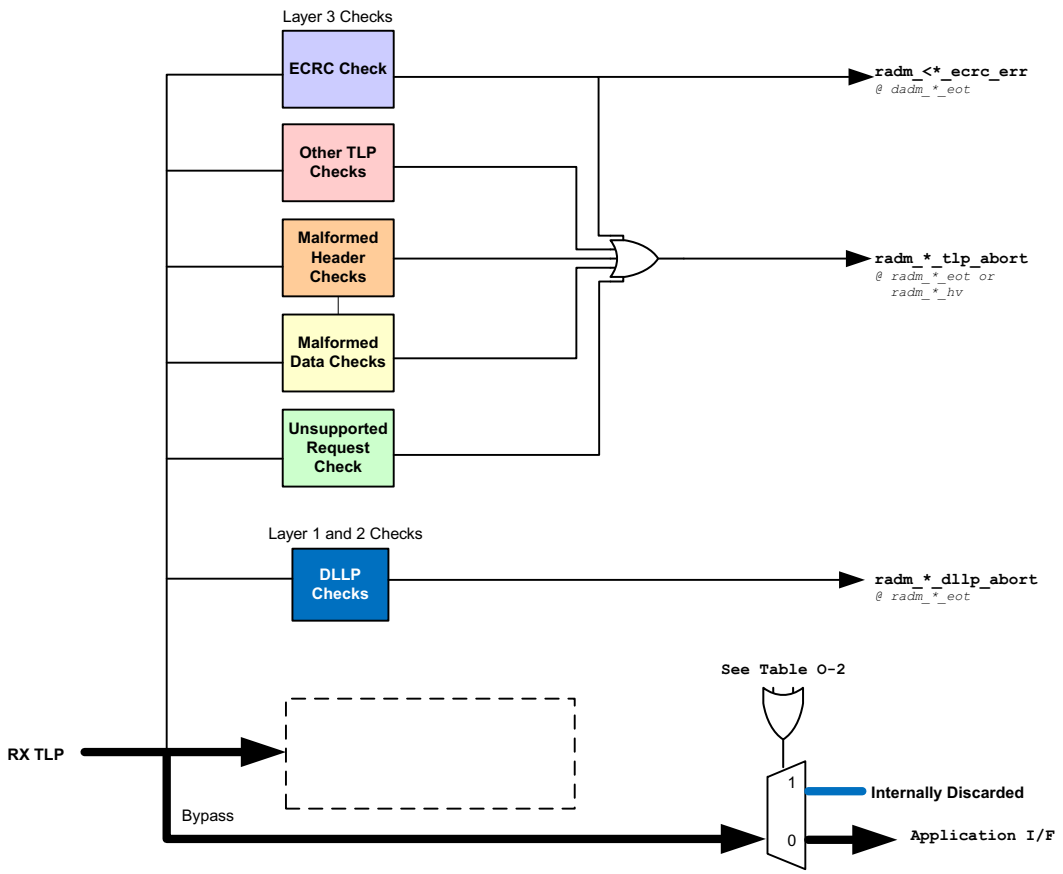


Table K-2 Processing of Received TLPs with Errors in Bypass or Cut-Though Mode

Error Type	TLP Destination	Abort Error Outputs Asserted
Unsupported Request / Completer Abort / CRS / Malformed Data	Application Logic	radm_<trgt1|bypass>_tlp_abort
Data Link Layer Error[a]	Application Logic	radm_<trgt1|bypass>_dllp_abort
ECRC[b] Error	Discarded	None
Request with Malformed Header	Application Logic	radm_bypass_tlp_abort
IDE errors	Application Logic	radm_<trgt1|bypass>_tlp_abort

a. If Layer 3 checks fail, then controller discards the request and does not assert radm_<trgt1|bypass>_tlp_abort.

b. If CX_FLT_MASK_ECRC_DISCARD =1, then TLP is forwarded to your application logic and radm_<trgt1|bypass>_ecrc_err is asserted (if ECRC_ERR_PASS_THROUGH =1). TLPs with errors are generally dropped unless the CX_FLT_MASK_ECRC_DISCARD filter rule is set, but because an ECRC error occurs at the end of the TLP, a TLP in cut-through mode with an ECRC error must be passed through. Note that cut-through mode decays into store-and-forward mode when the queues become full; if there is more than one TLP in the buffer when the queue is in cut-through mode, the TLPs being received are treated the same as store-and-forward mode. Only the first TLP in the queue behaves in cut-through mode.

Table K-3 Description of *_abort and *ecrc_err Outputs

Signal	Validated By	Notes
radm_*_tlp_abort	radm_*_eot[a]	The error is non-correctable. You should not expect the TLP to be replayed.
radm_*_dllp_abort	radm_*_eot	The error is correctable. You can normally[b] expect the DLLP to be replayed. Some DLL error conditions such as a missing end delimiter (EOT) can cause your applications receive buffer to overflow. Your application must implement some EOT timeout detect logic.
radm_*_ecrc_err	radm_*_eot	The error is non-correctable. You should not expect the TLP to be replayed.

a. Sometimes validated by radm_bypass_hv.
b. Except when the TLP fails a credit check (receiver overflow checking). In this case the controller ACKs the TLP. The assertion of radm_*_dllp_abort indicates that no credits are returned.

K.1.3 Disabling of Error Filtering

In many cases, the standard routing rules can be masked or ignored by setting the corresponding bit in the Symbol Timer and Filter Mask 1 register (`SYMBOL_TIMER_FILTER_1_OFF`) and Filter Mask Register 2 (`FILTER_MASK_2_OFF`). You can prevent all associated downstream effects such as error logging and completion generation (for non-posted requests).

You can also turn off most of the filter rules, and perform your own error checking for all TLPs, regardless of the receive queue mode. Your application must then report errors. This is not a recommended mode of operation. When you mask detection of completion timeout errors, through setting the `CPL_TIMEOUT_ERR_MASK` configuration parameter, then the controller does not automatically report completion timeout errors. Your application must check for completion timeouts, and report completion timeout errors using the "Application Error Reporting Interface" on page 724. Messages with UR status are always dropped, regardless of any changes that you make to the filter masks. For message routing see, "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.

Attention

When an LCRC error occurs on an inbound request which contains an ECRC, and the controller forwards the TLP to your application (queue is in bypass or cut-through), the radm_trgt1_dwen outputs are not accurate on the last beat (eot) of the TLP when the TLP payload (excluding ECRC) is not a multiple of the controller datapath width.

You must discard all data from this TLP including the additional data because radm_*_dllp_abort is active on radm_*_eot.

K.2 Routing of Request TLPs with Errors (AMBA Bridge Configuration)

When you are using the AXI bridge, the controller (native PCIe controller + AXI bridge) normally discards all TLPs with errors and never forwards them to your AXI application logic, unless you modify the default filtering setup as described in "Disabling of Error Filtering".

K.2.1 Disabling of Error Filtering

In many cases, the standard routing rules can be masked or ignored by setting the corresponding bit in the Symbol Timer and Filter Mask 1 register (`SYMBOL_TIMER_FILTER_1_OFF`) and Filter Mask Register 2 (`FILTER_MASK_2_OFF`). You can prevent all associated downstream effects such as error logging and completion generation (for non-posted requests).

You can also turn off most of the filter rules, and perform your own error checking for all TLPs, regardless of the receive queue mode. Your application must then report errors. This is not a recommended mode of operation in general but there are some cases where the application needs to handle errors. For example, the application may handle Poisoned TLP and Completion Timeout errors as Advisory Non-fatal errors. When you mask detection of completion timeout errors, through setting the `CPL_TIMEOUT_ERR_MASK` configuration parameter, then the controller does not automatically report completion timeout errors. Your application must check for completion timeouts, and report completion timeout errors using the "Application Error Reporting Interface" on page 724. As another example, to make the application handle Poisoned TLP errors, you need to turn off the controller's internal error reporting by setting `CX_FLT_MASK_POIS_ERROR_REPORTING` bit in `MASK_RADM_2` field of `FILTER_MASK_2_OFF` register. In addition, you need to set `CX_FLT_MASK_UR_POIS` bit in `MASK_RADM_1` field of `SYMBOL_TIMER_FILTER_1_OFF` register to make the controller not discard the TLP but forward it to the application. For AXI, see "Poisoned TLP Handling (RX Path)" on page 435. Messages with UR status are always dropped, regardless of any changes that you make to the filter masks. For message routing see, "Advanced Information: Advanced Filtering and Routing of TLPs" on page 691.

K.3 Routing of Completions with Errors (Native PCIe Controller)

Completion TLPs appear on the RBYP interface in bypass mode, and on the TRGT1 interface otherwise.

K.3.1 Store-and-Forward Queue Mode

In general when the controller detects an error in a received completion, it discards the completion and does not forward it to your application. Exceptions to this are detailed in Table K-4.

Figure K-3 Overview of Routing For Completion TLPs with Errors in Store-and- Forward Mode

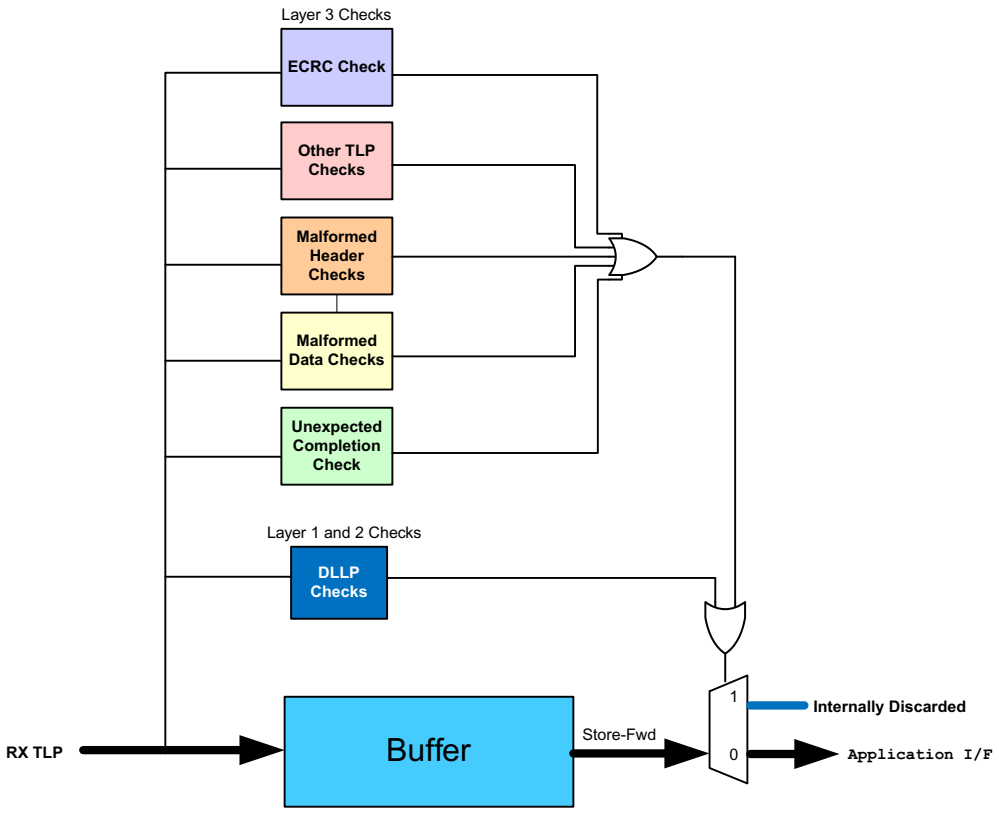


Table K-4 Processing of Received Completions with Errors in Store-and-Forward Mode

Error Type	CPL Destination	Abort Error Outputs Asserted
Replayed Completion/ Completion with Malformed Header or Data / Completion Status[a] / Data Link Layer Error	Discarded	None
Unexpected Completion	Discarded	None
ECRC[b] Error	Discarded	None
Unsuccessful Completion Status / Poisoned / Completion Timeout	Application Logic	None[c,d]
IDE errors	Discarded	None

a. Endpoint only.
b. If CX_FLT_MASK_CPL_ECRC_DISCARD =1, then the TLP is forwarded to your application logic. In addition, radm_trgt1_ecrc_err is asserted if ECRC_ERR_PASS_THROUGH =1. radm_trgt1_tlp_abort is never asserted.
c. Completion timeout: Requester information is returned on CPL Timeout interface.
d. Poisoned: radm_trgt1_ep is asserted.

K.3.2 Bypass/Cut-Through Queue Mode

In general when the controller detects an error in a received completion, it continues to forward the completion to your application, and asserts the `*tlp_abort` or `*dllp_abort` signals to indicate that your application should drop the TLP and rollback its buffer pointers. Exceptions to this are detailed in Table K-5.

Figure K-4 Overview of Routing for Completion TLPs with Errors in Bypass/Cut-Through Mode

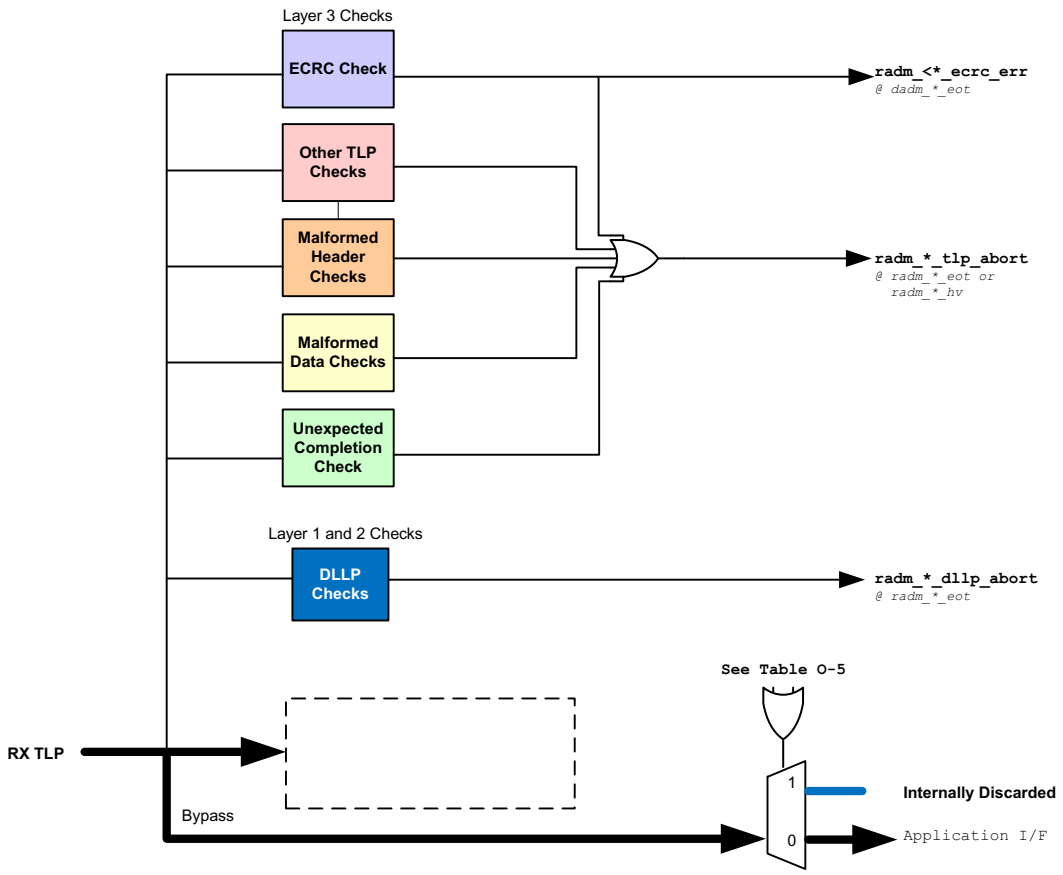


Table K-5 Processing of Received Completions with Errors in Bypass/Cut-Through Mode

Error Type	CPL Destination	Abort Error Outputs Asserted
Unsuccessful Completion Status / Poisoned / Completion Timeout	Application Logic	None[a,b]
Malformed Data / CRS Completion Status[c]	Application Logic	radm_bypass_tlp_abort

Error Type	CPL Destination	Abort Error Outputs Asserted
Data Link Layer Error[d]	Application Logic	radm_bypass_dllp_abort
ECRC Error	Application Logic	radm_bypass_tlp_abort
Unexpected Completion	Application Logic	radm_bypass_tlp_abort
Replayed Completion with Malformed Header	Application Logic	radm_bypass_tlp_abort
IDE error	Application Logic	radm_bypass_tlp_abort

a. Completion timeout: Requester information is returned on CPL Timeout interface.
b. Poisoned: radm_bypass_ep is asserted.
c. Endpoint only.
d. If Layer 3 header checks fail, then controller discards the completion and does not assert radm_bypass_dllp_abort.

Table K-6 Description of *_abort and *ecrc_err Outputs

Signal	Validated By	Notes
radm_*_tlp_abort	radm_*_eot[a]	The error is non-correctable. You should not expect the TLP to be replayed.
radm_*_dllp_abort	radm_*_eot	The error is correctable. You can normally[b] expect the DLLP to be replayed. Some DLL error conditions such as a missing end delimiter (EOT) can cause your applications receive buffer to overflow. Your application must implement some EOT timeout detect logic.
radm_*_ecrc_err	radm_*_eot	The error is non-correctable. You should not expect the TLP to be replayed.

a. Sometimes (for malformed TLP, UC or replayed CPL) validated by radm_bypass_hv.
b. Except when the TLP fails a credit check (receiver overflow checking). In this case the controller ACKs the TLP. The assertion of radm_*_dllp_abort indicates that no credits are returned.

Attention

When an LCRC error occurs on an inbound request which contains an ECRC, and the controller forwards the TLP to your application (queue is in bypass or cut-through), the `radm_trgt1_dwen` outputs are not accurate on the last beat (eot) of the TLP when the TLP payload (excluding ECRC) is not a multiple of the controller datapath width.

You must discard all data from this TLP including the additional data because `radm_*_dllp_abort` is active on `radm_*_eot`.

K.4 Routing of Completions with Errors (AMBA Bridge Configuration)

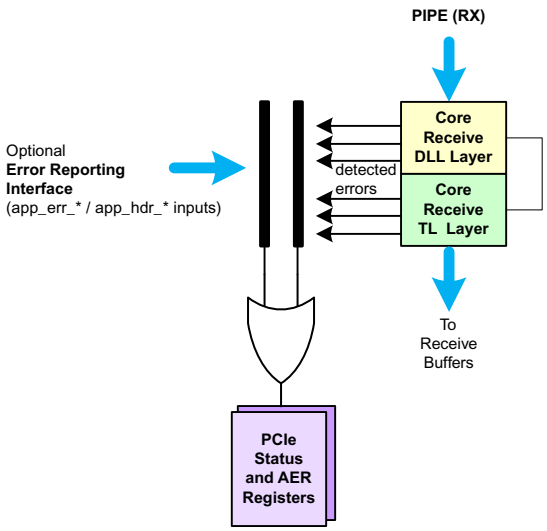
When you are using the AXI bridge, the controller (native PCIe controller + AXI bridge) normally[1] discards all TLPs with errors and never forwards them to your AXI application logic.

1. Unless you modify the default filtering setup. For more information, see "Disabling of Error Filtering" on page 719.

K.5 Application Error Reporting Interface

You can use this optional interface when you are implementing additional error checking on received TLPs in your application logic. For example, when you are implementing the "Reliability, Availability, and Serviceability (RAS)" on page 162. In this case, it is expected that your application implements an appropriate error handling scheme by monitoring the parity or ECC output signals. One way to use these signals is to "OR" them, and drive the Uncorrectable Error bit of the `app_err_bus` input.

Figure K-5 Error Logging (Downstream Port)



You must set the `APP_RETURN_ERR_EN` configuration parameter to enable the Application Error Reporting Interface.

Table K-7 Mapping of app_err_bus Bits to AER Status Registers

app_err_bus	Target Register Bit
0: Malformed TLP	AER Uncorrectable [18]
1: Receiver Overflow	AER Uncorrectable [17]
2: Unexpected CPL	AER Uncorrectable [16]
3: Completer Abort	AER Uncorrectable [15]
4: CPL Timeout	AER Uncorrectable [14]
5: Unsupported Request	AER Uncorrectable [20]
6: ECRC Check Failed	AER Uncorrectable [19]
7: Poisoned TLP Received	AER Uncorrectable [12]
8: AtomicOp Egress Blocked: valid only when `CX_ATOMIC_ROUTING_EN =1`	AER Uncorrectable [24]

app_err_bus	Target Register Bit
9: Uncorrectable Internal Error The controller sets the `INTERNAL_ERR_STATUS` bit in `UNCORR_ERR_STATUS_OFF` when your application asserts this input	AER Uncorrectable [22]
10: Corrected Internal Error	AER Correctable [14]
11: TLP Prefix Blocked Error Status: valid only for RP and when CX_NPRFX > 0	AER Correctable [25]
12: ACS Violation: only valid when `CX_ACS_ENABLE =1`	AER Uncorrectable[21]
13: Poisoned TLP Egress Blocked	AER Uncorrectable[26]
14: Deferrable Memory Write Request Egress Blocked	AER Uncorrectable[27]
15: IDE Checkfail Error	AER Uncorrectable[28]
16: IDE Misrouted Error	AER Uncorrectable[29]
17: IDE PCRC Error	AER Uncorrectable[30]
18: Data Link Protocol Error	AER Uncorrectable [4]
19: Flow Control Protocol Error	AER Uncorrectable [13]
20: Receiver Error	AER Correctable [0]
21: Bad TLP	AER Correctable [6]
22: Bad DLLP	AER Correctable [7]
23: REPLAY_NUM Rollover	AER Correctable [8]
24: Replay Timer Timeout	AER Correctable [12]
25: Advisory Non-Fatal Error	AER Correctable [13]
26: Header Log Overflow	AER Correctable [15]

Attention

When you report an error on this interface at the same time (exact same clock cycle) as when the controller is reporting an error internally, then there is a potential for Header Log Overflow to occur, as described in 6.2.4.2. “Multiple Error Handling” in the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The controller can only log one TLP header per function, and in this scenario it discards the header from the controller. However, both error bits are set in the Uncorrectable Error Status register, assuming that they are of different types.

L

Advanced Information: Calculating Gen1 Throughput

This appendix defines the throughput calculation (primarily) with respect to the PCI Express controller in Gen1 2.5 GT/s mode, and provides guidelines to enable you to meet your desired throughput by selecting the appropriate configuration parameters from those supplied with the controller.

This appendix 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 controller. The following topics are discussed.

- "PCI Express Bandwidth and Throughput"
- "Effective Throughput" on page 730

L.1 PCI Express Bandwidth and Throughput

PCI Express bandwidth is 2.5 Gb/s (gigabits per second), per lane, when operating at the Gen 1 data rate. The data is encoded using 8b10b encoding, so the effective maximum throughput is 250 MB/s (megabytes per second), per lane, calculated as follows:

2.5 Gb * 8b/10b =2 Gb * 1B/8b =250 MB/sec per lane

Throughput for other lane widths are shown in Table L-1.

Table L-1 Theoretical Throughput vs. Lane Width

Lane Width	Throughput after 8b/10b (MB/s)
x1	250
x2	500
x4	1000
x8	2000
x16	4000

L.1.1 Configuring the PCI Express Controller with Respect to Throughput

The PCI Express controller is available in 32, 64, and 128-bit data path widths (32 optimized for a one lane application). The PCI Express controller supports two clock frequencies. For the FPGA environment, the clock from the PHY and the controller clock can be configured to 125 MHz. For the ASIC, the controller 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 is different, based on the clock frequency selected in the controller.

Table L-2 shows the throughput for 32, 64, and 128-bit data path widths configured with a 125 MHz clock. Table L-3 shows the throughput for 32, 64, and 128-bit data path widths configured with a 250 MHz clock.

Table L-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 L-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

L.2 Effective Throughput

The effective throughput on the link is the payload throughput after all the PCIe protocol's overheads have been factored out. The key protocol features are:

- 8b10b encoding at the physical layer. This takes away 20% of the raw bandwidth.
- Acknowledge and flow control update packets at the Data Link Layer (DLLPs). This takes away 1% to 5% of the remaining bandwidth.
- Packet overhead (Figure L-1) at the transaction layer. Your design choices have a great effect here:
  - Small packets can take away 75% of the bandwidth.
  - Large packets can take away as little as 1%.

Figure L-1 TLP Packet (with associated Data Link Layer overhead bytes)

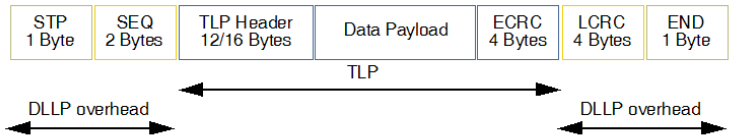


Table L-4 summarizes throughput calculations based on a payload size from 16 to 4K bytes.

Table L-4 Effect of Packet Overhead on Link Throughput

Packet Type	Data Payload Bytes	TLP Header Bytes	ECRC Bytes	DLLP and PHY Bytes	Calculation	Throughput before 8b10b encoding	Throughput after 8b10b encoding
Worst	16	16	4	8	16/44	36%	29%
Typical	128	16	0	8	128/152	84%	67%
	256	16	0	8	256/280	91%	73%
	512	16	0	8	512/536	96%	76%
Best	4096	12	0	8	4096/4116	99%	79%

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 76%.
- Increasing the payload size from 512 bytes to 4096 bytes, and using the smaller three DWORD header, only contributes an increase of 3% in the net throughput, and the storage requirements are more than doubled.
- In addition, a large payload size might 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 chip sets support 128 -byte or 256-byte payload sizes, with 512 bytes gaining in popularity.

A breakdown of the theoretical throughput for your specific configuration is available in the coreConsultant GUI “Report” tab.

L.2.1 Effect of Link Layer Flow Control and ACK/NAK DLLPs in Non-Flit Mode

DLLPs should be sent as often as required to avoid a negative impact on the TLP throughput. The controller sends a pending DLLP if there is no competing TLP traffic. Sending ACK/NAK and Update FC is controlled by timers. The controller 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 partner's retry buffer (payload) size might require a change in the ACK/NAK/Update FC time-out to obtain optimal system latency. The controller provides a feature to accumulate up to 255 ACKs before issuing an ACK. This feature offers-fine tuning of ACK frequency impact. The controller transmits ACKs at fixed intervals, by default. For more information, see "Ack Frequency and L0-L1 ASPM Control Register" `ACK_F_ASPM_CTRL_OFF`. Figure L-2 identifies the DLLP package. Table L-5 summarizes the throughput calculations.

Figure L-2 DLLP Package

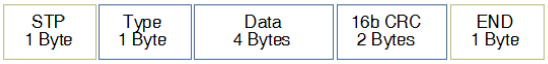


Table L-5 Effective Throughput

	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%

L.2.2 Other Factors Impacting Throughput

Replay Buffer and Receiver Queue Sizing

Replay buffer and receiver queue sizing should be optimal to avoid a harmful impact of larger ACK/NAK or Update FC latencies by the link partner. The controller performs an effective auto-sizing of the buffers taking into consideration the impact of these parameters.

M

Advanced Information: Calculating AXI Bridge Throughput

This appendix defines the throughput calculation (primarily) with respect to the PCI Express controller in Gen1 2.5 GT/s mode, and provides guidelines to enable you to meet your desired throughput by selecting the appropriate configuration parameters from those supplied with the controller.

This appendix 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 controller. It also presents some measured performance results for specific AXI controller configurations.

M.1 AXI Bridge Throughput Measurements

Performance figures for some specific AXI Gen1 controller configurations have been measured in simulation and are presented in Figure M-1 to Figure M-4. All data rates are evaluated after factoring out:

- 8/10b encoding
- Whole packet overhead (Link, Data, and Transaction Layer)
- Lane padding
- DLLP rate (as measured during simulations)

For comparison, the theoretical throughput is evaluated after factoring out:

- 8/10b encoding
- Whole packet overhead (Link, Data and Transaction Layer)
- Lane padding

The device under test (DUT) is configured for maximum performance and under the following conditions:

- AXI burst length is maximized up to a limit of 256 beats per burst.
- Single-buffer legacy queue architecture.
- A single traffic type is used for each test. That is, all reads or all writes. No mixed traffic is used.
- DLLP rate is minimized.
- Optimal buffer sizing for replay buffer and receiver queues. Maximum buffer size for bridge buffering resources.
- The maximum number of credits and tags are used.
- Requests do not cross an address page boundary and do not exceed the maximum transfer size. Therefore there is no decomposition.
- Each AXI transaction uses a unique ID. Therefore there are no AXI ordering rules to be enforced.
- Default ordering of PCIe rules is used.

Figure M-1 32-bit x4 Gen1 Controller @250MHz: 32-bit AXI @250MHz: Memory WRITES

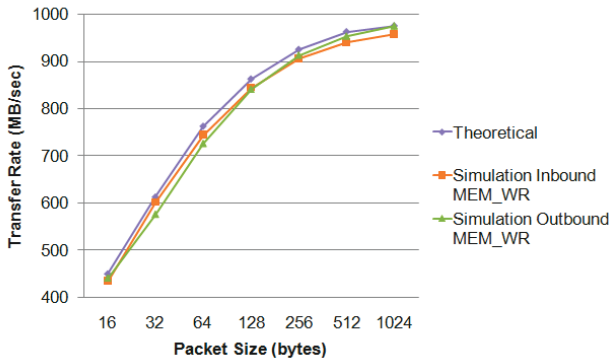


Figure M-2 32-bit x4 Gen2 Controller @500MHz: 64-bit AXI @500MHz: Memory WRITES

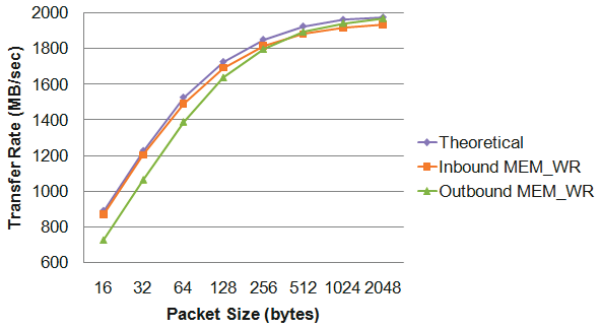


Figure M-3 32-bit x4 Gen1 Controller @250MHz: 32-bit AXI @250MHz: Memory READS

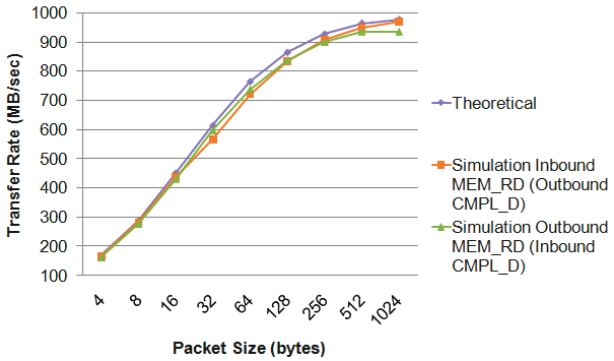
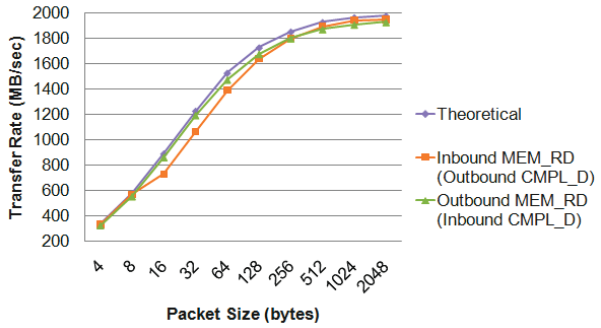


Figure M-4 32-bit x4 Gen2 Controller @500MHz: 64-bit AXI @500MHz: Memory READS

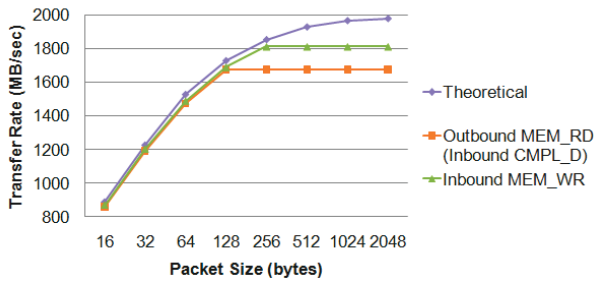


M.1.1 Areas for Maximizing AXI Bridge Performance

Decomposition

A detailed description of what decomposition is, what causes it and how to avoid it is available at "AXI Decomposition Rules" on page 435. Unnecessary decomposition results in the generation of smaller TLPs which increases total TLP header overhead. Unwanted decomposition also increases the consumption of PCIe tags for NP transactions.

Figure M-5 Effect of Decomposition on Memory READS where Max Payload Size =256 bytes and Remote Max Read Request Size =128 bytes



"Max Payload Size" (`CX_MAX_MTU`) is 256 bytes so that any Outbound MWr with a payload greater than 256 bytes is decomposed by the AXI bridge subordinate.

"Remote Max Read Request Size" (`CX_REMOTE_RD_REQ_SIZE`) is 128 bytes so that all inbound MRds request no more than 128 bytes.

FIFO Sizes

The subordinate decomposer FIFOs offload the AXI bus while waiting for access to the PCIe transmit link. They should be large enough to handle a burst and also to smooth out the transition from AXI to PCIe.

Number of Tags (AXI SPLIT_RETRY mode)

The number of PCIe tags directly determines the number of simultaneously outstanding read operations that the AXI bridge can generate. The greater the number of tags, the better the performance in a system where the read data is slow in returning to the device.

M.1.2 Known Performance Limitations in AXI

Posted and Non-Posted Ordering

The configuration specified in Table M-1 illustrates a performance loss for some kind of traffic profiles. When non-posted requests are received while posted requests are in the receive queue, then non-posted requests are sent only when all posted requests have been forwarded.

Table M-1 Configuration Parameter Settings That Cause Non-Posted Transaction Starvation

Parameter Name	Value
CX_RADMQ_MODE	Segment Buffer
RADM_CPL_QMODE_VC0	Store / Fwd

Posted and Completion Ordering

The configuration specified in Table M-2 illustrates a performance loss for some kind of traffic profiles. When completions are received while posted requests are in the receive queue, then the completions are sent only when all posted requests have been forwarded. This is to observe PCIe Ordering Rules

Table M-2 Configuration Parameter Settings That Cause Non-Posted Transaction Starvation

Parameter Name	Value
CX_RADMQ_MODE	Segment Buffer
RADM_CPL_QMODE_VC0	Store / Fwd

N

Advanced Information: Advanced Routing of Received Messages

This appendix discusses advanced features and operation associated with "Message Reception" on page 205. You should first read to familiarize yourself with basic information on the filtering and routing of received message TLPs.

Note

When the AXI bridge is configured, TRGT1 / RBYP become internal interfaces connecting RADM and AXI bridge master, and the messages are forwarded to your application on the AXI bridge master instead of TRGT1 / RBYP.

When the AXI bridge is configured, XALI0 / XALI1/XALI2 become internal interfaces connecting AXI bridge subordinate and XADM, and the messages are transmitted by your application on the AXI bridge master instead of XALI0 / XALI1/XALI2.

N.1 Routing of Received Messages

All error-free MSG requests are decoded internally, signaled on the SII interface and then dropped (not forwarded to your application[1] on TRGT1 or RBYP). To configure the controller to send messages to your application, then either (a) set the `FLT_DROP_MSG` configuration parameter to 0 or (b) set the register fields outlined in Table N-1 to 1. These registers allow you to override any decisions (regarding MSG routing) made at configuration time by the `FLT_DROP_MSG`, `DEFAULT_FILTER_MASK_1`, and `DEFAULT_FILTER_MASK_2` configuration parameters. When a MSG request is filtered with UR/CA/CRS status, the TLP is always dropped. Only MSG requests filtered with SC status, can potentially be forwarded to your application on TRGT1.

Table N-1 Controlling the Routing of Received Messages

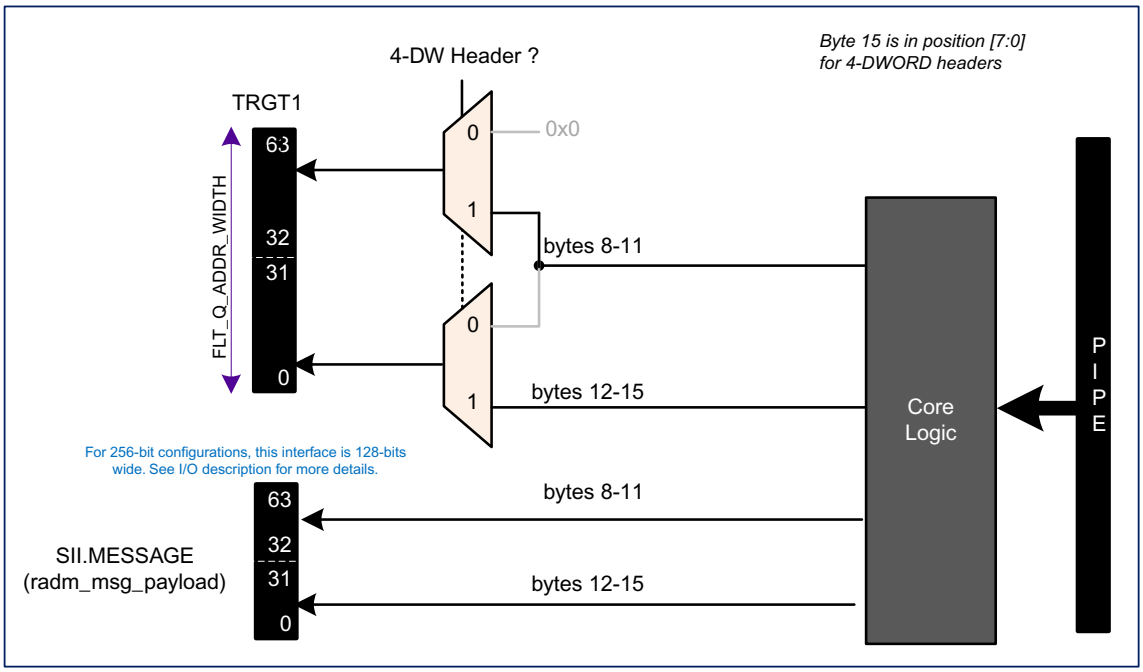
Register	Bit	Function	Default Value
Filter Mask Register 1	29	Mask the dropping of Non-Vendor Messages 0: Drop; 1: Do not drop	DEFAULT_FILTER_MASK_1[13] = ! FLT_DROP_MSG =0
Filter Mask Register 2	0	Mask the dropping of Vendor[a] Type0 Messages 0: Drop[b]; 1: Do not drop	DEFAULT_FILTER_MASK_2[0] =0
Filter Mask Register 2	1	Mask the dropping of Vendor1 Type1 Messages 0: Drop; 1: Do not drop	DEFAULT_FILTER_MASK_2[1] =0

a. For the masking (of the dropping) of Vendor messages, it is not possible to differentiate between Vendor Message without Payload (Msg) and Vendor Message with Payload (MsgD).
b. Vendor Type0 Messages are dropped with UR error reporting.

Full information of the Filter Mask Registers are in Symbol Timer and Filter Mask 1 register (`SYMBOL_TIMER_FILTER_1_OFF`) and Filter Mask 2 register (`FILTER_MASK_2_OFF`).

1. Vendor TYPE0 messages causes an UR error.

Figure N-1 Received 3rd and 4th Message Header Byte Mapping at Interfaces (native PCIe controller)



For more information, see the I/O descriptions of:

- `radm_cpl_data`
- `radm_bypass_data/radm_bypass_addr`
- `radm_trgt1_addr`
- `radm_msg_payload`

N.1.1 Message Reception on TRGT1

As previously mentioned in "Routing of Received Messages" on page 740, the controller does not route messages to your application on TRGT1. To configure the controller to send messages to your application, you must set the register fields outlined in Table N-1 to 1. To access the third and fourth TLP header DWORDs, you must support 64-bit addressing in your application.

N.1.2 Message Reception Considerations when Using AXI Bridge

When the AXI bridge is used, the third and fourth TLP header DWORDs (Bytes 8 to 15) are routed to your application through the AXI sideband (misc. bus) signals.

In the scenario where the received message is translated using the iATU Msg Type Match and the Single Address Location feature is enabled, the third and fourth TLP header DWORDs are routed to your application through the AXI manager databus. For more information, see "Message Type Match" on page 244.

Table N-2 describes the mapping of received Msg information using the AXI Signals.

Table N-2 Received Msg Information and AXI Signal Mapping

Received Msg Information		AXI Signal
Normal Operation	**iATU Msg Type Match Operation with Single AddressLocation Feature Enabled**	
TLP Type		TYPE field of `mstr_awmisc_info`/ `mstr_req_misc_info`
TLP Code		`mstr_awmisc_info[7:0]`
Bytes 8 to 15 of TLP header	0	`mstr_awmisc_info_hdr_34dw[63:0]`
Payload (MsgD[a] only)	Bytes 8 to 15 of TLP Header[b] andPayload[c] (MsgD[d] only)	AXI Manager Databus
`AXI_MSTR_MSG_ADDR_LOW_OFF` and `AXI_MSTR_MSG_ADDR_HIGH_OFF` registers	ATU Target address configured for the matched region	`mstr_awaddr`

a. Message with payload. That is (i) VDM with payload, and (ii) set_slot_power_limit.
b. TLP header DWORDs are placed on: - The first data beat (if MASTER_BUS_DATA_WIDTH > 32), or- First two data beats (if MASTER_BUS_DATA_WIDTH = 32)
c. To prevent decomposition and data corruption, you must ensure that Payload <= CC_MSTR_MTU-2 DWORDs
d. Message with payload. That is (i) VDM with payload, and (ii) set_slot_power_limit.

You must ensure that messages generated by the remote device do not trigger decomposition[1] as described in "AXI Decomposition Rules" on page 435.

Caution The Message should be consumed by your application before the next message arrives as all messages go to the same address.Address configured in AXI_MSTR_MSG_ADDR registers / iATU TRGT address registers should be aligned with the AXI manager databus width.

1. Or data corruption occurs.

Figure N-2 Received 3rd and 4th Message Header Byte Mapping at Interfaces

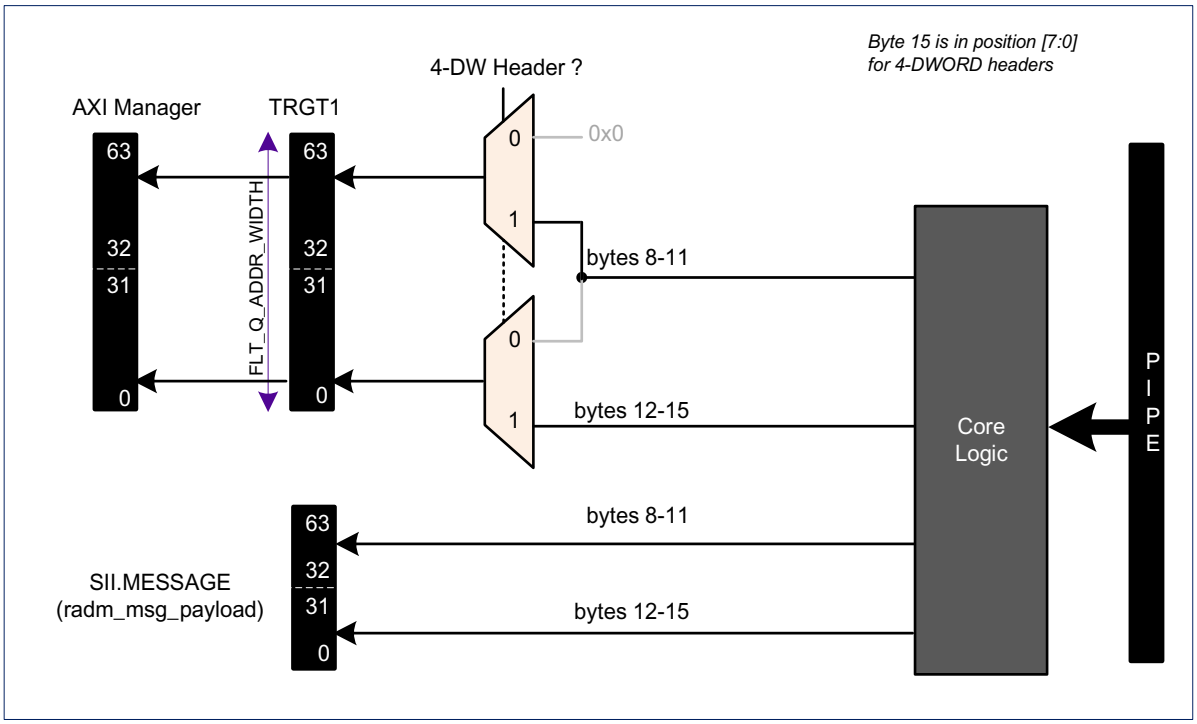


Table N-3 Processing of Inbound Messages when AXI and PCIe Controller Address Widths are Different

AXI Address Bus Width MASTER_BUS_ADDR_WIDTH	PCIe controller Address Bus Width FLT_Q_ADDR_WIDTH	Notes
32	64	The upper 32 bits of header field (bytes 8-11) are not forwarded by the bridge master.
64	32	The upper 32 bits of header field (bytes 8-11) are forced to 0.

Note

- It is preferable to use the SII interface to send and receive OBFF/LTR messages. The bridge (subordinate) can support message transmission of OBFF/LTR messages without any modifications the bridge master cannot support reception of OBFF/LTR messages.
- A full set of limitations for the AXI bridge is available at “AXI Bridge Limitations” on page 398.

O

Advanced Information: Replay Buffer Sizing

This appendix explains how coreConsultant automatically calculates the size of the "Transmit Replay" on page 126. The retry buffer should be large enough so that, under normal operating conditions, back-to-back transmission of maximum sized packets (of size `CX_MAX_MTU`) is never throttled by a full retry buffer. Normal operating conditions mean ACK DLLPs are received as opposed to NAK DLLPs. In determining the optimal buffer size, one must consider the following components which are additive:

1. The remote ACK_NAK latency timer starts when the last symbol of a TLP is received. Therefore one maximum sized packet from the local device must be stored in the replay buffer to accumulate transmit TLP data.
2. An Ack DLLP is not scheduled by the remote device until its ACK_NAK latency timer has expired. During the time interval between start and expiry of the ACK_NAK latency, the replay buffer must have additional space to accumulate transmit TLP data.
3. When the ACK_NAK latency timer expires in the remote device, there is a delay to the transmission of the Ack DLLP if the remote device has just started transmitting a TLP packet. Therefore additional space to store a remote maximum sized packet needs to be added to the local replay buffer size to accumulate transmit TLP data.
4. The L0s exit latency required by the device's receiver.
5. The delays caused by the:
   a. Transmission of a TLP by the local device.
   b. Processing of TLP by the remote device.
   c. Transmission of a Ack DLLP by the remote device.
   d. Processing of a Ack DLLP by the local device.
   e. Round trip latency of Re-timer device (if present). This latency is calculated from `CX_MAX_RETIMER` value.

The equation to derive this depth is as follows. The label in each {} represents the contribution by each factor detailed in the previous numbered list:

```
depth =data_size{1} + data_size{3} + (larger of(l0s_adj{4} or
retimer_latency{5.e})+ ack_lat_limit{2} + remote_dly_factor{5.b and 5.c})/cx_nb +
cx_internal_delay{5.a and 5.d}
```

where:

- `data_size{1}` and `data_size{3}` are both the `CX_MAX_MTU` size expressed in units of controller data width. For example, a 1024 byte `CX_MAX_MTU`for a 128-bit controller is 1024*8/128 =64 entries.
- `l0s_adj{4}`is `(CX_NFTS+1)*4`because there are four symbols in a fast training sequence (FTS).
- `ack_lat_limit{2}` comes directly from Tables 3-7,3-8, and 3-9 of the *PCI Express Base Specification, Revision 6.0, Version 1.0*.
- `cx_nb` is the number of symbols processed at the PIPE interface per lane For example, the value is two symbols (16-bit PIPE) for a Gen2 controller running at 250 MHz.
- `remote_dly_factor{5.b and 5.c}` are the MAC and PHY delays (Tx and Rx) in the remote device. To model this, Synopsys uses the *PCI Express Base Specification, Revision 6.0, Version 1.0* recommendation of a constant value of 19 Symbol Times for 2.5 GT/s mode operation, 70 Symbol Times for 5.0 GT/s mode operation, and 115 Symbol Times for 8.0 GT/s and 16 GT/s mode operation.
- `cx_internal_delay{5.a and 5.d}`are the MAC and PHY delays (Tx and Rx) of the local device.

It is possible that this equation gives a retry buffer depth which is too large for your requirements. For example, referring to `item {3}`, this can happen if the maximum packet size that is transmitted by the remote device is smaller than `CX_MAX_MTU` of the local device (the equation assumes these are the same)

OR

you choose to trade off some temporal drop in performance versus back pressure on the local transmit client interface for the retry buffer depth.

For more information on the actual parameters used, see “Parameter Descriptions” chapter of the *Reference Manual*.

P

Advanced Information: Endianness

PCIe transfers information as a serialized stream of bytes. At the byte level, information is transmitted and received over the wire with the leftmost byte(byte 0 in Figure P-1) being transmitted first. However, endianness is only relevant when considering the arrangement of bytes within (for example) a DWORD. Reading the *PCI Express Base Specification, Revision 6.0, Version 1.0*, it is possible to interpret parts of the TLP header byte ordering as big endian. For example, the most significant byte of the address field is transferred first so that it may be used for early address decode. However, the arrangement of the payload (data) is little endian.

Figure P-1 PCIe Express Specification Byte Ordering (Four DWORD Header)

Order of PIPE Transmission	**Byte 0**	Byte 1	Byte 2	Byte 3	
Byte 0	R / Fmt / **Type**				Byte 3
Byte 4					Byte 7
Byte 8	Adr[63:56]	Adr[55:48]	Adr[47:40]	Adr[39:32]	Byte 11
Byte 12	Adr[31:24]	Adr[23:16]	Adr[15:8]	**Adr[7:2]** / R	Byte 15
Byte 16	**Data[7:0]**	Data[15:8]	Data[23:16]	Data[31:24]	Byte 19

In a similar way, the controller implements the headers[1] as big endian and the datapath[2] as little endian. Figure P-2 shows the example of a 32-bit controller, where:

- First payload byte(byte 16 in Figure P-1 or Figure P-2, and byte 12 in Figure P-3) is at location (7:0)
- First address byte(byte 15 in Figure P-1 or Figure P-2, and byte 11 in Figure P-3) is at location (31:24)

1. Including header address and message data bytes on the following interfaces: radm_bypass_addr, radm_trgt1_addr, client0_tlp_addr, radm_msg_payload, and ven_msg_data.
2. Including payload bytes on the following interfaces: radm_cpl_data, radm_bypass_data, radm_trgt1_data, and client0_tlp_data.

Figure P-2 Synopsys Controller Internal Byte Ordering (Four DWORD Header)

Order of PIPE Transmission	Byte 3		Byte 2	Byte 1	**Byte 0**			
Core Datapath	31 24		23 16	15 8	7 0			
Byte 3	F				R	mt	**Type**	Byte 0
Byte 7								Byte 4
Byte 11	Adr[39:32]		Adr[47:40]	Adr[55:48]	Adr[63:56]			Byte 8
Byte 15	**Adr[7:2]**	R	Adr[15:8]	Adr[23:16]	Adr[31:24]			Byte 12
Byte 19	Data[31:24]		Data[23:16]	Data[15:9]	**Data[7:0]**			Byte 16

Figure P-3 Synopsys Controller Internal Byte Ordering (Three DWORD Header)

Order of PIPE Transmission	Byte 3		Byte 2	Byte 1	**Byte 0**			
Core Datapath	31 24		23 16	15 8	7 0			
Byte 3					R	Fmt	**Type**	Byte 0
Byte 7								Byte 4
Byte 11	**Adr[7:2]**	R	Adr[15:8]	Adr[23:16]	Adr[31:24]			Byte 8
Byte 15	Data[31:24]		Data[23:16]	Data[15:9]	**Data[7:0]**			Byte 12

Therefore, you do not have to do byte reordering in your application. The controller exactly maps the address bus to the address header field, and the data to the payload. Taking the example of `client0_tlp_addr`, the bits in this address bus map directly to the address bits in the TLP header. The third and fourth DWORDs of the header are mapped as follows:

- 3-DWORD header
  - Third DWORD (bytes 8-11) =`client0_tlp_addr[31:0]`
  - Therefore, bits [7:0] are mapped to byte 11 which is the lowest eight bits of the TLP header address.
- 4-DWORD header
  - Third DWORD (bytes 8-11) =`client0_tlp_addr[63:32]`
  - Fourth DWORD (bytes 12-15) =`client0_tlp_addr[31:0]`
  - Therefore, bits [7:0] are mapped to byte 15 which is the lowest eight bits of the TLP header address.

This simple I/O bus bit mapping to TLP header byte mapping also applies to `radm_bypass_addr`, `radm_trgt1_addr`, `radm_msg_payload`, and `ven_msg_data`.

Figure P-4 Transmitted 3rd and 4th Message Header Byte Mapping at Interfaces

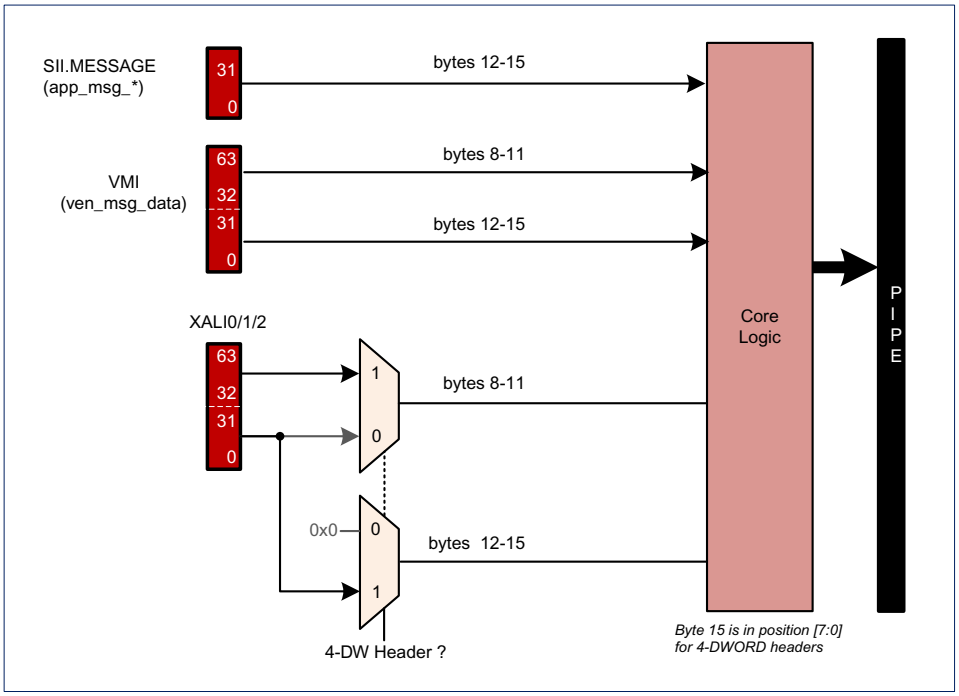
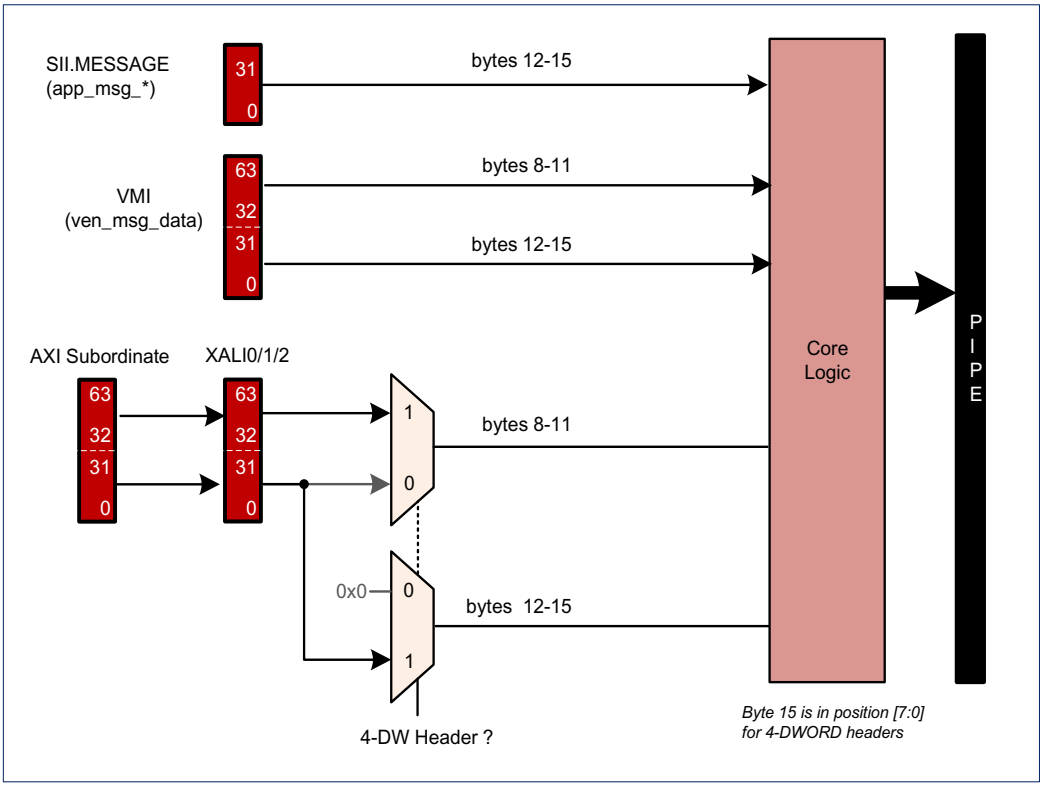


Figure P-5 Transmitted 3rd and 4th Message Header Byte Mapping at Interfaces (with AXI Bridge)



For more information on the format of message headers, see:

- "Routing of Received Messages" on page 740
- "Byte Mapping of Third and Fourth Message Header DWORDs at I/O Interfaces" on page 202

Note The optional TLP prefixes are implemented as little endian. Using the example of a controller with just one prefix, this means that `client0_tlp_prfx[31:0]` has byte 0 (the byte with the FMT and Type fields) of the prefix in the lower byte position of the DWORD `client0_tlp_prfx[31:0]`. That is, FMT =bits 7:5, and Type =bits 4:0

Q

Advanced Information: Area, Power Estimates, and RAM Sizes

Synopsys provides area, power consumption, and RAM figures in Table Q-1 for several example configurations[1] of the controller. The information provided here is intended to be used only as a guideline. For more information, contact Synopsys support through SolvNetPlus.

Table Q-1 Example Area, Power Consumption, and RAM Sizes

Description	Link
Area	http://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/PCIe_Gate_Count_ASIC.pdf
Power Consumption for slow, typical, and fast corners	https://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/PCIe_Power.pdf
RAM Size[a]	https://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/PCIe_RAM_Size_ASIC.pdf

a. Synopsys recommends that you use coreConsultant to generate the RAM size report for your configuration.

1. The configurations used are the default configurations, except for the parameters that are listed in the tables.

Q.1 Area Estimation

The area numbers are in kilo-gates. The total cell area is divided by the area of the x1 drive strength NAND cell in the library to get the gate count numbers. Synthesis results are generated using Design Compiler in topographical mode. The results do not include RAMs, scan chains, and scan logic.

The `CX_MAX_TAG/CX_REMOTE_MAX_TAG` parameter values have been changed from their default of 32/32 to the more realistic values of 16/7 for AXI configurations and 16/15 for other configurations.

Q.2 Power Estimation

Power values are generated using PrimeTime PX which uses the I/O and flip-flop switching activity file from an RTL simulation. The analysis complies with standard power analysis practices:

- Synopsys uses PrimeTime's time-based power analysis on the synthesized netlist.
- Parasitics are generated by topographical synthesis.
- Clock gating is inserted.
- The traffic profile used to generate the waveform FSDB used to measure the power is bespoke to each particular configuration.
- Power is measured only during an active traffic window; link training, reset states, registers, and configuration are not included in the power estimation.
- Typically, the traffic profile sends interleaved Reads and Writes in both direction and aims for good switching activity.

```
Total Power = Switch Power + Leakage Power + Internal Power
```

Table Q-2 Components of Total Power Consumption

Component	Description
Switch Power	The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output.
Leakage Power	Leakage power is the power dissipated by a cell when it is not switching, that is, when it is inactive or static.
Internal Power	Internal power is the dynamic power dissipated within the boundary of a cell. It includes the power dissipation due to charging or discharging of capacitances internal to the cell during switching; and the power dissipation due to the momentary short circuit between the P and N transistors of a gate while both are turned on.

Q.3 RAM Sizing

The size of the RAMs required is highly dependent on many configuration parameters, the main ones being:

- Link width (CX_NL)
- Flow control credits
- Number of virtual channels (CX_NVC)
- Maximum payload size (CX_MAX_MTU)
- CX_MAX_TAG when completion buffer is not in bypass mode
- CX_REMOTE_MAX_TAGS when AXI bridge is used

Q.3.1 Sizing the AXI Outbound Completion Composer RAM

You can resize AXI Subordinate TLP Completion Data Segment Composer FIFO RAM (u_ob_ccmp_data_ram) using the CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN parameter.

The u_ob_ccmp_data_ram has a segmented-buffer architecture to reorder the Slave Read Data. One segment is reserved per PCIe tag.

Table Q-3 u_ob_ccmp_data_ram Parameter Dependency

Parameter	Determines
SLAVE_BUS_DATA_WIDTH or CX_NW*32 (whichever is largest)	RAM width
■ CX_MAX_TAG, or ■ If DMA is enabled, CX_MAX_TAG - CC_NUM_DMA_RD_TAG	Number of segments
CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN	Depth of each segment

When you change the CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN parameter value the following are the repercussions:

- When you resize the RAM, it affects the performance of Slave Read traffic (initiated in AXI Subordinate Read, through PCIe Outbound Read Requests, RADM TRGT1, and ending in Slave Read)
  - If your traffic profile consists mainly of requests lesser than the CC_RADMX_COMPOSER_MAX_SEG_-BYTE_LEN parameter, you can decrease the RAM size without affecting performance.
  - Conversely, if your traffic profile consists mainly of requests greater than the CC_RADMX_COMPOS-ER_MAX_SEG_BYTE_LEN parameter, you can increase the throughput by increasing the RAM size.

Note The AXI Bridge does not issue any PCIe Outbound Read Requests with a value greater than the CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN parameter. It also decomposes any requests that exceed this length. For more information, see "Outbound Decomposition" on page 435.

- AXI Subordinate Read Address Channel interface:
  - For each request, this interface halts (slv_arready =0) for one clock cycle per decomposition.
  - If you set the CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN parameter value to a very low value, excessive decomposition halts the interface after all PCIe tags are exhausted.

- One PCIe wire tag is used per outstanding decomposed PCIe request.

- RADM TRGT1 interface
  - If you set the `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN` parameter value to a very low value, the AXI Bridge decomposition into smaller PCIe requests might result in lower CPLD throughput due to lower TLP efficiency (header-to-payload ratio).

You can automatically validate (using the coreConsultant) the selected `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN` parameter value to ensure that:

1. The RAM is large enough to avoid functional issues or deadlocks.
   - It must be able to store a value equal to the `CC_SLV_MTU` parameter value, when the application sets the `PCIE_CAP_MAX_READ_REQ_SIZE` field of the `DEVICE_CONTROL_DEVICE_STATUS` register to 0x0 (lowest value, that is 128 bytes)[1], and
   - Each segment must be able to hold at least one beat of the Slave or the controller data bus (whichever is largest)
2. The RAM must be small enough to be fully utilized. A single segment must not be greater than the value of the `CC_SLV_MTU` parameter.

Q.3.1.1 Adjusting Segment Byte Length

There is always an implied area-to-performance tradeoff when you set the buffer size. You can adjust it according to your design goals and expected traffic pattern. You can optimize your buffer size by using the following methods:

- Recommended value based on bandwidth and expected latency

  Using this method you can find a minimum segment length that matches the overall system bandwidth:

  - `RAW_REC_BYTE_LEN =((2^CX_MAX_PCIE_SPEED * CX_NL * ROUND_TRIP_LATENCY)/8 - 20)/BRIDGE_N_TAGS`

    where:

    - `BRIDGE_N_TAGS =CX_MAX_TAG + 1 - (CC_DMA_ENABLE ? CC_NUM_DMA_RD_TAG : 0)`
    - `BRIDGE_N_TAGS >=1 + CC_SLV_MTU/128`. If not, increase the number of tags.
  - `MIN_BLEN_1BEAT =min(CC_SLV_MTU, max(4*CX_NW, SLAVE_BUS_DATA_WIDTH/8))`
  - `MIN_BLEN_MTU =CC_SLV_MTU/(BRIDGE_N_TAGS-1)`

  Choose lowest value of `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN`, `which is >= min(CC_SLV_MTU, max(RAW_REC_BYTE_LEN, MIN_BLEN_1BEAT, MIN_BLEN_MTU))`. This is described with an example in the Table Q-4

  This method assumes that you perform most Read Requests with a size equal to or larger than `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN`. If you want to use lower request sizes, you can use a smaller segment length value.

1. Regardless of your segment byte length value, the following expression must be true: CX_MAX_TAG - (CC_DMA_ENABLE ? CC_NUM_DMA_RD_TAG : 0) >= CC_SLV_MTU/128. Additionally, if you configure the CX_MAX_TAG > 31 and disable the Extended Tag Support, ensure this rule is followed during the controller operation by limiting your slv_arlen according to "AXI Bridge Limitations" on page 398.

Table Q-4 Example - Value Based on Bandwidth and Expected Latency

Configured Values	Calculated Values
■ CC_DMA_ENABLE =1 ■ CX_MAX_TAG =127 ■ CC_NUM_DMA_RD_TAG =64 ■ CC_SLV_MTU =256 ■ CX_MAX_PCIE_SPEED =4 ■ ROUND_TRIP_LATENCY =500 ■ CX_NL =8 ■ CX_NW =8 ■ SLAVE_BUS_DATA_WIDTH =256	■ BRIDGE_N_TAGS =64 >=1+ 256/128 ■ RAW_REC_BYTE_LEN =124.7 ■ MIN_BLEN_1BEAT =32 ■ MIN_BLEN_MTU =4.1
Recommended CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN value =128	

- Sizing by high level design goals

  You can streamline your application and verification logic by adjusting the segment size using the design goals. If you set it significantly greater than the recommended segment length, in most cases, it does not improve the overall system bandwidth and or latency, as saturation occurs in other parts of the system.

  You can follow these strategies for sizing the segment length:

  - **Critical Payload Size**: You can configure each segment size as per your critical payload size where
    - any request equal to or lesser than this size has maximum performance, and
    - any request greater than this size incurs decomposition
  - **Slave Request Burst**: You can configure each segment size as per your AXI Subordinate Read Address Channel burst target. To avoid the `slv_arready` halt, due to segment length decomposition, ensure that `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN` is equal to or greater than your request length. To avoid the halt due to PCIe wire tag exhaustion, ensure the buffer can store all the data for your request burst.

    For example, if you have the following values and want to ensure that you can perform a burst of 32 AXI 2048B Read Requests without wire tag exhaustion halt:

    - `CX_MAX_TAG =63`
    - `CC_SLV_MTU =2048B`
    - `CC_MAX_SLV_TAG >=32`
    - `Max Read Request Size >=2048B`
    - `CC_DMA_ENABLE =0`

    Then `CC_RADMX_COMPOSER_MAX_SEG_BYTE_LEN >=(32*2048)/(63+1) =1024B`
  - **Outstanding Data Bytes**: You can configure each segment size as per your `CX_MAX_MTU` parameter value to ensure maximum CPLD efficiency. You can also increase it further up to `CC_SLV_MTU` parameter value to increase the amount of outstanding data bytes allowed by the AXI Bridge.

  You can additionally adjust this value by evaluating if the u_ob_ccmp_data_ram reaches a healthy level of utilization. Typically, low area applications can benefit from a segment byte length decrease from the default value, with the opposite case being true for high performance applications.

Q.4 SR-IOV Gate Count Costs

When you configure the controller to support SR-IOV, then you incur the following additional logic:

- Basic SR-IOV functionality adds 50K gates
- Each additional physical function results in about 20K extra gates
- Each additional virtual function results (VF) in about 2K extra gates
- When you enable MSI (optional) for VFs, add 1.6K gates for each VF
- When you enable AER (optional) for VFs, add extra 2K gates for each VF

R

Advanced Information: Flow Control Credit Calculation

This appendix explains how coreConsultant automatically calculates the default buffer sizes and credits for each TLP type from the number of lanes, the controller datapath width, the maximum PCIe payload (`CX_MAX_MTU`), flow control (FC) update latencies, internal delays, and the PHY latency. For more information on flow control, see "Flow Control" on page 216.

R.1 Calculation of Flow Control Latency

The Flow Control update latency is determined by a table lookup based upon values in "UpdateFC Transmission Latency Guidelines" table of the *PCI Express Base Specification, Revision 6.0, Version 1.0*. The specification adds 19, 51, and 115 symbol times (at Gen1, Gen2, and Gen3 speeds respectively) to the flow control latency value.

R.2 Calculation of Initial Flow Control Credits and Receive Buffer Sizes

The number of credits advertised should be large enough so that under normal operating conditions, back-to-back transmission of maximum sized packets (MTUs) is never throttled by a lack of credits. In determining the optimal credit advertisement, one must consider the following components which are additive:

1. The fact that the flow control latency timer starts when the last symbol of a TLP is received. Therefore one maximum sized packet from the local device must be stored in the receive buffer to accumulate received TLP data.
2. The fact that a flow control update DLLP is not scheduled by the local device until its flow control latency timer has expired. During the time interval between start and expiry of the flow control latency timer, the receive buffer must have additional space to accumulate received TLP data.
3. The delays caused by the:
   a. Transmission of a TLP by remote device.
   b. Processing of TLP by local device.
   c. Transmission of FC update by local device.
   d. Receiving FC update by remote device.
   e. Round trip latency of Re-timer device (if present). This latency is calculated from `CX_MAX_RETIMER` value.

   As an example, the equations to derive posted data credits are as follows where the label in each {} represents the contribution to each factor detailed in this section:

```
depth =(calc_bytes)/(4*CX_NW)/4
calc_bytes =roundUptoNextMTU(data_size{1} + fc_lat_limit{2} + fc_delay_bytes)
fc_lat_limit =data_size *fc_upd_fac
fc_delay_bytes =CX_INTERNAL_DELAY{3.b and 3.c}*CX_NB*CX_NL+ base_internal_delay{3.a and 3.d}*CX_NL + retimer_latency{3.e}*CX_NL
```

   where

- `data_size` is the `CX_MAX_MTU` + 28 that is specified in the *PCI Express Base Specification, Revision 6.0, Version 1.0* as TLP Overhead.
- `fc_upd_fac` comes directly from Tables 2-39, 2-40, and 2-41 of the *PCI Express Base Specification, Revision 6.0, Version 1.0.*
- `CX_NB` is the number of symbols processed at the PIPE interface per lane. For example, the value is two symbols (16-bit PIPE) for a Gen2 controller running at 250 MHz. For more information, see "Frequency, Speed, and Width Support" on page 36.
- `CX_NL` is the number of lanes of the device.
- `CX_NW` is the internal datapath width in DWORDs.
- `base_internal_delay` is the sum of the MAC (controller) and PHY delays (Tx and Rx) in the remote device. To model this, Synopsys uses the *PCI Express Base Specification, Revision 6.0, Version 1.0* recommendation of a constant value of 19 symbol times for 2.5 GT/s mode operation, 70 symbol times for 5.0 GT/s mode operation, and 115 symbol times for 8.0 GT/s and 16 GT/s mode operation.
- `CX_INTERNAL_DELAY` is the sum of the controller and PHY delays (Tx and Rx) of the local device. This also includes:
  - delays in the local device between receiving the EOT and starting the flow control latency timer.

  - delays associated from the time the flow control latency timer expires until the controller receives the flow control packet.
- The number of bytes calculated is rounded up to next MTU so that the device advertises at least another full MTU worth of bytes/credits. This helps to reduce credit starvation.

4. Default Non-Posted data credits (`RADM_NPQ_DCRD_VC0`) should be a fraction of the Non-Posted header credits because Non-Posted TLPs with data, that is, I/O and Configuration Write Transactions, occur infrequently (and Atomic Ops occur slightly more frequently).

- `RADM_NPQ_DCRD_VC0` is set to be ½ `RADM_NPQ_HCRD_VC0` if Atomic Ops are enabled, or ¼ `RADM_NPQ_HCRD_VC0` otherwise.
- Additionally enabled VCs in configurations where AtomicOps are not enabled, do not have I/O or Configuration Write Transactions, and thus require no data credits (`RADM_NPQ_DCRD_VC(1-7)`).

It is possible that these equations result in a number of credits (and a resulting receive queue depth) which is too large for your requirements, or you choose to trade off some temporal drop in performance versus back pressure on the remote transmitter for receive buffer depth. You can always advertise more than the default number of credits if you want to continue to offload the wire in the presence of application halting.

Taking the case of Non-Posted header credits (`RADM_NPQ_HCRD_VC(0-7)`), a conservative approach is used for the default calculation, and the total amount that can be transmitted during a Flow Control update window is selected to ensure saturation of the datapath. This however does not take into consideration the number of Non-Posted requests the application can handle; which is dependent on impacting factors such as the application response time, the link latency, the credit return latency, and the number of available Tags.

A more efficient approach can be taken by connecting the amount of Non-Posted header credits to the number of outstanding requests supported, along with an additional margin to account for packets currently in flight.

A general rule of thumb is:

- If the Target Completion Lookup Table (`TRGT_CPL_LUT_EN`) is full, the controller must have returned the Non-Posted credits associated with the transactions in the LUT already.
  - Therefore, in a full LUT situation there are already `CX_REMOTE_MAX_TAG` Non-Posted requests in the queues waiting to be offloaded.
- If the LUT is not full it implies that Non-Posted requests are being completed at a rate sufficient to prevent LUT full.
  - In this case it is only necessary to ensure that there are sufficient credits in flight to prevent starvation.
- Ideally the LUT should run close to full.
  - Advertising approximately the same number of credits as concurrent requests that can be handled is a good balance, allowing a few extra credits to account for delays across the link.
- Applications which do not use the Target Completion Lookup Table are likely to have a similar concept to `CX_REMOTE_MAX_TAG.`

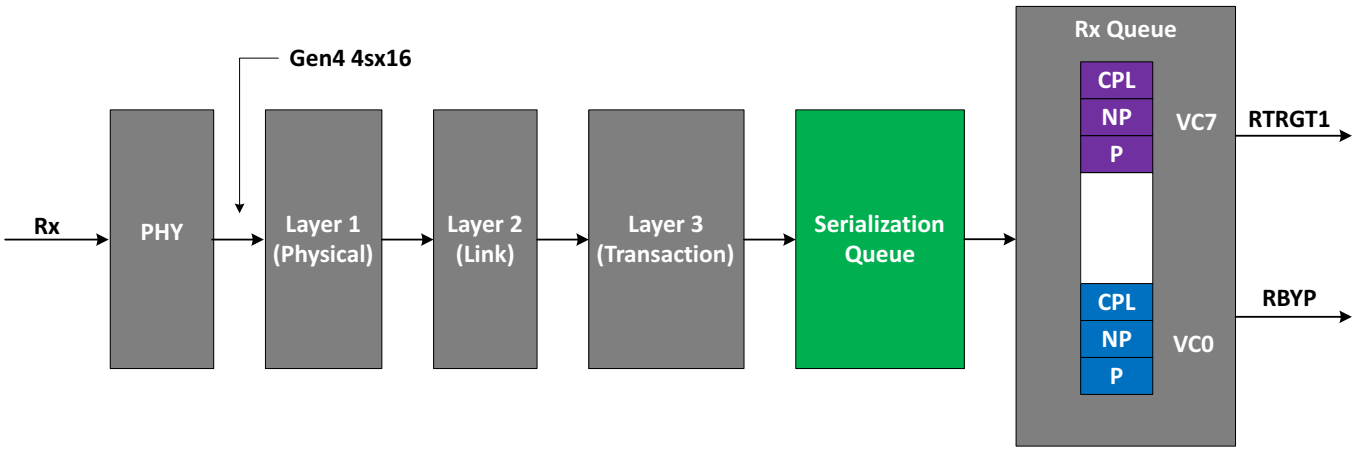
For example, it can be shown that for the case of 256-byte requests with a 1us round trip time, 32 tags are sufficient to keep an X4 link saturated. Adding additional tags or credits only increases RAM sizes and area without providing any further performance boost.

S

Advanced Information: Serialization Queue Almost Full Threshold

The TLP Serialization Queue resides between Layer3 and the Virtual Channel TLP Receive Queues as shown in Figure 1. Figure S-1.

Figure S-1 TLP Serialization Queue



The TLP Serialization Queue is used to consume one or more TLPs from the Transaction Layer (Layer3) in a single cycle and to send TLPs in series to the application (RTRGT1, RBYP), through the VC TLP Receive Queues. The number of TLPs to process in a single cycle depends on the TLP length. For example, for a stream of TLPs with 3 DWORDs of header the maximum number of TLPs to process in a single cycle is 4.

Depending on the mix of TLP length's the amount of data stored in the Serialization Queue can increase or decrease over time. For more information on how the data stored in the queue increases and decrease depending on the TLP length, see "Data Rates" on page 765.

Over time TLPs can accumulate and cause the Serialization Queue to overflow, unless its size is set to the VC TLP Receive Queue size. However, allowing the Serialization Queue to buffer up this amount of data has no benefit in terms of TLP throughput at the application interfaces (RTRGT1/RBYP). The application

can offload only one TLP at a time and therefore in the presence of short TLPs cannot keep pace with the maximum throughput on the wire.

To minimize the RAM footprint, without any performance penalty, you can significantly reduce the Serialization Queue size and maintain throughput at the application interface.

To prevent the reduced Serialization Queue from overflowing the controller takes advantage of the Data Link layers ACK/NACK protocol. At a certain threshold or water mark, named `ALMOST_FULL`, the Serialization Queue masks incoming TLPs and schedules the transmission of a NACK. The NACK sequence number indicates the last good TLP received. The remote link partner purges all acknowledged TLPs from its replay buffer and replays all other TLPs stored in the buffer in order. While the incoming TLPs are being masked the Serialization Queue continues to send TLPs to the VC TLP Receive Queues. Note: halting the RTRGT1 interface has no impact on the transfer of these TLPs. When the amount of data stored in the queue drops below `ALMOST_FULL` the Serialization Queue unmasks incoming TLPs enabling the replayed TLPs to be stored in the queue.

The following topics are discussed:

- "Serialization Queue Almost Full Threshold"
- "Serialization Queue Input and Output Data Rates" on page 773

S.1 Serialization Queue Almost Full Threshold

The almost full threshold (`ALMOST_FULL`) is determined by:

- The maximum output rate possible at the output of the serialization queue, and
- The round trip delay starting from the request to generate the NACK to the reception of the first replayed TLP.

`ALMOST_FULL` equates to the number of QDWORDs required to sustain the maximum TLP output rate for a given NACK request to first replayed TLP round trip latency. During this period TLPs are not written to the queue. The accumulated TLPs present in the queue are assumed to be read at the maximum output rate. To avoid a loss in performance the queue must not underflow.

S.1.1 Data Rates

The Transaction Layer (Layer3) receives TLPs at a rate of 16 DWORDs per cycle. These DWORDs are extracted, packed into QDWORDs and sent to the Serialization Queue. The maximum number of QDWORDs that can be sent in a single cycle is 7. The Serialization Queue sends TLPs to the application at a rate of one TLP per cycle.

Figure S-2 and Figure S-3 describe how streams of TLPs, with and without prefixes, accumulate in the Serialization Queue. The fastest accumulation rate without TLP Prefixes occurs for a TLP stream of 3 Header DWORDs and 1 Data DWORD. The input data rate is 5.33 QDWORDs/cycle; the output data rate is 2 QDWORDs/cycle. The fastest accumulation rate with TLP Prefixes occurs for a TLP stream of 1 Prefix DWORD, 3 Header DWORDs and 1 Data DWORD. The input data rate is 6.86 QDWORDs/cycle; the output data rate is 3 QDWORDs/cycle.

Figure S-2 Serialization Queue Accumulation Rate = 3.33 QDWORDs/cycle (No Prefix)

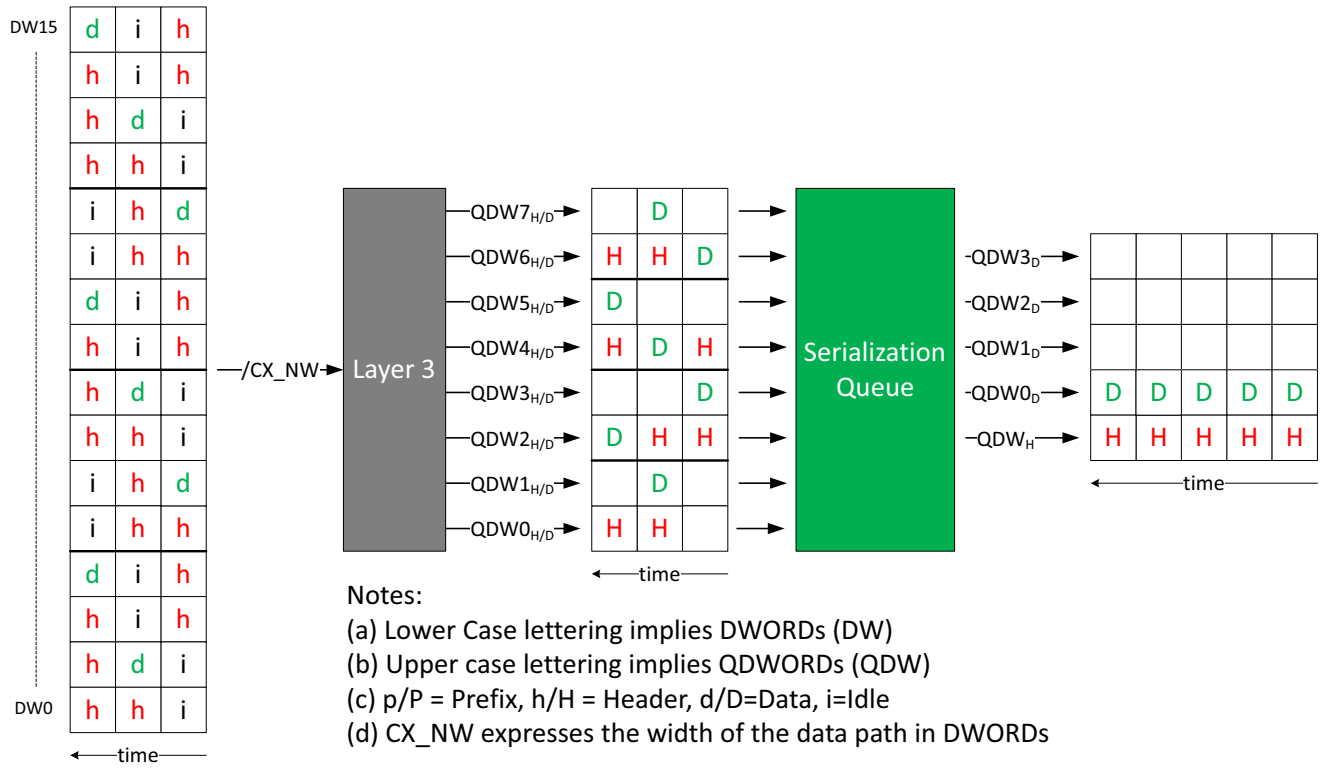


Figure S-3 Serialization Queue Accumulation Rate = ~3.86 QDWORDs/cycle (1 Prefix)

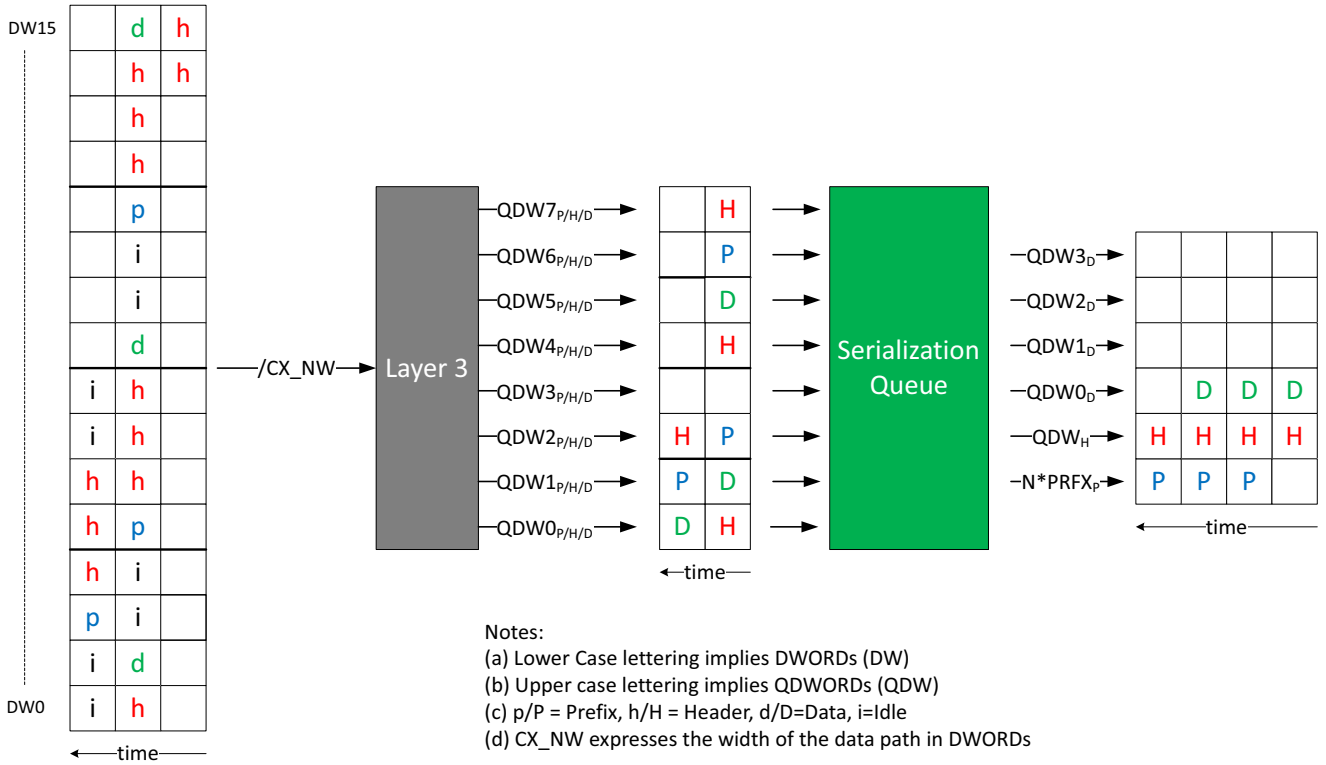
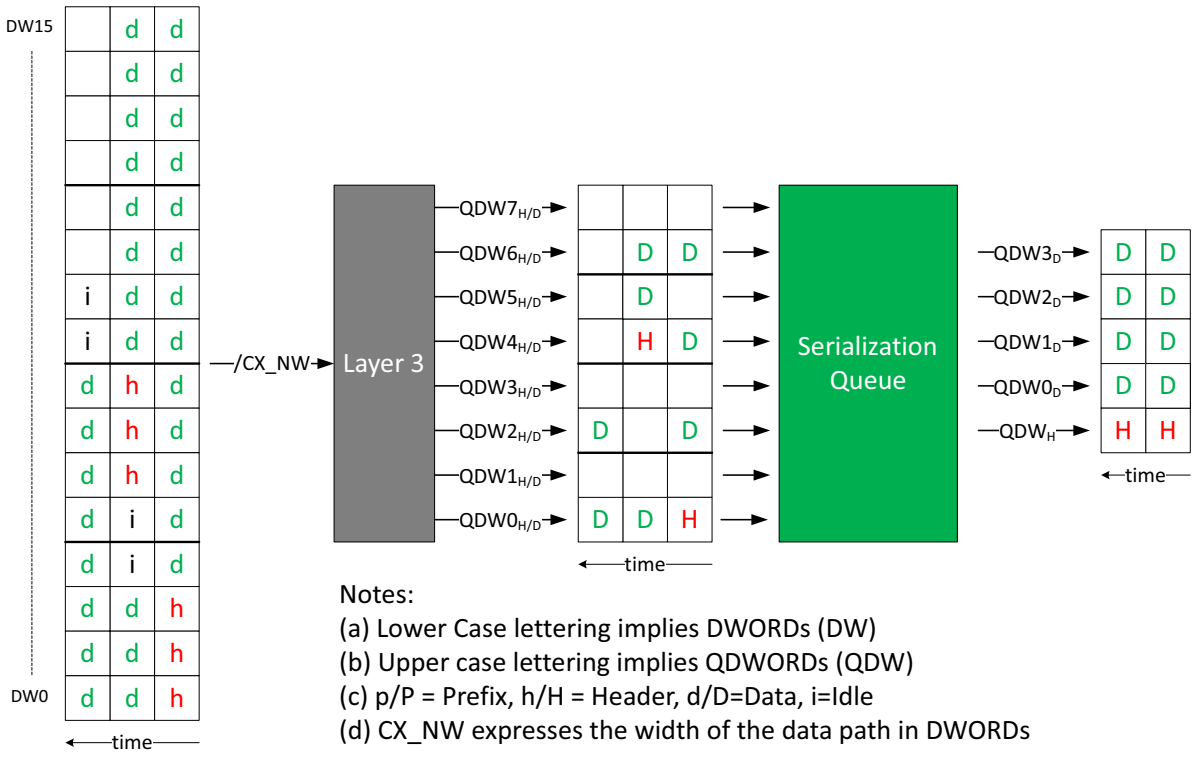


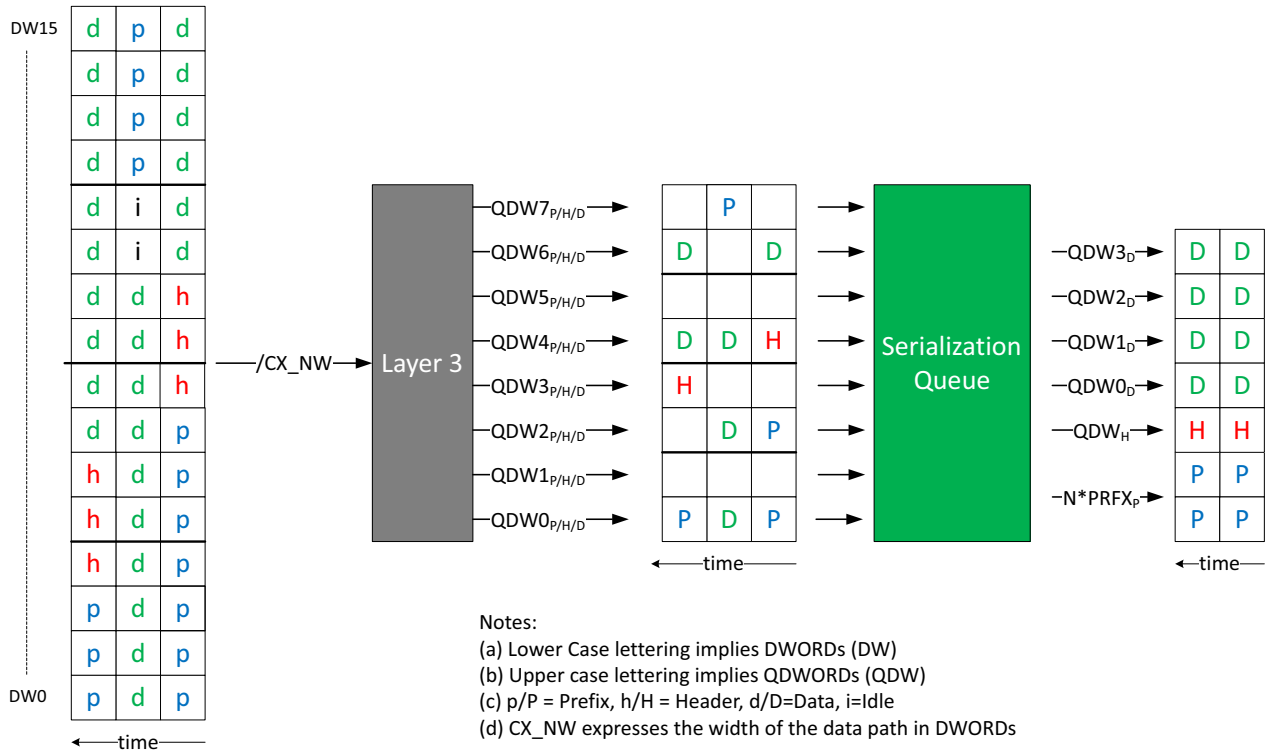
Figure S-4 and Figure S-5 describe how streams of TLPs, with and without prefixes, dissipate in the Serialization Queue. The fastest TLP dissipation rate without prefixes occurs for a stream TLPs consisting of 3 Header DWORDs and 16 Data DWORDs. The input data rate is 3.81 QDWORDs/cycle; the output data rate is 5 QDWORDs/cycle. The fastest TLP dissipation rate with TLP Prefixes occurs for a TLP stream of 7 Prefix DWORDs, 3 Header DWORDs and 16 Data DWORDs. The input data rate is 4 QDWORDs/cycle; the output data rate is 7 QDWORDs/cycle.

Figure S-4 Serialization Queue Dissipation Rate = 1.19 QDWORDs/cycle (No Prefix)



Notes:
(a) Lower Case lettering implies DWORDs (DW)
(b) Upper case lettering implies QDWORDs (QDW)
(c) p/P = Prefix, h/H = Header, d/D=Data, i=Idle
(d) CX_NW expresses the width of the data path in DWORDs

Figure S-5 Serialization Queue Dissipation Rate = 3 QDWORDs/cycle (7 Prefixes)



Notes:
(a) Lower Case lettering implies DWORDs (DW)
(b) Upper case lettering implies QDWORDs (QDW)
(c) p/P = Prefix, h/H = Header, d/D=Data, i=Idle
(d) CX_NW expresses the width of the data path in DWORDs

The QDWORD input and output rate is determined by the following equations:

```
Input Rate (QDWORDs per cycle) = (Number of TLP per cycle)*(Number of QDWORDs per
TLP)= NW/(PDW + HDW + DDW + TLP Overhead)*(PQDW + HQDW + DQDW)
Output Rate (QDWORDs per cycle) = (Tota Number of TLP QWORDs)/(Number of cycles to
offload the TLP data QDWORDs)= (PQDW + HQDW + DQDW)/ ROUNDUP(DQDW/(NW/4))
```

Where:

- `NW` is the width of the data path in DWORDs
- P_{DW} is the TLP prefix length in DWORDs
- H_{DW} is the TLP header length in DWORDs
- D_{DW} is the TLP data length in DWORDs
- P_{QDW} is the TLP prefix length in QDWORDs
- H_{QDW} is the TLP header length in QDWORDs
- D_{QDW} is the TLP data length in QDWORDs, and
- `TLP Overhead` is the TLP overhead stripped by the lower layers (Layer1 tokens and Layer2 LCRC).

For example, a TLP stream of 1 prefix DWORD, 3 header DWORDs, and 1 data DWORD results in the following input and output QDWORD rates:

```
Input Rate
= 16/(1+3+1+2)*(1+1+1)
= 6.86 QDWORDs/cycle
Output Rate
= (1+1+1)/1
= 3 QDWORDs/cycle
```

A TLP stream consisting of 3 header DWORDs and 16 data DWORDs results in the following input and output QDWORD rates:

```
Input Rate= 16/(3+16+2)*(1+4)= 3.81 QDWORDs/cycle
Output Rate= (1+4)/1= 5 QDWORDs/cycle
```

The increase or decrease in the number of QDWORDs stored in the queue can be expressed by the following equations:

```
Constant = (PDW + HDW + DDW + 2)% CX_NW =0Increase = (PDW + HDW + DDW + 2)% CX_NW >0
and <=(PDW + HDW + 2) Decrease = (PDW + HDW + DDW + 2)% CX_NW >(PDW + HDW + 2) or (DDW =0)
```

For a table of input and output data rates for various TLP lengths see "Serialization Queue Input and Output Data Rates" on page 773.

The `ALMOST_FULL` threshold is reached for any given stream of TLPs such that over time the input rate is greater than the output rate.

Note: These formulas are based on a stream of fixed length TLPs.

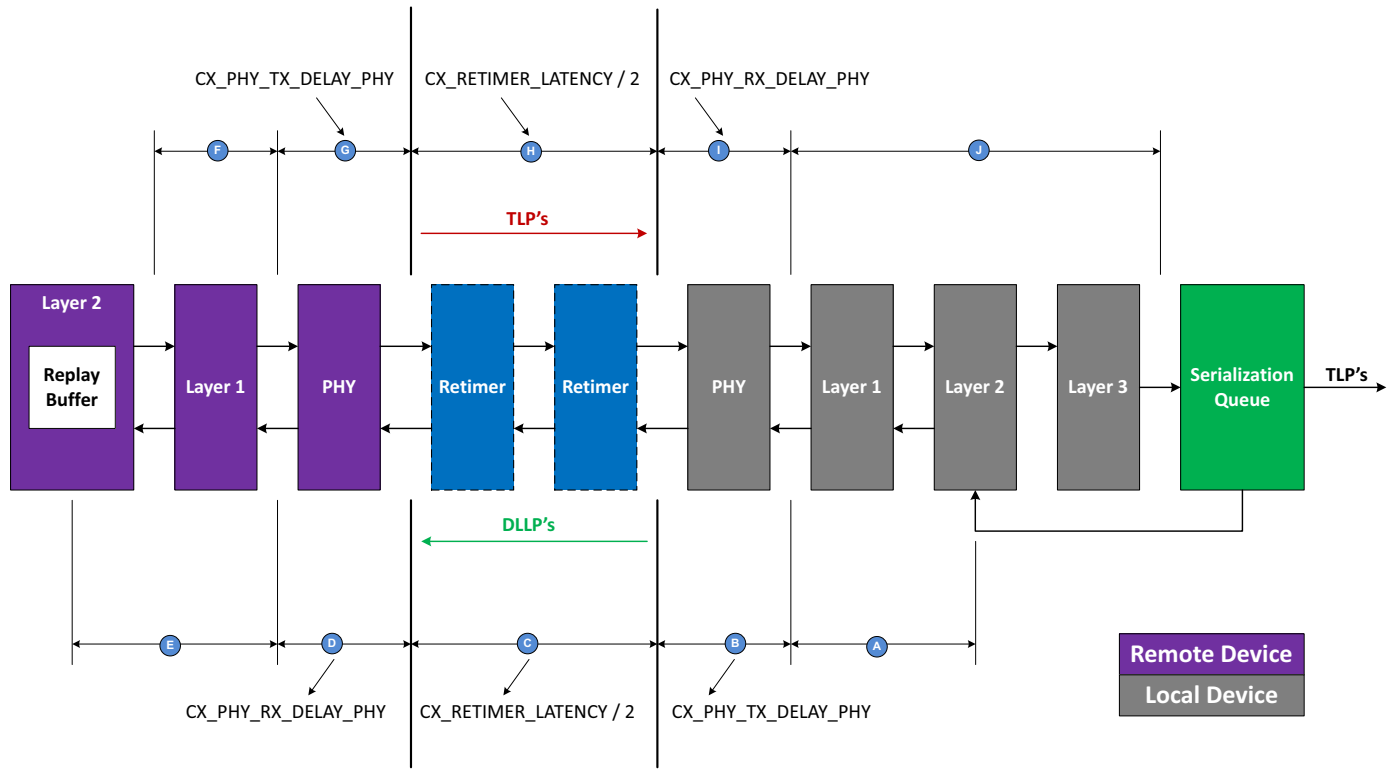
S.1.2 NACK Request to Replayed TLP Latency

The Serialization Queue output data rate must be sustained from the time the NACK (high priority DLLP) is requested to the first replayed TLP at the input the Serialization Queue. The NACK is requested when `ALMOST_FULL` is reached.

The round trip time from the NACK request to the first replayed TLP at the input to the Serialization Queue depends on the following latencies:

1. The Local Layer2 NACK request to PIPE latency
2. The Local PHY TX latency
3. The Link Retimer latency, Local to Remote
4. The Remote PHY Rx latency
5. The Remote NACK DLLP to replay request latency
6. The Remote replay request to PIPE latency
7. The Remote PHY TX latency
8. The Link Retimer latency, Remote to Local
9. The Local PHY Rx latency
10. The Local PIPE to Serialization Queue latency

Figure S-6 NACK Request to First Replayed TLP at Input to Serialization Queue



The round trip time from transmission of the Nack to the first replayed TLP at the input to the serialization queue depends on the following latencies:

- A **Local Layer2 NACK request to PIPE latency:** This is known by design and is set to 9 core clock cycles.
- B **The Local PHY TX latency (`CX_PHY_TX_DELAY_PHY`):** This is the worst case delay from the PIPE interface to the TX PHY serial pins, expressed in PIPE clock cycles. It is specified by your PHY provider and passed at build time through coreConsultant.
- C **Local to Remote Link Retimer latency (`CX_RETIMER_LATENCY/2`):** This is the worst case delay through the Retimers, expressed in symbols times. Its value defaults to the maximum delay specified in the PCIe Base Specification.
- D **Remote PHY Rx latency:** This is the worst case delay from the Rx PHY serial pins to the PIPE interface, expressed in PIPE clock cycles. It is matched to the Local PHY Rx latency, `CX_PHY_RX_DELAY_PHY`.
- E **Remote NACK DLLP to replay request latency:** This is extrapolated from the Local NACK DLLP to replay request latency. It is known by design and has a value of 10 core clock cycles.
- F **Remote replay request to PIPE latency:** This is extrapolated from the Local Remote replay request to PIPE latency. It is known by design and has a maximum value of 28 core clock cycles.
- G **Remote PHY TX latency:** This is the worst case delay from the PIPE interface to the TX PHY serial pins, expressed in PIPE clock cycles. It is matched to the Local PHY TX latency, `CX_PHY_TX_DELAY_PHY`.
- H **Remote to Local Link Retimer latency (`CX_RETIMER_LATENCY/2`):** This is the worst case delay through the Retimers, expressed in symbols times. Its value defaults to the maximum delay specified in the PCIe Base Specification.
- I **Local PHY Rx latency (`CX_PHY_RX_DELAY_PHY`):** This is the worst case delay from the Rx PHY serial pins to the PIPE interface, expressed in PIPE clock cycles. It is specified by your PHY provider and passed at build time through coreConsultant.
- J **Local PIPE to Serialization Queue latency:** This is known by design and has maximum value of 16 core clock cycles.

In summary, the latency from the transmission of the NACK, to the first replayed TLP at the input, to the serialization queue in controller clock cycles is:

```
NACK_TO_RPLYD_TLP =
(2 *(Core Clock to PIPE Clock Frequency Ratio
```

```
*(CX_PHY_TX_DELAY_PHY + CX_PHY_RX_DELAY_PHY)))
+(Symbol Time * CX_RETIMER_LATENCY/Core Clock Period) + 68
```

Note: 2s (16-bit) and 4s (32-bit) Controller-PHY Dynamic Frequency combinations are supported. For 2s the Controller Clock to PIPE Clock frequency ratio is 1:2, for 4s it is 1:1.

S.1.3 Almost Full Threshold Calculation

`ALMOST_FULL` is calculated for the worst case scenario where the threshold is reached and the TLPs stored in queue are read at the maximum QDWORD output rate.

The queue size, expressed in QDWORDs, is equal to the latency from the NACK request to the first replayed TLP at the input the Serialization Queue times the maximum QDWORD output rate.

```
ALMOST_FULL(QWORDS) = NACK_TO_RPLYD_TLP * Maximum QDWORD Output Rate
```

You can configure the PHY transmit latency, `CX_PHY_TX_DELAY_PHY`, the PHY receive latency, `CX_PHY_RX_DELAY_PHY`, and the round trip Retimer latency, `CX_RETIMER_LATENCY` through coreConsultant. For example, if the Controller-PHY PIPE IO width is 4 Bytes, that is, Controller clock =PIPE Clock, then setting `CX_PHY_TX_DELAY_PHY` to 12, `CX_PHY_RX_DELAY_PHY` to 28, and `CX_RETIMER_LATENCY` to 528 Symbol Times (Gen4 speed, SRIS enabled, MaxPayloadSize =256Bytes, 2 retimers) gives:

```
ALMOST_FULL(QWORDS) = 280 * Maximum QDWORD Output Rate
```

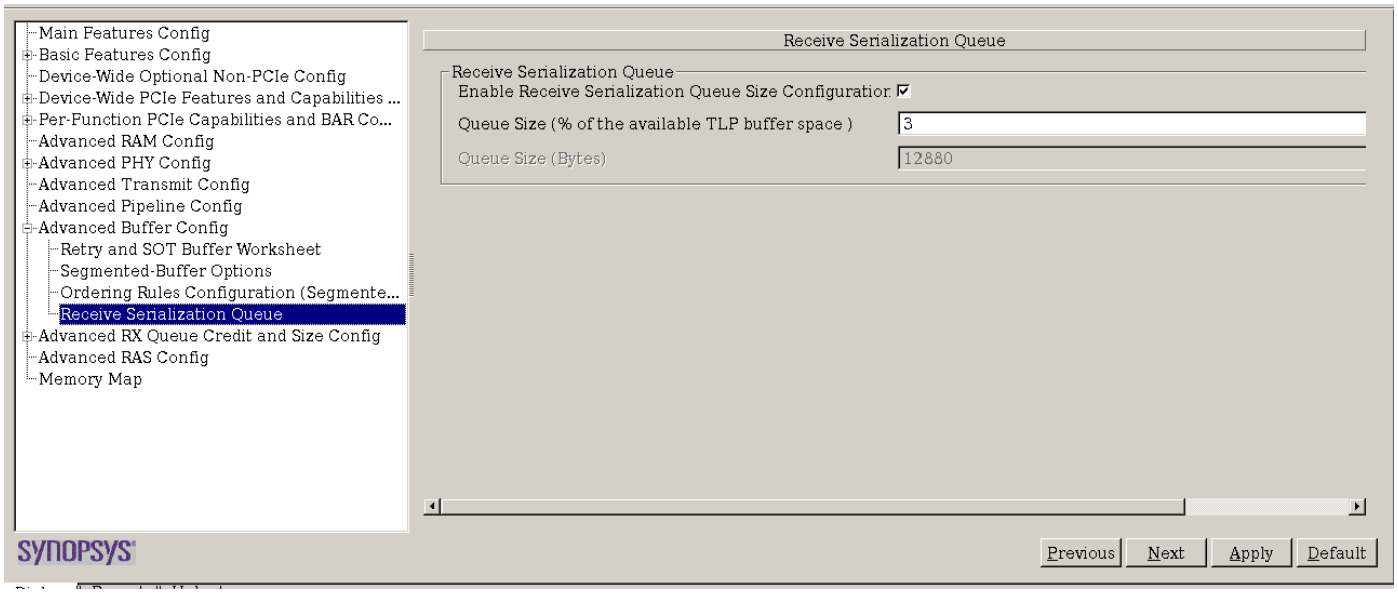
The maximum output rate depends on the number of TLP prefixes supported. Without TLP prefixes the maximum output rate is 5 QDWORDs/cycle. If TLP prefixes are supported, and the number of prefixes range from 1 to 4, then the maximum output rate is 6 QDWORDs/cycle; if the number of prefixes range from 5 to 7, then the maximum output rate is 7 QDWORDs/cycle. Table S-1 summarizes the auto calculated almost full threshold depending on the number of supported TLP prefixes.

Table S-1 Auto-calculated Almost Full Threshold

Number of Prefixes NPRFX	Maximum Output Rate (QDWORDs/cycle)	ALMOST_FULL (QDWORDs)	ALMOST_FULL (Bytes)
0	5	1400	22400
1 to 4	6	1680	26880
5 to 7	7	1960	31360

You can adjust the auto-calculated almost full threshold through the coreConsultant "Receive Serialization Queue" configuration page as shown in Figure S-7.

Figure S-7 Receive Serialization Queue Configuration Page in coreConsultant



`ALMOST_FULL` or Serialization Queue size is presented as a percentage of the available TLP buffer space, that is, as a percentage of the configured TLP Receive Queue size. The auto-calculated value is the optimal threshold to ensure TLPs are sent to the application without impacting performance, and is the default value. To increase `ALMOST_FULL`, you must tick the "Enable Receive Serialization Size Configuration" tick box prior to modifying its value.

Note: If the wire data rate is greater than the Serialization Queue output data rate then increasing the almost full threshold only serves to extend the time before the NACK is sent. If the wire data rate is not sustained over time, then increasing the almost full threshold can reduce the transmission frequency of the NACKs. For more information on the number of fixed length TLPs required to reach `ALMOST_FULL`, for a given NACK request to replayed TLP latency, see "Serialization Queue Input and Output Data Rates" on page 773.

Note: To prevent the controller from sending overflow prevention related NACKs you must set `ALMOST_FULL` to 100% of the configured TLP Receive Queue size.

S.2 Serialization Queue Input and Output Data Rates

Figure S-8 shows the input and output QDWORD data rates for streams of TLPs of various DWORD lengths (no TLP prefix). For simplicity DLLPs are not considered in the data stream. DLLPs reduce the TLP input rate, resulting in a small increase in the number of back-to-back TLPs required to reach ALMOST_FULL.

- P_{DW} is the TLP prefix length expressed in DWORDs
- H_{DW} is the TLP header length expressed in DWORDs
- $D_{DW}$is the TLP data length expressed in DWORDs
- P_{QDW} is the TLP prefix length expressed in QDWORDs
- H_{QDW} is the TLP header length expressed in QDWORDs
- D_{QDW} is the TLP data length expressed in QDWORDs
- l_i is the TLP input data rate expressed in QDWORDs/cycle
- l_o is the TLP output data rate expressed in QDWORDs/cycle
- l_i - l_o tracks the TLP accumulation/dissipation rate in QDWORDs/cycle

Note: In Figure S-8, Figure S-9, and Figure S-10, Num of TLPs to reach ALMOST_FULL column is calculated based on the PHY transmit and receive latencies specified in "Almost Full Threshold Calculation" on page 771, that is, `CX_PHY_TX_DELAY_PHY` =12 PIPE clock cycles, `CX_PHY_RX_DELAY_PHY` =28 PIPE clock cycles. In addition, the PCIe link includes 2 retimers.The data is shown is for information purposes only.

Figure S-8 Serialization Queue Input and Output Data Rates (No Prefixes)

P_{DW}	H_{DW}	D_{DW}	P_{qDW}	H_{qDW}	D_{qDW}	Num. TLPs / cyc	λ_ι qDW / cyc	λ_o qDW / cyc	$\lambda\iota - \lambda o$ qDW / cyc	Num cycles to reach ALMOST_FULL	Num TLPs to reach ALMOST_FULL
0	3	0	0	1	0	3.2	3.20	1.00	2.20	628	2009
0	3	1	0	1	1	2.7	5.33	2.00	3.33	414	1117
0	3	2	0	1	1	2.3	4.57	2.00	2.57	537	1235
0	3	3	0	1	1	2	4.00	2.00	2.00	690	1380
0	3	4	0	1	1	1.8	3.56	2.00	1.56	888	1598
0	3	5	0	1	2	1.6	4.80	3.00	1.80	767	1227
0	3	6	0	1	2	1.5	4.36	3.00	1.36	1012	1518
0	3	7	0	1	2	1.4	4.00	3.00	1.00	1380	1932
0	3	8	0	1	2	1.3	3.69	3.00	0.69	1994	2592
0	3	9	0	1	3	1.2	4.57	4.00	0.57	2415	2898
0	3	10	0	1	3	1.1	4.27	4.00	0.27	5175	5692
0	3	11	0	1	3	1	4.00	4.00	0.00	-	-
0	3	12	0	1	3	1	3.76	4.00	-0.24	-	-
0	3	13	0	1	4	0.9	4.44	5.00	-0.56	-	-
0	3	14	0	1	4	0.9	4.21	5.00	-0.79	-	-
0	3	15	0	1	4	0.8	4.00	5.00	-1.00	-	-
0	3	16	0	1	4	0.8	3.81	5.00	-1.19	-	-
0	3	17	0	1	5	0.8	4.36	3.00	1.36	1012	809
0	3	18	0	1	5	0.7	4.17	3.00	1.17	1176	823
0	3	19	0	1	5	0.7	4.00	3.00	1.00	1380	966
0	3	20	0	1	5	0.7	3.84	3.00	0.84	1643	1150
0	3	21	0	1	6	0.7	4.31	3.50	0.81	1709	1196
0	3	22	0	1	6	0.6	4.15	3.50	0.65	2130	1278
0	3	23	0	1	6	0.6	4.00	3.50	0.50	2760	1656
0	3	24	0	1	6	0.6	3.86	3.50	0.36	3812	2287
0	3	25	0	1	7	0.6	4.27	4.00	0.27	5175	3105
0	3	26	0	1	7	0.6	4.13	4.00	0.13	10695	6417
0	3	27	0	1	7	0.5	4.00	4.00	0.00	-	-
0	3	28	0	1	7	0.5	3.88	4.00	-0.12	-	-
0	3	29	0	1	8	0.5	4.24	4.50	-0.26	-	-
0	3	30	0	1	8	0.5	4.11	4.50	-0.39	-	-
0	3	31	0	1	8	0.5	4.00	4.50	-0.50	-	-

Figure S-9 shows the input and output QDWORD data rates for streams of TLPs of various DWORD lengths, with 1 prefix supported.

Figure S-9 Serialization Queue Input and Output Data Rates (1 Prefix)

P_{DW}	H_{DW}	D_{DW}	P_{qDW}	H_{qDW}	D_{qDW}	Num. TLPs / cyc	λ_i qDW / cyc	λ_o qDW / cyc	$\lambda_i - \lambda_o$ qDW / cyc	Num cycles to reach ALMOST_FULL	Num TLPs to reach ALMOST_FULL
1	3	0	1	1	0	2.7	5.33	2.00	3.33	414	1117
1	3	1	1	1	1	2.3	6.86	3.00	3.86	358	823
1	3	2	1	1	1	2	6.00	3.00	3.00	460	920
1	3	3	1	1	1	1.8	5.33	3.00	2.33	592	1065
1	3	4	1	1	1	1.6	4.80	3.00	1.80	767	1227
1	3	5	1	1	2	1.5	5.82	4.00	1.82	759	1138
1	3	6	1	1	2	1.4	5.33	4.00	1.33	1035	1449
1	3	7	1	1	2	1.3	4.92	4.00	0.92	1495	1943
1	3	8	1	1	2	1.2	4.57	4.00	0.57	2415	2898
1	3	9	1	1	3	1.1	5.33	5.00	0.33	4140	4554
1	3	10	1	1	3	1	5.00	5.00	0.00	-	-
1	3	11	1	1	3	1	4.71	5.00	-0.29	-	-
1	3	12	1	1	3	0.9	4.44	5.00	-0.56	-	-
1	3	13	1	1	4	0.9	5.05	6.00	-0.95	-	-
1	3	14	1	1	4	0.8	4.80	6.00	-1.20	-	-
1	3	15	1	1	4	0.8	4.57	6.00	-1.43	-	-
1	3	16	1	1	4	0.8	4.36	6.00	-1.64	-	-
1	3	17	1	1	5	0.7	4.87	3.50	1.37	1008	705
1	3	18	1	1	5	0.7	4.67	3.50	1.17	1183	828
1	3	19	1	1	5	0.7	4.48	3.50	0.98	1409	986
1	3	20	1	1	5	0.7	4.31	3.50	0.81	1709	1196
1	3	21	1	1	6	0.6	4.74	4.00	0.74	1863	1117
1	3	22	1	1	6	0.6	4.57	4.00	0.57	2415	1449
1	3	23	1	1	6	0.6	4.41	4.00	0.41	3335	2001
1	3	24	1	1	6	0.6	4.27	4.00	0.27	5175	3105
1	3	25	1	1	7	0.6	4.65	4.50	0.15	9507	5704
1	3	26	1	1	7	0.5	4.50	4.50	0.00	-	-
1	3	27	1	1	7	0.5	4.36	4.50	-0.14	-	-
1	3	28	1	1	7	0.5	4.24	4.50	-0.26	-	-
1	3	29	1	1	8	0.5	4.57	5.00	-0.43	-	-
1	3	30	1	1	8	0.5	4.44	5.00	-0.56	-	-
1	3	31	1	1	8	0.5	4.32	5.00	-0.68	-	-

Figure S-10 shows the input and output QDWORD data rates for streams of TLPs of various DWORD lengths, with 7 prefixes supported.

Figure S-10 Serialization Queue Input and Output Data Rates (7 Prefixes)

P_{DW}	H_{DW}	D_{DW}	P_{qDW}	H_{qDW}	D_{qDW}	Num. TLPs / cyc	λ_i qDW / cyc	λ_o qDW / cyc	$\lambda_i - \lambda_o$ qDW / cyc	Num cycles to reach ALMOST_FULL	Num TLPs to reach ALMOST_FULL
7	3	0	2	1	0	1.4	4.00	3.00	1.00	1380	1932
7	3	1	2	1	1	1.3	4.92	4.00	0.92	1495	1943
7	3	2	2	1	1	1.2	4.57	4.00	0.57	2415	2898
7	3	3	2	1	1	1.1	4.27	4.00	0.27	5175	5692
7	3	4	2	1	1	1	4.00	4.00	0.00	-	-
7	3	5	2	1	2	1	4.71	5.00	-0.29	-	-
7	3	6	2	1	2	0.9	4.44	5.00	-0.56	-	-
7	3	7	2	1	2	0.9	4.21	5.00	-0.79	-	-
7	3	8	2	1	2	0.8	4.00	5.00	-1.00	-	-
7	3	9	2	1	3	0.8	4.57	6.00	-1.43	-	-
7	3	10	2	1	3	0.8	4.36	6.00	-1.64	-	-
7	3	11	2	1	3	0.7	4.17	6.00	-1.83	-	-
7	3	12	2	1	3	0.7	4.00	6.00	-2.00	-	-
7	3	13	2	1	4	0.7	4.48	7.00	-2.52	-	-
7	3	14	2	1	4	0.7	4.31	7.00	-2.69	-	-
7	3	15	2	1	4	0.6	4.15	7.00	-2.85	-	-
7	3	16	2	1	4	0.6	4.00	7.00	-3.00	-	-
7	3	17	2	1	5	0.6	4.41	4.00	0.41	3335	2001
7	3	18	2	1	5	0.6	4.27	4.00	0.27	5175	3105
7	3	19	2	1	5	0.6	4.13	4.00	0.13	10695	6417
7	3	20	2	1	5	0.5	4.00	4.00	0.00	-	-
7	3	21	2	1	6	0.5	4.36	4.50	-0.14	-	-
7	3	22	2	1	6	0.5	4.24	4.50	-0.26	-	-
7	3	23	2	1	6	0.5	4.11	4.50	-0.39	-	-
7	3	24	2	1	6	0.5	4.00	4.50	-0.50	-	-
7	3	25	2	1	7	0.5	4.32	5.00	-0.68	-	-
7	3	26	2	1	7	0.5	4.21	5.00	-0.79	-	-
7	3	27	2	1	7	0.5	4.10	5.00	-0.90	-	-
7	3	28	2	1	7	0.4	4.00	5.00	-1.00	-	-
7	3	29	2	1	8	0.4	4.29	5.50	-1.21	-	-
7	3	30	2	1	8	0.4	4.19	5.50	-1.31	-	-
7	3	31	2	1	8	0.4	4.09	5.50	-1.41	-	-