# Virtex-6 FPGA Integrated Block for PCI Express

## *User Guide*

**UG517 (v5.1) September 21, 2010**

**XILINX**®

Xilinx is providing this product documentation, hereinafter "Information," to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

# Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 06/24/09 | 2.0 | Initial Xilinx release. |
| 09/16/09 | 3.0 | Updated core to v1.3 and ISE to v11.3. Updated to add Root Port capability. |
| 12/02/09 | 4.0 | Updated core to v1.4 and ISE to v11.4. Updated to add VHDL support. |
| 04/19/10 | 5.0 | Updated core to v1.5 and ISE to v12.1. Revised references to Gen1 and Gen2 to 2.5 Gb/s and 5.0 Gb/s link speeds. Removed requirement about operation at 5.0 Gb/s data rates needing a 250 MHz clock source from Table 2-5, Clocking, page 184, and System and PCI Express Interfaces, page 263. Removed notes or text about never asserting trn_rbar_hit_n[6:0] in Root Port configuration in Table 2-13, page 37, in Basic TLP Receive Operation, page 107, and in Packet Base Address Register Hit on Receive Transaction Interface, page 115. Changed "Extended Capability Structure" to "Capability Structure" in the descriptions of cfg_dstatus[15:0], cfg_dcommand[15:0], cfg_dcommand2[15:0], cfg_lstatus[15:0], and cfg_lcommand[15:0] in Table 2-17, page 42, Table 6-14, page 155, cfg_dstatus[15:0], page 157, cfg_dcommand[15:0], page 157, cfg_lstatus[15:0], page 158, cfg_lcommand[15:0], page 158, and cfg_dcommand2[15:0], page 159.<br><br>Chapter 2: In Table 2-2, changed address range 048h through 05Ch to optional, tagged address range 074h through 080h as Root Port only and divided into four rows, and added table notes 1 and 2. In Table 2-10, updated the description of trn_clk. In Table 2-14, added parenthetical text to the 19 encoding of pl_ltssm_state[5:0] on page 39. In Table 2-17, changed the cfg_byte_en_n[3:0] description to "write access" on page 42. In Table 2-20, removed "is not used" from cfg_interrupt_do[7:0] description.<br><br>Chapter 4: Added content to Running the Simulation. Added reference to Synplicity to step 2, page 62. Added the implement folder and subfolders to Example Design. Added introductory paragraph to <project directory>/<component name>.<br><br>Chapter 5: Replaced Figure 5-5 and Figure 5-13. Added note on page 71 about screen captures providing general information. Corrected number of lanes in Table 5-1. Added Buffering Optimized bullet to Block RAM Configuration Options. Indicated that the De-emphasis field in the Link Control 2 Register is not editable. Added Disable TX ASPM L0s and Link Number bullets to Advanced Physical Layer. Removed GT DRP Ports bullet from Debug Ports.<br><br>Chapter 6: Changed the *PCI Express Base Specification* version number to 2.0 in the introductory paragraph. In Basic TLP Transmit Operation, added bullet and note about user-implemented Configuration Space requests, and added new paragraph about filtering. In Table 6-8 and Table 6-9, added table note about TLP. In Design Considerations for a Directed Link Change, added statement about Link Partner being 5.0 Gb/s (Gen2) capable to sixth bullet and added new bullet about not initiating a retrain. Added table note defining checkmarks and X's in Table 6-35. Added a paragraph about when Per-Vector Masking is enabled to the end of MSI Mode on page 181.<br><br>Chapter 7: In Table 7-1, corrected X0Y8 in x8 column of the FF1759/X0Y0 row. In Table 7-2, added HX380T to FF1154/X0Y0, FF1154/X0Y1, FF1154/X0Y2, and FF1154/X0Y3 rows; added HX380T and HX255T to FF1923/X0Y1 row; changed device to HX255T in FF1923/X0Y0 row; removed FF1923/X0Y1 row; and added table notes 1 and 2.<br><br>Chapter 8: Revised Figure 8-1. Added paragraph to the end of Board Power in Real-World Systems on page 209. Added the Hot Plug Systems and Configuration Time Matrix: Non-ATX-Based Motherboards sections. Corrected measurement to 250 ms in Successful FPGA Recognition, page 213.<br><br>Added Chapter 9, Known Restrictions.<br><br>Appendix A: Added VHDL Test Selection section. Added IUS flow throughout the VHDL Flow section.<br><br>Appendix B: Removed Figure B-3, which was a duplicate of Figure B-2. |

| Date | Version | Revision |
|------|---------|----------|
| 09/21/10 | 5.1 | Updated ISE software to v12.3. Added cfg_pm_send_pme_to_n to Table 2-19. |
| | | Added Cadence INCISIV to Example Design Elements, page 55. Removed discussion about example design from Example Design Elements, page 58. Updated step 4, page 59 in Generating the Core. Added ISim to Simulating the Example Design, page 61. Added isim_cmd.tcl, simulate_isim.bat/simulate_isim.sh, and wave.wcfg to Table 4-13. |
| | | Updated first bullet under Design Considerations for a Directed Link Change, page 151. Updated Figure 6-52, Figure 6-53, and Figure 6-54. Updated third bullet in Reset, page 183. |
| | | Added SX315T to FF1156 package in Table 7-1. Added note 2 to Table 7-2. |
| | | Added Chapter 10, Hardware Verification. |
| | | Added ISim to Simulating the Design, page 237, Verilog Test Selection, page 238, Table A-11, and VHDL Flow, page 239. Replaced IUS with INCISIV in VHDL Flow, page 239. |
| | | Added ISim to Simulating the Design, page 257 and Table B-2. |
| | | Removed 5.0 Gb/s rate from description of PIPETXDEEMPH in Table G-4. Added 100b and 101b to description of CFGDEVCONTROLMAXREADREQ[2:0] in Table G-13. |

# *Table of Contents*

# Chapter 4: Getting Started Example Design

# Chapter 5: Generating and Customizing the Core

## Chapter 6: Designing with the Core

# Chapter 7: Core Constraints

# Chapter 8: FPGA Configuration

# Chapter 9: Known Restrictions

## Chapter 10: Hardware Verification

## Appendix A: Example Design and Model Test Bench for Endpoint Configuration

## Appendix B: Example Design and Model Test Bench for Root Port Configuration

# Appendix E:  Managing Receive-Buffer Space for Inbound Completions

# Appendix F:  Board Design Guidelines

## Appendix G: PCIE_2_0 Port Descriptions

## Appendix H: PCIE_2_0 Attribute Descriptions

## Appendix I: PCIE_2_0 Timing Parameter Descriptions

# EXILINX®

<div align="right"><em>Preface</em></div>

# *About This Guide*

This document describes the function and operation of the Virtex®-6 FPGA Integrated Block for PCI Express®, including how to design, customize, and implement it.

## Guide Contents

This manual contains these chapters and appendices:

- Chapter 1, Introduction, describes the core and related information, including recommended design experience and additional resources.

- Chapter 2, Core Overview, describes the main components of the Integrated Block architecture.

- Chapter 3, Licensing the Core, provides information about obtaining a license for the core.

- Chapter 4, Getting Started Example Design, provides instructions for quickly generating, simulating, and implementing the example design using the demonstration test bench.

- Chapter 5, Generating and Customizing the Core, describes how to use the graphical user interface (GUI) to configure the integrated block using the CORE Generator™ software.

- Chapter 6, Designing with the Core, provides instructions on how to design a device using the Integrated Block core.

- Chapter 7, Core Constraints, discusses the required and optional constraints for the integrated block.

- Chapter 8, FPGA Configuration, discusses considerations for FPGA configuration and PCI Express.

- Chapter 9, Known Restrictions, describes restrictions or issues where the integrated block deviates from the PCI Base Specification or in cases where the specification is ambiguous.

- Appendix A, Programmed Input/Output: Endpoint Example Design, describes the Programmed Input/Output (PIO) example design for use with the core and the Root Port model test bench environment, which provides a test program interface for use with the PIO example design.

- Appendix B, Example Design and Model Test Bench for Root Port Configuration, describes the Configurator example design for use with the core, and the Endpoint Model test bench environment for use with the Configurator example design.

- Appendix C, Migration Considerations, defines the differences in behavior and options between the Virtex-6 FPGA Integrated Block for PCI Express and the Endpoint Block Plus for PCI Express.

- **Appendix D, Debugging Designs**, provides information on resources available on the Xilinx support website, available debug tools, and a step-by-step process for debugging designs that use the Virtex-6 FPGA Integrated Block for PCI Express.

- **Appendix E, Managing Receive-Buffer Space for Inbound Completions**, provides example methods for handling finite receive buffer space for inbound completions with regards to the PCI Express Endpoint requirement to advertise infinite completion credits.

- **Appendix F, Board Design Guidelines**, discusses topics related to implementing a PCI Express design that uses the Virtex-6 FPGA on a printed circuit board.

- **Appendix G, PCIE_2_0 Port Descriptions**.

- **Appendix H, PCIE_2_0 Attribute Descriptions**.

- **Appendix I, PCIE_2_0 Timing Parameter Descriptions**.

# Additional Documentation

The following documents are also available for download at:
http://www.xilinx.com/support/documentation/virtex-6.htm.

- Virtex-6 Family Overview

    The features and product selection of the Virtex-6 family are outlined in this overview.

- Virtex-6 FPGA Data Sheet: DC and Switching Characteristics

    This data sheet contains the DC and Switching Characteristic specifications for the Virtex-6 family.

- Virtex-6 FPGA Packaging and Pinout Specifications

    This specification includes the tables for device/package combinations and maximum I/Os, pin definitions, pinout tables, pinout diagrams, mechanical drawings, and thermal specifications.

- Virtex-6 FPGA SelectIO Resources User Guide

    This guide describes the SelectIO™ resources available in all Virtex-6 devices.

- Virtex-6 FPGA Clocking Resources User Guide

    This guide describes the clocking resources available in all Virtex-6 devices, including the MMCM and PLLs.

- Virtex-6 FPGA Block RAM Resources User Guide

    The functionality of the block RAM and FIFO are described in this user guide.

- Virtex-6 FPGA Configurable Logic Block User Guide

    This guide describes the capabilities of the configurable logic blocks (CLBs) available in all Virtex-6 devices.

- Virtex-6 FPGA GTH Transceivers User Guide

    This guide describes the GTH transceivers available in all Virtex-6 HXT FPGAs except the XC6VHX250T and the XC6VHX380T in the FF1154 package.

- Virtex-6 FPGA GTX Transceivers User Guide

    This guide describes the GTX transceivers available in all Virtex-6 FPGAs except the XC6VLX760.

- Virtex-6 FPGA DSP48E1 Slice User Guide

This guide describes the architecture of the DSP48E1 slice in Virtex-6 FPGAs and provides configuration examples.

- Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide

This guide describes the dedicated Tri-Mode Ethernet Media Access Controller available in all Virtex-6 FPGAs except the XC6VLX760.

- Virtex-6 FPGA System Monitor User Guide

The System Monitor functionality available in all Virtex-6 devices is outlined in this guide.

- Virtex-6 FPGA PCB Design Guide

This guide provides information on PCB design for Virtex-6 devices, with a focus on strategies for making design decisions at the PCB and interface level.

## Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support.

**☰ XILINX®**

<div align="right">

*Chapter 1*

</div>

# *Introduction*

This chapter introduces the Virtex®-6 FPGA Integrated Block for PCI Express® core and provides related information including system requirements and recommended design experience.

## About the Core

The Virtex-6 FPGA Integrated Block for PCI Express core is a reliable, high-bandwidth, scalable serial interconnect building block for use with the Virtex-6 FPGA family. The core instantiates the Virtex-6 FPGA Integrated Block for PCI Express found in the Virtex-6 family, and supports both Verilog-HDL and VHDL.

The Virtex-6 FPGA Integrated Block for PCI Express is a CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the Virtex-6 FPGA Integrated Block for PCI Express product page. For information about licensing options, see Chapter 3, Licensing the Core.

## System Requirements

### Windows

- Windows XP Professional 32-bit/64-bit
- Windows Vista Business 32-bit/64-bit

### Linux

- Red Hat Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

### Software

- ISE® v12.3 software

  Check the release notes for the required Service Pack; ISE software Service Packs can be downloaded from www.xilinx.com/support/download/index.htm.

# Recommended Design Experience

Although the Virtex-6 FPGA Integrated Block for PCI Express core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high-performance, pipelined FPGA designs using Xilinx implementation software and User Constraints Files (UCFs) is recommended.

# Additional Core Resources

For detailed information and updates about the integrated block, refer to these documents:

- *LogiCORE IP Virtex-6 Integrated Block for PCI Express Data Sheet*
- *LogiCORE IP Virtex-6 Integrated Block for PCI Express Release Notes*

Additional information and resources related to the PCI Express technology are available from these websites:

- PCI Express at PCI-SIG
- PCI Express Developer's Forum

# *Core Overview*

This chapter describes the main components of the Virtex®-6 FPGA Integrated Block for PCI Express® architecture.

## Overview

The Virtex-6 FPGA Integrated Block for PCI Express contains full support for 2.5 Gb/s and 5.0 Gb/s PCI Express Endpoint and Root Port configurations. Table 2-1 defines the Integrated Block for PCIe® solutions.

*Table 2-1:* **Product Overview**

| Product Name | User Interface Width | Supported Lane Widths |
|---|---|---|
| 1-lane at 2.5 Gb/s, 5.0 Gb/s | 64 | x1 |
| 2-lane at 2.5 Gb/s, 5.0 Gb/s | 64 | x1, x2[1] |
| 4-lane at 2.5 Gb/s, 5.0 Gb/s | 64 | x1, x2, x4[1] |
| 8-lane at 2.5 Gb/s | 64 | x1, x2, x4, x8[1] |
| 8-lane at 2.5 Gb/s and 5.0 Gb/s[2] | 128 | x1, x2, x4, x8[1] |

**Notes:**
1. See Link Training: 2-Lane, 4-Lane, and 8-Lane Components for additional information.
2. Endpoint configuration only.

The LogiCORE IP Virtex-6 FPGA Integrated Block for PCI Express core internally instantiates the Virtex-6 FPGA Integrated Block for PCI Express (PCIE_2_0). The integrated block follows the *PCI Express Base Specification* layering model, which consists of the Physical, Data Link, and Transaction layers. The integrated block is compliant with the *PCI Express Base Specification, rev. 2.0*.
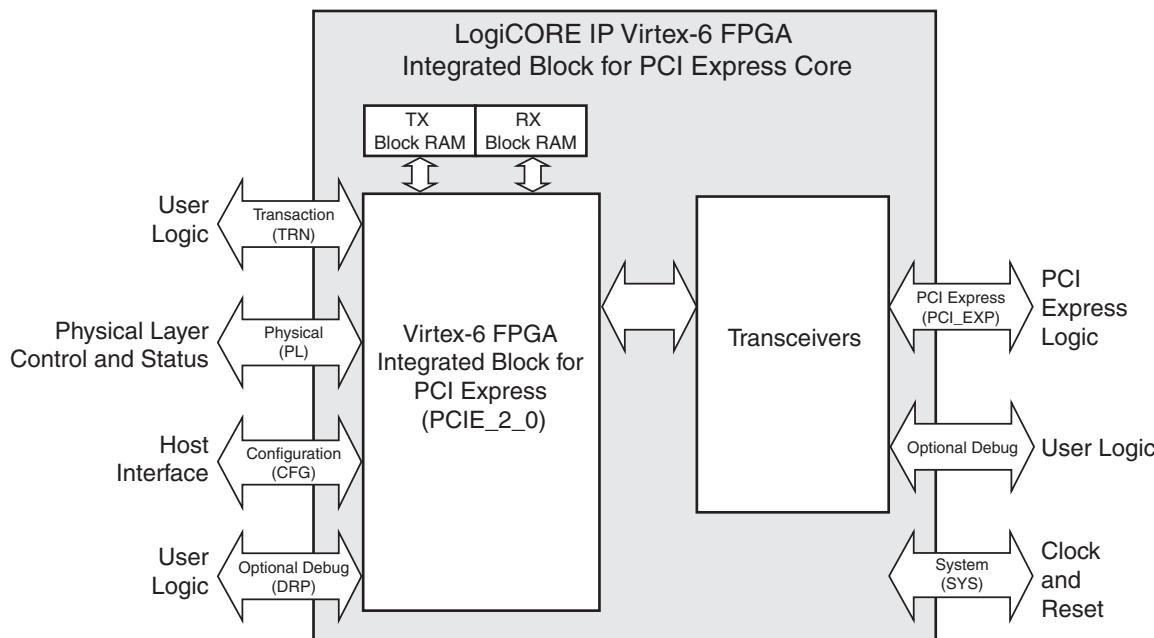
Figure 2-1 illustrates these interfaces to the Virtex-6 FPGA Integrated Block for PCI Express:

- System (SYS) interface
- PCI Express (PCI_EXP) interface
- Configuration (CFG) interface
- Transaction (TRN) interface
- Physical Layer Control and Status (PL) interface

The core uses packets to exchange information between the various modules. Packets are formed in the Transaction and Data Link Layers to carry information from the transmitting component to the receiving component. Necessary information is added to the packet

being transmitted, which is required to handle the packet at those layers. At the receiving end, each layer of the receiving element processes the incoming packet, strips the relevant information and forwards the packet to the next layer.

As a result, the received packets are transformed from their Physical Layer representation to their Data Link Layer representation and the Transaction Layer representation.



UG517_c2_02_020910

*Figure 2-1:* **Top-Level Functional Blocks and Interfaces**

# Protocol Layers

The functions of the protocol layers, as defined by the *PCI Express Base Specification*, include generation and processing of Transaction Layer Packets (TLPs), flow control management, initialization, power management, data protection, error checking and retry, physical link interface initialization, maintenance and status tracking, serialization, deserialization, and other circuitry for interface operation. Each layer is defined in the next subsections.

## Transaction Layer

The Transaction Layer is the upper layer of the PCI Express architecture, and its primary function is to accept, buffer, and disseminate Transaction Layer packets or TLPs. TLPs communicate information through the use of memory, I/O, configuration, and message transactions. To maximize the efficiency of communication between devices, the Transaction Layer enforces PCI-compliant Transaction ordering rules and manages TLP buffer space via credit-based flow control.

## Data Link Layer

The Data Link Layer acts as an intermediate stage between the Transaction Layer and the Physical Layer. Its primary responsibility is to provide a reliable mechanism for the exchange of TLPs between two components on a link.

Services provided by the Data Link Layer include data exchange (TLPs), error detection and recovery, initialization services and the generation and consumption of Data Link Layer Packets (DLLPs). DLLPs are used to transfer information between Data Link Layers of two directly connected components on the link. DLLPs convey information such as Power Management, Flow Control, and TLP acknowledgments.

## Physical Layer

The Physical Layer interfaces the Data Link Layer with signalling technology for link data interchange, and is subdivided into the Logical sub-block and the Electrical sub-block.

- The Logical sub-block frames and deframes TLPs and DLLPs. It also implements the Link Training and Status State machine (LTSSM), which handles link initialization, training, and maintenance. Scrambling, descrambling, and 8B/10B encoding and decoding of data is also performed in this sub-block.

- The Electrical sub-block defines the input and output buffer characteristics that interfaces the device to the PCIe link.

The Physical Layer also supports Lane Reversal (for multi-lane designs) and Lane Polarity Inversion, as indicated in the *PCI Express Base Specification, rev. 2.0* requirement.

## Configuration Management

The Configuration Management layer maintains the PCI™ Type 0 Endpoint configuration space and supports these features:

- Implements the PCI Configuration Space
- Supports Configuration Space accesses
- Power Management functions
- Implements error reporting and status functionality
- Implements packet processing functions
  - Receive
    - Configuration Reads and Writes
  - Transmit
    - Completions with or without data
    - TLM Error Messaging
    - User Error Messaging
    - Power Management Messaging/Handshake
- Implements MSI and INTx interrupt emulation
- Optionally implements MSIx Capability Structure in the PCI Configuration Space
- Optionally implements the Device Serial Number Capability in the PCI Express Extended Capability Space
- Optionally implements Virtual Channel Capability (support only for VC0) in the PCI Express Extended Capability Space
- Optionally implements Xilinx defined Vendor Specific Capability Structure in the PCI Express Extended Capability space to provide Loopback Control and Status

# PCI Configuration Space

The PCI configuration space consists of three primary parts, illustrated in Table 2-2. These include:

- Legacy PCI v3.0 Type 0/1 Configuration Space Header
  - Type 0 Configuration Space Header used by Endpoint applications (see Table 2-3)
  - Type 1 Configuration Space Header used by Root Port applications (see Table 2-4)
- Legacy Extended Capability Items
  - PCIe Capability Item
  - Power Management Capability Item
  - Message Signaled Interrupt (MSI) Capability Item
  - MSI-X Capability Item (optional)
- PCIe Extended Capabilities
  - Device Serial Number Extended Capability Structure (optional)
  - Virtual Channel Extended Capability Structure (optional)
  - Vendor Specific Extended Capability Structure (optional)

The core implements up to four legacy extended capability items. The remaining legacy extended capability space from address `0xA8` to `0xFF` is reserved or user-definable (Endpoint configuration only). Also, the locations for any optional capability structure that is not implemented is reserved. If the user does not use this space, the core returns `0x00000000` when this address range is read. If the user chooses to implement registers within user definable locations in the range `0xA8` to `0xFF`, this space must be implemented in the User Application. The user is also responsible for returning `0x00000000` for any address within this range that is not implemented in the User Application.

For more information about enabling this feature, see Chapter 5, Generating and Customizing the Core. For more information about designing with this feature, see Design with Configuration Space Registers and Configuration Interface in Chapter 6.

The core optionally implements up to three PCI Express Extended Capabilities. The remaining PCI Express Extended Capability Space is available for users to implement (Endpoint configuration only). The starting address of the space available to the users depends on which, if any, of the three optional PCIe Extended Capabilities are implemented. If the user chooses to implement registers in this space, the user can select the starting location of this space, and this space must be implemented in the User Application. For more information about enabling this feature, see PCI Express Extended Capabilities in Chapter 5. For more information about designing with this feature, see Design with Configuration Space Registers and Configuration Interface in Chapter 6.

*Table 2-2:* **Common PCI Configuration Space Header**

| 31 | | 16 | 15 | | 0 | |
|---|---|---|---|---|---|---|
| Device ID | | | Vendor ID | | | 000h |
| Status | | | Command | | | 004h |
| Class Code | | | | Rev ID | | 008h |
| BIST | Header | | Lat Timer | Cache Ln | | 00Ch |
| | | | | | | 010h |
| | | | | | | 014h |
| | | | | | | 018h |
| | | | | | | 01Ch |
| Header Type Specific | | | | | | 020h |
| (see Table 2-3 and Table 2-4) | | | | | | 024h |
| | | | | | | 028h |
| | | | | | | 02Ch |
| | | | | | | 030h |
| | | | | CapPtr | | 034h |
| | | | | | | 038h |
| | | | Intr Pin | Intr Line | | 03Ch |
| PM Capability | | | NxtCap | PM Cap | | 040h |
| Data | BSE | | PMCSR | | | 044h |
| Optional[1] MSI Control | | | NxtCap | MSI Cap | | 048h |
| Message Address (Lower) | | | | | | 04Ch |
| Message Address (Upper) | | | | | | 050h |
| Reserved | | | Message Data | | | 054h |
| Mask Bits | | | | | | 058h |
| Pending Bits | | | | | | 05Ch |
| PE Capability | | | NxtCap | PE Cap | | 060h |
| PCI Express Device Capabilities | | | | | | 064h |
| Device Status | | | Device Control | | | 068h |
| PCI Express Link Capabilities | | | | | | 06Ch |
| Link Status | | | Link Control | | | 070h |
| Root Port Only[2] Slot Capabilities | | | | | | 074h |
| Slot Status | | | Slot Control | | | 078h |
| Root Capabilities | | | Root Control | | | 07Ch |
| Root Status | | | | | | 080h |
| PCI Express Device Capabilities 2 | | | | | | 084h |
| Device Status 2 | | | Device Control 2 | | | 088h |
| PCI Express Link Capabilities 2 | | | | | | 08Ch |

*Table 2-2:* **Common PCI Configuration Space Header** *(Cont'd)*

| | 31 16 | | 15 0 | |
|---|---|---|---|---|
| | Link Status 2 | | Link Control 2 | 090h |
| | Unimplemented Configuration Space (Returns 0x00000000) | | | 094h–098h |
| Optional | MSIx Control | NxtCap | MSIx Cap | 09Ch |
| Optional | Table Offset | | Table BIR | 0A0h |
| | PBA Offset | | PBA BIR | 0A4h |
| | Reserved Legacy Configuration Space (Returns 0x00000000) | | | 0A8h–0FFh |
| Optional | Next Cap | Cap. Ver. | PCI Exp. Extended Capability - DSN | 100h |
| | PCI Express Device Serial Number (1st) | | | 104h |
| | PCI Express Device Serial Number (2nd) | | | 108h |
| Optional | Next Cap | Cap. Ver. | PCI Exp. Extended Capability - VC | 10Ch |
| | Port VC Capability Register 1 | | | 110h |
| | Port VC Capability Register 2 | | | 114h |
| | Port VC Status | | Port VC Control | 118h |
| | VC Resource Capability Register 0 | | | 11Ch |
| | VC Resource Control Register 0 | | | 120h |
| | VC Resource Status Register 0 | | | 124h |
| Optional | Next Cap | Cap. Ver. | PCI Exp. Extended Capability - Vsec | 128h |
| | Vendor Specific Header | | | 12Ch |
| | Vendor Specific - Loopback Command | | | 130h |
| | Vendor Specific - Loopback Status | | | 134h |
| | Vendor Specific - Error Count #1 | | | 138h |
| | Vendor Specific - Error Count #2 | | | 13Ch |
| | Reserved Extended Configuration Space (Returns Completion with 0x00000000) | | | 140h–FFFh |

**Notes:**

1. The MSI Capability Structure varies dependent on the selections in the CORE Generator tool GUI.
2. Reserved for Endpoint configurations (returns 0x00000000).

*Table 2-3:* **Type 0 PCI Configuration Space Header**

| 31 | | | 16 | 15 | | | 0 | |
|---|---|---|---|---|---|---|---|---|
| Device ID | | | | Vendor ID | | | | 00h |
| Status | | | | Command | | | | 04h |
| Class Code | | | | | | Rev ID | | 08h |
| BIST | Header | | Lat Timer | | | Cache Ln | | 0Ch |
| Base Address Register 0 | | | | | | | | 10h |
| Base Address Register 1 | | | | | | | | 14h |
| Base Address Register 2 | | | | | | | | 18h |
| Base Address Register 3 | | | | | | | | 1Ch |
| Base Address Register 4 | | | | | | | | 20h |
| Base Address Register 5 | | | | | | | | 24h |
| Cardbus CIS Pointer | | | | | | | | 28h |
| Subsystem ID | | | | Subsystem Vendor ID | | | | 2Ch |
| Expansion ROM Base Address | | | | | | | | 30h |
| Reserved | | | | | | CapPtr | | 34h |
| Reserved | | | | | | | | 38h |
| Max Lat | Min Gnt | | Intr Pin | | | Intr Line | | 3Ch |

*Table 2-4:* **Type 1 PCI Configuration Space Header**

| 31 | | 16 | 15 | | 0 | |
|---|---|---|---|---|---|---|
| Device ID | | | Vendor ID | | | 00h |
| Status | | | Command | | | 04h |
| Class Code | | | | Rev ID | | 08h |
| BIST | Header | | Lat Timer | | Cache Ln | 0Ch |
| Base Address Register 0 | | | | | | 10h |
| Base Address Register 1 | | | | | | 14h |
| Second Lat Timer | Sub Bus Number | | Second Bus Number | | Primary Bus Number | 18h |
| Secondary Status | | | I/O Limit | | I/O Base | 1Ch |
| Memory Limit | | | Memory Base | | | 20h |
| Prefetchable Memory Limit | | | Prefetchable Memory Base | | | 24h |
| Prefetchable Base Upper 32 Bits | | | | | | 28h |
| Prefetchable Limit Upper 32 Bits | | | | | | 2Ch |
| I/O Limit Upper 16 Bits | | | I/O Base Upper 16 Bits | | | 30h |
| Reserved | | | | | CapPtr | 34h |
| Expansion ROM Base Address | | | | | | 38h |
| Bridge Control | | | Intr Pin | | Intr Line | 3Ch |

# Core Interfaces

The Virtex-6 FPGA Integrated Block for PCI Express core includes top-level signal interfaces that have sub-groups for the receive direction, transmit direction, and signals common to both directions.

## System Interface

The System (SYS) interface consists of the system reset signal (sys_reset_n) and the system clock signal (sys_clk), as described in Table 2-5.

*Table 2-5:* **System Interface Signals**

| Function | Signal Name | Direction | Description |
|----------|-------------|-----------|-------------|
| System Reset | sys_reset_n | Input | Asynchronous, active Low signal. sys_reset_n must be asserted for at least 1500 ns during power on and warm reset operations. |
| System Clock | sys_clk | Input | Reference clock: Selectable frequency 100 MHz, 125 MHz, or 250 MHz. |

The system reset signal is an asynchronous active-Low input. The assertion of sys_reset_n causes a hard reset of the entire core. The system input clock must be 100 MHz, 125 MHz, or 250 MHz, as selected in the CORE Generator™ software GUI.

## PCI Express Interface

The PCI Express (PCI_EXP) interface consists of differential transmit and receive pairs organized in multiple lanes. A PCI Express lane consists of a pair of transmit differential signals (pci_exp_txp, pci_exp_txn) and a pair of receive differential signals {pci_exp_rxp, pci_exp_rxn}. The 1-lane core supports only Lane 0, the 2-lane core supports lanes 0-1, the 4-lane core supports lanes 0-3, and the 8-lane core supports lanes 0-7. Transmit and receive signals of the PCI_EXP interface are defined in Table 2-6 through Table 2-9.

*Table 2-6:* **PCI Express Interface Signals for the 1-Lane Core**

| Lane Number | Name | Direction | Description |
|-------------|------|-----------|-------------|
| 0 | pci_exp_txp0 | Output | PCI Express Transmit Positive: Serial Differential Output 0 (+) |
| 0 | pci_exp_txn0 | Output | PCI Express Transmit Negative: Serial Differential Output 0 (–) |
| 0 | pci_exp_rxp0 | Input | PCI Express Receive Positive: Serial Differential Input 0 (+) |
| 0 | pci_exp_rxn0 | Input | PCI Express Receive Negative: Serial Differential Input 0 (–) |

*Table 2-7:* **PCI Express Interface Signals for the 2-Lane Core**

| Lane Number | Name | Direction | Description |
|---|---|---|---|
| 0 | pci_exp_txp0 | Output | PCI Express Transmit Positive: Serial Differential Output 0 (+) |
| 0 | pci_exp_txn0 | Output | PCI Express Transmit Negative: Serial Differential Output 0 (–) |
| 0 | pci_exp_rxp0 | Input | PCI Express Receive Positive: Serial Differential Input 0 (+) |
| 0 | pci_exp_rxn0 | Input | PCI Express Receive Negative: Serial Differential Input 0 (–) |
| 1 | pci_exp_txp1 | Output | PCI Express Transmit Positive: Serial Differential Output 1 (+) |
| 1 | pci_exp_txn1 | Output | PCI Express Transmit Negative: Serial Differential Output 1 (–) |
| 1 | pci_exp_rxp1 | Input | PCI Express Receive Positive: Serial Differential Input 1 (+) |
| 1 | pci_exp_rxn1 | Input | PCI Express Receive Negative: Serial Differential Input 1 (–) |

*Table 2-8:* **PCI Express Interface Signals for the 4-Lane Core**

| Lane Number | Name | Direction | Description |
|---|---|---|---|
| 0 | pci_exp_txp0 | Output | PCI Express Transmit Positive: Serial Differential Output 0 (+) |
| 0 | pci_exp_txn0 | Output | PCI Express Transmit Negative: Serial Differential Output 0 (–) |
| 0 | pci_exp_rxp0 | Input | PCI Express Receive Positive: Serial Differential Input 0 (+) |
| 0 | pci_exp_rxn0 | Input | PCI Express Receive Negative: Serial Differential Input 0 (–) |
| 1 | pci_exp_txp1 | Output | PCI Express Transmit Positive: Serial Differential Output 1 (+) |
| 1 | pci_exp_txn1 | Output | PCI Express Transmit Negative: Serial Differential Output 1 (–) |
| 1 | pci_exp_rxp1 | Input | PCI Express Receive Positive: Serial Differential Input 1 (+) |
| 1 | pci_exp_rxn1 | Input | PCI Express Receive Negative: Serial Differential Input 1 (–) |
| 2 | pci_exp_txp2 | Output | PCI Express Transmit Positive: Serial Differential Output 2 (+) |

*Table 2-8:* **PCI Express Interface Signals for the 4-Lane Core** *(Cont'd)*

| Lane Number | Name | Direction | Description |
|---|---|---|---|
| 2 | pci_exp_txn2 | Output | PCI Express Transmit Negative: Serial Differential Output 2 (–) |
| 2 | pci_exp_rxp2 | Input | PCI Express Receive Positive: Serial Differential Input 2 (+) |
| 2 | pci_exp_rxn2 | Input | PCI Express Receive Negative: Serial Differential Input 2 (–) |
| 3 | pci_exp_txp3 | Output | PCI Express Transmit Positive: Serial Differential Output 3 (+) |
| 3 | pci_exp_txn3 | Output | PCI Express Transmit Negative: Serial Differential Output 3 (–) |
| 3 | pci_exp_rxp3 | Input | PCI Express Receive Positive: Serial Differential Input 3 (+) |
| 3 | pci_exp_rxn3 | Input | PCI Express Receive Negative: Serial Differential Input 3 (–) |

*Table 2-9:* **PCI Express Interface Signals for the 8-Lane Core**

| Lane Number | Name | Direction | Description |
|---|---|---|---|
| 0 | pci_exp_txp0 | Output | PCI Express Transmit Positive: Serial Differential Output 0 (+) |
| 0 | pci_exp_txn0 | Output | PCI Express Transmit Negative: Serial Differential Output 0 (–) |
| 0 | pci_exp_rxp0 | Input | PCI Express Receive Positive: Serial Differential Input 0 (+) |
| 0 | pci_exp_rxn0 | Input | PCI Express Receive Negative: Serial Differential Input 0 (–) |
| 1 | pci_exp_txp1 | Output | PCI Express Transmit Positive: Serial Differential Output 1 (+) |
| 1 | pci_exp_txn1 | Output | PCI Express Transmit Negative: Serial Differential Output 1 (–) |
| 1 | pci_exp_rxp1 | Input | PCI Express Receive Positive: Serial Differential Input 1 (+) |
| 1 | pci_exp_rxn1 | Input | PCI Express Receive Negative: Serial Differential Input 1 (–) |
| 2 | pci_exp_txp2 | Output | PCI Express Transmit Positive: Serial Differential Output 2 (+) |
| 2 | pci_exp_txn2 | Output | PCI Express Transmit Negative: Serial Differential Output 2 (–) |
| 2 | pci_exp_rxp2 | Input | PCI Express Receive Positive: Serial Differential Input 2 (+) |

*Table 2-9:* **PCI Express Interface Signals for the 8-Lane Core** *(Cont'd)*

| Lane Number | Name | Direction | Description |
|---|---|---|---|
| 2 | pci_exp_rxn2 | Input | PCI Express Receive Negative: Serial Differential Input 2 (–) |
| 3 | pci_exp_txp3 | Output | PCI Express Transmit Positive: Serial Differential Output 3 (+) |
| 3 | pci_exp_txn3 | Output | PCI Express Transmit Negative: Serial Differential Output 3 (–) |
| 3 | pci_exp_rxp3 | Input | PCI Express Receive Positive: Serial Differential Input 3 (+) |
| 3 | pci_exp_rxn3 | Input | PCI Express Receive Negative: Serial Differential Input 3 (–) |
| 4 | pci_exp_txp4 | Output | PCI Express Transmit Positive: Serial Differential Output 4 (+) |
| 4 | pci_exp_txn4 | Output | PCI Express Transmit Negative: Serial Differential Output 4 (–) |
| 4 | pci_exp_rxp4 | Input | PCI Express Receive Positive: Serial Differential Input 4 (+) |
| 4 | pci_exp_rxn4 | Input | PCI Express Receive Negative: Serial Differential Input 4 (–) |
| 5 | pci_exp_txp5 | Output | PCI Express Transmit Positive: Serial Differential Output 5 (+) |
| 5 | pci_exp_txn5 | Output | PCI Express Transmit Negative: Serial Differential Output 5 (–) |
| 5 | pci_exp_rxp5 | Input | PCI Express Receive Positive: Serial Differential Input 5 (+) |
| 5 | pci_exp_rxn5 | Input | PCI Express Receive Negative: Serial Differential Input 5 (–) |
| 6 | pci_exp_txp6 | Output | PCI Express Transmit Positive: Serial Differential Output 6 (+) |
| 6 | pci_exp_txn6 | Output | PCI Express Transmit Negative: Serial Differential Output 6 (–) |
| 6 | pci_exp_rxp6 | Input | PCI Express Receive Positive: Serial Differential Input 6 (+) |
| 6 | pci_exp_rxn6 | Input | PCI Express Receive Negative: Serial Differential Input 6 (–) |
| 7 | pci_exp_txp7 | Output | PCI Express Transmit Positive: Serial Differential Output 7 (+) |
| 7 | pci_exp_txn7 | Output | PCI Express Transmit Negative: Serial Differential Output 7 (–) |

*Table 2-9:* **PCI Express Interface Signals for the 8-Lane Core** *(Cont'd)*

| Lane Number | Name | Direction | Description |
|---|---|---|---|
| 7 | pci_exp_rxp7 | Input | PCI Express Receive Positive: Serial Differential Input 7 (+) |
| 7 | pci_exp_rxn7 | Input | PCI Express Receive Negative: Serial Differential Input 7 (–) |

# Transaction Interface

The Transaction (TRN) interface provides a mechanism for the user design to generate and consume TLPs. The signal names and signal descriptions for this interface are shown in Table 2-10, Table 2-12, and Table 2-13.

## Common TRN Interface

Table 2-10 defines and describes the common TRN interface signals.

*Table 2-10:* **Common Transaction Interface Signals**

| Name | Direction | Description |
|---|---|---|
| trn_clk | Output | Transaction Clock: Transaction, Configuration, and Physical Layer Control and Status Interface operations are referenced to and synchronous with the rising edge of this clock. This signal is active after power-on, and sys_reset_n has no effect on it. This signal is guaranteed to be stable at the selected operating frequency only after trn_reset_n is deasserted. The trn_clk clock output is a fixed frequency configured in the CORE Generator software. This signal does not change frequencies in case of link recovery or training down.<br><br>See Table 2-11 for recommended and optional frequencies. |
| trn_reset_n | Output | Transaction Reset: Active Low. User logic interacting with the Transaction and Configuration interfaces must use trn_reset_n to return to its quiescent state. This signal is deasserted synchronously with respect to trn_clk, and is deasserted and asserted asynchronously with sys_reset_n assertion. This signal is asserted for core in-band reset events such as Hot Reset or Link Disable. |
| trn_lnk_up_n | Output | Transaction Link Up: Active Low. Transaction link-up is asserted when the core and the connected upstream link partner port are ready and able to exchange data packets. Transaction link-up is deasserted when the core and link partner are attempting to establish communication, or when communication with the link partner is lost due to errors on the transmission channel. This signal is also deasserted when the core is driven to Hot Reset or Link Disable state by the link partner, and all TLPs stored in the core are lost. |
| trn_fc_ph[7:0] | Output | Posted Header Flow Control Credits: The number of Posted Header FC credits for the selected flow control type. |

*Table 2-10:* **Common Transaction Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|------|-----------|-------------|
| trn_fc_pd[11:0] | Output | Posted Data Flow Control Credits: The number of Posted Data FC credits for the selected flow control type. |
| trn_fc_nph[7:0] | Output | Non-Posted Header Flow Control Credits: The number of Non-Posted Header FC credits for the selected flow control type. |
| trn_fc_npd[11:0] | Output | Non-Posted Data Flow Control Credits: The number of Non-Posted Data FC credits for the selected flow control type. |
| trn_fc_cplh[7:0] | Output | Completion Header Flow Control Credits: The number of Completion Header FC credits for the selected flow control type. |
| trn_fc_cpld[11:0] | Output | Completion Data Flow Control Credits: The number of Completion Data FC credits for the selected flow control type. |
| trn_fc_sel[2:0] | Input | Flow Control Informational Select: Selects the type of flow control information presented on the trn_fc_* signals. Possible values:<br>• `000`: Receive buffer available space<br>• `001`: Receive credits granted to the link partner<br>• `010`: Receive credits consumed<br>• `100`: Transmit user credits available<br>• `101`: Transmit credit limit<br>• `110`: Transmit credits consumed |

*Table 2-11:* **Recommended and Optional Transaction Clock (trn_clk) Frequencies**

| Product | Link Speed | Recommended Frequency (MHz) | Optional Frequency (MHz) |
|---------|-----------|------------------------------|---------------------------|
| 1-lane | 2.5 Gb/s | 62.5 | 31.25, 125, 250 |
| 1-lane | 5.0 Gb/s | 62.5 | 125, 250 |
| 2-lane | 2.5 Gb/s | 62.5 | 125, 250 |
| 2-lane | 5.0 Gb/s | 125 | 250 |
| 4-lane | 2.5 Gb/s | 125 | 250 |
| 4-lane | 5.0 Gb/s | 250 | - |
| 8-lane | 2.5 Gb/s | 250 | - |
| 8-lane | 5.0 Gb/s | 250 | - |

**Notes:**

1. Endpoint configuration only.

## Transmit TRN Interface

Table 2-12 defines the transmit (TX) TRN interface signals.

*Table 2-12:* **Transaction Transmit Interface Signals**

| Name | Direction | Description |
|---|---|---|
| trn_tsof_n | Input | Transmit Start-of-Frame (SOF): Active Low. Signals the start of a packet. Valid only along with assertion of trn_tsrc_rdy_n. |
| trn_teof_n | Input | Transmit End-of-Frame (EOF): Active Low. Signals the end of a packet. Valid only along with assertion of trn_tsrc_rdy_n. |
| trn_td[W-1:0] | Input | Transmit Data: Packet data to be transmitted. <table><tr><th>Product</th><th>Data Bus Width (W)</th></tr><tr><td>1-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>2-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>4-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>8-lane (2.5 Gb/s)</td><td>64</td></tr><tr><td>8-lane (5.0 Gb/s)</td><td>128</td></tr></table> |
| trn_trem_n (64-bit interface) <br><br> trn_trem_n[1:0] (128-bit interface) | Input | Transmit Data Remainder: Valid only if trn_teof_n, trn_tsrc_rdy_n, and trn_tdst_rdy_n are all asserted. <br> *64-bit interface*: Legal values are: <br> • trn_trem_n = 0, packet data on trn_td[63:0] <br> • trn_trem_n = 1, packet data on trn_td[63:32] <br> *128-bit interface*: trn_trem_n[1:0] is used for the 128-bit interface. Legal values are: <br> • trn_trem_n[1:0] = 00, packet data on trn_td[127:0] <br> • trn_trem_n[1:0] = 01, packet data on trn_td[127:32] <br> • trn_trem_n[1:0] = 10, packet data on trn_td[127:64] <br> • trn_trem_n[1:0] = 11, packet data on trn_td[127:96] |
| trn_tsrc_rdy_n | Input | Transmit Source Ready: Active Low. Indicates that the User Application is presenting valid data on trn_td. |
| trn_tdst_rdy_n | Output | Transmit Destination Ready: Active Low. Indicates that the core is ready to accept data on trn_td. The simultaneous assertion of trn_tsrc_rdy_n and trn_tdst_rdy_n marks the successful transfer of one data beat on trn_td. |
| trn_tsrc_dsc_n | Input | Transmit Source Discontinue: Active Low. Can be asserted any time starting on the first cycle after SOF. trn_teof_n should be asserted along with trn_tsrc_dsc_n assertion. |

*Table 2-12:* **Transaction Transmit Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|------|-----------|-------------|
| trn_tbuf_av[5:0] | Output | Transmit Buffers Available: Indicates the number of free transmit buffers available in the core. Each free transmit buffer can accommodate one TLP up to the supported Maximum Payload Size. The maximum number of transmit buffers is determined by the Supported Maximum Payload Size and block RAM configuration selected. |
| trn_terr_drop_n | Output | Transmit Error Drop: Active Low. Indicates that the core discarded a packet because of a length violation or, when streaming, data was not presented on consecutive clock cycles. Length violations include packets longer than supported. |
| trn_tstr_n | Input | Transmit Streamed: Active Low. Indicates a packet will be presented on consecutive clock cycles and transmission on the link can begin before the entire packet has been written to the core. Commonly referred as transmit cut-through mode. |
| trn_tcfg_req_n (64-bit interface only) | Output | Transmit Configuration Request: Active Low. Asserted when the core is ready to transmit a Configuration Completion or other internally generated TLP. |
| trn_tcfg_gnt_n (64-bit interface only) | Input | Transmit Configuration Grant: Active Low. Asserted by the User Application in response to trn_tcfg_req_n, to allow the core to transmit an internally generated TLP. Holding trn_tcfg_gnt_n deasserted after trn_tcfg_req_n allows user-initiated TLPs to be given a higher priority of transmission over core-generated TLPs. trn_tcfg_req_n is asserted once for each internally generated packet. It cannot be deasserted immediately following trn_cfg_gnt_n if there are no transmit buffers available. If the user does not wish to alter the prioritization of the transmission of internally generated TLPs, this signal can be continuously asserted. |
| trn_terrfwd_n | Input | Transmit Error Forward: Active Low. This input marks the current packet in progress as error-poisoned. It can be asserted any time between SOF and EOF, inclusive. The trn_terrfwd_n signal must not be asserted if trn_tstr_n is asserted. |

## Receive TRN Interface

Table 2-13 defines the receive (RX) TRN interface signals.

*Table 2-13:* **Receive Transaction Interface Signals**

| Name | Direction | Description |
|---|---|---|
| trn_rsof_n | Output | Receive Start-of-Frame (SOF): Active Low. Signals the start of a packet. Valid only if trn_rsrc_rdy_n is also asserted. |
| trn_reof_n | Output | Receive End-of-Frame (EOF): Active Low. Signals the end of a packet. Valid only if trn_rsrc_rdy_n is also asserted. |
| trn_rd[W-1:0] | Output | Receive Data: Packet data being received. Valid only if trn_rsrc_rdy_n is also asserted. |

| Product | Data Bus Width (W) |
|---|---|
| 1-lane (2.5 Gb/s and 5.0 Gb/s) | 64 |
| 2-lane (2.5 Gb/s and 5.0 Gb/s) | 64 |
| 4-lane (2.5 Gb/s and 5.0 Gb/s) | 64 |
| 8-lane (2.5 Gb/s) | 64 |
| 8-lane (5.0 Gb/s) | 128 |

*128-bit interface only:* Unlike the Transmit TRN interface trn_td[127:0], received TLPs can begin on either the upper QWORD trn_rd[127:64] or lower QWORD trn_rd[63:0] of the bus. See the description of trn_rrem_n[1] for further explanation.

| Name | Direction | Description |
|---|---|---|
| trn_rrem_n (64-bit interface)<br><br>trn_rrem_n[1:0] (128-bit interface) | Output | Receive Data Remainder:<br><br>*64-bit interface*: Only trn_rrem_n is available and used for the 64-bit interface. Valid only if trn_reof_n, trn_rsrc_rdy_n, and trn_rdst_rdy_n are all asserted.<br><br>Legal values are:<br>• trn_rrem_n[0] = 0, packet data on all of trn_rd[63:0]<br>• trn_rrem_n[0] = 1, packet data only on trn_rd[63:32]<br><br>*128-bit interface*:<br>• trn_rrem_n[1]: Valid only if trn_rsrc_rdy_n and trn_rdst_rdy_n are asserted. When asserted along with trn_rsof_n or trn_reof_n, indicates location of SOF and/or EOF within beat.<br>  • trn_rrem_n[1] = 0: Indicates trn_rd[127:64] has SOF and/or trn_rd[63:0] has EOF<br>  • trn_rrem_n[1] = 1: Indicates trn_rd[127:64] has EOF and/or trn_rd[63:0] has SOF<br>• trn_rrem_n[0]: Valid only if trn_reof_n, trn_rsrc_rdy_n, and trn_rdst_rdy_n are all asserted.<br>If trn_rrem_n[1]=0:<br>  • trn_rrem_n[0] = 0, packet data on all of trn_rd[127:0]<br>  • trn_rrem_n[0] = 1, packet data only on trn_rd[127:32]<br>If trn_rrem_n[1]=1:<br>  • trn_rrem_n[0] = 0, packet data on all of trn_rd[127:64]<br>  • trn_rrem_n[0] = 1, packet data only on trn_rd[127:96] |

*Table 2-13:* **Receive Transaction Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|---|---|---|
| trn_rerrfwd_n | Output | Receive Error Forward:<br><br>*64-bit interface*: Active Low. When asserted, marks the packet in progress as error-poisoned. Asserted by the core for the entire length of the packet.<br><br>*128-bit interface*: Active Low. When asserted, marks the current packet in progress as error-poisoned. Asserted by the core for the entire length of the packet. If asserted during a straddled data transfer, SOF takes precedence. |
| trn_rsrc_rdy_n | Output | Receive Source Ready: Active Low. Indicates the core is presenting valid data on trn_rd. |
| trn_rdst_rdy_n | Input | Receive Destination Ready: Active Low. Indicates the User Application is ready to accept data on trn_rd. The simultaneous assertion of trn_rsrc_rdy_n and trn_rdst_rdy_n marks the successful transfer of one data beat on trn_td. |
| trn_rsrc_dsc_n | Output | Receive Source Discontinue:<br><br>*64-bit interface*: Active Low. Indicates the core is aborting the current packet. Asserted when the physical link is going into reset.<br><br>*128-bit interface*: Active Low. Indicates that the core is aborting the current packet transfer. If asserted during a straddled data transfer, the SOF takes precedence. Asserted when the physical link is going into reset. |
| trn_rnp_ok_n | Input | Receive Non-Posted OK: Active Low. The User Application asserts this signal when it is ready to accept Non-Posted Request TLPs. trn_rnp_ok_n must be deasserted when the User Application cannot process received Non-Posted TLPs, so that these can be buffered within the core's receive queue. In this case, Posted and Completion TLPs received after the Non-Posted TLPs will bypass the blocked TLPs.<br><br>When the User Application approaches a state where it is unable to service Non-Posted Requests, it must deassert trn_rnp_ok_n one clock cycle before the core presents EOF of the next-to-last Non-Posted TLP the User Application can accept. |
| trn_rbar_hit_n[6:0] | Output | Receive BAR Hit: Active Low. Indicates BAR(s) targeted by the current receive transaction. Asserted throughout the packet from trn_rsof_n to trn_reof_n.<br>• trn_rbar_hit_n[0]: BAR0<br>• trn_rbar_hit_n[1]: BAR1<br>• trn_rbar_hit_n[2]: BAR2<br>• trn_rbar_hit_n[3]: BAR3<br>• trn_rbar_hit_n[4]: BAR4<br>• trn_rbar_hit_n[5]: BAR5<br>• trn_rbar_hit_n[6]: Expansion ROM Address<br><br>If two BARs are configured into a single 64-bit address, both corresponding trn_rbar_hit_n bits are asserted. |

# Physical Layer Interface

The Physical Layer (PL) interface enables the user design to inspect the status of the Link and Link Partner and control the Link State. Table 2-14 defines and describes the signals for the PL interface.

*Table 2-14:* **Physical Layer Interface Signals**

| Name | Direction | Description |
|------|-----------|-------------|
| pl_initial_link_width[2:0] | Output | Initial Negotiated Link Width: Indicates the link width after the PCI Express port has achieved the first successful link training. Initial Negotiated Link Width represents the widest link width possible during normal operation of the link, and can be equal to or smaller than the capability link width (smaller of the two) supported by link partners.<br>• `000`: Link not Trained<br>• `001`: 1-Lane link<br>• `010`: 2-Lane link<br>• `011`: 4-Lane link<br>• `100`: 8-Lane link |
| pl_lane_reversal_mode[1:0] | Output | Lane Reversal Mode: Indicates the current Lane Reversal mode.<br>• `00`: No reversal<br>• `01`: Lanes 1:0 reversed<br>• `10`: Lanes 3:0 reversed<br>• `11`: Lanes 7:0 reversed |
| pl_link_gen2_capable | Output | Link Gen2 Capable: Indicates that the PCI Express link is 5.0 Gb/s (Gen 2) speed capable (both the Link Partner and the Device are Gen 2 capable)<br>• `0`: Link is not Gen2 Capable<br>• `1`: Link is Gen2 Capable |
| pl_link_partner_gen2_supported | Output | Link Partner Gen2 Capable: Indicates if the PCI Express link partner advertises 5.0 Gb/s (Gen2) capability. Valid only when trn_lnk_up_n is asserted.<br>• `0`: Link partner not Gen2 capable<br>• `1`: Link partner is Gen2 capable |
| pl_link_upcfg_capable | Output | Link Upconfigure Capable: Indicates the PCI Express link is Upconfigure capable. Valid only when trn_lnk_up_n is asserted.<br>• `0`: Link is not Upconfigure capable<br>• `1`: Link is Upconfigure capable |

*Table 2-14:* **Physical Layer Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|---|---|---|
| pl_sel_link_rate | Output | Current Link Rate: Reports the current link speed. Valid only when trn_lnk_up_n is asserted.<br>• `0`: 2.5 Gb/s<br>• `1`: 5.0 Gb/s |
| pl_sel_link_width[1:0] | Output | Current Link Width: Reports the current link width. Valid only when trn_lnk_up_n is asserted.<br>`00`: 1-Lane link<br>`01`: 2-Lane link<br>`10`: 4-Lane link<br>`11`: 8-Lane link |
| pl_ltssm_state[5:0] | Output | LTSSM State: Shows the current LTSSM state (hex).<br>`0`, `1`: Detect Quiet<br>`2`, `3`: Detect Active<br>`4`: Polling Active<br>`5`: Polling Configuration<br>`6`: Polling Compliance, Pre_Send_EIOS<br>`7`: Polling Compliance, Pre_Timeout<br>`8`: Polling Compliance, Send_Pattern<br>`9`: Polling Compliance, Post_Send_EIOS<br>`A`: Polling Compliance, Post_Timeout<br>`B`: Configuration Linkwidth, State 0<br>`C`: Configuration Linkwidth, State 1<br>`D`: Configuration Linkwidth, Accept 0<br>`E`: Configuration Linkwidth, Accept 1<br>`F`: Configuration Lanenum Wait<br>`10`: Configuration Lanenum, Accept<br>`11`: Configuration Complete x1<br>`12`: Configuration Complete x2<br>`13`: Configuration Complete x4<br>`14`: Configuration Complete x8<br>`15`: Configuration Idle<br>`16`: L0<br>`17`: L1 Entry0<br>`18`: L1 Entry1<br>`19`: L1 Entry2 (also used for the L2/L3 Ready pseudo state)<br>`1A`: L1 Idle<br>`1B`: L1 Exit<br>`1C`: Recovery Rcvrlock<br>`1D`: Recovery Rcvrcfg |

*Table 2-14:* **Physical Layer Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|---|---|---|
| pl_ltssm_state[5:0] *(Cont'd)* | Output | 1E: Recovery Speed_0<br><br>1F: Recovery Speed_1<br><br>20: Recovery Idle<br><br>21: Hot Reset<br><br>22: Disabled Entry 0<br><br>23: Disabled Entry 1<br><br>24: Disabled Entry 2<br><br>25: Disabled Idle<br><br>26: Root Port, Configuration, Linkwidth State 0<br><br>27: Root Port, Configuration, Linkwidth State 1<br><br>28: Root Port, Configuration, Linkwidth State 2<br><br>29: Root Port, Configuration, Link Width Accept 0<br><br>2A: Root Port, Configuration, Link Width Accept 1<br><br>2B: Root Port, Configuration, Lanenum_Wait<br><br>2C: Root Port, Configuration, Lanenum_Accept<br><br>2D: Timeout To Detect<br><br>2E: Loopback Entry0<br><br>2F: Loopback Entry1<br><br>30: Loopback Active0<br><br>31: Loopback Exit0<br><br>32: Loopback Exit1<br><br>33: Loopback Master Entry0 |
| pl_directed_link_auton | Input | Directed Autonomous Link Change: Specifies the reason for directed link width and speed change. This must be used in conjunction with pl_directed_link_change[1:0], pl_directed_link_speed, and pl_directed_link_width[1:0] inputs.<br><br>• 0: Link reliability driven<br>• 1: Application requirement driven (autonomous) |
| pl_directed_link_change[1:0] | Input | Directed Link Change Control: Directs the PCI Express Port to initiate a link width and/or speed change. Link change operation must be initiated when trn_lnk_up_n is asserted.<br><br>• 00: No change<br>• 01: Link width<br>• 10: Link speed<br>• 11: Link width and speed (level-triggered) |

*Table 2-14:* **Physical Layer Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|---|---|---|
| pl_directed_link_speed | Input | Directed Target Link Speed: Specifies the target link speed for a directed link change operation, in conjunction with the pl_directed_link_change[1:0] input.<br>• 0: 2.5 Gb/s<br>• 1: 5.0 Gb/s |
| pl_directed_link_width[1:0] | Input | Directed Target Link Width: Specifies the target link width for a directed link change operation, in conjunction with pl_directed_link_change[1:0] input.<br>Encoding Target Link Width:<br>• 00: 1-Lane link<br>• 01: 2-Lane link<br>• 10: 4-Lane link<br>• 11: 8-Lane link |
| pl_upstream_prefer_deemph | Input | Endpoint Preferred Transmitter De-emphasis: Enables the Endpoint to control de-emphasis used on the link at 5.0 Gb/s speeds. pl_upstream_prefer_deemph can be changed in conjunction with pl_directed_link_speed and pl_directed_link_change[1:0] inputs when transitioning from 2.5 Gb/s to 5.0 Gb/s data rates. Value presented on pl_upstream_prefer_deemph depends upon the property of PCI Express physical interconnect channel in use.<br>• 0: –6 dB de-emphasis recommended for short, reflection dominated channels.<br>• 1: –3.5 dB de-emphasis recommended for long, loss dominated channels. |

*Table 2-15:* **Role-Specific Physical Layer Interface Signals: Endpoint**

| Name | Direction | Description |
|---|---|---|
| pl_received_hot_rst | Output | Hot Reset Received: Indicates that an in-band hot reset command has been received. |

*Table 2-16:* **Role-Specific Physical Layer Interface Signals: Root Port**

| Name | Direction | Description |
|---|---|---|
| pl_transmit_hot_rst | Input | Transmit Hot Reset: Active High. Directs the PCI Express Port to transmit an In-Band Hot Reset. |

# Configuration Interface

The Configuration (CFG) interface enables the user design to inspect the state of the Endpoint for PCIe configuration space. The user provides a 10-bit configuration address, which selects one of the 1024 configuration space doubleword (DWORD) registers. The Endpoint returns the state of the selected register over the 32-bit data output port. Table 2-17 defines the Configuration interface signals. See Design with Configuration Space Registers and Configuration Interface, page 155 for usage.

*Table 2-17:* **Configuration Interface Signals**

| Name | Direction | Description |
|---|---|---|
| cfg_do[31:0] | Output | Configuration Data Out: A 32-bit data output port used to obtain read data from the configuration space inside the core. |
| cfg_rd_wr_done_n | Output | Configuration Read Write Done: Active-Low, read-write done signal indicates a successful completion of the user configuration register access operation. <br>• For a user configuration register read operation, this signal validates the cfg_do[31:0] data-bus value. <br>• For a user configuration register write operation, the assertion indicates completion of a successful write operation. |
| cfg_di[31:0] | Input | Configuration Data In: A 32-bit data input port used to provide write data to the configuration space inside the core. |
| cfg_dwaddr[9:0] | Input | Configuration DWORD Address: A 10-bit address input port used to provide a configuration register DWORD address during configuration register accesses. |
| cfg_byte_en_n[3:0] | Input | Configuration Byte Enable: Active-Low byte enables for configuration register write access. |
| cfg_wr_en_n | Input | Configuration Write Enable: Active-Low write enable for configuration register access. |
| cfg_rd_en_n | Input | Configuration Read Enable: Active-Low read enable for configuration register access. |
| cfg_status[15:0] | Output | Configuration Status: Status register from the Configuration Space Header. Not supported. |
| cfg_command[15:0] | Output | Configuration Command: Command register from the Configuration Space Header. |
| cfg_dstatus[15:0] | Output | Configuration Device Status: Device status register from the PCI Express Capability Structure. |
| cfg_dcommand[15:0] | Output | Configuration Device Command: Device control register from the PCI Express Capability Structure. |

*Table 2-17:* **Configuration Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|------|-----------|-------------|
| cfg_dcommand2[15:0] | Output | Configuration Device Command 2: Device control 2 register from the PCI Express Capability Structure. |
| cfg_lstatus[15:0] | Output | Configuration Link Status: Link status register from the PCI Express Capability Structure. |
| cfg_lcommand[15:0] | Output | Configuration Link Command: Link control register from the PCI Express Capability Structure. |
| cfg_pcie_link_state_n[2:0] | Output | PCI Express Link State: This encoded bus reports the PCI Express Link State information to the user.<br>• `000`: "L0"<br>• `001`: "PPM L1"<br>• `010`: "PPM L2/L3 Ready"<br>• `011`: "PM_PME"<br>• `100`: "in or transitioning to/from ASPM L0s"<br>• `101`: "transitioning to/from PPM L1"<br>• `110`: "transition to PPM L2/L3 Ready"<br>• `111`: Reserved |
| cfg_trn_pending_n | Input | User Transaction Pending: Active Low. If asserted, sets the Transactions Pending bit in the Device Status Register.<br>**Note**: The user is required to assert this input if the User Application has not received a completion to an upstream request. |
| cfg_dsn[63:0] | Input | Configuration Device Serial Number: Serial Number Register fields of the Device Serial Number extended capability. |
| cfg_pmcsr_pme_en | Output | PMCSR PME Enable: PME_En bit (bit 8) in the Power Management Control/Status Register. |
| cfg_pmcsr_pme_status | Output | PMCSR PME_Status: PME_Status bit (bit 15) in the Power Management Control/Status Register. |
| cfg_pmcsr_powerstate[1:0] | Output | PMCSR PowerState: PowerState bits (bits 1:0) in the Power Management Control/Status Register. |

*Table 2-18:* **Role-Specific Configuration Interface Signals: Endpoint**

| Name | Direction | Description |
|------|-----------|-------------|
| cfg_bus_number[7:0] | Output | Configuration Bus Number: Provides the assigned bus number for the device. The User Application must use this information in the Bus Number field of outgoing TLP requests. Default value after reset is 00h. Refreshed whenever a Type 0 Configuration packet is received. |
| cfg_device_number[4:0] | Output | Configuration Device Number: Provides the assigned device number for the device. The User Application must use this information in the Device Number field of outgoing TLP requests. Default value after reset is 00000b. Refreshed whenever a Type 0 Configuration packet is received. |
| cfg_function_number[2:0] | Output | Configuration Function Number: Provides the function number for the device. The User Application must use this information in the Function Number field of outgoing TLP request. Function number is hardwired to 000b. |
| cfg_to_turnoff_n | Output | Configuration To Turnoff: Active Low. Output that notifies the user that a PME_TURN_Off message has been received and the CMM will start polling the cfg_turnoff_ok_n input coming in from the user. After cfg_turnoff_ok_n is asserted, CMM sends a PME_To_Ack message to the upstream device. |
| cfg_turnoff_ok_n | Input | Configuration Turnoff OK: Active Low. The User Application can assert this to notify the Endpoint that it is safe to turn off power. |
| cfg_pm_wake_n | Input | Configuration Power Management Wake: A one-clock cycle active-Low assertion informs the core to generate and send a Power Management Wake Event (PM_PME) Message TLP to the upstream link partner. **Note:** The user is required to assert this input only under stable link conditions as reported on the cfg_pcie_link_state[2:0] bus. Assertion of this signal when the PCI Express link is in transition results in incorrect behavior on the PCI Express link. |

*Table 2-19:* **Role-Specific Configuration Interface Signals: Root Port**

| Name | Direction | Description |
|------|-----------|-------------|
| cfg_ds_bus_number | Input | Configuration Downstream Bus Number: Provides the bus number (Requester ID) of the Downstream Port. This is used in TLPs generated inside the core and does not affect the TLPs presented on the Transaction Interface. |
| cfg_ds_device_number | Input | Configuration Downstream Device Number: Provides the device number (Requester ID) of the Downstream Port. This is used in TLPs generated inside the core and does not affect the TLPs presented on the Transaction Interface. |
| cfg_wr_rw1c_as_rw_n | Input | Configuration Write RW1C Bit as RW: Active-Low signal indicating the current write operation should treat any RW1C bit as a RW bit. Normally, a RW1C bit is cleared by writing a "1" to it, and can normally only be set by internal core conditions. However, during a configuration register access operation with this signal asserted, for every bit on cfg_di that is "1", the corresponding RW1C configuration register bit is set to "1". A value of "0" on cfg_di during this operation has no effect, and non-RW1C bits are unaffected regardless of the value on cfg_di. |
| cfg_msg_received | Output | Message Received: Active High. Notifies the user that a Message TLP was received on the Link. |
| cfg_msg_data[15:0] | Output | Message Requester ID: The Requester ID of the Message was received. Valid only along with assertion of cfg_msg_received. |
| cfg_msg_received_ err_cor | Output | Received ERR_COR Message: Active High. Indicates that the core received an ERR_COR Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_ err_non_fatal | Output | Received ERR_NONFATAL Message: Active High. Indicates that the core received an ERR_NONFATAL Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_ err_fatal | Output | Received ERR_FATAL Message: Active High. Indicates that the core received an ERR_FATAL Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_pm_send_pme_ to_n | Input | Configuration Send Turn-off: Asserting this active-Low input causes the Root Port to send Turn Off Message. When the link partner responds with a Turn Off Ack, this is reported on cfg_msg_received_pme_to_ack, and the final transition to L3 Ready is reported on cfg_pcie_link_state. |

*Table 2-19:* **Role-Specific Configuration Interface Signals: Root Port** *(Cont'd)*

| Name | Direction | Description |
|------|-----------|-------------|
| cfg_msg_received_err_pme_to_ack | Output | Received PME_TO_Ack Message: Active High. Indicates that the core received an PME_TO_Ack Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_assert_inta | Output | Received Assert_INTA Message: Active High. Indicates that the core received an Assert INTA Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_assert_intb | Output | Received Assert_INTB Message: Active High. Indicates that the core received an Assert_INTB Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_assert_intc | Output | Received Assert_INTC Message: Active High. Indicates that the core received an Assert_INTC Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_assert_intd | Output | Received Assert_INTD Message: Active High. Indicates that the core received an Assert_INTD Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_deassert_inta | Output | Received Deassert_INTA Message: Active High. Indicates that the core received a Deassert_INTA Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_deassert_intb | Output | Received Deassert_INTB Message: Active High. Indicates that the core received a Deassert_INTB Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_deassert_intc | Output | Received Deassert_INTC Message: Active High. Indicates that the core received a Deassert_INTC Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |
| cfg_msg_received_deassert_intd | Output | Received Deassert_INTD Message: Active High. Indicates that the core received a Deassert_INTD Message. Valid only along with assertion of cfg_msg_received. The Requester ID of this Message Transaction is provided on cfg_msg_data[15:0]. |

# Interrupt Interface Signals

Table 2-20 defines the Interrupt Interface signals.

*Table 2-20:*   **Configuration Interface Signals: Interrupt Interface - Endpoint Only**

| Name | Direction | Description |
|---|---|---|
| cfg_interrupt_n | Input | Configuration Interrupt: Active-Low interrupt-request signal. The User Application can assert this input to cause the selected interrupt message type to be transmitted by the core. The signal should be held Low until cfg_interrupt_rdy_n is asserted. |
| cfg_interrupt_rdy_n | Output | Configuration Interrupt Ready: Active-Low interrupt grant signal. The simultaneous assertion of cfg_interrupt_rdy_n and cfg_interrupt_n indicates that the core has successfully transmitted the requested interrupt message. |
| cfg_interrupt_assert_n | Input | Configuration Legacy Interrupt Assert/Deassert Select: Selects between Assert and Deassert messages for Legacy interrupts when cfg_interrupt_n is asserted. Not used for MSI interrupts. <br> **Value    Message Type** <br> 0         Assert <br> 1         Deassert |
| cfg_interrupt_di[7:0] | Input | Configuration Interrupt Data In: For Message Signaling Interrupts (MSI), the portion of the Message Data that the Endpoint must drive to indicate MSI vector number, if Multi-Vector Interrupts are enabled. The value indicated by cfg_interrupt_mmenable[2:0] determines the number of lower-order bits of Message Data that the Endpoint provides; the remaining upper bits of cfg_interrupt_di[7:0] are not used. For Single-Vector Interrupts, cfg_interrupt_di[7:0] is not used. For Legacy Interrupt messages (Assert_INTx, Deassert_INTx), only INTA (00h) is supported. |
| cfg_interrupt_do[7:0] | Output | Configuration Interrupt Data Out: The value of the lowest eight bits of the Message Data field in the Endpoint's MSI capability structure. This value is provided for informational purposes and backwards compatibility. |
| cfg_interrupt_mmenable[2:0] | Output | Configuration Interrupt Multiple Message Enable: This is the value of the Multiple Message Enable field and defines the number of vectors the system allows for multi-vector MSI. Values range from 000b to 101b. A value of 000b indicates that single-vector MSI is enabled, while other values indicate the number of lower-order bits that can be overridden by cfg_interrupt_di[7:0]. <br> • 000, 0 bits <br> • 001, 1 bit <br> • 010, 2 bits <br> • 011, 3 bits <br> • 100, 4 bits <br> • 101, 5 bits |
| cfg_interrupt_msienable | Output | Configuration Interrupt MSI Enabled: Indicates that Message Signalling Interrupt (MSI) messaging is enabled. <br> • 0: Only Legacy (INTX) interrupts or MSI-X Interrupts can be sent. <br> • 1: Only MSI Interrupts should be sent. |

*Table 2-20:* **Configuration Interface Signals: Interrupt Interface - Endpoint Only** *(Cont'd)*

| Name | Direction | Description |
|------|-----------|-------------|
| cfg_interrupt_msixenable | Output | Configuration Interrupt MSI-X Enabled: Indicates that the Message Signalling Interrupt-X (MSI-X) messaging is enabled.<br>• `0`: Only Legacy (INTX) interrupts or MSI Interrupts can be sent.<br>• `1`: Only MSI-X Interrupts should be sent. |
| cfg_interrupt_msixfm | Output | Configuration Interrupt MSI-X Function Mask: Indicates the state of the Function Mask bit in the MSI-X Message Control field. If `0`, each vector's Mask bit determines its masking. If `1`, all vectors are masked, regardless of their per-vector Mask bit states. |

# Error Reporting Signals

Table 2-21 defines the User Application error-reporting signals.

*Table 2-21:* **User Application Error-Reporting Signals**

| Port Name | Direction | Description |
|-----------|-----------|-------------|
| cfg_err_ecrc_n | Input | ECRC Error Report: Active Low. The user can assert this signal to report an ECRC error (end-to-end CRC). |
| cfg_err_ur_n | Input | Configuration Error Unsupported Request: Active Low. The user can assert this signal to report that an unsupported request was received. This signal is ignored if cfg_err_cpl_rdy_n is deasserted. |
| cfg_err_cpl_timeout_n[1] | Input | Configuration Error Completion Time-out: Active Low. The user can assert this signal to report a completion timed out. |
| cfg_err_cpl_unexpect_n | Input | Configuration Error Completion Unexpected: Active Low. The user can assert this signal to report that an unexpected completion was received. |
| cfg_err_cpl_abort_n | Input | Configuration Error Completion Aborted: Active Low. The user can assert this signal to report that a completion was aborted. This signal is ignored if cfg_err_cpl_rdy_n is deasserted. |
| cfg_err_posted_n | Input | Configuration Error Posted: Active Low. This signal is used to further qualify any of the cfg_err_* input signals. When this input is asserted concurrently with one of the other signals, it indicates that the transaction that caused the error was a posted transaction. |
| cfg_err_cor_n[1] | Input | Configuration Error Correctable Error: Active Low. The user can assert this signal to report that a correctable error was detected. |

*Table 2-21:* **User Application Error-Reporting Signals** *(Cont'd)*

| Port Name | Direction | Description |
|---|---|---|
| cfg_err_tlp_cpl_header[47:0] | Input | Configuration Error TLP Completion Header: Accepts the header information from the user when an error is signaled. This information is required so that the core can issue a correct completion, if required. <br><br> This information should be extracted from the received error TLP and presented in the given format: <br><br> [47:41]    Lower Address <br> [40:29]    Byte Count <br> [28:26]    TC <br> [25:24]    Attr <br> [23:8]     Requester ID <br> [7:0]      Tag |
| cfg_err_cpl_rdy_n | Output | Configuration Error Completion Ready: Active Low. When asserted, this signal indicates that the core can accept assertions on cfg_err_ur_n and cfg_err_cpl_abort_n for Non-Posted Transactions. Assertions on cfg_err_ur_n and cfg_err_cpl_abort_n are ignored when cfg_err_cpl_rdy_n is deasserted. |
| cfg_err_locked_n | Input | Configuration Error Locked: Active Low. This signal is used to further qualify any of the cfg_err_* input signals. When this input is asserted concurrently with one of the other signals, it indicates that the transaction that caused the error was a locked transaction. <br><br> This signal is for use in Legacy mode. If the user needs to signal an unsupported request or an aborted completion for a locked transaction, this signal can be used to return a Completion Locked with UR or CA status. <br><br> **Note**: When not in Legacy mode, the core automatically returns a Completion Locked, if appropriate. |

**Notes:**

1. The user should assert these signals only if the device power state is D0. Asserting these signals in non-D0 device power states might result in an incorrect operation on the PCIe link. For additional information, see the *PCI Express Base Specification, rev. 2.0,* Section 5.3.1.2.

# Dynamic Reconfiguration Port Interface

The Dynamic Reconfiguration Port (DRP) interface allows for the dynamic change of FPGA configuration memory bits of the Virtex-6 FPGA Integrated Block for PCI Express core. These configuration bits are represented as attributes for the PCIE_2_0 library primitive, which is instantiated as part of this core. Table 2-22 defines the DRP interface signals. For detailed usage information, see Using the Dynamic Reconfiguration Port Interface, page 187.

*Table 2-22:* **Dynamic Reconfiguration Port Interface Signals**

| Name | Direction | Description |
|---|---|---|
| pcie_drp_clk | Input | PCI Express DRP Clock: The rising edge of this signal is the timing reference for all the other DRP signals. Normally, drp_clk is driven with a global clock buffer. The maximum frequency is defined in the *Virtex-6 FPGA Data Sheet*. |
| pcie_drp_den | Input | PCI Express DRP Data Enable: When asserted, this signal enables a read or write operation. If drp_dwe is deasserted, it is a read operation, otherwise a write operation. For any given drp_clk cycle, all other input signals are don't cares if drp_den is not active. |
| pcie_drp_dwe | Input | PCI Express DRP Write Enable: When asserted, this signal enables a write operation to the port (see drp_den). |
| pcie_drp_daddr[8:0] | Input | PCI Express DRP Address Bus: The value on this bus specifies the individual cell that is written or read. The address is presented in the cycle that drp_den is active. |
| pcie_drp_di[15:0] | Input | PCI Express DRP Data Input: The value on this bus is the data written to the addressed cell. The data is presented in the cycle that drp_den and drp_dwe are active, and is captured in a register at the end of that cycle, but the actual write occurs at an unspecified time before drp_drdy is returned. |
| pcie_drp_drdy | Output | PCI Express DRP Ready: This signal is a response to drp_den to indicate that the DRP cycle is complete and another DRP cycle can be initiated. In the case of a DRP read, the drp_do bus must be captured on the rising edge of drp_clk in the cycle that drp_drdy is active. The earliest that drp_den can go active to start the next port cycle is the same clock cycle that drp_drdy is active. |
| pcie_drp_do[15:0] | Output | PCI Express DRP Data Out: If drp_dwe was inactive when drp_den was activated, the value on this bus when drp_drdy goes active is the data read from the addressed cell. At all other times, the value on drp_do[15:0] is undefined. |

*Chapter 3*

# *Licensing the Core*

In ISE® v11.3 software and later, it is not necessary to install a license key for full access to the Virtex®-6 FPGA Integrated Block for PCI Express® core. The user can skip the license installation instructions in this chapter.

However, when ISE v11.2 software or older is used, the instructions in this chapter need to be followed for obtaining a license key before using the core in the design.

The Virtex-6 FPGA Integrated Block for PCI Express core is provided under the terms of the Xilinx End User Agreement.

## Before Beginning

This chapter assumes that the core has been installed by running Xilinx Update using either the CORE Generator™ IP Software Update installer, or by performing a manual installation after downloading the core from the web.

## License Options

If ISE v11.3 software or later is used, this section can be skipped.

The Virtex-6 FPGA Integrated Block for PCI Express core requires installation of a full license key and the relevant ISE software update. The full license key provides full access to all core functionality both in simulation and in hardware, including:

• Functional simulation support
• Full implementation support including place and route and bitstream generation
• Full functionality in the programmed device with no time-outs

## Obtaining the Full License Key

To obtain a full license key for versions of this core released with ISE v11.2 software or earlier, follow these instructions (in ISE v11.3 software and later, there is no longer a requirement to install a license key):

1.  Navigate to the product page for this core:

    www.xilinx.com/products/ipcenter/V6_PCI_Express_Block.htm

2.  Click **Get License**.

3.  Follow the instructions to install the required Xilinx® ISE software update, and generate the required license key on the Xilinx Product Download and Licensing Site, www.xilinx.com/getproduct.

# *Getting Started Example Design*

This chapter provides an overview of the Virtex®-6 FPGA Integrated Block for PCI Express® example design and instructions for generating the core. It also includes information about simulating and implementing the example design using the provided demonstration test bench.

## Integrated Block Endpoint Configuration Overview

The example simulation design for the Endpoint configuration of the integrated block consists of two discrete parts:

- The Root Port Model, a test bench that generates, consumes, and checks PCI Express bus traffic.
- The Programmed Input/Output (PIO) example design, a completer application for PCI Express. The PIO example design responds to Read and Write requests to its memory space and can be synthesized for testing in hardware.

### Simulation Design Overview

For the simulation design, transactions are sent from the Root Port Model to the Integrated Block core (configured as an Endpoint) and processed by the PIO example design. Figure 4-1 illustrates the simulation design provided with the Integrated Block core. For more information about the Root Port Model, see Root Port Model Test Bench for Endpoint in Appendix A.
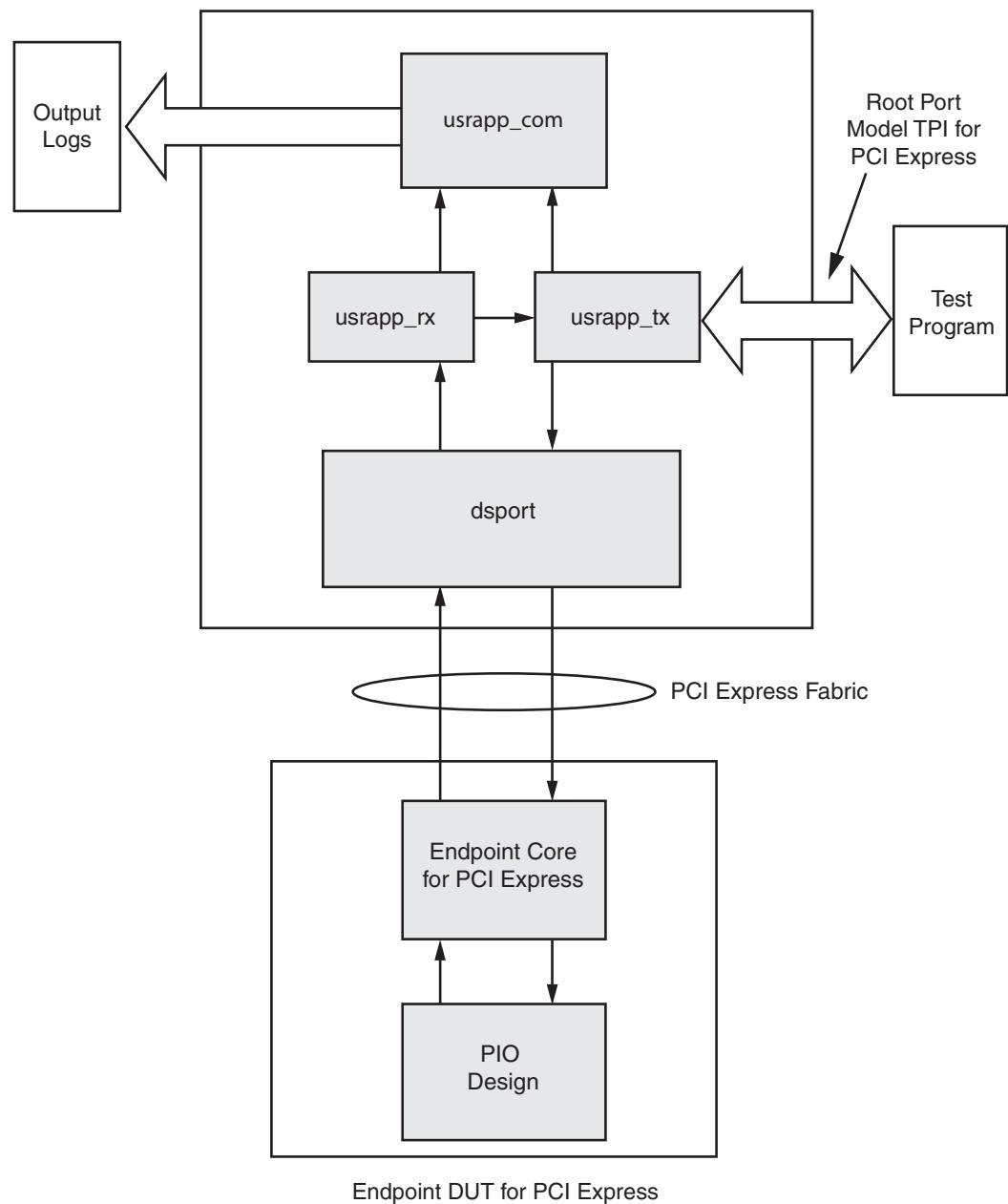
*Figure 4-1:*  **Simulation Example Design Block Diagram**

## Implementation Design Overview

The implementation design consists of a simple PIO example that can accept read and write transactions and respond to requests, as illustrated in Figure 4-2. Source code for the example is provided with the core. For more information about the PIO example design, see Appendix A, Example Design and Model Test Bench for Endpoint Configuration.
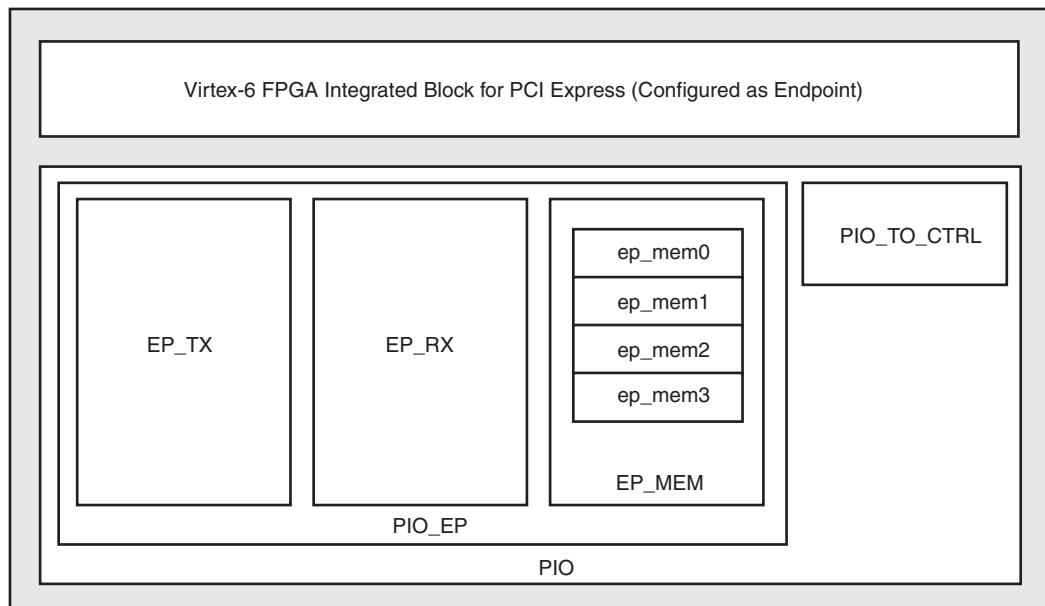


*Figure 4-2:*   **Implementation Example Design Block Diagram**

## Example Design Elements

The PIO example design elements include:

•   Core wrapper

•   An example Verilog HDL or VHDL wrapper (instantiates the cores and example design)

•   A customizable demonstration test bench to simulate the example design

The example design has been tested and verified with Xilinx® ISE® v12.3 software and these simulators:

•   Synopsys VCS and VCS MX 2009.12 and later

•   Mentor Graphics ModelSim v6.5c and later

•   ISE Simulator (ISim)

•   Cadence INCISIV v9.2 and later

# Integrated Block Root Port Configuration Overview

The example simulation design for the Integrated Block Root Port configuration consists of two discrete parts:
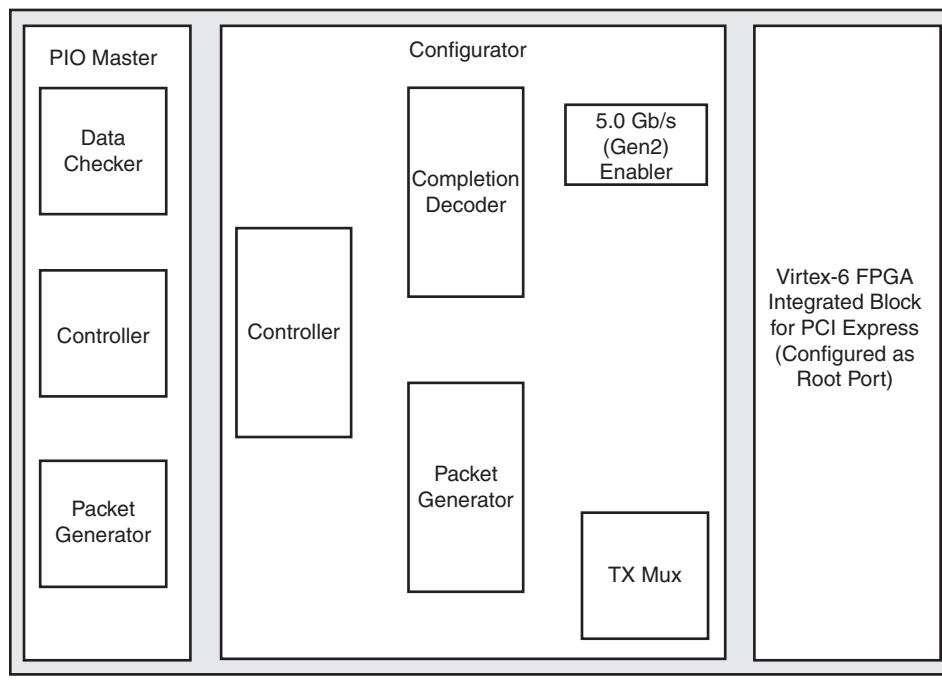
•   The Configurator example design which consists of a Configurator Engine and a PIO Master Example Design.

- The PIO Endpoint Model test bench, which consists of a PCI Express Endpoint, a PIO Slave design, and the test bench that monitors the bus traffic.

## Simulating the Configurator Example Design

The simulation environment for the Root Port example design includes a simple test bench. It also includes a fully functional Endpoint design with the Endpoint PIO Slave example design, which is downstream and is configured by the Root Port example design.

The test bench instantiates the Root Port and Configurator Example design and an example Endpoint with PIO Slave example design. The Root Port and Endpoint communicate with each other, and the test bench monitors progress. The overall block diagram is shown in Figure 4-3. The `board` file is the test bench. The `xilinx_pcie_2_0_rport_v6` file is the Configurator Example Design plus the Integrated Root Port, and the `ep_top` file is the Endpoint example design.



UG517_c4_03_021210

*Figure 4-3:* **Configurator Example Design Components**

To run a simulation with Mentor Graphics ModelSim or Questa, go to the `simulation/functional` directory and type **vsim simulate_mti.do** to run the simulation and bring up a wave window to show pertinent signals. Messages will appear in the log showing progress.

### Simulation Design Overview

For the simulation design, the PIO Master example design directs the Configurator example design to bring up the link partner at the appropriate time, and then generates and consumes bus traffic.

The Configurator example design, when instructed to, performs the Configuration transactions required to "bring-up" the connected PCI Express Endpoint and allows higher level interactions to occur.

Figure 4-4 illustrates the simulation design provided with the Root Port of the Virtex-6 FPGA Integrated Block for PCI Express. For more information about the Configurator example design and the Endpoint model test bench, see Appendix B, Example Design and Model Test Bench for Root Port Configuration.
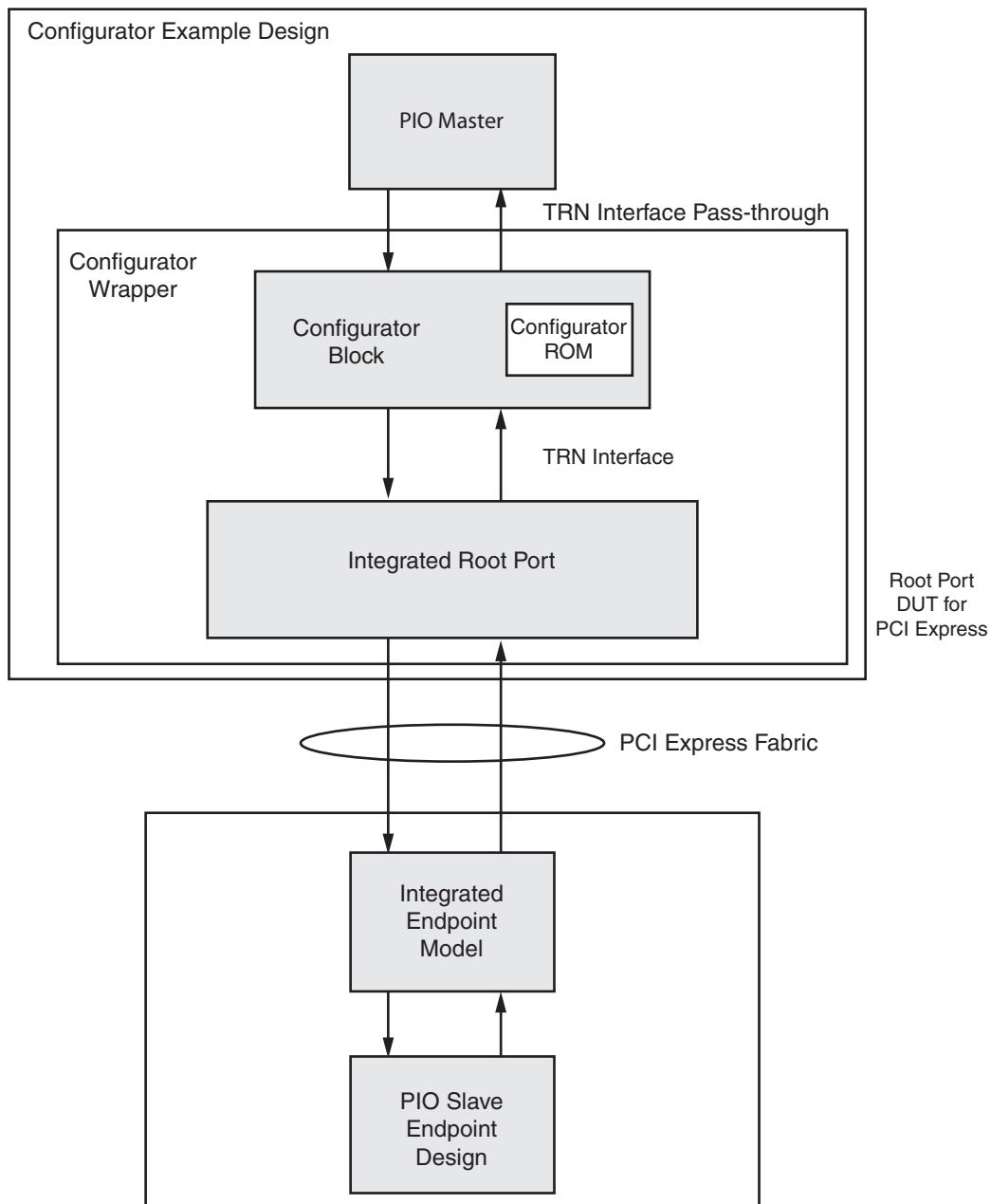


*Figure 4-4:*   **Simulation Example Design Block Diagram**

## Implementation Design Overview

The implementation design consists of a PIO Master example that can initiate configuration and enumeration of a downstream device and generate Reads and Writes and consume completions. It also consists of the Configurator example that performs the configuration tasks required to enumerate the downstream device. This design is

illustrated in Figure 4-3. Source code for the example is provided with the core. For more information about the example design, see Appendix B, Example Design and Model Test Bench for Root Port Configuration.

## Example Design Elements

The Configurator example design elements include:

- Core wrapper
- An example Verilog HDL or VHDL wrapper (instantiates the cores and example design)
- A customizable demonstration test bench to simulate the example design

# Generating the Core

To generate a core using the default values in the CORE Generator software Graphical User Interface (GUI), follow these steps:

1. Start the CORE Generator tool.

   For help starting and using the CORE Generator tool, see the *Xilinx CORE Generator Guide*, available from the ISE software documentation web page.

2. Choose **File** > **New Project**.

3. Enter a project name and location, then click **OK**. This example uses `project_name.cpg` and `project_dir`. The Project Options dialog box appears.
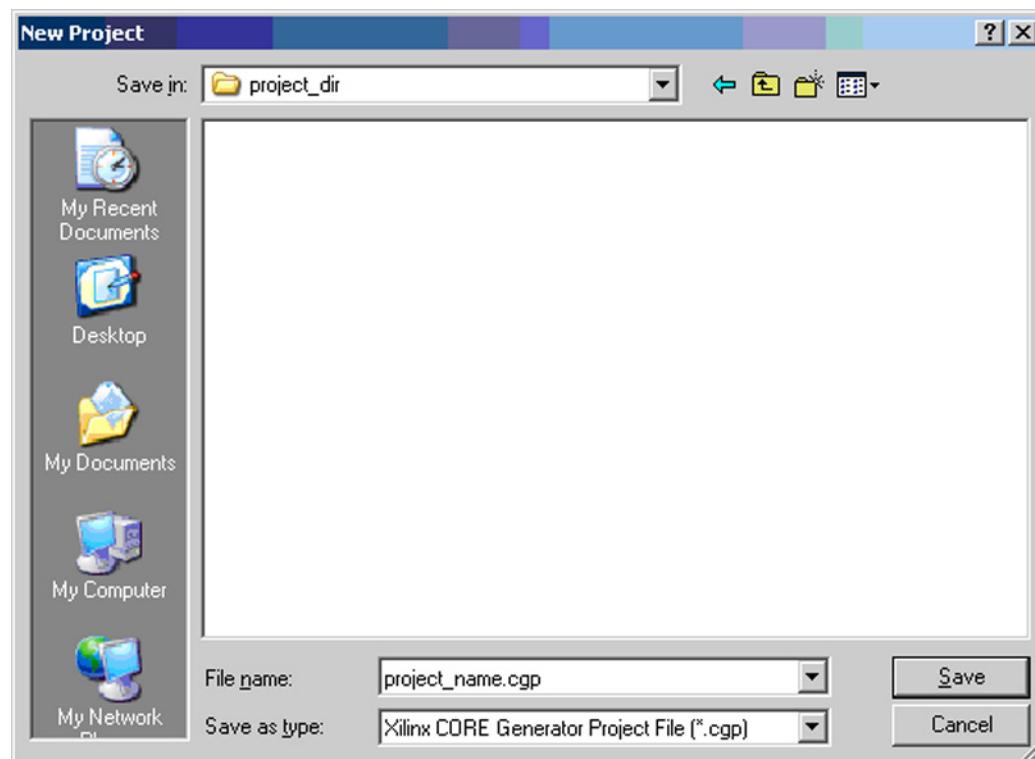


*Figure 4-5:* **New Project Dialog Box**
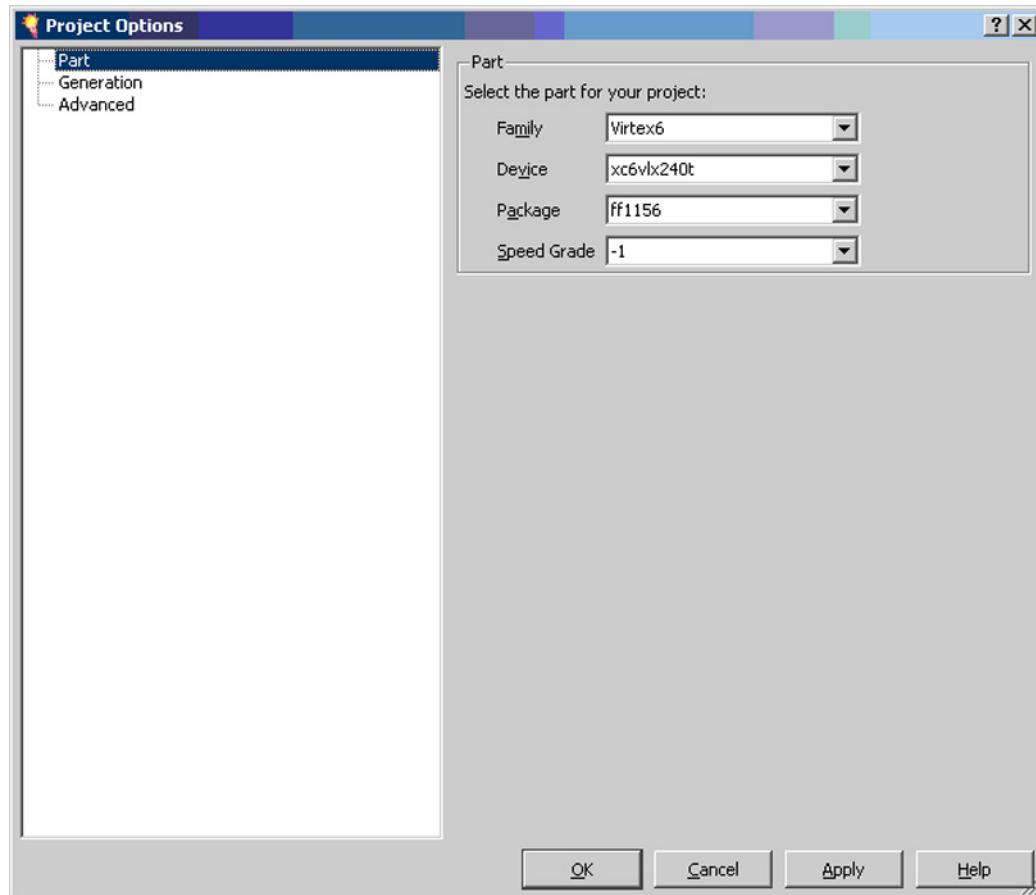
*Figure 4-6:* **Project Options**

4. Set the project options:

   From the Part tab, select these options:

   - **Family**: Virtex6
   - **Device**: xc6vlx240t
   - **Package**: ff1156
   - **Speed Grade**: -1

   ***Note:*** If an unsupported silicon device is selected, the core is dimmed (unavailable) in the list of cores.

   From the Generation tab, select these parameters, and then click **OK**.

   - **Design Entry**: Select **Verilog** or **VHDL**
   - **Vendor**: Select **ISE** (for XST) or **Synplicity** (if required license present).

   ***Note:*** Selecting Synplicity generates a sample Synplicity project file only for the top-level example design. The underlying PCI Express core is still synthesized using XST.

5. Locate the core in the selection tree under Standard Bus Interfaces/PCI Express; then double-click the core name to display the Integrated Block main screen.

6. In the Component Name field, enter a name for the core. `<component_name>` is used in this example.



*Figure 4-7:* **Integrated Block Main Screen**

7. From the Device/Port Type drop-down menu, select the appropriate device/port type of the core (Endpoint or Root Port).

8. Click **Finish** to generate the core using the default parameters. The core and its supporting files, including the example design and model test bench, are generated in the project directory. For detailed information about the example design files and directories, see Directory Structure and File Contents, page 64. In addition, see the README file.

# Simulating the Example Design

The example design provides a quick way to simulate and observe the behavior of the core.

## Endpoint Configuration

The simulation environment provided with the Virtex-6 FPGA Integrated Block for PCI Express core in Endpoint configuration performs simple memory access tests on the PIO example design. Transactions are generated by the Root Port Model and responded to by the PIO example design.

- PCI Express Transaction Layer Packets (TLPs) are generated by the test bench transmit User Application (`pci_exp_usrapp_tx`). As it transmits TLPs, it also generates a log file, `tx.dat`.

- PCI Express TLPs are received by the test bench receive User Application (`pci_exp_usrapp_rx`). As the User Application receives the TLPs, it generates a log file, `rx.dat`.

For more information about the test bench, see Root Port Model Test Bench for Endpoint in Appendix A.

## Root Port Configuration

The simulation environment provided with the Virtex-6 FPGA Integrated Block for PCI Express core in Root Port configuration performs simple initiation tasks to configure the Endpoint model and performs simple memory access tests on the PIO Endpoint model. The transactions are generated by the PIO Master example design and responded to by the PIO slave Endpoint model.

- PCI Express Transaction Layer Packets (TLPs) are generated by the Packet Generator in the PIO Master example design and are received by the PIO Slave Receive engine.

- The PIO Slave controller generates the response PCIe TLPs, which are in turn received and reviewed by the data checker in the PIO Master example design.

For more information about the test bench, see Endpoint Model Test Bench for Root Port in Appendix B.

## Setting up for Simulation

To run the gate-level simulation, the Xilinx Simulation Libraries must be compiled for the user system. See the Compiling Xilinx Simulation Libraries (COMPXLIB) in the *Xilinx ISE Synthesis and Verification Design Guide* and the *Xilinx ISE Software Manuals and Help*. Documents can be downloaded from www.xilinx.com/support/software_manuals.htm.

### Simulator Requirements

Virtex-6 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. This core supports these simulators:

- ModelSim: v6.4b

- Cadence INCISIV: v9.2 (Verilog only)

- Synopsys VCS and VCS MX: 2009.12 (Verilog only)

- ISE Simulator (ISim)

## Running the Simulation

The simulation scripts provided with the example design support pre-implementation (RTL) simulation. The existing test bench can be used to simulate with a post-implementation version of the example design.

The pre-implementation simulation consists of these components:

- Verilog or VHDL model of the test bench
- Verilog or VHDL RTL example design
- The Verilog or VHDL model of the Virtex-6 FPGA Integrated Block for PCI Express

1. To run the simulation, go to this directory:

   ```
   <project_dir>/<component_name>/simulation/functional
   ```

2. Launch the simulator and run the script that corresponds to the user simulation tool using one of the following:

   - **VCS** > `./simulate_vcs.sh`
   - **INCISIV** > `./simulate_ncsim.sh`
   - **ModelSim** > `do simulate_mti.do`
   - **ISim** (UNIX) > `./simulate_isim.sh`
   - **ISim** (Windows) > `simulate_isim.bat`

# Implementing the Example Design

After generating the core, the netlists and the example design can be processed using the Xilinx implementation tools. The generated output files include scripts to assist in running the Xilinx software.

To implement the example design:

Open a command prompt or terminal window and type the following:

- Windows

   ```
   ms-dos> cd <project_dir>\<component_name>\implement
   ms-dos> implement.bat
   ```

- Linux

   ```
   % cd <project_dir>/<component_name>/implement
   % ./implement.sh
   ```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the example design, and then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the `results` directory and execute these processes:

1. Removes data files from the previous runs.

2. Synthesizes the example design using XST or Synplicity based on the flow settings in the Project Options window.

3. `ngdbuild`: Builds a Xilinx design database for the example design.

   - Inputs:

      **Part-Package-Speed Grade selection**:

      XC6VLX240T-FF1156-1

      **Example design UCF**:

```
xilinx_pcie_2_0_ep_v6_01_lane_gen1_ML605.ucf
```

4.  `map`: Maps design to the selected FPGA using the constraints provided.

5.  `par`: Places cells onto FPGA resources and routes connectivity.

6.  `trce`: Performs static timing analysis on design using constraints specified.

7.  `netgen`: Generates a logical Verilog or VHDL HDL representation of the design and an SDF file for post-layout verification.

8.  `bitgen`: Generates a bitstream file for programming the FPGA.

These FPGA implementation related files are generated in the `results` directory:

- `routed.bit`
  FPGA configuration information.

- `routed.v[hd]`
  Verilog or VHDL functional Model.

- `routed.sdf`
  Timing model Standard Delay File.

- `mapped.mrp`
  Xilinx map report.

- `routed.par`
  Xilinx place and route report.

- `routed.twr`
  Xilinx timing analysis report.

The script file starts from an EDIF/NGC file and results in a bitstream file. It is possible to use the Xilinx ISE software GUI to implement the example design. However, the GUI flow is not presented in this document.

# Directory Structure and File Contents

The Virtex-6 FPGA Integrated Block for PCI Express example design directories and their associated files are defined in the sections that follow. Click a directory name to go to the desired directory and its associated files.

## Example Design

📁 **<project directory>**
Top-level project directory; name is user-defined

📁 <project directory>/<component name>
Core release notes readme file

📁 <component name>/doc
Product documentation

📁 <component name>/example_design
Verilog or VHDL design files

📁 <component name>/implement
Implementation script files

📁 implement/results
Contains implement script results

📁 implement/xst
Contains synthesis results, when XST is chosen as the synthesis tool

📁 implement/synplify
Contains synthesis results, when Synplify is chosen as the synthesis tool

📁 <component name>/source
Core source files

📁 <component name>/simulation
Simulation scripts

📁 simulation/dsport (for Endpoint configuration only)
Root Port Bus Functional Model

📁 simulation/ep (for Root Port configuration only)
Endpoint Functional Model

📁 simulation/functional
Functional simulation files

📁 simulation/tests (for Endpoint configuration only)
Test command files

## <project directory>

The project directory contains all the CORE Generator tool project files.

*Table 4-1:* **Project Directory**

| Name | Description |
|---|---|
| <project_dir> | |
| `<component_name>_flist.txt` | List of files delivered with core. |
| `<component_name>.{veo\|vho}` | Verilog or VHDL instantiation template. |
| `<component_name>_xmdf.tcl` | Xilinx standard IP Core information file used by Xilinx design tools. |

Back to Top

## <project directory>/<component name>

The component name directory contains the release notes in the readme file provided with the core, which can include tool requirements, updates, and issue resolution.

*Table 4-2:* **Component Name Directory**

| Name | Description |
|---|---|
| <project_dir>/<component_name> | |
| `v6_pcie_readme.txt` | Release notes file. |

Back to Top

## <component name>/doc

The doc directory contains the PDF documentation provided with the core.

*Table 4-3:* **Doc Directory**

| Name | Description |
|---|---|
| <project_dir>/<component_name>/doc | |
| `v6_pcie_ug517.pdf` | *Virtex-6 FPGA Integrated Block for PCI Express User Guide* |
| `v6_pcie_ds715.pdf` | *Virtex-6 FPGA Integrated Block for PCI Express Data Sheet* |

Back to Top

## <component name>/example_design

The example_design directory contains the example design files provided with the core. Table 4-4 and Table 4-5 show the directory contents for an Endpoint configuration core and a Root Port configuration core.

*Table 4-4:* **Example Design Directory: Endpoint Configuration**

| Name | Description |
|---|---|
| <project_dir>/<component_name>/example_design | |
| `xilinx_pcie_2_0_ep_v6.v[hd]` | Verilog or VHDL top-level PIO example design file. |
| `pci_exp_x_lane_64b_ep.v` | Virtex-6 FPGA Integrated Block for PCI Express black box for 64-bit designs synthesized with Synplicity. |

*Table 4-4:* **Example Design Directory: Endpoint Configuration** *(Cont'd)*

| Name | Description |
|---|---|
| `pci_exp_x_lane_128b_ep.v` | Virtex-6 FPGA Integrated Block for PCI Express black box for 128-bit designs synthesized with Synplicity. |
| `pcie_v6_app.v[hd]`<br>`EP_MEM.v[hd]`<br>`PIO.v[hd]`<br>`PIO_EP.v[hd]`<br>`PIO_EP_MEM_ACCESS.v[hd]`<br>`PIO_TO_CTRL.v[hd]`<br>`PIO_64.v[hd]`<br>`PIO_64_RX_ENGINE.v[hd]`<br>`PIO_64_TX_ENGINE.v[hd]` | PIO example design files. |

Back to Top

*Table 4-5:* **Example Design Directory: Root Port Configuration**

| Name | Description |
|---|---|
| `<project_dir>/<component_name>/example_design` | |
| `pio_master_checker.v[hd]`<br>`pio_master_controller.v[hd]`<br>`pio_master_pkt_generator.v[hd]`<br>`pio_master.v[hd]` | PIO Master Example Design files. |
| `cgator_wrapper.v[hd]` | Configurator Design wrapper file. |
| `xilinx_pcie_2_0_rport_v6.v[hd]` | Verilog or VHDL top-level Configurator example design file. |
| `xilinx_pcie_2_0_rport_v6_01_lane_gen1_ML605.ucf` | Example Design UCF. File name varies by lane width, link speed, part, package, PCIe block location, and Xilinx Development Board selected. |

Back to Top

## <component name>/implement

The `implement` directory contains the core implementation script files.

*Table 4-6:* **Implement Directory**

| Name | Description |
|---|---|
| `<project_dir>/<component_name>/implement` | |
| `implement.bat`<br>`implement.sh` | DOS and Linux implementation scripts. |
| `synplify.prj` | Synplify synthesis script. |
| `source_files_xst.prj` | XST file list for the core. |
| `xilinx_pcie_2_0_[ep/rport]_v6.cmd` | XST command file. |
| `xilinx_pcie_2_0_[ep/rport]_v6.xcf` | XST synthesis constraints file. |

Back to Top

## implement/results

The `results` directory is created by the implement script. The implement script results are placed in the `results` directory.

*Table 4-7:* **Results Directory**

| Name | Description |
|---|---|
| <project_dir>/<component_name>/implement/results | |
| Implement script result files. | |

**Back to Top**

## implement/xst

The `xst` directory is created by the XST script. The synthesis results are placed in the `xst` directory.

*Table 4-8:* **XST Results Directory**

| Name | Description |
|---|---|
| <project_dir>/<component_name>/implement/xst | |
| XST result files. | |

**Back to Top**

## implement/synplify

The `synplify` directory is created by the Synplify script. The synthesis results are placed in the `synplify` directory.

*Table 4-9:* **Synplify Results Directory**

| Name | Description |
|---|---|
| <project_dir>/<component_name>/implement/synplify | |
| Synplify result files. | |

**Back to Top**

## <component name>/source

The `source` directory contains the generated core source files.

*Table 4-10:* **Source Directory**

| Name | Description |
|---|---|
| `<project_dir>/<component_name>/source` ||
| `<component name>.v[hd]` | Verilog or VHDL top-level solution wrapper for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_reset_delay_v6.v[hd]` | Delay module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_clocking_v6.v[hd]` | Clocking module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_2_0_v6.v[hd]` | Solution Wrapper for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_gtx_v6.v[hd]` | GTX module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `gtx_wrapper_v6.v[hd]` | Solution wrapper for the Virtex-6 FPGA GTX transceivers |
| `gtx_tx_sync_rate_v6.v[hd]` | TX Sync module for the Virtex-6 FPGA GTX transceivers |
| `pcie_bram_top_v6.v[hd]` | Block RAM top-level module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `gtx_rx_valid_filter_v6.v[hd]` | Receive filter module for the Virtex-6 FPGA GTX transceivers |
| `pcie_brams_v6.v[hd]` | Block RAM module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_bram_v6.v[hd]` | Block RAM module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_pipe_v6.v[hd]` | PIPE module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_pipe_lane_v6.v[hd]` | PIPE per lane module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_pipe_misc_v6.v[hd]` | Miscellaneous PIPE module for the Virtex-6 FPGA Integrated Block for PCI Express |
| `pcie_lbw_fix_3333_v6.v[hd]`<br>`pcie_tgt_link_spd_fix_3429_v6.v[hd]`<br>`pcie_upconfig_fix_3451_v6.v[hd]` | Miscellaneous RTL files for the Virtex-6 FPGA Integrated Block for PCI Express. |

Back to Top

## <component name>/simulation

The `simulation` directory contains the simulation source files provided with the core.

### simulation/dsport

The `dsport` directory contains the files for the Root Port model test bench.

*Table 4-11:* **dsport Directory: Endpoint Configuration**

| Name | Description |
|------|-------------|
| <project_dir>/<component_name>/simulation/dsport | |
| pcie_2_0_rport_v6.v[hd]<br>pci_exp_expect_tasks.v<br>pci_exp_usrapp_cfg.v[hd]<br>pci_exp_usrapp_com.v<br>pci_exp_usrapp_pl.v[hd]<br>pci_exp_usrapp_rx.v[hd]<br>pci_exp_usrapp_tx.v[hd]<br>xilinx_pcie_2_0_rport_v6.v[hd]<br>test_interface.vhd | Root Port model files. |

Back to Top

### simulation/ep

The `ep` directory contains the Endpoint model files.

*Table 4-12:* **ep Directory: Root Port Configuration**

| Name | Description |
|------|-------------|
| <project_dir>/<component_name>/simulation/ep | |
| pcie_2_0_ep_v6.v[hd]<br>EP_MEM.v[hd]<br>PIO_64_RX_ENGINE.v[hd]<br>PIO_64_TX_ENGINE.v[hd]<br>PIO_64.v[hd]<br>PIO_EP_MEM_ACCESS.v[hd]<br>PIO_EP.v[hd]<br>PIO_TO_CTRL.v[hd]<br>PIO.v[hd]<br>pcie_app_v6.v[hd]<br>xilinx_pcie_2_0_ep_v6.v[hd] | Endpoint model files. |

Back to Top

### simulation/functional

The functional directory contains functional simulation scripts provided with the core.

*Table 4-13:* **Functional Directory**

| Name | Description |
|------|-------------|
| <project_dir>/<component_name>/simulation/functional | |
| `board.f` | List of files for RTL simulations. |
| `isim_cmd.tcl` | Simulation helper script for ISim. |
| `simulate_isim.bat`<br>`simulate_isim.sh` | Simulation scripts for ISim DOS/UNIX. |
| `wave.wcfg` | Simulation wave file for ISim. |
| `simulate_mti.do` | Simulation script for ModelSim. |
| `simulate_ncsim.sh` | Simulation script for Cadence INCISIV (Verilog only). |
| `simulate_vcs.sh` | Simulation script for VCS (Verilog only). |
| `xilinx_lib_vcs.f` | Points to the required SecureIP Model. |
| `board_common.v`<br>(Endpoint configuration only) | Contains test bench definitions (Verilog only). |
| `board.v[hd]` | Top-level simulation module. |
| `sys_clk_gen_ds.v[hd]`<br>(Endpoint configuration only) | System differential clock source. |
| `sys_clk_gen.v[hd]` | System clock source. |

Back to Top

### simulation/tests

*Note:* This directory exists for Endpoint configuration only.

The `tests` directory contains test definitions for the example test bench.

*Table 4-14:* **Tests Directory**

| Name | Description |
|------|-------------|
| <project_dir>/<component_name>/simulation/tests | |
| `sample_tests1.v`<br>`tests.v[hd]` | Test definitions for example test bench. |

Back to Top

**EXILINX** ®

# Generating and Customizing the Core

The Virtex®-6 FPGA Integrated Block for PCI Express® core is a fully configurable and highly customizable solution. The Virtex-6 FPGA Integrated Block for PCI Express is customized using the CORE Generator™ software.

*Note:* The screen captures in this chapter are conceptual representatives of their subjects and provide general information only. For the latest information, see the CORE Generator tool.

## Customizing the Core using the CORE Generator Software

The CORE Generator software GUI for the Virtex-6 FPGA Integrated Block for PCI Express consists of 12 screens:

- Screen 1: Basic Parameter Settings
- Screen 2: Base Address Registers
- Screen 3: PCI Registers
- Screens 4, 5, and 6: Configuration Register Settings
- Screen 7: Interrupt Capabilities
- Screen 8: Power Management Registers
- Screen 9: PCI Express Extended Capabilities
- Screen 10: Pinout Selection
- Screens 11 and 12: Advanced Settings

## Basic Parameter Settings

The initial customization screen shown in Figure 5-1 is used to define the basic parameters for the core, including the component name, lane width, and link speed.
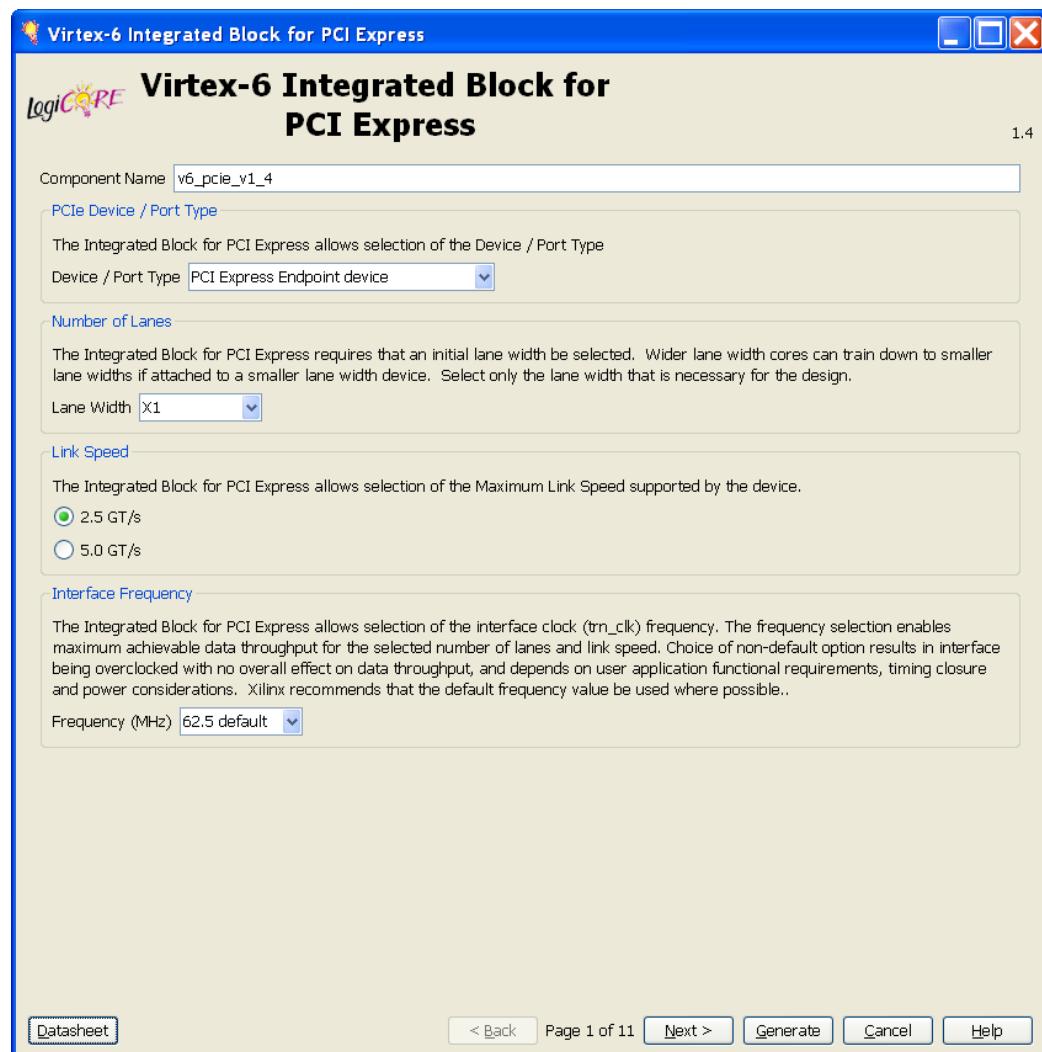


*Figure 5-1:* **Screen 1: Integrated Block for PCI Express Parameters**

### Component Name

Base name of the output files generated for the core. The name must begin with a letter and can be composed of these characters: a to z, 0 to 9, and "_."

### PCIe Device / Port Type

Indicates the PCI Express logical device type.

## Number of Lanes

The Virtex-6 FPGA Integrated Block for PCI Express requires the selection of the initial lane width. Table 5-1 defines the available widths and associated generated core. Wider lane width cores are capable of training down to smaller lane widths if attached to a smaller lane-width device. See Link Training: 2-Lane, 4-Lane, and 8-Lane Components, page 181 for more information.

*Table 5-1:* **Lane Width and Product Generated**

| Lane Width | Product Generated |
|---|---|
| x1 | 1-Lane Virtex-6 FPGA Integrated Block for PCI Express |
| x2 | 2-Lane Virtex-6 FPGA Integrated Block for PCI Express |
| x4 | 4-Lane Virtex-6 FPGA Integrated Block for PCI Express |
| x8 | 8-Lane Virtex-6 FPGA Integrated Block for PCI Express |

## Link Speed

The Virtex-6 FPGA Integrated Block for PCI Express allows the selection of Maximum Link Speed supported by the device. Table 5-2 defines the lane widths and link speeds supported by the device. Higher link speed cores are capable of training to a lower link speed if connected to a lower link speed capable device.

*Table 5-2:* **Lane Width and Link Speed**

| Lane Width | Link Speed |
|---|---|
| x1 | 2.5 Gb/s, 5 Gb/s |
| x2 | 2.5 Gb/s, 5 Gb/s |
| x4 | 2.5 Gb/s, 5 Gb/s |
| x8 | 2.5 Gb/s, 5 Gb/s (Endpoint Only) |

## Interface Frequency

It is possible to select the clock frequency of the core's user interface. Each lane width provides multiple frequency choices: a default frequency and alternative frequencies, as defined in Table 5-3. Where possible, Xilinx recommends using the default frequency. Selecting the alternate frequencies does not result in a difference in throughput in the core, but does allow the user application to run at an alternate speed.

*Table 5-3:* **Recommended and Optional Transaction Clock Frequencies**

| Product | Recommended Frequency (MHz) | Optional Frequency (MHz) |
|---|---|---|
| 1-lane 2.5 Gb/s | 62.5 | 31.25, 125, 250 |
| 1-lane 5.0 Gb/s | 62.5 | 125, 250 |
| 2-lane 2.5 Gb/s | 62.5 | 125, 250 |
| 2-lane 5.0 Gb/s | 125 | 250 |
| 4-lane 2.5 Gb/s | 125 | 250 |

*Table 5-3:* **Recommended and Optional Transaction Clock Frequencies** *(Cont'd)*

| Product | Recommended Frequency (MHz) | Optional Frequency (MHz) |
|---|---|---|
| 4-lane 5.0 Gb/s | 250 | - |
| 8-lane 2.5 Gb/s | 250 | - |
| 8-lane 5.0 Gb/s[1] | 250 | - |

**Notes:**
1. Endpoint configuration only.

## Base Address Registers

The Base Address Register (BAR) screen shown in Figure 5-2 sets the base address register space for the Endpoint configuration. Each Bar (0 through 5) represents a 32-bit parameter.
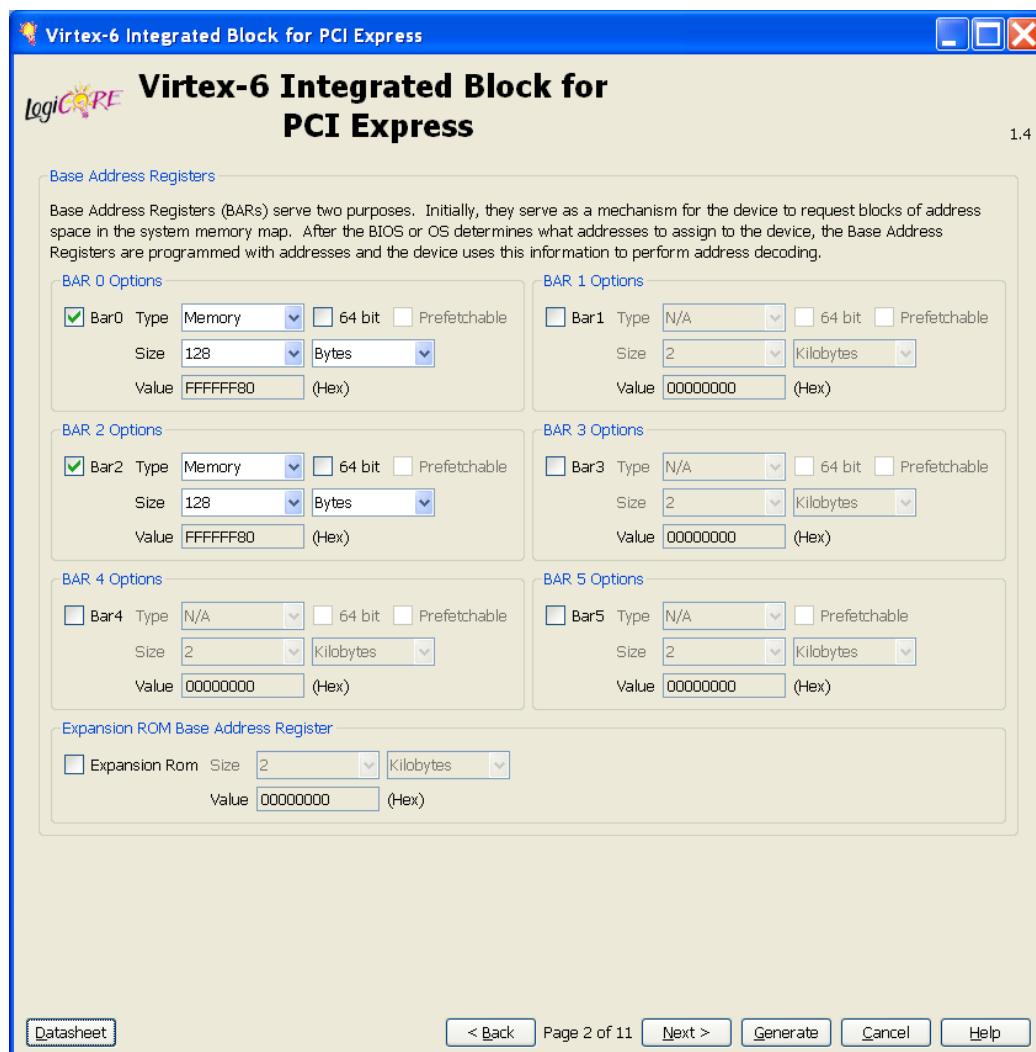


*Figure 5-2:* **Screen 2: BAR Options - Endpoint**

The Base Address Register (BAR) screen shown in Figure 5-3 sets the base address register space and I/O and Prefetchable Memory Base and Limit registers for the Root Port configuration.
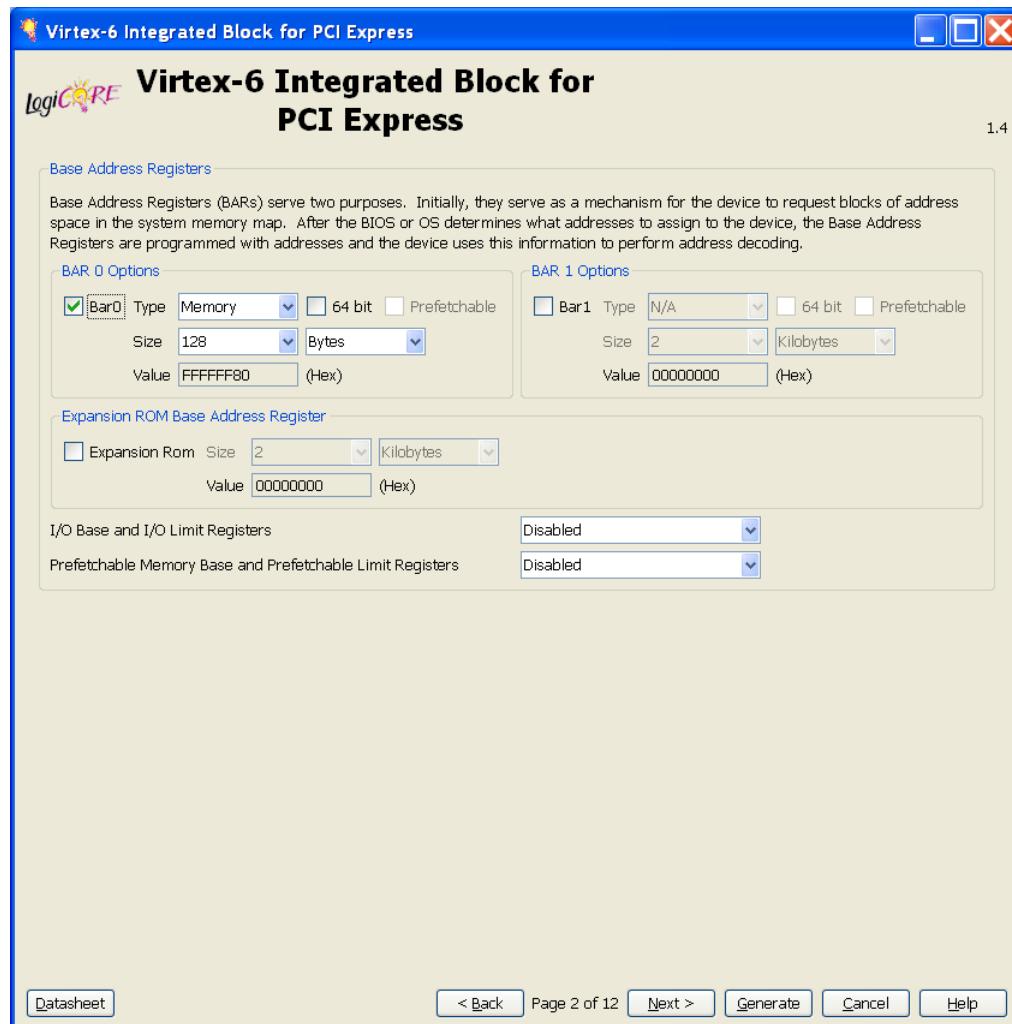


*Figure 5-3:* **Screen 2: BAR Options - Root Port**

## Base Address Register Overview

The Virtex-6 FPGA Integrated Block for PCI Express in Endpoint configuration supports up to six 32-bit Base Address Registers (BARs) or three 64-bit BARs, and the Expansion ROM BAR. The Virtex-6 FPGA Integrated Block for PCI Express in Root Port configuration supports up to two 32-bit BARs or one 64-bit BAR, and the Expansion ROM BAR.

BARs can be one of two sizes:

- **32-bit BARs**: The address space can be as small as 16 bytes or as large as 2 gigabytes. Used for Memory to I/O.

- **64-bit BARs**: The address space can be as small as 128 bytes or as large as 8 exabytes. Used for Memory only.

All BAR registers share these options:

- **Checkbox**: Click the checkbox to enable the BAR; deselect the checkbox to disable the BAR.
- **Type**: BARs can either be I/O or Memory.
  - *I/O*: I/O BARs can only be 32-bit; the Prefetchable option does not apply to I/O BARs. I/O BARs are only enabled for the Legacy PCI Express Endpoint core.
  - *Memory*: Memory BARs can be either 64-bit or 32-bit and can be prefetchable. When a BAR is set as 64 bits, it uses the next BAR for the extended address space and makes the next BAR inaccessible to the user.
- Size: The available Size range depends on the PCIe® Device/Port Type and the Type of BAR selected. Table 5-4 lists the available BAR size ranges.

*Table 5-4:* **BAR Size Ranges for Device Configuration**

| PCIe Device / Port Type | BAR Type | BAR Size Range |
|---|---|---|
| PCI Express Endpoint | 32-bit Memory | 128 Bytes – 2 Gigabytes |
| | 64-bit Memory | 128 Bytes – 8 Exabytes |
| Legacy PCI Express Endpoint | 32-bit Memory | 16 Bytes – 2 Gigabytes |
| | 64-bit Memory | 16 Bytes – 8 Exabytes |
| | I/O | 16 Bytes – 2 Gigabytes |

- **Prefetchable**: Identifies the ability of the memory space to be prefetched.
- **Value**: The value assigned to the BAR based on the current selections.

For more information about managing the Base Address Register settings, see Managing Base Address Register Settings.

### Expansion ROM Base Address Register

If selected, the Expansion ROM is activated and can be a value from 2 KB to 4 GB. According to the *PCI 3.0 Local Bus Specification*, the maximum size for the Expansion ROM BAR should be no larger than 16 MB. Selecting an address space larger than 16 MB might result in a non-compliant core.

### I/O Base and I/O Limit Registers: Root Port Configuration Only

For the Virtex-6 FPGA Integrated Block for PCI Express in the Root Port configuration, the I/O Base and I/O Limit Registers are used to define an address range that can be used by an implemented PCI™ to PCI Bridge to determine how to forward I/O transactions.

### Prefetchable Memory Base and Prefetchable Memory Limit Registers: Root Port Configuration Only

For the Virtex-6 FPGA Integrated Block for PCI Express in the Root Port configuration, the Prefetchable Memory Base and Prefetchable Memory Limit Registers are used to define a prefetchable memory address range that can be used by an implemented PCI-PCI Bridge to determine how to forward Memory transactions.

## Managing Base Address Register Settings

Memory, I/O, Type, and Prefetchable settings are handled by setting the appropriate GUI settings for the desired base address register.

Memory or I/O settings indicate whether the address space is defined as memory or I/O. The base address register only responds to commands that access the specified address space. Generally, memory spaces less than 4 KB in size should be avoided. The minimum I/O space allowed is 16 bytes; use of I/O space should be avoided in all new designs.

Prefetchability is the ability of memory space to be prefetched. A memory space is prefetchable if there are no side effects on reads (that is, data is not destroyed by reading, as from a RAM). Byte write operations can be merged into a single doubleword write, when applicable.

When configuring the core as an Endpoint for PCIe (non-Legacy), 64-bit addressing must be supported for all BARs (except BAR5) that have the prefetchable bit set. 32-bit addressing is permitted for all BARs that do not have the prefetchable bit set. The prefetchable bit related requirement does not apply to a Legacy Endpoint. The minimum memory address range supported by a BAR is 128 bytes for a PCI Express Endpoint and 16 bytes for a Legacy PCI Express Endpoint.

### Disabling Unused Resources

For best results, disable unused base address registers to conserve system resources. A base address register is disabled by deselecting unused BARs in the GUI.

## PCI Registers

The PCI Registers Screen shown in Figure 5-4 is used to customize the IP initial values, class code and Cardbus CIS pointer information.
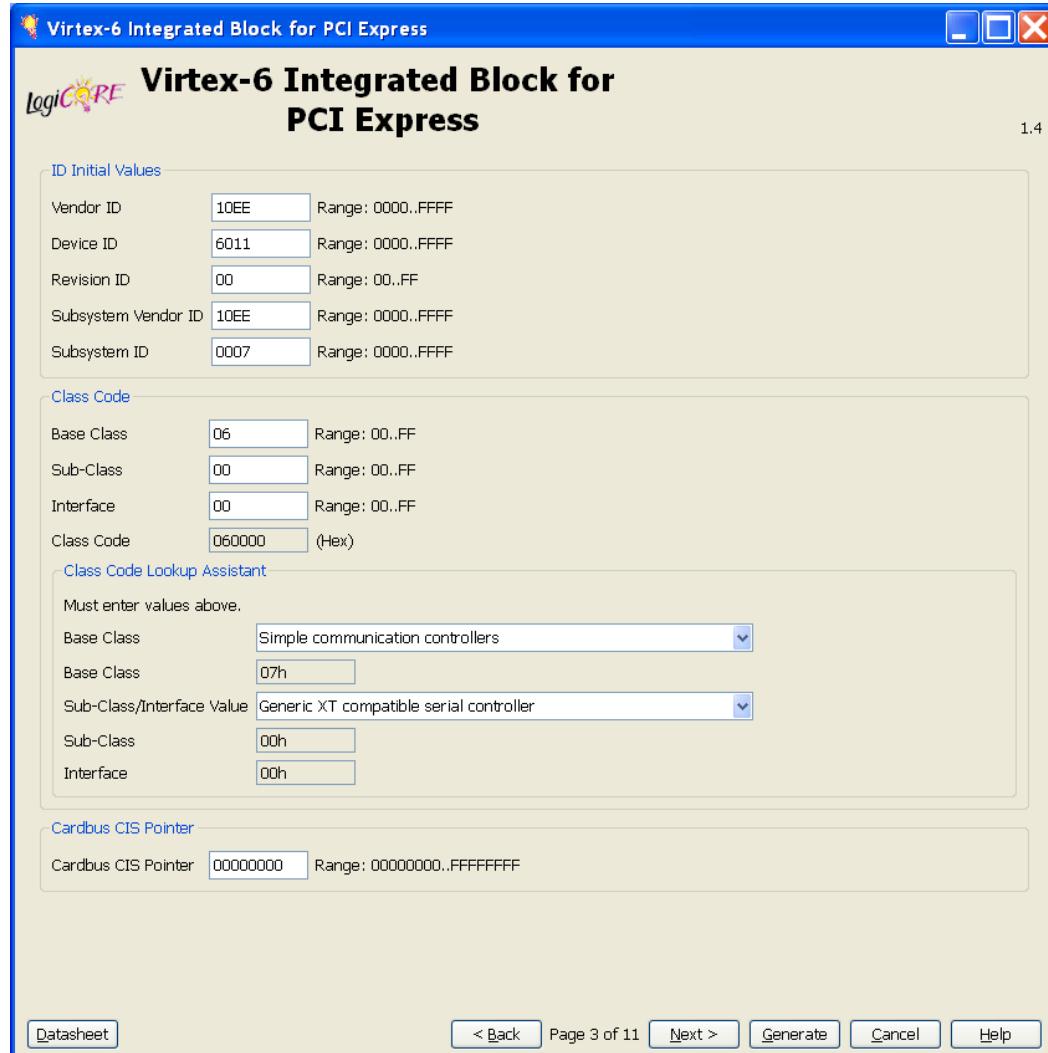


*Figure 5-4:* **PCI Registers: Screen 3**

## ID Initial Values

- **Vendor ID**: Identifies the manufacturer of the device or application. Valid identifiers are assigned by the PCI Special Interest Group to guarantee that each identifier is unique. The default value, `10EEh`, is the Vendor ID for Xilinx. Enter a vendor identification number here. `FFFFh` is reserved.

- **Device ID**: A unique identifier for the application; the default value, which depends on the configuration selected, is 60<*link speed*><*link width*>h. This field can be any value; change this value for the application.

- **Revision ID:** Indicates the revision of the device or application; an extension of the Device ID. The default value is `00h`; enter values appropriate for the application.

- **Subsystem Vendor ID**: Further qualifies the manufacturer of the device or application. Enter a Subsystem Vendor ID here; the default value is `10EE`. Typically, this value is the same as Vendor ID. Setting the value to `0000h` can cause compliance testing issues.

- **Subsystem ID:** Further qualifies the manufacturer of the device or application. This value is typically the same as the Device ID; the default value depends on the lane width and link speed selected. Setting the value to `0000h` can cause compliance testing issues.

## Class Code

The Class Code identifies the general function of a device, and is divided into three byte-size fields:

- **Base Class**: Broadly identifies the type of function performed by the device.

- **Sub-Class**: More specifically identifies the device function.

- **Interface**: Defines a specific register-level programming interface, if any, allowing device-independent software to interface with the device.

Class code encoding can be found at www.pcisig.com.

## Class Code Look-up Assistant

The Class Code Look-up Assistant provides the Base Class, Sub-Class and Interface values for a selected general function of a device. This Look-up Assistant tool only displays the three values for a selected function. The user must enter the values in Class Code for these values to be translated into device settings.

## Cardbus CIS Pointer

Used in cardbus systems and points to the Card Information Structure for the cardbus card. If this field is non-zero, an appropriate Card Information Structure must exist in the correct location. The default value is `0000_0000h`; the value range is `0000_0000h-FFFF_FFFFh`.

## Configuration Register Settings

The Configuration Registers screens shown in Figure 5-5 and Figure 5-6 show the options for the Device Capabilities and Device Capabilities2 Registers, the Block RAM Configuration Options, the Link Capabilities Register, Link Control2 Register and the Link Status Register. The Configuration Register screen shown in Figure 5-7 is displayed for the Root Port configuration only and shows the options for the Root Capabilities Register and the Slot Capabilities Register.



UG517_c5_05_030110
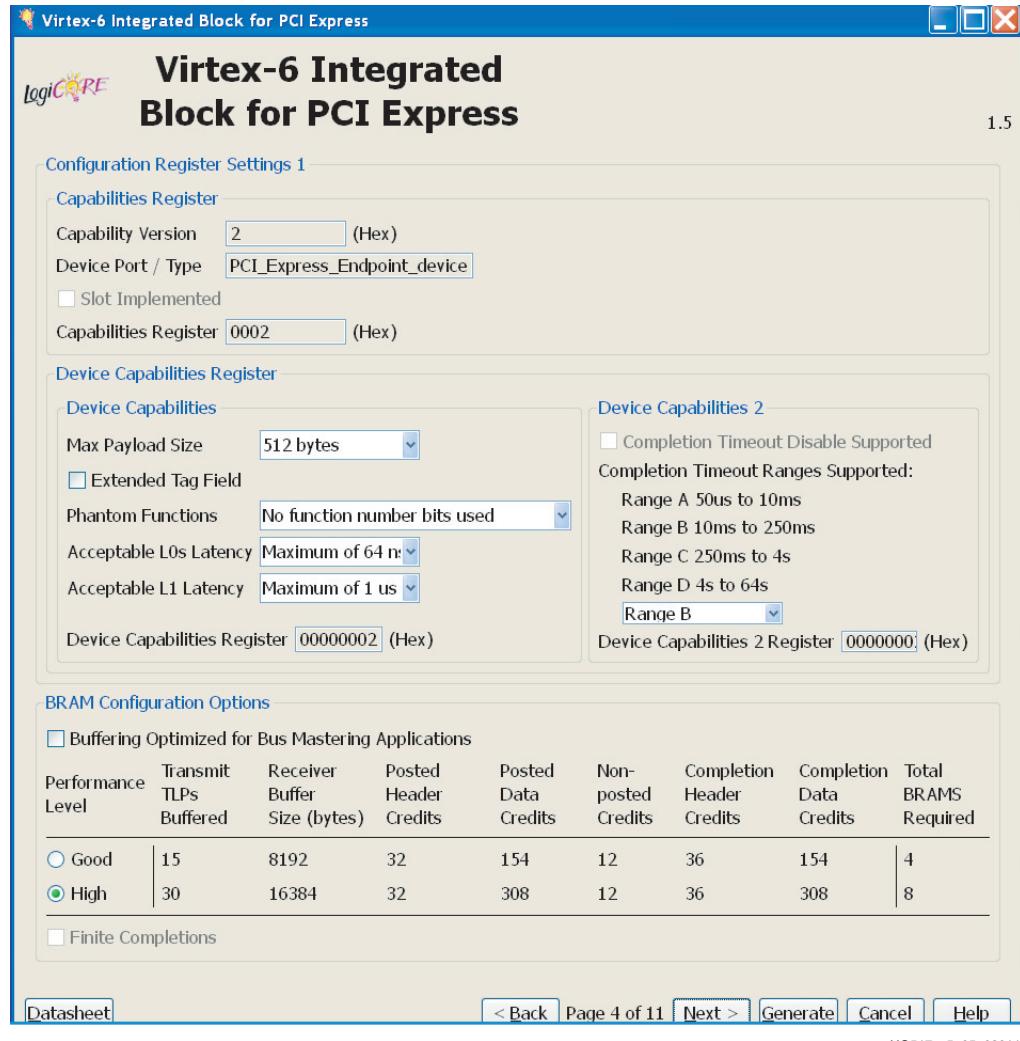
*Figure 5-5:* **Screen 4: Configuration Settings**
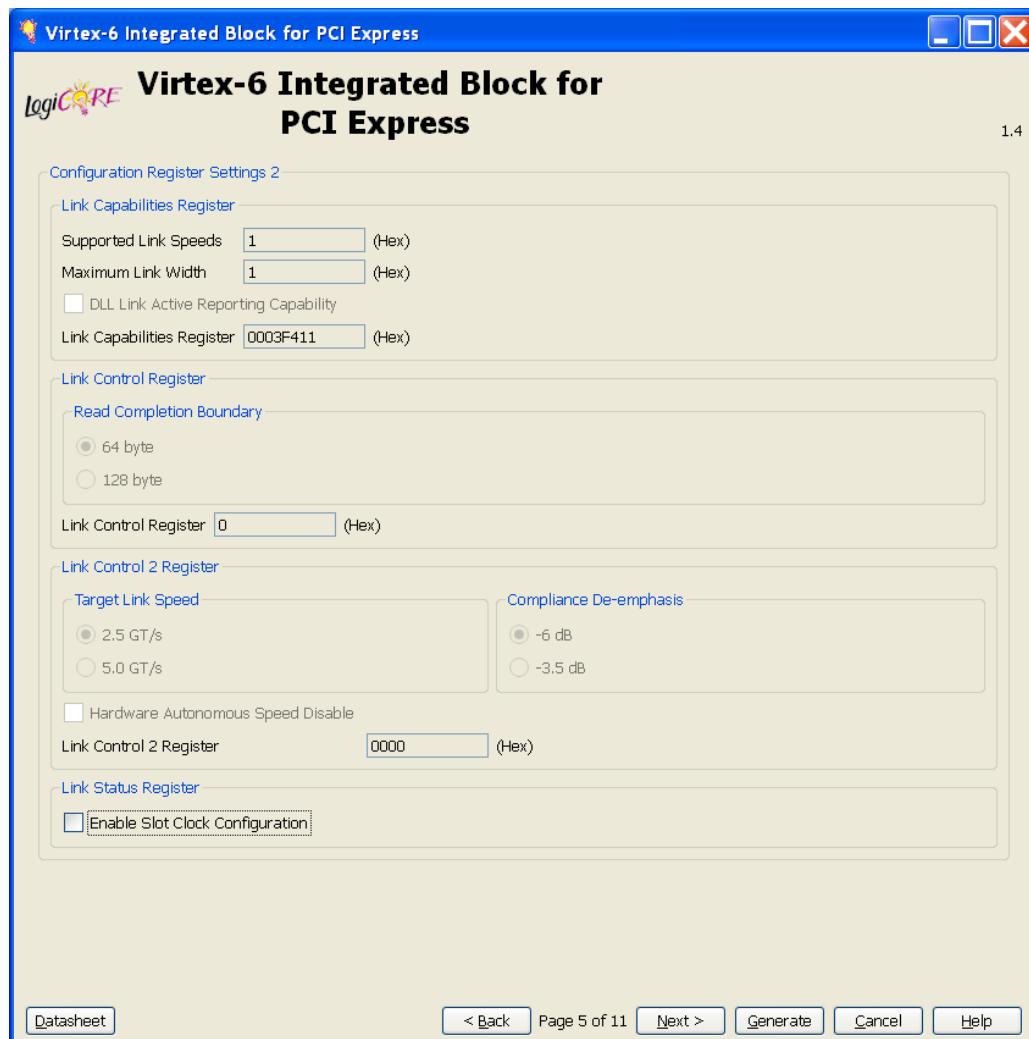
*Figure 5-6:* **Screen 5: Configuration Settings**
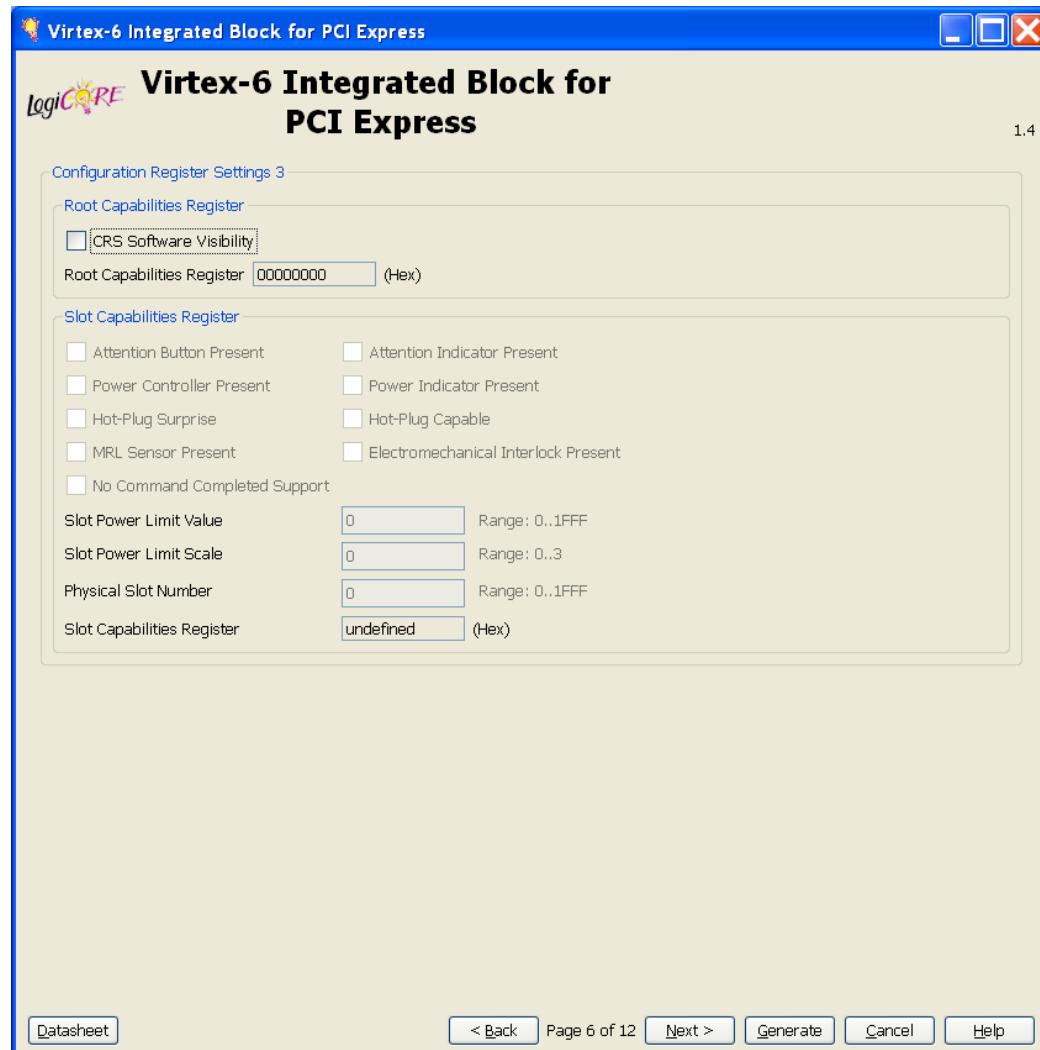
*Figure 5-7:* **Screen 6: Configuration Settings**

## Capabilities Register

- **Capability Version**: Indicates the PCI-SIG defined PCI Express capability structure version number; this value cannot be changed.

- **Device Port Type**: Indicates the PCI Express logical device type.

- **Slot Implemented**: Indicates the PCI Express Link associated with this port is connected to a slot. Only valid for a Root Port of a PCI Express Root Complex or a Downstream Port of a PCI Express Switch.

- **Capabilities Register**: Displays the value of the Capabilities register presented by the integrated block, and is not editable.

### Device Capabilities Register

- **Max Payload Size**: Indicates the maximum payload size that the device/function can support for TLPs.

- **Extended Tag Field**: Indicates the maximum supported size of the Tag field as a Requester. When selected, indicates 8-bit Tag field support. When deselected, indicates 5-bit Tag field support.

- **Phantom Functions**: Indicates the support for use of unclaimed function numbers to extend the number of outstanding transactions allowed by logically combining unclaimed function numbers (called Phantom Functions) with the Tag identifier. See Section 2.2.6.2 of the *PCI Express Base Specification, rev. 2.0* for a description of Tag Extensions. This field indicates the number of most significant bits of the function number portion of Requester ID that are logically combined with the Tag identifier.

- **Acceptable L0s Latency**: Indicates the acceptable total latency that an Endpoint can withstand due to the transition from L0s state to the L0 state.

- **Acceptable L1 Latency**: Indicates the acceptable latency that an Endpoint can withstand due to the transition from L1 state to the L0 state.

- **Device Capabilities Register**: Displays the value of the Device Capabilities register presented by the integrated block and is not editable.

### Device Capabilities 2 Register

- **Completion Timeout Disable Supported**: Indicates support for Completion Timeout Disable mechanism

- **Completion Timeout Ranges Supported**: Indicates Device Function support for the optional Completion Timeout mechanism. It is strongly recommended that the Completion Timeout mechanism not expire in less than 10 ms.

- **Device Capabilities2 Register**: Displays the value of the Device Capabilities2 Register sent to the Core and is not editable.

### Block RAM Configuration Options

- **Buffering Optimized for Bus Mastering Applications**: Causes the device to advertise to its Link Partner credit settings that are optimized for Bus Mastering applications.

- **Performance Level**: Selects the Performance Level settings, which determines the Receiver and Transmitter Sizes. The table displayed specifies the Receiver and Transmitter settings - number of TLPs buffered in the Transmitter, the Receiver Size, the Credits advertised by the Core to the Link Partner and the Number of Block RAMs required for the configuration, corresponding to the Max Payload Size selected, for each of the Performance Level options.

- **Finite Completions**: If selected, causes the device to advertise to the Link Partner the actual amount of space available for Completions in the Receiver. For an Endpoint, this is not compliant to the *PCI Express Base Specification, rev. 2.0*, as endpoints are required to advertise an infinite amount of completion space.

### Link Capabilities Register

This section is used to set the Link Capabilities register.

- **Supported Link Speed**: Indicates the supported link speed of the given PCI Express Link. This value is set to the Link Speed specified in the first GUI screen and is not editable.

- **Maximum Link Width**: This value is set to the initial lane width specified in the first GUI screen and is not editable.
- **DLL Link Active Reporting Capability**: Indicates the optional Capability of reporting the DL_Active state of the Data Link Control and Management State Machine.
- **Link Capabilities Register**: Displays the value of the Link Capabilities register sent to the core and is not editable.

## Link Control Register

- **Read Completion Boundar**y: Indicates the Read Completion Boundary for the Root Port.
- **Link Control Register**: Displays the value of the Link Control Register sent to the core and is not editable.

## Link Control 2 Register

- **Target Link Speed**: Sets an upper limit on the link operational speed. This is used to set the target Compliance Mode speed. The value is set to the supported link speed and can be edited only if the link speed is set to 5.0 Gb/s.
- **Hardware Autonomous Speed Disable**: When checked, this field disables the hardware from changing the link speed for device specific reasons other than attempting to correct unreliable link operation by reducing link speed.
- **De-emphasis**: Sets the level of de-emphasis for an Upstream component, when the Link is operating at 5.0 Gb/s. This feature is not editable.
- **Link Control 2 Register**: Displays the value of the Link Control 2 Register sent to the core and is not editable.

## Link Status Register

- **Enable Slot Clock Configuration**: Indicates that the Endpoint uses the platform-provided physical reference clock available on the connector. Must be cleared if the Endpoint uses an independent reference clock.

## Root Capabilities Register: Root Port Configuration Only

- **CRS Software Visibility**: Indicates that the Root Port is capable of returning Configuration Request Retry Status (CRS) Completion Status to software.
- **Root Capabilities Register**: Displays the value of the Root Capabilities Register sent to the Core and is not editable.

## Slot Capabilities Register: Root Port Configuration Only

- **Attention Button Present**: Indicates that an Attention Button for this slot is implemented on the chassis.
- **Attention Indicator Present**: Indicates that an Attention Indicator for this slot is implemented on the chassis.
- **Power Controller Present**: Indicates that a software programmable Power Controller is implemented for this slot.
- **Power Indicator Present**: Indicates that a Power Indicator for this slot is implemented on the chassis.

- **Hot-Plug Surprise**: Indicates that an adaptor in this slot might be removed from the system without any prior notification.

- **Hot-Plug Capable**: Indicates that this slot is capable of supporting Hot-Plug operations.

- **MRL Sensor Present**: Indicates that an MRL (Manually operated Retention Latch) sensor is implemented for this slot on the chassis.

- **Electromechanical Interlock Present**: Indicates that an Electromechanical Interlock is implemented on the chassis for this slot.

- **No Command Completed Support**: Indicates that the slot does not generate software notification when and issue command is completed by the Hot-Plug Controller.

- **Slot Power Limit Value**: Specifies the Upper Limit on power supplied to the slot, in combination with Slot Power Limit Scale.

- **Slot Power Limit Scale**: Specifies the Scale used for the Slot Power Limit value.

- **Physical Slot Number**: Specifies the Physical Slot Number attached to this Port. This field must be hardware initialized to a value that assigns a slot number that is unique within the chassis, regardless of form factor associated with this slot.

- **Slot Capabilities Register**: Displays the value of the Slot Capabilities Register sent to the Core and is not editable.

## Interrupt Capabilities

The Interrupt Settings screen shown in Figure 5-8 sets the Legacy Interrupt Settings, MSI Capabilities and MSI-X Capabilities.
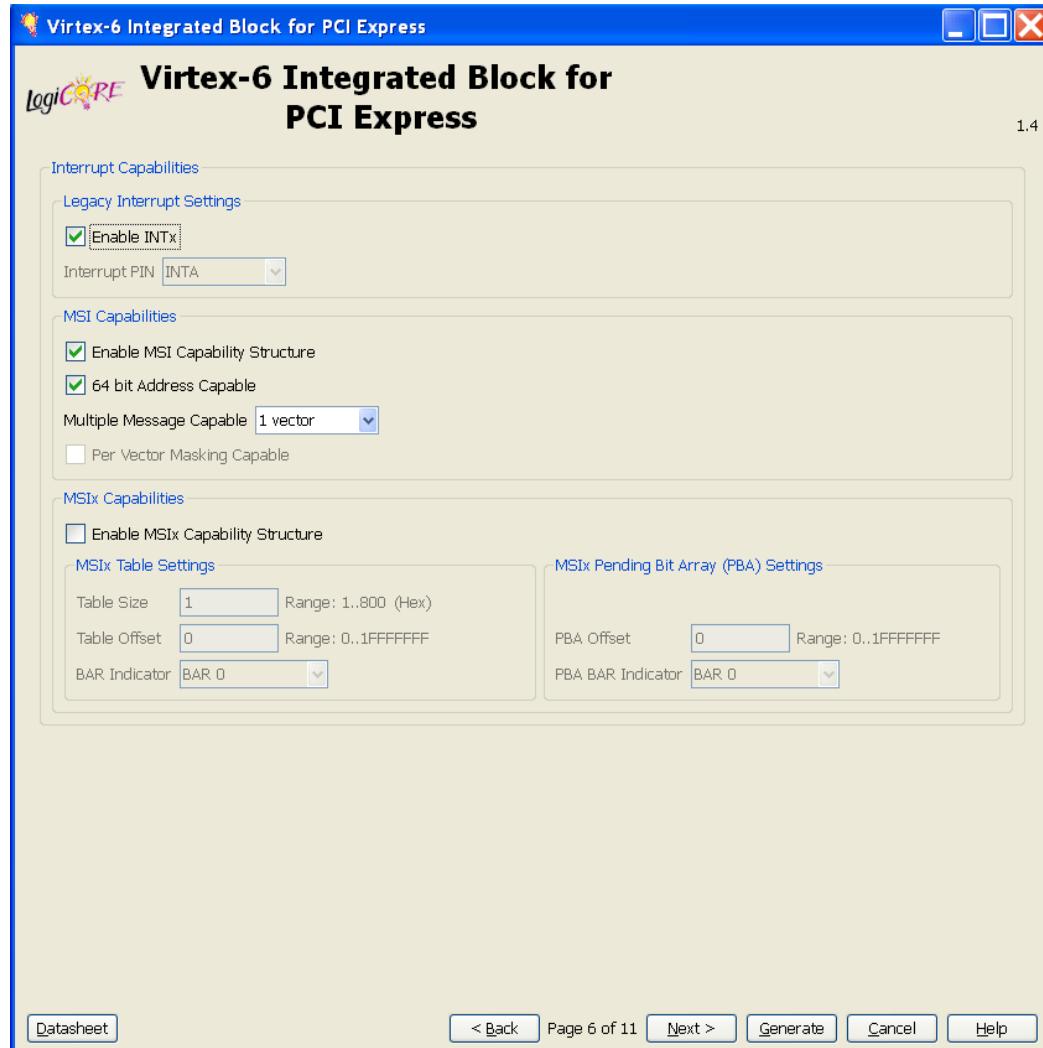


*Figure 5-8:* **Screen 6: Interrupt Capabilities**

### Legacy Interrupt Settings

- **Enable INTX**: Enables the ability of the PCI Express function to generate INTx interrupts
- **Interrupt PIN**: Indicates the mapping for Legacy Interrupt messages. A setting of "None" indicates no Legacy Interrupts are used.

### MSI Capabilities

- **Enable MSI Capability Structure**: Indicates that the MSI Capability structure exists.
- **64 bit Address Capable**: Indicates that the function is capable of sending a 64-bit Message Address.

- **Multiple Message Capable**: Selects the number of MSI vectors to request from the Root Complex.

- **Per Vector Masking Capable**: Indicates that the function supports MSI per-vector Masking.

## MSI-X Capabilities

- **Enable MSIx Capability Structure**: Indicates that the MSI-X Capability structure exists.

   *Note:* This Capability Structure needs at least one Memory BAR to be configured.

- **MSIx Table Settings**: Defines the MSI-X Table Structure.

   - *Table Size*: Specifies the MSI-X Table Size.

   - *Table Offset*: Specifies the Offset from the Base Address Register that points to the Base of the MSI-X Table.

   - *BAR Indicator*: Indicates the Base Address Register in the Configuration Space that is used to map the function's MSI-X Table, onto Memory Space. For a 64-bit Base Address Register, this indicates the lower DWORD.

- **MSIx Pending Bit Array (PBA) Settings**: Defines the MSI-X Pending Bit Array (PBA) Structure.

   - *PBA Offset*: Specifies the Offset from the Base Address Register that points to the Base of the MSI-X Pending Bit Array (PBA).

   - *PBA BAR Indicator*: Indicates the Base Address Register in the Configuration Space that is used to map the function's MSI-X Pending Bit Array (PBA), onto Memory Space.

## Power Management Registers

The Power Management Registers screen shown in Figure 5-9 includes settings for the Power Management Registers, power consumption and power dissipation options.
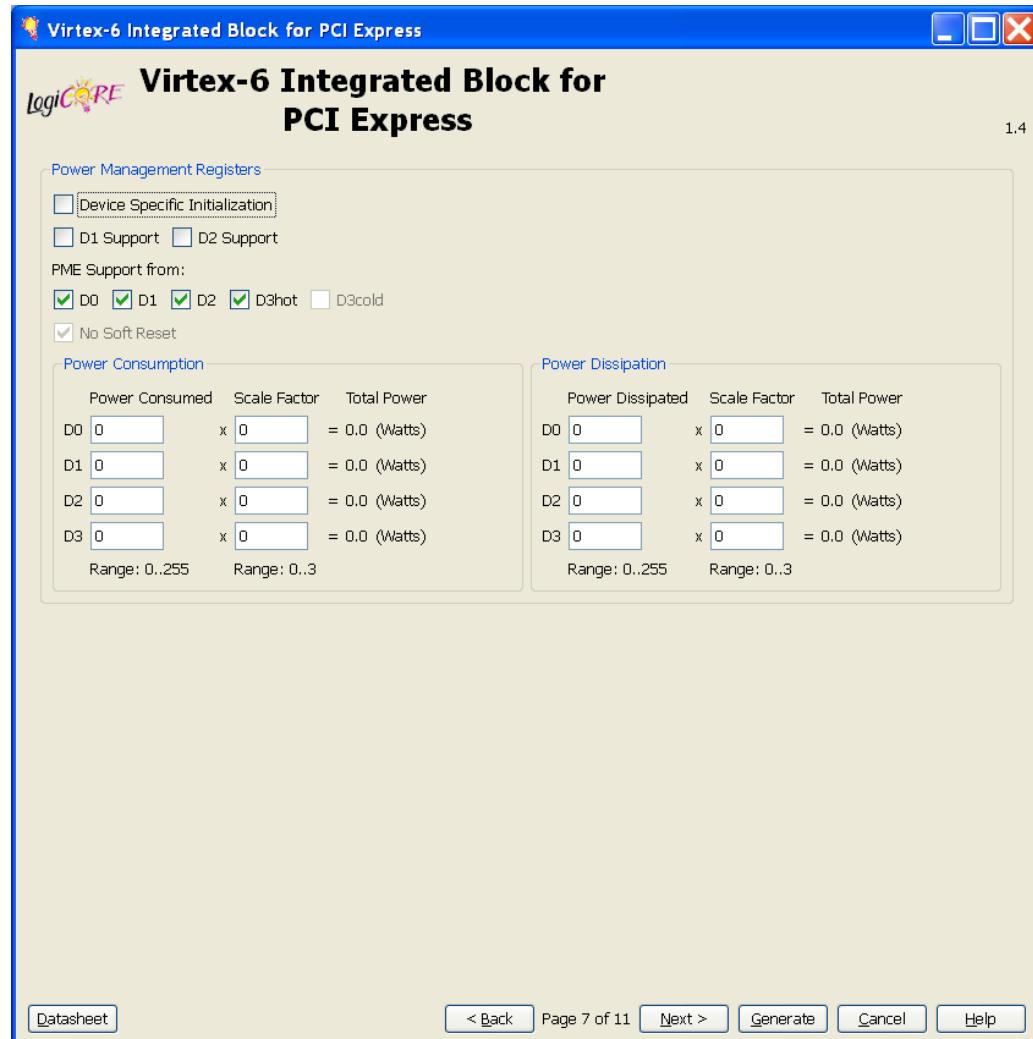


*Figure 5-9:* **Power Management Registers: Screen 7**

- **Device Specific Initialization**: This bit indicates whether special initialization of this function is required (beyond the standard PCI configuration header) before the generic class device driver is able to use it. When selected, this option indicates that the function requires a device specific initialization sequence following transition to the D0 uninitialized state. See section 3.2.3 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

- **D1 Support**: When selected, this option indicates that the function supports the D1 Power Management State. See section 3.2.3 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

- **D2 Support**: When selected, this option indicates that the function supports the D2 Power Management State. See section 3.2.3 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

- **PME Support From**: When this option is selected, it indicates the power states in which the function can assert cfg_pm_wake_n. See section 3.2.3 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

- **No Soft Reset**: Checking this box indicates that if the device transitions from D3hot to D0 because of a Power State Command, it does not perform an internal reset and Configuration context is preserved. This option is not supported.

## Power Consumption

The Virtex-6 FPGA Integrated Block for PCI Express always reports a power budget of 0W. For information about power consumption, see section 3.2.6 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

## Power Dissipated

The Virtex-6 FPGA Integrated Block for PCI Express always reports a power dissipation of 0W. For information about power dissipation, see section 3.2.6 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

## PCI Express Extended Capabilities

The PCIe Extended Capabilities screen shown in Figure 5-10 includes settings for Device Serial Number Capability, Virtual Channel Capability, Vendor Specific Capability, and optional user-defined Configuration capabilities.
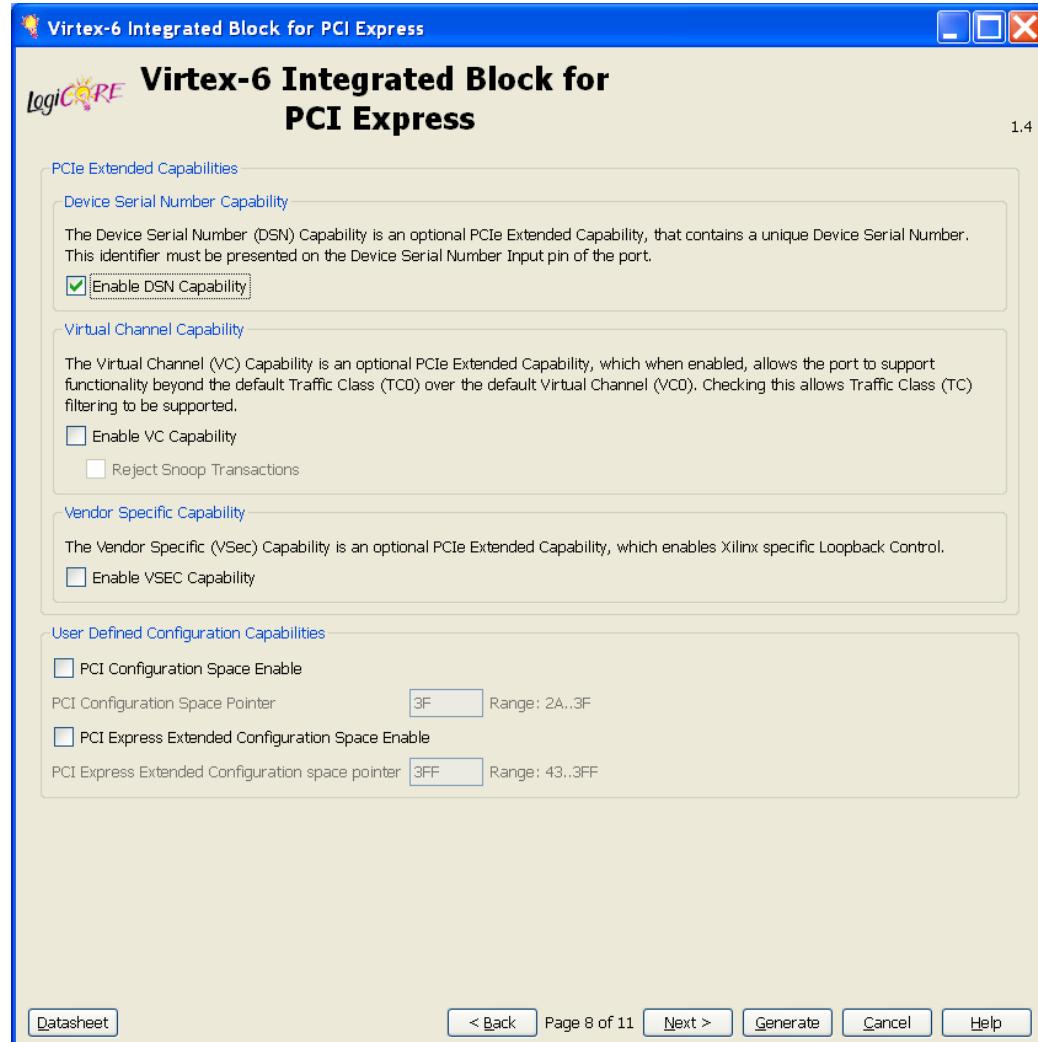


*Figure 5-10:* **Screen 8: PCIe Extended Capabilities**

## Device Serial Number Capability

- **Device Serial Number Capability**: An optional PCIe Extended Capability containing a unique Device Serial Number. When this Capability is enabled, the DSN identifier must be presented on the Device Serial Number input pin of the port. This Capability must be turned on to enable the Virtual Channel and Vendor Specific Capabilities

## Virtual Channel Capability

- **Virtual Channel Capability**: An optional PCIe Extended Capability which allows the user application to be operated in TCn / VC0 mode. Checking this allows Traffic Class filtering to be supported.

- **Reject Snoop Transactions (Root Port Configuration Only)**: When enabled, any transactions for which the No Snoop attribute is applicable, but is not set in the TLP header, can be rejected as an Unsupported Request.

## Vendor Specific Capability

- **Vendor Specific Capability**: An optional PCIe Extended Capability that allows PCI Express component vendors to expose Vendor Specific Registers. When checked, enables Xilinx specific Loopback Control.

## User Defined Configuration Capabilities: Endpoint Configuration Only

- **PCI Configuration Space Enable**: Allows the user application to add/implement PCI Legacy capability registers. This option should be selected if the user application implements a legacy capability configuration space. This option enables the routing of Configuration Requests to addresses outside the built-in PCI-Compatible Configuration Space address range to the Transaction Interface.

- **PCI Configuration Space Pointer**: Sets the starting Dword aligned address of the user definable PCI-Compatible Configuration Space. The available Dword address range is 2Ah - 3Fh.

- **PCI Express Extended Configuration Space Enable**: Allows the user application to add/implement PCI Express Extended capability registers. This option should be selected if the user application implements such an extended capability configuration space. This enables the routing of Configuration Requests to addresses outside the built-in PCI Express Extended Configuration Space address range to the User Application.

- **PCI Configuration Space Pointer**: Sets the starting Dword aligned address of the PCI Express Extended Configuration Space implemented by the user application. This action enables routing of Configuration Requests with Dword addresses greater than or equal to the value set in the user application. The available address range depends on the PCIe Extended Capabilities selected. For more information, see Chapter 6, Designing with the Core.

## Pinout Selection

The Pinout Selection screen shown in Figure 5-11 includes options for pinouts specific to Xilinx Development Boards and PCIe Block Location.
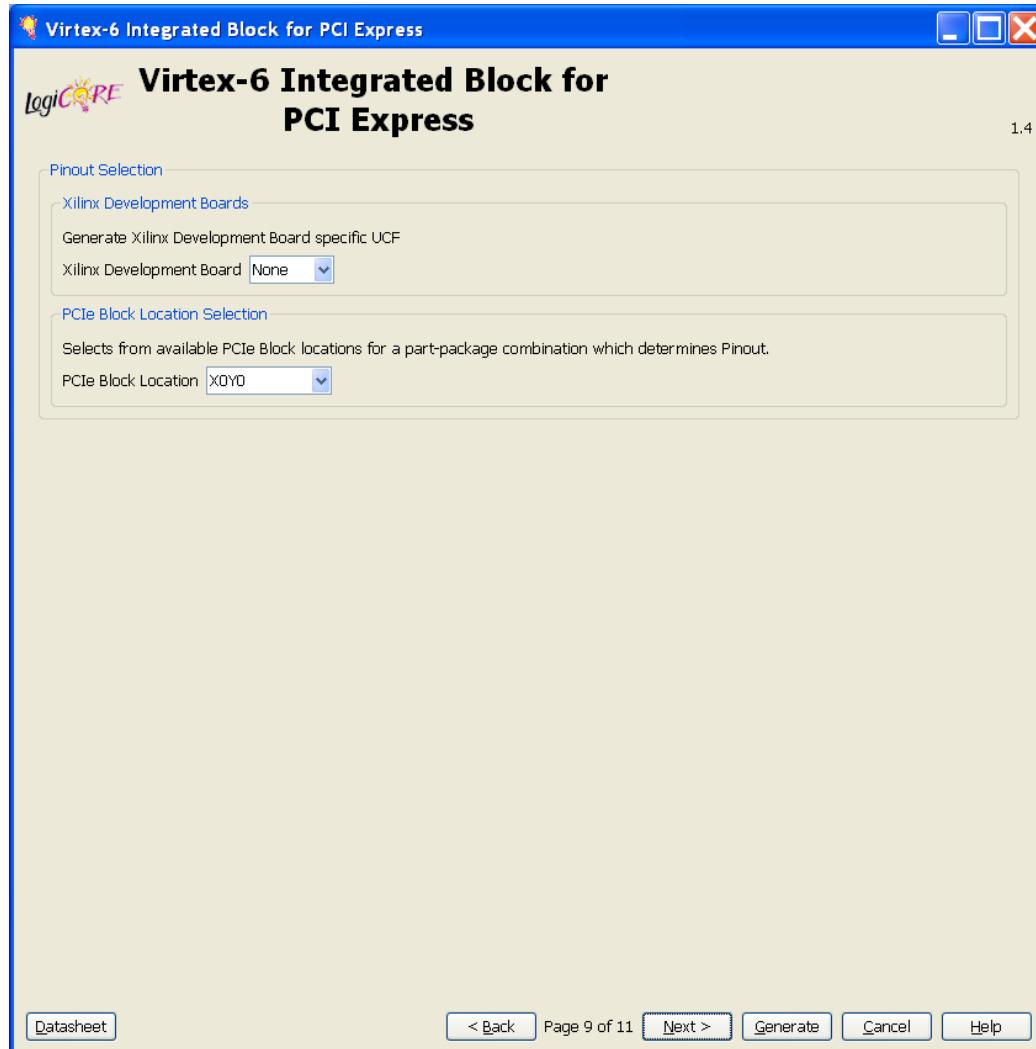


*Figure 5-11:* **Screen 9: Pinout Selection**

- **Xilinx Development Boards**: Selects the Xilinx Development Board to enable the generation of Xilinx Development Board specific constraints files.

- **PCIe Block Location Selection**: Selects from the PCIe Blocks available to enable generation of location specific constraint files and pinouts. When options "X0Y0 & X0Y1" or "X0Y2 & X0Y3" are selected, constraints files for both PCIe Block locations are generated, and the constraints file for the X0Y0 or X0Y3 location is used.

  This option is not available if a Xilinx Development Board is selected.

## Advanced Settings

The Advanced Settings screens shown in Figure 5-12 and Figure 5-13 include settings for Transaction Layer, Link Layer, Physical Layer, DRP Ports, and Reference Clock Frequency options.
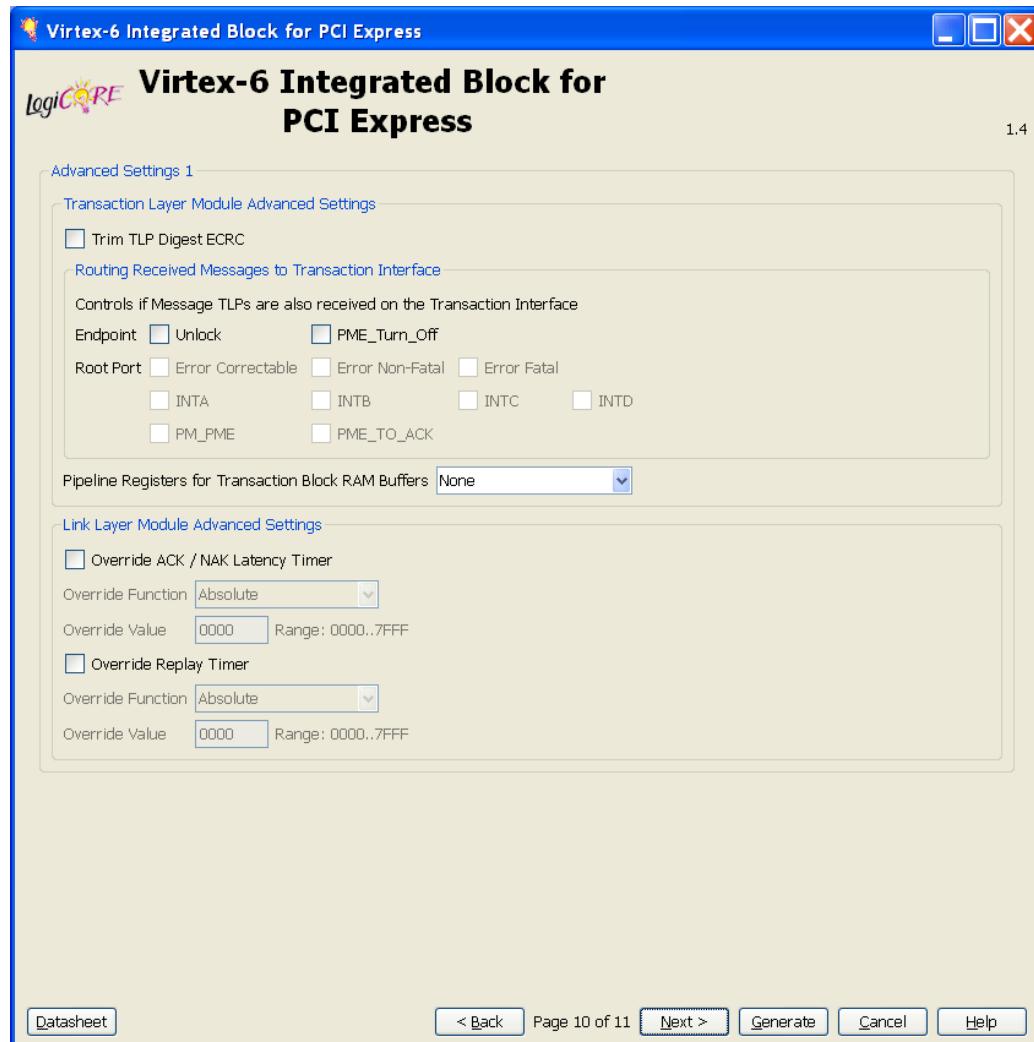


*Figure 5-12:* **Screen 10: Advanced Settings 1**

UG517_c5_13_021210
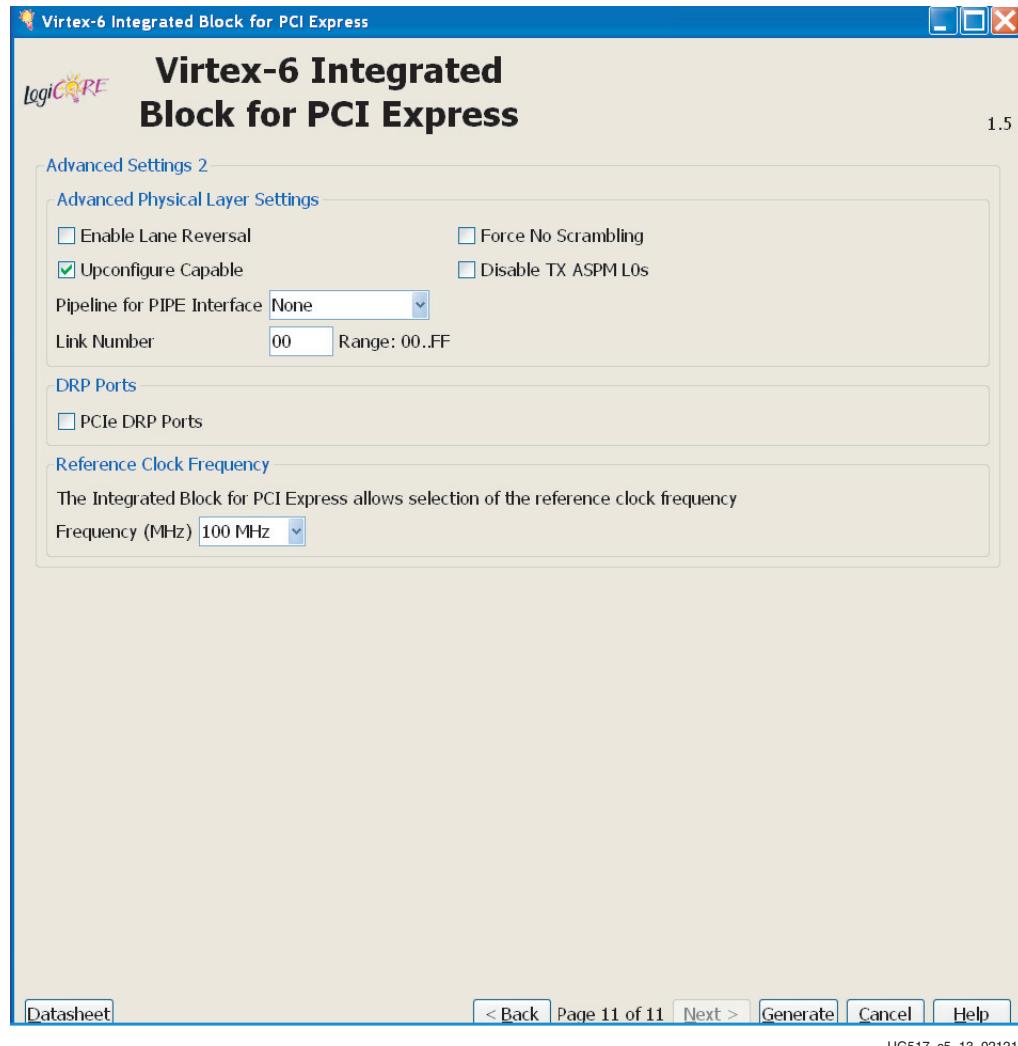
*Figure 5-13:* **Screen 11: Advanced Settings 2**

## Transaction Layer Module

- **Trim TLP Digest ECRC**: Causes the core to trim any TLP digest from an inbound packet and clear the TLP Digest bit in the TLP header, before presenting it to the user.

- **Enable Message Routing**: Controls if message TLPs are also received on the Transaction Interface.

  - Endpoint:

    - Unlock and PME_Turn_Off Messages

  - Root Port:

    - Error Messages - Error Correctable, Error Non-Fatal, Error Fatal

    - Assert/Deassert INT Messages - INTA, INTB, INTC, INTD

    - Power Management Messages - PM_PME, PME_TO_ACK

- **Pipeline Registers for Transaction Block RAM Buffers**: Selects the Pipeline registers enabled for the Transaction Buffers. Pipeline registers can be enabled on either the Write path or both the Read and Write paths of the Transaction Block RAM buffers.

## Link Layer Module

- **Override ACK/NAK Latency Timer**: Checking this box enables the user to override the ACK/NAK latency timer values set in the device. Use of this feature could cause the ACK timeout values to be non-compliant to the *PCI Express Base Specification, rev. 2.0.* This setting can be used to perform advanced debugging operations. Any modifications to default attributes must be made only if directed by Xilinx Technical Support.

- **ACK Latency Timer Override Function**: This setting determines how the override value is used by the device with respect to the ACK/NAK Latency Timer Value in the device. Options are "Absolute", "Add", and "Subtract". The first two settings could cause the ACK timeout values to be non-compliant with the *PCI Express Base Specification, rev. 2.0.*

- **ACK Latency Timer Override Value**: This setting determines the ACK/NAK latency timer value used by the device depending on if the ACK Latency Timer Override Function enabled. The built-in table value depends on the Negotiated Link Width and Programmed MPS of the device.

- **Override Replay Timer**: Checking this box enables the user to override the replay timer values set in the device. Use of this feature could cause the replay timeout values to be non-compliant to the *PCI Express Base Specification, rev. 2.0.* This setting can be used to perform advanced debugging operations. Any modifications to default attributes must be made only if directed by Xilinx Technical Support.

- **Replay Timer Override Function**: This setting determines how the override value is used by the device with respect to the replay timer value in the device. Options are "Absolute", "Add", and "Subtract". The first two settings could cause the replay timeout values to be non-compliant with the *PCI Express Base Specification, rev. 2.0.*

- **Replay Timer Override Value**: This setting determines the replay timer value used by the device depending on if the Replay Timer Override Function enabled. The built-in table value depends on the Negotiated Link Width and Programmed MPS of the device. The user must ensure that the final timeout value does not overflow the 15-bit timeout value.

## Advanced Physical Layer

- **Enable Lane Reversal**: When checked, enables the Lane Reversal feature.

- **Force No Scrambling**: Used for diagnostic purposes only and should never be enabled in a working design. Setting this bit results in the data scramblers being turned off so that the serial data stream can be analyzed.

- **Upconfigure Capable**: When unchecked, the port is advertised as "Not Upconfigure Capable" during Link Training.

- **Disable TX ASPM L0s**: Recommended for a link that interconnects a Virtex-6 FPGA to any Xilinx component. This prevents the device transmitter from entering the L0s state.

- **Pipeline for PIPE Interface**: Selects the number of Pipeline stages enabled on the PIPE Interface.

- **Link Number**: Specifies the link number advertised by the device in TS1 and TS2 ordered sets during Link training. Used in downstream facing mode only.

## Debug Ports

- **PCIe DRP Ports**: Checking this box enables the generation of DRP ports for the PCIe Hard Block, giving users dynamic control over the PCIe Hard Block attributes. This setting can be used to perform advanced debugging. Any modifications to the PCIe default attributes must be made only if directed by Xilinx Technical Support.

## Reference Clock Frequency

Selects the frequency of the reference clock provided on sys_clk. For important information about clocking the Virtex-6 FPGA Integrated Block for PCI Express, see Clocking and Reset of the Integrated Block Core, page 183.

**EX XILINX**

*Chapter 6*

# *Designing with the Core*

This chapter provides design instructions for the Virtex®-6 FPGA Integrated Block for PCI Express® User Interface and assumes knowledge of the PCI Express Transaction Layer Packet (TLP) header fields. Header fields are defined in *PCI Express Base Specification v2.0*, the "Transaction Layer Specification" chapter.

This chapter includes these design guidelines:

- Transmitting Outbound Packets
- Receiving Inbound Packets
- Design with Configuration Space Registers and Configuration Interface
- Additional Packet Handling Requirements
- Power Management
- Generating Interrupt Requests
- Link Training: 2-Lane, 4-Lane, and 8-Lane Components
- Clocking and Reset of the Integrated Block Core

## Designing with the Transaction Layer Interface

### Designing with the 64-bit Transaction Layer Interface

#### TLP Format on the Transaction Interface

Data is transmitted and received in Big-Endian order as required by the *PCI Express Base Specification*. See the "Transaction Layer Specification" chapter of the *PCI Express Base Specification* for detailed information about TLP packet ordering. Figure 6-1 represents a typical 32-bit addressable Memory Write Request TLP (as illustrated in the "Transaction Layer Specification" chapter of the specification).
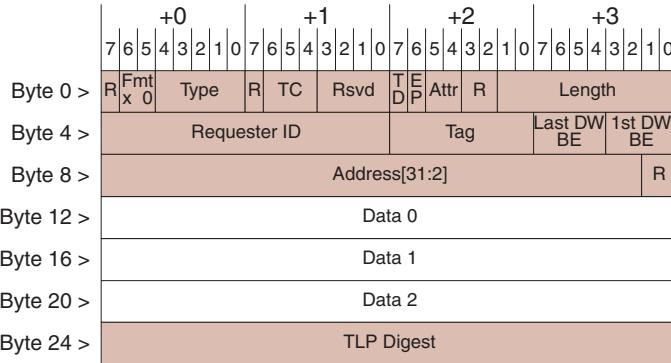
*Figure 6-1:* **PCI Express Base Specification Byte Order**

When using the Transaction interface, packets are arranged on the entire 64-bit datapath. Figure 6-2 shows the same example packet on the Transaction interface. Byte 0 of the packet appears on trn_td[63:56] (outbound) or trn_rd[63:56] (inbound) of the first QWORD, byte 1 on trn_td[55:48] or trn_rd[55:48], and so forth. Byte 8 of the packet then appears on trn_td[63:56] or trn_rd[63:56] of the second QWORD. The Header section of the packet consists of either three or four DWORDs, determined by the TLP format and type as described in section 2.2 of the *PCI Express Base Specification*.
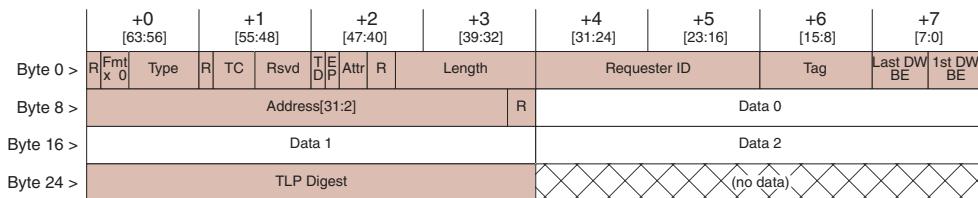


*Figure 6-2:* **Endpoint Integrated Block Byte Order**

Packets sent to the core for transmission must follow the formatting rules for Transaction Layer Packets (TLPs) as specified in the "Transaction Layer Specification" chapter of the *PCI Express Base Specification*. The User Application is responsible for ensuring its packets' validity, as the core does not check packet validity or validate packets. The exact fields of a given TLP vary depending on the type of packet being transmitted.

The core allows the User Application to add an extra level of error checking by using the optional TLP Digest Field in the TLP header. The presence of a TLP Digest or ECRC is indicated by the value of TD field in the TLP Header section. When TD=1, a correctly computed CRC32 remainder is expected to be presented as the last DWORD of the packet. The CRC32 remainder DWORD is not included in the length field of the TLP header. The User Application must calculate and present the TLP Digest as part of the packet when transmitting packets. Upon receiving packets with a TLP Digest present, the User Application must check the validity of the CRC32 based on the contents of the packet. The core does not check the TLP Digest for incoming packets. The *PCI Express Base Specification* requires Advanced Error Reporting (AER) capability when implementing ECRC. Although the integrated block does not support AER, users can still implement ECRC for custom solutions that do not require compliance with the *PCI Express Base Specification*.

## Transmitting Outbound Packets

### Basic TLP Transmit Operation

The Virtex-6 FPGA Integrated Block for PCI Express core automatically transmits these types of packets:

- Completions to a remote device in response to Configuration Space requests.

- Error-message responses to inbound requests that are malformed or unrecognized by the core.

   **Note:** Certain unrecognized requests, for example, unexpected completions, can only be detected by the User Application, which is responsible for generating the appropriate response.

The User Application is responsible for constructing these types of outbound packets:

- Memory and I/O Requests to remote devices.

- Completions in response to requests to the User Application, for example, a Memory Read Request.

- Completions in response to user-implemented Configuration Space requests, when enabled. These requests include PCI™ legacy capability registers beyond address `BFh` and PCI Express extended capability registers beyond address `1FFh`.

   **Note:** For important information about accessing user-implemented Configuration Space while in a low-power state, see Power Management, page 175.

When configured as an Endpoint, the Virtex-6 FPGA Integrated Block for PCI Express core notifies the User Application of pending internally generated TLPs that will be arbitrating for the transmit datapath by asserting trn_tcfg_req_n (`0b`). The User Application can choose to give priority to core-generated TLPs by asserting trn_tcfg_gnt_n (`0b`) permanently, without regard to trn_tcfg_req_n. Doing so prevents user-application-generated TLPs from being transmitted when a core-generated TLP is pending. Alternatively, the User Application can reserve priority for a user-application-generated TLP over core-generated TLPs, by deasserting trn_tcfg_gnt_n (`1b`) until the user transaction is complete. Then, it is asserted (`0b`) for one clock cycle. Users must not delay asserting trn_cfg_gnt_n indefinitely, because this might cause a completion time-out in the Requester. See the *PCI Express Base Specification* for more information on the Completion Timeout Mechanism.

The integrated block does not do any filtering on the Base/Limit registers (Root Port only). The user is responsible for determining if filtering is required. These registers can be read out of the Type 1 Configuration Header space via the Configuration Interface (see Design with Configuration Space Registers and Configuration Interface, page 155).

Table 2-12, page 34 defines the transmit User Application signals. To transmit a TLP, the User Application must perform this sequence of events on the transmit Transaction interface:

1. The User Application logic asserts trn_tsrc_rdy_n and trn_tsof_n and presents the first TLP QWORD on trn_td[63:0] when it is ready to transmit data. If the core is asserting trn_tdst_rdy_n, the QWORD is accepted immediately; otherwise, the User Application must keep the QWORD presented until the core asserts trn_tdst_rdy_n.

2. The User Application asserts trn_tsrc_rdy_n and presents the remainder of the TLP QWORDs on trn_td[63:0] for subsequent clock cycles (for which the core asserts trn_tdst_rdy_n).

3. The User Application asserts trn_tsrc_rdy_n and trn_teof_n together with the last QWORD data. If all eight data bytes of the last transfer are valid, they are presented on

trn_td[63:0] and trn_trem_n is driven to `0b`; otherwise, the four remaining data bytes are presented on trn_td[63:32] and trn_trem_n is driven to `1b`.

4. At the next clock cycle, the User Application deasserts trn_tsrc_rdy_n to signal the end of valid transfers on trn_td[63:0].

Figure 6-3 illustrates a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request. When the User Application asserts trn_teof_n, it also places a value of `1b` on trn_trem_n, notifying the core that only trn_td[63:32] contains valid data.
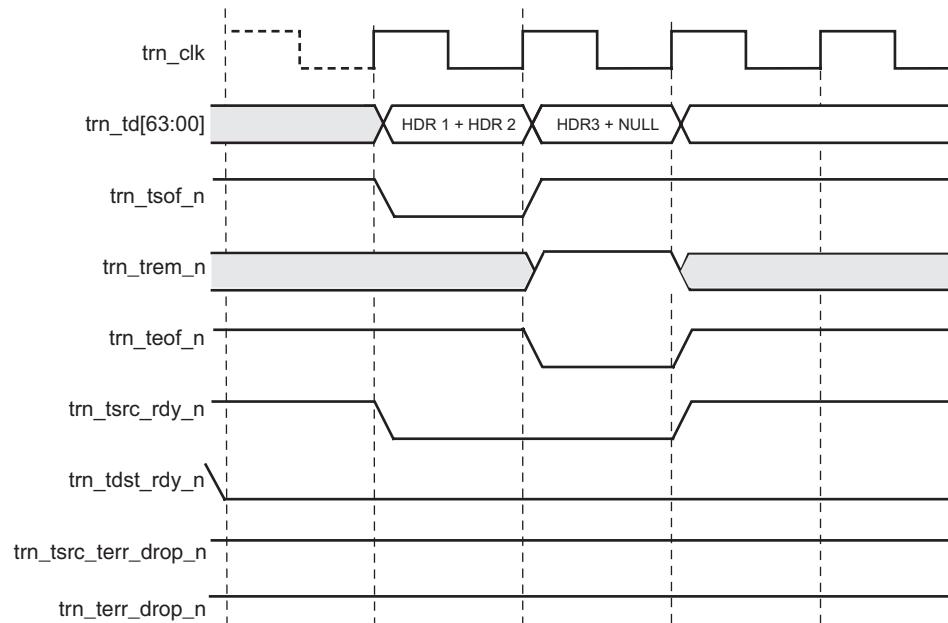


*Figure 6-3:* **TLP 3-DW Header without Payload**

Figure 6-4 illustrates a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request. When the User Application asserts trn_teof_n, it also places a value of 0b on trn_trem_n, notifying the core that trn_td[63:0] contains valid data.
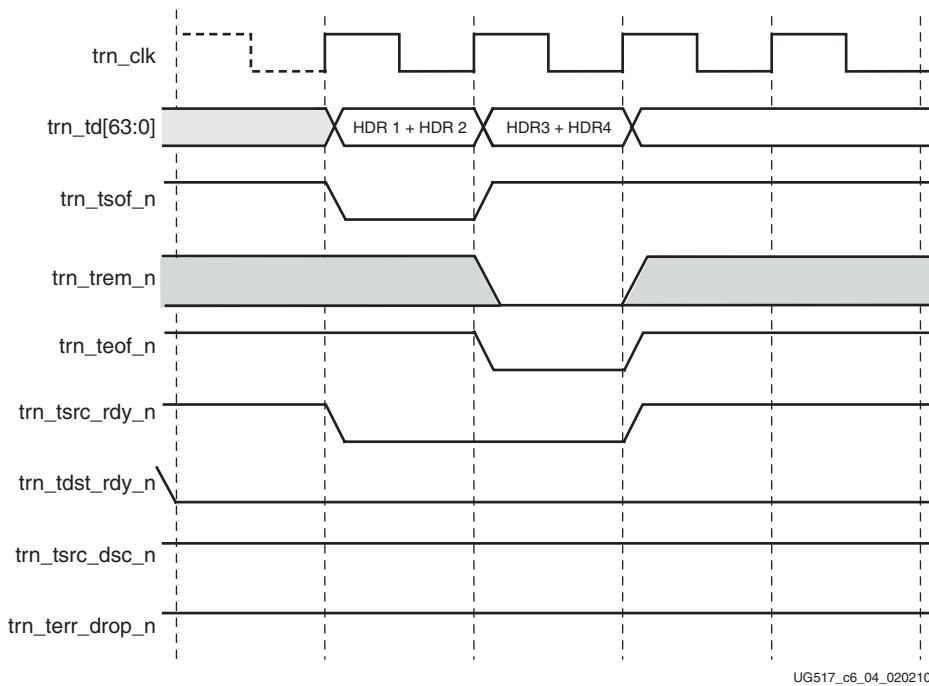


*Figure 6-4:* **TLP with 4-DW Header without Payload**

Figure 6-5 illustrates a 3-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request. When the User Application asserts trn_teof_n, it also puts a value of 0b on trn_trem_n, notifying the core that trn_td[63:0] contains valid data. The user must ensure the remainder field selected for the final data cycle creates a packet of length equivalent to the length field in the packet header.
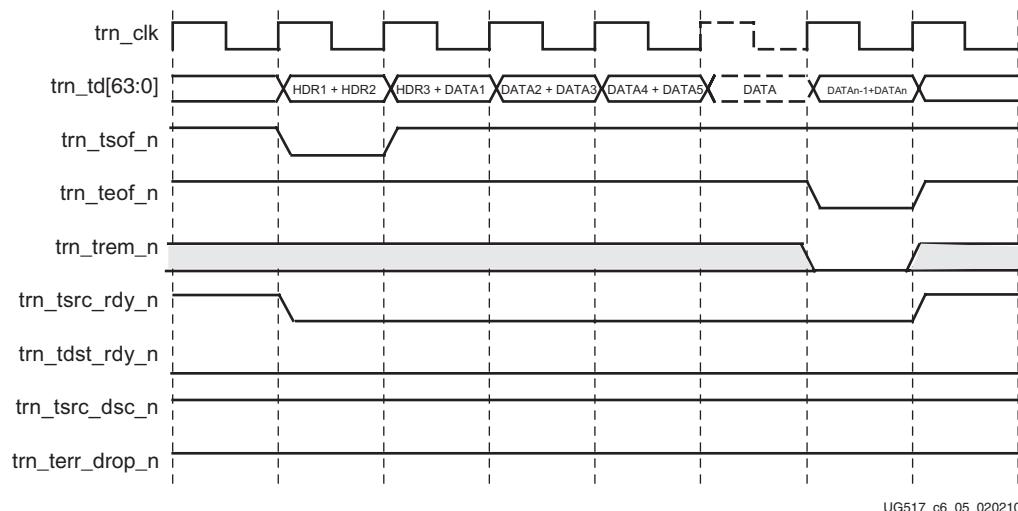


*Figure 6-5:* **TLP with 3-DW Header with Payload**

Figure 6-6 illustrates a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request. When the User Application asserts trn_teof_n, it also places a value of 1b on trn_trem_n, notifying the core that only trn_td[63:32] contains valid data. The user must ensure the remainder field is selected for the final data cycle creates a packet of length equivalent to the length field in the packet header.
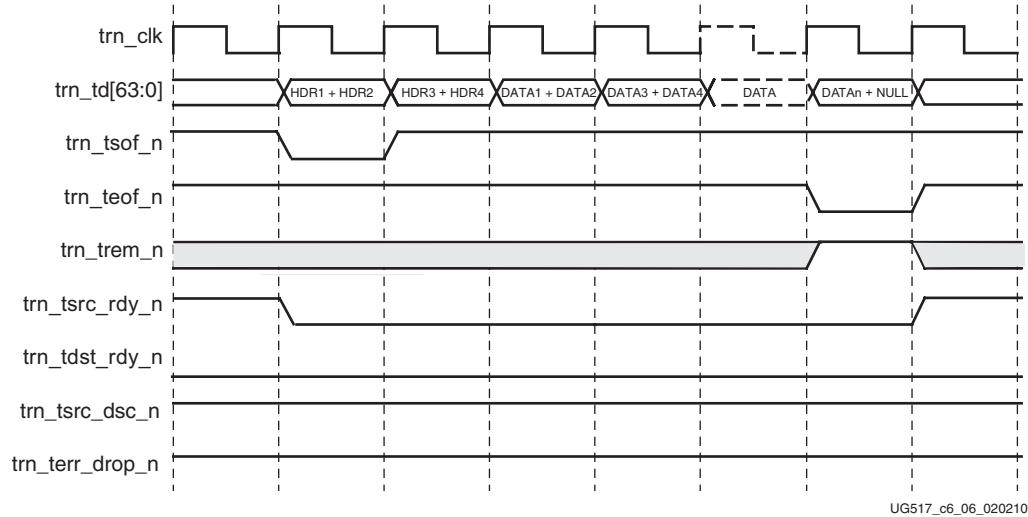


*Figure 6-6:* **TLP with 4-DW Header with Payload**

## Presenting Back-to-Back Transactions on the Transmit Interface

The User Application can present back-to-back TLPs on the transmit Transaction interface to maximize bandwidth utilization. Figure 6-7 illustrates back-to-back TLPs presented on the transmit interface. The User Application asserts trn_tsof_n and presents a new TLP on the next clock cycle after asserting trn_teof_n for the previous TLP.
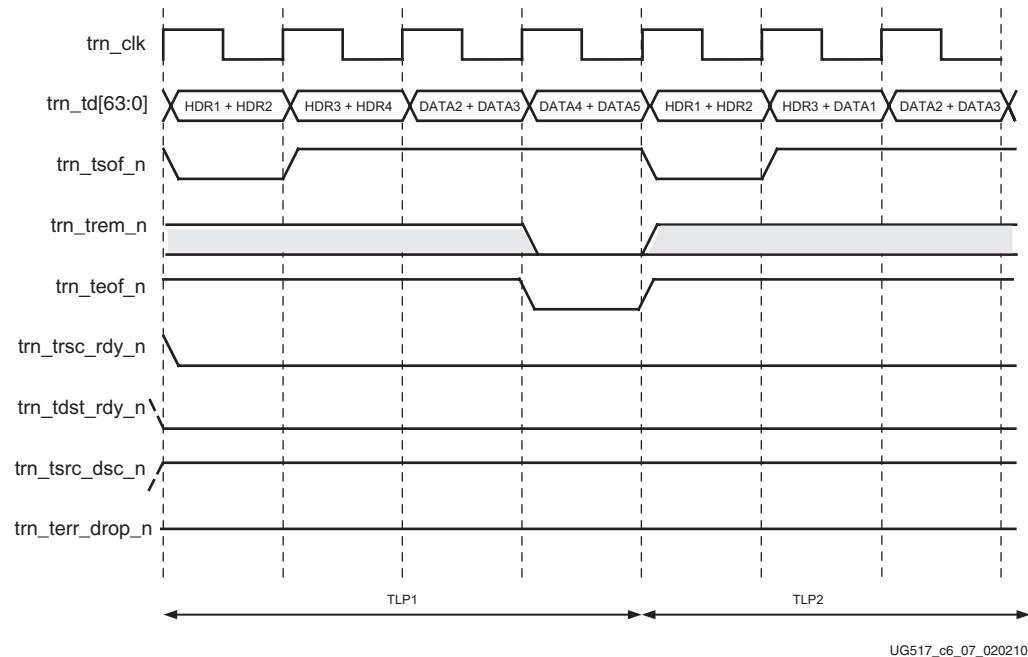


*Figure 6-7:* **Back-to-Back Transaction on Transmit Transaction Interface**

## Source Throttling on the Transmit Datapath

The Transaction interface lets the User Application throttle back if it has no data to present on trn_td[63:0]. When this condition occurs, the User Application deasserts trn_tsrc_rdy_n, which instructs the core Transaction interface to disregard data presented on trn_td[63:0]. Figure 6-8 illustrates the source throttling mechanism, where the User Application does not have data to present every clock cycle, and for this reason must deassert trn_tsrc_rdy_n during these cycles.
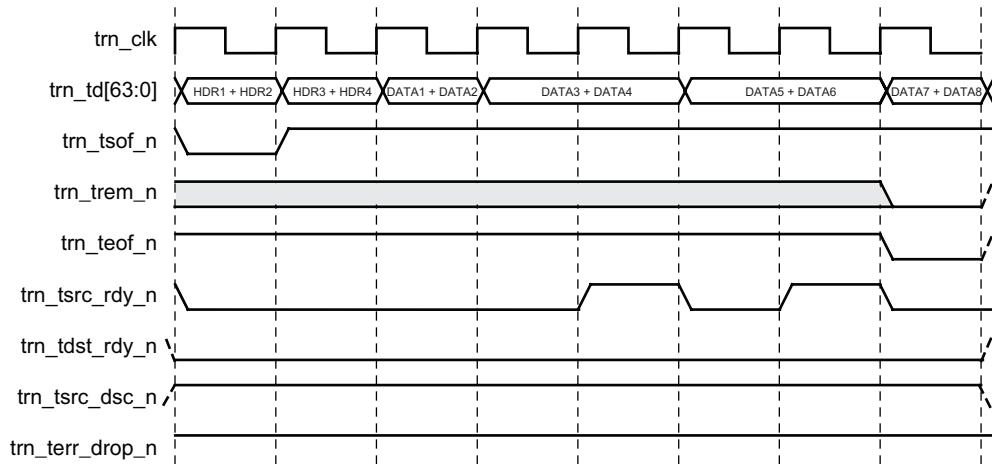


*Figure 6-8:* **Source Throttling on the Transmit Datapath**

## Destination Throttling of the Transmit Datapath

The core Transaction interface throttles the transmit User Application if there is no space left for a new TLP in its transmit buffer pool. This can occur if the link partner is not processing incoming packets at a rate equal to or greater than the rate at which the User Application is presenting TLPs. Figure 6-9 illustrates the deassertion of trn_tdst_rdy_n to throttle the User Application when the internal transmit buffers of the core are full.
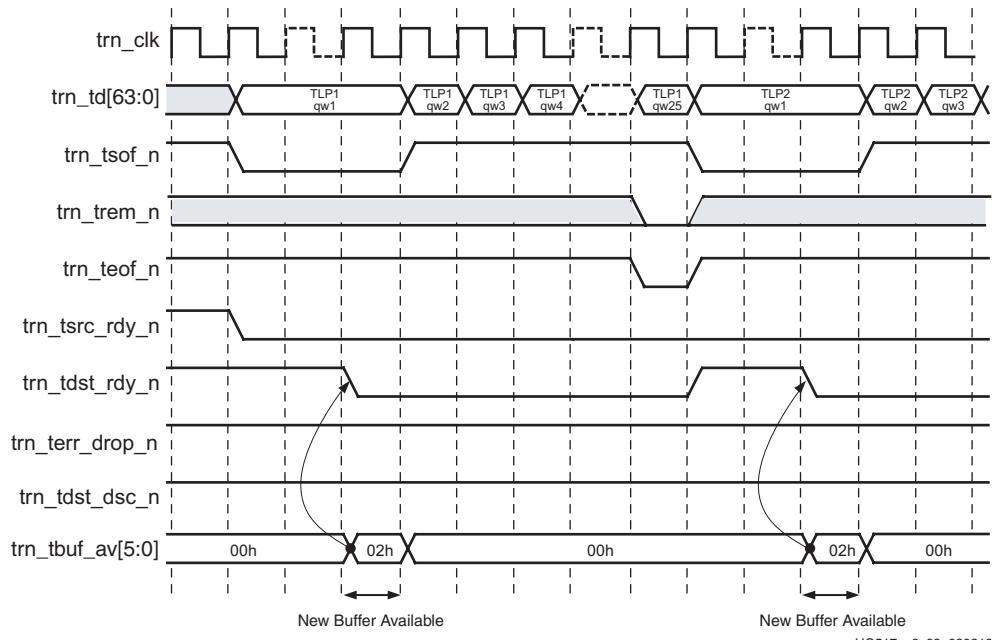


*Figure 6-9:* **Destination Throttling of the Endpoint Transmit Transaction Interface**

If the core transmit Transaction interface accepts the start of a TLP by asserting trn_tdst_rdy_n, it is guaranteed to accept the complete TLP with a size up to the value contained in the Max_Payload_Size field of the PCI Express Device Capability Register (offset 04H). To stay compliant to the *PCI Express Base Specification* users should not violate the Max_Payload_Size field of the PCI Express Device Control Register (offset 08H). The core transmit Transaction interface deasserts trn_tdst_rdy_n only under these conditions:

- After it has accepted the TLP completely and has no buffer space available for a new TLP.

- When the core is transmitting an internally generated TLP (configuration Completion TLP, error Message TLP or error response as requested by the User Application on the cfg_err interface), after it has been granted use of the transmit datapath by the User Application, by assertion of trn_tcfg_gnt_n. The core subsequently asserts trn_tdst_rdy_n after transmitting the internally generated TLP.

On deassertion of trn_tdst_rdy_n by the core, the User Application needs to hold all control and data signals until the core asserts trn_tdst_rdy_n.

The core transmit Transaction interface throttles the User Application when the Power State field in Power Management Control/Status Register (offset 0x4) of the PCI Power Management Capability Structure is changed to a non-D0 state. When this occurs, any ongoing TLP is accepted completely and trn_tdst_rdy_n is subsequently deasserted, disallowing the User Application from initiating any new transactions—for the duration that the core is in the non-D0 power state.

### Discontinuing Transmission of Transaction by Source

The core Transaction interface lets the User Application terminate transmission of a TLP by asserting trn_tsrc_dsc_n. Both trn_tsrc_rdy_n and trn_tdst_rdy_n must be asserted together with trn_tsrc_dsc_n for the TLP to be discontinued. The signal trn_tsrc_dsc_n must not be asserted together with trn_tsof_n. It can be asserted on any cycle after trn_sof_n deasserts up to and including the assertion of trn_teof_n. Asserting trn_tsrc_dsc_n has no effect if no TLP transaction is in progress on the transmit interface. Figure 6-10 illustrates the User Application discontinuing a packet using trn_tsrc_dsc_n. Asserting trn_teof_n together with trn_tsrc_dsc_n is optional.



*Figure 6-10:* **Source Driven Transaction Discontinue on Transmit Interface**

### Discarding of Transaction by Destination

The core transmit Transaction interface discards a TLP for three reasons:

- PCI Express Link goes down.
- Presented TLP violates the Max_Payload_Size field of the Device Capability Register (offset 04H) for PCI Express. It is the user's responsibility to not violate the Max_Payload_Size field of the Device Control Register (offset 08H).
- trn_tstr_n is asserted and data is not presented on consecutive clock cycles, that is, trn_tsrc_rdy_n is deasserted in the middle of a TLP transfer.

When any of these occur, the transmit Transaction interface continues to accept the remainder of the presented TLP and asserts trn_terr_drop_n no later than the second clock cycle following the EOF of the discarded TLP. Figure 6-11 illustrates the core signaling that a packet was discarded using trn_terr_drop_n.
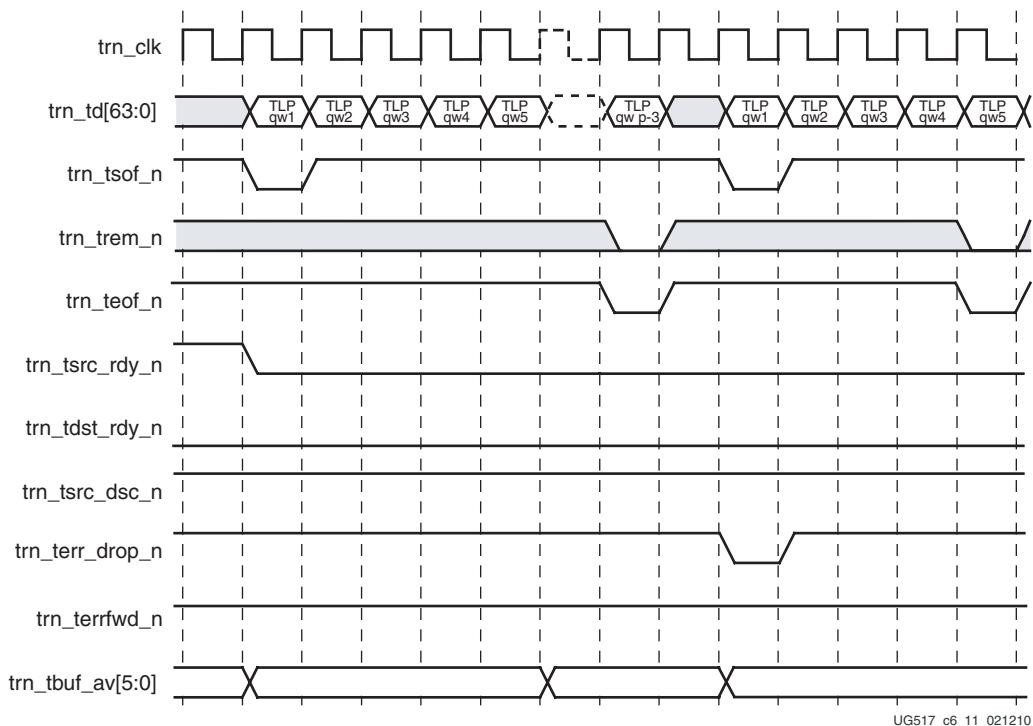


*Figure 6-11:* **Discarding of Transaction by Destination on Transmit Interface**

### Packet Data Poisoning on the Transmit Transaction Interface

The User Application uses this mechanism to mark the data payload of a transmitted TLP as poisoned:

- Set EP = 1 in the TLP header. This mechanism can be used if the payload is known to be poisoned when the first DWORD of the header is presented to the core on the TRN interface.
- Assert trn_terr_fwd_n for at least one valid data transfer cycle any time during the packet transmission, as shown in Figure 6-12. This causes the core to set EP = 1 in the TLP header when it transmits the packet onto the PCI Express fabric. This mechanism can be used if the User Application does not know whether a packet could be poisoned at the start of packet transmission. Use of trn_terrfwd_n is not supported for packets when trn_tstr_n is asserted (streamed transmit packets).
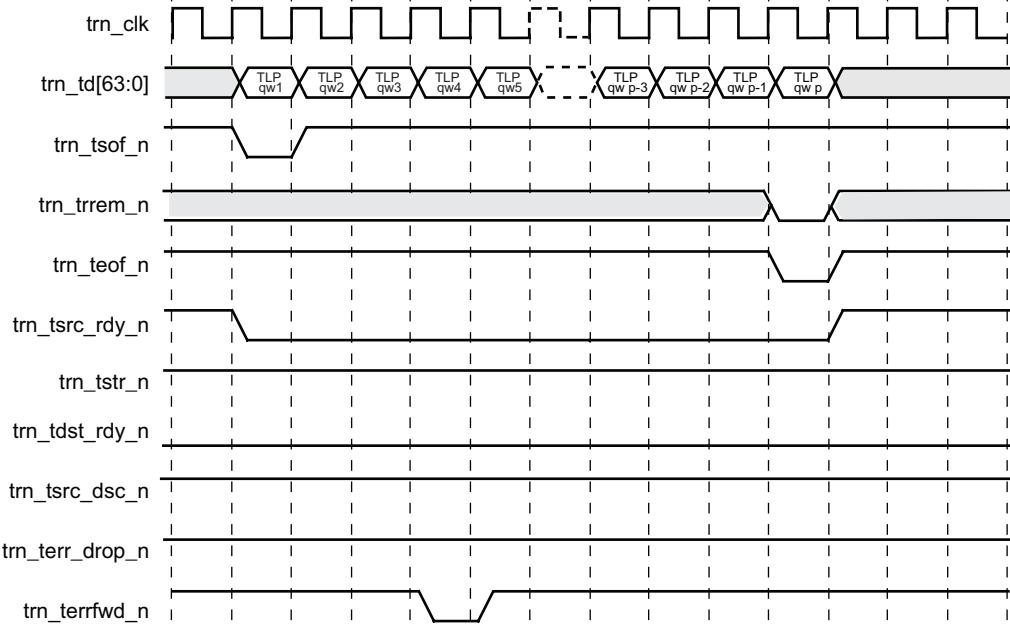
*Figure 6-12:* **Packet Data Poisoning on the Transmit Transaction Interface**

Streaming Mode for Transactions on the Transmit Interface

The Virtex-6 FPGA Integrated Block for PCI Express core allows the User Application to enable Streaming (cut-through) mode for transmission of a TLP, when possible, to reduce latency of operation. To enable this feature, the User Application must hold trn_tstr_n asserted for the entire duration of the transmitted TLP. In addition, the User Application must present valid frames on every clock cycle until the final cycle of the TLP. In other words, the User Application must not deassert trn_tsrc_rdy_n for the duration of the presented TLP. Source throttling of the transaction while in streaming mode of operation causes the transaction to be dropped (trn_terr_drop_n is asserted) and a nullified TLP to be signaled on the PCI Express link. Figure 6-13 illustrates the streaming mode of operation, where the first TLP is streamed and the second TLP is dropped due to source throttling.
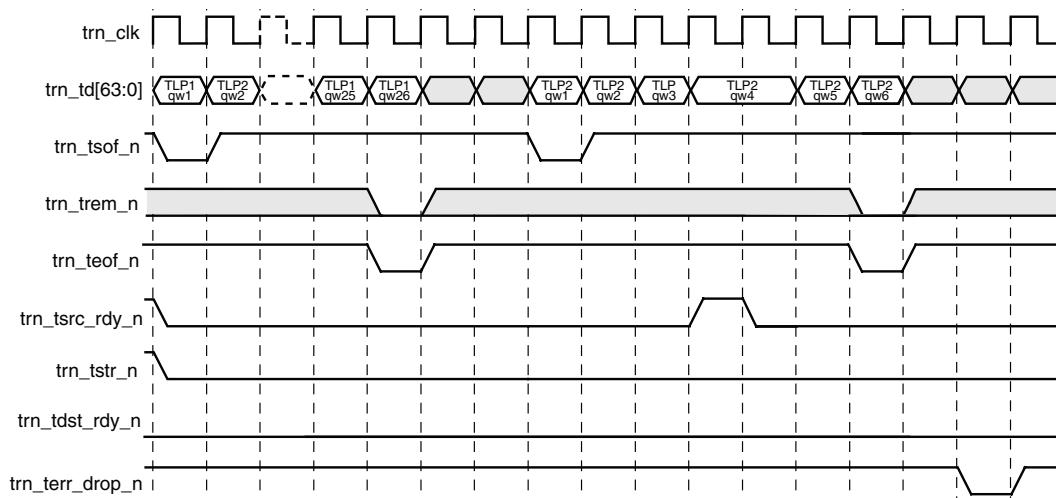


*Figure 6-13:* **Streaming Mode on the Transmit Interface**

**Note:** Source Driven Transaction Discontinue (assertion of trn_tsrc_dsc_n) is not supported when in the streaming mode of operation.

### Appending ECRC to Protect TLPs

If the User Application needs to send a TLP Digest associated with a TLP, it must construct the TLP header such that the TD bit is set and the User Application must properly compute and append the 1-DWORD TLP Digest after the last valid TLP payload section (if applicable). TLPs originating within the core do not have a TLP Digest appended, for example Completions, Error Messages, and Interrupts.

## Receiving Inbound Packets

### Basic TLP Receive Operation

Table 2-13, page 36 defines the receive Transaction interface signals. This sequence of events must occur on the receive Transaction interface for the Endpoint core to present a TLP to the User Application logic:

1. When the User Application is ready to receive data, it asserts trn_rdst_rdy_n.

2. When the core is ready to transfer data, the core asserts trn_rsrc_rdy_n with trn_rsof_n and presents the first complete TLP QWORD on trn_rd[63:0].

3. The core then deasserts trn_rsof_n, asserts trn_rsrc_rdy_n, and presents TLP QWORDs on trn_rd[63:0] for subsequent clock cycles, for which the User Application logic asserts trn_rdst_rdy_n.

4. The core then asserts trn_rsrc_rdy_n with trn_reof_n and presents either the last QWORD on trn_td[63:0] and a value of 0b on trn_rrem_n or the last DWORD on trn_td[63:32] and a value of 1b on trn_rrem_n.

5. If no further TLPs are available at the next clock cycle, the core deasserts trn_rsrc_rdy_n to signal the end of valid transfers on trn_rd[63:0].

***Note:*** The User Application should ignore any assertions of trn_rsof_n, trn_reof_n, trn_rrem_n, and trn_rd unless trn_rsrc_rdy_n is concurrently asserted.

Figure 6-14 shows a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request. When the core asserts trn_reof_n, it also places a value of 1b on trn_rrem_n, notifying the user that only trn_rd[63:32] contains valid data.



UG517_c6_14_020210
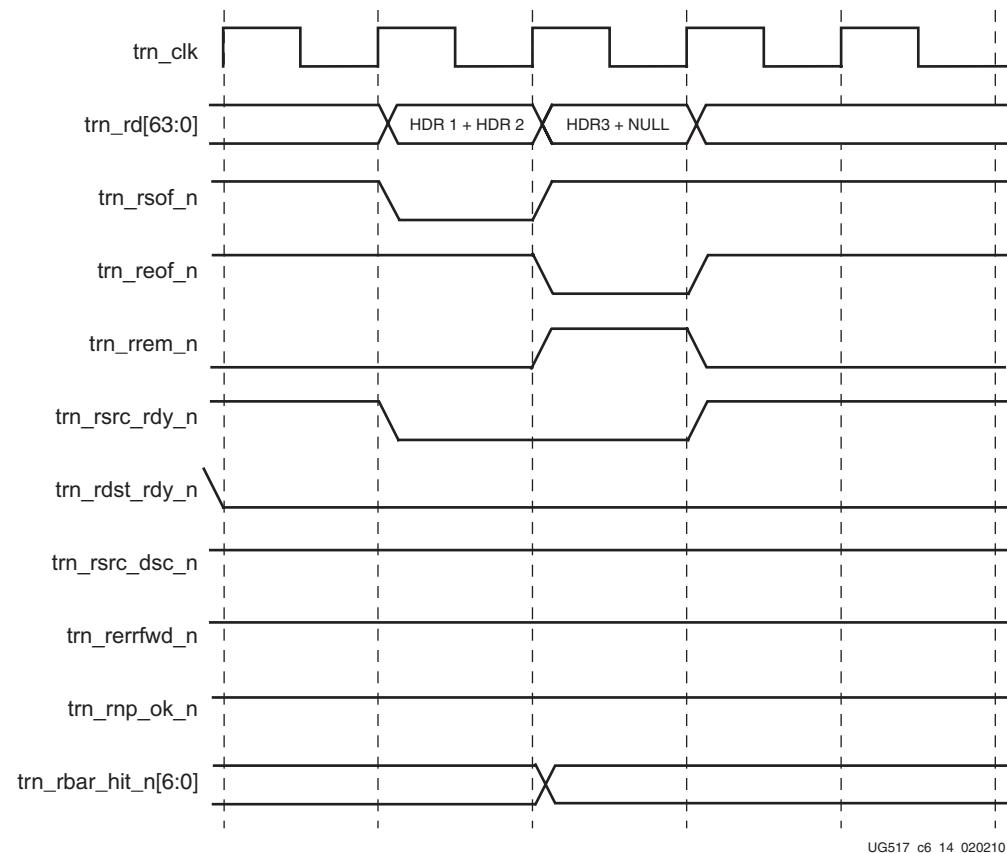
*Figure 6-14:* **TLP 3-DW Header without Payload**

Figure 6-15 shows a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request. When the core asserts trn_reof_n, it also places a value of 0b on trn_rrem_n, notifying the user that trn_rd[63:0] contains valid data.
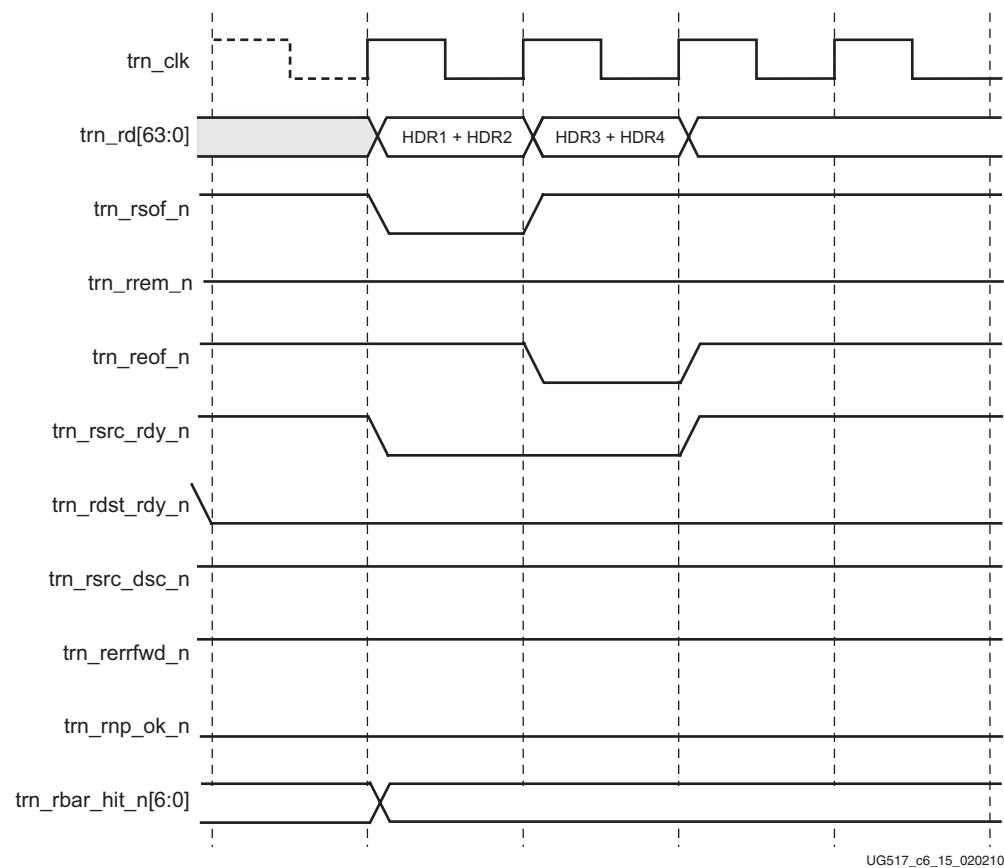


*Figure 6-15:* **TLP 4-DW Header without Payload**

Figure 6-16 shows a 3-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request. When the core asserts trn_reof_n, it also places a value of 0b on trn_rrem_n, notifying the user that trn_rd[63:0] contains valid data.
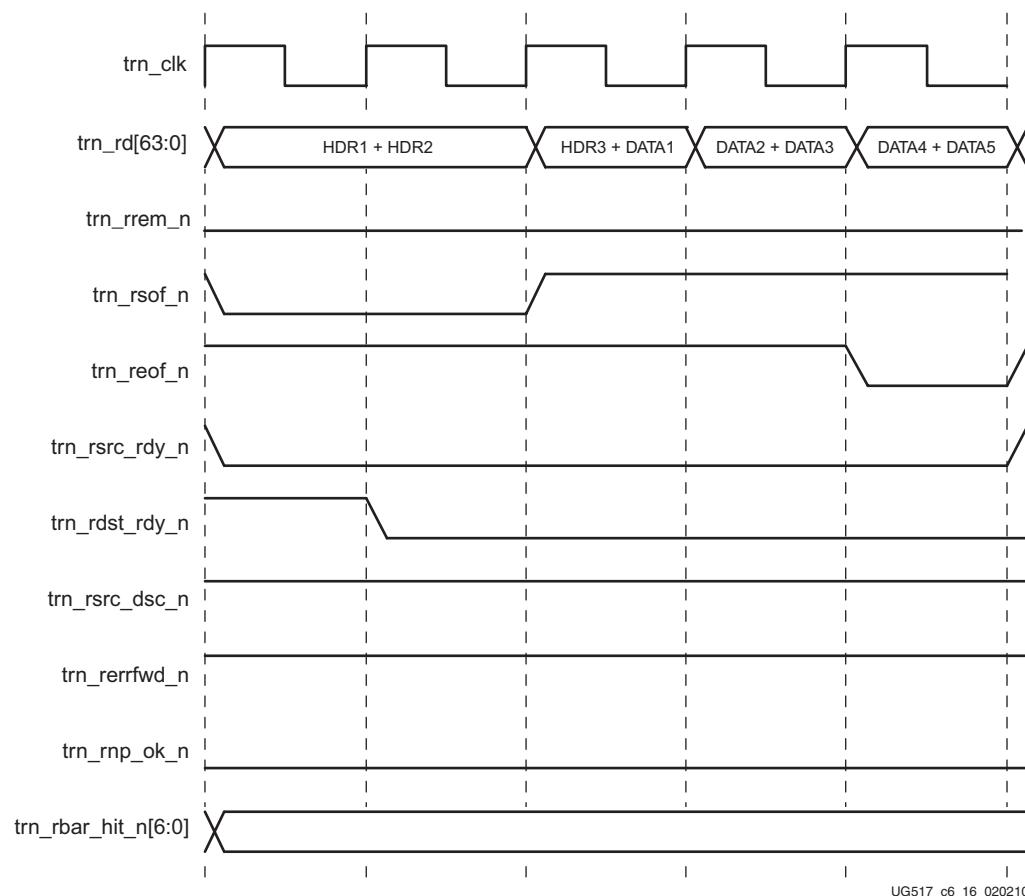


*Figure 6-16:* **TLP 3-DW Header with Payload**

Figure 6-17 shows a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request. When the core asserts trn_reof_n, it also places a value of 1b on trn_rrem_n, notifying the user that only trn_rd[63:32] contains valid data.
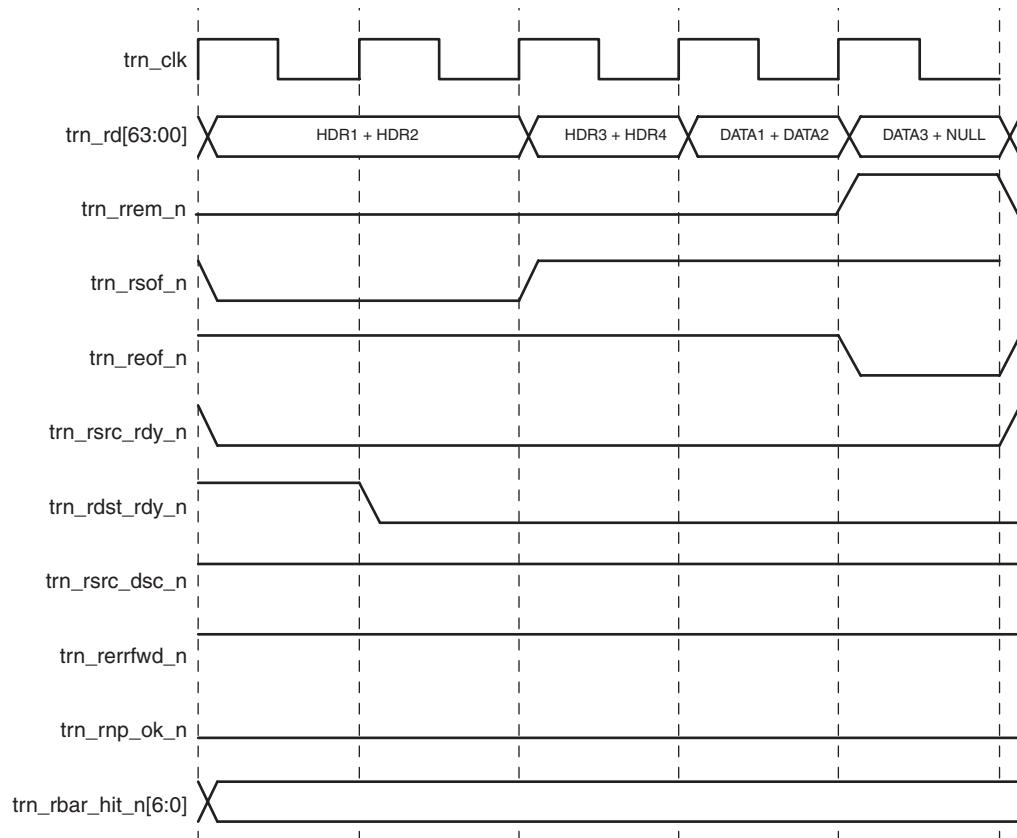


*Figure 6-17:* **TLP 4-DW Header with Payload**

## Throttling the Datapath on the Receive Transaction Interface

The User Application can stall the transfer of data from the core at any time by deasserting trn_rdst_rdy_n. If the user deasserts trn_rdst_rdy_n while no transfer is in progress and if a TLP becomes available, the core asserts trn_rsrc_rdy_n and trn_rsof_n and presents the first TLP QWORD on trn_rd[63:0]. The core remains in this state until the user asserts trn_rdst_rdy_n to signal the acceptance of the data presented on trn_rd[63:0]. At that point, the core presents subsequent TLP QWORDs as long as trn_rdst_rdy_n remains asserted. If the user deasserts trn_rdst_rdy_n during the middle of a transfer, the core stalls the transfer of data until the user asserts trn_rdst_rdy_n again. There is no limit to the number of cycles the user can keep trn_rdst_rdy_n deasserted. The core pauses until the user is again ready to receive TLPs.

Figure 6-18 illustrates the core asserting trn_rsrc_rdy_n and trn_rsof_n along with presenting data on trn_rd[63:0]. The User Application logic inserts wait states by deasserting trn_rdst_rdy_n. The core does not present the next TLP QWORD until it detects trn_rdst_rdy_n assertion. The User Application logic can assert or deassert trn_rdst_rdy_n as required to balance receipt of new TLP transfers with the rate of TLP data processing inside the application logic.
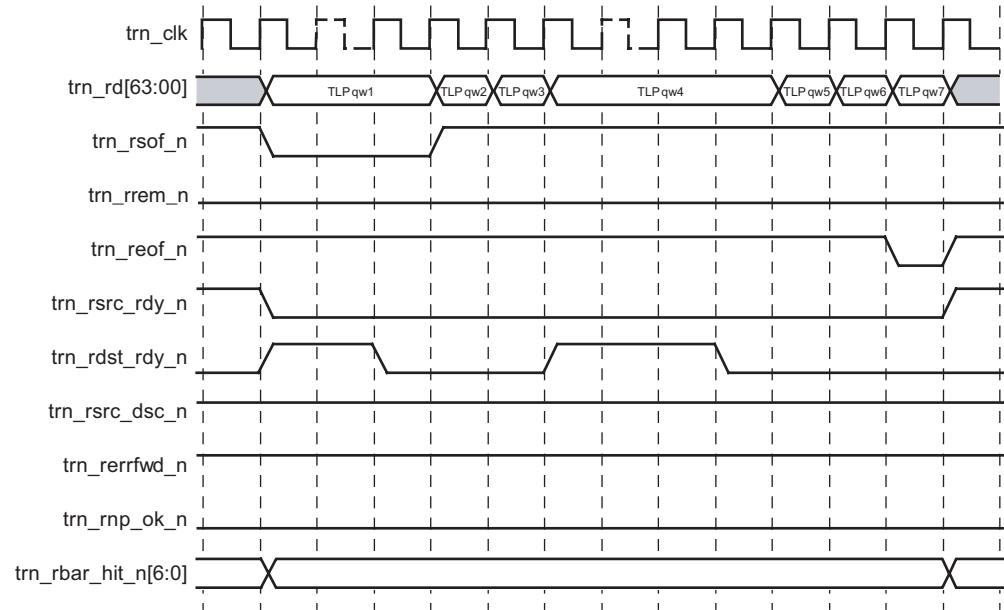
*Figure 6-18:* **User Application Throttling Receive TLP**

## Receiving Back-To-Back Transactions on the Receive Transaction Interface

The User Application logic must be designed to handle presentation of back-to-back TLPs on the receive Transaction interface by the core. The core can assert trn_rsof_n for a new TLP at the clock cycle after trn_reof_n assertion for the previous TLP. Figure 6-19 illustrates back-to-back TLPs presented on the receive interface.
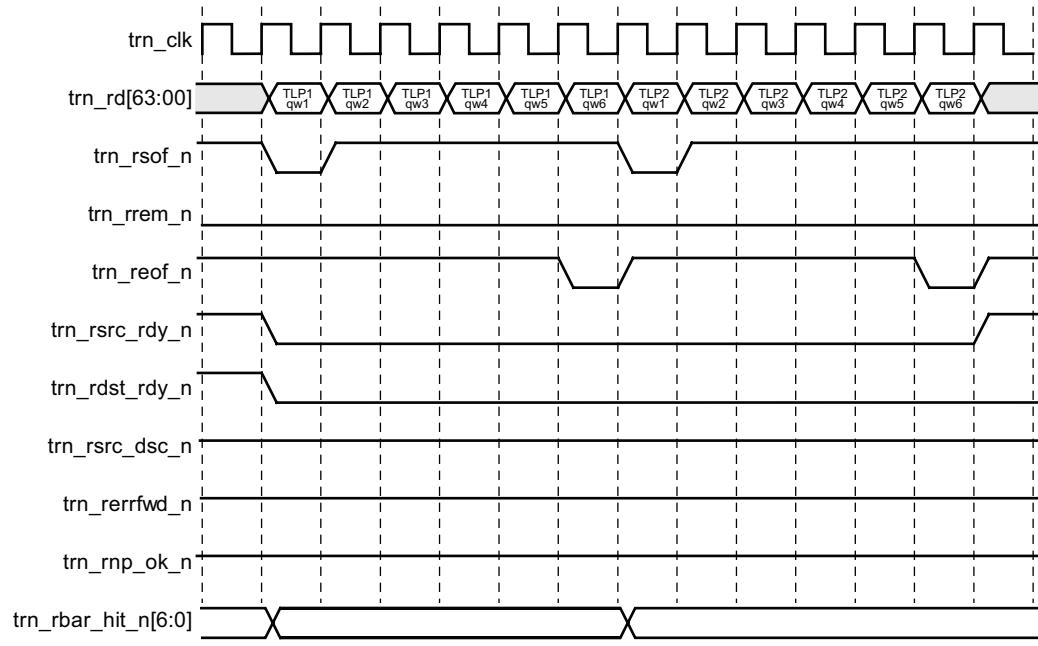


*Figure 6-19:* **Receive Back-to-Back Transactions**

If the User Application cannot accept back-to-back packets, it can stall the transfer of the TLP by deasserting trn_rdst_rdy_n as discussed in the Throttling the Datapath on the Receive Transaction Interface section. Figure 6-20 shows an example of using trn_rdst_rdy_n to pause the acceptance of the second TLP.
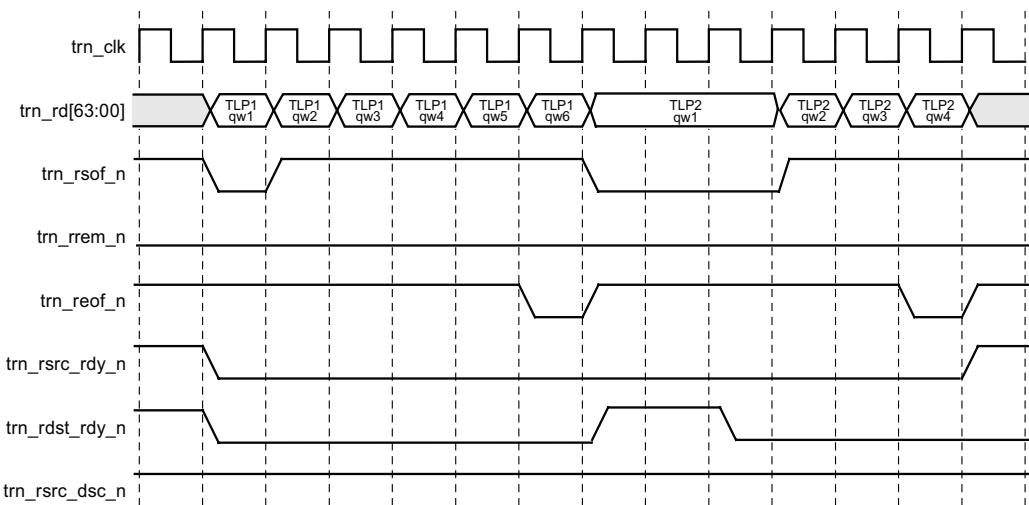


*Figure 6-20:* **User Application Throttling Back-to-Back TLPs**

### Packet Re-ordering on Receive Transaction Interface

Transaction processing in the core receiver is fully compliant with the PCI transaction ordering rules. The transaction ordering rules allow Posted and Completion TLPs to bypass blocked Non-Posted TLPs.

The User Application can deassert trn_rnp_ok_n if it is not ready to accept Non-Posted Transactions from the core, but can receive Posted and Completion Transactions, as shown in Figure 6-21. The User Application must deassert trn_rnp_ok_n at least one clock cycle before trn_eof_n of the next-to-last Non-Posted TLP the user can accept. While trn_rnp_ok_n is deasserted, received Posted and Completion Transactions pass Non-Posted Transactions. After the User Application is ready to accept Non-Posted Transactions, it must reassert trn_rnp_ok_n. Previously bypassed Non-Posted Transactions are presented to the User Application before other received TLPs.
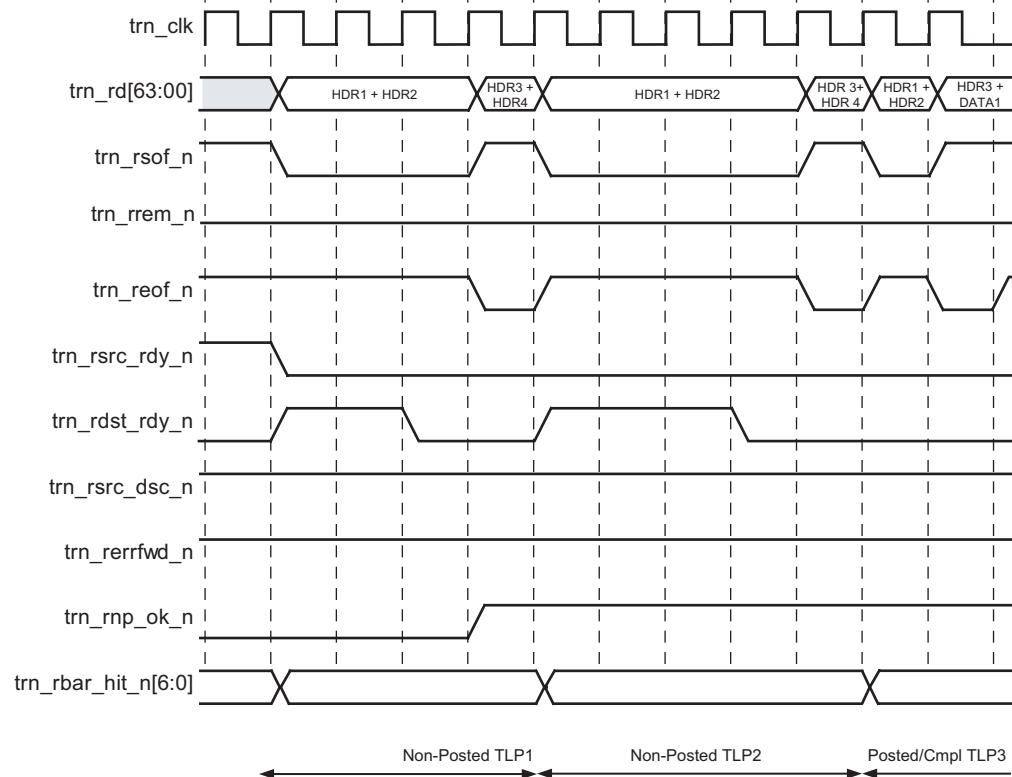
*Figure 6-21:* **Packet Re-ordering on Receive Transaction Interface**

Packet re-ordering allows the User Application to optimize the rate at which Non-Posted TLPs are processed, while continuing to receive and process Posted and Completion TLPs in a non-blocking fashion. The trn_rnp_ok_n signaling restrictions require that the User Application be able to receive and buffer at least three Non-Posted TLPs. This algorithm describes the process of managing the Non-Posted TLP buffers:

Consider that Non-Posted_Buffers_Available denotes the size of Non-Posted buffer space available to User Application. The size of the Non-Posted buffer space is three Non-Posted TLPs. Non-Posted_Buffers_Available is decremented when Non Posted TLP is accepted for processing from the core, and is incremented when Non-Posted TLP is drained for processing by the User Application.

```
For every clock cycle do {
  if (Non-Posted_Buffers_Available <= 2) {
    if (Valid transaction Start-of-Frame accepted by user
                application) {
      Extract TLP Format and Type from the 1st TLP DW
      if (TLP type == Non-Posted) {
        Deassert trn_rnp_ok_n on the following clock
                            cycle
        - or -
        Other optional user policies to stall NP
                              transactions
      } else {
      }
    }
  } else { // Non-Posted_Buffers_Available > 2
    Assert trn_rnp_ok_n on the following clock cycle.
  }
}
```

### Packet Data Poisoning and TLP Digest on Receive Transaction Interface

To simplify logic within the User Application, the core performs automatic pre-processing based on values of TLP Digest (TD) and Data Poisoning (EP) header bit fields on the received TLP.

All received TLPs with the Data Poisoning bit in the header set (EP=1) are presented to the user. The core asserts the trn_rerrfwd_n signal for the duration of each poisoned TLP, as illustrated in Figure 6-47.

If the TLP Digest bit field in the TLP header is set (TD=1), the TLP contains an End-to-End CRC (ECRC). The core performs these operations based on how the user configured the core during core generation:

- If the Trim TLP Digest option is on, the core removes and discards the ECRC field from the received TLP and clears the TLP Digest bit in the TLP header.

- If the Trim TLP Digest option is off, the core does not remove the ECRC field from the received TLP and presents the entire TLP including TLP Digest to the User Application receiver interface.

See Chapter 5, Generating and Customizing the Core, for more information about how to enable the Trim TLP Digest option during core generation.



*Figure 6-22:* **Receive Transaction Data Poisoning**

### Packet Base Address Register Hit on Receive Transaction Interface

The Virtex-6 FPGA Integrated Block for PCI Express in Root Port configuration does not perform any BAR decoding/filtering.

The Virtex-6 FPGA Integrated Block for PCI Express in Endpoint configuration decodes incoming Memory and I/O TLP request addresses to determine which Base Address Register (BAR) in the core's Type0 configuration space is being targeted, and indicates the decoded base address on trn_rbar_hit_n[6:0]. For each received Memory or I/O TLP, a minimum of one and a maximum of two (adjacent) bit(s) are set to 0. If the received TLP targets a 32-bit Memory or I/O BAR, only one bit is asserted. If the received TLP targets a 64-bit Memory BAR, two adjacent bits are asserted. If the core receives a TLP that is not

decoded by one of the BARs (that is, a misdirected TLP), then the core drops it without presenting it to the user and it automatically generates an Unsupported Request message. Even if the core is configured for a 64-bit BAR, the system might not always allocate a 64-bit address, in which case only one trn_rbar_hit_n[6:0] signal is asserted.

Table 6-1 illustrates mapping between trn_rbar_hit_n[6:0] and the BARs, and the corresponding byte offsets in the core Type0 configuration header.

*Table 6-1:* **trn_rbar_hit_n to Base Address Register Mapping**

| trn_rbar_hit_n[x] | BAR | Byte Offset |
|:---:|:---:|:---:|
| 0 | 0 | 10h |
| 1 | 1 | 14h |
| 2 | 2 | 18h |
| 3 | 3 | 1Ch |
| 4 | 4 | 20h |
| 5 | 5 | 24h |
| 6 | Expansion ROM BAR | 30h |

For a Memory or I/O TLP Transaction on the receive interface, trn_rbar_hit_n[6:0] is valid for the entire TLP, starting with the assertion of trn_rsof_n, as shown in Figure 6-23. When receiving non-Memory and non-I/O transactions. The signal trn_rbar_hit_n[6:0] is undefined.
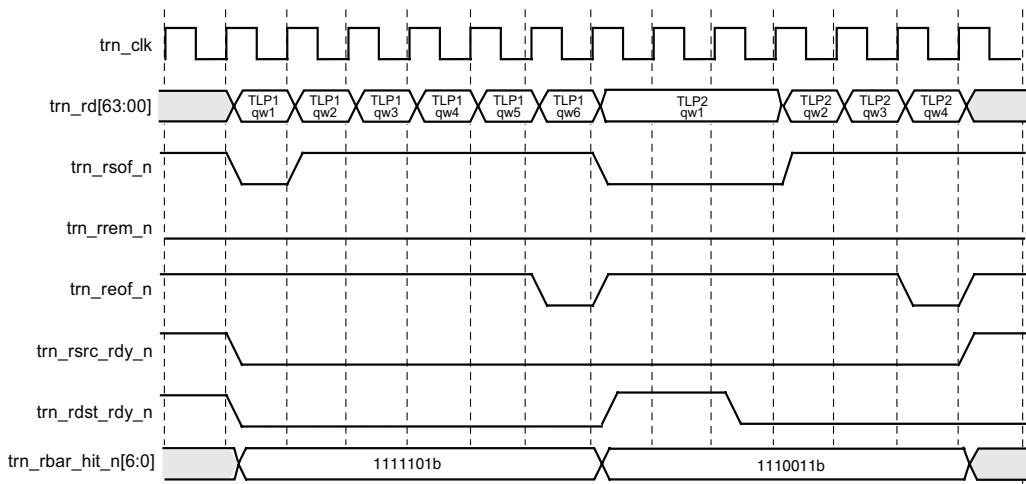


*Figure 6-23:* **BAR Target Determination using trn_rbar_hit**

The signal trn_rbar_hit_n[6:0] enables received Memory and I/O Transactions to be directed to the appropriate destination Memory and I/O apertures in the User Application. By utilizing trn_rbar_hit_n[6:0], application logic can inspect only the lower order Memory and I/O address bits within the address aperture to simplify decoding logic.

### Packet Transfer Discontinue on Receive Transaction Interface

The Endpoint for PCIe asserts trn_rsrc_dsc_n if communication with the link partner is lost, which results in the termination of an *in-progress* TLP. The loss of communication with the link partner is signaled by deassertion of trn_lnk_up_n. When trn_lnk_up_n is deasserted, it effectively acts as a *Hot Reset* to the entire core. For this reason, all TLPs stored inside the core or being presented to the receive interface are irrecoverably lost. Figure 6-24 illustrates packet transfer discontinue scenario.
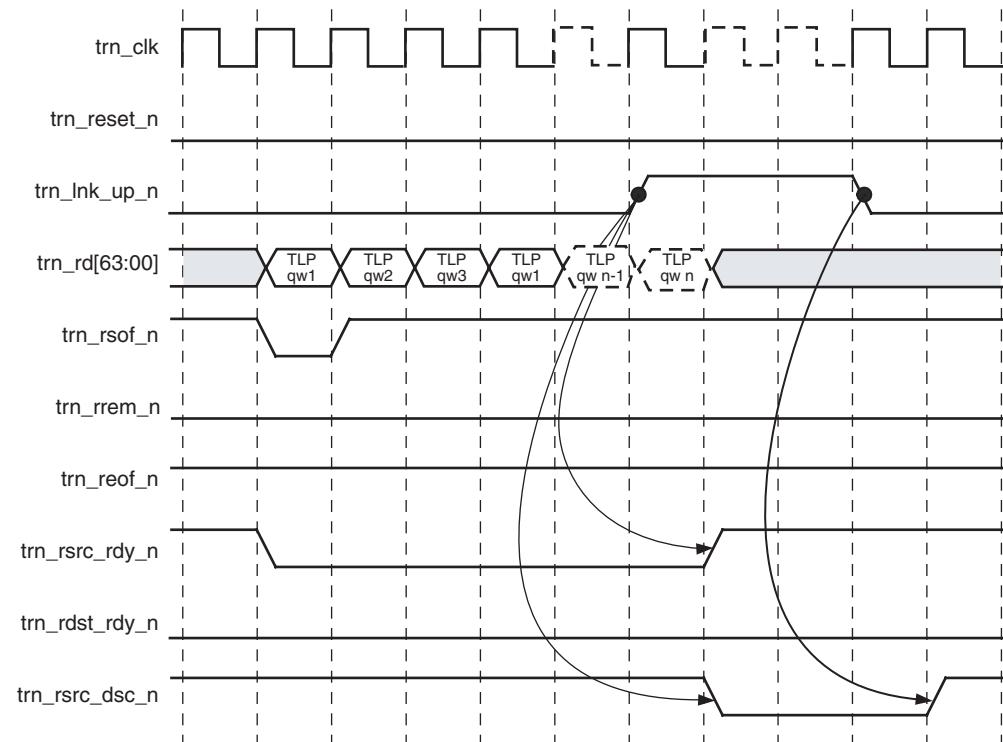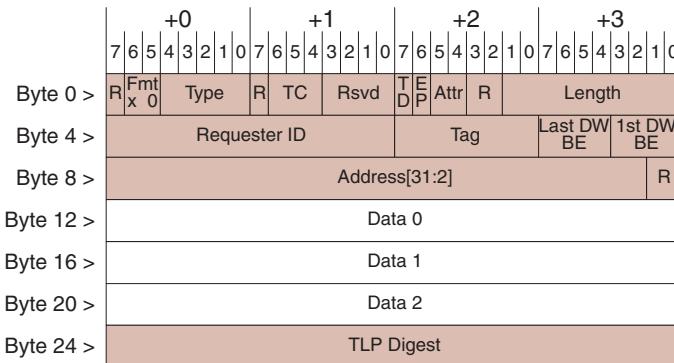


*Figure 6-24:* **Receive Transaction Discontinue**

## Designing with the 128-bit Transaction Layer Interface

***Note:*** This interface is only available in the 8-lane at 5.0 Gb/s Endpoint for PCIe product. The Root Port configuration does not support this interface.
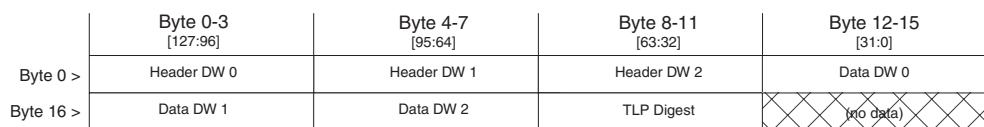
### TLP Format in the Transaction Interface

Data is transmitted and received in Big-Endian order as required by the *PCI Express Base Specification*. See Chapter 2 of the *PCI Express Base Specification* for detailed information about TLP packet ordering. Figure 6-25 represents a typical 32-bit addressable Memory Write Request TLP (as illustrated in Chapter 2 of the specification).



*Figure 6-25:* **PCI Express Base Specification Byte Order**

When using the Transaction interface, packets are arranged on the entire 128-bit datapath. Figure 6-26 shows the same example packet on the Transaction interface. Byte 0 of the packet appears on trn_td[127:120] (outbound) or trn_rd[127:120] (inbound) of the first DWORD, byte 1 on trn_td[119:112] or trn_rd[119:112], and so forth. The Header section of the packet consists of either three or four DWORDs, determined by the TLP format and type as described in section 2.2 of the *PCI Express Base Specification*.



*Figure 6-26:* **Endpoint Integrated Block Byte Order**

Packets sent to the core for transmission must follow the formatting rules for Transaction Layer Packets (TLPs) as specified in Chapter 2 of the *PCI Express Base Specification*. The User Application is responsible for ensuring its packets' validity, as the core does not check packet validity or validate packets. The exact fields of a given TLP vary depending on the type of packet being transmitted.

The core allows the User Application to add an extra level of error checking by using the optional TLP Digest Field in the TLP header. The presence of a TLP Digest or ECRC is indicated by the value of TD field in the TLP Header section. When TD=1, a correctly computed CRC32 remainder is expected to be presented as the last DWORD of the packet. The CRC32 remainder DWORD is not included in the length field of the TLP header. The User Application must calculate and present the TLP Digest as part of the packet when transmitting packets. Upon receiving packets with a TLP Digest present, the User Application must check the validity of the CRC32 based on the contents of the packet. The core does not check the TLP Digest for incoming packets. The *PCI Express Base Specification*

requires Advanced Error Reporting (AER) capability when implementing ECRC. Although the integrated block does not support AER, users can still implement ECRC for custom solutions that do not require compliance with the *PCI Express Base Specification*.

## Transmitting Outbound Packets

### Basic TLP Transmit Operation

The Virtex-6 FPGA Integrated Block for PCI Express core automatically transmits these types of packets:

- Completions to a remote device in response to Configuration Space requests.
- Error-message responses to inbound requests that are malformed or unrecognized by the core.

   ***Note:*** Certain unrecognized requests, for example, unexpected completions, can only be detected by the User Application, which is responsible for generating the appropriate response.

The User Application is responsible for constructing these types of outbound packets:

- Memory and I/O Requests to remote devices.
- Completions in response to requests to the User Application, for example, a Memory Read Request.

Table 2-12, page 34 defines the transmit User Application signals. To transmit a TLP, the User Application must perform this sequence of events on the transmit Transaction interface:

1. The User Application logic asserts trn_tsrc_rdy_n, trn_tsof_n, and presents the first TLP DQWORD on trn_td[127:0] when it is ready to transmit data. If the core is asserting trn_tdst_rdy_n, the DQWORD is accepted immediately; otherwise, the User Application must keep the DQWORD presented until the core asserts trn_tdst_rdy_n.

2. The User Application asserts trn_tsrc_rdy_n and presents the remainder of the TLP DQWORDs on trn_td[127:0] for subsequent clock cycles (for which the core asserts trn_tdst_rdy_n).

3. The User Application asserts trn_tsrc_rdy_n and trn_teof_n together with the last DQWORD data. The user must ensure that the remainder field is selected for the final data cycle to create a packet of length equivalent to the length field in the packet header. For more information on the trn_trem_n[1:0] signalling, refer to Table 6-11 and Table 6-14.

4. At the next clock cycle, the User Application deasserts trn_tsrc_rdy_n to signal the end of valid transfers on trn_td[127:0].

This section uses the notation H*n* and D*n* to denote Header QW*n* and Data QW*n*, respectively. Table 6-2 lists the possible single-cycle packet signaling where trn_tsof_n and trn_teof_n are asserted in the same cycle.

*Table 6-2:* **TX: EOF Scenarios, Single-Cycle**

| | trn_td[127:0] | | |
|---|---|---|---|
| | **H0 H1 H2 H3** | **H0 H1 H2 --** | **H0 H1 H2 D0** |
| trn_tsof_n | 0 | 0 | 0 |
| trn_teof_n | 0 | 0 | 0 |
| trn_trem_n[1] | 0 | 0 | 0 |
| trn_trem_n[0] | 0 | 1 | 0 |

Table 6-3 lists the possible signaling for ending a multicycle packet. If a packet ends in the upper QW of the data bus, the next packet cannot start in the lower QW of that beat. All packets must start in the upper QW of the data bus. trn_trem_n[1] indicates whether the EOF occurs in the upper or lower QW of the data bus.

*Table 6-3:* **TX: EOF Scenarios, Multi-Cycle**

|  | trn_td[127:0] | | | |
|---|---|---|---|---|
|  | **D0 D1 D2 D3** | **D0 D1 D2 --** | **D0 D1 -- --** | **D0 -- -- --** |
| trn_tsof_n | 1 | 1 | 1 | 1 |
| trn_teof_n | 0 | 0 | 0 | 0 |
| trn_trem_n[1] | 0 | 0 | 1 | 1 |
| trn_trem_n[0] | 0 | 1 | 0 | 1 |

Figure 6-27 illustrates a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request. When the User Application asserts trn_teof_n, it also places a value of `01b` on trn_trem_n[1:0], notifying the core that only trn_td[127:32] contains valid data.
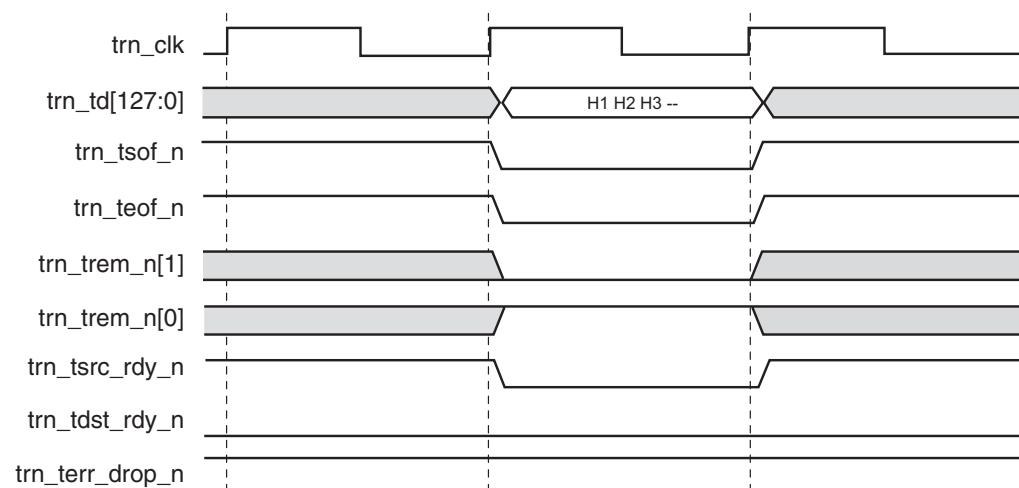


*Figure 6-27:* **TLP 3-DW Header without Payload**

Figure 6-28 illustrates a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request. When the User Application asserts trn_teof_n, it also places a value of `00b` on trn_trem_n[1:0] notifying the core that trn_td[127:0] contains valid data and the EOF occurs in the lower QW.
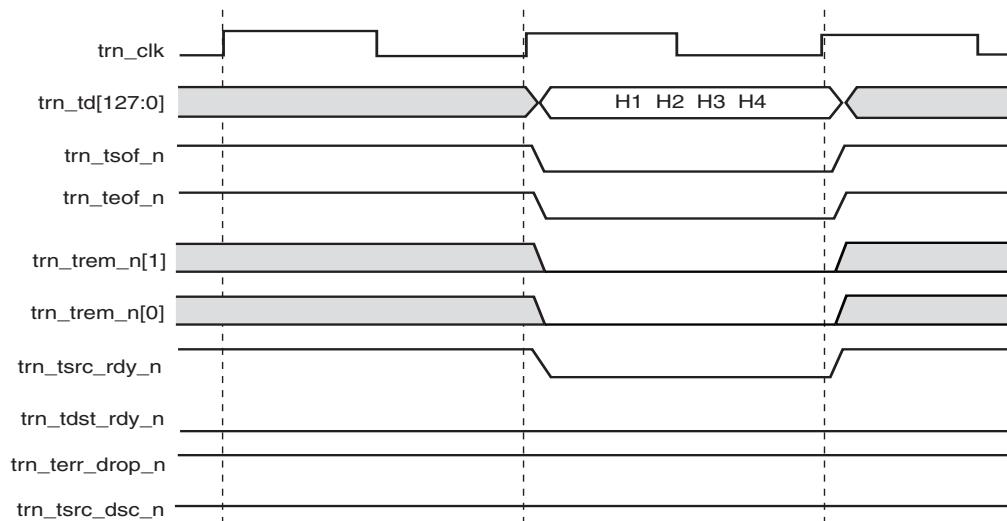


*Figure 6-28:*  **TLP with 4-DW Header without Payload**

Figure 6-29 illustrates a 3-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request. When the User Application asserts trn_teof_n, it also puts a value of `10b` on trn_trem_n[1:0] notifying the core that trn_td[127:64] contains valid data and the EOF occurs in the upper QW. The user must ensure the remainder field selected for the final data cycle creates a packet of length equivalent to the length field in the packet header.
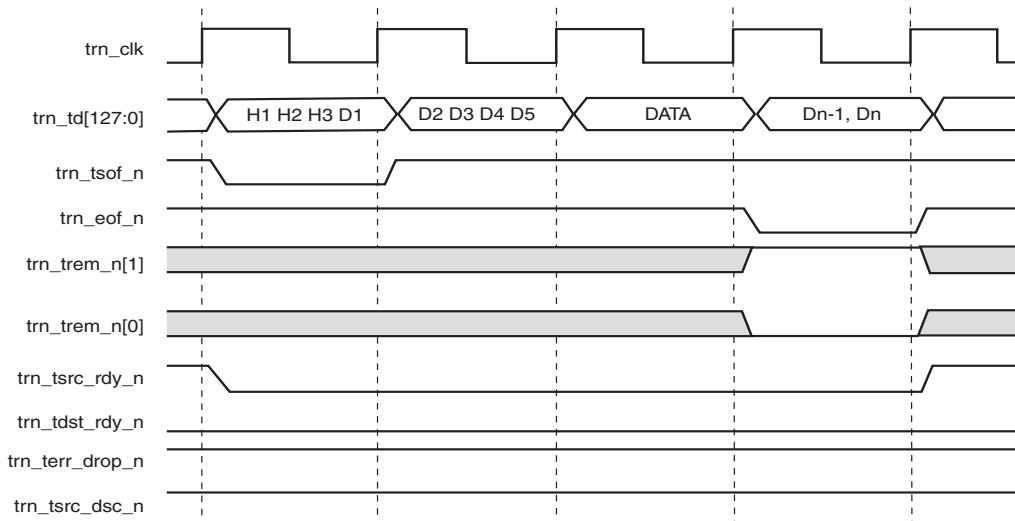


*Figure 6-29:*  **TLP with 3-DW Header with Payload**

Figure 6-30 illustrates a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request. When the User Application asserts trn_teof_n, it also places a value of `01b` on trn_trem_n[1:0], notifying the core that only trn_td[127:64] contains valid data. The user must ensure the remainder field is selected for the final data cycle creates a packet of length equivalent to the length field in the packet header.
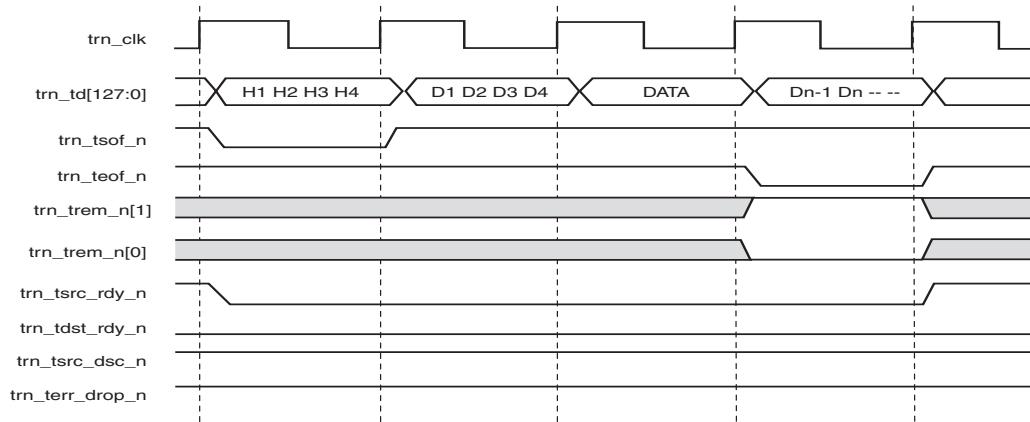


*Figure 6-30:* **TLP with 4-DW Header with Payload**

## Presenting Back-to-Back Transactions on the Transmit Interface

The User Application can present back-to-back TLPs on the transmit Transaction interface to maximize bandwidth utilization. Figure 6-31 illustrates back-to-back TLPs presented on the transmit interface, with the restriction that all TLPs must start in the upper QW of the data bus [127:64]. The User Application asserts trn_tsof_n and presents a new TLP on the next clock cycle after asserting trn_teof_n for the previous TLP.
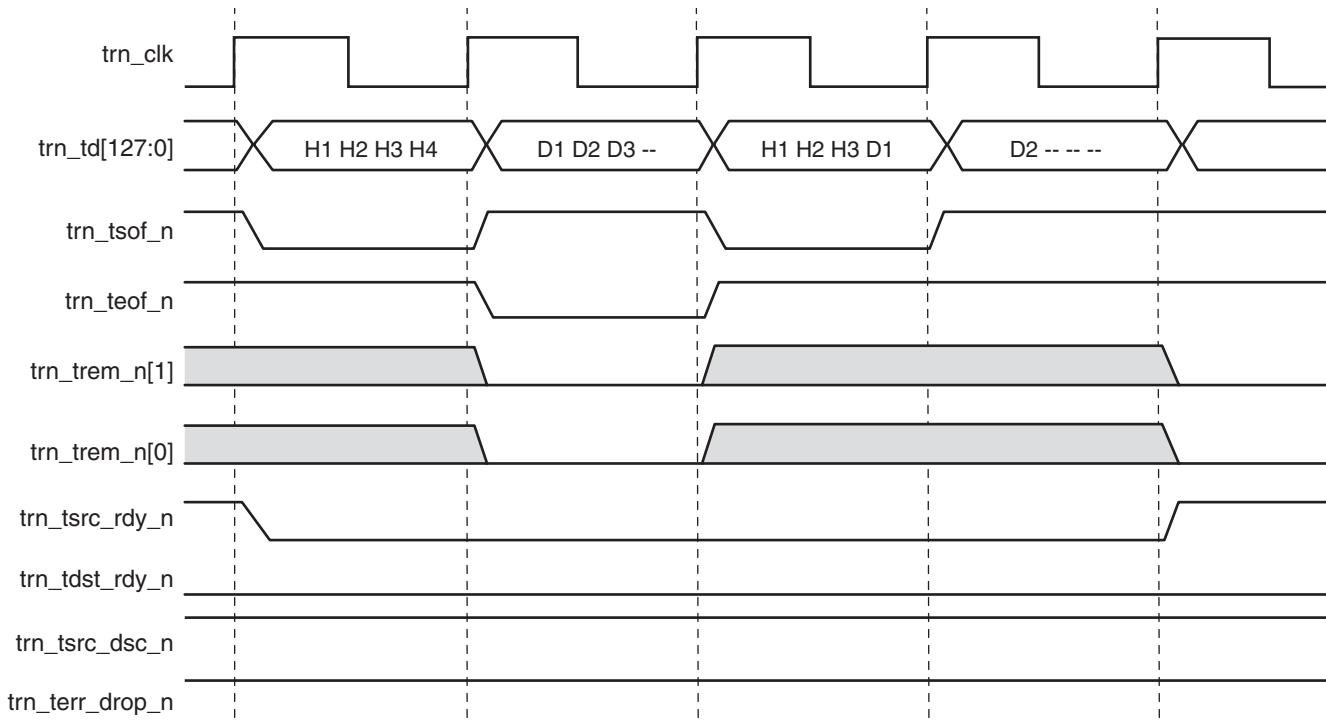


UG517_c6_31_020210

*Figure 6-31:* **Back-to-Back Transaction on Transmit Transaction Interface**

## Source Throttling on the Transmit Datapath

The Transaction interface lets the User Application throttle back if it has no data to present on trn_td[127:0]. When this condition occurs, the User Application deasserts trn_tsrc_rdy_n, which instructs the core Transaction interface to disregard data presented on trn_td[127:0]. Figure 6-32 illustrates the source throttling mechanism, where the User Application does not have data to present every clock cycle, and for this reason must deassert trn_tsrc_rdy_n during these cycles.
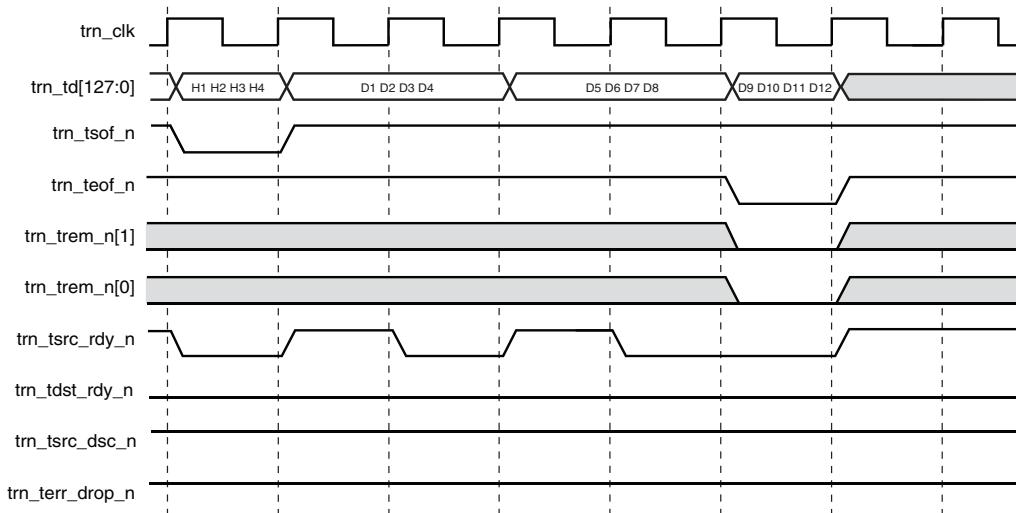


*Figure 6-32:* **Source Throttling on the Transmit Datapath**

### Destination Throttling of the Transmit Datapath

The core Transaction interface throttles the transmit User Application if there is no space left for a new TLP in its transmit buffer pool. This can occur if the link partner is not processing incoming packets at a rate equal to or greater than the rate at which the User Application is presenting TLPs. Figure 6-33 illustrates the deassertion of trn_tdst_rdy_n to throttle the User Application when the core's internal transmit buffers are full. If the core needs to throttle the User Application, it does so after the current packet has completed. If another packet starts immediately after the current packet, the throttle occurs in the new packet's SOF.
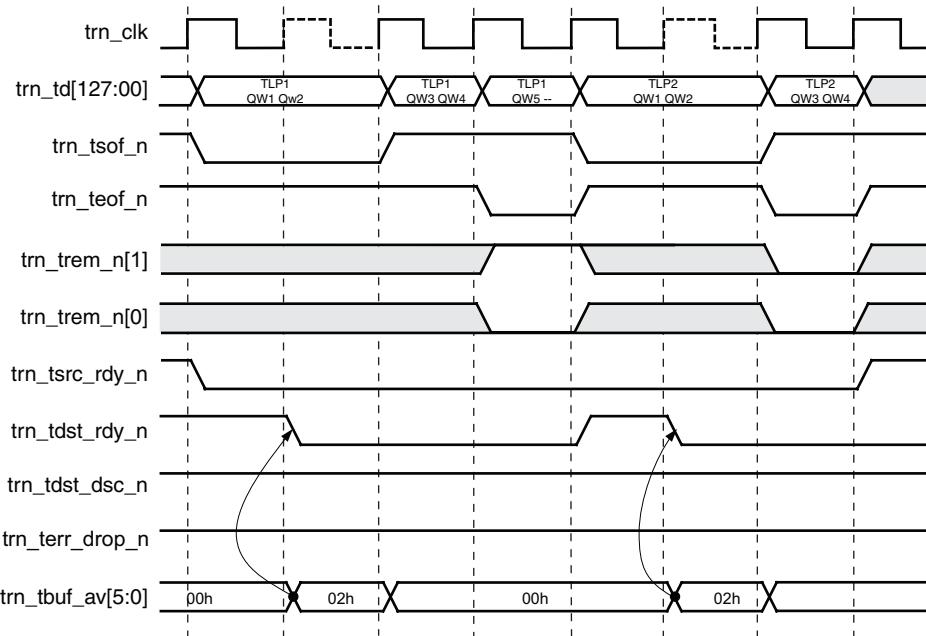


*Figure 6-33:* **Destination Throttling of the Endpoint Transmit Transaction Interface**

If the core transmit Transaction interface accepts the start of a TLP by asserting trn_tdst_rdy_n, it is guaranteed to accept the complete TLP with a size up to the value contained in the Max_Payload_Size field of the PCI Express Device Capability Register (offset 04H). To stay compliant to the *PCI Express Base Specification* users should not violate the Max_Payload_Size field of the PCI Express Device Control Register (offset 08H). The core transmit Transaction interface deasserts trn_tdst_rdy_n only under these conditions:

- After it has accepted the TLP completely and has no buffer space available for a new TLP.

- When the core is transmitting an internally generated TLP (configuration Completion TLP, error Message TLP or error response as requested by the User Application on the cfg_err interface).

On deassertion of trn_tdst_rdy_n by the core, the User Application needs to hold all control and data signals until the core asserts trn_tdst_rdy_n.

The core transmit Transaction interface throttles the User Application when the Power State field in Power Management Control/Status Register (offset 0x4) of the PCI Power Management Capability Structure is changed to a non-D0 state. When this occurs, any ongoing TLP is accepted completely and trn_tdst_rdy_n is subsequently deasserted, disallowing the User Application from initiating any new transactions—for the duration that the core is in the non-D0 power state.

## Discontinuing Transmission of Transaction by Source

The core Transaction interface lets the User Application terminate transmission of a TLP by asserting trn_tsrc_dsc_n. Both trn_tsrc_rdy_n and trn_tdst_rdy_n must be asserted together with trn_tsrc_dsc_n for the TLP to be discontinued. The signal trn_tsrc_dsc_n must not be asserted together with trn_tsof_n. It can be asserted on any cycle after trn_sof_n deasserts up to and including the assertion of trn_teof_n. Asserting trn_tsrc_dsc_n has no effect if no TLP transaction is in progress on the transmit interface. Figure 6-34 illustrates the User Application discontinuing a packet using trn_tsrc_dsc_n. Asserting trn_teof_n together with trn_tsrc_dsc_n is optional.
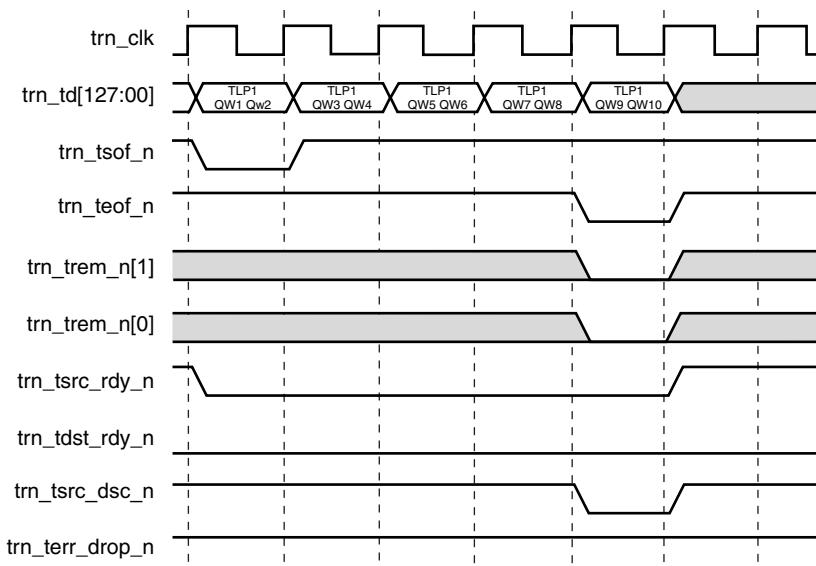


*Figure 6-34:* **Source Driven Transaction Discontinue on Transmit Interface**

Discarding of Transaction by Destination

The core transmit Transaction interface discards a TLP for three reasons:

- PCI Express Link goes down.

- Presented TLP violates the Max_Payload_Size field of the Device Capability Register (offset 04H) for PCI Express. It is the user's responsibility to not violate the Max_Payload_Size field of the Device Control Register (offset 08H).

- trn_tstr_n is asserted and data is not presented on consecutive clock cycles, that is, trn_tsrc_rdy_n is deasserted in the middle of a TLP transfer

When any of these occur, the transmit Transaction interface continues to accept the remainder of the presented TLP and asserts trn_terr_drop_n no later than the third clock cycle following the EOF of the discarded TLP. Figure 6-35 illustrates the core signaling that a packet was discarded using trn_terr_drop_n.



UG517_c6_35_020210
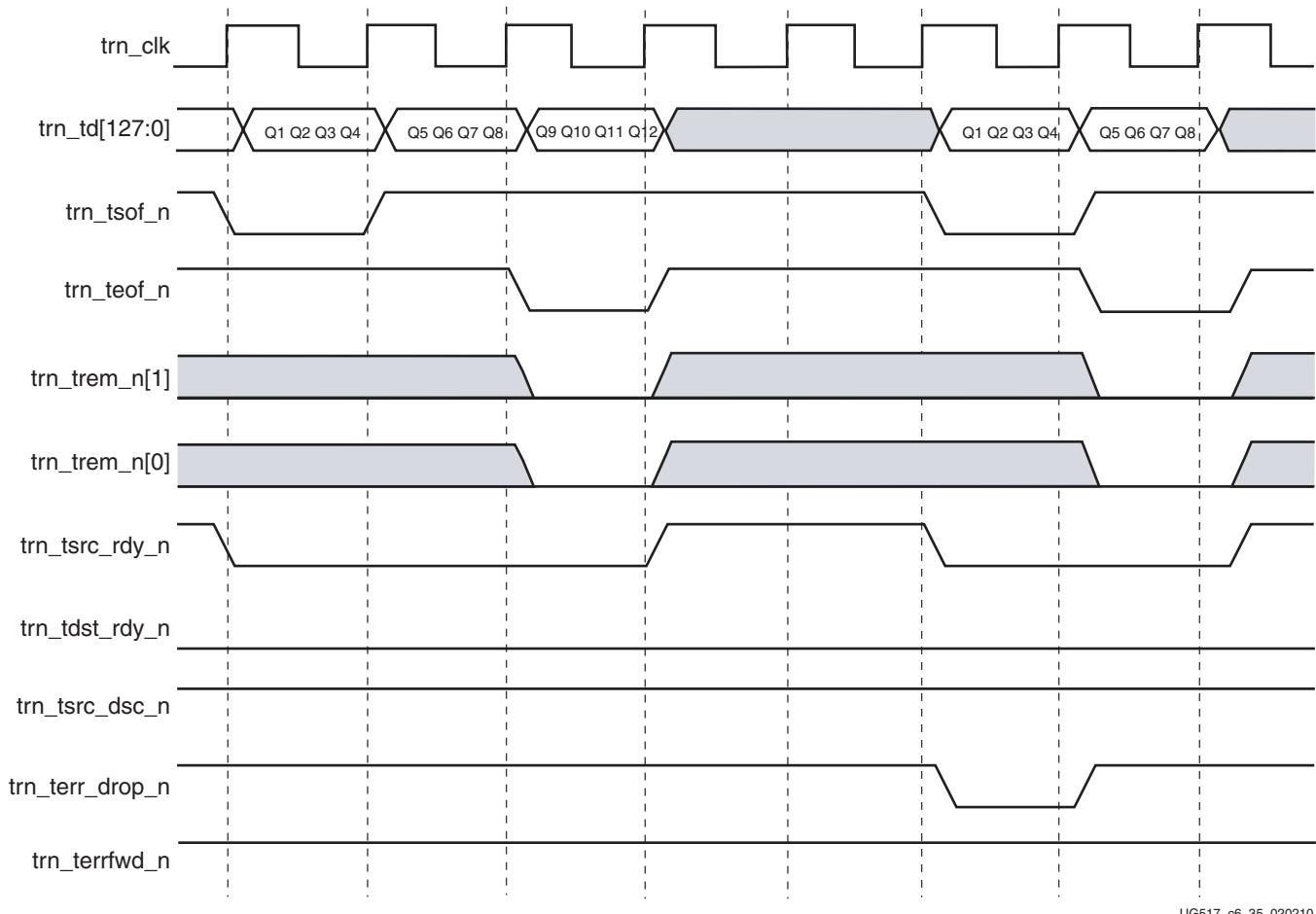
*Figure 6-35:* **Discarding of Transaction by Destination on Transmit Interface**

Packet Data Poisoning on the Transmit Transaction Interface

The User Application uses the following mechanism to mark the data payload of a transmitted TLP as poisoned:

- Set EP = 1 in the TLP header. This mechanism can be used if the payload is known to be poisoned when the first DWORD of the header is presented to the core on the TRN interface.

- Assert trn_terr_fwd_n for at least 1 valid data transfer cycle any time during the packet transmission, as shown in Figure 6-36. This causes the core to set EP = 1 in the TLP header when it transmits the packet onto the PCI Express fabric. This mechanism can be used if the User Application does not know whether a packet could be poisoned at the start of packet transmission. Use of trn_terrfwd_n is not supported for packets when trn_tstr_n is asserted (streamed transmit packets).
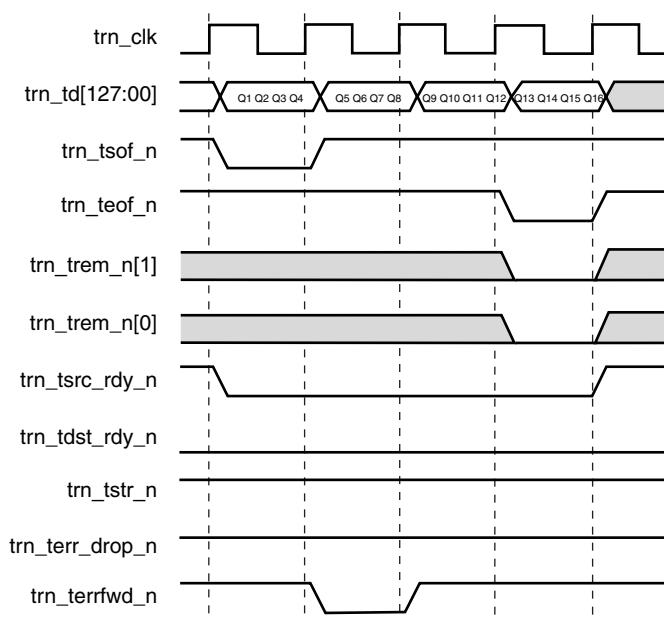


*Figure 6-36:* **Packet Data Poisoning on the Transmit Transaction Interface**

### Streaming Mode for Transactions on the Transmit Interface

The Virtex-6 FPGA Integrated Block for PCI Express core allows the User Application to enable Streaming (cut-through) mode for transmission of a TLP, when possible, to reduce latency of operation. To enable this feature, the User Application must assert trn_tstr_n for the entire duration of the transmitted TLP. In addition, the User Application must present valid frames on every clock cycle until the final cycle of the TLP. In other words, the User Application must not deassert trn_tsrc_rdy_n for the duration of the presented TLP. Source throttling of the transaction while in streaming mode of operation causes the transaction to be dropped (trn_terr_drop_n is asserted) and a nullified TLP to be signaled on the PCI Express link. Figure 6-37 illustrates the streaming mode of operation, where the first TLP is streamed and the second TLP is dropped due to source throttling.



UG517_c6_37_020810
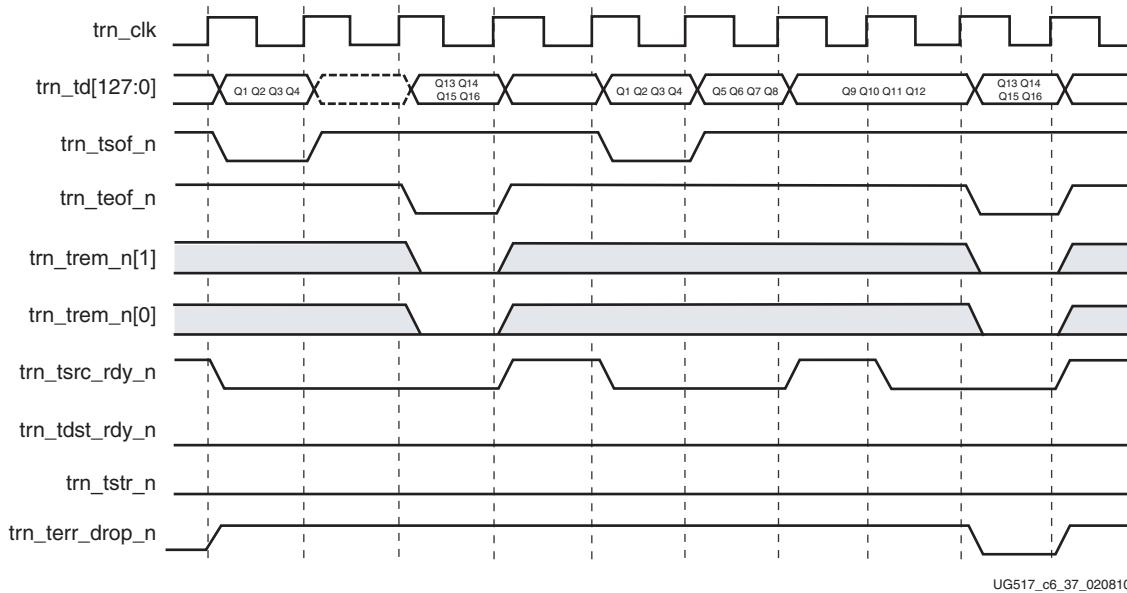
*Figure 6-37:* **Streaming Mode on the Transmit Interface**

**Note:** Source Driven Transaction Discontinue (assertion of trn_tsrc_dsc_n) is not supported when in the streaming mode of operation.

### Appending ECRC to Protect TLPs

If the User Application needs to send a TLP Digest associated with a TLP, it must construct the TLP header such that the TD bit is set and the User Application must properly compute and append the 1-DWORD TLP Digest after the last valid TLP payload section (if applicable). TLPs originating within the core do not have a TLP Digest appended, for example Completions, Error Messages, and Interrupts.

## Receiving Inbound Packets

### Basic TLP Receive Operation

Table 2-13, page 36 defines the receive Transaction interface signals. This sequence of events must occur on the receive Transaction interface for the Endpoint core to present a TLP to the User Application logic:

1. When the User Application is ready to receive data, it asserts trn_rdst_rdy_n.

2. When the core is ready to transfer data, the core asserts trn_rsrc_rdy_n with trn_rsof_n and presents the first complete TLP DQWORD on trn_rd[127:0].

3. The core then deasserts trn_rsof_n, asserts trn_rsrc_rdy_n, and presents TLP DQWORDs on trn_rd[127:0] for subsequent clock cycles, for which the User Application logic asserts trn_rdst_rdy_n.

4. If no further TLPs are available at the next clock cycle, the core deasserts trn_rsrc_rdy_n to signal the end of valid transfers on trn_rd[127:0].

**Note**: The User Application should ignore any assertions of trn_rsof_n, trn_reof_n, trn_rrem_n, and trn_rd unless trn_rsrc_rdy_n is concurrently asserted.

Table 6-4 and Table 6-5 list the signaling for all the valid cases where a packet can end within a single beat. These tables use the notation H*n* and D*n* to denote Header QW *n* and Data QW *n* respectively. Table 6-4 lists the possible single-cycle packet signaling where trn_rsof_n and trn_reof_n are asserted in the same cycle.

*Table 6-4:* **Single-Cycle TLP Scenarios (Header and Data)**

| | trn_rd[127:0] | | |
| --- | --- | --- | --- |
| | **H0 H1 H2 H3** | **H0 H1 H2 --** | **H0 H1 H2 D0** |
| trn_rsof_n | 0 | 0 | 0 |
| trn_reof_n | 0 | 0 | 0 |
| trn_rrem_n[1] | 0 | 0 | 0 |
| trn_rrem_n[0] | 0 | 1 | 0 |

Table 6-5 lists the possible signaling for ending a multicycle packet. If a packet ends in the upper QW of the data bus, the next packet can start in the lower QW of that beat. trn_trem_n[1] indicates whether the EOF occurs in the upper or lower QW of the data bus.

*Table 6-5:* **Receive - EOF Scenarios (Data)**

| | trn_rd[127:0] | | | |
| --- | --- | --- | --- | --- |
| | **D0 D1 D2 D3** | **D0 D1 D2 --** | **D0 D1 -- --** | **D0 -- -- --** |
| trn_rsof_n | 1 | 1 | 1 | 1 |
| trn_reof_n | 0 | 0 | 0 | 0 |
| trn_rrem_n[1] | 0 | 0 | 1 | 1 |
| trn_rrem_n[0] | 0 | 1 | 0 | 1 |

Table 6-6 lists the possible signaling for a straddled data transfer beat. A straddled data transfer beat occurs when one packet ends in the upper QW and a new packet starts in the lower QW of the same cycle. Straddled data transfers only occur in the receive direction. A packet can start in the lower QW without having a packet in the upper QW.

*Table 6-6:* **Receive - Straddle Cases**

|  | trn_rd[127:0] | | |
| --- | --- | --- | --- |
|  | **D0 D1 H0 H1** | **D0 – H0 H1** | **-- -- H0 H1** |
| trn_rsof_n | 0 | 0 | 0 |
| trn_reof_n | 0 | 0 | 1 |
| trn_rrem_n[1] | 1 | 1 | 1 |
| trn_rrem_n[0] | 0 | 1 | - |

Figure 6-38 shows a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request. When the core asserts trn_reof_n, it also places a value of `01b` on trn_rrem_n, notifying the user that only trn_rd[127:32] contains valid data.
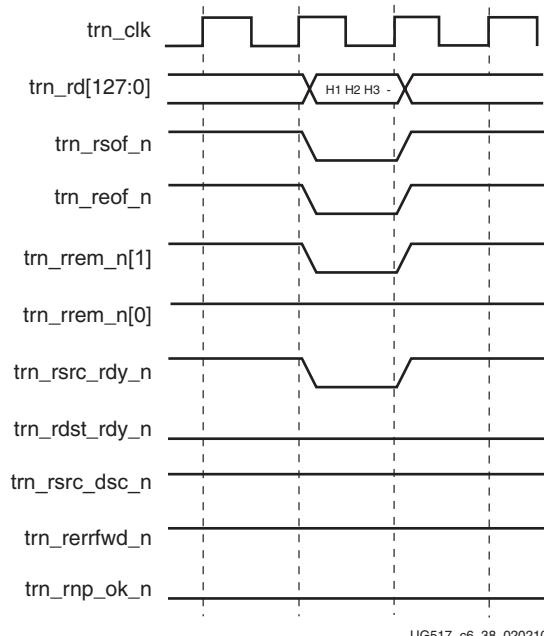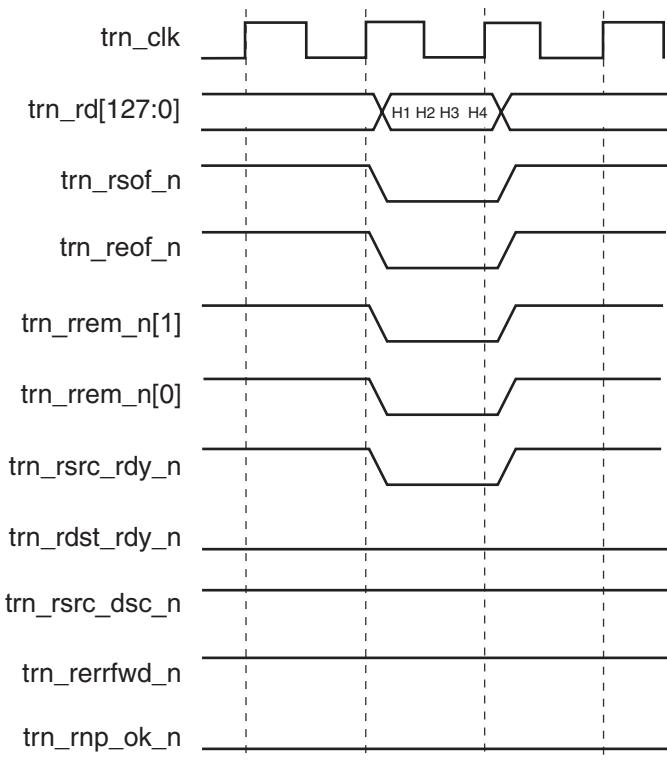


*Figure 6-38:* **TLP 3-DW Header without Payload**

Figure 6-39 shows a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request. When the core asserts trn_reof_n, it also places a value of `00b` on trn_rrem_n, notifying the user that trn_rd[127:0] contains valid data.



*Figure 6-39:* **TLP 4-DW Header without Payload**

Figure 6-40 shows a 3-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request. When the core asserts trn_reof_n, it also places a value of `00b` on trn_rrem_n, notifying the user that trn_rd[127:0] contains valid data.
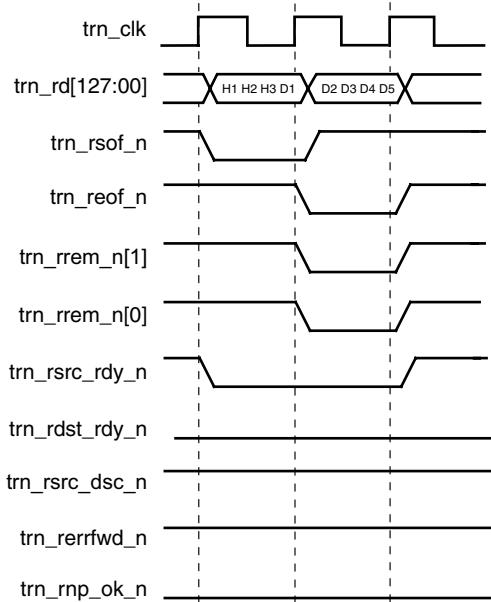


*Figure 6-40:* **TLP 3-DW Header with Payload**

Figure 6-41 shows a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request. When the core asserts trn_reof_n, it also places a value of `11b` on trn_rrem_n, notifying the user that only trn_rd[127:96] contains valid data.
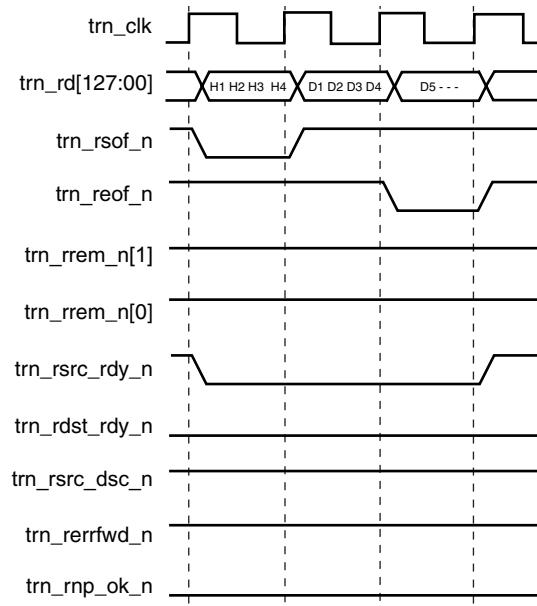


*Figure 6-41:* **TLP 4-DW Header with Payload**

## Throttling the Datapath on the Receive Transaction Interface

The User Application can stall the transfer of data from the core at any time by deasserting trn_rdst_rdy_n. If the user deasserts trn_rdst_rdy_n while no transfer is in progress and if a TLP becomes available, the core asserts trn_rsrc_rdy_n and trn_rsof_n and presents the first TLP DQWORD on trn_rd[127:0]. The core remains in this state until the user asserts trn_rdst_rdy_n to signal the acceptance of the data presented on trn_rd[127:0]. At that point, the core presents subsequent TLP DQWORDs as long as trn_rdst_rdy_n remains asserted. If the user deasserts trn_rdst_rdy_n during the middle of a transfer, the core stalls the transfer of data until the user asserts trn_rdst_rdy_n again. There is no limit to the number of cycles the user can keep trn_rdst_rdy_n deasserted. The core pauses until the user is again ready to receive TLPs.

Figure 6-42 illustrates the core asserting trn_rsrc_rdy_n and trn_rsof_n along with presenting data on trn_rd[127:0]. The User Application logic inserts wait states by deasserting trn_rdst_rdy_n. The core does not present the next TLP DQWORD until it detects trn_rdst_rdy_n assertion. The User Application logic can assert or deassert trn_rdst_rdy_n as required to balance receipt of new TLP transfers with the rate of TLP data processing inside the application logic.
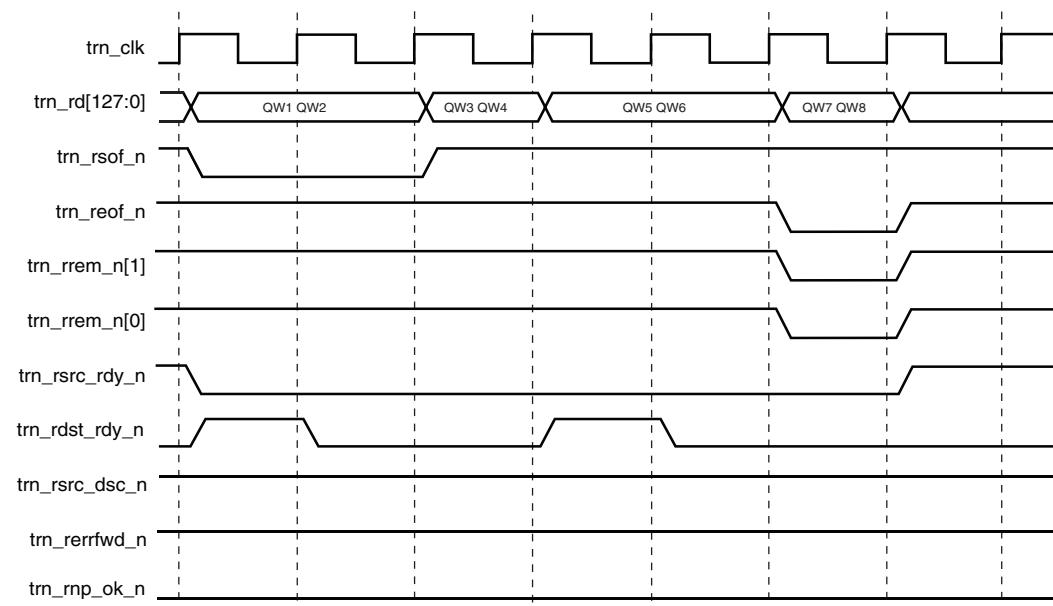


*Figure 6-42:* **User Application Throttling Receive TLP**

### Receiving Back-to-Back Transactions on the Receive Transaction Interface

The User Application logic must be designed to handle presentation of back-to-back TLPs on the receive Transaction Interface by the core. The core can assert trn_rsof_n for a new TLP at the clock cycle after trn_reof_n assertion for the previous TLP. Figure 6-43 illustrates back-to-back TLPs presented on the receive interface.
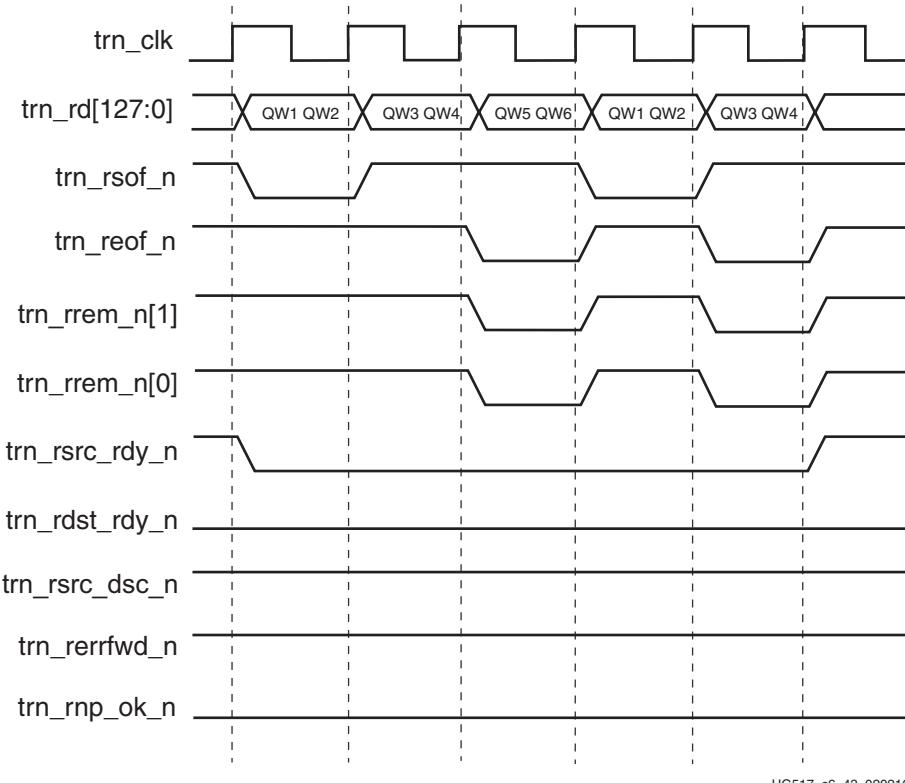


*Figure 6-43:* **Receive Back-to-Back Transactions**

If the User Application cannot accept back-to-back packets, it can stall the transfer of the TLP by deasserting trn_rdst_rdy_n as discussed in the Throttling the Datapath on the Receive Transaction Interface section. Figure 6-44 shows an example of using trn_rdst_rdy_n to pause the acceptance of the second TLP.



UG517_c6_44_020210
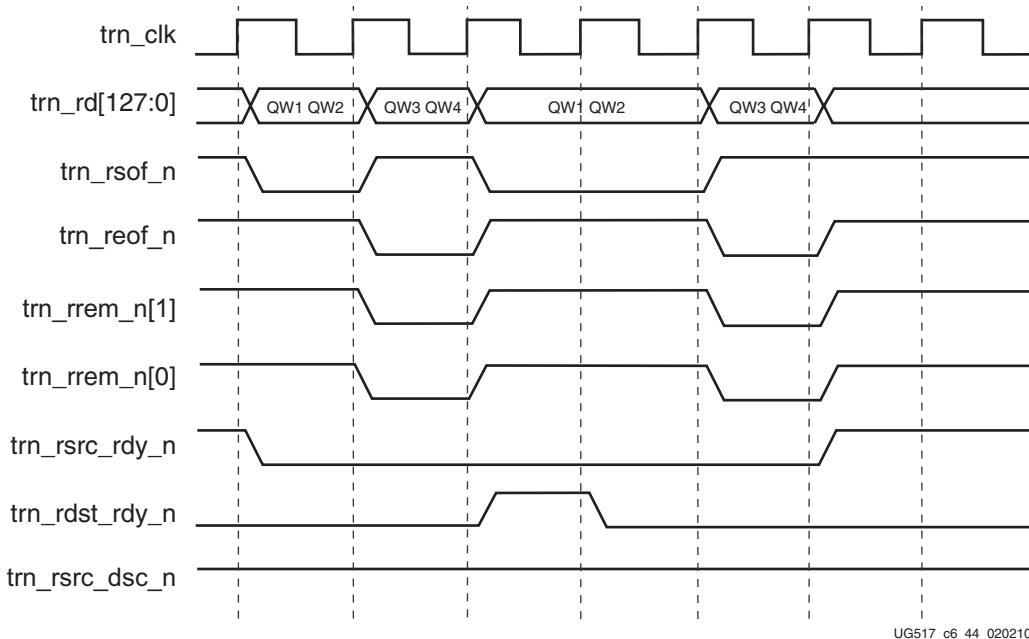
*Figure 6-44:*    **User Application Throttling Back-to-Back TLPs**

### Receiving Straddled Packets on the Receive Transaction Interface

The User Application logic must be designed to handle presentation of straddled TLPs on the receive Transaction Interface by the core. The core can assert trn_rsof_n for a new TLP on the same clock cycle as the trn_reof_n for the previous TLP, when the previous TLP ends in the upper QW. Figure 6-45 illustrates straddled TLPs presented on the receive interface.
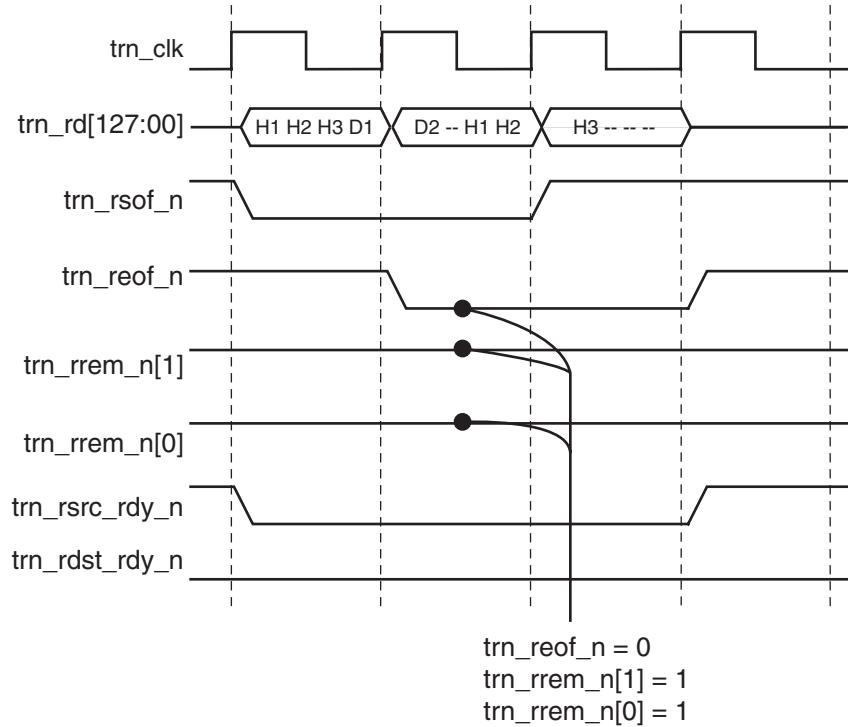
*Figure 6-45:* **Receive Straddled Transactions**

In Figure 6-45, the first packet is a 3 DW packet with 64 bits of data and the second packet is a 3 DW packet which begins on the lower QWORD portion of the bus. In the figure, assertion of trn_reof_n and trn_rrem_n[1] = 1'b1 indicates that the EOF of the previous occurs in bits [127:64]. Simultaneous deassertion of trn_rrem_n[0] (1'b1) indicates that only bits [127:96] are valid.

## Packet Re-ordering on Receive Transaction Interface

Transaction processing in the core receiver is fully compliant with the PCI transaction ordering rules. The transaction ordering rules allow Posted and Completion TLPs to bypass blocked Non-Posted TLPs.

The User Application can deassert trn_rnp_ok_n if it is not ready to accept Non-Posted Transactions from the core, but can receive Posted and Completion Transactions, as shown in Figure 6-46. The User Application must deassert trn_rnp_ok_n at least one clock cycle before trn_eof_n of the next-to-last Non-Posted TLP the user can accept. While trn_rnp_ok_n is deasserted, received Posted and Completion Transactions pass Non-Posted Transactions. After the User Application is ready to accept Non-Posted Transactions, it must reassert trn_rnp_ok_n. Previously bypassed Non-Posted Transactions are presented to the User Application before other received TLPs.
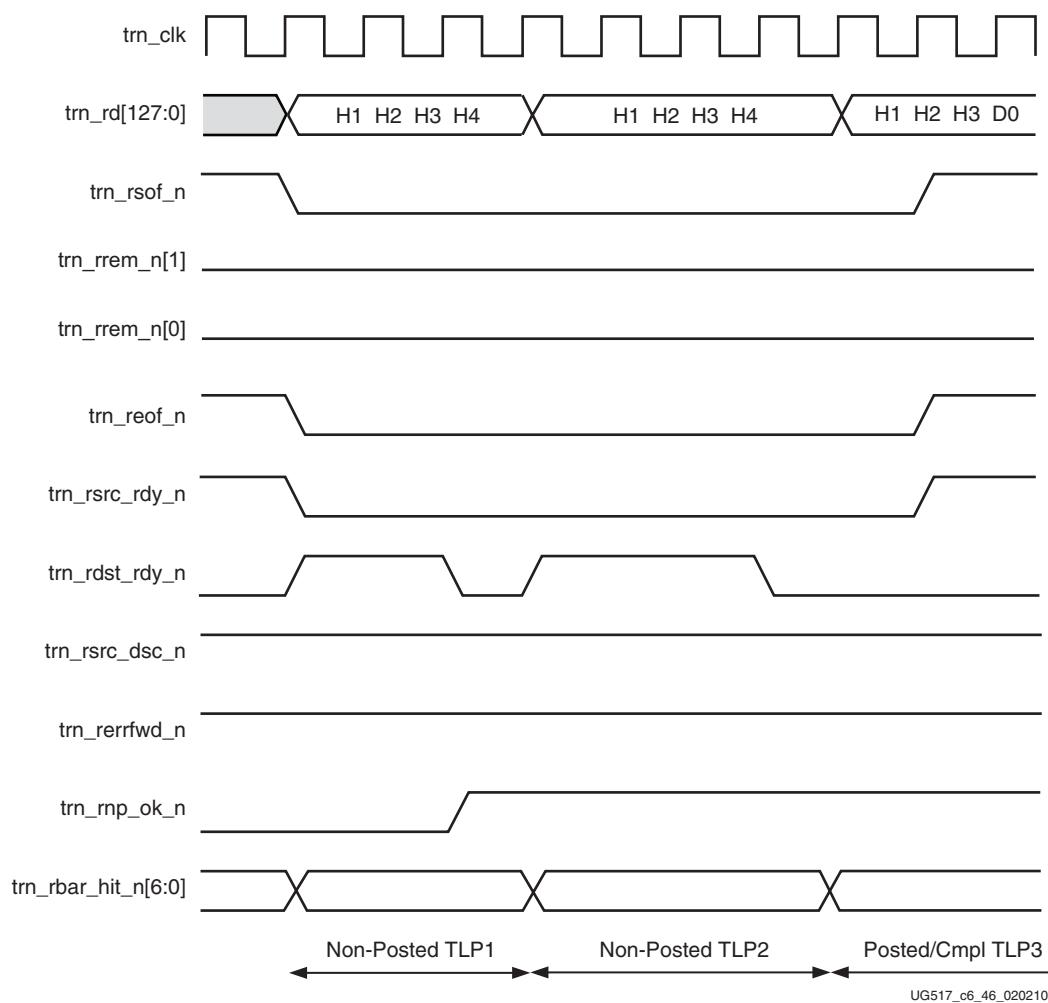


*Figure 6-46:*   **Packet Re-ordering on Receive Transaction Interface**

Packet re-ordering allows the User Application to optimize the rate at which Non-Posted TLPs are processed, while continuing to receive and process Posted and Completion TLPs in a non-blocking fashion. The trn_rnp_ok_n signaling restrictions require that the User Application be able to receive and buffer at least three Non-Posted TLPs. This algorithm describes the process of managing the Non-Posted TLP buffers:

Consider that Non-Posted_Buffers_Available denotes the size of Non-Posted buffer space available to User Application. The size of the Non-Posted buffer space is three Non-Posted TLPs. Non-Posted_Buffers_Available is decremented when Non Posted TLP is accepted for processing from the core, and is incremented when Non-Posted TLP is drained for processing by the User Application.

```
For every clock cycle do {
  if (Non-Posted_Buffers_Available <= 2) {
    if (Valid transaction Start-of-Frame accepted by user
              application) {
      Extract TLP Format and Type from the 1st TLP DW
      if (TLP type == Non-Posted) {
        Deassert trn_rnp_ok_n on the following clock
                            cycle
        - or -
        Other optional user policies to stall NP
                            transactions
      } else {
      }
    }
  } else { // Non-Posted_Buffers_Available > 2
    Assert trn_rnp_ok_n on the following clock cycle.
  }
}
```

### Packet Data Poisoning and TLP Digest on Receive Transaction Interface

To simplify logic within the User Application, the core performs automatic pre-processing based on values of TLP Digest (TD) and Data Poisoning (EP) header bit fields on the received TLP.

All received TLPs with the Data Poisoning bit in the header set (EP=1) are presented to the user. The core asserts the trn_rerrfwd_n signal for the duration of each poisoned TLP, as illustrated in Figure 6-47.



*Figure 6-47:*   **Receive Transaction Data Poisoning**

If the TLP Digest bit field in the TLP header is set (TD=1), the TLP contains an End-to-End CRC (ECRC). The core performs these operations based on how the user configured the core during core generation:

• If the Trim TLP Digest option is on, the core removes and discards the ECRC field from the received TLP and clears the TLP Digest bit in the TLP header.

• If the Trim TLP Digest option is off, the core does not remove the ECRC field from the received TLP and presents the entire TLP including TLP Digest to the User Application receiver interface.

See Chapter 5, Generating and Customizing the Core, for more information about how to enable the Trim TLP Digest option during core generation.

### Packet Base Address Register Hit on Receive Transaction Interface

The core decodes incoming Memory and I/O TLP request addresses to determine which Base Address Register (BAR) in the core's Type0 configuration space is being targeted, and indicates the decoded base address on trn_rbar_hit_n[6:0]. For each received Memory or I/O TLP, a minimum of one and a maximum of two (adjacent) bit(s) are set to 0. If the received TLP targets a 32-bit Memory or I/O BAR, only one bit is asserted. If the received TLP targets a 64-bit Memory BAR, two adjacent bits are asserted. If the core receives a TLP that is not decoded by one of the BARs (that is, a misdirected TLP), then the core drops it without presenting it to the user, and it automatically generates an Unsupported Request message. Even if the core is configured for a 64-bit BAR, the system might not always allocate a 64-bit address, in which case only one trn_rbar_hit_n[6:0] signal is asserted.

Table 6-7 illustrates mapping between trn_rbar_hit_n[6:0] and the BARs, and the corresponding byte offsets in the core Type0 configuration header.

*Table 6-7:* **trn_rbar_hit_n to Base Address Register Mapping**

| trn_rbar_hit_n[x] | BAR | Byte Offset |
|:---:|:---:|:---:|
| 0 | 0 | 10h |
| 1 | 1 | 14h |
| 2 | 2 | 18h |
| 3 | 3 | 1Ch |
| 4 | 4 | 20h |
| 5 | 5 | 24h |
| 6 | Expansion ROM BAR | 30h |

For a Memory or I/O TLP Transaction on the receive interface, trn_rbar_hit_n[6:0] is valid for the entire TLP, starting with the assertion of trn_rsof_n, as shown in Figure 6-48. For straddled data transfer beats, trn_rbar_hit_n corresponds to the new packet (packet corresponding to the trn_rsof_n). When receiving non-Memory and non-I/O transactions, trn_rbar_hit_n[6:0] is undefined.
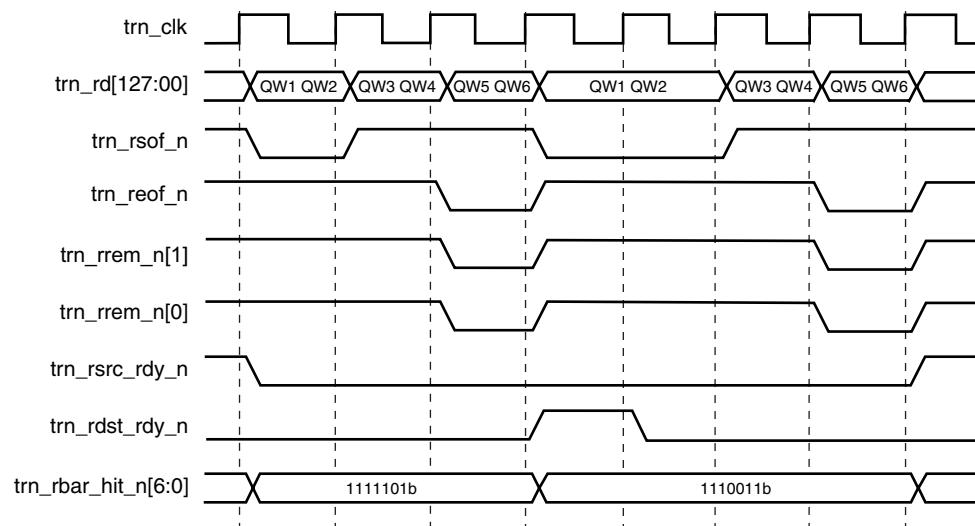


*Figure 6-48:*  **BAR Target Determination using trn_rbar_hit**

The signal trn_rbar_hit_n[6:0] enables received Memory and I/O transactions to be directed to the appropriate destination Memory and I/O apertures in the User Application. By utilizing trn_rbar_hit_n[6:0], application logic can inspect only the lower order Memory and I/O address bits within the address aperture to simplify decoding logic.

### Packet Transfer Discontinue on Receive Transaction Interface

The Endpoint for PCIe asserts trn_rsrc_dsc_n if communication with the link partner is lost, which results in the termination of an *in-progress* TLP. The loss of communication with the link partner is signaled by deassertion of trn_lnk_up_n. When trn_lnk_up_n is deasserted, it effectively acts as a *Hot Reset* to the entire core. For this reason, all TLPs stored inside the core or being presented to the receive interface are irrecoverably lost. Figure 6-49 illustrates packet transfer discontinue scenario.
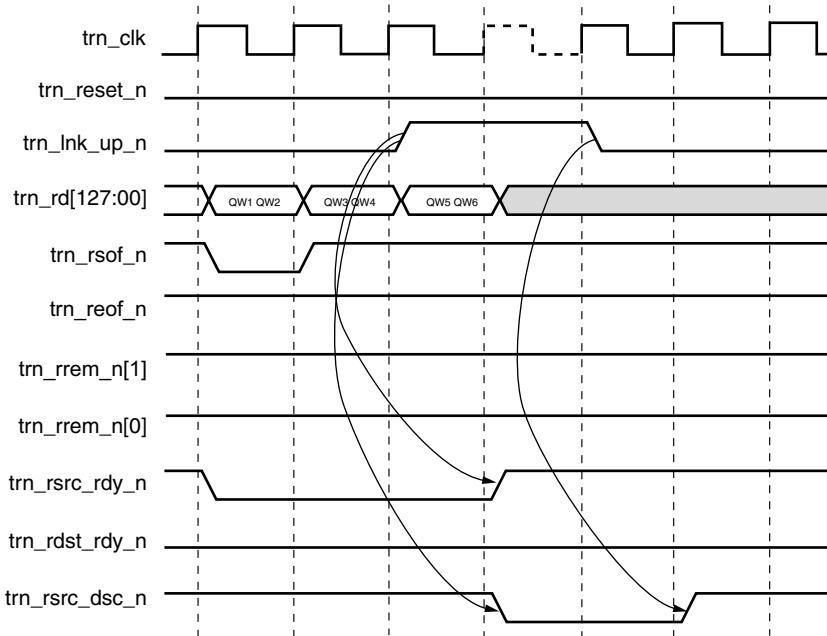
*Figure 6-49:* **Receive Transaction Discontinue**

## Transaction Processing on Receive Transaction Interface

Transaction processing in the Virtex-6 FPGA Integrated Block for PCI Express is fully compliant with the PCI Express Received TLP handling rules, as specified in the *PCI Express Base Specification, rev. 2.0.*

The Virtex-6 FPGA Integrated Block for PCI Express performs checks on received Transaction Layer Packets (TLPs) and passes valid TLPs to the User Application. It handles erroneous TLPs in the manner indicated in Table 6-8 and Table 6-9. Any errors associated with a TLP that are presented to the User Application that the core does not check for must be signaled by the User Application logic using the cfg_err_* interface.

Table 6-8 and Table 6-9 describe the packet disposition implemented in the Virtex-6 FPGA Integrated Block for PCI Express based on received TLP type and condition of core/TLP error for the Endpoint and Root Port configurations.

*Table 6-8:* **TLP Disposition on Receive Transaction Interface: Endpoint**

| TLP Type | Condition of Core or TLP Error | Core Response to TLP |
|---|---|---|
| Memory Read | BAR Miss | Unsupported Request |
| Memory Write | Received when in Non-D0 PM State | Unsupported Request |
| I/O Read | | |
| I/O Write | Neither of above conditions | TLP presented to User Application |

*Table 6-8:* **TLP Disposition on Receive Transaction Interface: Endpoint** *(Cont'd)*

| TLP Type | | Condition of Core or TLP Error | | Core Response to TLP |
|---|---|---|---|---|
| Memory Read Locked | | Received by a non-Legacy PCI Express Endpoint | | Unsupported Request |
| | | Legacy Endpoint | BAR Miss | Unsupported Request |
| | | | Received when in Non-D0 PM State | Unsupported Request |
| | | | Neither of above conditions | TLP presented to User Application |
| Configuration Read/Write Type 0 | | Internal Config Space | | TLP consumed by the core, to read/write internal Configuration Space and a CplD/Cpl is generated |
| | | User Defined Config Space | | TLP presented to User Application |
| Configuration Read/Write Type 1 | | Received by an Endpoint | | Unsupported Request |
| Completion Completion Locked | | Requester ID Miss | | Unexpected Completion |
| | | Received when in Non-D0 PM State | | Unexpected Completion |
| | | Neither of above conditions | | TLP presented to User Application |
| Messages | Set Slot Power Limit | Received by an Endpoint | | TLP consumed by the core and used to program the Captured Slot Power Limit Scale/Value fields of the Device Capabilities Register |
| | PM_PME PME_TO_Ack | Received by an Endpoint | | Unsupported Request |
| | PM_Active_State_NAK PME_Turn_Off | Received by an Endpoint | | TLP consumed by the core and used to control Power Management |
| | Unlock | Received by a non-Legacy Endpoint | | Ignored |
| | | Received by a Legacy Endpoint | | TLP presented to User Application[1] |
| | INTX | Received by an Endpoint | | Fatal Error |
| | Error_Fatal Error Non-Fatal Error Correctable | Received by an Endpoint | | Unsupported Request |
| | Vendor Defined Type 0 Vendor Defined Type 1 | Received by an Endpoint | | TLP presented to User Application[1] |
| | Hot Plug Messages | Received by an Endpoint | | TLP dropped by the core |

**Notes:**

1. The TLP is indicated on the cfg_msg* interface and also appears on the trn_r* interface *only if* enabled in the GUI.

*Table 6-9:* **TLP Disposition on Receive Transaction Interface: Root Port**

| TLP Type | | Condition of Core or TLP Error | Core Response to TLP |
|---|---|---|---|
| Memory Read<br>Memory Write<br>I/O Read<br>I/O Write | | BAR Miss | No BAR Filtering in Root Port configuration: TLP presented to User Application |
| | | Received when in Non-D0 PM State | Unsupported Request |
| | | Neither of above conditions | TLP presented to User Application |
| Memory Read Locked | | Received by a Root Port | TLP presented to User Application |
| Configuration Read / Write Type 0 | | Received by a Root Port | Unsupported Request |
| Configuration Read / Write Type 1 | | Received by a Root Port | Unsupported Request |
| Completion<br>Completion Locked | | Received by a Root Port | TLP presented to User Application |
| Messages | Set Slot Power Limit | Received by a Root Port | Unsupported Request |
| | PM_PME<br>PME_TO_Ack | Received by a Root Port | TLP presented to User Application[1] |
| | PM_Active_State_NAK | Received by a Root Port | Unsupported Request |
| | PME_Turn_Off | Received by a Root Port | Fatal Error |
| | Unlock | Received by a Root Port | Fatal Error |
| | INTX | Received by a Root Port | TLP presented to User Application[1] |
| | Error_Fatal<br>Error Non-Fatal<br>Error Correctable | Received by a Root Port | TLP presented to User Application[1] |
| | Vendor Defined Type 0<br>Vendor Defined Type 1 | Received by a Root Port | TLP presented to User Application[1] |
| | Hot Plug Messages | Received by a Root Port | TLP dropped by the core |

**Notes:**

1. The TLP is indicated on the cfg_msg* interface and also appears on the trn_r* interface *only if* enabled in the GUI.

## Core Buffering and Flow Control

### Maximum Payload Size

TLP size is restricted by the capabilities of both link partners. After the link is trained, the root complex sets the MAX_PAYLOAD_SIZE value in the Device Control register. This value is equal to or less than the value advertised by the core's Device Capability register. The advertised value in the Device Capability register of the Integrated Block core is either 128, 256, 512, or 1024 bytes, depending on the setting in the CORE Generator software GUI (1024 is not supported for the 8-lane, 5.0 Gb/s 128-bit core). For more information about these registers, see section 7.8 of the *PCI Express Base Specification*. The value of the core's Device Control register is provided to the User Application on the cfg_dcommand[15:0] output. See Design with Configuration Space Registers and Configuration Interface, page 155 for information about this output.

### Transmit Buffers

The Endpoint for PCIe transmit Transaction interface provides trn_tbuf_av, an instantaneous indication of the number of Max_Payload_Size buffers available for use in the transmit buffer pool. Table 6-10 defines the number of transmit buffers available and maximum supported payload size for a specific core.

*Table 6-10:* **Transmit Buffers Available**

| Capability Max Payload Size (Bytes) | Performance Level[1] | |
|---|---|---|
| | Good (Minimize Block RAM Usage) | High (Maximize Performance) |
| 128 | 26 | 32 |
| 256 | 14 | 29 |
| 512 | 15 | 30 |
| 1024[2] | 15 | 31 |

**Notes:**

1. Performance level is set through a CORE Generator software GUI selection.
2. 1024 is not supported for the 8-lane, 5.0 Gb/s, 128-bit core.

Each buffer can hold one maximum sized TLP. A maximum sized TLP is a TLP with a 4-DWORD header plus a data payload equal to the MAX_PAYLOAD_SIZE of the core (as defined in the Device Capability register) plus a TLP Digest. After the link is trained, the root complex sets the MAX_PAYLOAD_SIZE value in the Device Control register. This value is equal to or less than the value advertised by the core's Device Capability register. For more information about these registers, see section 7.8 of the *PCI Express Base Specification*. A TLP is held in the transmit buffer of the core until the link partner acknowledges receipt of the packet, at which time the buffer is released and a new TLP can be loaded into it by the User Application.

For example, if the Capability Max Payload Size selected for the Endpoint core is 256 bytes, and the performance level selected is high, there are 29 total transmit buffers. Each of these buffers can hold at a maximum one 64-bit Memory Write Request (4 DWORD header) plus 256 bytes of data (64 DWORDs) plus TLP Digest (1 DWORD) for a total of 69 DWORDs. This example assumes the root complex set the MAX_PAYLOAD_SIZE register of the Device Control register to 256 bytes, which is the maximum capability advertised by this core. For this reason, at any given time, this core could have 29 of these 69 DWORD TLPs awaiting transmittal. There is no sharing of buffers among multiple TLPs, so even if user is

sending smaller TLPs such as 32-bit Memory Read request with no TLP Digest totaling 3 DWORDs only per TLP, each transmit buffer still holds only one TLP at any time.

The internal transmit buffers are shared between the User Application and the core's configuration management module (CMM). Due to this, the trn_tbuf_av bus can fluctuate even if the User Application is not transmitting packets. The CMM generates completion TLPs in response to configuration reads or writes, interrupt TLPs at the request of the User Application, and message TLPs when needed.

The Transmit Buffers Available indication enables the User Application to completely utilize the PCI transaction ordering feature of the core transmitter. The transaction ordering rules allow for Posted and Completion TLPs to bypass Non-Posted TLPs. See section 2.4 of the *PCI Express Base Specification* for more information about ordering rules.

The core supports the transaction ordering rules and promotes Posted and Completion packets ahead of blocked Non-Posted TLPs. Non-Posted TLPs can become blocked if the link partner is in a state where it momentarily has no Non-Posted receive buffers available, which it advertises through Flow Control updates. In this case, the core promotes Completion and Posted TLPs ahead of these blocked Non-Posted TLPs. However, this can only occur if the Completion or Posted TLP has been loaded into the core by the User Application. By monitoring the trn_tbuf_av bus, the User Application can ensure there is at least one free buffer available for any Completion or Posted TLP. Promotion of Completion and Posted TLPs only occurs when Non-Posted TLPs are blocked; otherwise packets are sent on the link in the order they are received from the User Application.

## Receiver Flow Control Credits Available

The Endpoint for PCIe provides the User Application information about the state of the receiver buffer pool queues. This information represents the current space available for the Posted, Non-Posted, and Completion queues.

One Header Credit is equal to either a 3 or 4 DWORD TLP Header and one Data Credit is equal to 16 bytes of payload data. Table 6-11 provides values on credits available immediately after trn_lnk_up_n assertion but before the reception of any TLP. If space available for any of the above categories is exhausted, the corresponding credit available signals indicate a value of zero. Credits available return to initial values after the receiver has drained all TLPs.

*Table 6-11:* **Transaction Receiver Credits Available Initial Values**

| Credit Category | Performance Level | Capability Maximum Payload Size | | | |
|---|---|---|---|---|---|
| | | 128 byte | 256 byte | 512 byte | 1024 byte |
| Non-Posted Header | Good | 12 | | | |
| | High | | | | |
| Non-Posted Data | Good | 12 | | | |
| | High | | | | |
| Posted Header | Good | 32 | | | |
| | High | | | | |
| Posted Data | Good | 77 | 77 | 154 | 308 |
| | High | 154 | 154 | 308 | 616 |

*Table 6-11:* **Transaction Receiver Credits Available Initial Values** *(Cont'd)*

| Credit Category | Performance Level | Capability Maximum Payload Size | | | |
|---|---|---|---|---|---|
| | | **128 byte** | **256 byte** | **512 byte** | **1024 byte** |
| Completion Header | Good | 36 | | | |
| | High | | | | |
| Completion Data | Good | 77 | 77 | 154 | 308 |
| | High | 154 | 154 | 308 | 616 |

The User Application can use the trn_fc_ph[7:0], trn_fc_pd[11:0], trn_fc_nph[7:0], trn_fc_npd[11:0], trn_fc_cplh[7:0], trn_fc_cpld[11:0], and trn_fc_sel[2:0] signals to efficiently utilize and manage receiver buffer space available in the core and the core application. For additional information, see Flow Control Credit Information.

Endpoint cores for PCI Express have a unique requirement where the User Application must use advanced methods to prevent buffer overflows while requesting Non-Posted Read Requests from an upstream component. According to the specification, a PCI Express Endpoint is required to advertise infinite storage credits for Completion Transactions in its receivers. This means that endpoints must internally manage Memory Read Requests transmitted upstream and not overflow the receiver when the corresponding Completions are received. The User Application transmit logic must use Completion credit information presented to modulate the rate and size of Memory Read requests, to stay within the instantaneous Completion space available in the core receiver. For additional information, see Appendix E, Managing Receive-Buffer Space for Inbound Completions.

## Flow Control Credit Information

### Using the Flow Control Credit Signals

The integrated block provides the user application with information about the state of the Transaction Layer transmit and receive buffer credit pools. This information represents the current space available, as well as the credit "limit" and "consumed" information for the Posted, Non-Posted, and Completion pools.

Table 2-10, page 32 defines the Flow Control Credit signals. Credit status information is presented on these signals:

- trn_fc_ph[7:0]
- trn_fc_pd[11:0]
- trn_fc_nph[7:0]
- trn_fc_npd[11:0]
- trn_fc_cplh[7:0]
- trn_fc_cpld[11:0]

Collectively, these signals are referred to as trn_fc_*.

The trn_fc_* signals provide information about each of the six credit pools defined in the *PCI Express Base Specification*: Header and Data Credits for Each of Posted, Non-Posted, and Completion.

Six different types of flow control information can be read by the user application. The trn_fc_sel[2:0] input selects the type of flow control information represented by the trn_fc_* outputs. The Flow Control Information Types are shown in Table 6-12.

*Table 6-12:* **Flow Control Information Types**

| trn_fc_sel[2:0] | Flow Control Information Type |
|---|---|
| 000 | Receive Credits Available Space |
| 001 | Receive Credits Limit |
| 010 | Receive Credits Consumed |
| 011 | Reserved |
| 100 | Transmit Credits Available Space |
| 101 | Transmit Credit Limit |
| 110 | Transmit Credits Consumed |
| 111 | Reserved |

trn_fc_sel[2:0] can be changed on every clock cycle to indicate a different Flow Control Information Type. There is a two clock-cycle delay between the value of trn_fc_sel[2:0] changing and the corresponding Flow Control Information Type being presented on the trn_fc_* outputs for the 64-bit interface and a four clock cycle delay for the 128-bit interface. Figure 6-50 and Figure 6-51 illustrate the timing of the Flow Control Credits signals for the 64-bit and 128-bit interfaces, respectively.



UG517_c6_50_020910
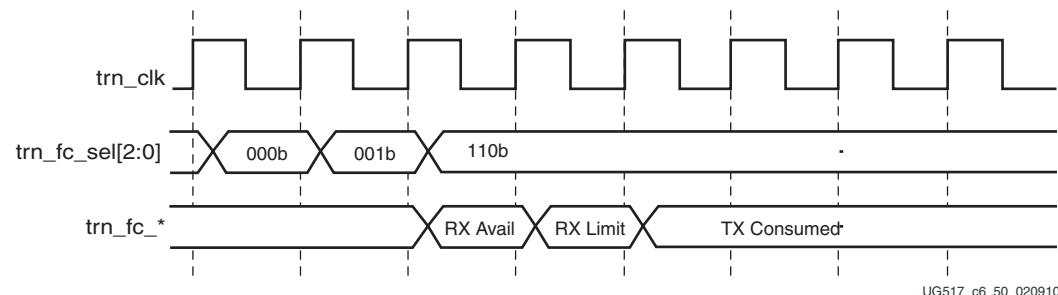
*Figure 6-50:* **Flow Control Credits for 64-bit Interface**



UG517_c6_51_020910
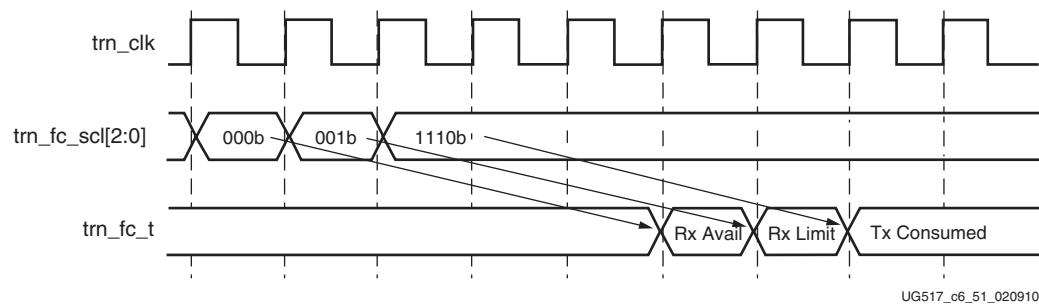
*Figure 6-51:* **Flow Control Credits for 128-bit Interface**

The output values of the trn_fc_* signals represent credit values as defined in the *PCI Express Base Specification*. One Header Credit is equal to either a 3 or 4 DWORD TLP Header and one Data Credit is equal to 16 bytes of payload data. Initial credit information

is available immediately after trn_lnk_up_n assertion, but before the reception of any TLP. Table 6-13 defines the possible values presented on the trn_fc_* signals. Initial credit information varies depending on the size of the receive buffers within the integrated block and the Link Partner.

*Table 6-13:*   **trn_fc_* Value Definition**

| Header Credit Value | Data Credit Value | Meaning |
|---|---|---|
| 00 – 7F | 000 – 7FF | User credits |
| FF-80 | FFF-800 | Negative credits available[1] |
| 7F | 7FF | Infinite credits available[1] |

**Notes:**

1. Only Transmit Credits Available Space indicate Negative or Infinite credits available.

## Receive Credit Flow Control Information

Receive Credit Flow Control Information can be obtained by setting trn_fc_sel[2:0] to 000b, 001b, or 010b. The Receive Credit Flow Control information indicates the current status of the receive buffers within the integrated block.

Receive Credits Available Space: trn_fc_sel[2:0] = 000b

Receive Credits Available Space shows the credit space available in the integrated block's Transaction Layer local receive buffers for each credit pool. If space available for any of the credit pools is exhausted, the corresponding trn_fc_* signal indicates a value of zero. Receive Credits Available Space returns to its initial values after the user application has drained all TLPs from the integrated block.

In the case where infinite credits have been advertised to the Link Partner for a specific Credit pool, such as Completion Credits for Endpoints, the user application should use this value along with the methods described in Appendix E, Managing Receive-Buffer Space for Inbound Completions, to avoid completion buffer overflow.

Receive Credits Limit: trn_fc_sel[2:0] = 001b

Receive Credits Limit shows the credits granted to the link partner. The trn_fc_* values are initialized with the values advertised by the integrated block during Flow Control initialization and are updated as a cumulative count as TLPs are read out of the Transaction Layer's receive buffers via the TRN interface. This value is referred to as CREDITS_ALLOCATED within the *PCI Express Base Specification*.

In the case where infinite credits have been advertised for a specific credit pool, the Receive Buffer Credits Limit for that pool always indicates zero credits.

Receive Credits Consumed: trn_fc_sel[2:0] = 010b

Receive Buffer Credits Consumed shows the credits consumed by the link partner (and received by the integrated block). The initial trn_fc_* values are always zero and are updated as a cumulative count, as packets are received by the Transaction Layers receive buffers. This value is referred to as CREDITS_RECEIVED in the *PCI Express Base Specification*.

## Transmit Credit Flow Control Information

Transmit Credit Flow Control Information can be obtained by setting trn_fc_sel[2:0] to 100b, 101b, or 110b. The Transmit Credit Flow Control information indicates the current status of the receive buffers within the Link Partner.

Transmit Credits Available Space: trn_fc_sel[2:0] = 100b

Transmit Credits Available Space indicates the available credit space within the receive buffers of the Link Partner for each credit pool. If space available for any of the credit pools is exhausted, the corresponding trn_fc_* signal indicates a value of zero or negative. Transmit Credits Available Space returns to its initial values after the integrated block has successfully sent all TLPs to the Link Partner.

If the value is negative, more header or data has been written into the integrated block's local transmit buffers than the Link Partner can currently consume. Because the block does not allow posted packets to pass completions, a posted packet that is written is not transmitted if there is a completion ahead of it waiting for credits (as indicated by a zero or negative value). Similarly, a completion that is written is not transmitted if a posted packet is ahead of it waiting for credits. The user application can monitor the Transmit Credits Available Space to ensure that these temporary blocking conditions do not occur, and that the bandwidth of the PCI Express Link is fully utilized by only writing packets to the integrated block that have sufficient space within the Link Partner's Receive buffer. Non-Posted packets can always be bypassed within the integrated block; so, any Posted or Completion packet written passes Non-Posted packets waiting for credits.

The Link Partner can advertise infinite credits for one or more of the three traffic types. Infinite credits are indicated to the user by setting the Header and Data credit outputs to their maximum value as indicated in Table 6-13.

Transmit Credits Limit: trn_fc_sel[2:0] = 101b

Transmit Credits Limit shows the receive buffer limits of the Link Partner for each credit pool. The trn_fc_* values are initialized with the values advertised by the Link Partner during Flow Control initialization and are updated as a cumulative count as Flow Control updates are received from the Link Partner. This value is referred to as CREDITS_LIMIT in the *PCI Express Base Specification*.

In the case where infinite credits have been advertised for a specific Credit pool, the Transmit Buffer Credits Limit always indicates zero credits for that pool.

Transmit Credits Consumed: trn_fc_sel[2:0] = 110b

Transmit Credits Consumed show the credits consumed of the Receive Buffer of the Link Partner by the integrated block. The initial value is always zero and is updated as a cumulative count, as packets are transmitted to the Link Partner. This value is referred to as CREDITS_CONSUMED in the *PCI Express Base Specification*.

# Designing with the Physical Layer Control and Status Interface

Physical Layer Control and Status enables the User Application to change link width and speed in response to data throughput and power requirements.

## Design Considerations for a Directed Link Change

These points should be considered during a Directed Link Change:

- Link change operation must be initiated only while trn_lnk_up_n is asserted and the core is in the L0 state, as indicated by the signal pl_ltssm_state[5:0].

- Link Width Change should not be used when Lane Reversal is enabled.

- Target Link Width of a Link Width Change operation must be equal to or less than the width indicated by pl_initial_link_width output.

- When pl_link_upcfg_capable is set to 1b, the PCI Express link is Upconfigure capable. This allows the link width to be varied between the Initial Negotiated Link Width and any smaller link width supported by both the Port and link partner (this is for link reliability or application reasons).

- If a link is not Upconfigure capable, the Negotiated link width can only be varied to a width less than the Negotiated Link Width that is supported by both the link partner and device.

- Before initiating a link speed change from 2.5 Gb/s to 5.0 Gb/s, the User Application must ensure that the link is 5.0 Gb/s (Gen2) capable (that is, pl_linkgen2_capable is 1b) and the Link Partner is also Gen2 capable (pl_linkpartner_gen2_capable is 1b).

- A link width change that benefits the application must be initiated only when cfg_lcommand[9] (the Hardware Autonomous Width Disable Bit) is 0b. In addition, for both link speed and/or width change driven by application need, pl_directed_link_auton must be driven (1b). The user might also want to have the User Application check that pl_link_upcfg_capable is 1b, which guarantees that the link width and speed can be restored to the original (higher) width and speed.

- If the User Application directs the link to a width not supported by the link partner, the resulting link width is the next narrower mutually supported link width. For example, an 8-lane link is directed to a 4-lane operation, but the link partner supports only 1-lane train down operations. So, this would result in a 1-lane operation.

- The Endpoint should initiate directed link change only when the device is in D0 power state (cfg_pmcsr_powerstate[1:0] = 00b).

- A retrain should not be initiated (using directed link change pins (Root or Endpoint) or by setting the retrain bit (Root only), if the cfg_pcie_link_state = 101b (transitioning to/from PPM L1) or 110b (transitioning to PPM L2/L3 Ready).

## Directed Link Width Change

Figure 6-52 shows the directed link width change process that must be implemented by the User Application. Here target_link_width[1:0] is the application-driven new link width request.
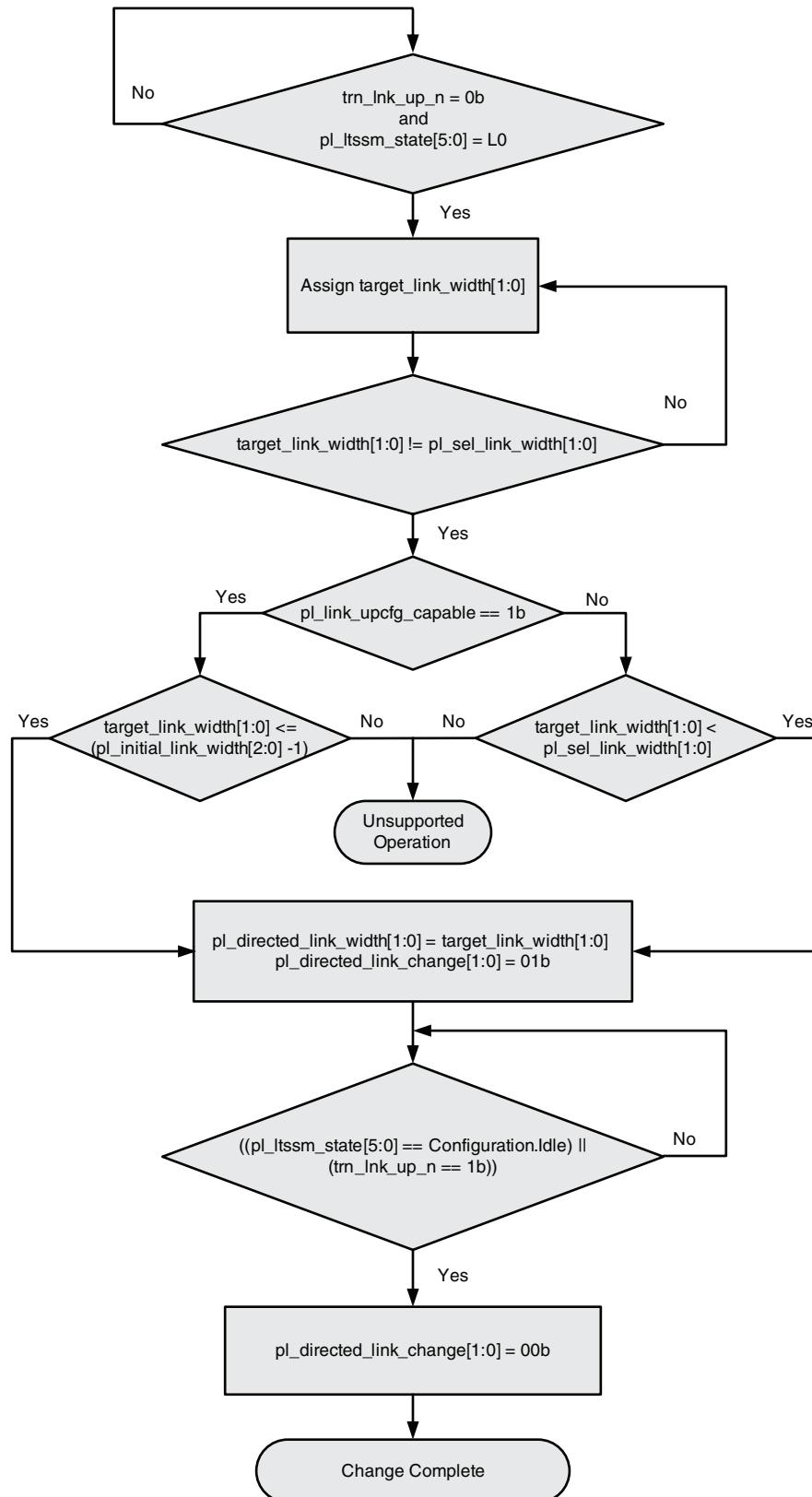
*Figure 6-52:* **Directed Link Width Change**

## Directed Link Speed Change

Figure 6-53 shows the directed link speed change process that must be implemented by the User Application. Here target_link_speed is the application-driven new link speed request.
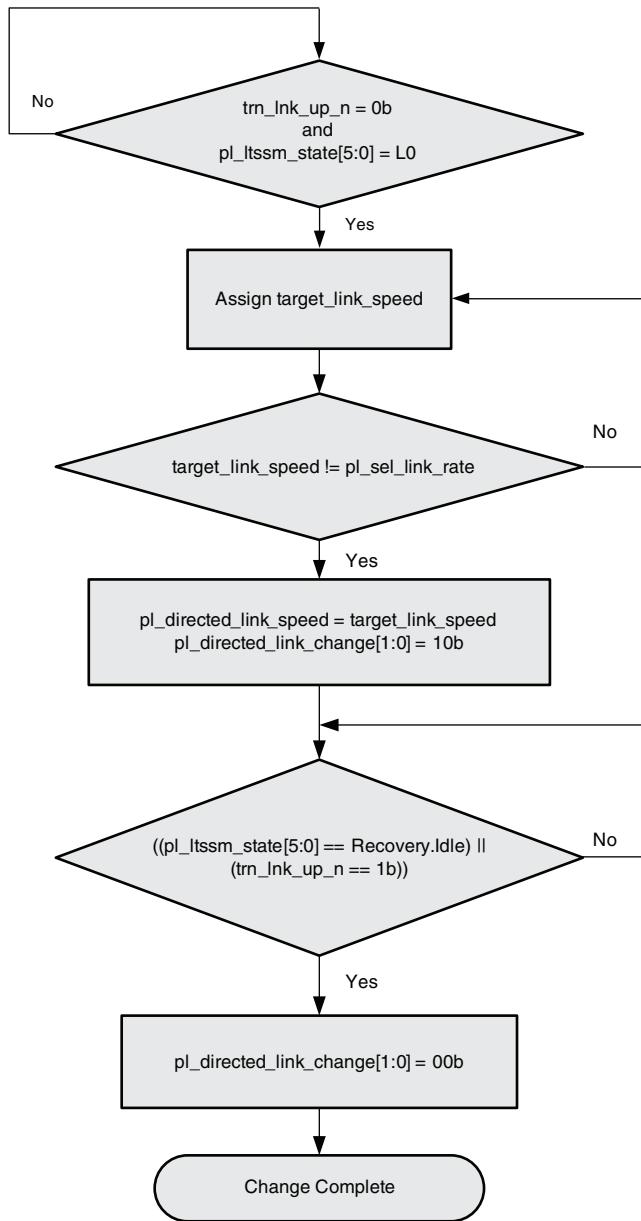


*Figure 6-53:* **Directed Link Speed Change**

## Directed Link Width and Speed Change

Figure 6-54 shows the directed link width and speed change process that must be implemented by the User Application. Here target_link_width[1:0] is the application driven new link width request, while target_link_speed is the application driven new link speed request.
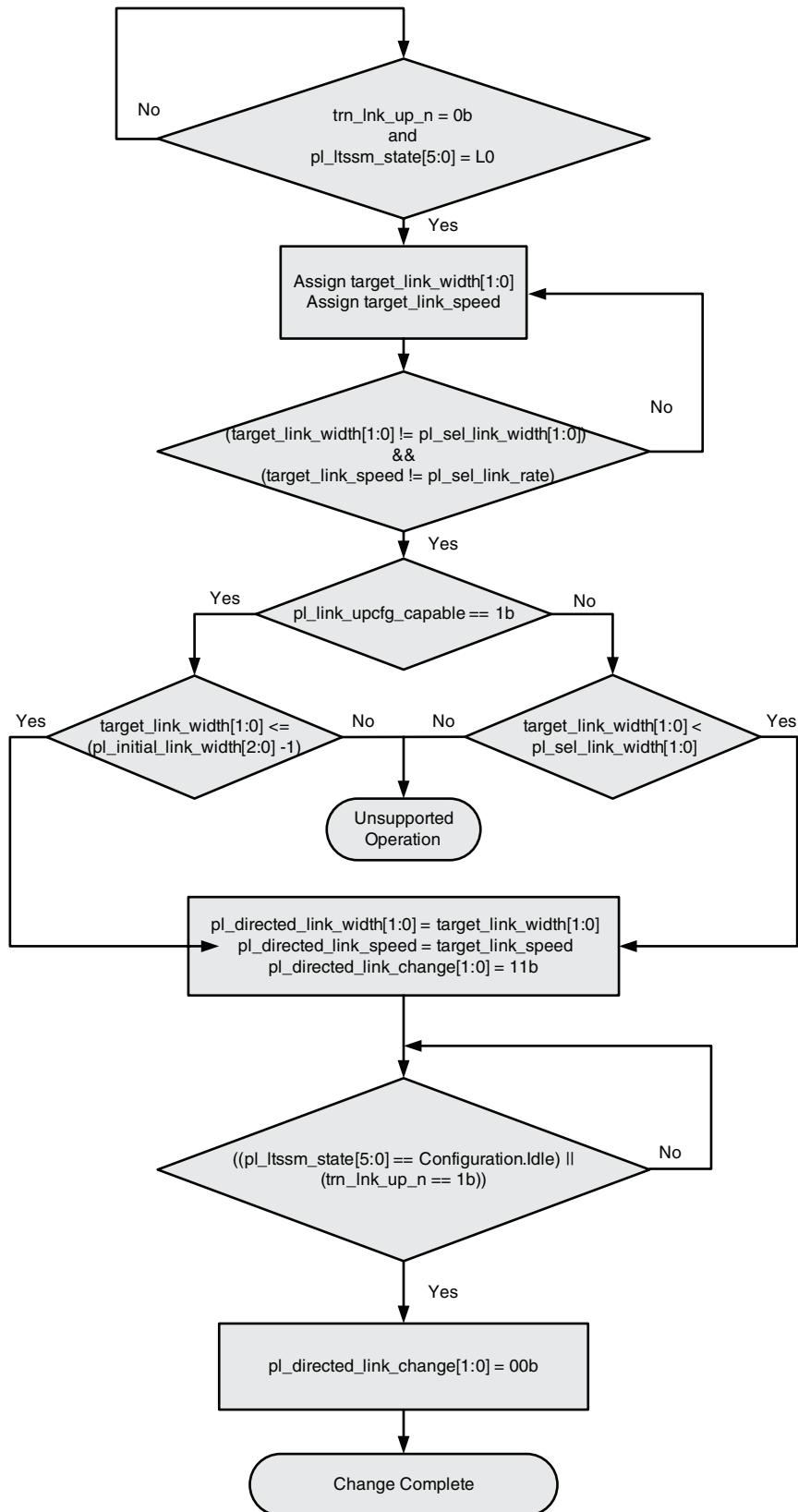
*Figure 6-54:* **Directed Link Width and Speed Change**

# Design with Configuration Space Registers and Configuration Interface

This section describes the use of the Configuration Interface for accessing the PCI Express Configuration Space Type 0 or Type 1 registers that are part of the Integrated Block core. The Configuration Interface includes a read/write Configuration Port for accessing the registers. In addition, some commonly used registers are mapped directly on the Configuration Interface for convenience.

## Registers Mapped Directly onto the Configuration Interface

The Integrated Block core provides direct access to select command and status registers in its Configuration Space. For endpoints, the values in these registers are typically modified by Configuration Writes received from the Root Complex; however, the User Application can also modify these values using the Configuration Port. In the Root Port configuration, the Configuration Port must always be used to modify these values. Table 6-14 defines the command and status registers mapped to the configuration port.

*Table 6-14:*  **Command and Status Registers Mapped to the Configuration Port**

| Port Name | Direction | Description |
|---|---|---|
| cfg_bus_number[7:0] | Output | Bus Number: Default value after reset is 00h. Refreshed whenever a Type 0 Configuration Write packet is received. |
| cfg_device_number[4:0] | Output | Device Number: Default value after reset is 00000b. Refreshed whenever a Type 0 Configuration Write packet is received. |
| cfg_function_number[2:0] | Output | Function Number: Function number of the core, hardwired to 000b. |
| cfg_status[15:0] | Output | Status Register: Status register from the Configuration Space Header. Not supported. |
| cfg_command[15:0] | Output | Command Register: Command register from the Configuration Space Header. |
| cfg_dstatus[15:0] | Output | Device Status Register: Device status register from the PCI Express Capability Structure. |
| cfg_dcommand[15:0] | Output | Device Command Register: Device control register from the PCI Express Capability Structure. |
| cfg_dcommand2[15:0] | Output | Device Command 2 Register: Device control 2 register from the PCI Express Capability Structure. |
| cfg_lstatus[15:0] | Output | Link Status Register: Link status register from the PCI Express Capability Structure. |
| cfg_lcommand[15:0] | Output | Link Command Register: Link control register from the PCI Express Capability Structure. |

# Device Control and Status Register Definitions

## cfg_bus_number[7:0], cfg_device_number[4:0], cfg_function_number[2:0]

Together, these three values comprise the core ID, which the core captures from the corresponding fields of inbound Type 0 Configuration Write accesses. The User Application is responsible for using this core ID as the Requestor ID on any requests it originates, and using it as the Completer ID on any Completion response it sends. This core supports only one function; for this reason, the function number is hardwired to `000b`.

## cfg_status[15:0]

This output bus is not supported. If the user wishes to retrieve this information, this can be derived by Read access of the Configuration Space in the Virtex-6 FPGA Integrated Block for PCI Express via the Configuration Port.

## cfg_command[15:0]

This bus reflects the value stored in the Command register in the PCI Configuration Space Header. Table 6-15 provides the definitions for each bit in this bus. See the *PCI Express Base Specification* for detailed information.

*Table 6-15:* **Bit Mapping on Header Command Register**

| Bit | Name |
|---|---|
| cfg_command[15:11] | Reserved |
| cfg_command[10] | Interrupt Disable |
| cfg_command[9] | Fast Back-to-Back Transactions Enable (hardwired to 0) |
| cfg_command[8] | SERR Enable |
| cfg_command[7] | IDSEL Stepping/Wait Cycle Control (hardwired to 0) |
| cfg_command[6] | Parity Error Enable - Not Supported |
| cfg_command[5] | VGA Palette Snoop (hardwired to 0) |
| cfg_command[4] | Memory Write and Invalidate (hardwired to 0) |
| cfg_command[3] | Special Cycle Enable (hardwired to 0) |
| cfg_command[2] | Bus Master Enable |
| cfg_command[1] | Memory Address Space Decoder Enable |
| cfg_command[0] | I/O Address Space Decoder Enable |

The User Application must monitor the Bus Master Enable bit (cfg_command[2]) and refrain from transmitting requests while this bit is not set. This requirement applies only to requests; completions can be transmitted regardless of this bit.

### cfg_dstatus[15:0]

This bus reflects the value stored in the Device Status register of the PCI Express Capabilities Structure. Table 6-16 defines each bit in the cfg_dstatus bus. See the *PCI Express Base Specification* for detailed information.

*Table 6-16:* **Bit Mapping on PCI Express Device Status Register**

| Bit | Name |
|---|---|
| cfg_dstatus[15:6] | Reserved |
| cfg_dstatus[5] | Transaction Pending |
| cfg_dstatus[4] | AUX Power Detected (hardwired to 0) |
| cfg_dstatus[3] | Unsupported Request Detected |
| cfg_dstatus[2] | Fatal Error Detected |
| cfg_dstatus[1] | Non-Fatal Error Detected |
| cfg_dstatus[0] | Correctable Error Detected |

### cfg_dcommand[15:0]

This bus reflects the value stored in the Device Control register of the PCI Express Capabilities Structure. Table 6-17 defines each bit in the cfg_dcommand bus. See the *PCI Express Base Specification* for detailed information.

*Table 6-17:* **Bit Mapping of PCI Express Device Control Register**

| Bit | Name |
|---|---|
| cfg_dcommand[15] | Reserved |
| cfg_dcommand[14:12] | Max_Read_Request_Size |
| cfg_dcommand[11] | Enable No Snoop |
| cfg_dcommand[10] | Auxiliary Power PM Enable |
| cfg_dcommand[9] | Phantom Functions Enable |
| cfg_dcommand[8] | Extended Tag Field Enable |
| cfg_dcommand[7:5][1] | Max_Payload_Size |
| cfg_dcommand[4] | Enable Relaxed Ordering |
| cfg_dcommand[3] | Unsupported Request Reporting Enable |
| cfg_dcommand[2] | Fatal Error Reporting Enable |
| cfg_dcommand[1] | Non-Fatal Error Reporting Enable |
| cfg_dcommand[0] | Correctable Error Reporting Enable |

**Notes:**

1. During L1 negotiation, the user should not trigger a link retrain by writing a 1 to cfg_lcommand[5]. L1 negotiation can be observed by monitoring the cfg_pcie_link_state_n port.

### cfg_lstatus[15:0]

This bus reflects the value stored in the Link Status register in the PCI Express Capabilities Structure. Table 6-18 defines each bit in the cfg_lstatus bus. See the *PCI Express Base Specification* for details.

*Table 6-18:* **Bit Mapping of PCI Express Link Status Register**

| Bit | Name |
| --- | --- |
| cfg_lstatus[15] | Link Autonomous Bandwidth Status |
| cfg_lstatus[14] | Link Bandwidth Management Status |
| cfg_lstatus[13] | Data Link Layer Link Active |
| cfg_lstatus[12] | Slot Clock Configuration |
| cfg_lstatus[11] | Link Training |
| cfg_lstatus[10] | Reserved |
| cfg_lstatus[9:4] | Negotiated Link Width |
| cfg_lstatus[3:0] | Current Link Speed |

### cfg_lcommand[15:0]

This bus reflects the value stored in the Link Control register of the PCI Express Capabilities Structure. Table 6-19 provides the definition of each bit in cfg_lcommand bus. See the *PCI Express Base Specification, rev. 2.0* for more details.

*Table 6-19:* **Bit Mapping of PCI Express Link Control Register**

| Bit | Name |
| --- | --- |
| cfg_lcommand[15:12] | Reserved |
| cfg_lcommand[11] | Link Autonomous Bandwidth Interrupt Enable |
| cfg_lcommand[10] | Link Bandwidth Management Interrupt Enable |
| cfg_lcommand[9] | Hardware Autonomous Width Disable |
| cfg_lcommand[8] | Enable Clock Power Management |
| cfg_lcommand[7] | Extended Synch |
| cfg_lcommand[6] | Common Clock Configuration |
| cfg_lcommand[5] | Retrain Link (Reserved for an Endpoint device) |
| cfg_lcommand[4] | Link Disable |
| cfg_lcommand[3] | Read Completion Boundary |
| cfg_lcommand[2] | Reserved |
| cfg_lcommand[1:0] | Active State Link PM Control |

### cfg_dcommand2[15:0]

This bus reflects the value stored in the Device Control 2 register of the PCI Express Capabilities Structure. Table 6-20 defines each bit in the cfg_dcommand bus. See the *PCI Express Base Specification* for detailed information.

*Table 6-20:* **Bit Mapping of PCI Express Device Control 2 Register**

| Bit | Name |
|---|---|
| cfg_dcommand2[15:5] | Reserved |
| cfg_dcommand2[4] | Completion Timeout Disable |
| cfg_dcommand2[3:0] | Completion Timeout Value |

## Core Response to Command Register Settings

Table 6-21 and Table 6-22 illustrate the behavior of the Virtex-6 FPGA Integrated Block for PCI Express based on the Command Register settings when configured as either an Endpoint or a Root Port.

*Table 6-21:* **Command Register (0x004): Endpoint**

| Bit(s) | Name | Attr | Endpoint Core Behavior |
|---|---|---|---|
| 0 | I/O Space Enable | RW | The Endpoint does not permit a BAR hit on I/O space unless this is enabled. |
| 1 | Memory Space Enable | RW | The Endpoint does not permit a BAR hit on Memory space unless this is enabled. |
| 2 | Bus Master Enable | RW | The Endpoint does not enforce this; user could send a TLP via TRN interface. |
| 5:3 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 6 | Parity Error Response | RW | Enables Master Data Parity Error (Status[8]) to be set. |
| 7 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 8 | SERR# Enable | RW | Can enable Error NonFatal / Error Fatal Message generation, and enables Status[14] ("Signaled System Error"). |
| 9 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 10 | Interrupt Disable | RW | If set to "1", the cfg_interrupt* interface is unable to cause INTx messages to be sent. |
| 15:11 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |

*Table 6-22:* **Command Register (0x004): Root Port**

| Bit(s) | Name | Attr | Root Port Core behavior |
|---|---|---|---|
| 0 | I/O Space Enable | RW | The Root Port ignores this setting. If disabled, it still accept s I/O TLP from the user side and passes downstream. User application logic must enforce not sending I/O TLPs downstream if this is unset. |
| 1 | Memory Space Enable | RW | The Root Port ignores this setting. If disabled, it still accepts Mem TLPs from the user side and passes downstream. User application logic must enforce not sending Mem TLPs downstream if this is unset. |
| 2 | Bus Master Enable | RW | When set to 0, the Root Port responds to target transactions such as an Upstream Mem or I/O TLPs as a UR (that is, UR bit set if enabled or Cpl w/ UR packet sent if TLP was a Non-Posted). When set to 1, all target transactions are passed to the user. |
| 5:3 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 6 | Parity Error Response | RW | Enables Master Data Parity Error (Status[8]) to be set. |
| 7 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 8 | SERR# Enable | RW | If enabled, Error Fatal/Error Non-Fatal Messages can be forwarded from TRN interface or cfg_err*, or internally generated. The Root Port does not enforce the requirement that Error Fatal/Error Non-Fatal Messages received on the link not be forwarded if this bit unset; user logic must do that. **Note**: Error conditions detected internal to the Root Port are indicated on cfg_msg* interface. |
| 9 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 10 | Interrupt Disable | RW | Not applicable to Root Port. |
| 15:11 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |

## Status Register Response to Error Conditions

Table 6-23 throughTable 6-25 illustrate the conditions that cause the Status Register bits to be set in the Virtex-6 FPGA Integrated Block for PCI Express when configured as either an Endpoint or a Root Port.

*Table 6-23:* **Status Register (0x006): Endpoint**

| Bit(s) | Name | Attr | Cause in an Endpoint |
|---|---|---|---|
| 2:0 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 3 | Interrupt Status | RO | • Set when interrupt signaled by user. • Clears when interrupt is cleared by the Interrupt handler. |
| 4 | Capabilities List | RO | Wired to 1. |
| 7:5 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |

*Table 6-23:* **Status Register (0x006): Endpoint** *(Cont'd)*

| Bit(s) | Name | Attr | Cause in an Endpoint |
|---|---|---|---|
| 8 | Master Data Parity Error | RW1C | Set if Parity Error Response is set and a Poisoned Cpl TLP is received on the link, or a Poisoned Write TLP is sent. |
| 10:9 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 11 | Signaled Target Abort | RW1C | Set if a Completion with status Completer Abort is sent upstream by the user via the cfg_err* interface. |
| 12 | Received Target Abort | RW1C | Set if a Completion with status Completer Abort is received. |
| 13 | Received Master Abort | RW1C | Set if a Completion with status Unsupported Request is received. |
| 14 | Signaled System Error | RW1C | Set if an Error Non-Fatal / Error Fatal Message is sent, and SERR# Enable (Command[8]) is set. |
| 15 | Detected Parity Error | RW1C | Set if a Poisoned TLP is received on the link. |

*Table 6-24:* **Status Register (0x006): Root Port**

| Bit(s) | Name | Attr | Cause in a Root Port |
|---|---|---|---|
| 2:0 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 3 | Interrupt Status | RO | Has no function in the Root Port. |
| 4 | Capabilities List | RO | Wired to 1. |
| 7:5 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 8 | Master Data Parity Error | RW1C | Set if Parity Error Response is set and a Poisoned Completion TLP is received on the link. |
| 10:9 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 11 | Signaled Target Abort | RW1C | Never set by the Root Port |
| 12 | Received Target Abort | RW1C | Never set by the Root Port |
| 13 | Received Master Abort | RW1C | Never set by the Root Port |
| 14 | Signaled System Error | RW1C | Set if the Root Port:<br>• Receives an Error Non-Fatal / Error Fatal Message and both SERR# Enable and Secondary SERR# enable are set.<br>• Indicates on the cfg_msg* interface that a Error Fatal / Error Non-Fatal Message should be generated upstream and SERR# enable is set. |
| 15 | Detected Parity Error | RW1C | Set if a Poisoned TLP is transmitted downstream. |

*Table 6-25:* **Secondary Status Register (0x01E): Root Port**

| Bit(s) | Name | Attr | Cause in a Root Port |
|---|---|---|---|
| 7:0 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 8 | Secondary Master Data Parity Error | RW1C | Set when the Root Port:<br>Receives a Poisoned Completion TLP, and Secondary Parity Error Response==1<br>Transmits a Poisoned Write TLP, and Secondary Parity Error Response==1 |
| 10:9 | Reserved | RO | Wired to 0. Not applicable to PCI Express. |
| 11 | Secondary Signaled Target Abort | RW1C | Set when User indicates a Completer-Abort via *cfg_err_cpl_abort_n* |
| 12 | Secondary Received Target Abort | RW1C | Set when the Root Port receives a Completion TLP with status Completer-Abort. |
| 13 | Secondary Received Master Abort | RW1C | Set when the Root Port receives a Completion TLP with status Unsupported Request |
| 14 | Secondary Received System Error | RW1C | Set when the Root Port receives an Error Fatal / Error Non-Fatal Message. |
| 15 | Secondary Detected Parity Error | RW1C | Set when the Root Port receives a Poisoned TLP. |

## Accessing Registers through the Configuration Port

Configuration registers that are not directly mapped to the user interface can be accessed by configuration-space address using the ports shown in Table 2-17, page 42. Root Ports must use the Configuration Port to setup the Configuration Space. Endpoints can also use the Configuration Port to read and write; however, care must be taken to avoid adverse system side effects.

The User Application must supply the address as a DWORD address, not a byte address. To calculate the DWORD address for a register, divide the byte address by four. For example:

- The DWORD address of the Command/Status Register in the PCI Configuration Space Header is `01h`. (The byte address is `04h`.)
- The DWORD address for BAR0 is `04h`. (The byte address is `10h`.)

To read any register in configuration space, shown in Table 2-2, page 25, the User Application drives the register DWORD address onto cfg_dwaddr[9:0]. The core drives the content of the addressed register onto cfg_do[31:0]. The value on cfg_do[31:0] is qualified by signal assertion on cfg_rd_wr_done_n. Figure 6-55 illustrates an example with two consecutive reads from the Configuration Space.
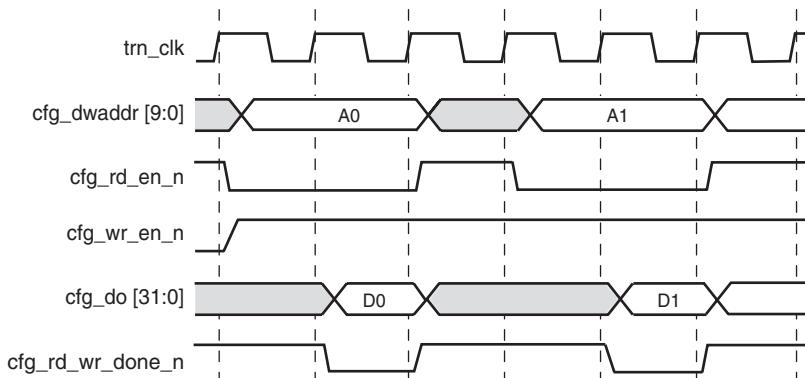
*Figure 6-55:* **Example Configuration Space Read Access**

Configuration Space registers which are defined as "RW" by the PCI Local Bus Specification and PCI Express Base Specification are writable via the Configuration Management Interface. To write a register in this address space, the User Application drives the register DWORD address onto cfg_dwaddr[9:0] and the data onto cfg_di[31:0]. This data is further qualified by cfg_byte_en_n[3:0], which validates the bytes of data presented on cfg_di[31:0]. These signals should be held asserted until cfg_rd_wr_done_n is asserted. Figure 6-56 illustrates an example with two consecutive writes to the Configuration Space, the first write with the User Application writing to all 32 bits of data, and the second write with the User Application selectively writing to only bits [23:26].

**Note:** Writing to the Configuration Space could have adverse system side effects. Users should ensure these writes do not negatively impact the overall system functionality.
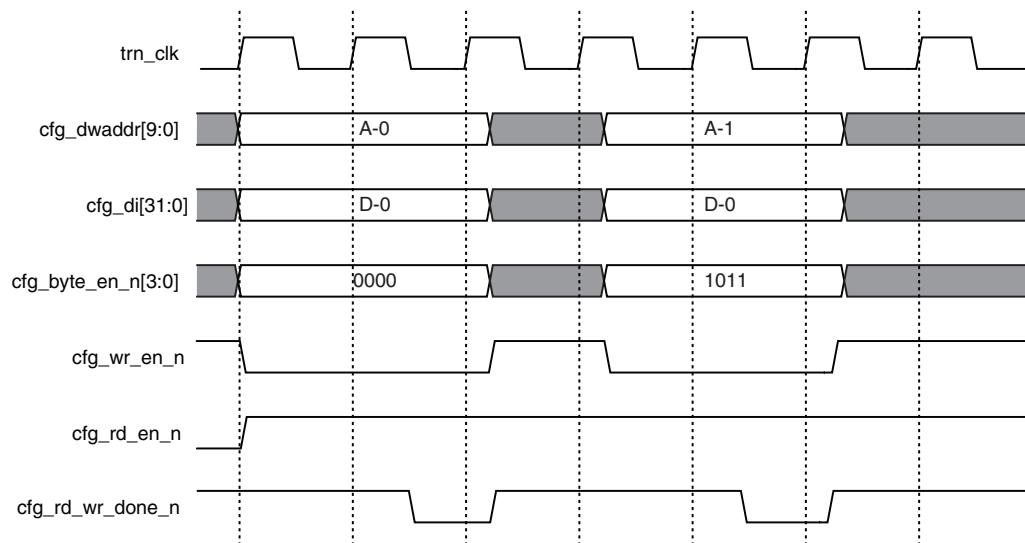


*Figure 6-56:* **Example Configuration Space Write Access**

## Optional PCI Express Extended Capabilities

The Virtex-6 FPGA Integrated Block for PCI Express optionally implements up to three PCI Express Extended Capabilities: Device Serial Number Capability, Virtual Channel Capability, and Vendor Specific Capability. Using the CORE Generator software, the user can choose which of these to enable. If the user wants to enable any of the Extended Capabilities, then the user must also enable the Device Serial Number Capability Structure. The relative order of the capabilities implemented is always the same. The order is:

1. Device Serial Number (DSN) Capability
2. Virtual Channel (VC) Capability
3. Vendor Specific (VSEC) Capability

Table 6-26 lists the Start addresses of the three Capability Structures, depending on the combination of PCI Express Extended Capabilities selected.

*Table 6-26:* **Start Addresses of the Extended Capabilities**

|  | No Capabilities Selected | DSN Only | DSN and VC | DSN and VSEC | All Three Capabilities Selected |
|---|---|---|---|---|---|
| DSN Base Pointer | - | `100 h` | `100 h` | `100 h` | `100 h` |
| VC Cap Base Pointer | - | - | `10C h` | - | `10C h` |
| VSEC Base Pointer | - | - | - | `10C h` | `128 h` |

The rest of the PCI Express Extended Configuration Space is optionally available for the Users to implement.

## Xilinx Defined Vendor Specific Capability

The Virtex-6 FPGA Integrated Block for PCI Express supports Xilinx defined Vendor Specific Capability that provides Control and Status for Loopback Master function for both the Root Port and Endpoint configurations. It is recommended that Loopback Master functionality be used only to perform in-system test of the physical link, when the application is not active. User logic is required to control the Loopback Master functionality by assessing the Vendor Specific Extended Capability (VSEC) structure via the Configuration Interface.

Figure 6-57 shows the VSEC structure in the PCIe Extended Configuration Space implemented in the integrated block.

31                                                                                           0    **Byte Offset**

| Next Capability Offset | Capability Version = `1h` | PCI Express extended capability = `000Bh` | `00h` |
|---|---|---|---|
| VSEC Length = 24 bytes | VSEC Rev = `0h` | VSEC ID = 0h | `04h` |
| Loopback Control Register | | | `08h` |
| Loopback Status Register | | | `0Ch` |
| Loopback Error Count Register 1 | | | `10h` |
| Loopback Error Count Register 2 | | | `14h` |

*Figure 6-57:* **Xilinx Defined Vendor Specific Capability Structure**

## Loopback Control Register (Offset 08h)

The Loopback Control Register controls Xilinx Defined Loopback specific parameters. Table 6-27 shows the bit locations and definitions.

*Table 6-27:*    **Loopback Control Register**

| Bit Location | Register Description | Attributes |
|:---:|---|:---:|
| 0 | Start Loopback: When set to 1b and *pl_ltssm_state[5:0]* is indicating L0 (16H), the block transitions to Loopback Master state and starts the loopback test. When set to 0b, the block exits the loopback master mode. | RW |
| 1 | Force Loopback: The loopback master can force the slave which fails to achieve symbol lock at specified "link speed" and "de-emphasis level" to enter loopback.active state by setting this bit to 1b. The start bit must be set to 1b when force is set to 1b. | RW |
| 3:2 | Loopback Link Speed: Advertised link speed in the TS1s sent by master with loopback bit set to 1b. The master can control the loopback link speed by properly controlling these bits. | RW |
| 4 | Loopback De-emphasis: Advertised de-emphasis level in the TS1s sent by master. This also sets the De-emphasis level for the loopback slave. | RW |
| 5 | Loopback Modified Compliance: The loopback master generates modified compliance pattern when in loopback mode else compliance pattern is generated. Only one SKP OS is generated instead of two while in modified compliance. | RW |
| 6 | Loopback Suppress SKP OS: When this bit is set to 1b then SKP OS are not transmitted by Loopback Master. This bit is ignored when send_modified_compliance pattern is set to 0b. | RW |
| 15:7 | Reserved | RO |
| 23:16 | Reserved | RO |
| 31:24 | Reserved | RO |

## Loopback Status Register (Offset 0Ch)

The Loopback Status Register provides information about Xilinx Defined Loopback specific parameters. Table 6-28 shows the bit locations and definitions.

*Table 6-28:*    **Loopback Status Register**

| Bit Location | Register Description | Attributes |
|:---:|---|:---:|
| 0 | Loopback Slave: This bit is set by hardware, if the device is currently in loopback slave mode. When this bit is set to 1b, the Start Loopback bit must not be set to 1b. | RO |
| 1 | Loopback Slave Failed: This bit is set by Loopback Master hardware, when the master receives no TS1's while Loopback bit set to 1b, within 100 ms of "Loopback.Active". This bit is never set to 1b, when the Force Loopback bit is set to 1b. Setting the Start Loopback bit to 1b clears this bit to 0b. | RO |

*Table 6-28:* **Loopback Status Register** *(Cont'd)*

| Bit Location | Register Description | Attributes |
|---|---|---|
| 7:2 | Reserved | RO |
| 15:8 | Loopback Tested: These bits are set to 0b, when the Start Loopback bit is set to 1b. These bits are set to 1b when loopback test has been performed on a given lane and the Loopback_Err_count_$n$ for the corresponding lane is valid.<br><br>Bit Positions    Lane<br>8                  Lane 0 Tested<br>9                  Lane 1 Tested<br>10               Lane 2 Tested<br>11               Lane 3 Tested<br>12               Lane 4 Tested<br>13               Lane 5 Tested<br>14               Lane 6 Tested<br>15               Lane 7 Tested | RO |
| 31:16 | Reserved | RO |

## Loopback Error Count Register 1 (Offset 10h)

The Loopback Error Count Register 1 provides information about the Error Count on the Physical Lanes 0 - 3, as tested by Xilinx Defined Loopback Control Test. A lane will have error count reported zero if that lane was not tested in loopback. This could be the case the lane is either not part of configured port or has not detected a receiver at the other end. Table 6-29 shows the bit locations and definitions.

*Table 6-29:* **Loopback Error Count Register 1**

| Bit Location | Register Description | Attributes |
|---|---|---|
| 7:0 | Loopback Error Count 0: This specifies the Error Count on Lane 0. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 0 Tested is set to 1b. | RO |
| 15:8 | Loopback Error Count 1: This specifies the Error Count on Lane 1. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 1 Tested is set to 1b. | RO |
| 23:16 | Loopback Error Count 2: This specifies the Error Count on Lane 2. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 2 Tested is set to 1b. | RO |
| 31:24 | Loopback Error Count 3: This specifies the Error Count on Lane 3. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 3 Tested is set to 1b. | RO |

## Loopback Error Count Register 2 (Offset 14h)

The Loopback Error Count Register 2 provides information about the Error Count on the Physical Lanes 7 - 4, as tested by Xilinx Defined Loopback Control Test. A lane will have error count reported zero if that lane was not tested in loopback. This could be the case the lane is either not part of configured port or has not detected a receiver at the other end. Table 6-30 shows the bit locations and definitions.

*Table 6-30:*    **Loopback Error Count Register 2**

| Bit Location | Register Description | Attributes |
|:---:|:---|:---:|
| 7:0 | Loopback Error Count 4: This specifies the Error Count on Lane 4. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 4 Tested is set to 1b. | RO |
| 15:8 | Loopback Error Count 5: This specifies the Error Count on Lane 5. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 5 Tested is set to 1b. | RO |
| 23:16 | Loopback Error Count 6: This specifies the Error Count on Lane 6. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the Lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 6 Tested is set to 1b. | RO |
| 31:24 | Loopback Error Count 7: This specifies the Error Count on Lane 7. An error is said to have occurred if there is an 8B/10B error or disparity error signaled on the lane. Setting Loopback Start bit to 1b clears the error count to 0h. This is only valid when Loopback Tested: Lane 7 Tested is set to 1b. | RO |

# User Implemented Configuration Space

The Virtex-6 FPGA Integrated Block for PCI Express enables users to optionally implement registers in the PCI Configuration Space, the PCI Express Extended Configuration Space, or both, in the User Application. The User Application is required to return Config Completions for all address within this space. For more information about enabling and customizing this feature, see Chapter 5, Generating and Customizing the Core.

## PCI Configuration Space

If the user chooses to implement registers within 0xA8 to 0xFF in the PCI Configuration Space, the start address of the address region they wish to implement can be defined during the core generation process.

The User Application is responsible for generating all Completions to Configuration Reads and Writes from the user-defined start address to the end of PCI Configuration Space (0xFF). Configuration Reads to unimplemented registers within this range should be responded to with a Completion with 0x00000000 as the data, and configuration writes should be responded to with a successful Completion.

For example, to implement address range `0xC0` to `0xCF`, there are several address ranges defined that should be treated differently depending on the access. See Table 6-31 for more details on this example.

*Table 6-31:* **Example: User Implemented Space 0xC0 to 0xCF**

|  | Configuration Writes | Configuration Reads |
|---|---|---|
| `0x00` to `0xBF` | Core responds automatically | Core responds automatically |
| `0xC0` to `0xCF` | User Application responds with Successful Completion | User Application responds with register contents |
| `0xD0` to `0xFF` | User Application responds with Successful Completion | User Application responds with `0x00000000` |

## PCI Express Extended Configuration Space

The starting address of the region in the PCI Express Extended Configuration Space that is optionally available for users to implement depends on the PCI Express Extended Capabilities that the user has enabled in the Virtex-6 FPGA Integrated Block for PCI Express, as shown in Table 6-32.

*Table 6-32:* **Min Start Addresses of the User Implemented Extended Capabilities**

|  | No Capabilities Selected | DSN Only | DSN and VC | DSN and VSEC | All Three Capabilities Selected |
|---|---|---|---|---|---|
| Starting byte address available | `100 h` | `10C h` | `128 h` | `124 h` | `140 h` |

The Virtex-6 FPGA Integrated Block for PCI Express allows the user to select the start address of the user implemented PCI Express Extended Configuration Space. This space must be implemented in the User Application. The User Application is required to generate a CplD with `0x00000000` for Configuration Read and successful Cpl for Configuration Write to addresses in this selected range not implemented in the user application.

The user can choose to implement a Configuration Space with a start address other than that allowed by the Virtex-6 FPGA Integrated Block for PCI Express. In such a case, the core returns a completion with `0x00000000` for configuration accesses to the region that the user has chosen to not implement. Table 6-33 further illustrates this scenario.

*Table 6-33:* **Example: User-Defined Start Address for Configuration Space**

| Configuration Space | Byte Address |
|---|---|
| DSN Capability | `100h - 108h` |
| VSEC Capability | `10Ch - 120h` |
| Reserved Extended Configuration Space (Core Returns Successful Completion with `0x00000000`) | `124h - 164h` |
| User Implemented PCI Express Extended Configuration Space | `168h - 47Ch` |
| User Implemented Reserved PCI Express Extended Configuration Space (User Application Returns Successful Completion with `0x00000000`) | `480h - FFFh` |

Table 6-33 illustrates an example Configuration of the PCI Express Extended Configuration Space, with these settings:

- DSN Capability Enabled
- VSEC Capability Enabled
- User Implemented PCI Express Extended Configuration Space Enabled
- User Implemented PCI Express Extended Configuration Space Start Address `168h`

In this configuration, the DSN Capability occupies the registers at `100h-108h`, and the VSEC Capability occupies registers at addresses `10Ch` to `120h`.

The remaining PCI Express Extended Configuration Space, starting at address `124h` is available to the user to implement. For this example, the user has chosen to implement registers in the address region starting `168h`.

In this scenario, the core returns successful Completions with `0x00000000` for Configuration accesses to registers `124h-164h`. Table 6-33 also illustrates a case where the user only implements the registers from `168h` to `47Ch`. In this case, the user is responsible for returning successful Completions with `0x00000000` for configuration accesses to `480h-FFFh`.

## Additional Packet Handling Requirements

The User Application must manage the following mechanisms to ensure protocol compliance, because the core does not manage them automatically.

### Generation of Completions

The Integrated Block core does not generate Completions for Memory Reads or I/O requests made by a remote device. The user is expected to service these completions according to the rules specified in the *PCI Express Base Specification*.

### Tracking Non-Posted Requests and Inbound Completions

The integrated block does not track transmitted I/O requests or Memory Reads that have yet to be serviced with inbound Completions. The User Application is required to keep track of such requests using the Tag ID or other information.

Keep in mind that one Memory Read request can be answered by several Completion packets. The User Application must accept all inbound Completions associated with the original Memory Read until all requested data has been received.

The *PCI Express Base Specification* requires that an endpoint advertise infinite Completion Flow Control credits as a receiver; the endpoint can only transmit Memory Reads and I/O requests if it has enough space to receive subsequent Completions.

The integrated block does not keep track of receive-buffer space for Completion. Rather, it sets aside a fixed amount of buffer space for inbound Completions. The User Application must keep track of this buffer space to know if it can transmit requests requiring a Completion response. See Appendix E, Managing Receive-Buffer Space for Inbound Completions for Inbound Completions for more information.

## Handling Message TLPs

By default, the Virtex-6 FPGA Integrated Block for PCI Express does not route any received messages to the Transaction Interface. It signals the receipt of messages on the cfg_msg_* interface. The user can, however, choose to receive these messages, in addition to signaling on this interface, by enabling this feature during customization of the core through the CORE Generator software.

### Root Port Configuration

The Root Port of a PCI Express Root Complex does not send any internally generated messages on the PCI Express link, although messages can still be sent via the Transaction Transmit Interface, such as a Set Slot Power Limit message. Any errors detected by the Integrated Block in Root Port configuration that could cause an error message to be sent are therefore signaled to the User Application on the cfg_msg_* interface.

The Integrated Block for PCI Express in Root Port configuration also decodes received messages and signals these to the User Application on this interface. When configured as a Root Port, the Integrated Block distinguishes between these received messages and error conditions detected internally by the asserting the cfg_msg_received signal.

## Reporting User Error Conditions

The User Application must report errors that occur during Completion handling using dedicated error signals on the core interface, and must observe the Device Power State before signaling an error to the core. If the User Application detects an error (for example, a Completion Timeout) while the device has been programmed to a non-D0 state, the User Application is responsible to signal the error after the device is programmed back to the D0 state.

After the User Application signals an error, the core reports the error on the PCI Express Link and also sets the appropriate status bit(s) in the Configuration Space. Because status bits must be set in the appropriate Configuration Space register, the User Application cannot generate error reporting packets on the transmit interface. The type of error-reporting packets transmitted depends on whether or not the error resulted from a Posted or Non-Posted Request. User-reported Posted errors cause Message packets to be sent to the Root Complex if enabled to do so through the Device Control Error Reporting bits and/or the Status SERR Enable bit. User-reported non-Posted errors cause Completion packets with non-successful status to be sent to the Root Complex, unless the error is regarded as an Advisory Non-Fatal Error. For more information about Advisory Non-Fatal Errors, see Chapter 6 of the *PCI Express Base Specification*. Errors on Non-Posted Requests can result in either Messages to the Root Complex or Completion packets with non-Successful status sent to the original Requester.

## Error Types

The User Application triggers six types of errors using the signals defined in Table 2-21, page 48.

- End-to-end CRC ECRC Error
- Unsupported Request Error
- Completion Timeout Error
- Unexpected Completion Error
- Completer Abort Error
- Correctable Error

Multiple errors can be detected in the same received packet; for example, the same packet can be an Unsupported Request and have an ECRC error. If this happens, only one error should be reported. Because all user-reported errors have the same severity, the User Application design can determine which error to report. The cfg_err_posted_n signal, combined with the appropriate error reporting signal, indicates what type of error-reporting packets are transmitted. The user can signal only one error per clock cycle. See Figure 6-58, Figure 6-59, and Figure 6-60, and Table 6-34 and Table 6-35.

The User Application must ensure that the device is in a D0 Power state prior to reporting any errors via the cfg_err_ interface. The User Application can ensure this by checking that the PMCSR PowerState (cfg_pmcsr_pme_powerstate[1:0]) is set to 2'b00. If the PowerState is not set to 2'b00 (the core is in a non-D0 power state) and PME_EN cfg_pmcsr_pme_en is asserted (1'b1), then the user can assert (pulse) cfg_pm_wake_n and wait for the Root to set the PMCSR PowerState bits to 2'b00. If the PowerState (cfg_pmcsr_pme_powerstate) is not equal to 2'b00 and PME_EN cfg_pmcsr_pme_en is deasserted (1'b0), the user must wait for the Root to set the PowerState to 2'b00.

*Table 6-34:* **User-indicated Error Signaling**

| Reported Error | cfg_err_posted_n | Action |
|---|---|---|
| None | Don't care | No Action Taken |
| cfg_err_ur_n | 0 or 1 | 0: If enabled, a Non-Fatal Error Message is sent.<br>1: A Completion with a status Unsupported Request is sent. |
| cfg_err_cpl_abort_n | 0 or 1 | 0: If enabled, a Non-Fatal Error message is sent.<br>1: A Completion with status Unsupported Request is sent. |
| cfg_err_cpl_timeout_n | Don't care | If enabled, a Non-Fatal Error Message is sent. |
| cfg_err_ecrc_n | Don't care | If enabled, a Non-Fatal Error Message is sent. |
| cfg_err_cor_n | Don't care | If enabled, a Correctable Error Message is sent. |
| cfg_err_cpl_unexpected_n | Don't care | Regarded as an Advisory Non-Fatal Error (ANFE); no action taken. |

*Table 6-35:* **Possible Error Conditions for TLPs Received by the User Application**

| | | Possible Error Condition | | | | | Error Qualifying Signal Status | |
|---|---|---|---|---|---|---|---|---|
| Received TLP Type | | Unsupported Request (cfg_err_ur_n) | Completion Abort (cfg_err_cpl_ abort_n) | Correctable Error (cfg_err_ cor_n) | ECRC Error (cfg_err_ ecrc_n) | Unexpected Completion (cfg_err_cpl_ unexpect_n) | Value to Drive on (cfg_err_ posted_n) | Drive Data on (cfg_err_tlp_ cpl_header[47:0]) |
| | Memory Write | ✓ | X | N/A | ✓ | X | 0 | No |
| | Memory Read | ✓ | ✓ | N/A | ✓ | X | 1 | Yes |
| | I/O | ✓ | ✓ | N/A | ✓ | X | 1 | Yes |
| | Completion | X | X | N/A | ✓ | ✓ | 0 | No |

**Notes:**
1. A checkmark indicates a possible error condition for a given TLP type. For example, users can signal Unsupported Request or ECRC Error for a Memory Write TLP, if these errors are detected. An X indicates not a valid error condition for a given TLP type. For example, users should never signal Completion Abort in response to a Memory Write TLP.

Whenever an error is detected in a Non-Posted Request, the User Application deasserts cfg_err_posted_n and provides header information on cfg_err_tlp_cpl_header[47:0] during the same clock cycle the error is reported, as illustrated in Figure 6-58. The additional header information is necessary to construct the required Completion with non-Successful status. Additional information about when to assert or deassert cfg_err_posted_n is provided in the remainder of this section.

If an error is detected on a Posted Request, the User Application instead asserts cfg_err_posted_n, but otherwise follows the same signaling protocol. This results in a Non-Fatal Message to be sent, if enabled.

If several non-Posted errors are signaled on cfg_err_ur_n or cfg_err_cpl_abort_n in a short amount of time, it is possible for the core to be unable to buffer them all. If that occurs, then cfg_err_cpl_rdy_n is deasserted and the user must cease signaling those types of errors on the same cycle. The user must not resume signaling those types of errors until cfg_err_cpl_rdy_n is reasserted.

The core's ability to generate error messages can be disabled by the Root Complex issuing a configuration write to the Endpoint core's Device Control register and the PCI Command register setting the appropriate bits to 0. For more information about these registers, see Chapter 7 of the *PCI Express Base Specification*. However, error-reporting status bits are always set in the Configuration Space whether or not their Messages are disabled.

*Internal signal not appearing on the User Interface
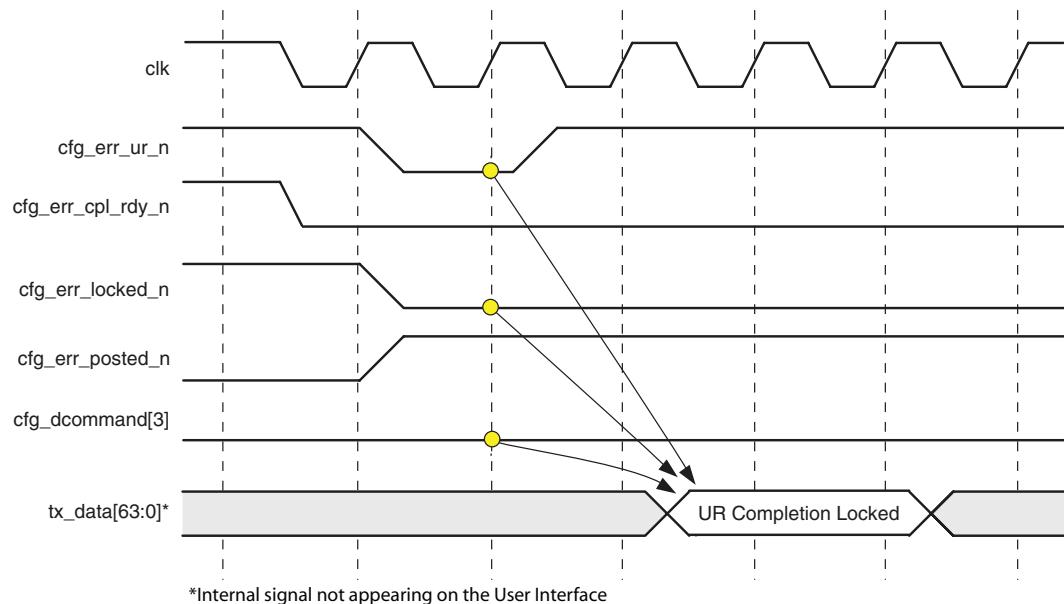
*Figure 6-58:* **Signaling Unsupported Request for Non-Posted TLP**



* Internal signals not appearing on User Interface
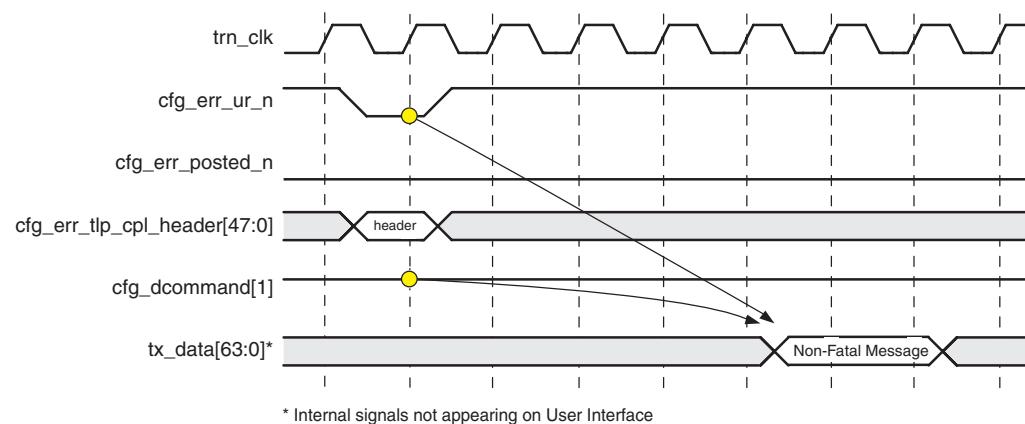
*Figure 6-59:* **Signaling Unsupported Request for Posted TLP**

*Internal signal not appearing on the User Interface
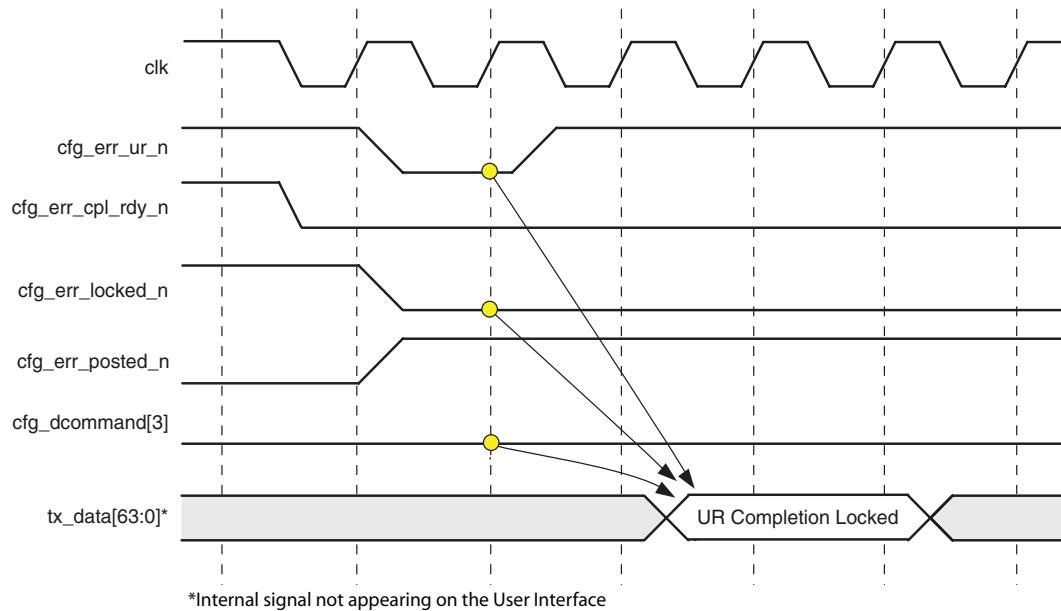
*Figure 6-60:* **Signaling Locked Unsupported Request for Locked Non-Posted TLP**

## Completion Timeouts

The Integrated Block core does not implement Completion timers; for this reason, the User Application must track how long its pending Non-Posted Requests have each been waiting for a Completion and trigger timeouts on them accordingly. The core has no method of knowing when such a timeout has occurred, and for this reason does not filter out inbound Completions for expired requests.

If a request times out, the User Application must assert cfg_err_cpl_timeout_n, which causes an error message to be sent to the Root Complex. If a Completion is later received after a request times out, the User Application must treat it as an Unexpected Completion.

## Unexpected Completions

The Integrated Block core automatically reports Unexpected Completions in response to inbound Completions whose Requestor ID is different than the Endpoint ID programmed in the Configuration Space. These completions are not passed to the User Application. The current version of the core regards an Unexpected Completion to be an Advisory Non-Fatal Error (ANFE), and no message is sent.

## Completer Abort

If the User Application is unable to transmit a normal Completion in response to a Non-Posted Request it receives, it must signal cfg_err_cpl_abort_n. The cfg_err_posted_n signal can also be set to 1 simultaneously to indicate Non-Posted and the appropriate request information placed on cfg_err_tlp_cpl_header[47:0]. This sends a Completion with non-Successful status to the original Requester, but does not send an Error Message. When in Legacy mode if the cfg_err_locked_n signal is set to 0 (to indicate the transaction causing the error was a locked transaction), a Completion Locked with Non-Successful status is sent. If the cfg_err_posted_n signal is set to 0 (to indicate a Posted transaction), no Completion is sent, but a Non-Fatal Error Message is sent (if enabled).

### Unsupported Request

If the User Application receives an inbound Request it does not support or recognize, it must assert cfg_err_ur_n to signal an Unsupported Request. The cfg_err_posted_n signal must also be asserted or deasserted depending on whether the packet in question is a Posted or Non-Posted Request. If the packet is Posted, a Non-Fatal Error Message is sent out (if enabled); if the packet is Non-Posted, a Completion with a non-Successful status is sent to the original Requester. When in Legacy mode if the cfg_err_locked_n signal is set to 0 (to indicate the transaction causing the error was a locked transaction), a Completion Locked with Unsupported Request status is sent.

The Unsupported Request condition can occur for several reasons, including:

- An inbound Memory Write packet violates the User Application's programming model, for example, if the User Application has been allotted a 4 KB address space but only uses 3 KB, and the inbound packet addresses the unused portion.

  *Note:* If this occurs on a Non-Posted Request, the User Application should use cfg_err_cpl_abort_n to flag the error.

- An inbound packet uses a packet Type not supported by the User Application, for example, an I/O request to a memory-only device.

### ECRC Error

The Integrated Block core does not check the ECRC field for validity. If the User Application chooses to check this field, and finds the CRC is in error, it can assert cfg_err_ecrc_n, causing a Non-Fatal Error Message to be sent.

# Power Management

The Integrated Block core supports these power management modes:

- Active State Power Management (ASPM)
- Programmed Power Management (PPM)

Implementing these power management functions as part of the PCI Express design enables the PCI Express hierarchy to seamlessly exchange power-management messages to save system power. All power management message identification functions are implemented. The subsections below describe the user logic definition to support the above modes of power management.

For additional information on ASPM and PPM implementation, see the *PCI Express Base Specification*.

## Active State Power Management

The Active State Power Management (ASPM) functionality is autonomous and transparent from a user-logic function perspective. The core supports the conditions required for ASPM. The Integrated Block supports ASPM L0s.

# Programmed Power Management

To achieve considerable power savings on the PCI Express hierarchy tree, the core supports these link states of Programmed Power Management (PPM):

- L0: Active State (data exchange state)
- L1: Higher Latency, lower power standby state
- L3: Link Off State

The Programmed Power Management Protocol is initiated by the Downstream Component / Upstream Port.

## PPM L0 State

The L0 state represents *normal* operation and is transparent to the user logic. The core reaches the L0 (active state) after a successful initialization and training of the PCI Express Link(s) as per the protocol.

## PPM L1 State

These steps outline the transition of the core to the PPM L1 state:

1. The transition to a lower power PPM L1 state is always initiated by an upstream device, by programming the PCI Express device power state to D3-hot (or to D1 or D2 if they are supported).

2. The device power state is communicated to the user logic through the cfg_pmcsr_powerstate[1:0] output.

3. The core then throttles/stalls the user logic from initiating any new transactions on the user interface by deasserting trn_tdst_rdy_n. Any pending transactions on the user interface are, however, accepted fully and can be completed later.

   There are two exceptions to this rule:

   - The core is configured as an Endpoint and the User Configuration Space is enabled. In this situation, the user must refrain from sending new Request TLPs if cfg_pmcsr_powerstate[1:0] indicates non-D0, but the user can return Completions to Configuration transactions targeting User Configuration space.

   - The core is configured as a Root Port. To be compliant in this situation, the user should refrain from sending new Requests if cfg_pmcsr_powerstate[1:0] indicates non-D0.

4. The core exchanges appropriate power management DLLPs with its link partner to successfully transition the link to a lower power PPM L1 state. This action is transparent to the user logic.

5. All user transactions are stalled for the duration of time when the device power state is non-D0, with the exceptions indicated in step 3.

***Note:*** The user logic, after identifying the device power state as non-D0, can initiate a request through the cfg_pm_wake_n to the upstream link partner to configure the device back to the D0 power state. If the upstream link partner has not configured the device to allow the generation of PM_PME messages (cfg_pmcsr_pme_en = 0), the assertion of cfg_pm_wake_n is ignored by the core.

## PPM L3 State

These steps outline the transition of the Endpoint for PCI Express to the PPM L3 state:

1. The core negotiates a transition to the L23 Ready Link State upon receiving a PME_Turn_Off message from the upstream link partner.

2. Upon receiving a PME_Turn_Off message, the core initiates a handshake with the user logic through cfg_to_turnoff_n (see Table 6-36) and expects a cfg_turnoff_ok_n back from the user logic.

3. A successful handshake results in a transmission of the Power Management Turn-off Acknowledge (PME-turnoff_ack) Message by the core to its upstream link partner.

4. The core closes all its interfaces, disables the Physical/Data-Link/Transaction layers and is ready for *removal* of power to the core.

   There are two exceptions to this rule:

   • The core is configured as an Endpoint and the User Configuration Space is enabled. In this situation, the user must refrain from sending new Request TLPs if cfg_pmcsr_powerstate[1:0] indicates non-D0, but the user can return Completions to Configuration transactions targeting User Configuration space.

   • The core is configured as a Root Port. TO be compliant in this situation, the user should refrain from sending new Requests if cfg_pmcsr_powerstate[1:0] indicates non-D0.

*Table 6-36:* **Power Management Handshaking Signals**

| Port Name | Direction | Description |
|---|---|---|
| cfg_to_turnoff_n | Output | Asserted if a power-down request TLP is received from the upstream device. After assertion, cfg_to_turnoff_n remains asserted until the user asserts cfg_turnoff_ok_n. |
| cfg_turnoff_ok_n | Input | Asserted by the User Application when it is safe to power down. |

Power-down negotiation follows these steps:

1. Before power and clock are turned off, the Root Complex or the Hot-Plug controller in a downstream switch issues a PME_Turn_Off broadcast message.

2. When the core receives this TLP, it asserts cfg_to_turnoff_n to the User Application and starts polling the cfg_turnoff_ok_n input.

3. When the User Application detects the assertion of cfg_to_turnoff_n, it must complete any packet in progress and stop generating any new packets. After the User Application is ready to be turned off, it asserts cfg_turnoff_ok_n to the core. After assertion of cfg_turnoff_ok_n, the User Application has committed to being turned off.

4. The core sends a PME_TO_Ack when it detects assertion of cfg_turnoff_ok_n, as displayed in Figure 6-61 (64-bit).



* Internal signals not appearing on User Interface
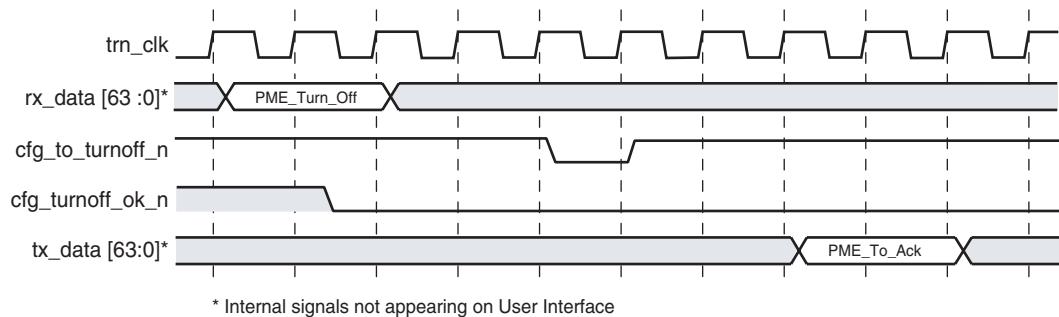
*Figure 6-61:* **Power Management Handshaking: 64-bit**

# Generating Interrupt Requests

**Note:** This section is only applicable to the Endpoint Configuration of the Virtex-6 FPGA Integrated Block for PCI Express.

The Integrated Block core supports sending interrupt requests as either legacy interrupts or Message Signaled Interrupts (MSI), or MSI-X interrupts. The mode is programmed using the MSI Enable bit in the Message Control Register of the MSI Capability Structure and the MSI-X Enable bit in the MSI-X Message Control Register of the MSI-X Capability Structure. For more information on the MSI and MSI-X capability structures, see section 6.8 of the *PCI Local Base Specification v3.0*.

The state of the MSI Enable and MSI-X Enabled bits are reflected by the cfg_interrupt_msienable and cfg_interrupt_msixeable outputs, respectively. Table 6-37 describes the Interrupt Mode the device has been programmed to, based on the cfg_interrupt_msienable and cfg_interrupt_msixenable outputs of the core.

*Table 6-37:* **Interrupt Modes**

|  | cfg_interrupt_msixenable=0 | cfg_interrupt_msixenable=1 |
|---|---|---|
| **cfg_interrupt_ msienable=0** | Legacy Interrupt (INTx) mode. The cfg_interrupt interface only sends INTx messages. | MSI-X mode. MSI-X interrupts must be generated by the user by composing MWr TLPs on the TRN TX interface; Do not use the cfg_interrupt interface. The cfg_interrupt interface is active and sends INTx messages, but the user should refrain from doing so. |
| **cfg_interrupt_ msienable=1** | MSI mode. The cfg_interrupt interface only sends MSI interrupts (MWr TLPs). | Undefined. System software is not supposed to permit this. However, the cfg_interrupt interface is active and sends MSI interrupts (MWr TLPs) if the user chooses to do so. |

The MSI Enable bit in the MSI control register, the MSI-X Enable bit in the MSI-X Control Register, and the Interrupt Disable bit in the PCI Command register are programmed by the Root Complex. The User Application has no direct control over these bits.

The Internal Interrupt Controller in the Virtex-6 FPGA Integrated Block for PCI Express core only generates Legacy Interrupts and MSI Interrupts. MSI-X Interrupts need to be generated by the User Application and presented on the TRN TX Interface. The status of

cfg_interrupt_msienable determines the type of interrupt generated by the internal Interrupt Controller:

If the MSI Enable bit is set to a 1, then the core generates MSI requests by sending Memory Write TLPs. If the MSI Enable bit is set to 0, the core generates legacy interrupt messages as long as the Interrupt Disable bit in the PCI Command Register is set to 0:

- cfg_command[10] = 0: INTx interrupts enabled
- cfg_command[10] = 1: INTx interrupts disabled (request are blocked by the core)
- cfg_interrupt_msienable = 0: Legacy Interrupt
- cfg_interrupt_msienable = 1: MSI

Regardless of the interrupt type used (Legacy or MSI), the user initiates interrupt requests through the use of cfg_interrupt_n and cfg_interrupt_rdy_n as shown in Table 6-38.

*Table 6-38:*   **Interrupt Signalling**

| Port Name | Direction | Description |
|---|---|---|
| cfg_interrupt_n | Input | Assert to request an interrupt. Leave asserted until the interrupt is serviced. |
| cfg_interrupt_rdy_n | Output | Asserted when the core accepts the signaled interrupt request. |

The User Application requests interrupt service in one of two ways, each of which are described below.

## Legacy Interrupt Mode

- As shown in Figure 6-62, the User Application first asserts cfg_interrupt_n and cfg_interrupt_assert_n to assert the interrupt. The User application should select a specific interrupt (INTA) using cfg_interrupt_di[7:0] as shown in Table 6-39.

- The core then asserts cfg_interrupt_rdy_n to indicate the interrupt has been accepted. On the following clock cycle, the User Application deasserts cfg_interrupt_n and, if the Interrupt Disable bit in the PCI Command register is set to 0, the core sends an assert interrupt message (Assert_INTA, Assert_INTB, and so forth).

- Once the User Application has determined that the interrupt has been serviced, it asserts cfg_interrupt_n while deasserting cfg_interrupt_assert_n to deassert the interrupt. The appropriate interrupt must be indicated via cfg_interrupt_di[7:0].

- The core then asserts cfg_interrupt_rdy_n to indicate the interrupt deassertion has been accepted. On the following clock cycle, the User Application deasserts cfg_interrupt_n and the core sends a deassert interrupt message (Deassert_INTA, Deassert_INTB, and so forth).
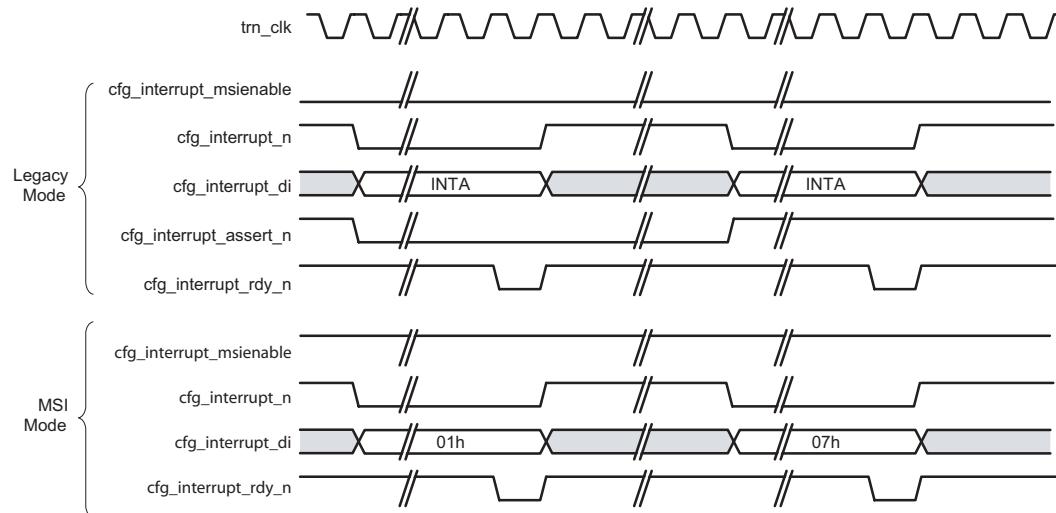
*Figure 6-62:* **Requesting Interrupt Service: MSI and Legacy Mode**

*Table 6-39:* **Legacy Interrupt Mapping**

| cfg_interrupt_di[7:0] value | Legacy Interrupt |
|---|---|
| 00h | INTA |
| 01h - FFh | Not Supported |

## MSI Mode

- As shown in Figure 6-62, the User Application first asserts cfg_interrupt_n. Additionally the User Application supplies a value on cfg_interrupt_di[7:0] if Multi-Vector MSI is enabled (see below).

- The core asserts cfg_interrupt_rdy_n to signal that the interrupt has been accepted and the core sends a MSI Memory Write TLP. On the following clock cycle, the User Application deasserts cfg_interrupt_n if no further interrupts are to be sent.

The MSI request is either a 32-bit addressable Memory Write TLP or a 64-bit addressable Memory Write TLP. The address is taken from the Message Address and Message Upper Address fields of the MSI Capability Structure, while the payload is taken from the Message Data field. These values are programmed by system software through configuration writes to the MSI Capability structure. When the core is configured for Multi-Vector MSI, system software can permit Multi-Vector MSI messages by programming a non-zero value to the Multiple Message Enable field.

The type of MSI TLP sent (32-bit addressable or 64-bit addressable) depends on the value of the Upper Address field in the MSI capability structure. By default, MSI messages are sent as 32-bit addressable Memory Write TLPs. MSI messages use 64-bit addressable Memory Write TLPs only if the system software programs a non-zero value into the Upper Address register.

When Multi-Vector MSI messages are enabled, the User Application can override one or more of the lower-order bits in the Message Data field of each transmitted MSI TLP to differentiate between the various MSI messages sent upstream. The number of lower-order bits in the Message Data field available to the User Application is determined by the lesser of the value of the Multiple Message Capable field, as set in the CORE Generator software, and the Multiple Message Enable field, as set by system software and available as the

cfg_interrupt_mmenable[2:0] core output. The core masks any bits in cfg_interrupt_di[7:0] which are not configured by system software via Multiple Message Enable.

This pseudo-code shows the processing required:

```
// Value MSI_Vector_Num must be in range: 0 ≤ MSI_Vector_Num ≤
(2^cfg_interrupt_mmenable)-1

if (cfg_interrupt_msienable) {         // MSI Enabled
  if (cfg_interrupt_mmenable > 0) {  // Multi-Vector MSI Enabled
    cfg_interrupt_di[7:0] = {Padding_0s, MSI_Vector_Num};
  } else {                            // Single-Vector MSI Enabled
    cfg_interrupt_di[7:0] = Padding_0s;
  }
} else {
  // Legacy Interrupts Enabled
}
```

For example:

1. If cfg_interrupt_mmenable[2:0] == 000b, i.e., 1 MSI Vector Enabled, then cfg_interrupt_di[7:0] = 00h;

2. if cfg_interrupt_mmenable[2:0] == 101b, i.e., 32 MSI Vectors Enabled, then cfg_interrupt_di[7:0] = {{000b}, {MSI_Vector#}};

where MSI_Vector# is a 5-bit value and is allowed to be 00000b ≤ MSI_Vector# ≤ 11111b.

If Per-Vector Masking is enabled, the user must first verify that the vector being signaled is not masked in the Mask register. This is done by reading this register on the Configuration Interface (the core does not look at the Mask register).

### MSI-X Mode

The Virtex-6 FPGA Integrated Block for PCI Express optionally supports the MSI-X Capability Structure. The MSI-X vector table and the MSI-X Pending Bit Array need to be implemented as part of the user's logic, by claiming a BAR aperture.

If the cfg_interrupt_msixenable output of the core is asserted, the User Application should compose and present the MSI-X interrupts on the TRN TX Interface.

# Link Training: 2-Lane, 4-Lane, and 8-Lane Components

The 2-Lane, 4-lane, and 8-lane Integrated Block for PCI Express can operate at less than the maximum lane width as required by the *PCI Express Base Specification*. Two cases cause core to operate at less than its specified maximum lane width, as defined in these subsections.

### Link Partner Supports Fewer Lanes

When the 2-lane core is connected to a device that implements only 1 lane, the 2-lane core trains and operates as a 1-lane device using lane 0.

When the 4-lane core is connected to a device that implements 1 lane, the 4-lane core trains and operates as a 1-lane device using lane 0, as shown in Figure 6-63. Similarly, if the 4-lane core is connected to a 2-lane device, the core trains and operates as a 2-lane device using lanes 0 and 1.

When the 8-lane core is connected to a device that only implements 4 lanes, it trains and operates as a 4-lane device using lanes 0-3. Additionally, if the connected device only implements 1 or 2 lanes, the 8-lane core trains and operates as a 1- or 2-lane device.
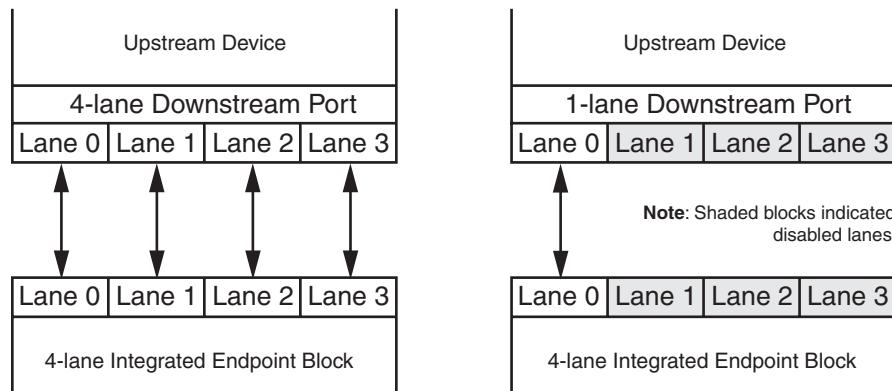


*Figure 6-63:* **Scaling of 4-Lane Endpoint Core from 4-Lane to 1-Lane Operation**

## Lane Becomes Faulty

If a link becomes faulty after training to the maximum lane width supported by the core and the link partner device, the core attempts to recover and train to a lower lane width, if available. If lane 0 becomes faulty, the link is irrecoverably lost. If any or all of lanes 1–7 become faulty, the link goes into *recovery* and attempts to recover the largest viable link with whichever lanes are still operational.

For example, when using the 8-lane core, loss of lane 1 yields a recovery to 1-lane operation on lane 0, whereas the loss of lane 6 yields a recovery to 4-lane operation on lanes 0-3. After recovery occurs, if the failed lane(s) becomes *alive* again, the core does not attempt to recover to a wider link width. The only way a wider link width can occur is if the link actually goes down and it attempts to retrain from scratch.

The trn_clk clock output is a fixed frequency configured in the CORE Generator software GUI. trn_clk does not shift frequencies in case of link recovery or training down.

# Lane Reversal

The integrated Endpoint block supports limited lane reversal capabilities and therefore provides flexibility in the design of the board for the link partner. The link partner can choose to lay out the board with reversed lane numbers and the integrated Endpoint block continues to link train successfully and operate normally. The configurations that have lane reversal support are x8 and x4 (excluding downshift modes). Downshift refers to the link width negotiation process that occurs when link partners have different lane width capabilities advertised. As a result of lane width negotiation, the link partners negotiate down to the smaller of the two advertised lane widths. Table 6-40 describes the several possible combinations including downshift modes and availability of lane reversal support.

*Table 6-40:*   **Lane Reversal Support**

| Endpoint Block Advertised Lane Width | Negotiated Lane Width | Lane Number Mapping (Endpoint Link Partner) | | Lane Reversal Supported |
|---|---|---|---|---|
| | | **Endpoint** | **Link Partner** | |
| x8 | x8 | Lane 0 ... Lane 7 | Lane 7 ... Lane 0 | Yes |
| x8 | x4 | Lane 0 ... Lane 3 | Lane 7 ... Lane 4 | No[1] |
| x8 | x2 | Lane 0 ... Lane 3 | Lane 7 ... Lane 6 | No[1] |
| x4 | x4 | Lane 0 ... Lane 3 | Lane 3 ... Lane 0 | Yes |
| x4 | x2 | Lane 0 ... Lane 1 | Lane 3 ... Lane 2 | No[1] |
| x2 | x2 | Lane 0 ... Lane 1 | Lane 1... Lane 0 | Yes |
| x2 | x1 | Lane 0 ... Lane 1 | Lane 1 | No[1] |

**Notes:**

1. When the lanes are reversed in the board layout and a downshift adapter card is inserted between the Endpoint and link partner, Lane 0 of the link partner remains unconnected (as shown by the lane mapping in Table 6-40) and therefore does not link train.

# Clocking and Reset of the Integrated Block Core

## Reset

The Virtex-6 FPGA Integrated Block for PCI Express core uses sys_reset_n to reset the system, an asynchronous, active-Low reset signal asserted during the PCI Express Fundamental Reset. Asserting this signal causes a hard reset of the entire core, including the GTX transceivers. After the reset is released, the core attempts to link train and resume normal operation. In a typical endpoint application, for example, an add-in card, a sideband reset signal is normally present and should be connected to sys_reset_n. For Endpoint applications that do not have a sideband system reset signal, the initial hardware reset should be generated locally. Three reset events can occur in PCI Express:

- **Cold Reset**. A Fundamental Reset that occurs at the application of power. The signal sys_reset_n is asserted to cause the cold reset of the core.

- **Warm Reset**. A Fundamental Reset triggered by hardware without the removal and re-application of power. The sys_reset_n signal is asserted to cause the warm reset to the core.

- **Hot Reset**: In-band propagation of a reset across the PCI Express Link through the protocol. In this case, sys_reset_n is not used. In the case of Hot Reset, the received_hot_reset signal is asserted to indicate the source of the reset.

The User Application interface of the core has an output signal called trn_reset_n. This signal is deasserted synchronously with respect to trn_clk. trn_reset_n is asserted as a result of any of these conditions:

- **Fundamental Reset**: Occurs (cold or warm) due to assertion of sys_reset_n.
- **PLL within the core wrapper**: Loses lock, indicating an issue with the stability of the clock input.
- **Loss of Transceiver PLL Lock:** Any transceiver loses lock, indicating an issue with the PCI Express Link.

The trn_reset_n signal deasserts synchronously with trn_clk after all of the above reasons are resolved, allowing the core to attempt to train and resume normal operation.

**Important Note**: Systems designed to the PCI Express electro-mechanical specification provide a sideband reset signal, which uses 3.3V signaling levels—see the FPGA device data sheet to understand the requirements for interfacing to such signals.

## Clocking

The Integrated Block input system clock signal is called sys_clk. The core requires a 100 MHz,125 MHz, or 250 MHz clock input. The clock frequency used must match the clock frequency selection in the CORE Generator software GUI. For more information, see Answer Record 18329.

In a typical PCI Express solution, the PCI Express reference clock is a Spread Spectrum Clock (SSC), provided at 100 MHz. In most commercial PCI Express systems, SSC cannot be disabled. For more information regarding SSC and PCI Express, see section 4.3.1.1.1 of the *PCI Express Base Specification*.

### Synchronous and Non-Synchronous Clocking

There are two ways to clock the PCI Express system:
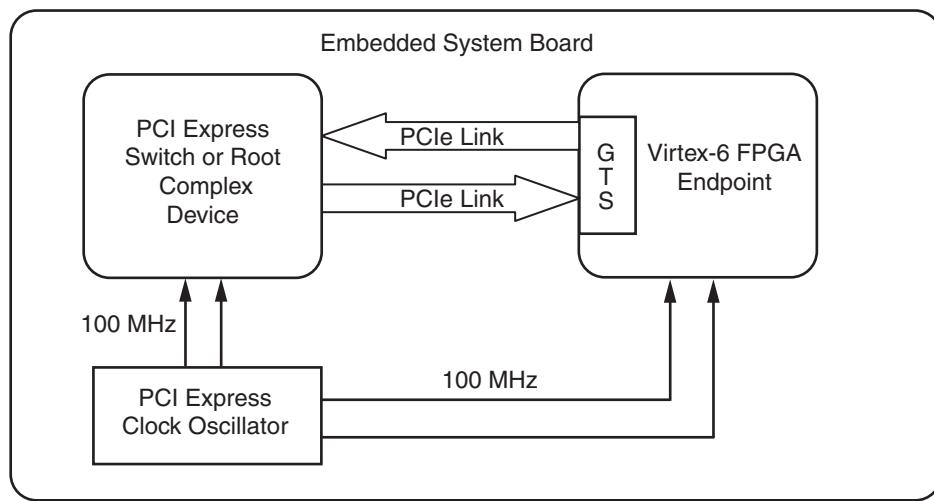
- Using synchronous clocking, where a shared clock source is used for all devices.
- Using non-synchronous clocking, where each device has its own clock source. ASPM must not be used in systems with non-synchronous clocking.

  **Important Note**: Xilinx recommends that designers use synchronous clocking when using the core. All add-in card designs must use synchronous clocking due to the characteristics of the provided reference clock. For devices using the Slot clock, the "Slot Clock Configuration" setting in the Link Status Register must be enabled in the CORE Generator software GUI. See the *Virtex-6 FPGA GTX Transceivers User Guide* and device data sheet for additional information regarding reference clock requirements.

  For synchronous clocked systems, each link partner device shares the same clock source. Figure 6-64 and Figure 6-66 show a system using a 100 MHz reference clock. When using the 125 MHz or the 250 MHz reference clock option, an external PLL must be used to do a multiply of 5/4 and 5/2 to convert the 100 MHz clock to 125 MHz and 250 MHz respectively, as illustrated in Figure 6-65 and Figure 6-67. See Answer Record 18329 for more information about clocking requirements.
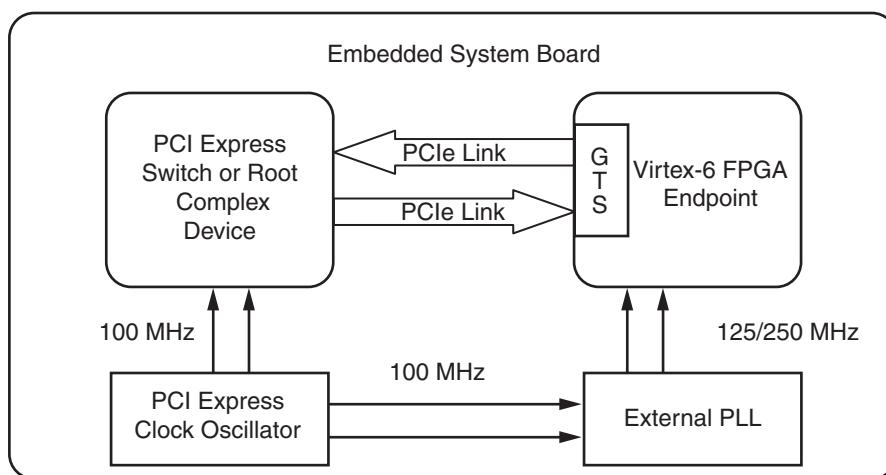
  Further, even if the device is part of an embedded system, if the system uses commercial PCI Express root complexes or switches along with typical motherboard clocking schemes, synchronous clocking should still be used as shown in Figure 6-64 and Figure 6-65.

Figure 6-64 through Figure 6-67 illustrate high-level representations of the board layouts. Designers must ensure that proper coupling, termination, and so forth are used when laying out the board.



UG517_c6_64_021010

*Figure 6-64:* **Embedded System Using 100 MHz Reference Clock**



UG517_c6_65_021010

*Figure 6-65:* **Embedded System Using 125/250 MHz Reference Clock**
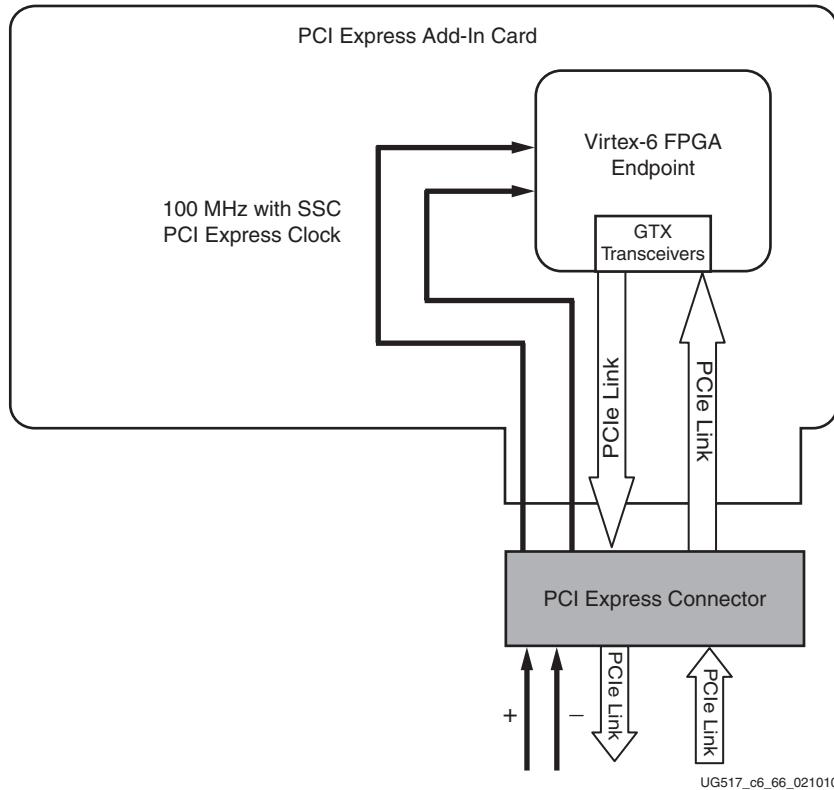
*Figure 6-66:* **Open System Add-In Card Using 100 MHz Reference Clock**
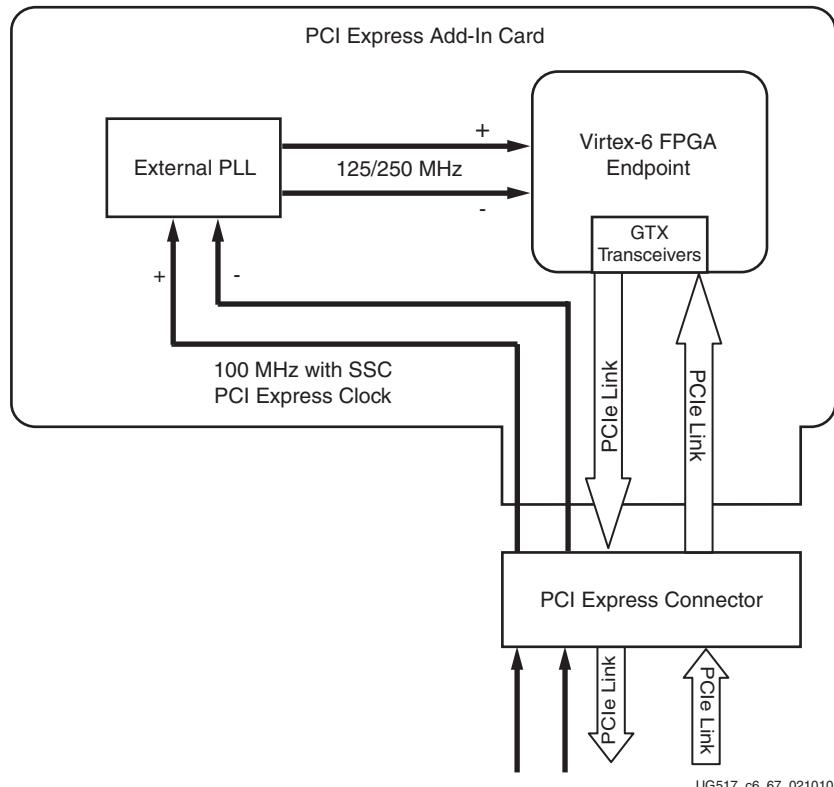


*Figure 6-67:* **Open System Add-In Card Using 125/250 MHz Reference Clock**

# Using the Dynamic Reconfiguration Port Interface

The Dynamic Reconfiguration Port (DRP) interface allows read and write access to the FPGA configuration memory bits of the integrated block instantiated as part of the core. These configuration memory bits are represented as attributes of the PCIE_2_0 library element. Descriptions of these attributes are found in *Virtex-6 FPGA Integrated Block for PCI Express Designs Reference Designs*.

The DRP interface is a standard interface found on many integrated IP blocks in Xilinx devices. For detailed information about how the DRP interface works with the FPGA configuration memory, see the *Virtex-6 FPGA Configuration Guide*.

## Writing and Reading the DRP Interface

The interface is a processor-friendly synchronous interface with an address bus (drp_daddr) and separated data buses for reading (drp_do) and writing (drp_di) configuration data to the PCIE_2_0 block. An enable signal (drp_den), a read/write signal (drp_dwe), and a ready/valid signal (drp_drdy) are the control signals that implement read and write operations, indicate operation completion, or indicate the availability of data. Figure 6-68 shows a write cycle, and Figure 6-69 shows a read cycle.
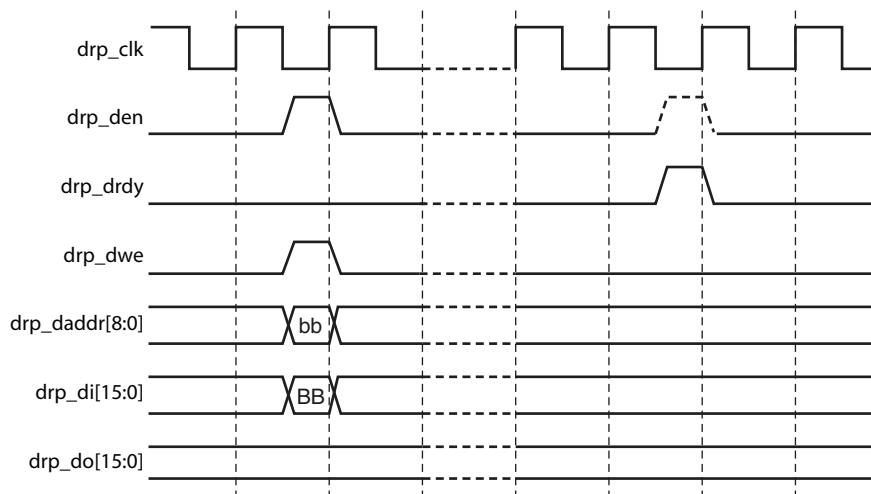


*Figure 6-68:*   **DRP Interface Write Cycle**
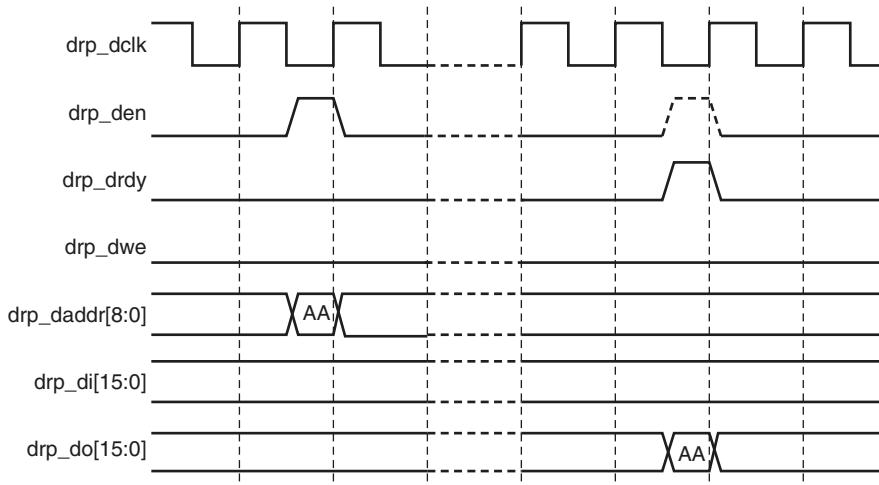
*Figure 6-69:* **DRP Interface Read Cycle**

## Other Considerations for the DRP Interface

Updating attribute values through the DRP port is only supported while the core is in reset with sys_reset_n asserted. Behavior of the core is undefined if attributes are updated on-the-fly with sys_rst_n deasserted. Reading attributes through the DRP port is independent of sys_rst_n.

Attributes larger than 16 bits span two drp_daddr addresses, for example BAR0[31:0] requires two accesses to read or write the attribute. Additionally, some attributes share a single drp_daddr address. The user should employ a read-modify-write approach so that shared-address attributes are not modified unintentionally.

There are a number of attributes that should not be modified via DRP, because these attributes need to be set in an aligned manner with the rest of the design. For example, changing the memory latency attributes on the PCIE_2_0 block without changing the actual number of pipeline registers attached to the block RAM causes a functional failure. These attributes are included in this category:

- DEV_CAP_MAX_PAYLOAD_SUPPORTED
- VC0_TX_LASTPACKET
- TL_TX_RAM_RADDR_LATENCY
- TL_TX_RAM_RDATA_LATENCY
- TL_TX_RAM_WRITE_LATENCY
- VC0_RX_LIMIT
- TL_RX_RAM_RADDR_LATENCY
- TL_RX_RAM_RDATA_LATENCY
- TL_RX_RAM_WRITE_LATENCY

## DRP Address Map

Table 6-41 defines the DRP address map for the `PCIE_2_0` library element attributes. Some attributes span two addresses, for example, `BAR0`. In addition, some addresses contain multiple attributes, for example address `0x01d` contains both DSN_CAP_NEXTPTR[11:0] and DSN_CAP_ON.

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes**

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| AER_CAP_ECRC_CHECK_CAPABLE | 0x000 | [0] |
| AER_CAP_ECRC_GEN_CAPABLE | 0x000 | [1] |
| AER_CAP_ID[15:0] | 0x001 | [15:0] |
| AER_CAP_INT_MSG_NUM_MSI[4:0] | 0x002 | [4:0] |
| AER_CAP_INT_MSG_NUM_MSIX[4:0] | 0x002 | [9:5] |
| AER_CAP_PERMIT_ROOTERR_UPDATE | 0x002 | [10] |
| AER_CAP_VERSION[3:0] | 0x002 | [14:11] |
| AER_BASE_PTR[11:0] | 0x003 | [11:0] |
| AER_CAP_NEXTPTR[11:0] | 0x004 | [11:0] |
| AER_CAP_ON | 0x004 | [12] |
| BAR0[15:0] | 0x005 | [15:0] |
| BAR0[31:16] | 0x006 | [15:0] |
| BAR1[15:0] | 0x007 | [15:0] |
| BAR1[31:16] | 0x008 | [15:0] |
| BAR2[15:0] | 0x009 | [15:0] |
| BAR2[31:16] | 0x00a | [15:0] |
| BAR3[15:0] | 0x00b | [15:0] |
| BAR3[31:16] | 0x00c | [15:0] |
| BAR4[15:0] | 0x00d | [15:0] |
| BAR4[31:16] | 0x00e | [15:0] |
| BAR5[15:0] | 0x00f | [15:0] |
| BAR5[31:16] | 0x010 | [15:0] |
| EXPANSION_ROM[15:0] | 0x011 | [15:0] |
| EXPANSION_ROM[31:16] | 0x012 | [15:0] |
| CAPABILITIES_PTR[7:0] | 0x013 | [7:0] |
| CARDBUS_CIS_POINTER[15:0] | 0x014 | [15:0] |
| CARDBUS_CIS_POINTER[31:16] | 0x015 | [15:0] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| CLASS_CODE[15:0] | 0x016 | [15:0] |
| CLASS_CODE[23:16] | 0x017 | [7:0] |
| CMD_INTX_IMPLEMENTED | 0x017 | [8] |
| CPL_TIMEOUT_DISABLE_SUPPORTED | 0x017 | [9] |
| CPL_TIMEOUT_RANGES_SUPPORTED[3:0] | 0x017 | [13:10] |
| DEV_CAP_ENABLE_SLOT_PWR_LIMIT_SCALE | 0x017 | [14] |
| DEV_CAP_ENABLE_SLOT_PWR_LIMIT_VALUE | 0x017 | [15] |
| DEV_CAP_ENDPOINT_L0S_LATENCY[2:0] | 0x018 | [2:0] |
| DEV_CAP_ENDPOINT_L1_LATENCY[2:0] | 0x018 | [5:3] |
| DEV_CAP_EXT_TAG_SUPPORTED | 0x018 | [6] |
| DEV_CAP_FUNCTION_LEVEL_RESET_CAPABLE | 0x018 | [7] |
| DEV_CAP_MAX_PAYLOAD_SUPPORTED[2:0] | 0x018 | [10:8] |
| DEV_CAP_PHANTOM_FUNCTIONS_SUPPORT[1:0] | 0x018 | [12:11] |
| DEV_CAP_ROLE_BASED_ERROR | 0x018 | [13] |
| DEV_CAP_RSVD_14_12[2:0] | 0x019 | [2:0] |
| DEV_CAP_RSVD_17_16[1:0] | 0x019 | [4:3] |
| DEV_CAP_RSVD_31_29[2:0] | 0x019 | [7:5] |
| DEV_CONTROL_AUX_POWER_SUPPORTED | 0x019 | [8] |
| DEVICE_ID[15:0] | 0x01a | [15:0] |
| DSN_BASE_PTR[11:0] | 0x01b | [11:0] |
| DSN_CAP_ID[15:0] | 0x01c | [15:0] |
| DSN_CAP_NEXTPTR[11:0] | 0x01d | [11:0] |
| DSN_CAP_ON | 0x01d | [12] |
| DSN_CAP_VERSION[3:0] | 0x01e | [3:0] |
| EXT_CFG_CAP_PTR[5:0] | 0x01e | [9:4] |
| EXT_CFG_XP_CAP_PTR[9:0] | 0x01f | [9:0] |
| HEADER_TYPE[7:0] | 0x020 | [7:0] |
| INTERRUPT_PIN[7:0] | 0x020 | [15:8] |
| IS_SWITCH | 0x021 | [0] |
| LAST_CONFIG_DWORD[9:0] | 0x021 | [10:1] |
| LINK_CAP_ASPM_SUPPORT[1:0] | 0x021 | [12:11] |
| LINK_CAP_CLOCK_POWER_MANAGEMENT | 0x021 | [13] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| LINK_CAP_DLL_LINK_ACTIVE_REPORTING_CAP | 0x021 | [14] |
| LINK_CAP_L0S_EXIT_LATENCY_COMCLK_GEN1[2:0] | 0x022 | [2:0] |
| LINK_CAP_L0S_EXIT_LATENCY_COMCLK_GEN2[2:0] | 0x022 | [5:3] |
| LINK_CAP_L0S_EXIT_LATENCY_GEN1[2:0] | 0x022 | [8:6] |
| LINK_CAP_L0S_EXIT_LATENCY_GEN2[2:0] | 0x022 | [11:9] |
| LINK_CAP_L1_EXIT_LATENCY_COMCLK_GEN1[2:0] | 0x022 | [14:12] |
| LINK_CAP_L1_EXIT_LATENCY_COMCLK_GEN2[2:0] | 0x023 | [2:0] |
| LINK_CAP_L1_EXIT_LATENCY_GEN1[2:0] | 0x023 | [5:3] |
| LINK_CAP_L1_EXIT_LATENCY_GEN2[2:0] | 0x023 | [8:6] |
| LINK_CAP_LINK_BANDWIDTH_NOTIFICATION_CAP | 0x023 | [9] |
| LINK_CAP_MAX_LINK_SPEED[3:0] | 0x023 | [13:10] |
| LINK_CAP_RSVD_23_22[1:0] | 0x023 | [15:14] |
| LINK_CAP_SURPRISE_DOWN_ERROR_CAPABLE | 0x024 | [0] |
| LINK_CONTROL_RCB | 0x024 | [1] |
| LINK_CTRL2_DEEMPHASIS | 0x024 | [2] |
| LINK_CTRL2_HW_AUTONOMOUS_SPEED_DISABLE | 0x024 | [3] |
| LINK_CTRL2_TARGET_LINK_SPEED[3:0] | 0x024 | [7:4] |
| LINK_STATUS_SLOT_CLOCK_CONFIG | 0x024 | [8] |
| MSI_BASE_PTR[7:0] | 0x025 | [7:0] |
| MSI_CAP_64_BIT_ADDR_CAPABLE | 0x025 | [8] |
| MSI_CAP_ID[7:0] | 0x026 | [7:0] |
| MSI_CAP_MULTIMSG_EXTENSION | 0x026 | [8] |
| MSI_CAP_MULTIMSGCAP[2:0] | 0x026 | [11:9] |
| MSI_CAP_NEXTPTR[7:0] | 0x027 | [7:0] |
| MSI_CAP_ON | 0x027 | [8] |
| MSI_CAP_PER_VECTOR_MASKING_CAPABLE | 0x027 | [9] |
| MSIX_BASE_PTR[7:0] | 0x028 | [7:0] |
| MSIX_CAP_ID[7:0] | 0x028 | [15:8] |
| MSIX_CAP_NEXTPTR[7:0] | 0x029 | [7:0] |
| MSIX_CAP_ON | 0x029 | [8] |
| MSIX_CAP_PBA_BIR[2:0] | 0x029 | [11:9] |
| MSIX_CAP_PBA_OFFSET[15:0] | 0x02a | [15:0] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| MSIX_CAP_PBA_OFFSET[28:16] | 0x02b | [12:0] |
| MSIX_CAP_TABLE_BIR[2:0] | 0x02b | [15:13] |
| MSIX_CAP_TABLE_OFFSET[15:0] | 0x02c | [15:0] |
| MSIX_CAP_TABLE_OFFSET[28:16] | 0x02d | [12:0] |
| MSIX_CAP_TABLE_SIZE[10:0] | 0x02e | [10:0] |
| PCIE_BASE_PTR[7:0] | 0x02f | [7:0] |
| PCIE_CAP_CAPABILITY_ID[7:0] | 0x02f | [15:8] |
| PCIE_CAP_CAPABILITY_VERSION[3:0] | 0x030 | [3:0] |
| PCIE_CAP_DEVICE_PORT_TYPE[3:0] | 0x030 | [7:4] |
| PCIE_CAP_INT_MSG_NUM[4:0] | 0x030 | [12:8] |
| PCIE_CAP_NEXTPTR[7:0] | 0x031 | [7:0] |
| PCIE_CAP_ON | 0x031 | [8] |
| PCIE_CAP_RSVD_15_14[1:0] | 0x031 | [10:9] |
| PCIE_CAP_SLOT_IMPLEMENTED | 0x031 | [11] |
| PCIE_REVISION[3:0] | 0x031 | [15:12] |
| PM_BASE_PTR[7:0] | 0x032 | [7:0] |
| PM_CAP_AUXCURRENT[2:0] | 0x032 | [10:8] |
| PM_CAP_D1SUPPORT | 0x032 | [11] |
| PM_CAP_D2SUPPORT | 0x032 | [12] |
| PM_CAP_DSI | 0x032 | [13] |
| PM_CAP_ID[7:0] | 0x033 | [7:0] |
| PM_CAP_NEXTPTR[7:0] | 0x033 | [15:8] |
| PM_CAP_ON | 0x034 | [0] |
| PM_CAP_PME_CLOCK | 0x034 | [1] |
| PM_CAP_PMESUPPORT[4:0] | 0x034 | [6:2] |
| PM_CAP_RSVD_04 | 0x034 | [7] |
| PM_CAP_VERSION[2:0] | 0x034 | [10:8] |
| PM_CSR_B2B3 | 0x034 | [11] |
| PM_CSR_BPCCEN | 0x034 | [12] |
| PM_CSR_NOSOFTRST | 0x034 | [13] |
| PM_DATA_SCALE0[1:0] | 0x034 | [15:14] |
| PM_DATA_SCALE1[1:0] | 0x035 | [1:0] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| PM_DATA_SCALE2[1:0] | 0x035 | [3:2] |
| PM_DATA_SCALE3[1:0] | 0x035 | [5:4] |
| PM_DATA_SCALE4[1:0] | 0x035 | [7:6] |
| PM_DATA_SCALE5[1:0] | 0x035 | [9:8] |
| PM_DATA_SCALE6[1:0] | 0x035 | [11:10] |
| PM_DATA_SCALE7[1:0] | 0x035 | [13:12] |
| PM_DATA0[7:0] | 0x036 | [7:0] |
| PM_DATA1[7:0] | 0x036 | [15:8] |
| PM_DATA2[7:0] | 0x037 | [7:0] |
| PM_DATA3[7:0] | 0x037 | [15:8] |
| PM_DATA4[7:0] | 0x038 | [7:0] |
| PM_DATA5[7:0] | 0x038 | [15:8] |
| PM_DATA6[7:0] | 0x039 | [7:0] |
| PM_DATA7[7:0] | 0x039 | [15:8] |
| REVISION_ID[7:0] | 0x03a | [7:0] |
| ROOT_CAP_CRS_SW_VISIBILITY | 0x03a | [8] |
| SELECT_DLL_IF | 0x03a | [9] |
| SLOT_CAP_ATT_BUTTON_PRESENT | 0x03a | [10] |
| SLOT_CAP_ATT_INDICATOR_PRESENT | 0x03a | [11] |
| SLOT_CAP_ELEC_INTERLOCK_PRESENT | 0x03a | [12] |
| SLOT_CAP_HOTPLUG_CAPABLE | 0x03a | [13] |
| SLOT_CAP_HOTPLUG_SURPRISE | 0x03a | [14] |
| SLOT_CAP_MRL_SENSOR_PRESENT | 0x03a | [15] |
| SLOT_CAP_NO_CMD_COMPLETED_SUPPORT | 0x03b | [0] |
| SLOT_CAP_PHYSICAL_SLOT_NUM[12:0] | 0x03b | [13:1] |
| SLOT_CAP_POWER_CONTROLLER_PRESENT | 0x03b | [14] |
| SLOT_CAP_POWER_INDICATOR_PRESENT | 0x03b | [15] |
| SLOT_CAP_SLOT_POWER_LIMIT_SCALE[1:0] | 0x03c | [1:0] |
| SLOT_CAP_SLOT_POWER_LIMIT_VALUE[7:0] | 0x03c | [9:2] |
| SUBSYSTEM_ID[15:0] | 0x03d | [15:0] |
| SUBSYSTEM_VENDOR_ID[15:0] | 0x03e | [15:0] |
| VC_BASE_PTR[11:0] | 0x03f | [11:0] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| VC_CAP_NEXTPTR[11:0] | 0x040 | [11:0] |
| VC_CAP_ON | 0x040 | [12] |
| VC_CAP_ID[15:0] | 0x041 | [15:0] |
| VC_CAP_REJECT_SNOOP_TRANSACTIONS | 0x042 | [0] |
| VENDOR_ID[15:0] | 0x043 | [15:0] |
| VSEC_BASE_PTR[11:0] | 0x044 | [11:0] |
| VSEC_CAP_HDR_ID[15:0] | 0x045 | [15:0] |
| VSEC_CAP_HDR_LENGTH[11:0] | 0x046 | [11:0] |
| VSEC_CAP_HDR_REVISION[3:0] | 0x046 | [15:12] |
| VSEC_CAP_ID[15:0] | 0x047 | [15:0] |
| VSEC_CAP_IS_LINK_VISIBLE | 0x048 | [0] |
| VSEC_CAP_NEXTPTR[11:0] | 0x048 | [12:1] |
| VSEC_CAP_ON | 0x048 | [13] |
| VSEC_CAP_VERSION[3:0] | 0x049 | [3:0] |
| USER_CLK_FREQ[2:0] | 0x049 | [6:4] |
| CRM_MODULE_RSTS[6:0] | 0x049 | [13:7] |
| LL_ACK_TIMEOUT[14:0] | 0x04a | [14:0] |
| LL_ACK_TIMEOUT_EN | 0x04a | [15] |
| LL_ACK_TIMEOUT_FUNC[1:0] | 0x04b | [1:0] |
| LL_REPLAY_TIMEOUT[14:0] | 0x04c | [14:0] |
| LL_REPLAY_TIMEOUT_EN | 0x04c | [15] |
| LL_REPLAY_TIMEOUT_FUNC[1:0] | 0x04d | [1:0] |
| DISABLE_LANE_REVERSAL | 0x04d | [2] |
| DISABLE_SCRAMBLING | 0x04d | [3] |
| ENTER_RVRY_EI_L0 | 0x04d | [4] |
| INFER_EI[4:0] | 0x04d | [9:5] |
| LINK_CAP_MAX_LINK_WIDTH[5:0] | 0x04d | [15:10] |
| LTSSM_MAX_LINK_WIDTH[5:0] | 0x04e | [5:0] |
| N_FTS_COMCLK_GEN1[7:0] | 0x04e | [13:6] |
| N_FTS_COMCLK_GEN2[7:0] | 0x04f | [7:0] |
| N_FTS_GEN1[7:0] | 0x04f | [15:8] |
| N_FTS_GEN2[7:0] | 0x050 | [7:0] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| ALLOW_X8_GEN2 | 0x050 | [8] |
| PL_AUTO_CONFIG[2:0] | 0x050 | [11:9] |
| PL_FAST_TRAIN | 0x050 | [12] |
| UPCONFIG_CAPABLE | 0x050 | [13] |
| UPSTREAM_FACING | 0x050 | [14] |
| EXIT_LOOPBACK_ON_EI | 0x050 | [15] |
| DNSTREAM_LINK_NUM[7:0] | 0x051 | [7:0] |
| DISABLE_ASPM_L1_TIMER | 0x051 | [8] |
| DISABLE_BAR_FILTERING | 0x051 | [9] |
| DISABLE_ID_CHECK | 0x051 | [10] |
| DISABLE_RX_TC_FILTER | 0x051 | [11] |
| ENABLE_MSG_ROUTE[10:0] | 0x052 | [10:0] |
| ENABLE_RX_TD_ECRC_TRIM | 0x052 | [11] |
| TL_RX_RAM_RADDR_LATENCY | 0x052 | [12] |
| TL_RX_RAM_RDATA_LATENCY[1:0] | 0x052 | [14:13] |
| TL_RX_RAM_WRITE_LATENCY | 0x052 | [15] |
| TL_TFC_DISABLE | 0x053 | [0] |
| TL_TX_CHECKS_DISABLE | 0x053 | [1] |
| TL_RBYPASS | 0x053 | [2] |
| TL_TX_RAM_RADDR_LATENCY | 0x053 | [3] |
| TL_TX_RAM_RDATA_LATENCY[1:0] | 0x053 | [5:4] |
| TL_TX_RAM_WRITE_LATENCY | 0x053 | [6] |
| VC_CAP_VERSION[3:0] | 0x053 | [10:7] |
| VC0_CPL_INFINITE | 0x053 | [11] |
| VC0_RX_RAM_LIMIT[12:0] | 0x054 | [12:0] |
| VC0_TOTAL_CREDITS_CD[10:0] | 0x055 | [10:0] |
| VC0_TOTAL_CREDITS_CH[6:0] | 0x056 | [6:0] |
| VC0_TOTAL_CREDITS_NPH[6:0] | 0x056 | [13:7] |
| VC0_TOTAL_CREDITS_PD[10:0] | 0x057 | [10:0] |
| VC0_TOTAL_CREDITS_PH[6:0] | 0x058 | [6:0] |
| VC0_TX_LASTPACKET[4:0] | 0x058 | [11:7] |
| RECRC_CHK[1:0] | 0x058 | [13:12] |

*Table 6-41:* **DRP Address Map for PCIE_2_0 Library Element Attributes** *(Cont'd)*

| Attribute Name | Address drp_daddr[8:0] | Data Bits drp_di[15:0] or drp_do[15:0] |
|---|---|---|
| RECRC_CHK_TRIM | 0x058 | [14] |
| UR_INV_REQ | 0x058 | [15] |
| PGL0_LANE[2:0] | 0x059 | [2:0] |
| PGL1_LANE[2:0] | 0x059 | [5:3] |
| PGL2_LANE[2:0] | 0x059 | [8:6] |
| PGL3_LANE[2:0] | 0x059 | [11:9] |
| PGL4_LANE[2:0] | 0x059 | [14:12] |
| PGL5_LANE[2:0] | 0x05a | [2:0] |
| PGL6_LANE[2:0] | 0x05a | [5:3] |
| PGL7_LANE[2:0] | 0x05a | [8:6] |
| TEST_MODE_PIN_CHAR | 0x05a | [9] |

**EXILINX**

# Core Constraints

The Virtex®-6 FPGA Integrated Block for PCI Express® solution requires the specification of timing and other physical implementation constraints to meet specified performance requirements for PCI Express. These constraints are provided with the Endpoint and Root Port solutions in a User Constraints File (UCF). Pinouts and hierarchy names in the generated UCF correspond to the provided example design.

To achieve consistent implementation results, a UCF containing these original, unmodified constraints must be used when a design is run through the Xilinx tools. For additional details on the definition and use of a UCF or specific constraints, see the Xilinx® Libraries Guide and/or Development System Reference Guide.

Constraints provided with Integrated Block solution have been tested in hardware and provide consistent results. Constraints can be modified, but modifications should only be made with a thorough understanding of the effect of each constraint. Additionally, support is not provided for designs that deviate from the provided constraints.

## Contents of the User Constraints File

Although the UCF delivered with each core shares the same overall structure and sequence of information, the content of each core's UCF varies. The sections that follow define the structure and sequence of information in a generic UCF file.

### Part Selection Constraints: Device, Package, and Speed Grade

The first section of the UCF specifies the exact device for the implementation tools to target, including the specific part, package, and speed grade. In some cases, device-specific options can be included. The device in the UCF reflects the device chosen in the CORE Generator™ software project.

### User Timing Constraints

The User Timing constraints section is not populated; it is a placeholder for the designer to provide timing constraints on user-implemented logic.

### User Physical Constraints

The User Physical constraints section is not populated; it is a placeholder for the designer to provide physical constraints on user-implemented logic.

## Core Pinout and I/O Constraints

The Core Pinout and I/O constraints section contains constraints for I/Os belonging to the core's System (SYS) and PCI Express (PCI_EXP) interfaces. It includes location constraints for pins and I/O logic as well as I/O standard constraints.

## Core Physical Constraints

Physical constraints are used to limit the core to a specific area of the device and to specify locations for clock buffering and other logic instantiated by the core.

## Core Timing Constraints

This Core Timing constraints section defines clock frequency requirements for the core and specifies which nets the timing analysis tool should ignore.

# Required Modifications

Several constraints provided in the UCF utilize hierarchical paths to elements within the Integrated Block. These constraints assume an instance name of *core* for the core. If a different instance name is used, replace *core* with the actual instance name in all hierarchical constraints.

For example:

Using *xilinx_pcie_ep* as the instance name, the physical constraint

```
INST "core/pcie_2_0_i/pcie_gt_i/gtx_v6_i/GTXD[0].GTX"
LOC = GTXE1_X0Y15;
```

becomes

```
INST "xilinx_pci_ep/pcie_2_0_i/pcie_gt_i/gtx_v6_i/GTXD[0].GTX"
LOC = GTXE1_X0Y15;
```

The provided UCF includes blank sections for constraining user-implemented logic. While the constraints provided adequately constrain the Integrated Block core itself, they cannot adequately constrain user-implemented logic interfaced to the core. Additional constraints must be implemented by the designer.

# Device Selection

The device selection portion of the UCF informs the implementation tools which part, package, and speed grade to target for the design. Because Integrated Block cores are designed for specific part and package combinations, this section should not be modified by the designer.

The device selection section always contains a part selection line, but can also contain part or package-specific options. An example part selection line:

```
CONFIG PART = XC6VLX240T-FF1156-1
```

# Core I/O Assignments

This section controls the placement and options for I/Os belonging to the core's System (SYS) interface and PCI Express (PCI_EXP) interface. NET constraints in this section control the pin location and I/O options for signals in the SYS group. Locations and options vary depending on which derivative of the core is used and should not be changed without fully understanding the system requirements.

For example:

```
NET "sys_reset_n" LOC = "AE13"| IOSTANDARD = LVCMOS25| PULLUP |
NODELAY;
NET "sys_clk_p" LOC = "P5"
NET "sys_clk_n" LOC = "P6"
INST  "refclk_ibuf" LOC = IBUFDS_GTXE1_X0Y7;
```

See Clocking and Reset of the Integrated Block Core, page 183 for detailed information about reset and clock requirements.

For GTX transceiver pinout information, see the "Package Placement Information" section in the *Virtex-6 FPGA GTX Transceivers User Guide*.

INST constraints are used to control placement of signals that belong to the PCI_EXP group. These constraints control the location of the transceiver(s) used, which implicitly controls pin locations for the transmit and receive differential pair.

For example:

```
INST "core/pcie_2_0_i/pcie_gt_i/gtx_v6_i/GTXD[0].GTX"
LOC = GTXE1_X0Y15;
```

# Core Physical Constraints

Physical constraints are included in the constraints file to control the location of clocking and other elements and to limit the core to a specific area of the FPGA logic. Specific physical constraints are chosen to match each supported device and package combination—it is very important to leave these constraints unmodified.

Physical constraints example:

```
INST "core/*"  AREA_GROUP = "AG_core" ;
AREA_GROUP "AG_core"   RANGE = SLICE_X136Y147:SLICE_X155Y120 ;
```

# Core Timing Constraints

Timing constraints are provided for all integrated block solutions, although they differ based on core configuration. In all cases they are crucial and must not be modified, except to specify the top-level hierarchical name. Timing constraints are divided into two categories:

- TIG constraints. Used on paths where specific delays are unimportant, to instruct the timing analysis tools to refrain from issuing *Unconstrained Path* warnings.

- Frequency constraints. Group clock nets into time groups and assign properties and requirements to those groups.

TIG constraints example:

```
NET "sys_reset_n" TIG;
```

Clock constraints example:

First, the input reference clock period is specified, which can be 100 MHz, 125 MHz, or 250 MHz (selected in the CORE Generator software GUI).

```
NET "sys_clk_c"              TNM_NET = "SYSCLK" ;
TIMESPEC "TS_SYSCLK" = PERIOD "SYSCLK" 100.00 MHz HIGH 50 % PRIORITY
100;  # OR
```

Next, the internally generated clock net and period are specified, which can be 100 MHz, 125 MHz, or 250 MHz. (*Both* clock constraints must be specified as 100 MHz, 125 MHz, or 250 MHz.)

```
NET "core/pcie_clocking_i/clk_125" TNM_NET = "CLK_125" ;
TIMESPEC "TS_CLK_125" = PERIOD "CLK_125" TS_SYSCLK*1.25 HIGH 50 %
PRIORITY 1;
```

# Relocating the Integrated Block Core

While Xilinx does not provide technical support for designs whose system clock input, GTXE transceivers, or block RAM locations are different from the provided examples, it is possible to relocate the core within the FPGA. The locations selected in the provided examples are the recommended pinouts. These locations have been chosen based on the proximity to the PCIe block, which enables meeting 250 MHz timing, and because they are conducive to layout requirements for add-in card design. If the core is moved, the relative location of all transceivers and clocking resources should be maintained to ensure timing closure.

# Supported Core Pinouts

Table 7-1 and Table 7-2 define the supported core pinouts for the available LXT, SXT, CXT, and HXT part and package combinations. The CORE Generator software provides a UCF for the selected part and package that matches the table contents.

*Table 7-1:* **Supported Core Pinouts**

| Package | Virtex-6 FPGA | PCIe Block Location | | X1 | X2 | X4 | X8 |
|---|---|---|---|---|---|---|---|
| FF484 | LX75T | X0Y0 | Lane 0 | X0Y7 | X0Y7 | X0Y7 | X0Y7 |
| FF784 | CX75T<br>LX75T | | | | | | |
| FF1156 | CX130T<br>CX195T<br>CX240T<br>LX130T<br>LX195T<br>LX240T<br>LX365T<br>SX315T | | Lane 1 | | X0Y6 | X0Y6 | X0Y6 |
| | | | Lane 2 | | | X0Y5 | X0Y5 |
| | | | Lane 3 | | | X0Y4 | X0Y4 |
| FF1759 | LX240T<br>LX365T<br>SX315T | | Lane 4 | | | | X0Y3 |
| | | | Lane 5 | | | | X0Y2 |
| | | | Lane 6 | | | | X0Y1 |
| | | | Lane 7 | | | | X0Y0 |
| FF484 | CX75T | X0Y0 | Lane 0 | X0Y11 | X0Y11 | X0Y11 | X0Y11 |
| | | | Lane 1 | | X0Y10 | X0Y10 | X0Y10 |
| | | | Lane 2 | | | X0Y9 | X0Y9 |
| | | | Lane 3 | | | X0Y8 | X0Y8 |
| FF784 | CX75T | | Lane 4 | | | | X0Y7 |
| | | | Lane 5 | | | | X0Y6 |
| | | | Lane 6 | | | | X0Y5 |
| | | | Lane 7 | | | | X0Y4 |
| FF484 | CX130T | X0Y1 | Lane 0 | X0Y11 | X0Y11 | X0Y11 | X0Y11 |
| | | | Lane 1 | | X0Y10 | X0Y10 | X0Y10 |
| | | | Lane 2 | | | X0Y9 | X0Y9 |
| | | | Lane 3 | | | X0Y8 | X0Y8 |
| | | | Lane 4 | | | | X0Y7 |
| | | | Lane 5 | | | | X0Y6 |
| | | | Lane 6 | | | | X0Y5 |
| | | | Lane 7 | | | | X0Y4 |

*Table 7-1:* **Supported Core Pinouts** *(Cont'd)*

| Package | Virtex-6 FPGA | PCIe Block Location | | X1 | X2 | X4 | X8 |
|---------|---------------|---------------------|---|----|----|----|----|
| FF484 | LX130T | X0Y1 | Lane 0 | X0Y15 | X0Y15 | X0Y15 | X0Y15 |
| FF1156 | CX130T CX195T CX240T LX130T LX195T LX240T LX365T SX315T | X0Y1 | Lane 1 | | X0Y14 | X0Y14 | X0Y14 |
| FF1759 | LX240T LX365T SX315T | X0Y1 | Lane 2 | | | X0Y13 | X0Y13 |
| | | | Lane 3 | | | X0Y12 | X0Y12 |
| FF784 | CX130T CX195T CX240T LX130T LX195T LX240T | X0Y1 | Lane 4 | | | | X0Y11 |
| | | | Lane 5 | | | | X0Y10 |
| | | | Lane 6 | | | | X0Y9 |
| | | | Lane 7 | | | | X0Y8 |
| FF1156 | SX475T | X0Y0 | Lane 0 | X0Y15 | X0Y15 | X0Y15 | X0Y15 |
| | | | Lane 1 | | X0Y14 | X0Y14 | X0Y14 |
| FF1759 | LX550T SX475T | X0Y0 | Lane 2 | | | X0Y13 | X0Y13 |
| | | | Lane 3 | | | X0Y12 | X0Y12 |
| | | | Lane 4 | | | | X0Y11 |
| | | | Lane 5 | | | | X0Y10 |
| | | | Lane 6 | | | | X0Y9 |
| | | | Lane 7 | | | | X0Y8 |
| FF1156 | SX475T | X0Y1 | Lane 0 | X0Y23 | X0Y23 | X0Y23 | X0Y23 |
| | | | Lane 1 | | X0Y22 | X0Y22 | X0Y22 |
| | | | Lane 2 | | | X0Y21 | X0Y21 |
| | | | Lane 3 | | | X0Y20 | X0Y20 |
| FF1759 | LX550T SX475T | X0Y1 | Lane 4 | | | | X0Y19 |
| | | | Lane 5 | | | | X0Y18 |
| | | | Lane 6 | | | | X0Y17 |
| | | | Lane 7 | | | | X0Y16 |

*Table 7-2:* **Supported Core Pinouts Virtex-6 HXT FPGAs**

| Package | Virtex-6 FPGA | PCIe Block Location | | X1 | X2 | X4 | X8 |
|---------|---------------|---------------------|------|-----|-----|-----|-----|
| FF1154 | HX250T<br>HX380T | X0Y0[1][2] | Lane 0 | X1Y3 | X1Y3 | X1Y3 | X1Y7 |
| | | | Lane 1 | | X1Y2 | X1Y2 | X1Y6 |
| FF1155 | HX255T | | Lane 2 | | | X1Y1 | X1Y5 |
| | | | Lane 3 | | | X1Y0 | X1Y4 |
| FF1923 | HX255T | | Lane 4 | | | | X1Y3 |
| | | | Lane 5 | | | | X1Y2 |
| FF1924 | HX380T<br>HX565T | | Lane 6 | | | | X1Y1 |
| | | | Lane 7 | | | | X1Y0 |
| FF1154 | HX250T<br>HX380T | X0Y1 | Lane 0 | X1Y7 | X1Y7 | X1Y7 | X1Y7 |
| | | | Lane 1 | | X1Y6 | X1Y6 | X1Y6 |
| | | | Lane 2 | | | X1Y5 | X1Y5 |
| | | | Lane 3 | | | X1Y4 | X1Y4 |
| FF1924 | HX380T<br>HX565T | | Lane 4 | | | | X1Y3 |
| | | | Lane 5 | | | | X1Y2 |
| | | | Lane 6 | | | | X1Y1 |
| | | | Lane 7 | | | | X1Y0 |
| FF1155 | HX255T | X0Y1 | Lane 0 | X1Y11 | X1Y11 | X1Y11 | X1Y11 |
| | | | Lane 1 | | X1Y10 | X1Y10 | X1Y10 |
| | | | Lane 2 | | | X1Y9 | X1Y9 |
| | | | Lane 3 | | | X1Y8 | X1Y8 |
| FF1923 | HX255T<br>HX380T[3]<br>HX565T | | Lane 4 | | | | X1Y7 |
| | | | Lane 5 | | | | X1Y6 |
| | | | Lane 6 | | | | X1Y5 |
| | | | Lane 7 | | | | X1Y4 |

*Table 7-2:* **Supported Core Pinouts Virtex-6 HXT FPGAs** *(Cont'd)*

| Package | Virtex-6 FPGA | PCIe Block Location | | X1 | X2 | X4 | X8 |
|---------|---------------|---------------------|---|-----|-----|-----|-----|
| FF1154 | HX250T HX380T | X0Y2 | Lane 0 | X1Y15 | X1Y15 | X1Y15 | X1Y15 |
| | | | Lane 1 | | X1Y14 | X1Y14 | X1Y14 |
| FF1923 | HX380T HX565T | | Lane 2 | | | X1Y13 | X1Y13 |
| | | | Lane 3 | | | X1Y12 | X1Y12 |
| | | | Lane 4 | | | | X1Y11 |
| FF1924 | HX380T HX565T | | Lane 5 | | | | X1Y10 |
| | | | Lane 6 | | | | X1Y9 |
| | | | Lane 7 | | | | X1Y8 |
| FF1155 | HX380T | | Lane 0 | X1Y19 | X1Y19 | X1Y19 | X1Y19 |
| | | | Lane 1 | | X1Y18 | X1Y18 | X1Y18 |
| | | | Lane 2 | | | X1Y17 | X1Y17 |
| | | | Lane 3 | | | X1Y16 | X1Y16 |
| | | | Lane 4 | | | | X1Y15 |
| | | | Lane 5 | | | | X1Y14 |
| | | | Lane 6 | | | | X1Y13 |
| | | | Lane 7 | | | | X1Y12 |
| FF1154 | HX250T HX380T | X0Y3 | Lane 0 | X1Y23 | X1Y23 | X1Y23 | X1Y23 |
| | | | Lane 1 | | X1Y22 | X1Y22 | X1Y22 |
| FF1155 | HX380T | | Lane 2 | | | X1Y21 | X1Y21 |
| | | | Lane 3 | | | X1Y20 | X1Y20 |
| FF1923 | HX380T HX565T | | Lane 4 | | | | X1Y19 |
| | | | Lane 5 | | | | X1Y18 |
| FF1924 | HX380T HX565T | | Lane 6 | | | | X1Y17 |
| | | | Lane 7 | | | | X1Y16 |

**Notes:**

1. 8-lane Endpoint configuration at 5.0 Gb/s with 512 byte MPS is not supported on this PCIe block location.

2. High performance level for MPS settings 512 bytes and 1024 bytes is not supported for this PCIe Block location on this device.

3. 8-lane configuration is not supported for this PCIe block location on this device.

# *FPGA Configuration*

This chapter discusses how to configure the Virtex®-6 FPGA so that the device can link up and be recognized by the system. This information is provided for the user to choose the correct FPGA configuration method for the system and verify that it will work as expected.

This chapter discusses how specific requirements of the *PCI Express Base Specification* and *PCI Express Card Electromechanical Specification* apply to FPGA configuration. Where appropriate, Xilinx recommends that the user read the actual specifications for detailed information. This chapter is divided into four sections:

- Configuration Terminology: Defines terms used in this chapter.
- Configuration Access Time. Several specification items govern when an Endpoint device needs to be ready to receive configuration accesses from the host (Root Complex).
- Board Power in Real-World Systems. Understanding real-world system constraints related to board power and how they affect the specification requirements.
- Recommendations. Describes methods for FPGA configuration and includes sample problem analysis for FPGA configuration timing issues.

## Configuration Terminology

In this chapter, these terms are used to differentiate between FPGA configuration and configuration of the PCI Express device:

- Configuration of the FPGA. *FPGA configuration* is used.
- Configuration of the PCI Express device. After the link is active, *configuration* is used.

## Configuration Access Time

In standard systems for PCI Express, when the system is powered up, configuration software running on the processor starts scanning the PCI Express bus to discover the machine topology.

The process of scanning the PCI Express hierarchy to determine its topology is referred to as the *enumeration process*. The root complex accomplishes this by initiating configuration transactions to devices as it traverses and determines the topology.

All PCI Express devices are expected to have established the link with their link partner and be ready to accept configuration requests during the enumeration process. As a result, there are requirements as to when a device needs to be ready to accept configuration requests after power up; if the requirements are not met, the following occurs:

- If a device is not ready and does not respond to configuration requests, the root complex does not discover it and treats it as non-existent.

- The operating system will not report the device's existence and the user's application will not be able to communicate with the device.

Choosing the appropriate FPGA configuration method is key to ensuring the device is able to communicate with the system in time to achieve link up and respond to the configuration accesses.

## Configuration Access Specification Requirements

Two PCI Express specification items are relevant to configuration access:

1. Section 6.6 of *PCI Express Base Specification*, rev 1.1 states "A system must guarantee that all components intended to be software visible at boot time are ready to receive Configuration Requests within 100 ms of the end of Fundamental Reset at the Root Complex." For detailed information about how this is accomplished, see the specification; it is beyond the scope of this discussion.

Xilinx compliance to this specification is validated by the PCI Express-CV tests. The PCI Special Interest Group (PCI-SIG) provides the PCI Express Configuration Test Software to verify the device meets the requirement of being able to receive configuration accesses within 100 ms of the end of the fundamental reset. The software, available to any member of the PCI-SIG, generates several resets using the in-band reset mechanism and PERST# toggling to validate robustness and compliance to the specification.

2. Section 6.6 of *PCI Express Base Specification v1.1* defines three parameters necessary "where power and PERST# are supplied." The parameter $T_{PVPERL}$ applies to FPGA configuration timing and is defined as:

   $T_{PVPERL}$ - PERST# must remain active at least this long after power becomes valid.

The *PCI Express Base Specification* does not give a specific value for $T_{PVPERL}$ – only its meaning is defined. The most common form factor used by designers with the Integrated Block core is an ATX-based form factor. The *PCI Express Card Electromechanical Specification* focuses on requirements for ATX-based form factors. This applies to most designs targeted to standard desktop or server type motherboards. Figure 8-1 shows the relationship between Power Stable and PERST#.
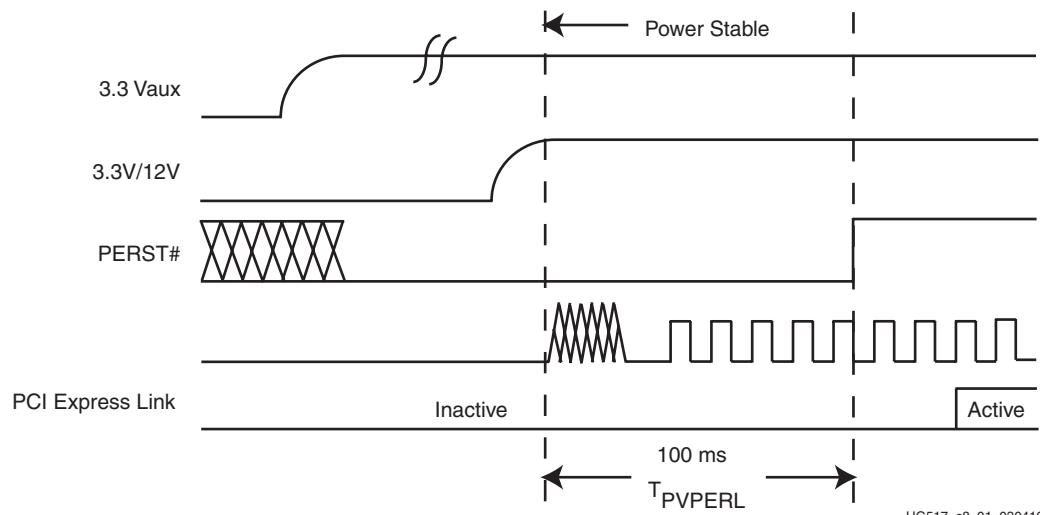


*Figure 8-1:* **Power Up**

Section 2.6.2 of the *PCI Express Card Electromechanical Specification, v1.1* defines $T_{PVPREL}$ as a minimum of 100 ms, indicating that from the time power is stable the system reset is asserted for at least 100 ms (as shown in Table 8-1).

*Table 8-1:* **$T_{PVPERL}$ Specification**

| Symbol | Parameter | Min | Max | Units |
|--------|-----------|-----|-----|-------|
| $T_{PVPERL}$ | Power stable to PERST# inactive | 100 | | ms |

From Figure 8-1 and Table 8-1, it is possible to obtain a simple equation to define the FPGA configuration time as follows:

$$\text{FPGA Configuration Time} \leq T_{PWRVLD} + T_{PVPERL} \qquad \textit{Equation 8-1}$$

Given that $T_{PVPERL}$ is defined as 100 ms minimum, this becomes:

$$\text{FPGA Configuration Time} \leq T_{PWRVLD} + 100 \text{ ms} \qquad \textit{Equation 8-2}$$

*Note:* Although $T_{PWRVLD}$ is included in Equation 8-2, it has yet to be defined in this discussion because it depends on the type of system in use. The Board Power in Real-World Systems section defines $T_{PWRVLD}$ for both ATX-based and non ATX-based systems.

FPGA configuration time is only relevant at cold boot; subsequent warm or hot resets do not cause reconfiguration of the FPGA. If the design appears to be having problems due to FPGA configuration, the user should issue a warm reset as a simple test, which resets the system, including the PCI Express link, but keeps the board powered. If the problem does not appear, the issue could be FPGA configuration time related.

# Board Power in Real-World Systems

Several boards are used in PCI Express systems. The *ATX Power Supply Design* specification, endorsed by Intel, is used as a guideline and for this reason followed in the majority of mother boards and 100% of the time if it is an Intel-based motherboard. The relationship between power rails and power valid signaling is described in the ATX 12V Power Supply Design Guide. Figure 8-2, redrawn here and simplified to show the information relevant to FPGA configuration, is based on the information and diagram found in section 3.3 of the *ATX 12V Power Supply Design Guide*. For the entire diagram and definition of all parameters, see the *ATX 12V Power Supply Design Guide*.

Figure 8-2 shows that power stable indication from Figure 8-1 for the PCI Express system is indicated by the assertion of PWR_OK. PWR_OK is asserted High after some delay when the power supply has reached 95% of nominal.



T1 = Power On Time (T1 < 500 ms)
T2 = Risetime (0.1 ms <= T2 <= 20 ms)
T3 = PWR_OK Delay (100 ms < T3 < 500 ms)
T4 = PWR_OK risetime (T4 <= 10 ms)
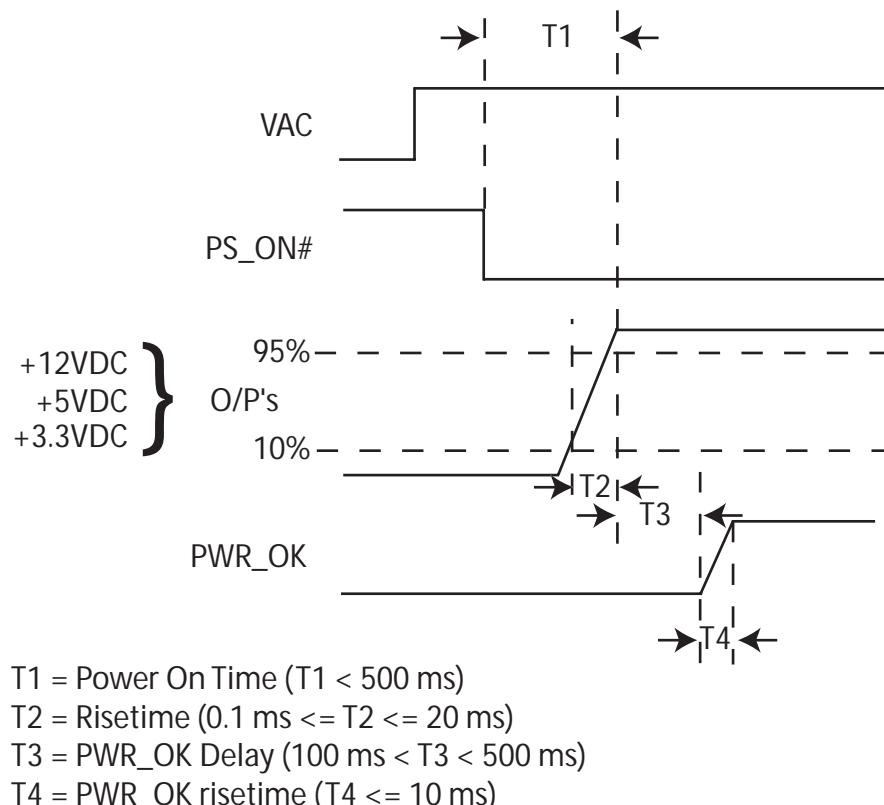
*Figure 8-2:* **ATX Power Supply**

Figure 8-2 shows that power is actually valid before PWR_OK is asserted High. This is represented by T3 and is the PWR_OK delay. The *ATX 12V Power Supply Design Guide* defines PWR_OK as 100 ms < T3 < 500 ms, indicating the following: From the point at which the power level reaches 95% of nominal, there is a minimum of at least 100 ms but no more than 500 ms of delay before PWR_OK is asserted. Remember, according to the *PCI Express Card Electromechanical Specification*, the PERST# is guaranteed to be asserted a minimum of 100 ms from when power is stable indicated in an ATX system by the assertion of PWR_OK.

Again, the FPGA configuration time equation is:

$$\text{FPGA Configuration Time} \leq T_{PWRVLD} + 100 \text{ ms} \qquad \textit{Equation 8-3}$$

$T_{PWRVLD}$ is defined as PWR_OK delay period; that is, $T_{PWRVLD}$ represents the amount of time that power is valid in the system before PWR_OK is asserted. This time can be added to the amount of time the FPGA has to configure. The minimum values of T2 and T4 are negligible and considered zero for purposes of these calculations. For ATX-based motherboards, which represent the majority of real-world motherboards in use, $T_{PWRVLD}$ can be defined as:

$$100 \text{ ms} \leq T_{PWRVLD} \leq 500 \text{ ms} \qquad \textit{Equation 8-4}$$

This provides the following requirement for FPGA configuration time in both ATX and non-ATX-based motherboards:

- FPGA Configuration Time $\leq$ 200 ms (for ATX based motherboard)
- FPGA Configuration Time $\leq$ 100 ms (for non-ATX based motherboard)

The second equation for the non-ATX based motherboards assumes a $T_{PWRVLD}$ value of 0 ms because it is not defined in this context. Designers with non-ATX based motherboards should evaluate their own power supply design to obtain a value for $T_{PWRVLD}$.

This chapter assumes that the FPGA power ($V_{CCINT}$) is stable before or at the same time that PWR_OK is asserted. If this is not the case, then additional time must be subtracted from the available time for FPGA configuration. Xilinx recommends to avoid designing add-in cards with staggered voltage regulators with long delays.

## Hot Plug Systems

Hot Plug systems generally employ the use of a Hot-Plug Power Controller located on the system motherboard. Many discrete Hot-Plug Power Controllers extend $T_{PVPERL}$ beyond the minimum 100 ms. Add-in card designers should consult the Hot-Plug Power Controller data sheet to determine the value of $T_{PVPERL}$. If the Hot-Plug Power Controller is unknown, then a $T_{PVPERL}$ value of 100 ms should be assumed.

# Recommendations

Xilinx recommends using the [Platform Flash XL High-Density Storage and Configuration Device](#) (XCF128X) in Slave Map x16 Mode with a CCLK frequency of 50 MHz, which allows time for FPGA configuration on any Virtex-6 FPGA in ATX-based motherboards. Other valid configuration options are represented by green cells in Table 8-2 and Table 8-3 depending on the type of system in use. This section discusses these recommendations and includes sample analysis of potential problems that might arise during FPGA configuration.

## FPGA Configuration Times for Virtex-6 Devices

During power up, the FPGA configuration sequence is performed in four steps:

1. Wait for Power on Reset (POR) for all voltages ($V_{CCINT}$, $V_{CCAUX}$, and $V_{CCO}$) in the FPGA to trip, referred to as POR Trip Time.

2. Wait for completion (deassertion) of INIT to allow the FPGA to initialize before accepting a bitstream transfer.

   **Note:** As a general rule, steps 1 and 2 require $\leq$ 50 ms

3. Wait for assertion of DONE, the actual time required for a bitstream to transfer, and depends on:

- Bitstream size
- Clock frequency
- Transfer mode used in the Flash Device
    - SPI = Serial Peripheral Interface
    - BPI = Byte Peripheral Interface
    - PFP = Platform Flash PROMs

For detailed information about the configuration process, see the *Virtex-6 FPGA Configuration User Guide*.

Table 8-2 and Table 8-3 show the comparative data for all Virtex-6 FPGA LXT, and SXT devices with respect to a variety of flash devices and programming modes. The default clock rate for configuring the device is always 2 MHz. Any reference to a different clock rate implies a change in the settings of the device being used to program the FPGA. The configuration clock (CCLK), when driven by the FPGA, has variation and is not exact. See the *Virtex-6 FPGA Configuration Guide* for more information on CCLK tolerances.

## Configuration Time Matrix: ATX Motherboards

Table 8-2 shows the configuration methods that allow the device to be configured before the end of the fundamental reset in ATX-based systems. The table values represent the bitstream transfer time only. The matrix is color-coded to show which configuration methods allow the device to configure within 200 ms once the FPGA initialization time is included. A configuration method shaded in green is chosen when using ATX-based systems to ensure that the device is recognized.

*Table 8-2:* **Configuration Time Matrix (ATX Motherboards): Virtex-6 FPGA Bitstream Transfer Time (ms)**

| Virtex-6 FPGA | Bitstream (Bits) | SPIx1[1] | BPIx16[2] (Page mode) | PFPx8[3] | XCF128X (Master-BPIx16) | XCF128X[4] (Slave-SMAPx16) |
|---|---|---|---|---|---|---|
| XC6VLX75T | 26,239,328 | 525 | 240 | 100 | 137 | 33 |
| XC6VLX130T | 43,719,776 | 875 | 399 | 166 | 228 | 55 |
| XC6VLX195T | 61,552,736 | 1232 | 562 | 234 | 321 | 77 |
| XC6VLX240T | 73,859,552 | 1478 | 674 | 280 | 385 | 93 |
| XC6VLX365T | 96,067,808 | 1922 | 876 | 364 | 501 | 121 |
| XC6VSX315T | 104,465,888 | 2090 | 953 | 396 | 545 | 131 |
| XC6VLX550T | 144,092,384 | 2882 | 1314 | 546 | 751 | [5] |
| XC6VSX475T | 156,689,504 | 3134 | 1429 | 594 | 817 | [5] |

GREEN: Bitstream Transfer Time + FPGA INIT Time (50 ms) ≤ 200 ms.

YELLOW: Bitstream Transfer Time + FPGA INIT Time (50 ms) > 200 ms

RED: Bitstream Transfer Time > 200 ms

**Notes:**

1. SPI flash assumptions: 50 MHz maximum.
2. BPIx16 assumptions: P30 4-page read with 4-cycle first page read (4-1-1-1), maximum configuration time.
3. PFP assumptions: 33 MHz maximum.
4. XCF128X Slave-SMAPx16 assumptions: CCLK=50 MHz.
5. The XC6VLX550T and XC6VSX475T devices will not fit into a single XCF128X Platform Flash - a Dual Flash solution is required and is currently under development.

## Configuration Time Matrix: Non-ATX-Based Motherboards

Table 8-3 shows the configuration methods that allow the device to be configured before the end of the fundamental reset in non-ATX-based systems. This assumes $T_{PWRVLD}$ is zero. The table values represent the bitstream transfer time only. The matrix is color-coded to show which configuration methods allow the device to configure within 100 ms once the FPGA initialization time is included. A configuration method shaded in green is chosen when using non-ATX-based systems to ensure that the device is recognized.

For some of the larger FPGAs, it might not be possible to configure within the 100 ms window. In these cases, the user system needs to be evaluated to see if any margin is available that can be assigned to $T_{PWRVLD}$.

*Table 8-3:* **Configuration Time Matrix (Generic Platforms: Non-ATX Motherboards): Virtex-6 FPGA Bitstream Transfer Time (ms)**

| Virtex-6 FPGA | Bitstream (Bits) | SPIx1[1] | BPIx16[2] (Page mode) | PFPx8[3] | XCF128X (Master-BPIx16) | XCF128X[4] (Slave-SMAPx16) |
|---|---|---|---|---|---|---|
| XC6VLX75T | 26,239,328 | 525 | 240 | 100 | 137 | 33 |
| XC6VLX130T | 43,719,776 | 875 | 399 | 166 | 228 | 55 |
| XC6VLX195T | 61,552,736 | 1232 | 562 | 234 | 321 | 77 |
| XC6VLX240T | 73,859,552 | 1478 | 674 | 280 | 385 | 93 |
| XC6VLX365T | 96,067,808 | 1922 | 876 | 364 | 501 | 121 |
| XC6VSX315T | 104,465,888 | 2090 | 953 | 396 | 545 | 131 |
| XC6VLX550T | 144,092,384 | 2882 | 1314 | 546 | 751 | [5] |
| XC6VSX475T | 156,689,504 | 3134 | 1429 | 594 | 817 | [5] |

GREEN: Bitstream Transfer Time + FPGA INIT Time (50 ms) ≤ 100 ms.

YELLOW: Bitstream Transfer Time + FPGA INIT Time (50 ms) > 100 ms

RED: Bitstream Transfer Time > 100 ms

**Notes:**
1. SPI flash assumptions: 50 MHz maximum.
2. BPIx16 assumptions: P30 4-page read with 4-cycle first page read (4-1-1-1), maximum configuration time.
3. PFP assumptions: 33 MHz maximum.
4. XCF128X Slave-SMAPx16 assumptions: CCLK=50 MHz.
5. The XC6VLX550T and XC6VSX475T devices will not fit into a single XCF128X Platform Flash. A Dual Flash solution will be required and is currently under development.

## Sample Problem Analysis

This section presents data from an ASUS PL5 system to demonstrate the relationships between Power Valid, FPGA Configuration, and PERST#. Figure 8-3 shows a case where the endpoint failed to be recognized due to a FPGA configuration time issue. Figure 8-4 shows a successful FPGA configuration with the endpoint being recognized by the system.

### Failed FPGA Recognition

Figure 8-3 illustrates a failed cold boot test using the default configuration time on an LX50T FPGA. In this example, the host failed to recognize the Xilinx FPGA. Although a second PERST# pulse assists in allowing more time for the FPGA to configure, the slowness of the FPGA configuration clock (2 MHz) causes configuration to complete well after this second deassertion. During this time, the system enumerated the bus and did not recognize the FPGA.
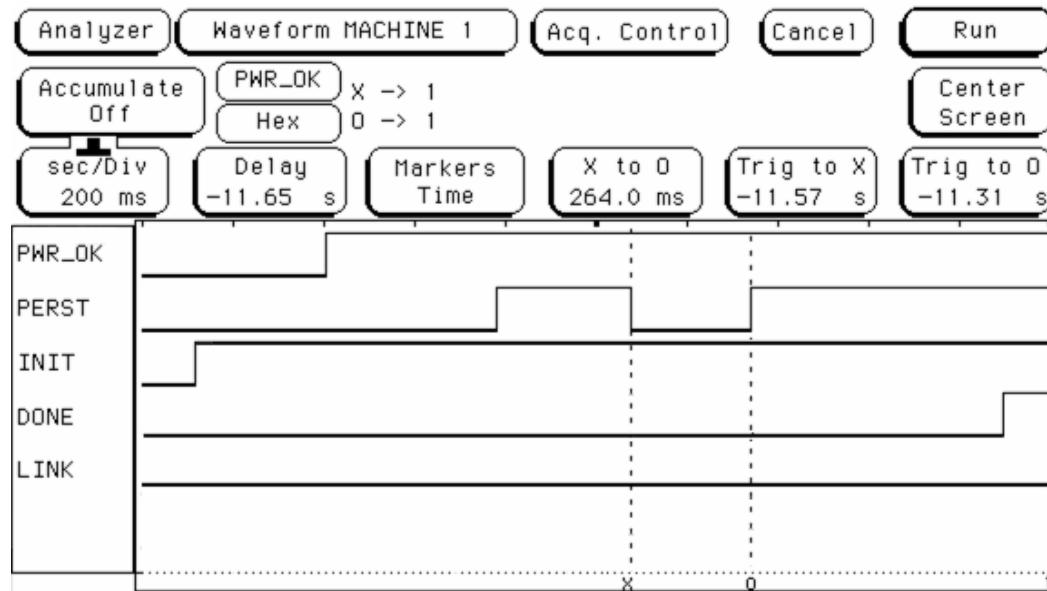


*Figure 8-3:* **Default Configuration Time on LX50T Device (2 MHz Clock)**

## Successful FPGA Recognition

Figure 8-4 illustrates a successful cold boot test on the same system. In this test, the CCLK was running at 50 MHz, allowing the FPGA to configure in time to be enumerated and recognized. The figure shows that the FPGA began initialization approximately 250 ms before PWR_OK. DONE going High shows that the FPGA was configured even before PWR_OK was asserted.
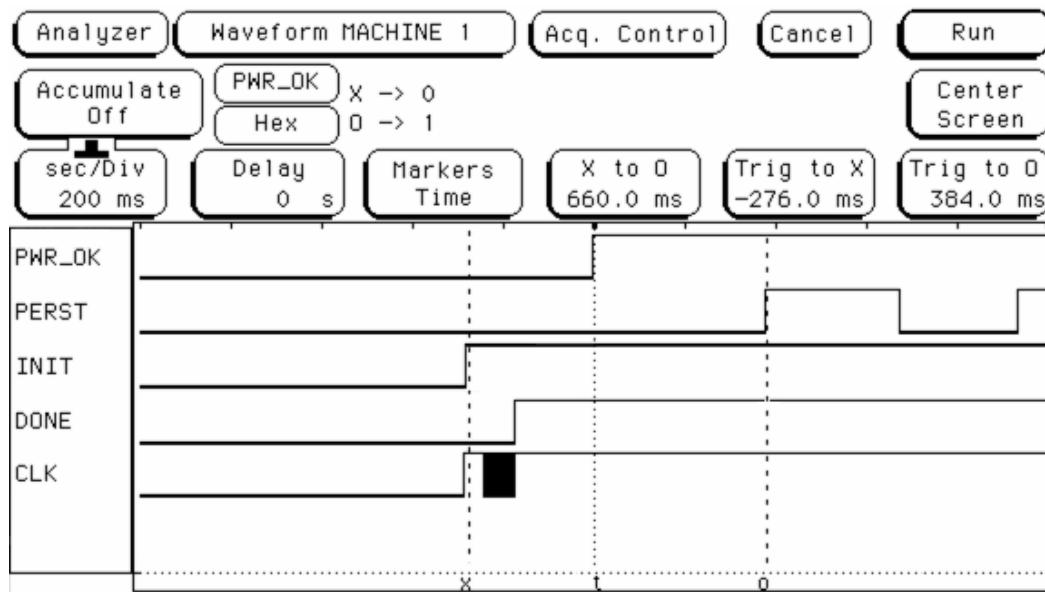


*Figure 8-4:* **Fast Configuration Time on LX50T Device (50 MHz Clock)**

## Workarounds for Closed Systems

For failing FPGA configuration combinations, as represented by pink cells and yellow cells in Table 8-2 and Table 8-3, designers might be able to work around the problem in closed systems or systems where they can guarantee behavior. These options are not recommended for products where the targeted end system is unknown.

1. Check if the motherboard and BIOS generate multiple PERST# pulses at startup. This can be determined by capturing the signal on the board using an oscilloscope. (This is similar to what is shown in Figure 8-3. If multiple PERST#s are generated, this typically adds extra time for FPGA configuration.

   Define $T_{PERSTPERIOD}$ as the total sum of the pulse width of PERST# and deassertion period before the next PERST# pulse arrives. Because the FPGA is not power cycled or reconfigured with additional PERST# assertions, the $T_{PERSTPERIOD}$ number can be added to the FPGA configuration equation.

   $$\text{FPGA Configuration Time} \leq T_{PWRVLD} + T_{PERSTPERIOD} + 100 \text{ ms} \qquad \textit{Equation 8-5}$$

2. In closed systems, it might be possible to create scripts to force the system to perform a warm reset after the FPGA is configured, after the initial power up sequence. This resets the system along with the PCI Express subsystem allowing the device to be recognized by the system.

# *Known Restrictions*

This chapter describes several restrictions or issues where the integrated block deviates from the PCI Base Specification, v2.0 or in cases where the specification is ambiguous. All issues listed in this chapter are considered low impact and are not a concern for most applications. The Comments sections describe where the associated problem might occur so that designers can decide quickly if further investigation is needed.

## Bit Errors in the Data Rate Identifier

Bit errors in the Data Rate Identifier of a TS2 ordered set might lead to a link-down condition.

### Area of Impact

Link Training

### Detailed Description

During a speed change, a bit error in symbol 4 of a TS2 ordered set (the Data Rate Identifier) can cause the integrated block to erroneously move to LTSSM state Detect, in turn causing a link-down condition. In addition to the bit error occurring during symbol 4, the error must also occur:

• During any of the last eight TS2 ordered sets, and

• Before the link partner transitions to electrical idle from the LTSSM state Recovery.Speed

### Comments

The probability of this error occurring is extremely small.

Bit errors must occur on the link for this condition to appear. Bit errors on the link are typically an indication of poor signal integrity or other severe disturbances on the link, and which might lead to other issues such as poor data throughput. Additionally, the bit error must occur at a very precise moment in time, as discussed above, in order for a Link-down condition to occur.

This issue was discovered during targeted error-injection testing of the integrated block. It has not been seen in any interoperability testing as of the publication of this document.

This issue does not affect designs that operate at 2.5 Gb/s data rates only.

There are no known workarounds for this issue. Designers should maintain good signal integrity and BER on the link to avoid this issue.

# Bit Errors in the Rate Change Bit

Bit errors in the Rate Change Bit of a TS2 ordered set can lead to unsuccessful speed negotiation.

## Area of Impact

Link Training

## Detailed Description

During link training, a bit error on bit 2 of symbol 4 of a TS2 ordered set (Data Rate Supported) can cause an unsuccessful speed negotiation. In addition to the bit error occurring during the specific symbol and bit, this bit error must also occur during the last received TS2 in the LTSSM state Recovery.RcvrCfg before transitioning to Recovery.Speed.

If this error occurs, the speed change from 2.5 Gb/s to 5.0 Gb/s line rate fails, and the link remains at the 2.5 Gb/s rate.

## Comments

The probability of this error occurring is extremely small.

Bit errors must occur on the link for this condition to appear. Bit errors on the link are typically an indication of poor signal integrity or other severe disturbances on the link, and might lead to other issues such as poor data throughput. Additionally, the bit error must occur at a very specific moment in time, as discussed in the Detailed Description, for an unsuccessful speed negotiation to occur.

This issue was discovered during targeted error-injection testing of the integrated block. This issue has not been seen in any interoperability testing as of the publication of this document. This issue does not affect designs that operate at 2.5 Gb/s data rates only.

Users can attempt a directed speed change to work around this issue.

# Link Bandwidth Management Status Bit Error

The Link Bandwidth Management Status bit is not set.

## Area of Impact

Root Port Configuration Space

## Detailed Description

The integrated block does not always set Link Status[14] (the Link Bandwidth Management Status bit) in the PCI Configuration space when it should. Specifically, it does not set this bit if hardware has changed the link speed or width to attempt to correct unreliable link operation.

## Comments

This issue only affects Root Port configurations.

Currently there is no pre-engineered workaround for this issue. Contact Xilinx Technical Support for options if Link Bandwidth Management is necessary for the design.

# De-emphasis Value Error

The preferred de-emphasis value captured from the link partner is overwritten.

## Area of Impact

Physical Layer; Root Port only

## Detailed Description

When configured as a Root Port, the integrated block misinterprets bit 6 of symbol 4 of received TS2s from the Downstream component when in the LTSSM Recovery state. During Recovery, this bit is meant to indicate a directed versus autonomous speed or width change. However, the integrated block interprets this bit as the Downstream component's preferred de-emphasis level.

## Comments

Generally, preferred de-emphasis levels are a function of the channel and are known by the designer of the link ahead of time. When this is the case, the de-emphasis level can be statically set via the PLDOWNSTREAMDEMPHSOURCE attribute to 1b as a simple workaround. This forces the Root Port transmitter to use the de-emphasis value from Link Control 2 bit 12, and the de-emphasis value from the downstream component is ignored.

Currently, there is no pre-engineered workaround for this issue when the de-emphasis level is not known ahead of time. Contact Xilinx Technical Support for options if programmable de-emphasis is necessary for the design.

# Erroneous Compliance Pattern

A modified compliance pattern was sent erroneously instead of a regular compliance pattern.

## Area of Impact

Simulation of compliance mode (LTSSM state Polling.Compliance)

## Detailed Description

If the integrated block LTSSM is in the Polling.Active state prior to entering the Polling.Compliance state, and the PIPERXnELECIDLE signal on any receiver lane is toggled during this time (indicating a change between electrical idle and active states on the receiver lanes), the integrated block enters the Polling.Compliance state and sends the modified compliance pattern even if the Enter Modified Compliance bit in the Link Control 2 register is 0.

## Comments

The *PCI Express Base Specification 2.0* does not specify any requirements for the electrical state of the receiver lanes while in the Polling.Active state.

This issue does not affect PCI-SIG compliance testing. The PCI-SIG compliance test fixtures (CLB2.0 and CBB2.0) terminate all receivers and therefore statically drive electrical idle on the receivers for the duration of the compliance test.

This issue can occur in simulation when using commercially available testbenches that enter into the Polling.Compliance state. Some testbenches might switch between electrical idle and active states on the receivers while in the Polling.Active state.

# Writable MSI Mask Register Reserved Bits

The reserved bits in the Message Signaled Interrupts (MSI) Mask register are writable.

## Area of Impact

Configuration Space

## Detailed Description

The *PCI Express Base Specification 2.0* defines reserved configuration space registers and bits as read-only. However, all 32 bits of the MSI Mask register in the MSI Capability Structure of the integrated block are read/write regardless if they are considered reserved.

For example, the MSI Message Control register bits [6:4] are `000b` indicating only one MSI vector is enabled. This indicates that only bit 0 in the MSI Mask register is read/write; bits [31:1] should be read-only.

## Comments

This issue only affects Endpoint configurations that use MSI.

There are no hardware-related side effects to writing the reserved bits in the MSI Mask register.

System software effects are system dependent; however, it is unlikely that software will react to these bits changing value. Users have the option to disable the MSI Mask/Pending extension.

# Hardware Verification

## PCI Special Interest Group

Xilinx attends the PCI Special Interest Group (PCI-SIG®) Compliance Workshops to verify the Integrated Block compliance and interoperability with various systems available on the market and those that are not yet released. While Xilinx cannot list the actual systems tested at the PCI-SIG Compliance Workshops due to requirements of the PCI-SIG by-laws, Xilinx IP designers can view the PCI-SIG integrators list.

The integrators list confirms that Xilinx satisfies the PCI-SIG Compliance Program and that the Integrated Block wrapper successfully interoperates with other available systems at the PCI-SIG Compliance Workshop. Virtex®-6 FPGA entries can be found in the Components and Add-In Cards sections under the company name Xilinx.

Hardware validation testing is performed at Xilinx for each release of the Integrated Block core with the chipsets listed below. Hardware testing performed by Xilinx that is also available to customers includes the PCIECV tool available from the PCI-SIG website, BMD Design, available from XAPP1052, *Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions,* and the MET design, available from XAPP1022, *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores.*

The hardware validation chipsets are:

- Intel x58
- Intel 5400
- Intel P55
- AMD 790
- AMD 780
- Intel Atom
- Intel x38

# *Example Design and Model Test Bench for Endpoint Configuration*

## Programmed Input/Output: Endpoint Example Design

Programmed Input/Output (PIO) transactions are generally used by a PCI Express® system host CPU to access Memory Mapped Input/Output (MMIO) and Configuration Mapped Input/Output (CMIO) locations in the PCI Express fabric. Endpoints for PCI Express accept Memory and I/O Write transactions and respond to Memory and I/O Read transactions with Completion with Data transactions.

The PIO example design (PIO design) is included with the Virtex®-6 FPGA Integrated Block for PCI Express in Endpoint configuration generated by the CORE Generator™ software, which allows users to easily bring up their system board with a known established working design to verify the link and functionality of the board.

**Note:** The PIO design Port Model is shared by the Virtex-6 FPGA Integrated Block Endpoint for PCI Express, Endpoint Block Plus for PCI Express, and Endpoint PIPE for PCI Express solutions. This appendix represents all the solutions generically using the name Endpoint for PCI Express (or Endpoint for PCIe®).

### System Overview

The PIO design is a simple target-only application that interfaces with the Endpoint for PCIe core's Transaction (TRN) interface and is provided as a starting point for customers to build their own designs. The following features are included:

- Four transaction-specific 2 KB target regions using the internal Xilinx® FPGA block RAMs, providing a total target space of 8192 bytes

- Supports single DWORD payload Read and Write PCI Express transactions to 32-/64-bit address memory spaces and I/O space with support for completion TLPs

- Utilizes the core's trn_rbar_hit_n[6:0] signals to differentiate between TLP destination Base Address Registers

- Provides separate implementations optimized for 32-bit, 64-bit, and 128-bit TRN interfaces

Figure A-1 illustrates the PCI Express system architecture components, consisting of a Root Complex, a PCI Express switch device, and an Endpoint for PCIe. PIO operations move data *downstream* from the Root Complex (CPU register) to the Endpoint, and/or *upstream* from the Endpoint to the Root Complex (CPU register). In either case, the PCI Express protocol request to move the data is initiated by the host CPU.
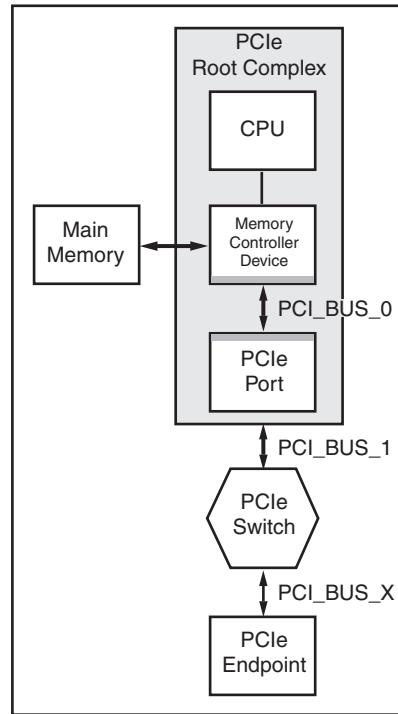
*Figure A-1:* **System Overview**

Data is moved downstream when the CPU issues a store register to a MMIO address command. The Root Complex typically generates a Memory Write TLP with the appropriate MMIO location address, byte enables, and the register contents. The transaction terminates when the Endpoint receives the Memory Write TLP and updates the corresponding local register.

Data is moved upstream when the CPU issues a load register from a MMIO address command. The Root Complex typically generates a Memory Read TLP with the appropriate MMIO location address and byte enables. The Endpoint generates a Completion with Data TLP once it receives the Memory Read TLP. The Completion is steered to the Root Complex and payload is loaded into the target register, completing the transaction.

## PIO Hardware

The PIO design implements a 8192 byte target space in FPGA block RAM, behind the Endpoint for PCIe. This 32-bit target space is accessible through single DWORD I/O Read, I/O Write, Memory Read 64, Memory Write 64, Memory Read 32, and Memory Write 32 TLPs.

The PIO design generates a completion with 1 DWORD of payload in response to a valid Memory Read 32 TLP, Memory Read 64 TLP, or I/O Read TLP request presented to it by the core. In addition, the PIO design returns a completion without data with successful status for I/O Write TLP request.

The PIO design processes a Memory or I/O Write TLP with 1 DWORD payload by updating the payload into the target address in the FPGA block RAM space.

## Base Address Register Support

The PIO design supports four discrete target spaces, each consisting of a 2 KB block of memory represented by a separate Base Address Register (BAR). Using the default parameters, the CORE Generator software produces a core configured to work with the PIO design defined in this section, consisting of:

- One 64-bit addressable Memory Space BAR
- One 32-bit Addressable Memory Space BAR

Users can change the default parameters used by the PIO design; however, in some cases they might need to change the back-end User Application depending on their system. See Changing CORE Generator Software Default BAR Settings for information about changing the default CORE Generator software parameters and the effect on the PIO design.

Each of the four 2 KB address spaces represented by the BARs corresponds to one of four 2 KB address regions in the PIO design. Each 2 KB region is implemented using a 2 KB dual-port block RAM. As transactions are received by the core, the core decodes the address and determines which of the four regions is being targeted. The core presents the TLP to the PIO design and asserts the appropriate bits of trn_rbar_hit_n[6:0], as defined in Table A-1.

*Table A-1:*   **TLP Traffic Types**

| Block RAM | TLP Transaction Type | Default BAR | trn_rbar_hit_n[6:0] |
|-----------|---------------------|-------------|---------------------|
| ep_mem0 | I/O TLP transactions | Disabled | Disabled |
| ep_mem1 | 32-bit address Memory TLP transactions | 2 | `111_1011b` |
| ep_mem2 | 64-bit address Memory TLP transactions | 0-1 | `111_1100b` |
| ep_mem3 | 32-bit address Memory TLP transactions destined for EROM | Expansion ROM | `011_1111b` |

### Changing CORE Generator Software Default BAR Settings

Users can change the CORE Generator software parameters and continue to use the PIO design to create customized Verilog or VHDL source to match the selected BAR settings. However, because the PIO design parameters are more limited than the core parameters, consider the following example design limitations when changing the default CORE Generator software parameters:

- The example design supports one I/O space BAR, one 32-bit Memory space (that cannot be the Expansion ROM space), and one 64-bit Memory space. If these limits are exceeded, only the first space of a given type is active—accesses to the other spaces do not result in completions.
- Each space is implemented with a 2 KB memory. If the corresponding BAR is configured to a wider aperture, accesses beyond the 2 KB limit wrap around and overlap the 2 KB memory space.
- The PIO design supports one I/O space BAR, which by default is disabled, but can be changed if desired.

Although there are limitations to the PIO design, Verilog or VHDL source code is provided so the user can tailor the example design to their specific needs.

## TLP Data Flow

This section defines the data flow of a TLP successfully processed by the PIO design. For detailed information about the interface signals within the sub-blocks of the PIO design, see Receive Path, page 228 and Transmit Path, page 230.

The PIO design successfully processes single DWORD payload Memory Read and Write TLPs and I/O Read and Write TLPs. Memory Read or Memory Write TLPs of lengths larger than one DWORD are not processed correctly by the PIO design; however, the core *does* accept these TLPs and passes them along to the PIO design. If the PIO design receives a TLP with a length of greater than 1 DWORD, the TLP is received completely from the core and discarded. No corresponding completion is generated.

### Memory and I/O Write TLP Processing

When the Endpoint for PCIe receives a Memory or I/O Write TLP, the TLP destination address and transaction type are compared with the values in the core BARs. If the TLP passes this comparison check, the core passes the TLP to the Receive TRN interface of the PIO design. The PIO design handles Memory writes and I/O TLP writes in different ways: the PIO design responds to *I/O writes* by generating a Completion Without Data (cpl), a requirement of the PCI Express specification.

Along with the start of packet, end of packet, and ready handshaking signals, the Receive TRN interface also asserts the appropriate trn_rbar_hit_n[6:0] signal to indicate to the PIO design the specific destination BAR that matched the incoming TLP. On reception, the PIO design's RX State Machine processes the incoming Write TLP and extracts the TLPs data and relevant address fields so that it can pass this along to the PIO design's internal block RAM write request controller.

Based on the specific trn_rbar_hit_n[6:0] signal asserted, the RX State Machine indicates to the internal write controller the appropriate 2 KB block RAM to use prior to asserting the write enable request. For example, if an I/O Write Request is received by the core targeting BAR0, the core passes the TLP to the PIO design and asserts trn_rbar_hit_n[0]. The RX State machine extracts the lower address bits and the data field from the I/O Write TLP and instructs the internal Memory Write controller to begin a write to the block RAM.

In this example, the assertion of trn_rbar_hit_n[0] instructed the PIO memory write controller to access ep_mem0 (which by default represents 2 KB of I/O space). While the write is being carried out to the FPGA block RAM, the PIO design RX state machine deasserts the trn_rdst_rdy_n signal, causing the Receive TRN interface to stall receiving any further TLPs until the internal Memory Write controller completes the write to the block RAM. Deasserting trn_rdst_rdy_n in this way is not required for all designs using the core—the PIO design uses this method to simplify the control logic of the RX state machine.

### Memory and I/O Read TLP Processing

When the Endpoint for PCIe receives a Memory or I/O Read TLP, the TLP destination address and transaction type are compared with the values programmed in the core BARs. If the TLP passes this comparison check, the core passes the TLP to the Receive TRN interface of the PIO design.

Along with the start of packet, end of packet, and ready handshaking signals, the Receive TRN interface also asserts the appropriate trn_rbar_hit_n[6:0] signal to indicate to the PIO design the specific destination BAR that matched the incoming TLP. On reception, the PIO design's state machine processes the incoming Read TLP and extracts the relevant TLP information and passes it along to the PIO design's internal block RAM read request controller.

Based on the specific trn_rbar_hit_n[6:0] signal asserted, the RX state machine indicates to the internal read request controller the appropriate 2 KB block RAM to use before asserting the read enable request. For example, if a Memory Read 32 Request TLP is received by the core targeting the default MEM32 BAR2, the core passes the TLP to the PIO design and asserts trn_rbar_hit_n[2]. The RX State machine extracts the lower address bits from the Memory 32 Read TLP and instructs the internal Memory Read Request controller to start a read operation.

In this example, the assertion of trn_rbar_hit_n[2] instructs the PIO memory read controller to access the Mem32 space, which by default represents 2 KB of memory space. A notable difference in handling of memory write and read TLPs is the requirement of the receiving device to return a Completion with Data TLP in the case of memory or I/O read request.

While the read is being processed, the PIO design RX state machine deasserts trn_rdst_rdy_n, causing the Receive TRN interface to stall receiving any further TLPs until the internal Memory Read controller completes the read access from the block RAM and generates the completion. Deasserting trn_rst_rdy_n in this way is not required for all designs using the core. The PIO design uses this method to simplify the control logic of the RX state machine.

## PIO File Structure

Table A-2 defines the PIO design file structure. Based on the specific core targeted, not all files delivered by the CORE Generator software are necessary, and some files might not be delivered. The major difference is that some of the Endpoint for PCIe solutions use a 32-bit user datapath, others use a 64-bit datapath, and the PIO design works with both. The width of the datapath depends on the specific core being targeted.

*Table A-2:* **PIO Design File Structure**

| File | Description |
|------|-------------|
| PIO.v | Top-level design wrapper |
| PIO_EP.v | PIO application module |
| PIO_TO_CTRL.v | PIO turn-off controller module |
| PIO_32.v | 32-bit interface macro define |
| PIO_64.v | 64-bit interface macro define |
| PIO_128.v | 128-bit interface macro define |
| PIO_32_RX_ENGINE.v | 32-bit Receive engine |
| PIO_32_TX_ENGINE.v | 32-bit Transmit engine |
| PIO_64_RX_ENGINE.v | 64-bit Receive engine |
| PIO_64_TX_ENGINE.v | 64-bit Transmit engine |
| PIO_128_RX_ENGINE.v | 128-bit Receive engine |
| PIO_128_TX_ENGINE.v | 128-bit Transmit engine |
| PIO_EP_MEM_ACCESS.v | Endpoint memory access module |
| PIO_EP_MEM.v | Endpoint memory |

Three configurations of the PIO design are provided: PIO_32, PIO_64, and PIO_128 with 32-, 64-, and 128-bit TRN interfaces, respectively. The PIO configuration generated depends on the selected endpoint type (that is, Virtex-6 FPGA Integrated Block, PIPE, PCI Express, and Block Plus) as well as the number of PCI Express lanes selected by the user. Table A-3 identifies the PIO configuration generated based on the user's selection.
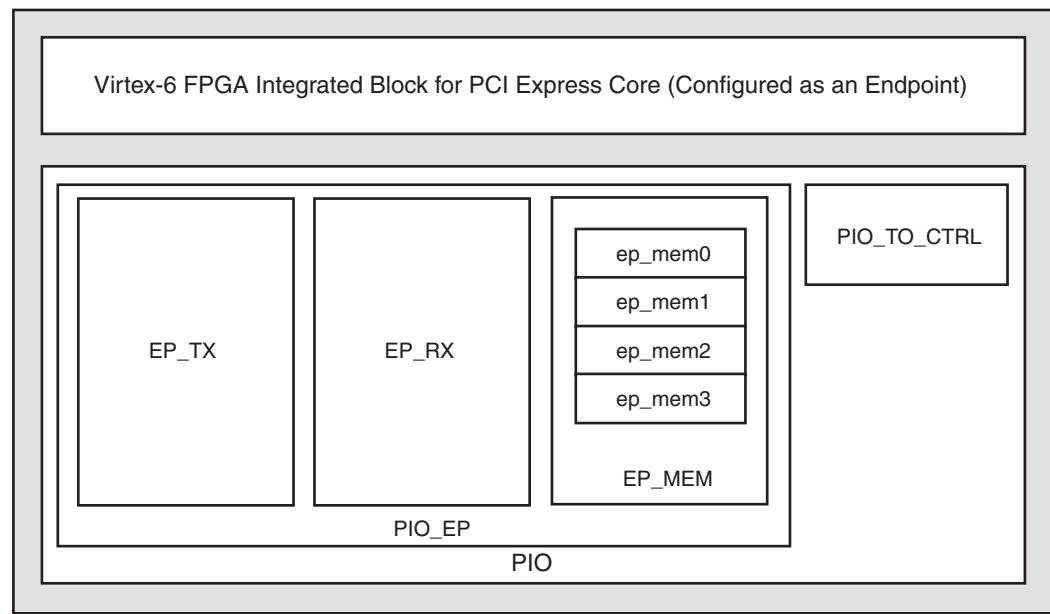
*Table A-3:* **PIO Configuration**

| Core | x1 | x2 | x4 | x8 |
|------|-----|-----|-----|-----|
| Endpoint for PIPE | PIO_32 | NA | NA | NA |
| Endpoint for PCI Express | PIO_32 | NA | PIO_64 | PIO_64 |
| Endpoint for PCI Express Block Plus | PIO_64 | NA | PIO_64 | PIO_64 |
| Virtex-6 FPGA Integrated Block | PIO_64 | PIO_64 | PIO_64 | PIO_64, PIO_128[1] |
| Spartan®-6 FPGA Integrated Endpoint Block | PIO_32 | NA | NA | NA |

**Notes:**

1. The PIO_128 configuration is only provided for the 128-bit x8 5.0 Gb/s core.

Figure A-2 shows the various components of the PIO design, which is separated into four main parts: the TX Engine, RX Engine, Memory Access Controller, and Power Management Turn-Off Controller.



UG517_aA_02_021210

*Figure A-2:* **PIO Design Components**

## PIO Application

Figure A-3 and Figure A-4 depict 64-bit and 32-bit PIO application top-level connectivity, respectively. The datapath width, either 32 bits or 64 bits, depends on which Endpoint for PCIe core is used The PIO_EP module contains the PIO FPGA block RAM modules and the transmit and receive engines. The PIO_TO_CTRL module is the Endpoint Turn-Off controller unit, which responds to power turn-off message from the host CPU with an acknowledgment.

The PIO_EP module connects to the Endpoint Transaction (trn) and Configuration (cfg) interfaces.
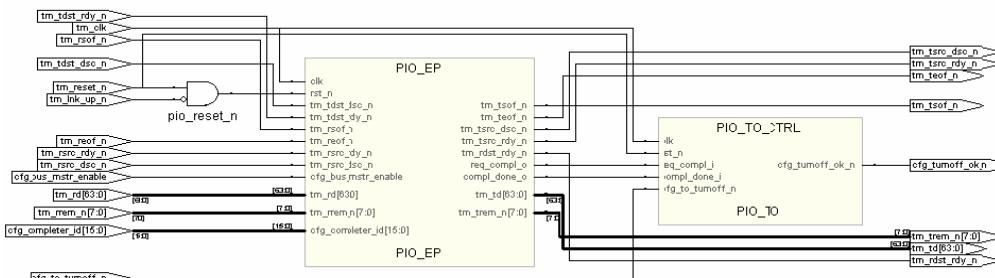


*Figure A-3:* **PIO 64-bit Application**
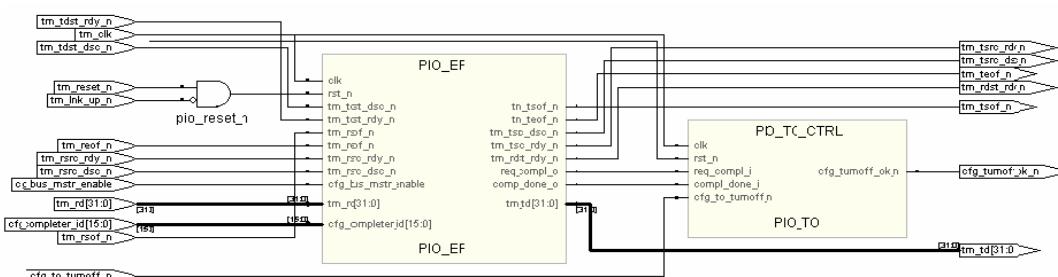


*Figure A-4:* **PIO 32-bit Application**

## Receive Path

Figure A-5 illustrates the PIO_32_RX_ENGINE and PIO_64_RX_ENGINE modules. The datapath of the module must match the datapath of the core being used. These modules connect with Endpoint for PCIe Transaction Receive (trn_r*) interface.
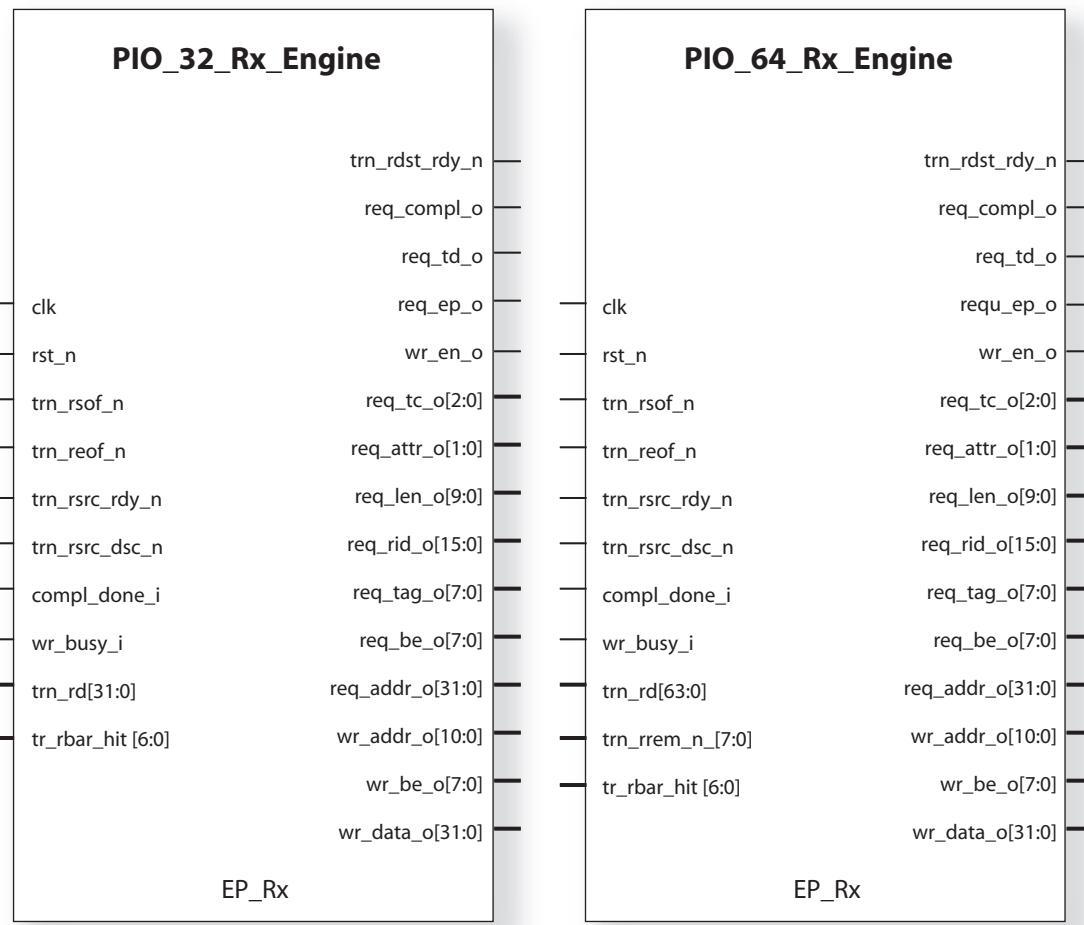


*Figure A-5:* **RX Engines**

The PIO_32_RX_ENGINE, PIO_64_RX_ENGINE and PIO_128_RX_ENGINE modules receive and parse incoming read and write TLPs.

The RX engine parses 1 DWORD 32- and 64-bit addressable memory and I/O read requests. The RX state machine extracts needed information from the TLP and passes it to the memory controller, as defined in Table A-4.

*Table A-4:* **RX Engine: Read Outputs**

| Port | Description |
|---|---|
| req_compl_o | Completion request (active High) |
| req_td_o | Request TLP Digest bit |
| req_ep_o | Request Error Poisoning bit |
| req_tc_o[2:0] | Request Traffic Class |

*Table A-4:* **RX Engine: Read Outputs** *(Cont'd)*

| Port | Description |
|------|-------------|
| req_attr_o[1:0] | Request Attributes |
| req_len_o[9:0] | Request Length |
| req_rid_o[15:0] | Request Requester Identifier |
| req_tag_o[7:0] | Request Tag |
| req_be_o[7:0] | Request Byte Enable |
| req_addr_o[10:0] | Request Address |

The RX Engine parses 1 DWORD 32- and 64-bit addressable memory and I/O write requests. The RX state machine extracts needed information from the TLP and passes it to the memory controller, as defined in Table A-5.

*Table A-5:* **Rx Engine: Write Outputs**

| Port | Description |
|------|-------------|
| wr_en_o | Write received |
| wr_addr_o[10:0] | Write address |
| wr_be_o[7:0] | Write byte enable |
| wr_data_o[31:0] | Write data |

The read datapath stops accepting new transactions from the core while the application is processing the current TLP. This is accomplished by trn_rdst_rdy_n deassertion. For an ongoing Memory or I/O Read transaction, the module waits for compl_done_i input to be asserted before it accepts the next TLP, while an ongoing Memory or I/O Write transaction is deemed complete after wr_busy_i is deasserted.

## Transmit Path

Figure A-6 shows the PIO_32_TX_ENGINE and PIO_64_TX_ENGINE modules. The datapath of the module must match the datapath of the core being used. These modules connect with the core Transaction Transmit (trn_r*) interface.
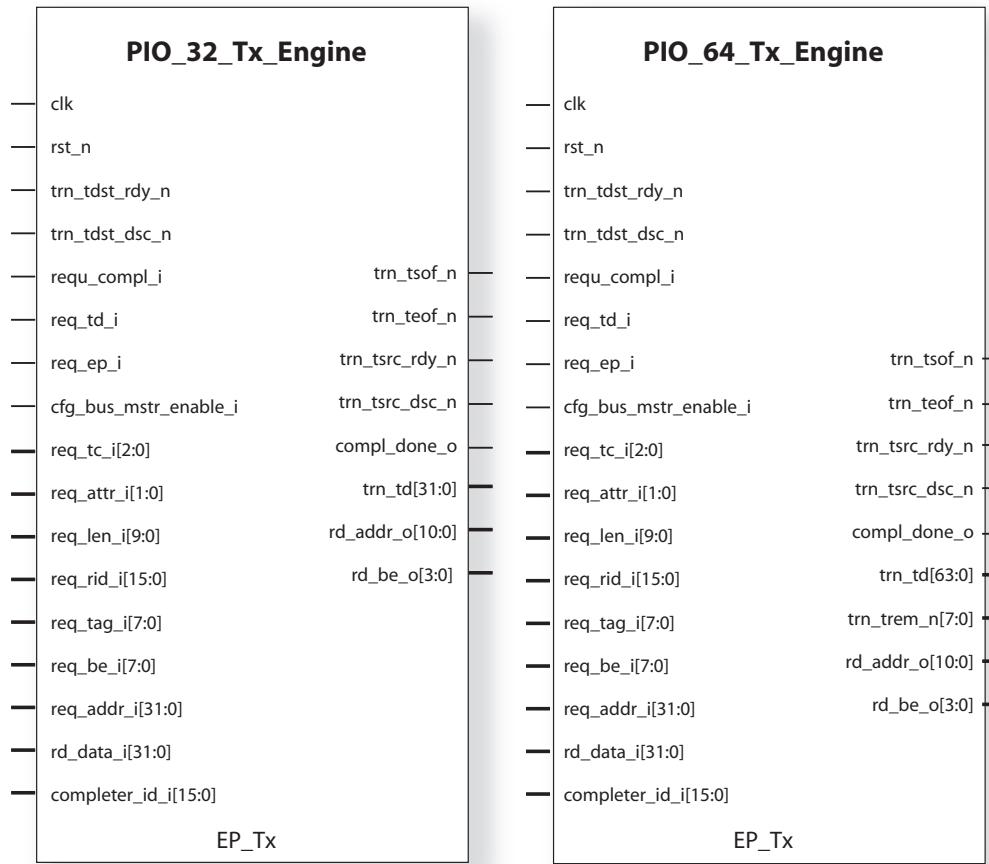


*Figure A-6:* **Tx Engines**

The PIO_32_TX_ENGINE, PIO_64_TX_ENGINE and PIO_128_TX_ENGINE modules generate completions for received memory and I/O read TLPs. The PIO design does not generate outbound read or write requests. However, users can add this functionality to further customize the design.

The PIO_32_TX_ENGINE, PIO_64_TX_ENGINE, and PIO_128_TX_ENGINE modules generate completions in response to 1 DWORD 32- and 64-bit addressable memory and I/O read requests. Information necessary to generate the completion is passed to the TX Engine, as defined in Table A-6.

*Table A-6:* **TX Engine Inputs**

| Port | Description |
|---|---|
| req_compl_i | Completion request (active High) |
| req_td_i | Request TLP Digest bit |
| req_ep_i | Request Error Poisoning bit |
| req_tc_i[2:0] | Request Traffic Class |
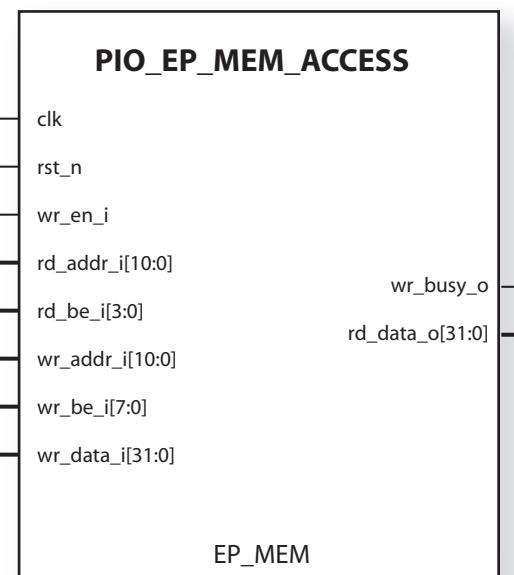
*Table A-6:* **TX Engine Inputs** *(Cont'd)*

| Port | Description |
|------|-------------|
| req_attr_i[1:0] | Request Attributes |
| req_len_i[9:0] | Request Length |
| req_rid_i[15:0] | Request Requester Identifier |
| req_tag_i[7:0] | Request Tag |
| req_be_i[7:0] | Request Byte Enable |
| req_addr_i[10:0] | Request Address |

After the completion is sent, the TX engine asserts the compl_done_i output indicating to the RX engine that it can assert trn_rdst_rdy_n and continue receiving TLPs.

## Endpoint Memory

Figure A-7 displays the PIO_EP_MEM_ACCESS module. This module contains the Endpoint memory space.



*Figure A-7:* **EP Memory Access**

The PIO_EP_MEM_ACCESS module processes data written to the memory from incoming Memory and I/O Write TLPs and provides data read from the memory in response to Memory and I/O Read TLPs.

The EP_MEM module processes 1 DWORD 32- and 64-bit addressable Memory and I/O Write requests based on the information received from the RX Engine, as defined in Table A-7. While the memory controller is processing the write, it asserts the wr_busy_o output indicating it is busy.

*Table A-7:* **EP Memory: Write Inputs**

| Port | Description |
|------|-------------|
| wr_en_i | Write received |
| wr_addr_i[10:0] | Write address |
| wr_be_i[7:0] | Write byte enable |
| wr_data_i[31:0] | Write data |

Both 32- and 64-bit Memory and I/O Read requests of one DWORD are processed based on the inputs defined in Table A-8. After the read request is processed, the data is returned on rd_data_o[31:0].

*Table A-8:* **EP Memory: Read Inputs**

| Port | Description |
|------|-------------|
| req_be_i[7:0] | Request Byte Enable |
| req_addr_i[31:0] | Request Address |

## PIO Operation

### PIO Read Transaction

Figure A-8 depicts a Back-to-Back Memory Read request to the PIO design. The receive engine deasserts trn_rdst_rdy_n as soon as the first TLP is completely received. The next Read transaction is accepted only after compl_done_o is asserted by the transmit engine, indicating that Completion for the first request was successfully transmitted.
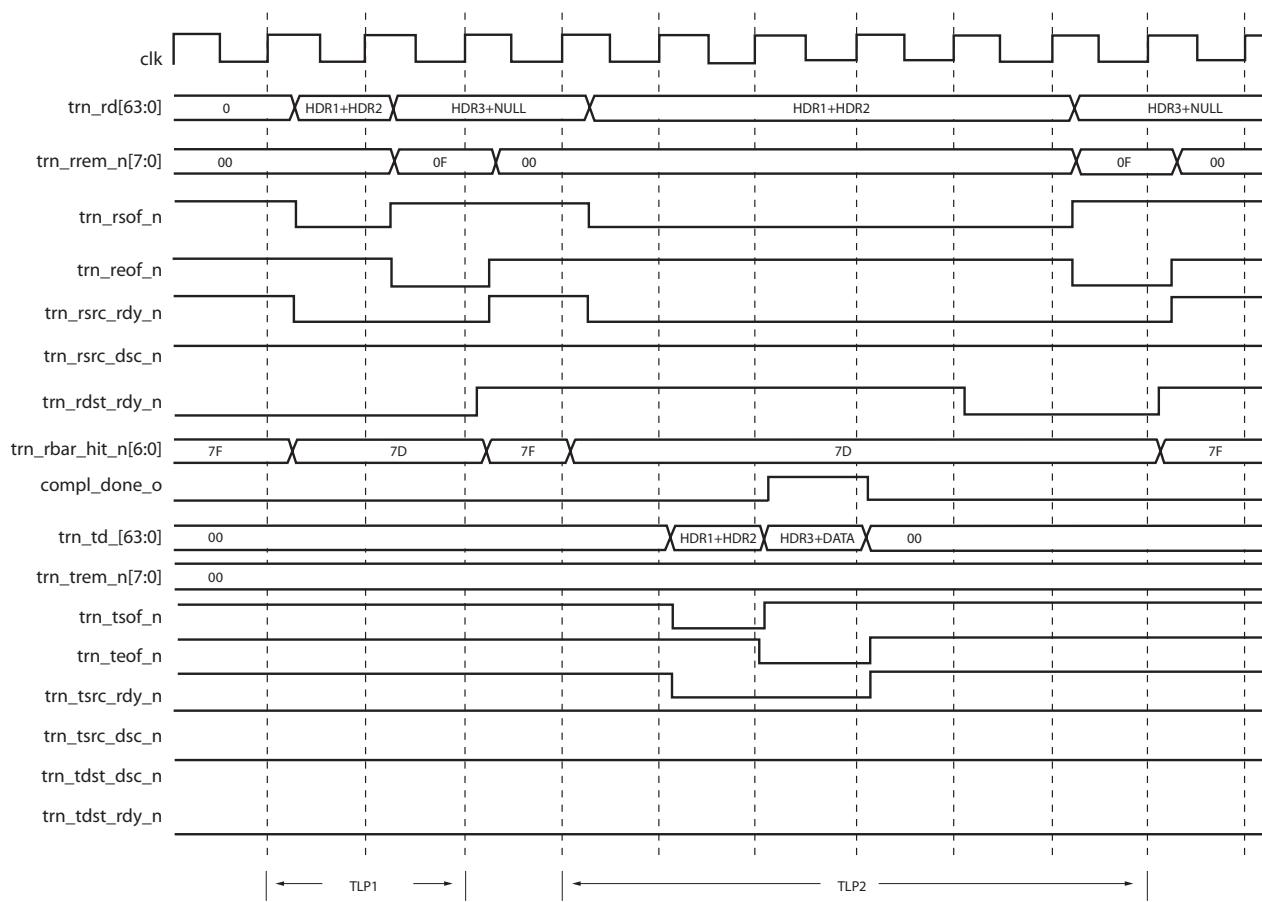


*Figure A-8:* **Back-to-Back Read Transactions**

## PIO Write Transaction

Figure A-9 depicts a back-to-back Memory Write to the PIO design. The next Write transaction is accepted only after wr_busy_o is deasserted by the memory access unit, indicating that data associated with the first request was successfully written to the memory aperture.
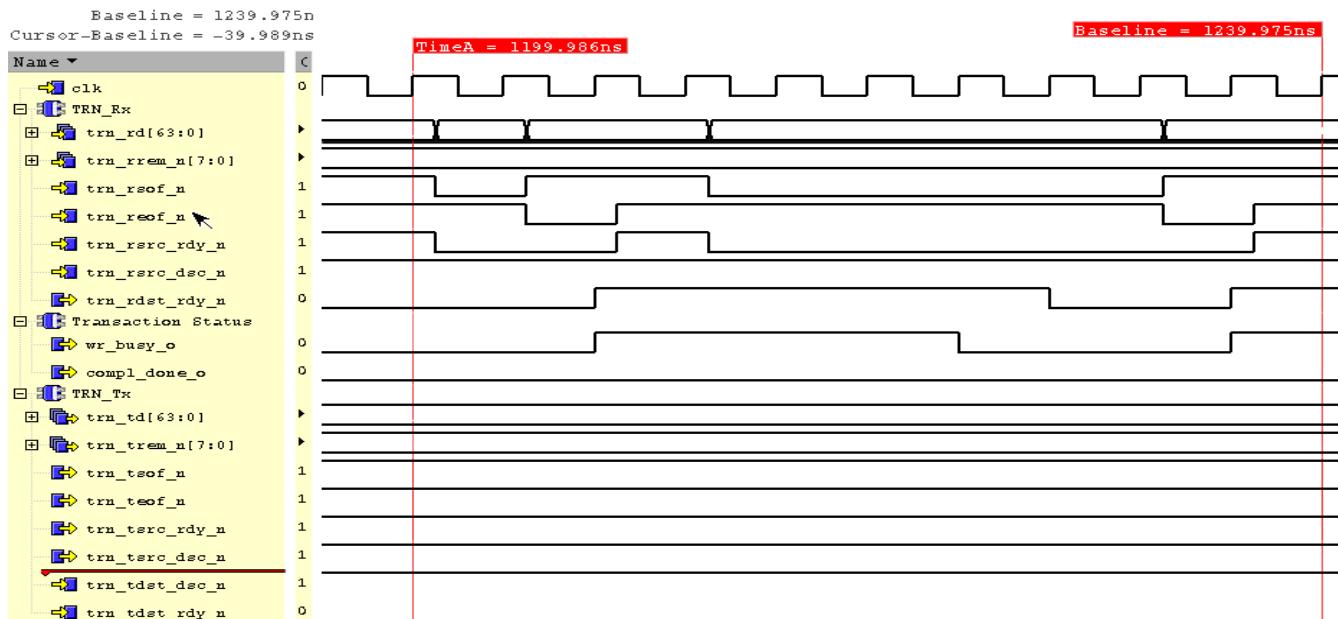


*Figure A-9:* **Back-to-Back Write Transactions**

## Device Utilization

Table A-9 shows the PIO design FPGA resource utilization.

*Table A-9:* **PIO Design FPGA Resources**

| Resources | Utilization |
|---|---|
| LUTs | 300 |
| Flip-Flops | 500 |
| Block RAMs | 4 |

# Summary

The PIO design demonstrates the Endpoint for PCIe and its interface capabilities. In addition, it enables rapid bring-up and basic validation of end user endpoint add-in card FPGA hardware on PCI Express platforms. Users can leverage standard operating system utilities that enable generation of read and write transactions to the target space in the reference design.

# Root Port Model Test Bench for Endpoint

The PCI Express Root Port Model is a robust test bench environment that provides a test program interface that can be used with the provided PIO design or with the user's design. The purpose of the Root Port Model is to provide a source mechanism for generating downstream PCI Express TLP traffic to stimulate the customer design, and a destination mechanism for receiving upstream PCI Express TLP traffic from the customer design in a simulation environment.

**Note:** The Root Port Model is shared by the Virtex-6 FPGA Integrated Block and Endpoint Block Plus for PCI Express solutions. This appendix represents these solutions.

Source code for the Root Port Model is included to provide the model for a starting point for the user test bench. All the significant work for initializing the core's configuration space, creating TLP transactions, generating TLP logs, and providing an interface for creating and verifying tests are complete, allowing the user to dedicate efforts to verifying the correct functionality of the design rather than spending time developing an Endpoint core test bench infrastructure.

The Root Port Model consists of:

- Test Programming Interface (TPI), which allows the user to stimulate the Endpoint device for the PCI Express

- Example tests that illustrate how to use the test program TPI

- Verilog or VHDL source code for all Root Port Model components, which allow the user to customize the test bench

Figure A-10 illustrates the illustrates the Root Port Model coupled with the PIO design.
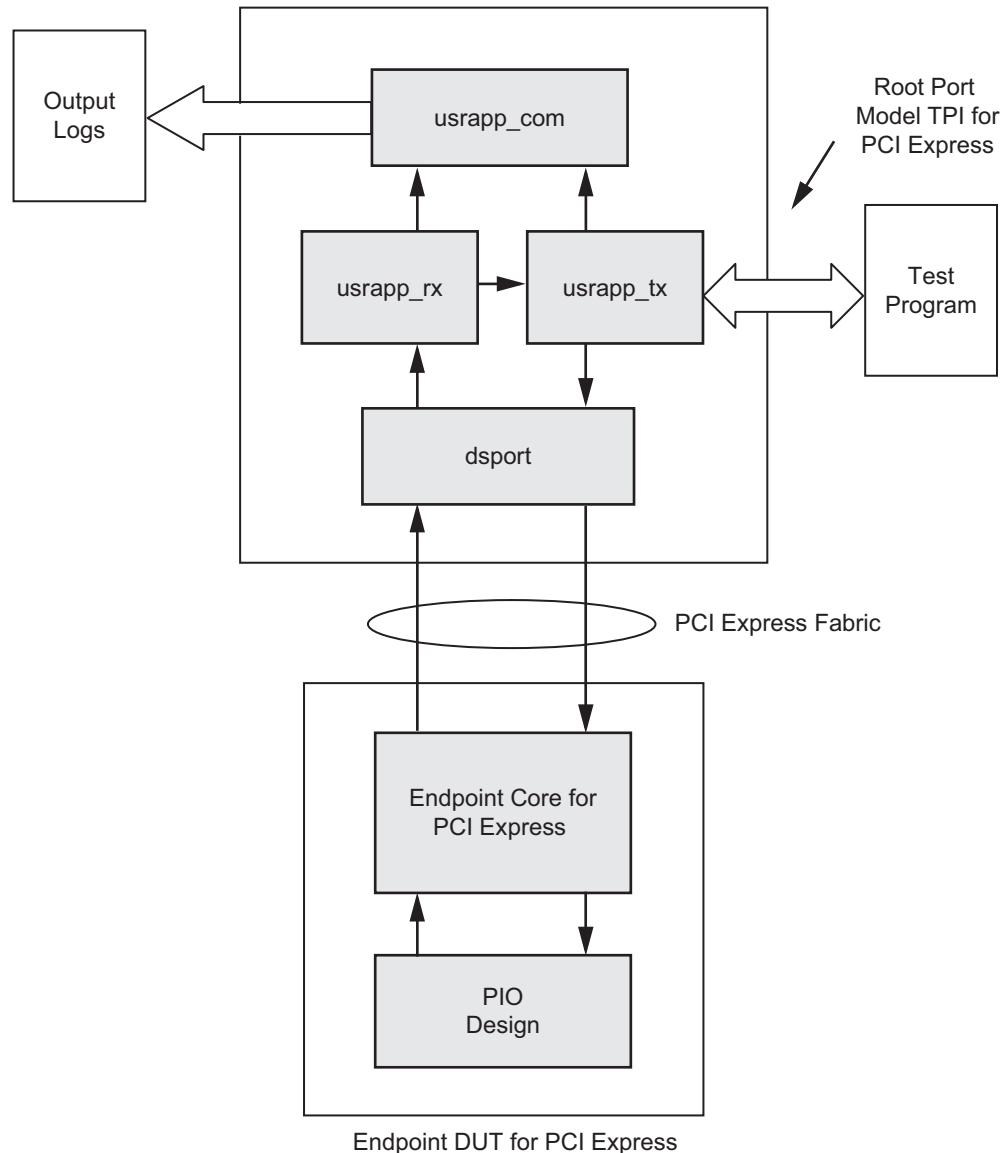
*Figure A-10:* **Root Port Model and Top-Level Endpoint**

## Architecture

The Root Port Model consists of these blocks, illustrated in Figure A-10:

- dsport (Root Port)
- usrapp_tx
- usrapp_rx
- usrapp_com (Verilog only)

The usrapp_tx and usrapp_rx blocks interface with the dsport block for transmission and reception of TLPs to/from the Endpoint Design Under Test (DUT). The Endpoint DUT consists of the Endpoint for PCIe and the PIO design (displayed) or customer design.

The usrapp_tx block sends TLPs to the dsport block for transmission across the PCI Express Link to the Endpoint DUT. In turn, the Endpoint DUT device transmits TLPs

across the PCI Express Link to the dsport block, which are subsequently passed to the usrapp_rx block. The dsport and core are responsible for the data link layer and physical link layer processing when communicating across the PCI Express fabric. Both usrapp_tx and usrapp_rx utilize the usrapp_com block for shared functions, for example, TLP processing and log file outputting. Transaction sequences or test programs are initiated by the usrapp_tx block to stimulate the endpoint device's fabric interface. TLP responses from the endpoint device are received by the usrapp_rx block. Communication between the usrapp_tx and usrapp_rx blocks allow the usrapp_tx block to verify correct behavior and act accordingly when the usrapp_rx block has received TLPs from the endpoint device.

## Simulating the Design

Three simulation script files are provided with the model to facilitate simulation with Synopsys VCS and VCS MX, Cadence INCISIV, Mentor Graphics ModelSim, and Xilinx ISim simulators:

- `simulate_vcs.sh` (Verilog Only)
- `simulate_ncsim.sh`
- `simulate_isim.bat/simulate_isim.sh`
- `simulate_mti.do`

The example simulation script files are located in the following directory:

    <project_dir>/<component_name>/simulation/functional

Instructions for simulating the PIO design using the Root Port Model are provided in Chapter 4, Getting Started Example Design.

***Note:*** For Cadence INCISIV users, the work construct must be manually inserted into the cds.lib file: `DEFINE WORK WORK`.

## Scaled Simulation Timeouts

The simulation model of the Virtex-6 FPGA Integrated Block for PCI Express uses scaled down times during link training to allow for the link to train in a reasonable amount of time during simulation. According to the *PCI Express Specification, rev. 2.0*, there are various timeouts associated with the link training and status state machine (LTSSM) states. The Virtex-6 FPGA Integrated Block for PCI Express scales these timeouts by a factor of 256 in simulation, except in the Recovery Speed_1 LTSSM state, where the timeouts are not scaled.

## Test Selection

Table A-10 describes the tests provided with the Root Port Model, followed by specific sections for VHDL and Verilog test selection.

*Table A-10:* **Root Port Model Provided Tests**

| Test Name | Test in VHDL/Verilog | Description |
|---|---|---|
| sample_smoke_test0 | Verilog and VHDL | Issues a PCI Type 0 Configuration Read TLP and waits for the completion TLP; then compares the value returned with the expected Device/Vendor ID value. |
| sample_smoke_test1 | Verilog | Performs the same operation as sample_smoke_test0 but makes use of expectation tasks. This test uses two separate test program threads: one thread issues the PCI Type 0 Configuration Read TLP and the second thread issues the Completion with Data TLP expectation task. This test illustrates the form for a parallel test that uses expectation tasks. This test form allows for confirming reception of any TLPs from the customer's design. Additionally, this method can be used to confirm reception of TLPs when ordering is unimportant. |

### VHDL Test Selection

Test selection is implemented in the VHDL Downstream Port Model by means of overriding the test_selector generic within the *tests* entity. The test_selector generic is a string with a one-to-one correspondence to each test within the tests entity.

The user can modify the generic mapping of the instantiation of the tests entity within the pci_exp_usrapp_tx entity. Currently, there is one test defined inside the tests entity, `sample_smoke_test0`. Additional customer-defined tests should be added inside `tests.vhd`. Currently, specific tests cannot be selected from the VHDL simulation scripts.

### Verilog Test Selection

The Verilog test model used for the Root Port Model lets the user specify the name of the test to be run as a command line parameter to the simulator. For example, the `simulate_ncsim.sh` script file, used to start the Cadence INCISIV simulator, explicitly specifies the test `sample_smoke_test0` to be run using this command line syntax:

```
ncsim work.boardx01 +TESTNAME=sample_smoke_test0
```

To change the test to be run, change the value provided to TESTNAME defined in the test files `sample_tests1.v` and `pio_tests.v`. The same mechanism is used for VCS and ModelSim. ISim uses the -testplusarg options to specify TESTNAME, for example:

```
demo_tb.exe -gui -view wave.wcfg -wdb wave_isim -tclbatch isim_cmd.tcl
-testplusarg TESTNAME=sample_smoke_test0
```

### VHDL and Verilog Root Port Model Differences

The following subsections identify differences between the VHDL and Verilog Root Port Model.

### Verilog Expectation Tasks

The most significant difference between the Verilog and the VHDL test bench is that the Verilog test bench has Expectation Tasks. Expectation tasks are API calls used in conjunction with a bus mastering customer design. The test program issues a series of expectation task calls, that is, the task calls expect a memory write TLP and a memory read TLP. If the customer design does not respond with the expected TLPs, the test program fails. This functionality was implemented using the fork-join construct in Verilog, which is not available in VHDL and subsequently not implemented.

### Verilog Command Line versus VHDL tests.vhd Module

The Verilog test bench allows test programs to be specified at the command line, while the VHDL test bench specifies test programs within the `tests.vhd` module.

### Generating Wave Files

- The Verilog test bench uses recordvars and dumpfile commands within the code to generate wave files.
- The VHDL test bench leaves the generating wave file functionality up to the simulator.

### Speed Differences

The VHDL test bench is slower than the Verilog test bench, especially when testing the x8 core. For initial design simulation and speed enhancement, the user might want to use the x1 core, identify basic functionality issues, and then move to x4 or x8 simulation when testing design performance.

## Waveform Dumping

Table A-11 describes the available simulator waveform dump file formats, each of which is provided in the simulator's native file format. The same mechanism is used for VCS and ModelSim.

*Table A-11:* **Simulator Dump File Format**

| Simulator | Dump File Format |
|---|---|
| Synopsys VCS | `.vpd` |
| ModelSim | `.vcd` |
| Cadence INCISIV | `.trn` |
| ISim | `.wdb` |

### VHDL Flow

Waveform dumping in the VHDL flow does not use the +dump_all mechanism described in the Verilog Flow section. Because the VHDL language itself does not provide a common interface for dumping waveforms, each VHDL simulator has its own interface for supporting waveform dumping. For both the supported ModelSim and INCISIV flows, dumping is supported by invoking the VHDL simulator command line with a command line option that specifies the respective waveform command file, `wave.do` (ModelSim) and `wave.sv` (INCISIV) and `wave.wcfg` (ISim). This command line can be found in the respective simulation script files `simulate_mti.do`, `simulate_ncsim.sh`, and `simulate_isim.bat[.sh]`.

### ModelSim

This command line initiates waveform dumping for the ModelSim flow using the VHDL test bench:

```
>vsim +notimingchecks -do wave.do -L unisim -L work work.board
```

### INCISIV

This command line initiates waveform dumping for the INCISIV flow using the VHDL test bench:

```
>ncsim -gui work.board -input @"simvision -input wave.sv"
```

## Verilog Flow

The Root Port Model provides a mechanism for outputting the simulation waveform to file by specifying the +dump_all command line parameter to the simulator.

For example, the script file `simulate_ncsim.sh` (used to start the Cadence INCISIV simulator) can indicate to the Root Port Model that the waveform should be saved to a file using this command line:

```
ncsim work.boardx01 +TESTNAME=sample_smoke_test0 +dump_all
```

# Output Logging

When a test fails on the example or customer design, the test programmer debugs the offending test case. Typically, the test programmer inspects the wave file for the simulation and cross-reference this to the messages displayed on the standard output. Because this approach can be very time consuming, the Root Port Model offers an output logging mechanism to assist the tester with debugging failing test cases to speed the process.

The Root Port Model creates three output files (`tx.dat`, `rx.dat`, and `error.dat`) during each simulation run. Log files `rx.dat` and `tx.dat` each contain a detailed record of every TLP that was received and transmitted, respectively, by the Root Port Model. With an understanding of the expected TLP transmission during a specific test case, the test programmer can more easily isolate the failure.

The log file `error.dat` is used in conjunction with the expectation tasks. Test programs that utilize the expectation tasks will generate a general error message to standard output. Detailed information about the specific comparison failures that have occurred due to the expectation error is located within error.dat.

# Parallel Test Programs

There are two classes of tests are supported by the Root Port Model:

- Sequential tests. Tests that exist within one process and behave similarly to sequential programs. The test depicted in Test Program: pio_writeReadBack_test0, page 242 is an example of a sequential test. Sequential tests are very useful when verifying behavior that have events with a known order.

- Parallel tests. Tests involving more than one process thread. The test `sample_smoke_test1` is an example of a parallel test with two process threads. Parallel tests are very useful when verifying that a specific set of events have occurred, however the order of these events are not known.

A typical parallel test uses the form of one command thread and one or more expectation threads. These threads work together to verify a device's functionality. The role of the command thread is to create the necessary TLP transactions that cause the device to receive and generate TLPs. The role of the expectation threads is to verify the reception of an expected TLP. The Root Port Model TPI has a complete set of expectation tasks to be used in conjunction with parallel tests.

Because the example design is a target-only device, only Completion TLPs can be expected by parallel test programs while using the PIO design. However, the full library of expectation tasks can be used for expecting any TLP type when used in conjunction with the customer's design (which can include bus-mastering functionality). Currently, the VHDL version of the Root Port Model Test Bench does not support Parallel tests.

## Test Description

The Root Port Model provides a Test Program Interface (TPI). The TPI provides the means to create tests by simply invoking a series of Verilog tasks. All Root Port Model tests should follow the same six steps:

1. Perform conditional comparison of a unique test name
2. Set up master timeout in case simulation hangs
3. Wait for Reset and link-up
4. Initialize the configuration space of the endpoint
5. Transmit and receive TLPs between the Root Port Model and the Endpoint DUT
6. Verify that the test succeeded

## Test Program: pio_writeReadBack_test0

```
1.      else if(testname == "pio_writeReadBack_test1"
2.      begin
3.      // This test performs a 32 bit write to a 32 bit Memory space and performs a read back
4.      TSK_SIMULATION_TIMEOUT(10050);
5.      TSK_SYSTEM_INITIALIZATION;
6.      TSK_BAR_INIT;
7.      for (ii = 0; ii <= 6; ii = ii + 1) begin
8.          if (BAR_INIT_P_BAR_ENABLED[ii] > 2'b00) // bar is enabled
9.          case(BAR_INIT_P_BAR_ENABLED[ii])
10.             2'b01 : // IO SPACE
11.             begin
12.                 $display("[%t] : NOTHING: to IO 32 Space BAR %x", $realtime, ii);
13.             end
14.             2'b10 : // MEM 32 SPACE
15.             begin
16.             $display("[%t] : Transmitting TLPs to Memory 32 Space BAR %x",
17.                         $realtime, ii);
18.          //-----------------------------------------------------------------------
19.          // Event : Memory Write 32 bit TLP
20.          //-----------------------------------------------------------------------
21.                 DATA_STORE[0] = 8'h04;
22.                 DATA_STORE[1] = 8'h03;
23.                 DATA_STORE[2] = 8'h02;
24.                 DATA_STORE[3] = 8'h01;
25.                 P_READ_DATA = 32'hffff_ffff; // make sure P_READ_DATA has known initial value
26.                 TSK_TX_MEMORY_WRITE_32(DEFAULT_TAG, DEFAULT_TC, 10'd1, BAR_INIT_P_BAR[ii][31:0] , 4'hF,
        4'hF, 1'b0);
27.                 TSK_TX_CLK_EAT(10);
28.                 DEFAULT_TAG = DEFAULT_TAG + 1;
29.          //-----------------------------------------------------------------------
30.          // Event : Memory Read 32 bit TLP
31.          //-----------------------------------------------------------------------
32.                 TSK_TX_MEMORY_READ_32(DEFAULT_TAG, DEFAULT_TC, 10'd1, BAR_INIT_P_BAR[ii][31:0], 4'hF,
        4'hF);
33.                 TSK_WAIT_FOR_READ_DATA;
34.                 if  (P_READ_DATA != {DATA_STORE[3], DATA_STORE[2], DATA_STORE[1], DATA_STORE[0] })
35.                   begin
36.                    $display("[%t] : Test FAILED --- Data Error Mismatch, Write Data %x != Read Data %x",
        $realtime,{DATA_STORE[3], DATA_STORE[2], DATA_STORE[1], DATA_STORE[0]},  P_READ_DATA);
37.                   end
38.                 else
39.                   begin
40.                    $display("[%t] : Test PASSED --- Write Data: %x successfully received", $realtime,
        P_READ_DATA);
41.                   end
```

## Expanding the Root Port Model

The Root Port Model was created to work with the PIO design, and for this reason is tailored to make specific checks and warnings based on the limitations of the PIO design. These checks and warnings are enabled by default when the Root Port Model is generated by the CORE Generator software. However, these limitations can easily be disabled so that they do not affect the customer's design.

Because the PIO design was created to support at most one I/O BAR, one Mem64 BAR, and two Mem32 BARs (one of which must be the EROM space), the Root Port Model by default makes a check during device configuration that verifies that the core has been configured to meet this requirement. A violation of this check causes a warning message to be displayed as well as for the offending BAR to be gracefully disabled in the test bench. This check can be disabled by setting the `pio_check_design` variable to zero in the `pci_exp_usrapp_tx.v` file.

## Root Port Model TPI Task List

The Root Port Model TPI tasks include the following, which are further defined in Tables A-12 through A-16.

- Test Setup Tasks
- TLP Tasks
- BAR Initialization Tasks
- Example PIO Design Tasks
- Expectation Tasks

*Table A-12:* **Test Setup Tasks**

| Name | Input(s) | | Description |
|---|---|---|---|
| TSK_SYSTEM_INITIALIZATION | None | | Waits for transaction interface reset and link-up between the Root Port Model and the Endpoint DUT. |
| | | | This task must be invoked prior to the Endpoint core initialization. |
| TSK_USR_DATA_SETUP_SEQ | None | | Initializes global 4096 byte DATA_STORE array entries to sequential values from zero to 4095. |
| TSK_TX_CLK_EAT | clock count | 31:30 | Waits clock_count transaction interface clocks. |
| TSK_SIMULATION_TIMEOUT | timeout | 31:0 | Sets master simulation timeout value in units of transaction interface clocks. This task should be used to ensure that all DUT tests complete. |

*Table A-13:* **TLP Tasks**

| Name | Input(s) | | Description |
|---|---|---|---|
| TSK_TX_TYPE0_CONFIGURATION_READ | tag_<br>reg_addr_<br>first_dw_be_ | 7:0<br>11:0<br>3:0 | Waits for transaction interface reset and link-up between the Root Port Model and the Endpoint DUT. |
| | | | This task must be invoked prior to Endpoint core initialization. |
| TSK_TX_TYPE1_CONFIGURATION_READ | tag_<br>reg_addr_<br>first_dw_be_ | 7:0<br>11:0<br>3:0 | Sends a Type 1 PCI Express Config Read TLP from Root Port Model to reg_addr_ of Endpoint DUT with tag_ and first_dw_be_ inputs. |
| | | | CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |

*Table A-13:* **TLP Tasks** *(Cont'd)*

| Name | Input(s) | | Description |
|------|----------|--|-------------|
| TSK_TX_TYPE0_CONFIGURATION_WRITE | tag_<br>reg_addr_<br>reg_data_<br>first_dw_be_ | 7:0<br>11:0<br>31:0<br>3:0 | Sends a Type 0 PCI Express Config Write TLP from Root Port Model to reg_addr_ of Endpoint DUT with tag_ and first_dw_be_ inputs.<br>Cpl returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_TYPE1_CONFIGURATION_WRITE | tag_<br>reg_addr_<br>reg_data_<br>first_dw_be_ | 7:0<br>11:0<br>31:0<br>3:0 | Sends a Type 1 PCI Express Config Write TLP from Root Port Model to reg_addr_ of Endpoint DUT with tag_ and first_dw_be_ inputs.<br>Cpl returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_MEMORY_READ_32 | tag_<br>tc_<br>len_<br>addr_<br>last_dw_be_<br>first_dw_be_ | 7:0<br>2:0<br>9:0<br>31:0<br>3:0<br>3:0 | Sends a PCI Express Memory Read TLP from Root Port to 32-bit memory address addr_ of Endpoint DUT.<br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_MEMORY_READ_64 | tag_<br>tc_<br>len_<br>addr_<br>last_dw_be_<br>first_dw_be_ | 7:0<br>2:0<br>9:0<br>63:0<br>3:0<br>3:0 | Sends a PCI Express Memory Read TLP from Root Port Model to 64-bit memory address addr_ of Endpoint DUT.<br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_MEMORY_WRITE_32 | tag_<br>tc_<br>len_<br>addr_<br>last_dw_be_<br>first_dw_be_<br>ep_ | 7:0<br>2:0<br>9:0<br>31:0<br>3:0<br>3:0<br>– | Sends a PCI Express Memory Write TLP from Root Port Model to 32-bit memory address addr_ of Endpoint DUT.<br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID.<br>The global DATA_STORE byte array is used to pass write data to task. |
| TSK_TX_MEMORY_WRITE_64 | tag_<br>tc_<br>len_<br>addr_<br>last_dw_be_<br>first_dw_be_<br>ep_ | 7:0<br>2:0<br>9:0<br>63:0<br>3:0<br>3:0<br>– | Sends a PCI Express Memory Write TLP from Root Port Model to 64-bit memory address addr_ of Endpoint DUT.<br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID.<br>The global DATA_STORE byte array is used to pass write data to task. |

*Table A-13:* **TLP Tasks** *(Cont'd)*

| Name | Input(s) | | Description |
|------|----------|---|-------------|
| TSK_TX_COMPLETION | tag_<br>tc_<br>len_<br>comp_status_ | 7:0<br>2:0<br>9:0<br>2:0 | Sends a PCI Express Completion TLP from Root Port Model to the Endpoint DUT using global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_COMPLETION_DATA | tag_<br>tc_<br>len_<br>byte_count<br>lower_addr<br>comp_status<br>ep_ | 7:0<br>2:0<br>9:0<br>11:0<br>6:0<br>2:0<br>– | Sends a PCI Express Completion with Data TLP from Root Port Model to the Endpoint DUT using global COMPLETE_ID_CFG as the completion ID.<br>The global DATA_STORE byte array is used to pass completion data to task. |
| TSK_TX_MESSAGE | tag_<br>tc_<br>len_<br>data<br>message_rtg<br>message_code | 7:0<br>2:0<br>9:0<br>63:0<br>2:0<br>7:0 | Sends a PCI Express Message TLP from Root Port Model to Endpoint DUT.<br>Completion returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_MESSAGE_DATA | tag_<br>tc_<br>len_<br>data<br>message_rtg<br>message_code | 7:0<br>2:0<br>9:0<br>63:0<br>2:0<br>7:0 | Sends a PCI Express Message with Data TLP from Root Port Model to Endpoint DUT.<br>The global DATA_STORE byte array is used to pass message data to task.<br>Completion returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_IO_READ | tag_<br>addr_<br>first_dw_be_ | 7:0<br>31:0<br>3:0 | Sends a PCI Express I/O Read TLP from Root Port Model to I/O address addr_[31:2] of the Endpoint DUT.<br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_IO_WRITE | tag_<br>addr_<br>first_dw_be_<br>data | 7:0<br>31:0<br>3:0<br>31:0 | Sends a PCI Express I/O Write TLP from Root Port Model to I/O address addr_[31:2] of the Endpoint DUT.<br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |

*Table A-13:* **TLP Tasks** *(Cont'd)*

| Name | Input(s) | | Description |
|---|---|---|---|
| TSK_TX_BAR_READ | bar_index<br>byte_offset<br>tag_<br>tc_ | 2:0<br>31:0<br>7:0<br>2:0 | Sends a PCI Express 1 DWORD Memory 32, Memory 64, or I/O Read TLP from the Root Port Model to the target address corresponding to offset byte_offset from BAR bar_index of the Endpoint DUT. This task sends the appropriate Read TLP based on how BAR bar_index has been configured during initialization. This task can only be called after TSK_BAR_INIT has successfully completed.<br><br>CplD returned from the Endpoint DUT will use the contents of global COMPLETE_ID_CFG as the completion ID. |
| TSK_TX_BAR_WRITE | bar_index<br>byte_offset<br>tag_<br>tc_<br>data_ | 2:0<br>31:0<br>7:0<br>2:0<br>31:0 | Sends a PCI Express 1 DWORD Memory 32, Memory 64, or I/O Write TLP from the Root Port to the target address corresponding to offset byte_offset from BAR bar_index of the Endpoint DUT.<br><br>This task sends the appropriate Write TLP based on how BAR bar_index has been configured during initialization. This task can only be called after TSK_BAR_INIT has successfully completed. |
| TSK_WAIT_FOR_READ_DATA | None | | Waits for the next completion with data TLP that was sent by the Endpoint DUT. On successful completion, the first DWORD of data from the CplD will be stored in the global P_READ_DATA. This task should be called immediately following any of the read tasks in the TPI that request Completion with Data TLPs to avoid any race conditions.<br><br>By default this task will locally time out and terminate the simulation after 1000 transaction interface clocks. The global cpld_to_finish can be set to zero so that local time out returns execution to the calling test and does not result in simulation timeout. For this case test programs should check the global cpld_to, which when set to one indicates that this task has timed out and that the contents of P_READ_DATA are invalid. |

*Table A-14:* **BAR Initialization Tasks**

| Name | Input(s) | Description |
|------|----------|-------------|
| TSK_BAR_INIT | None | Performs a standard sequence of Base Address Register initialization tasks to the Endpoint device using the PCI Express fabric. Performs a scan of the Endpoint's PCI BAR range requirements, performs the necessary memory and I/O space mapping calculations, and finally programs the Endpoint so that it is ready to be accessed. On completion, the user test program can begin memory and I/O transactions to the device. This function displays to standard output a memory and I/O table that details how the Endpoint has been initialized. This task also initializes global variables within the Root Port Model that are available for test program usage. This task should only be called after TSK_SYSTEM_INITIALIZATION. |
| TSK_BAR_SCAN | None | Performs a sequence of PCI Type 0 Configuration Writes and Configuration Reads using the PCI Express fabric in order to determine the memory and I/O requirements for the Endpoint. The task stores this information in the global array BAR_INIT_P_BAR_RANGE[]. This task should only be called after TSK_SYSTEM_INITIALIZATION. |
| TSK_BUILD_PCIE_MAP | None | Performs memory and I/O mapping algorithm and allocates Memory 32, Memory 64, and I/O space based on the Endpoint requirements. This task has been customized to work in conjunction with the limitations of the PIO design and should only be called after completion of TSK_BAR_SCAN. |
| TSK_DISPLAY_PCIE_MAP | None | Displays the memory mapping information of the Endpoint core's PCI Base Address Registers. For each BAR, the BAR value, the BAR range, and BAR type is given. This task should only be called after completion of TSK_BUILD_PCIE_MAP. |

*Table A-15:* **Example PIO Design Tasks**

| Name | Input(s) | | Description |
|---|---|---|---|
| TSK_TX_READBACK_CONFIG | None | | Performs a sequence of PCI Type 0 Configuration Reads to the Endpoint device's Base Address Registers, PCI Command Register, and PCIe Device Control Register using the PCI Express fabric. |
| | | | This task should only be called after TSK_SYSTEM_INITIALIZATION. |
| TSK_MEM_TEST_DATA_BUS | bar_index | 2:0 | Tests whether the PIO design FPGA block RAM data bus interface is correctly connected by performing a 32-bit walking ones data test to the I/O or memory address pointed to by the input bar_index. |
| | | | For an exhaustive test, this task should be called four times, once for each block RAM used in the PIO design. |
| TSK_MEM_TEST_ADDR_BUS | bar_index<br>nBytes | 2:0<br>31:0 | Tests whether the PIO design FPGA block RAM address bus interface is accurately connected by performing a walking ones address test starting at the I/O or memory address pointed to by the input bar_index. |
| | | | For an exhaustive test, this task should be called four times, once for each block RAM used in the PIO design. Additionally, the nBytes input should specify the entire size of the individual block RAM. |
| TSK_MEM_TEST_DEVICE | bar_index<br>nBytes | 2:0<br>31:0 | Tests the integrity of each bit of the PIO design FPGA block RAM by performing an increment/decrement test on all bits starting at the block RAM pointed to by the input bar_index with the range specified by input nBytes. |
| | | | For an exhaustive test, this task should be called four times, once for each block RAM used in the PIO design. Additionally, the nBytes input should specify the entire size of the individual block RAM. |

*Table A-16:* **Expectation Tasks**

| Name | Input(s) | | Output | Description |
|---|---|---|---|---|
| TSK_EXPECT_CPLD | traffic_class | 2:0 | Expect status | Waits for a Completion with Data TLP that matches traffic_class, td, ep, attr, length, and payload. Returns a 1 on successful completion; 0 otherwise. |
| | td | - | | |
| | ep | - | | |
| | attr | 1:0 | | |
| | length | 9:0 | | |
| | completer_id | 15:0 | | |
| | completer_status | 2:0 | | |
| | bcm | - | | |
| | byte_count | 11:0 | | |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | |
| | address_low | 6:0 | | |
| TSK_EXPECT_CPL | traffic_class | 2:0 | Expect status | Waits for a Completion without Data TLP that matches traffic_class, td, ep, attr, and length. Returns a 1 on successful completion; 0 otherwise. |
| | td | - | | |
| | ep | - | | |
| | attr | 1:0 | | |
| | completer_id | 15:0 | | |
| | completer_status | 2:0 | | |
| | bcm | - | | |
| | byte_count | 11:0 | | |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | |
| | address_low | 6:0 | | |
| TSK_EXPECT_MEMRD | traffic_class | 2:0 | Expect status | Waits for a 32-bit Address Memory Read TLP with matching header fields. Returns a 1 on successful completion; 0 otherwise. This task can only be used in conjunction with Bus Master designs. |
| | td | - | | |
| | ep | - | | |
| | attr | 1:0 | | |
| | length | 9:0 | | |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | |
| | last_dw_be | 3:0 | | |
| | first_dw_be | 3:0 | | |
| | address | 29:0 | | |

*Table A-16:* **Expectation Tasks** *(Cont'd)*

| Name | Input(s) | | Output | Description |
|---|---|---|---|---|
| TSK_EXPECT_MEMRD64 | traffic_class | 2:0 | Expect status | Waits for a 64-bit Address Memory Read TLP with matching header fields. Returns a 1 on successful completion; 0 otherwise. |
| | td | - | | |
| | ep | - | | |
| | attr | 1:0 | | |
| | length | 9:0 | | This task can only be used in conjunction with Bus Master designs. |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | |
| | last_dw_be | 3:0 | | |
| | first_dw_be | 3:0 | | |
| | address | 61:0 | | |
| TSK_EXPECT_MEMWR | traffic_class | 2:0 | Expect status | Waits for a 32-bit Address Memory Write TLP with matching header fields. Returns a 1 on successful completion; 0 otherwise. |
| | td | - | | |
| | ep | - | | |
| | attr | 1:0 | | |
| | length | 9:0 | | This task can only be used in conjunction with Bus Master designs. |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | |
| | last_dw_be | 3:0 | | |
| | first_dw_be | 3:0 | | |
| | address | 29:0 | | |
| TSK_EXPECT_MEMWR64 | traffic_class | 2:0 | Expect status | Waits for a 64-bit Address Memory Write TLP with matching header fields. Returns a 1 on successful completion; 0 otherwise. |
| | td | - | | |
| | ep | - | | |
| | attr | 1:0 | | |
| | length | 9:0 | | This task can only be used in conjunction with Bus Master designs. |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | |
| | last_dw_be | 3:0 | | |
| | first_dw_be | 3:0 | | |
| | address | 61:0 | | |
| TSK_EXPECT_IOWR | td | - | Expect status | Waits for an I/O Write TLP with matching header fields. Returns a 1 on successful completion; 0 otherwise. |
| | ep | - | | |
| | requester_id | 15:0 | | |
| | tag | 7:0 | | This task can only be used in conjunction with Bus Master designs. |
| | first_dw_be | 3:0 | | |
| | address | 31:0 | | |
| | data | 31:0 | | |

# Example Design and Model Test Bench for Root Port Configuration

## Configurator Example Design

The Configurator example design, included with the Virtex®-6 FPGA Integrated Block for PCI Express® in Root Port configuration generated by the CORE Generator™ software, is a synthesizeable, lightweight design that demonstrates the minimum setup required for the integrated block in Root Port configuration to begin application-level transactions with an Endpoint.

### System Overview

PCI Express devices require setup after power-on, before devices in the system can begin application specific communication with each other. Minimally, two devices connected via a PCI Express Link must have their Configuration spaces initialized and be enumerated in order to communicate.

Root Ports facilitate PCI Express enumeration and configuration by sending Configuration Read (CfgRd) and Write (CfgWr) TLPs to the downstream devices such as Endpoints and Switches to set up the configuration spaces of those devices. When this process is complete, higher-level interactions, such as Memory Reads (MemRd TLPs) and Writes (MemWr TLPs), can occur within the PCI Express System.

The Configurator example design described herein performs the configuration transactions required to enumerate and configure the Configuration space of a single connected PCI Express Endpoint and allow application-specific interactions to occur.

### Configurator Example Design Hardware

The Configurator example design consists of four high-level blocks:

- Root Port: The Virtex-6 FPGA Integrated Block in Root Port configuration.
- Configurator Block: Logical block which interacts with the configuration space of a PCI Express Endpoint device connected to the Root Port.
- Configurator ROM: Read-only memory that sources configuration transactions to the Configurator Block.
- PIO Master: Logical block which interacts with the user logic connected to the Endpoint by exchanging data packets and checking the validity of the received data. The data packets are limited to a single DWORD and represent the type of traffic that would be generated by a CPU.

**Note:** The Configurator Block and Configurator ROM, and Root Port are logically grouped in the RTL code within a wrapper file called the Configurator Wrapper.

The Configurator example design, as delivered, is designed to be used with the PIO Slave example included with Xilinx® Endpoint cores and described in Appendix A, Example Design and Model Test Bench for Endpoint Configuration. The PIO Master is useful for simple bring-up and debugging, and is an example of how to interact with the Configurator Wrapper. The Configurator example design can be easily modified to be used with other Endpoints.

Figure B-1 shows the various components of the Configurator example design.



UG517_aB_01_021210
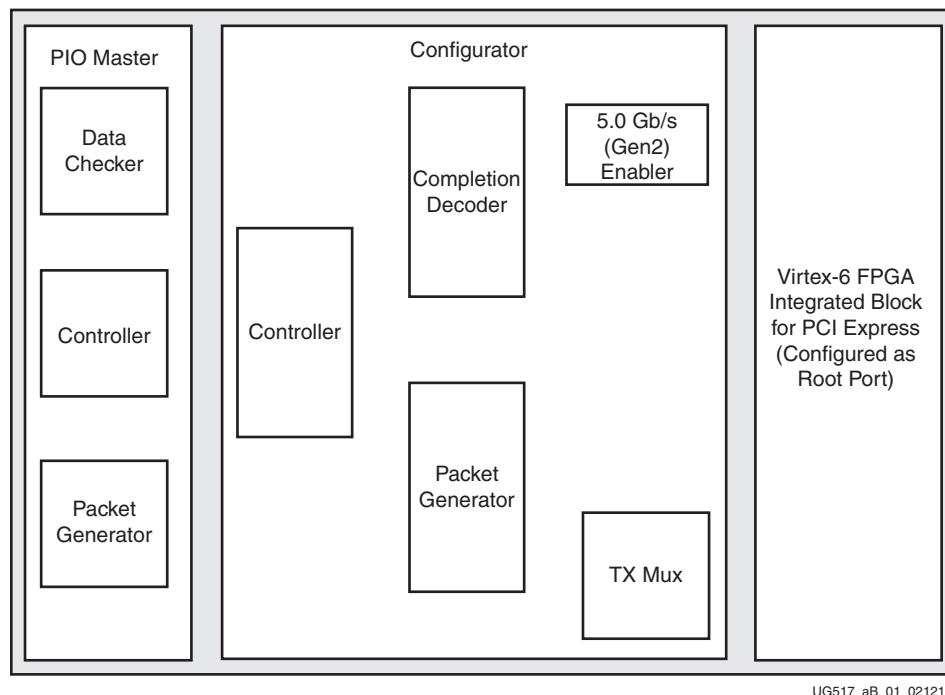
*Figure B-1:*  **Configurator Example Design Components**

Figure B-2 shows how the blocks are connected in an overall system view.
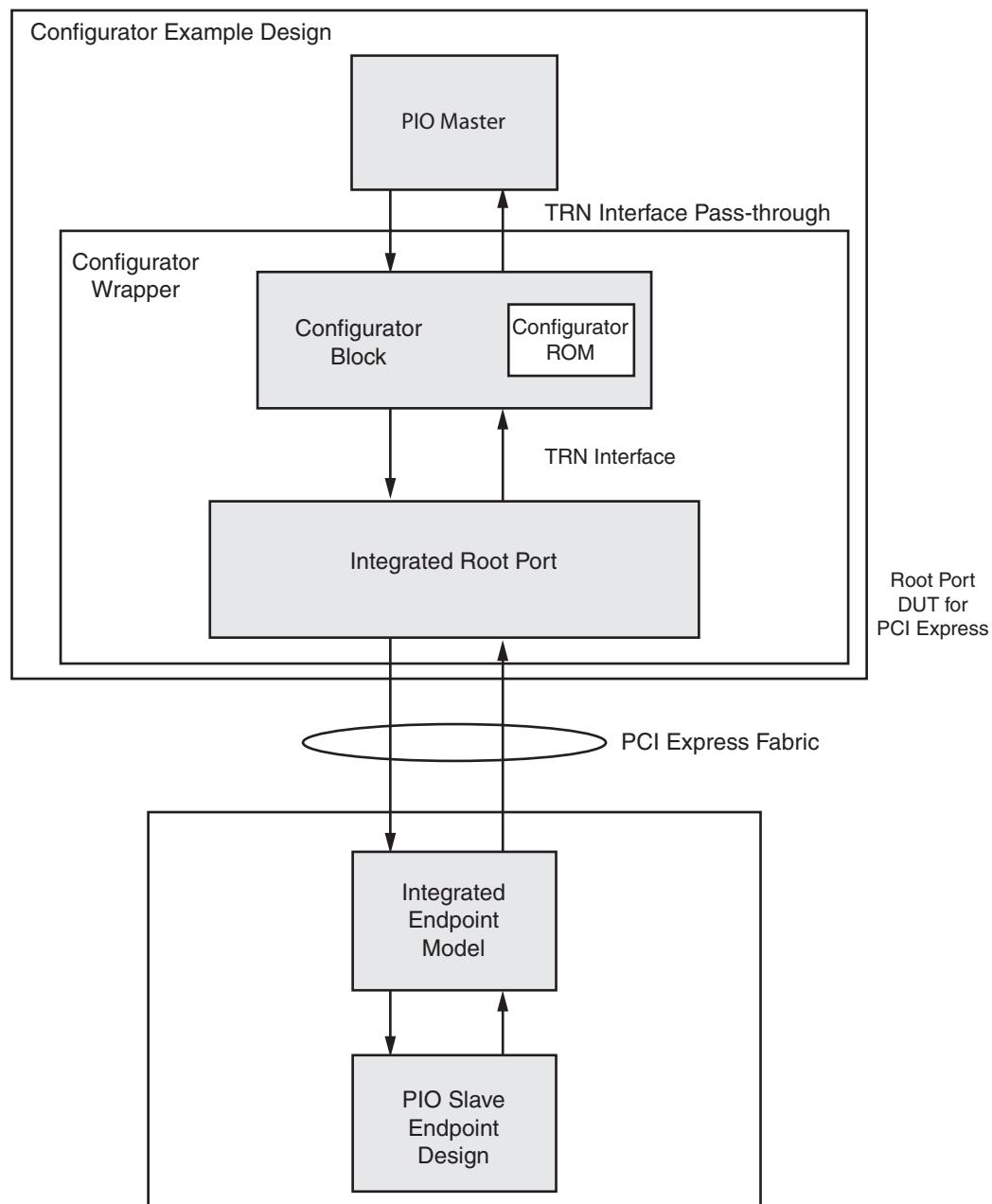


*Figure B-2:* **Configurator Example Design**

## Configurator Block

The Configurator Block is responsible for generating CfgRd and CfgWr TLPs and presenting them to TRN interface of the Integrated Block in Root Port configuration. The TLPs that the Configurator Block generates are determined by the contents of the Configurator ROM.

The generated configuration traffic is predetermined by the designer to address their particular system requirements. The configuration traffic is encoded in a memory-

initialization file (the Configurator ROM) which is synthesized as part of the Configurator. The Configurator Block and the attached Configurator ROM is intended to be usable a part of a real-world embedded design.

The Configurator Block steps through the Configuration ROM file and sends the TLPs specified therein. Supported TLP types are Message, Message w/Data, Configuration Write (Type 0), and Configuration Read (Type 0). For the Configuration packets, the Configurator Block waits for a Completion to be returned before transmitting the next TLP. If the Completion TLP fields do not match the expected values, PCI Express configuration fails. However, the Data field of Completion TLPs is ignored and not checked

***Note:*** There is no completion timeout mechanism in the Configurator Block, so if no Completion is returned, the Configurator Block waits forever.

The Configurator Block has these parameters, which can be altered by the user:

- **TCQ**: Clock-to-out delay modeled by all registers in design.

- **EXTRA_PIPELINE**: Controls insertion of an extra pipeline stage on the RX TRN interface for timing.

- **ROM_FILE**: File name containing configuration steps to perform.

- **ROM_SIZE**: Number of lines in ROM_FILE containing data (equals number of TLPs to send/2).

- **REQUESTER_ID**: Value for the Requester ID field in outgoing TLPs.

When the Configurator Block design is used, all TLP traffic must pass through the Configurator Block. The user design is responsible for asserting the start_config input (for one clock cycle) to initiate the configuration process when trn_lnk_up_n has been asserted by the core. Following start_config, the Configurator Block performs whatever configuration steps have been specified in the Configuration ROM. During configuration, the Configurator Block controls the core's TRN interface. Following configuration, all TRN traffic is routed to/from the User Application, which in the case of this example design is the PIO Master. The end of configuration is signaled by the assertion of finished_config. If configuration is unsuccessful for some reason, failed_config is also asserted.

If used in a system that supports PCIe v2.0 5.0 Gb/s links, the Configurator Block begins its process by attempting to up-train the link from 2.5 Gb/s to 5.0 Gb/s. This feature is enabled depending on the LINK_CAP_MAX_LINK_SPEED parameter on the Configurator Wrapper.

The Configurator does not support the user throttling received data on the trn_r* interface. Because of this, the Root Port inputs which control throttling are not included on the Configurator Wrapper. These signals are trn_rdst_rdy_n and trn_rnp_ok_n. This is a limitation of the Configurator Example Design and not of the Integrated Block for PCI Express in Root Port configuration. This means that the user design interfacing with the Configurator Example Design must be able to accept received data at line rate.

## Configurator ROM

The Configurator ROM stores the necessary configuration transactions to configure a PCI Express Endpoint. This ROM interfaces with the Configurator Block to send these transactions over the PCI Express link.

The example ROM file included with this design shows the operations needed to configure a Virtex-6 FPGA Integrated Endpoint Block for PCI Express and PIO Example Design.

The Configurator ROM can be customized for other Endpoints and PCI Express system topologies. The unique set of configuration transactions required depends on the Endpoint

that will be interacting with the Root Port. This information can be obtained from the documentation provided with the Endpoint.

The ROM file follows the format specified in the Verilog specification (IEEE 1364-2001) section 17.2.8, which describes using the $readmemb function to pre-load data into a RAM or ROM. Verilog-style comments are allowed.

The file is read by the simulator or synthesis tool and each memory value encountered is used as a single location in memory. Digits can be separated by an underscore character (_) for clarity without constituting a new location.

Each configuration transaction specified uses two adjacent memory locations - the first location specifies the header fields, while the second location specifies the 32-bit data payload. (For CfgRd TLPs and Messages without data, the data location is unused but still present). In other words, header fields are on even addresses, while data payloads are on odd addresses.

For headers, Messages and CfgRd/CfgWr TLPs use different fields. For all TLPs, two bits specify the TLP type. For Messages, Message Routing and Message Code are specified. For CfgRd/CfgWr TLPs, Function Number, Register Number, and 1st Dword Byte-Enable are specified. The specific bit layout is shown in the example ROM file.

## PIO Master

The PIO Master demonstrates how a user-application design might interact with the Configurator Block. It directs the Configurator Block to bring up the link partner at the appropriate time, and then (after successful bring-up) generates and consumes bus traffic. The PIO Master performs writes and reads across the PCI Express Link to the PIO Slave Example Design (from the Endpoint core) to confirm basic operation of the link and the Endpoint.

The PIO Master waits until trn_lnk_up_n is asserted by the Root Port. It then asserts start_config to the Configurator Block. When the Configurator Block asserts finished_config, the PIO Master writes and reads to/from each BAR in the PIO Slave design. If the readback data matches what was written, the PIO Master asserts its pio_test_finished output. If there is a data mismatch or the Configurator Block fails to configure the Endpoint, the PIO Master asserts its pio_test_failed output. The PIO Master's operation can be restarted by asserting its pio_test_restart input for one clock cycle.

## Configurator File Structure

Table B-1 defines the Configurator example design file structure.

*Table B-1:*   **Example Design File Structure**

| File | Description |
|------|-------------|
| `xilinx_pcie_2_0_rport_v6.v` | Top-level wrapper file for Configurator example design |
| `cgator_wrapper.v` | Wrapper for Configurator and Root Port |
| `cgator.v` | Wrapper for Configurator sub-blocks |
| `cgator_cpl_decoder.v` | Completion decoder |
| `cgator_pkt_generator.v` | Configuration TLP generator |
| `cgator_tx_mux.v` | TX TRN muxing logic |

*Table B-1:* **Example Design File Structure** *(Cont'd)*

| File | Description |
|------|-------------|
| `cgator_gen2_enabler.v` | 5.0 Gb/s directed speed change module |
| `cgator_controller.v` | Configurator transmit engine |
| `cgator_cfg_rom.data` | Configurator ROM file |
| `pio_master.v` | Wrapper for PIO Master |
| `pio_master_controller.v` | TX and RX Engine for PIO Master |
| `pio_master_checker.v` | Checks incoming User-Application Completion TLPs |
| `pio_master_pkt_generator.v` | Generates User-Application TLPs |

The hierarchy of the Configurator example design is:

- xilinx_pcie_2_0_rport_v6
  - cgator_wrapper
    - pcie_2_0_rport_v6 (in the source directory)
      This directory contains all the source files for the Integrated Block for PCI Express in Root Port Configuration.
    - cgator
    - cgator_cpl_decoder
    - cgator_pkt_generator
    - cgator_tx_mux
    - cgator_gen2_enabler
    - cgator_controller
      This directory contains <cgator_cfg_rom.data> (specified by ROM_FILE)*
  - pio_master
    - pio_master_controller
    - pio_master_checker
    - pio_master_pkt_generator

**Note:** `cgator_cfg_rom.data` is the default name of the ROM data file. The user can override this by changing the value of the ROM_FILE parameter.

## Configurator Example Design Summary

The Configurator example design is a synthesizable design that demonstrates the capabilities of the Virtex-6 FPGA Integrated Block for PCI Express when configured as a Root Port. The example is provided via the CORE Generator software and uses the Endpoint PIO example as a target for PCI Express enumeration and configuration. The design can be easily modified to target other Endpoints by changing the contents of a ROM file.

# Endpoint Model Test Bench for Root Port

The Endpoint model test bench for the Virtex-6 FPGA Integrated Block for PCI Express in Root Port configuration is a simple example test bench that connects the Configurator example design and the PCI Express Endpoint model allowing the two to operate like two devices in a physical system. As the Configurator example design consists of logic that initializes itself and generates and consumes bus traffic, the example test bench only implements logic to monitor the operation of the system and terminate the simulation.

The Endpoint model test bench consists of:

- Verilog or VHDL source code for all Endpoint model components
- PIO slave design

Figure B-2, page 253 illustrates the Endpoint model coupled with the Configurator example design.

## Architecture

The Endpoint model consists of these blocks:

- PCI Express Endpoint (Virtex-6 FPGA Integrated Block for PCI Express in Endpoint configuration) model.
- PIO slave design, consisting of:
    - PIO_RX_ENGINE
    - PIO_TX_ENGINE
    - PIO_EP_MEM
    - PIO_TO_CTRL

The PIO_RX_ENGINE and PIO_TX_ENGINE blocks interface with the ep block for reception and transmission of TLPs from/to the Root Port Design Under Test (DUT). The Root Port DUT consists of the Integrated Block for PCI Express configured as a Root Port and the Configurator Example Design, which consists of a Configurator block and a PIO Master design, or customer design.

The PIO slave design is described in detail in Appendix A, Programmed Input/Output: Endpoint Example Design.

## Simulating the Design

Three simulation script files are provided with the model to facilitate simulation with Synopsys VCS and VCS MX, Cadence INCISIV, Mentor Graphics ModelSim, and Xilinx ISE® Simulator (ISim) simulators:

- `simulate_vcs.sh` (Verilog only)
- `simulate_ncsim.sh`
- `simulate_isim.sh[.bat]`
- `simulate_mti.do`

The example simulation script files are located in this directory:

    <project_dir>/<component_name>/simulation/functional

Instructions for simulating the Configurator example design with the Endpoint model are provided in Chapter 4, Getting Started Example Design.

***Note:*** For Cadence INCISIV users, the work construct must be manually inserted into the cds.lib file: DEFINE WORK WORK.

## Scaled Simulation Timeouts

The simulation model of the Virtex-6 FPGA Integrated Block for PCI Express uses scaled down times during link training to allow for the link to train in a reasonable amount of time during simulation. According to the *PCI Express Specification, rev. 2.0*, there are various timeouts associated with the link training and status state machine (LTSSM) states. The Virtex-6 FPGA Integrated Block for PCI Express scales these timeouts by a factor of 256 in simulation, except in the Recovery Speed_1 LTSSM state, where the timeouts are not scaled.

## Waveform Dumping

Table B-2 describes the available simulator waveform dump file formats, each of which is provided in the simulators native file format. The same mechanism is used for VCS and ModelSim.

*Table B-2:*   **Simulator Dump File Format**

| Simulator | Dump File Format |
|---|---|
| Synopsys VCS and VCS MX | `.vpd` |
| ModelSim | `.vcd` |
| Cadence INCISIV | `.trn` |
| ISim | `.wdb` |

The Endpoint model test bench provides a mechanism for outputting the simulation waveform to file by specifying the +dump_all command line parameter to the simulator.

For example, the script file `simulate_ncsim.sh` (used to start the Cadence INCISIV simulator) can indicate to the Endpoint model that the waveform should be saved to a file using this command line:

```
ncsim work.boardx01 +dump_all
```

## Output Logging

The test bench will output messages, captured in the simulation log, indicating the time at which the following occur:

* trn_reset deasserted
* trn_lnk_up_n asserted
* cfg_done asserted by the Configurator
* pio_test_finished asserted by the PIO Master
* Simulation Timeout (if pio_test_finished or pio_test_failed never asserted)

**EXILINX**

*Appendix C*

# Migration Considerations

For users migrating to the Virtex®-6 FPGA Integrated Block for PCI Express® from the Endpoint Block Plus for PCI Express, the following list describes the differences in behaviors and options between the Virtex-6 FPGA Integrated Block for PCI Express core and the Endpoint Block Plus core.

## Core Capability Differences

- **Gen2 Speed Operation**: The Virtex-6 FPGA Integrated Block for PCI Express supports the 5.0 Gb/s speed operation for all available Capability Link Widths (x1, x2, x4, and x8).

- **Max Payload Size Capability**: The Virtex-6 FPGA Integrated Block for PCI Express supports a Max_Payload_Size capability of 1024 bytes, except the x8 5.0 Gb/s product configuration, which continues to support a Max_Payload_Size capability of 512 bytes. The products can be configured to support a smaller Max_Payload_Size capability, if required.

- **Link Up-configure**: Table C-1 describes the up-configure capability of the Virtex-6 FPGA Integrated Block for PCI Express.

*Table C-1:* **Up-configure Capability**

| Capability Link Width | Supported Link Speed | Up Configurability | |
|---|---|---|---|
| | | Current Link Speed = 2.5 Gb/s | Current Link Speed = 5.0 Gb/s |
| X1 | 2.5 Gb/s | - | NA |
| X1 | 5.0 Gb/s | X1 | X1 |
| X2 | 2.5 Gb/s | X1, X2 | NA |
| X2 | 5.0 Gb/s | X1, X2 | X1, X2 |
| X4 | 2.5 Gb/s | X1, X2, X4 | NA |
| X4 | 5.0 Gb/s | X1, X2, X4 | X1, X2, X4 |
| X8 | 2.5 Gb/s | X1, X2, X4, X8 | NA |
| X8 | 5.0 Gb/s | X1, X2, X4, X8 | X1, X2, X4, X8 |

# Transaction Interface

Table C-2 shows which transaction interface signals were changed, created, or deprecated.

*Table C-2:* **Transaction Interface Changes**

| Name | Signal | Description |
|---|---|---|
| **Changed Interface Signals** | | |
| Transaction Clock Frequency | trn_clk | Transaction clock speeds can be selected on the basis of the core's Capability Link Width and Supported Link Speed (2.5 Gb/s only or 5.0 Gb/s) and are constant throughout the operation of the core. See Transaction Interface in Chapter 2 for detailed information. |
| Transmit Data Remainder | trn_trem_n | The Transmit Data Remainder bus width varies between the two cores. See Transaction Interface in Chapter 2 for detailed information. |
| Transmit Buffers Available | trn_tbuf_av[5:0] | Bit definitions for trn_tbuf_av vary between the two cores. See Transaction Interface in Chapter 2 for detailed information. |
| **New Interface Signals** | | |
| Transmit Streaming | trn_tstr_n | Enables streaming (cut-through) mode of transmission of a TLP, when possible, to reduce latency of operation. |
| Transmit Error Forward | trn_terrfwd_n | Marks the current packet in progress as error-poisoned. |
| Transmit Error Drop | trn_terr_drop_n | This new core output signal indicates that the core has internally discarded a packet (TLP) due to either a Max_Payload_Size rule violation or the streaming mode requirement violation. |
| Transmit Configuration Request/Grant | trn_tcfg_req_n trn_tcfg_gnt_n | When asserted, this core output signal warns the User Application that a core internally generated TLP will be arbitrating for the transmit datapath. For the User Application generated TLP to get priority over the core generated TLP, the User Application must hold the core input signal trn_tcfg_gnt_n de-asserted until the user transaction is complete, and then assert it for 1 cycle. Alternately, the User Application can choose to ignore trn_tcfg_req_n by simply driving trn_tcfg_gnt_n asserted (0b) permanently. This lets the core decide the arbitration priority. |

*Table C-2:*   **Transaction Interface Changes**

| Name | Signal | Description |
|------|--------|-------------|
| Flow Control Information | trn_fc_sel[2:0]<br>trn_fc_ph[7:0]<br>trn_fc_pd[11:0]<br>trn_fc_nph[7:0]<br>trn_fc_npd[11:0]<br>trn_fc_cplh[7:0]<br>trn_fc_cpld[11:0] | trn_fc_sel[2:0] input is used to select the type of Flow Control information available to the User Application. See Flow Control Credit Information in Chapter 6 for detailed information. |
| **Deprecated Interface Signals** | | |
| | trn_tdst_dsc_n | |
| | trn_rfc_ph_av[7:0] | |
| | trn_rfc_pd_av[11:0] | |
| | trn_rfc_nph_av[7:0] | |
| | trn_rfc_npd_av[11:0] | |
| | trn_rfc_cplh_av[7:0] | |
| | trn_rfc_cpld_av[11:0] | |

# Configuration Interface

Table C-2 shows which configuration interface signals were changed and created.

*Table C-3:*   **Configuration Interface Changes**

| Name | Signal | Description |
|------|--------|-------------|
| **Changed Interface Signals** | | |
| Configuration Write | cfg_di[31:0]<br>cfg_wr_en_n | The Configuration Write Interface is supported in the Virtex-6 FPGA Integrated Block for PCI Express core. This consists of the Configuration Write Data (cfg_di[31:0]) and Configuration Write Enable (cfg_wr_en_n). See Design with Configuration Space Registers and Configuration Interface in Chapter 6 for detailed information. |
| Configuration Byte Enable | cfg_byte_en_n[3:0] | The Configuration Byte Enable for configuration register access is supported in the Virtex-6 FPGA Integrated Block for PCI Express core. See Design with Configuration Space Registers and Configuration Interface in Chapter 6 for detailed information. |
| PCI Express Link State | cfg_pcie_link_state_n[2:0] | Bit definitions for cfg_pcie_link_state_n[2:0] vary between the two cores. See Design with Configuration Space Registers and Configuration Interface in Chapter 6 for detailed information. |
| **New Interface Signals** | | |
| Configuration Interrupt MSI-X Enabled | cfg_interrupt_msixenable | Indicates that MSI-X messaging is enabled. SeeGenerating Interrupt Requests in Chapter 6 for detailed information. |

*Table C-3:* **Configuration Interface Changes** *(Cont'd)*

| Name | Signal | Description |
|------|--------|-------------|
| Configuration Interrupt MSI-X Function Mask | cfg_interrupt_msixfm | Indicates the state of the Function Mask bit in the MSI-X Message Control field. See Generating Interrupt Requests in Chapter 6 for detailed information. |
| Configuration Device Command 2 | cfg_dcommand2 | The Configuration Device Command 2 is the Device Control 2 Register from the PCI Express Extended Capability Structure. See Design with Configuration Space Registers and Configuration Interface in Chapter 6 for detailed information. |
| PME Turn OFF | cfg_to_turnoff_n cfg_turnoff_ok_n | A new core handshake mechanism whereby the output cfg_to_turnoff_n notifies the user that a PME_TURN_OFF message has been received by the core and the core starts polling the cfg_turnoff_ok_n input signal. See Power Management in Chapter 6 for detailed information. |
| Power Management Control Status Register | cfg_pmcsr_pme_en cfg_pmcsr_pme_status cfg_pmcsr_powerstate[1:0] | In the Power Management Control/Status Register, these new output signals indicate the status of:<br>• cfg_pmcsr_pme_en: PME_En bit (bit 8)<br>• cfg_pmcsr_pme_status: PME_Status bit (bit 15)<br>• cfg_pmcsr_powerstate[1:0]: PowerState bits (bits1:0) |

# Physical Layer Interface

The Physical Layer Interface is a new interface in the Virtex-6 FPGA Integrated Block for PCI Express. This interface enables the user design to inspect the status of the Link and Link Partner and control the Link State. See Physical Layer Interface in Chapter 2 for detailed information. The signals that comprise this interface are:

- `pl_initial_link_width[2:0]`
- `pl_lane_reversal_mode[1:0]`
- `pl_link_gen2_capable`
- `pl_link_partner_gen2_supported`
- `pl_link_upcfg_capable`
- `pl_sel_link_rate`
- `pl_sel_link_width[1:0]`
- `pl_ltssm_state[5:0]`
- `pl_directed_link_auton`
- `pl_directed_link_change[1:0]`
- `pl_directed_link_speed`
- `pl_directed_link_width[1:0]`
- `pl_upstream_prefer_deemph`
- `pl_received_hot_rst`

# System and PCI Express Interfaces

The reference clock (sys_clk) frequency is selectable: 100 MHz, 125 MHz, or 250 MHz. The reference clock input is a single-ended signal.

# Configuration Space

- MSI-X Support: The MSI-X Capability Structure is optionally supported. MSI-X Vector Table and the Pending Bit Array need to be implemented as part of the user's logic, by claiming a BAR aperture.

- Device Serial Number Capability: Device Serial Number Capability can optionally be disabled.

- Virtual Channel Capability: Virtual Channel Capability is optionally supported. When enabled, the User Application is allowed to operate in a TCx-VC0 mode. If disabled, the user application must operate in TC0-VC0 mode.

- Vendor Specific Capability - Loopback Master: Vendor Specific Capability enables Xilinx specific PCI Express Loopback Control is optionally supported. This enables Electrical compliance testing based on Endpoint Loopback Master.

- User Implemented Configuration Space: The Virtex-6 FPGA Integrated Block optionally enables User Implemented Configuration Space, in either the Legacy PCI Configuration Space or the PCI Express Extended Configuration Space, or in both.

**★ XILINX** ®

<div align="right"><i>Appendix D</i></div>

# *Debugging Designs*

This appendix provides information on using resources available on the Xilinx Support website, available debug tools, and a step-by-step process for debugging designs that use the Virtex®-6 FPGA Integrated Block for PCI Express. This appendix uses flow diagrams to guide the user through the debug process.

The following information is found in this appendix:

- Finding Help on Xilinx.com
- Contacting Xilinx Technical Support
- Debug Tools
- Hardware Debug
- Simulation Debug

## Finding Help on Xilinx.com

To help in the design and debug process when using the Virtex-6 FPGA, the Xilinx Support webpage (www.xilinx.com/support) contains key resources such as Product documentation, Release Notes, Answer Records, and links to opening a Technical Support case.

### Documentation

The Data Sheet and User Guide are the main documents associated with the Virtex-6 FPGA Integrated Block, as shown in Table D-1.

*Table D-1:* **Virtex-6 FPGA Integrated Block for PCI Express Documentation**

| Designation | Description |
|:---:|:---|
| DS | Data Sheet: provides a high-level description of the Integrated Block and key features. It includes information on which ISE software version is supported by the current LogiCORE IP version used to instantiate the Integrated Block. |
| UG | User Guide: provides information on generating a Integrated Block design, detailed descriptions of the interface and how to use the product. The User Guide contains waveforms to show interactions with the block and other important information needed to design with the product. |

These Integrated Block for PCI Express documents along with documentation related to all products that aid in the design process can be found on the Xilinx Support webpage. Documentation is sorted by product family at the main support page or by solution at the Documentation Center.

To see the available documentation by device family:

- Navigate to www.xilinx.com/support.

- Select **Virtex-6** from the **Device List** drop-down menu.

- This will sort all available Virtex-6 FPGA documentation by Hardware Documentation, Configuration Solutions Documentation, Related Software Documentation, Tools, IP, and Data Files.

To see the available documentation by solution:

- Navigate to www.xilinx.com/support.

- Select the **Documentation** tab located at the top of the webpage.

- This is the Documentation Center where Xilinx documentation is sorted by Devices, Boards, IP, Design Tools, Doc Type, and Topic.

## Release Notes and Known Issues

Known issues for all cores, including the Virtex-6 FPGA Integrated Block for PCI Express, are described in the IP Release Notes Guide.

## Answer Records

Answer Records include information on commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a product. Answer Records are created and maintained daily ensuring users have access to the most up-to-date information on Xilinx products. Answer Records can be found by searching the Answers Database.

To use the Answers Database Search:

- Navigate to www.xilinx.com/support. The Answers Database Search is located at the top of this webpage.

- Enter keywords in the provided search field and select **Search**.

  - Examples of searchable keywords are product names, error messages, or a generic summary of the issue encountered.

  - To see all answer records directly related to the Virtex-6 FPGA Integrated Block for PCI Express, search for the phrase "Virtex-6 FPGA Integrated Block for PCI Express"

# Contacting Xilinx Technical Support

Xilinx provides premier technical support for customers encountering issues that requires additional assistance.

To contact Technical Support:

• Navigate to www.xilinx.com/support.

• Open a WebCase by selecting the WebCase link located under **Support Quick Links**.

When opening a WebCase, please include:

• Target FPGA including package and speed grade

• All applicable software versions of ISE, synthesis (if not XST), and simulator

• The XCO file created during generation of the LogiCORE IP wrapper

  • This file is located in the directory targeted for the CORE Generator software project

Additional files might be required based on the specific issue. Please see the relevant sections in this debug guide for further information on specific files to include with the WebCase.

# Debug Tools

There are many tools available to debug PCI Express design issues. It is important to know which tools would be useful for debugging for the various situations encountered. This appendix references the following tools:

## Example Design

Xilinx Endpoint for PCI Express products come with a synthesizable back-end application called the PIO design that has been tested and is proven to be interoperable in available systems. The design appropriately handles all incoming 1 DWORD read and write transactions. It returns completions for non-posted transactions and updates the target memory space for writes. For more information, see , Programmed Input/Output: Endpoint Example Design.

## ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O software cores directly into the user design. The ChipScope Pro tool allows the user to set trigger conditions to capture application and Integrated Block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information on the ChipScope Pro tool, visit www.xilinx.com/chipscope.

## Link Analyzers

Third party link analyzers show link traffic in a graphical or text format. Lecroy, Agilent, and Vmetro are companies that make common analyzers available today. These tools greatly assist in debugging link issues and allow users to capture data which Xilinx support representatives can view to assist in interpreting link behavior.

## Third Party Software Tools

This section describes third-party software tools that can be useful in debugging.

### LSPCI (Linux)

LSPCI is available on Linux platforms and allows users to view the PCI Express device configuration space. LSPCI is usually found in the `/sbin` directory. LSPCI will display a list of devices on the PCI buses in the system. See the LSPCI manual for all command options. Some useful commands for debugging include:

- `lspci -x -d [<vendor>]:[<device>]`

  This will display the first 64 bytes of configuration space in hexadecimal form for the device with vendor and device ID specified (omit the -d option to display information for all devices). The default Vendor/Device ID for Xilinx cores is 10EE:6012. Below is a sample of a read of the configuration space of a Xilinx device:

  ```
  > lspci -x -d 10EE:6012
  81:00.0 Memory controller: Xilinx Corporation: Unknown device 6012
  00: ee 10 12 60 07 00 10 00 00 80 05 10 00 00 00
  10: 00 00 80 fa 00 00 00 00 00 00 00 00 00 00 00 00
  20: 00 00 00 00 00 00 00 00 00 00 00 00 ee 10 6f 50
  30: 00 00 00 00 40 00 00 00 00 00 00 00 05 01 00 00
  ```

  Included in this section of the configuration space are the Device ID, Vendor ID, Class Code, Status and Command registers, and Base Address Registers.

- `lspci –xxxx -d [<vendor>]:[<device>]`

  This displays the extended configuration space of the device. It can be useful to read the extended configuration space on the root and look for the Advanced Error Reporting (AER) registers. These registers provide more information on why the device has flagged an error (for example, it might show that a correctable error was issued because of a replay timer time-out).

- `lspci -k`

  Shows kernel drivers handling each device and kernel modules capable of handling it (works with kernel 2.6 or later).

### PCItree (Windows)

PCItree can be downloaded at www.pcitree.de and allows the user to view the PCI Express device configuration space and perform 1 DWORD memory writes and reads to the aperture.

The configuration space is displayed by default in the lower right corner when the device is selected, as shown in Figure D-1.
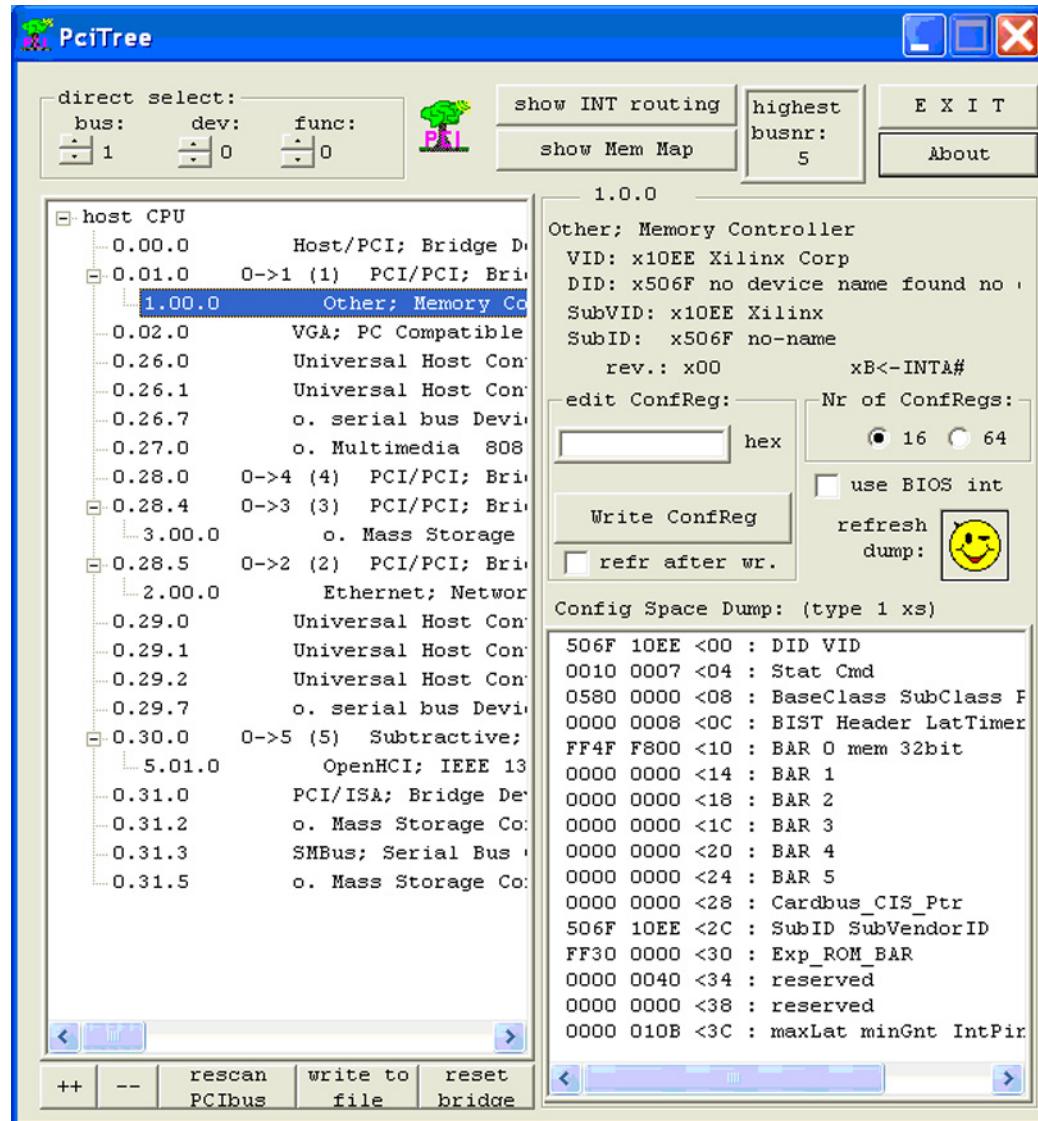


*Figure D-1:* **PCItree with Read of Configuration Space**

## HWDIRECT (Windows)

HWDIRECT can be purchased at www.eprotek.com and allows the user to view the PCI Express device configuration space as well as the extended configuration space (including the AER registers on the root).
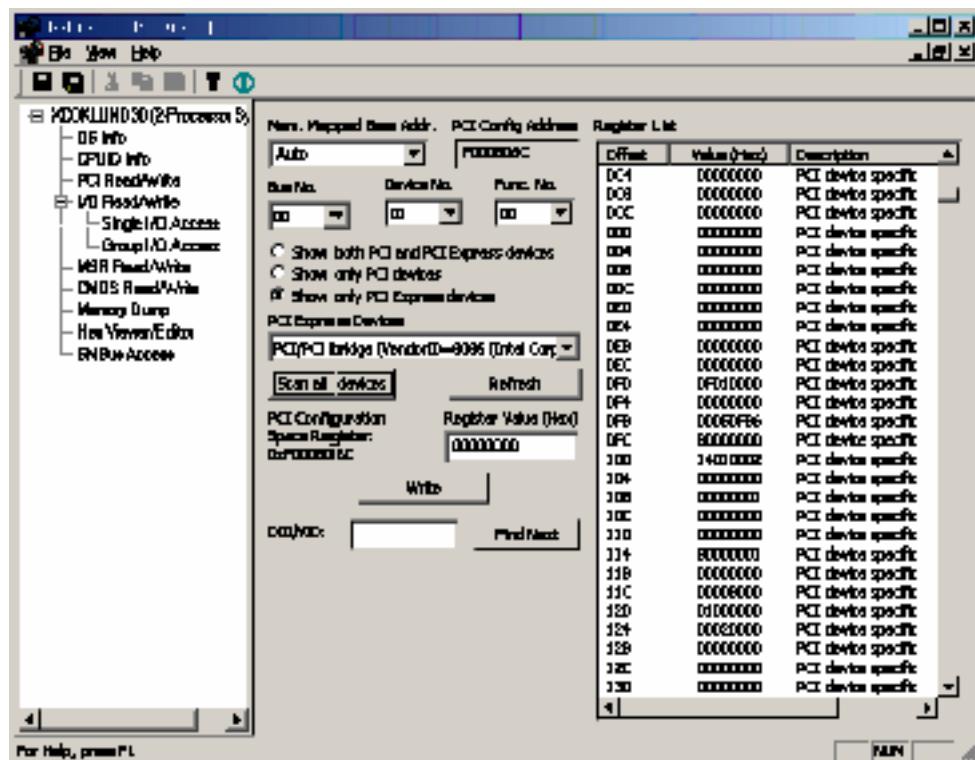


*Figure D-2:* **HWDIRECT with Read of Configuration Space**

## PCI-SIG Software Suites

PCI-SIG® software suites such as PCIE-CV can be used to test compliance with the specification. This software can be downloaded at www.pcisig.com.

# Hardware Debug

Hardware issues can range from device recognition issues to problems seen after hours of testing. This section provides debug flow diagrams for some of the most common issues experienced by users. Endpoints that are shaded gray indicate that more information can be found in sections below Figure D-3.

Using probes, an LED, ChipScope or some other method, determine if trn_lnk_up_n is asserted. When trn_lnk_up_n is low, it indicates the core has achieved link up meaning the LTSSM is in L0 state and the data link layer is in the DL_Active state.

To eliminate FGPA configuration as a root cause, perform a soft restart of the system. Performing a soft reset on the system will keep power applied and forces re-enumeration of the device.

One reason trn_reset_n stays asserted other than the system reset being asserted, is due to a faulty clock. This may keep the PLL from locking which holds trn_reset_n asserted.

Multi-lane links are susceptible to crosstalk and noise when all lanes are switching during training. A quick test for this is forcing one lane operation. This can be done by using an interposer or adapter to isolate the upper lanes or use a tape such as Scotch tape and tape off the upper lanes on the connector. If its a embedded board, remove the AC capacitors if possible to isolate the lanes.

Design Fails in Hardware

Is trn_lnk_up_n asserted? (trn_lnk_up_n = 0) — Yes → See "Link is Training Debug" section.

No

Does a soft reset fix the problem? (trn_lnk_up=0) — Yes → See "FPGA Configuration Time Debug" section.

No

Is trn_reset_n deasserted? (trn_reset_n = 1) — No → See "Clock Debug" section.

Yes

Is it a multi-lane link? — Yes → Force x1 Operation

No

Does trn_lnk_up_n = 0 when using as x1 only? — Yes → There are potentially problems with the board layout causing interference when all lanes are switching. See board debug suggestions.

Do you have a link analyzer?

No → ChipScope may be used try and determine the point of failure.

Yes → Use the link analyzer to monitor the training sequence and to determine the point of failure. Have the analyzer trigger on the first TS1 that it recognizes and then compare the output to the LTSSM state machine sequences outlined in Chapter 4 of the *PCI Express Base Specification*.

*Figure D-3:* **Design Fails in Hardware Debug Flow Diagram**

## FPGA Configuration Time Debug

Device initialization and configuration issues can be caused by not having the FPGA configured fast enough to enter link training and be recognized by the system. Section 6.6 of *PCI Express Base Specification, rev. 2.0* states two rules that might be impacted by FPGA Configuration Time:

- A component must enter the LTSSM Detect state within 20 ms of the end of the Fundamental reset.

- A system must guarantee that all components intended to be software visible at boot time are ready to receive Configuration Requests within 100 ms of the end of Conventional Reset at the Root Complex.

These statements basically mean the FPGA must be configured within a certain finite time, and not meeting these requirements could cause problems with link training and device recognition.

Configuration can be accomplished using an onboard PROM or dynamically using JTAG. When using JTAG to configure the device, configuration typically occurs after the Chipset has enumerated each peripheral. After configuring the FPGA, a soft reset is required to restart enumeration and configuration of the device. A soft reset on a Windows based PC is performed by going to **Start** → **Shut Down** and then selecting **Restart**.

To eliminate FPGA configuration as a root cause, perform a soft restart of the system. Performing a soft reset on the system will keep power applied and forces re-enumeration of the device. If the device links up and is recognized after a soft reset is performed, then FPGA configuration is most likely the problem. Most typical systems use ATX power supplies which provides some margin on this 100 ms window as the power supply is normally valid before the 100 ms window starts. For more information on FPGA configuration, see Chapter 8, FPGA Configuration.

## Link is Training Debug

PCITREE and lspci will scan the the system and display devices recognized during startup. These tools show the PCI configuration space and its settings within the device.

To eliminate FGPA configuration as a root cause, perform a soft restart of the system. Performing a soft reset on the system will keep power applied and forces re-enumeration of the device. If this fixes the problem, then it is likely the FPGA is not configured in time for the host to access the card.

The PIO design is known to work. Often, the PIO design will work when a user design will not. This usually indicates some parameter or resource conflict due to settings used for the user design configuration. It is recommended to mirror the PIO CORE Generator GUI settings into the user design. Even though the design may not function, it should still be recognized by the system.

Link is Training
(trn_lnk_up_n = 0)

Is the device recognized by the system? Can it be seen by PCITREE (Windows) or lspci (Linux)?
— Yes → See "Data Transfer Failing Debug" section.
— No ↓

Does a soft reset fix the problem? (trn_lnk_up=0)
— Yes → See "FPGA Configuration Time Debug" section.
— No ↓

Does using the PIO example design fix the problem?
— Yes → Does mirroring the PIO CORE Generator GUI settings for the user design fix the problem?
  — Yes → Check for configuration settings conflict. See the "Debugging PCI Configuration Space Parameters" section.
  — No → If the PIO design works, but mirroring the configuration parameters does not fix the problem, then attention should be focused on the user application design. See the "Application Requirements" section.
— No ↓

Do you have a link analyzer?
— No → With no link analyzer, it is possible to use ChipScope to gather the same information.
— Yes → It is likely, the problem is due to the device not responding properly to some type of access. A link analyzer allows you to view the link traffic and determine if something is incorrect. See the "Using a Link Analyzer to Debug Device Recognition Issues" section.

*Figure D-4:* **Link Trained Debug Flow Diagram**

## FPGA Configuration Time Debug

Device initialization and configuration issues can be caused by not having the FPGA configured fast enough to enter link training and be recognized by the system. Section 6.6 of *PCI Express Base Specification, rev. 2.0* states two rules that might be impacted by FPGA Configuration Time:

- A component must enter the LTSSM Detect state within 20 ms of the end of the Fundamental reset.

- A system must guarantee that all components intended to be software visible at boot time are ready to receive Configuration Requests within 100 ms of the end of Conventional Reset at the Root Complex.

These statements basically mean the FPGA must be configured within a certain finite time, and not meeting these requirements could cause problems with link training and device recognition.

Configuration can be accomplished using an onboard PROM or dynamically using JTAG. When using JTAG to configure the device, configuration typically occurs after the Chipset has enumerated each peripheral. After configuring the FPGA, a soft reset is required to restart enumeration and configuration of the device. A soft reset on a Windows based PC is performed by going to **Start** → **Shut Down** and then selecting **Restart**.

To eliminate FPGA configuration as a root cause, perform a soft restart of the system. Performing a soft reset on the system will keep power applied and forces re-enumeration of the device. If the device links up and is recognized after a soft reset is performed, then FPGA configuration is most likely the problem. Most typical systems use ATX power supplies which provides some margin on this 100 ms window as the power supply is normally valid before the 100 ms window starts. For more information on FPGA configuration, see Chapter 8, FPGA Configuration.

## Debugging PCI Configuration Space Parameters

Often, a user application will fail to be recognized by the system, but the Xilinx PIO Example design will work. In these cases, the user application is often using a PCI configuration space setting that is interfering with the system systems ability to recognize and allocate resources to the card.

Xilinx solutions for PCI Express handle all configuration transactions internally and will generate the correct responses to incoming configuration requests. Chipsets have limits as to the amount of system resources it can allocate and the core must be configured to adhere to these limitations.

The resources requested by the endpoint are identified by the BAR settings within the Endpoint configuration space. Verify that the resources requested in each BAR can be allocated by the chipset. I/O BARs are especially limited so configuring a large I/O BAR will typically prevent the chipset from configuring the device. Generate a core that implements a small amount of memory (approximately 2 KB) to identify if this is the root cause.

The Class Code setting selected in the CORE Generator software GUI can also affect configuration. The Class Code informs the Chipset as to what type of device the Endpoint is. Chipsets might expect a certain type of device to be plugged into the PCI Express slot and configuration might fail if it reads an unexpected Class Code. The BIOS could be configurable to work around this issue.

Use the PIO design with default settings to rule out any device allocation issues. The PIO design default settings have proven to work in all systems encountered when debugging problems. If the default settings allow the device to be recognized, then change the PIO design settings to match the intended user application by changing the PIO configuration the CORE Generator software GUI. Trial and error might be required to pinpoint the problem if a link analyzer is not available.

Using a link analyzer, it is possible to monitor the link traffic and possibly determine when during the enumeration and configuration process problems occur.

## Application Requirements

During enumeration, it is possible for the chipset to issue TLP traffic that will be passed from the core to the backend application. A common oversight when designing custom backend applications is to not have logic which handles every type incoming request. As a result, no response is created and problems arise. The PIO design has the necessary backend functions to respond correctly to any incoming request. It is the responsibility of the application to generate the correct response. The following packet types will be presented to the application:

- Requests targeting the Expansion ROM (if enabled)
- Message TLP's
- Memory or I/O requests targeting a BAR
- All completion packets

The PIO design, can be used to rule out any of these types of concerns, as the PIO design will respond to all incoming transactions to the user application in some way to ensure the host receives the proper response allowing the system to progress. If the PIO design works, but the custom application does not, this means that some transaction is not being handled properly.

The ChipScope tool should be implemented on the wrapper TRN Receive interface to identify if requests targeting the backend application are drained and completed successfully. The TRN interface signals that should be probed in the ChipScope tool are defined in .

## Using a Link Analyzer to Debug Device Recognition Issues

In cases where the link is up (trn_lnk_up_n = 0), but the device is not recognized by the system, a link analyzer can help solve the problem. It is likely the FPGA is not responding properly to some type of access. Use the link view to analyzer the traffic and see if anything looks out of place.

To focus on the problem, it might be necessary to try different triggers. Here are some trigger examples:

- Trigger on the first `INIT_FC1` and/or `UPDATE_FC` in either direction. This allows the analyzer to begin capture after link up.
- The first TLP normally transmitted to an endpoint is the Set Slot Power Limit Message. This usually occurs before Configuration traffic begins. This might be a good trigger point.
- Trigger on Configuration TLPs.
- Trigger on Memory Read or Memory Write TLPs.

## Data Transfer Failing Debug

The most often cause of a system freeze or hang is due to a completion timeout occurring on the host. This happens, when the host issues a non-posted transaction (usually a memory read) to the Endpoint and the Endpoint's user application does not properly respond.

If trn_lnk_up_n is toggling, it usually means the physical link is marginal. In these cases, the link may be established but may then fail once traffic begins to flow. Use ChipScope Pro or probe trn_lnk_up_n to a logic analyzer and determine if it is toggling.

Errors are reported to the user interface on the output cfg_dstatus[3:0]. This is a copy of the device status register. Using ChipScope monitor this bus for errors.

Link is Up (trn_lnk_up_n = 0) Device is recognized by system. Data Transfers failing.

Is the system freezing or hanging? — Yes → Ensure that completions are returned for all incoming Non-Posted traffic.

No

Is trn_lnk_up_n toggling? — Yes → Link could be marginal and packets are failing to pass LCRC check.

No

Fatal Error? Blue screen? Other errors? — Yes → Errors flagged by the core are due to problems on the receive data path. Use a link analyzer if possible to check incoming packets. See the "Identifying Errors" section.

No

Is the problem with receiving or transmitting TLPs?

Receive → Do incoming packets appear on TRN receive interface?

No

If read or write transactions do not appear on the trn interface, it means that most likely the incoming packet did not hit a BAR. Verify incoming TLP addresses against BAR allocation.

A memory write that misses a BAR results in a Non-Fatal error message. A non-posted transaction that misses a BAR results in a Completion with UR status.

Transmit → Do outgoing packets arrive at destination?

No

If completion packets fail to reach their destination, ensure the packet contained the correct requester ID as captured from the original Non-Posted TLP.

If other packets fail, ensure the address targeted is valid.

*Figure D-5:* **Data Transfer Debug Flow Diagram**

## Identifying Errors

Hardware symptoms of system lock up issues are indicated when the system hangs or a blue screen appears (PC systems). The *PCI Express Base Specification, rev. 2.0* requires that error detection be implemented at the receiver. A system lock up or hang is commonly the result of a Fatal Error and will be reported in bit two of the receivers Device Status register. Using the ChipScope tool, monitor the core's device status register to see if a fatal error is being reported.

A fatal error reported at the Root complex implies an issue on the transmit side of the EP. The Root Complex Device Status register can often times be seen using PCITree (Windows) or LSPCI (Linux). If a fatal error is detected, refer to the Transmit section. A Root Complex can often implement Advanced Error Reporting (AER), which further distinguishes the type of error reported. AER provides valuable information as to why a certain error was flagged and is provided as an extended capability within a devices configuration space. Section 7.10 of the *PCI Express Base Specification, rev. 2.0* provides more information on AER registers.

### Transmit

#### Fatal Error Detected on Root or Link Partner

Check to make sure the TLP is correctly formed and that the payload (if one is attached) matches what is stated in the header length field. The Endpoints device status register does not report errors created by traffic on the transmit channel.

The signals shown in Table D-2 should be monitored on the Transmit interface to verify all traffic being initiated is correct.

*Table D-2:* **TRN Transmit Interface Signals**

| Name | Direction | Description |
|------|-----------|-------------|
| trn_lnk_up_n | Output | Transaction Link Up: Active Low. Transaction link-up is asserted when the core and the connected upstream link partner port are ready and able to exchange data packets. Transaction link-up is deasserted when the core and link partner are attempting to establish communication, and when communication with the link partner is lost due to errors on the transmission channel. When the core is driven to Hot Reset and Link Disable states by the link partner, trn_lnk_up_n is deasserted and all TLP's stored in the Endpoint core are lost. |
| trn_tsof_n | Input | Transmit Start-of-Frame (SOF): Active Low. Signals the start of a packet. Valid only along with assertion of trn_tsrc_rdy_n. |
| trn_teof_n | Input | Transmit End-of-Frame (EOF): Active Low. Signals the end of a packet. Valid only along with assertion of trn_tsrc_rdy_n. |
| trn_td[63:0] | Input | Transmit Data: Packet data to be transmitted. |
| trn_trem_n | Input | Transmit Data Remainder: Valid only if trn_teof_n, trn_tsrc_rdy_n, and trn_tdst_rdy_n are all asserted. Legal values are:<br>• 0 = packet data on all of trn_td[63:0]<br>• 1 = packet data only on trn_td[63:32] |

*Table D-2:* **TRN Transmit Interface Signals** *(Cont'd)*

| Name | Direction | Description |
|------|-----------|-------------|
| trn_tsrc_rdy_n | Input | Transmit Source Ready: Active Low. Indicates that the User Application is presenting valid data on trn_td[63:0]. |
| trn_tdst_rdy_n | Output | Transmit Destination Ready: Active Low. Indicates that the core is ready to accept data on trn_td[63:0]. The simultaneous assertion of trn_tsrc_rdy_n and trn_tdst_rdy_n marks the successful transfer of one data beat on trn_td[63:0]. |

### Fatal Error Not Detected

Ensure that the address provided in the TLP header is valid. The kernel mode driver attached to the device is responsible for obtaining the system resources allocated to the device. In a Bus Mastering design, the driver is also responsible for providing the application with a valid address range. System hangs or blue screens might occur if a TLP contains an address that does not target the designated address range for that device.

## Receive

Xilinx solutions for PCI Express provide the Device Status register to the application on CFG_DSTATUS[3:0].

*Table D-3:* **Description of CFG_DSTATUS[3:0]**

| CFG_DSTATUS[3:0] | Description |
|------------------|-------------|
| CFG_DSTATUS[0] | Correctable Error Detected |
| CFG_DSTATUS[1] | Non-Fatal Error Detected |
| CFG_DSTATUS[2] | Fatal Error Detected |
| CFG_DSTATUS[3] | UR Detected |

System lock up conditions due to issues on the receive channel of the PCI Express core are often result of an error message being sent upstream to the root. Error messages are only sent when error reporting is enabled in the Device Control register.

A fatal condition is reported if any of these events occur:

- Training Error
- DLL Protocol Error
- Flow Control Protocol Error
- Malformed TLP
- Receiver Overflow

The first four bullets are not common in hardware because both Xilinx solutions for PCI Express and connected components have been thoroughly tested in simulation and hardware. However, a receiver overflow is a possibility. Users must ensure they follow requirements discussed in the section Receiver Flow Control Credits Available in Chapter 6 when issuing memory reads.

## Non-Fatal Errors

Below are lists of conditions that are reported as Non-Fatal errors. See the *PCI Express Base Specification, rev. 2.0* for more details.

If the error is being reported by the root, the Advanced Error Reporting (AER) registers can be read to determine the condition that led to the error. Use a tool such as HWDIRECT, discussed in Third Party Software Tools, page 268, to read the root's AER registers. Chapter 7 of the *PCI Express Base Specification* defines the AER registers. If the error is signaled by the endpoint, debug ports are available to help determine the specific cause of the error.

Correctable Non-Fatal errors are:

- Receiver Error
- Bad TLP
- Bad DLLP
- Replay Timeout
- Replay NUM Rollover

The first three errors listed above are detected by the receiver and are not common in hardware systems. The replay error conditions are signaled by the transmitter. If an ACK is not received for a packet within the allowed time, it will be replayed by the transmitter. Throughput can be reduced if many packets are being replayed, and the source can usually be determined by examining the link analyzer or ChipScope tool captures.

Uncorrectable Non-Fatal errors are:

- Poisoned TLP
- Received ECRC Check Failed
- Unsupported Request (UR)
- Completion Timeout
- Completer Abort
- Unexpected Completion
- ACS Violation

An unsupported request usually indicates that the address in the TLP did not fall within the address space allocated to the BAR. This often points to a problem with the address translation performed by the driver. Ensure also that the BAR has been assigned correctly by the root at startup. LSPCI or PCItree discussed in Third Party Software Tools, page 268 can be used to read the BAR values for each device.

A completion timeout indicates that no completion was returned for a transmitted TLP and is reported by the requester. This can cause the system to hang (could include a blue screen on Windows) and is usually caused when one of the devices locks up and stops responding to incoming TLPs. If the root is reporting the completion timeout, the ChipScope tool can be used to investigate why the User Application did not respond to a TLP (for example, the User Application is busy, there are no transmit buffers available, or `trn_tdst_rdy_n` is deasserted). If the endpoint is reporting the Completion timeout, a link analyzer would show the traffic patterns during the time of failure and would be useful in determining the root cause.

## Next Steps

If the debug suggestions listed above do not resolve the issue, open a support case to have the appropriate Xilinx expert assist with the issue.

To create a technical support case in Webcase, see the Xilinx website at:

www.xilinx.com/support/clearexpress/websupport.htm

Items to include when opening a case:

• Detailed description of the issue and results of the steps listed above.

• Attach ChipScope tool VCD captures taken in the steps above.

To discuss possible solutions, use the Xilinx User Community:

forums.xilinx.com/xlnx/

# Simulation Debug

This section provides simulation debug flow diagrams for some of the most common issues experienced by users. Endpoints that are shaded gray indicate that more information can be found in sections below Figure D-6.

## ModelSim Debug

SecureIP models are used to simulate the integrated block for PCI Express and the MGTs. To use these models, a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator is required.

A Verilog license is required to simulate with the SecureIP models. If the user design uses VHDL, a mixed-mode simulation license is required.

The PIO Example design should allow the user to quickly determine if the simulator is set up correctly. The default test will achieve link up (trn_lnk_up_n = 0) and issue a Configuration Read to the core's Device and VendorID.

If the libraries are not compiled and mapped correctly, it will cause errors such as:
# ** Error: (vopt-19) Failed to access library 'secureip' at "secureip".
# No such file or directory. (errno = ENOENT)
# ** Error: ../../example_design/ xilinx_pcie_2_0_ep_v6.v(820): Library secureip not found.

To model the Integrated block for PCI Express and the MGTs, the SecureIP models are used. These models must be referenced during the vsim call. Also, it is necessary to reference the unisims library and possibly xilinxcorelib depending on the design.

One of the most common mistakes in simulation of an Endpoint is forgetting to set the Memory, IO, and Bus Master Enable bits to a 1 in the PCI Command register in the configuration space.

**ModelSim Simulation Debug**

Are you using ModelSim version 6.4a or later? — No → Update ModelSim to version 6.4a or later.

Yes ↓

If using VHDL, do you have a mixed-mode simulation license? — No → Obtain a mixed-mode simulation license.

Yes ↓

Does simulating the PIO Example Design give the expected output? — Yes → See "PIO Simulator Expected Output" section.

No ↓

Do you get errors referring to failing to access library? — Yes → Need to compile and map the proper libraries. See "Compiling Simulation Libraries Section."

No ↓

Do you get errors indicating "PCIE_2_0" or other elements like "BUFG" not defined? — Yes → Add the "-L" switch with the appropriate library reference to the vsim command line. For example: -L secureip or -L unisims_ver.

No ↓

Are you able to receive packets on the TRN RX interface and transmit packets on the TRN TX interface? — Yes → If problem is more design specific, open a case with Xilinx Technical Support and include a wlf file dump of the simulation. For the best results, dump the entire design hierarchy.

No ↓

In the DSPORT test bench application, issue a Configuration Write to the PCI Command register at DWORD address offset 0x04 and set bits [2:0] to 111b.

*Figure D-6:*   **ModelSim Debug Flow Diagram**

## PIO Simulator Expected Output

The PIO design simulation should give the output as follows:

```
# Loading work.board(fast)
# Loading unisims_ver.IBUFDS_GTXE1(fast)
# Loading work.pcie_clocking_v6(fast)
# Loading unisims_ver.PCIE_2_0(fast)
# Loading work.pcie_gtx_v6(fast)
# Loading unisims_ver.GTXE1(fast)
# Loading unisims_ver.RAMB36(fast)
# Loading unisims_ver.RAMB16_S36_S36(fast)
# Loading unisims_ver.PCIE_2_0(fast__1)
# Loading work.glbl(fast)
# [                  0] board.EP.core.pcie_2_0_i.pcie_bram_i ROWS_TX 1 COLS_TX 2
# [                  0] board.EP.core.pcie_2_0_i.pcie_bram_i ROWS_RX 1 COLS_RX 2
# [                  0] board.EP.core.pcie_2_0_i.pcie_bram_i.pcie_brams_tx NUM_BRAMS 2
DOB_REG 1 WIDTH 36 RAM_WRITE_LATENCY 0 RAM_RADDR_LATENCY 0 RAM_RDATA_LATENCY 2
# [                  0] board.EP.core.pcie_2_0_i.pcie_bram_i.pcie_brams_rx NUM_BRAMS 2
DOB_REG 1 WIDTH 36 RAM_WRITE_LATENCY 0 RAM_RADDR_LATENCY 0 RAM_RDATA_LATENCY 2
# [                  0] board.RP.rport.pcie_2_0_i.pcie_bram_i ROWS_TX 1 COLS_TX 2
# [                  0] board.RP.rport.pcie_2_0_i.pcie_bram_i ROWS_RX 1 COLS_RX 2
# [                  0] board.RP.rport.pcie_2_0_i.pcie_bram_i.pcie_brams_tx NUM_BRAMS 2
DOB_REG 1 WIDTH 36 RAM_WRITE_LATENCY 0 RAM_RADDR_LATENCY 0 RAM_RDATA_LATENCY 2
# [                  0] board.RP.rport.pcie_2_0_i.pcie_bram_i.pcie_brams_rx NUM_BRAMS 2
DOB_REG 1 WIDTH 36 RAM_WRITE_LATENCY 0 RAM_RADDR_LATENCY 0 RAM_RDATA_LATENCY 2
# Running test {sample_smoke_test0}......
# [                  0] : System Reset Asserted...
# [            4995000] : System Reset De-asserted...
# [           64069100] : Transaction Reset Is De-asserted...
# [           73661100] : Transaction Link Is Up...
# [           73661100] : Expected Device/Vendor ID = 000710ee
# [           73661100] : Reading from PCI/PCI-Express Configuration Register 0x00
# [           73673000] : TSK_PARSE_FRAME on Transmit
# [           74941000] : TSK_PARSE_FRAME on Receive
# [           75273000] : TEST PASSED --- Device/Vendor ID 000710ee successfully received
# ** Note: $finish    : ../tests/sample_tests1.v(29)
#    Time: 75273 ns  Iteration: 3  Instance: /board/RP/tx_usrapp
```

## Compiling Simulation Libraries

Use the `compxlib` command to compile simulation libraries. This tool is delivered as part of the Xilinx software. For more information see the ISE Software Manuals and specifically "Development System Reference Guide" under the section titled compxlib.

Assuming the Xilinx and ModelSim environments are set up correctly, this is an example of compiling the SecureIP and UniSims libraries for Verilog into the current directory:

```
compxlib -s mti_se -arch virtex6 -l verilog -lib secureip -lib unisims
-dir ./
```

There are many other options available for compxlib described in the *Development System Reference Guide*.

Compxlib will produce a `modelsim.ini` file containing the library mappings. In ModelSim, to see the current library mappings type **vmap** at the prompt. The mappings can be updated in the ini file, or to map a library at the ModelSim prompt, type:

```
vmap [<logical_name>] [<path>]
```

For example:

```
Vmap unisims_ver C:\my_unisim_lib
```

## Next Step

If the debug suggestions listed above do not resolve the issue, a support case should be opened to have the appropriate Xilinx expert assist with the issue.

To create a technical support case in Webcase, see the Xilinx website at:

[www.xilinx.com/support/clearexpress/websupport.htm](www.xilinx.com/support/clearexpress/websupport.htm)

Items to include when opening a case:

- Detailed description of the issue and results of the steps listed above.
- Attach a VCD or WLF dump of the simulation.

To discuss possible solutions, use the Xilinx User Community:

[forums.xilinx.com/xlnx/](forums.xilinx.com/xlnx/)

*Appendix E*

# Managing Receive-Buffer Space for Inbound Completions

The PCI Express® Base Specification requires all Endpoints to advertise infinite Flow Control credits for received Completions to their link partners. This means that an Endpoint must only transmit Non-Posted Requests for which it has space to accept Completion responses. This appendix describes how a User Application can manage the receive-buffer space in the PCI Express Endpoint core to fulfill this requirement.

## General Considerations and Concepts

### Completion Space

Table E-1 defines the completion space reserved in the receive buffer by the core. The values differ depending on the different Capability Max Payload Size settings of the core and the performance level selected by the designer. If the designer chooses to not have TLP Digests (ECRC) removed from the incoming packet stream, the TLP Digests (ECRC) must be accounted for as part of the data payload. Values are credits, expressed in decimal.

*Table E-1:*   **Receiver-Buffer Completion Space**

| Capability Max Payload Size (bytes) | Performance Level: Good | | Performance Level: High | |
|---|---|---|---|---|
| | Cpl. Hdr. (Total_CplH) | Cpl. Data (Total_CplD) | Cpl. Hdr. (Total_CplH) | Cpl. Data (Total_CplD) |
| 128 | 36 | 77 | 36 | 154 |
| 156 | 36 | 77 | 36 | 154 |
| 512 | 36 | 154 | 36 | 308 |
| 1024 | 36 | 308 | 36 | 616 |

## Maximum Request Size

A Memory Read cannot request more than the value stated in Max_Request_Size, which is given by Configuration bits cfg_dcommand[14:12] as defined in Table E-2. If the User Application chooses not to read the Max_Request_Size value, it must use the default value of 128 bytes.

*Table E-2:* **Max Request Size Settings**

| cfg_dcommand[14:12] | Max_Request_Size | | | |
|---|---|---|---|---|
| | **Bytes** | **DW** | **QW** | **Credits** |
| 000b | 128 | 32 | 16 | 8 |
| 001b | 256 | 64 | 32 | 16 |
| 010b | 512 | 128 | 64 | 32 |
| 011b | 1024 | 256 | 128 | 64 |
| 100b | 2048 | 512 | 256 | 128 |
| 101b | 4096 | 1024 | 512 | 256 |
| 110b–111b | Reserved | | | |

## Read Completion Boundary

A Memory Read can be answered with multiple Completions, which when put together return all requested data. To make room for packet-header overhead, the User Application must allocate enough space for the maximum number of Completions that might be returned.

To make this process easier, the *Base Specification* quantizes the length of all Completion packets such that each must start and end on a naturally aligned Read Completion Boundary (RCB), unless it services the starting or ending address of the original request. The value of RCB is determined by Configuration bit cfg_lcommand[3] as defined in Table E-3. If the User Application chooses not to read the RCB value, it must use the default value of 64 bytes.

*Table E-3:* **Read Completion Boundary Settings**

| cfg_lcommand[3] | Read Completion Boundary | | | |
|---|---|---|---|---|
| | **Bytes** | **DW** | **QW** | **Credits** |
| 0 | 64 | 16 | 8 | 4 |
| 1 | 128 | 32 | 16 | 8 |

When calculating the number of Completion credits a Non-Posted Request requires, the user must determine how many RCB-bounded blocks the Completion response might require; this is the same as the number of Completion Header credits required.

# Methods of Managing Completion Space

A User Application can choose one of five methods to manage receive-buffer Completion space, as listed in Table E-4. For convenience, this discussion refers to these methods as LIMIT_FC, PACKET_FC, RCB_FC, DATA_FC, and STREAM_FC. Each has advantages and disadvantages that the designer needs to consider when developing the User Application.

*Table E-4:* **Managing Receive Completion Space Methods**

| Method | Description | Advantage | Disadvantage |
|---|---|---|---|
| LIMIT_FC | Limit the total number of outstanding NP Requests | Simplest method to implement in user logic | Much Completion capacity goes unused |
| PACKET_FC | Track the number of outstanding CplH and CplD credits; allocate and deallocate on a per-packet basis | Relatively simple user logic; finer allocation granularity means less wasted capacity than LIMIT_FC | As with LIMIT_FC, credits for an NP are still tied up until the Request is completely satisfied |
| RCB_FC | Track the number of outstanding CplH and CplD credits; allocate and deallocate on a per-RCB basis | Ties up credits for less time than PACKET_FC | More complex user logic than LIMIT_FC or PACKET_FC |
| DATA_FC | Track the number of outstanding CplH and CplD credits; allocate and deallocate on a per-RCB basis | Lowest amount of wasted capacity | More complex user logic than LIMIT_FC, PACKET_FC, and RCB_FC |
| STREAM_FC | Stream packets out of the core at line rate | Very high performance | User must accept and process Downstream Completion and Posted Transactions at line rate; Most complex user logic |

## LIMIT_FC Method

The LIMIT_FC method is the simplest to implement. The User Application assesses the maximum number of outstanding Non-Posted Requests allowed at one time, MAX_NP. To calculate this value, perform the following steps:

1. Determine the number of CplH credits required by a Max_Request_Size packet:

   Max_Header_Count = ceiling(Max_Request_Size / RCB)

2. Determine the greatest number of maximum-sized Completions supported by the CplD credit pool:

   Max_Packet_Count_CplD = floor(CplD / Max_Request_Size)

3.  Determine the greatest number of maximum-sized Completions supported by the CplH credit pool:

    Max_Packet_Count_CplH = floor(CplH / Max_Header_Count)

4.  Use the *smaller* of the two quantities from steps 2 and 3 to obtain the maximum number of outstanding Non-Posted requests:

    MAX_NP = min(Max_Packet_Count_CplH, Max_Packet_Count_CplD)

With knowledge of MAX_NP, the User Application can load a register NP_PENDING with zero at reset and make sure it always stays with the range 0 to MAX_NP. When a Non-Posted Request is transmitted, NP_PENDING decrements by one. When *all* Completions for an outstanding NP Request are received, NP_PENDING increments by one.

Although this method is the simplest to implement, it potentially wastes the most receiver space because an entire Max_Request_Size block of Completion credit is allocated for each Non-Posted Request, regardless of actual request size. The amount of waste becomes greater when the User Application issues a larger proportion of short Memory Reads (on the order of a single DWORD), I/O Reads and I/O Writes.

## PACKET_FC Method

The PACKET_FC method allocates blocks of credit in finer granularities than LIMIT_FC, using the receive Completion space more efficiently with a small increase in user logic.

Start with two registers, CPLH_PENDING and CPLD_PENDING, (loaded with zero at reset), and then perform these steps:

1.  When the User Application needs to send an NP request, determine the potential number of CplH and CplD credits it might require:

    NP_CplH = ceiling[((Start_Address mod RCB) + Request_Size) / RCB]

    NP_CplD = ceiling[((Start_Address mod 16 bytes) + Request_Size) /16 bytes] (except I/O Write, which returns zero data)

    The modulo and ceiling functions ensure that any fractional RCB or credit blocks are rounded up. For example, if a Memory Read requests 8 bytes of data from address `7Ch`, the returned data can potentially be returned over two Completion packets (`7Ch-7Fh`, followed by `80h-83h`). This would require two RCB blocks and two data credits.

2.  Check the following:

    CPLH_PENDING + NP_CplH $\leq$ Total_CplH (from Table E-1)

    CPLD_PENDING + NP_CplD $\leq$ Total_CplD (from Table E-1)

3.  If both inequalities are true, transmit the Non-Posted Request, increase CPLH_PENDING by NP_CplH and CPLD_PENDING by NP_CplD. For each NP Request transmitted, keep NP_CplH and NP_CplD for later use.

4.  When all Completion data is returned for an NP Request, decrement CPLH_PENDING and CPLD_PENDING accordingly.

This method is less wasteful than LIMIT_FC but still ties up all of an NP Request's Completion space until the *entire* request is satisfied. RCB_FC and DATA_FC provide finer deallocation granularity at the expense of more logic.

## RCB_FC Method

The RCB_FC method allocates and de-allocates blocks of credit in RCB granularity. Credit is freed on a per-RCB basis.

As with PACKET_FC, start with two registers, CPLH_PENDING and CPLD_PENDING (loaded with zero at reset).

1. Calculate the number of data credits per RCB:

   CplD_PER_RCB = RCB / 16 bytes

2. When the User Application needs to send an NP request, determine the potential number of CplH credits it might require. Use this to allocate CplD credits with RCB granularity:

   NP_CplH = ceiling[((Start_Address mod RCB) + Request_Size) / RCB]

   NP_CplD = NP_CplH × CplD_PER_RCB

3. Check the following:

   CPLH_PENDING + NP_CplH ≤ Total_CplH

   CPLD_PENDING + NP_CplD ≤ Total_CplD

4. If both inequalities are true, transmit the Non-Posted Request, increase CPLH_PENDING by NP_CplH and CPLD_PENDING by NP_CplD.

5. At the start of each incoming Completion, or when that Completion begins at or crosses an RCB without ending at that RCB, decrement CPLH_PENDING by 1 and CPLD_PENDING by CplD_PER_RCB. Any Completion could cross more than one RCB. The number of RCB crossings can be calculated by:

   RCB_CROSSED = ceiling[((Lower_Address mod RCB) + Length) / RCB]

   Lower_Address and Length are fields that can be parsed from the Completion header. Alternatively, a designer can load a register CUR_ADDR with Lower_Address at the start of each incoming Completion, increment per DW or QW as appropriate, then count an RCB whenever CUR_ADDR rolls over.

This method is less wasteful than PACKET_FC but still gives us an RCB granularity. If a User Application transmits I/O requests, the User Application could adopt a policy of only allocating one CplD credit for each I/O Read and zero CplD credits for each I/O Write. The User Application would have to match each incoming Completion's Tag with the Type (Memory Write, I/O Read, I/O Write) of the original NP Request.

## DATA_FC Method

The DATA_FC method provides the finest allocation granularity at the expense of logic.

As with PACKET_FC and RCB_FC, start with two registers, CPLH_PENDING and CPLD_PENDING (loaded with zero at reset).

1. When the User Application needs to send an NP request, determine the potential number of CplH and CplD credits it might require:

   NP_CplH = ceiling[((Start_Address mod RCB) + Request_Size) / RCB]

   NP_CplD = ceiling[((Start_Address mod 16 bytes) + Request_Size) / 16 bytes] (except I/O Write, which returns zero data)

2. Check the following:

   CPLH_PENDING + NP_CplH ≤ Total_CplH

$$CPLD\_PENDING + NP\_CplD \leq Total\_CplD$$

3. If both inequalities are true, transmit the Non-Posted Request, increase CPLH_PENDING by NP_CplH and CPLD_PENDING by NP_CplD.

4. At the start of each incoming Completion, or when that Completion begins at or crosses an RCB without ending at that RCB, decrement CPLH_PENDING by 1. The number of RCB crossings can be calculated by:

$$RCB\_CROSSED = ceiling[((Lower\_Address \bmod RCB) + Length) / RCB]$$

Lower_Address and Length are fields that can be parsed from the Completion header. Alternatively, a designer can load a register CUR_ADDR with Lower_Address at the start of each incoming Completion, increment per DW or QW as appropriate, then count an RCB whenever CUR_ADDR rolls over.

5. At the start of each incoming Completion, or when that Completion begins at or crosses at a naturally aligned credit boundary, decrement CPLD_PENDING by 1. The number of credit-boundary crossings is given by:

$$DATA\_CROSSED = ceiling[((Lower\_Address \bmod 16\ B) + Length) / 16\ B]$$

Alternatively, a designer can load a register CUR_ADDR with Lower_Address at the start of each incoming Completion, increment per DW or QW as appropriate, then count an RCB whenever CUR_ADDR rolls over each 16-byte address boundary.

This method is the least wasteful but requires the greatest amount of user logic. If even finer granularity is desired, the user can scale the Total_CplD value by 2 or 4 to get the number of Completion QWORDS or DWORDS, respectively, and adjust the data calculations accordingly.

## STREAM_FC Method

When configured as an Endpoint, user applications can maximize Downstream (away from Root Complex) data throughput by streaming Memory Read Transactions Upstream (towards the Root Complex) at the highest rate allowed on the Integrated Block Transaction transmit interface. Streaming Memory Reads are allowed only if trn_rdst_rdy_n can be held asserted; so that Downstream Completion Transactions, along with Posted Transactions, can be presented on the integrated block's receive Transaction interface and processed at line rate. Asserting trn_rdst_rdy_n in this manner guarantees that the Completion space within the receive buffer is not oversubscribed (that is, Receiver Overflow will not occur).

# Board Design Guidelines

## Overview

This appendix discusses topics related to implementing a PCI Express® design that uses the Virtex®-6 FPGA on a printed circuit board (PCB). Optimal performance requires an understanding of the functionality of the device pins and needs to address issues such as device interfacing, protocol specifications, and signal integrity.

Recommendations made in this appendix are guidelines and do not guarantee a working design.

The information presented here discusses PCB considerations specific to the PCI Express specifications. This appendix should be used in conjuction with the following documents for a comprehensive understanding of PCB design with Xilinx® FPGAs.

- [UG366](), *Virtex-6 FPGA GTX Transceivers User Guide* - Specifically, refer to the "Board Design Guidelines" chapter.
- [UG373](), *Virtex-6 FPGA PCB Design Guide.*

The PCI-SIG maintains multiple specifications that can apply depending on the form factor of the design. This document only considers the subset of these specifications focused on chip-to-chip and add-in card implementations. Table F-1 shows the specifications that correlate to the applicable form factors.

*Table F-1:* **PCI-SIG Specifications and Form Factor**

| Specification Name | Form-factor |
| --- | --- |
| PCI Express Base Specification Revision 2.0 | Chip-to-chip on a single PCB |
| PCI Express Card Electromechanical Specification (CEM) Revision 2.0 | ATX: desktop/server consisting of System card and Add-in card |

## Example PCB Reference

Xilinx delivers the ML605 board with an x8 PCI Express add-in card connection. This appendix uses this board as an example for certain recommendations.

For documentation such as schematics, Gerbers, and a bill-of-material for the ML605 board, see the Virtex-6 FPGA ML605 Evaluation Kit product page:

[www.xilinx.com /ml605]()

# Board Stackup

Board stackup design is dependent on many variables, including design, manufacturing, and cost constraints. See the information on board stackup design in <u>UG373</u> and <u>UG366</u>.

Generally speaking, signal layers for high speed signals such as PCI Express data signals should be sandwiched between ground planes. It is also preferable to use the layers closest to the top or bottom of the chip so that via stubs are minimized.

## ML605 Example

Figure F-1 shows the stackup that the ML605 Add-in Card reference board employs. All internal signal layers are sandwiched between (uninterrupted) ground planes. Power planes are located in the center of the stackup and are not adjacent to signal layers.
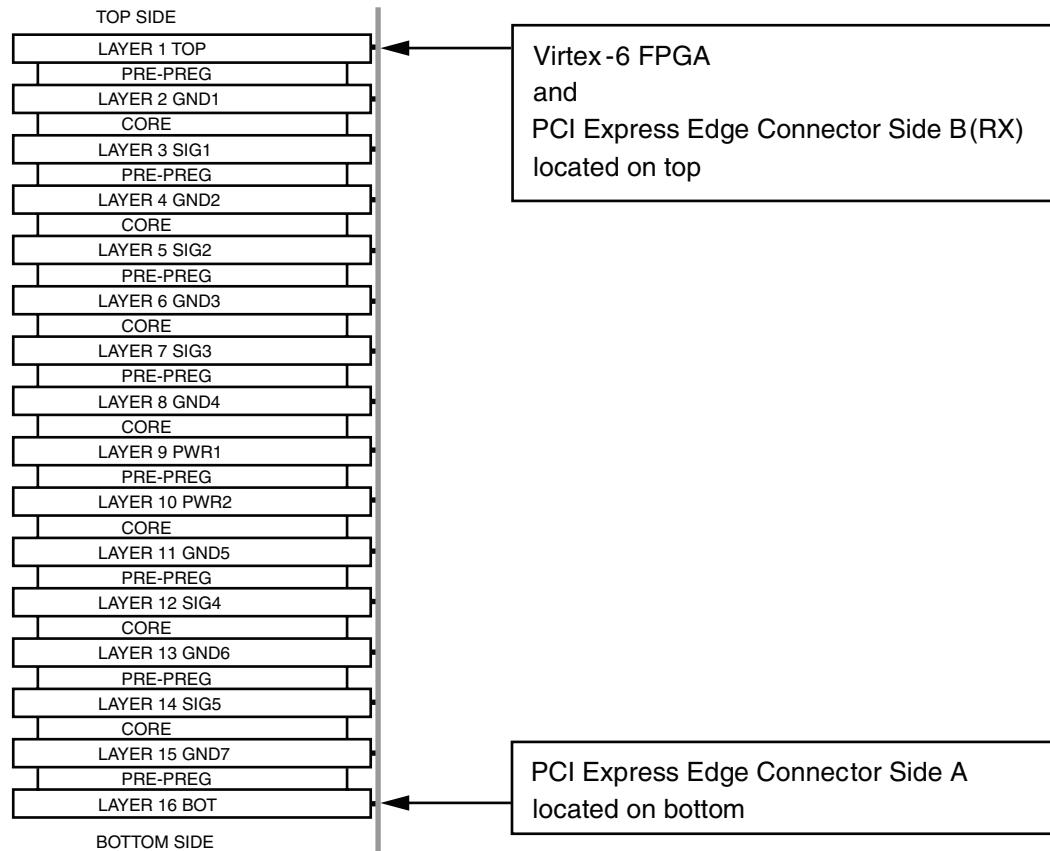


*Figure F-1:* **ML605 Board Stackup**

Transmit (TX) data lines initiate from the FPGA on the top layer, immediately drop to SIG1 (Layer 3) for routing across the PCB, and then terminate at the PCI Express edge connector side A on the bottom layer.

Receive (RX) data lines initiate from the FPGA on the top layer, immediately drop to SIG5 (Layer 14) for routing across the PCB, and then terminate at the PCI Express edge connector side B on the top layer.
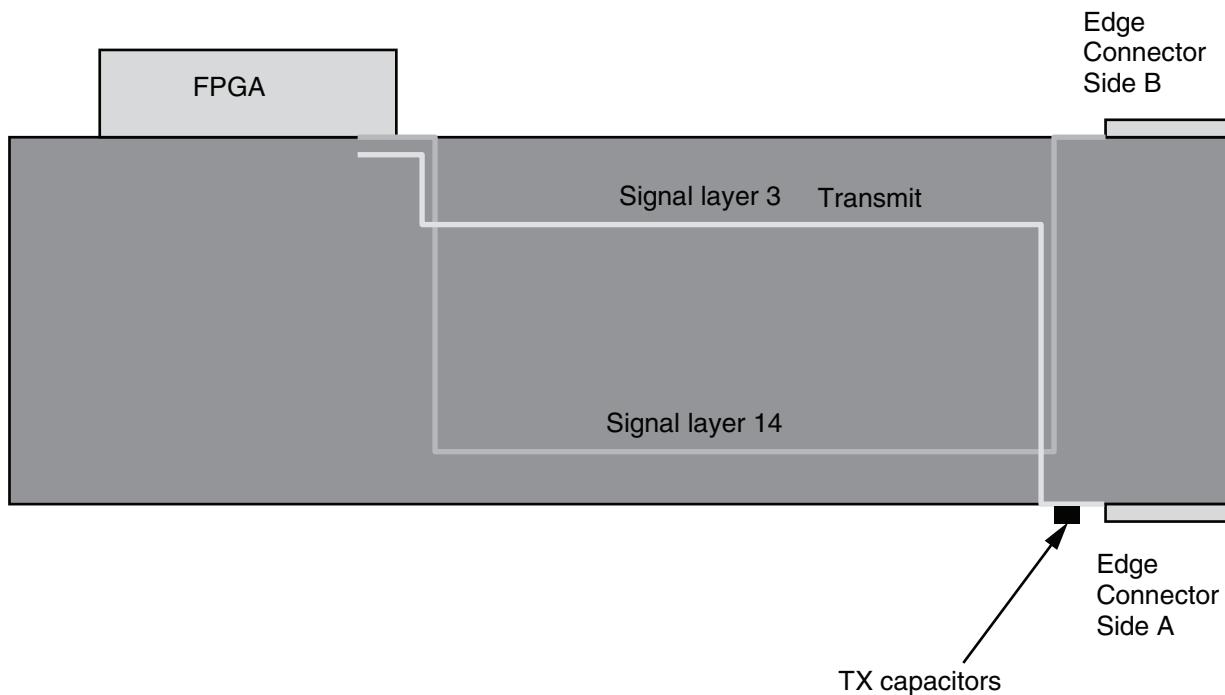
*Figure F-2:* **Transmit and Receive Data Lines**

## Power Supply Design

UG373 discusses general Power Distribution System (PDS) design for the FPGA, including the required decoupling capacitors for the VCCINT, VCCO, and VCCAUX supplies.

It is also imperative to ensure a clean power supply on MGTAVCC, and MGTAVTT power supplies. Consult UG366 for more details on GTX transceiver power supply layout and other requirements for filtering and design.

# Data Routing Guidelines

## Breakout from FPGA BGA

UG366 discusses how to break out the high-speed GTX transceiver signals from the BGA and provides examples of such. Design constraints might require microstrips for the BGA exit path or from via to the PCI Express edge connector launch or SMT pads. In such cases, the microstrip trace must be kept as short as possible.

Example Receive and Transmit breakout patterns from the ML605 board are shown in Figure F-3 and Figure F-4.
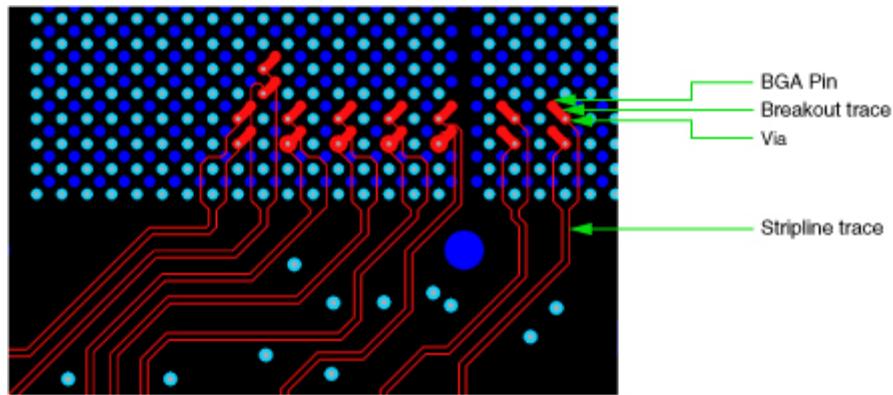


*Figure F-3:* **Receive Breakout Pattern**
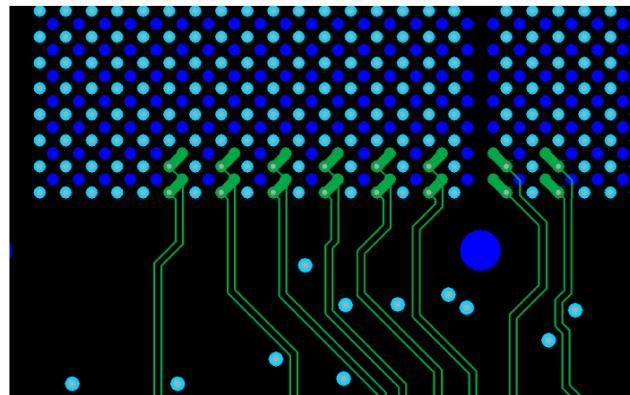


*Figure F-4:* **Transmit Breakout Pattern**

## Microstrip vs. Stripline

Striplines are to be used whenever possible, as are the uppermost and lowermost stripline layers to minimize via stubs. When the stackup is being planned, these layers should be placed as close to the top and bottom layers whenever possible.

## Plane Reference and Splits

Ground planes should be used as reference planes for signals, as opposed to noisier power planes. Each reference plane should be contiguous for the length of the trace, because routing over plane-splits creates an impedance discontinuity. In this case, the impedance of the trace changes because its coupling to the reference plane is changed abruptly at the plane split.

## Bends

Follow the recommendations in UG373 regarding microstrip and stripline bends. Tight bends (such as 90 degrees) should be avoided; only mitered, 45-degree or less, bends are recommended.

## Propagation Delay

PCI Express generally does not specify a maximum propagation delay for data signals, with the exception of add-in cards. Add-in card designs should meet the propagation delay specification in the CEM specification for data traces. The delay from the edge finger to the GTX transceiver must not exceed 750 ps.

## Lane-to-Lane Skew

Lane-to-lane skew is generally not an issue for a PCI Express link as the specification allows large amounts of skew, and GTX transceivers can handle large amounts of lane-to-lane skew. Designers should not violate the PCI Express Specifications where dictated.

The lane-to-lane skew between any two lanes in a multi-lane link should not exceed the summarized specifications in Table F-2. These specifications include PCB skew as well as any skew introduced by repeater or re-timing devices.

*Table F-2:* **Allowable Skew per the PCI-SIG Specifications**

| PCIe Specification and Form Factor | Maximum Capable Data Rate | Maximum Allowable Lane-to-Lane Skew |
|---|---|---|
| Base | 2.5 Gb/s | 18.7 ns |
| Base | 5.0 Gb/s | 6.7 ns |
| CEM: Add-in Card | 5.0 Gb/s | 0.35 ns |
| CEM: System Board | 5.0 Gb/s | 1.25 ns |

## Intrapair Skew

Intrapair skew refers to the skew between a P and N leg of a differential pair. Skew can introduce common-mode effects, which lead to increased EMI, crosstalk, and other DC effects. It is important to match the skew for differential pairs as close as possible.

Xilinx recommends intrapair trace length-matching to within 5 mils to minimize these effects.

## Symmetrical Routing

Always use symmetrical routing to prevent common-mode effects, such as EMI, from being introduced into the system.

Figure F-5 illustrates two examples of non-symmetrical routing, which should be avoided.
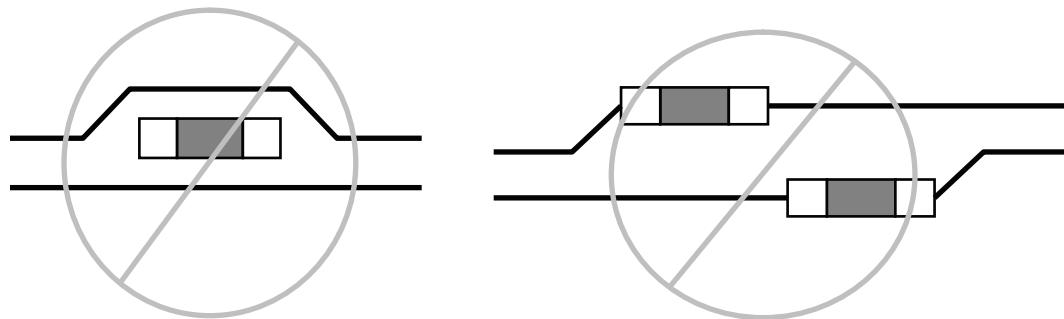


*Figure F-5:* **Non-Symmetrical Routing Examples**

## Vias

Users should follow the recommendations in UG373 for differential vias. Specifically, wherever high-speed signals must transition signal layers, a Ground-Signal-Signal-Ground (GSSG) type via should be used if possible. This will provide a low inductance return current path.

All vias for a differential pair should employ symmetrical routing rules.

## Trace Impedance

Differential data-line trace impedance was not specified in the Rev 1.0, 1.0a, or 1.1 (1.x) of the PCI Express Base and PCI Express CEM Specifications. The transmitters and receivers were specified to have 100Ω nominal differential impedance; therefore, most 1.x designs opt for a default 100Ω differential trace impedance for all PCI Express differential connections.

The *PCI Express CEM Specification Rev 2.0* now specifies a differential trace impedance for data lines that are 5.0 Gb/s capable in the range of 68Ω to 105Ω (85Ω nominal). Designers targeting PCI Express compliant add-in cards or system boards (motherboards) should adhere to this specification. Although 100Ω falls within the limits of this specification, PCB design-for-manufacturability tolerances for trace impedance are generally greater than 5%. Therefore 5.0 Gb/s add-in card designs that use 100Ω might fall above the 105Ω upper limit.

PCI Express add-in card connector vendors are now targeting 85Ω for 5.0 Gb/s capable connections; therefore, Xilinx recommends that 5.0 Gb/s capable designs for open systems target 85Ω differential impedance for data lines.

Xilinx recommends using simulation techniques to determine the optimum trace impedance. Simulation using HSPICE or Hyperlynx can help determine the optimum trace impedance to reduce signal loss.

PCB dielectric material, board stack up, microstrip, and stripline traces affect signal impedance. It is important that all of these factors are taken into consideration together.

If a simulator is not available, Xilinx recommends these basic guidelines for differential data-line trace impedance targets:

- 100Ω ± 10% for 2.5 Gb/s only links
- 85W ± 10% for 5.0 Gb/s capable links

## Trace Separation

Generally, simulation or post-layout analysis tools should be used to determine the optimum spacing required to reduce crosstalk from nearby aggressor signals. In the absence of these tools, Xilinx suggests that spacing between differential pairs and other non-PCI Express signals should be at least three times the dielectric height above the reference planes to minimize crosstalk. Exceptions to this are allowed in the break-out area of the FPGA; however, these sections should be kept as short as possible.

## Lane Reversal

Lane reversal is an optional feature of the *PCI Express Base Specification* and provides flexibility in the design of the PCB. The Virtex-6 FPGA Integrated Block for PCI Express supports lane reversal capabilities with some restrictions. See the section titled Lane Reversal in Chapter 6 for a description of these restrictions.

## Lane Polarity Inversion

The *PCI Express Base Specification* (1.x and 2.0) requires that all PCI Express receivers support polarity inversion. This gives the PCB designer flexibility to avoid having to cross P and N lines within a given differential pair.

Virtex-6 FPGA GTX receivers support lane polarity inversion on a per transceiver basis.

## AC Coupling

### System and Add-in Cards

AC coupling capacitors should be placed on the TX pairs. Place the capacitors either near the edge connector or the FPGA - not in the middle of the interconnect.

### Chip-to-Chip

AC coupling capacitors can be placed anywhere on the interconnect, except in the very middle.

### General Guidelines

Capacitors for coupled traces should always be located at the same relative place as its partner, that is, symmetrical routing guidelines apply for differential pairs.

Use 0.1 uF ceramic chip capacitors in the smallest package possible.

## Data Signal Termination

No external resistor terminators are required with the exception of a precision 100Ω resistor connected to the RCAL circuitry for the GTX column. Make sure the trace length and geometry to both legs of the resistor are equal. See UG366 for more information.

## Additional Considerations for Add-In Card Designs

1. Board thickness for add-in cards should not to exceed 0.062 inches.

2. Care must be taken when connecting the RX and TX data lines to the edge connector. The edge connector pin names for the TX and RX data lines as defined in the CEM specification are named from the view of the system board. That is, the RX (PERxx) lines are connected to the receiver on the system board and the transmitter on the add-in card. Similarly the TX (PETxx) lines are connected to the transmitter on the system board and the receiver on the add-in card. That means the add-in card should route the edge connector PERxx pins to the transmitter and the PETxx pins to the receiver on an Endpoint configured FPGA. Figure F-6 illustrates how to connect the data lines for an add-in card design.
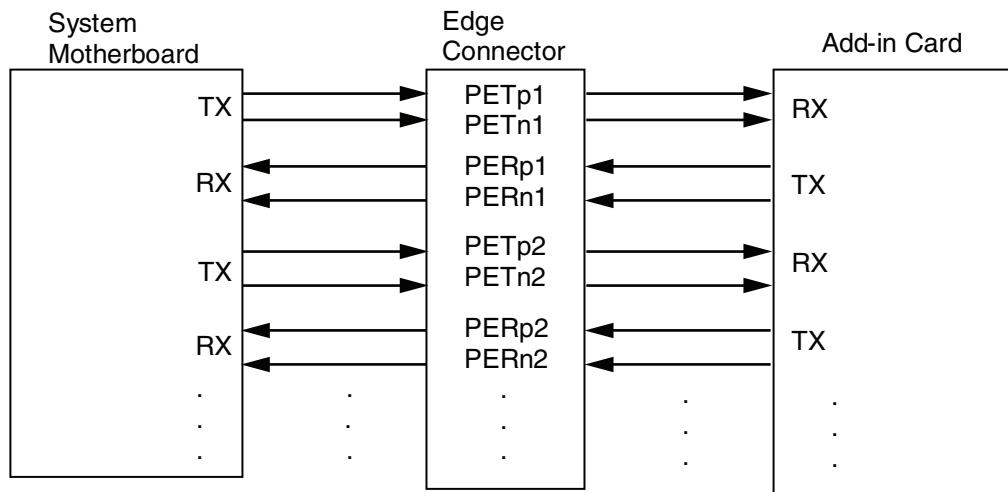


*Figure F-6:* **Add-In Card Design Connections**

# Reference Clock Considerations

## Jitter

Reference clock jitter has the potential to close both the TX and RX eyes, depending of the frequency content of the phase jitter. Therefore, it is very important to maintain as clean a reference clock as possible.

Reduce crosstalk on the REFCLK signal by isolating the clock signal from nearby high-speed traces. Maintain a separation of at least 25 mils from the nearest aggressor signals.

Ensure a clean power supply on MGTAVCC power supply. See UG366 for more details on GTX transceiver power supply layout and design.

In some cases where the designer has no control over the clock source, it might be desirable to add a jitter attenuator chip.

If an external PLL or jitter attenuator chip is used, ensure that it meets the specifications for PLL bandwidth as defined in the *PCI Express Base Specification*. The PLL bandwidth specification is different for 1.x and 2.0 versions of the specification.

## Trace Impedance

The reference clock should use a 100Ω differential trace impedance.

## Termination

The REFCLK signal should be routed to the dedicated reference clock input pins on the MGT, and the user design should instantiate an IBUFDS_GTXE1 primitive in the user design. An internal 100Ω differential termination biased to 4/5 MGTAVCC is automatically included on these input pins when the IBUFDS_GTXE1 is used, and no external termination is required or needed for Virtex-6 devices. This is true for both HSCL and LVDS clocks.

See UG366 for more information on GTX transceiver reference clock termination.

## AC Coupling

The REFCLK signal should be AC coupled at the input to the FPGA. Xilinx recommends 0.1 μF ceramic-chip capacitors for this purpose. See UG366 for more information

## Fanout

If the reference clock needs to be routed to more than one location, then a dedicated clock fanout chip should be used. Make sure to follow the specifications for the fanout chip. For instance, 100Ω termination might be required on the input to the fanout chip.

Figure F-7 shows an example of a clock fanout chip used to route the reference clock to multiple locations. The Virtex-6 FPGA requires no external resistive termination (just AC coupling capacitors). The fanout chip is shown with a single resistor terminator at its clock input pins.



UG517_aF_07_020510
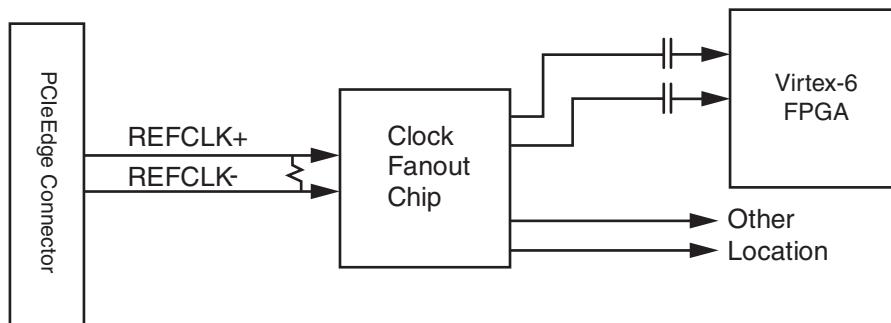
*Figure F-7:*   **Fanout Block Diagram**

## Sideband PCI Express Signals

### PERST#

The PERST# signal must be routed to the FPGA for add-in cards. This signal is a 3.3 V signal, and therefore an external circuit is necessary to interface with the Virtex-6 FPGA inputs since they do not accept this high of a voltage. This external circuit could consist of a level translator such as the ST Micro ST2378E, a resistor network, or other transistor-based circuit. There is no termination required for this signal, although the Integrated Endpoint Block core implements a pull-up on the input from within the example UCF file.

### PRSNT#

The PRSNT# pins should be connected as recommended in the CEM specification. Also see the ML605 board for an example.

## Summary Checklist

Table F-3 provides a checklist which summarizes the items discussed in this appendix.

*Table F-3:* **Board Design Checklist**

| | Item |
|---|---|
| | **Board Stackup** |
| | Follow guidelines in UG373 and UG366 |
| | **Power Supply Design** |
| | Follow guidelines in UG373 and UG366 |
| | **High-Speed Data-Signal Routing** |
| | Use stripline routing when possible |
| | Avoid routing over reference plane splits or voids |
| | Bends < 45 degrees |
| | Add-in cards must not exceed 750 ps propagation delay |
| | Follow PCI Express specification for lane-lane skew |
| | Length match intrapair skew to within 3 ps |
| | Use Ground-Signal-Signal-Ground (GSSG) type vias when possible |
| | Limit the number of vias |
| | 85Ω differential trace impedance for 5.0 Gb/s capable signals |
| | 100Ω differential trace impedance for 2.5 Gb/s only signals |
| | 20 mil trace separation between differential pairs (exception in breakout area) |
| | AC coupling 0.1 µF ceramic chip capacitors on all TX lines |
| | 100Ω precision resistor connected to the RCAL circuit for GTX transceivers (see UG366) |
| | Add-in cards must not exceed 0.062 inches in thickness |

*Table F-3:* **Board Design Checklist** *(Cont'd)*

| Item |
|------|
| **Reference Clock (REFCLK)** |
| 100Ω differential trace impedance |
| Maintain separation of at least 25 mils from nearby aggressor signals |
| Ensure clean power supply on MGTAVCC |
| No external termination required at input to FPGA (however, the user must instantiate IBUFDS_GTXE1 primitive) |
| AC coupling 0.1 μF ceramic chip capacitors |
| **Sideband signals for Add-in cards** |
| PERST# requires external circuitry to limit 3.3 V input, i.e., level translator |
| PRSNT# connect as recommended in CEM specification |

*Appendix G*

# PCIE_2_0 Port Descriptions

This appendix describes the physical interfaces visible on the Virtex®-6 FPGA Integrated Block's software primitive, PCIE_2_0.

This appendix contains these sections:

- Clock and Reset Interface
- Transaction Layer Interface
- Block RAM Interface
- GTX Transceiver Interface
- Configuration Management Interface
- Dynamic Reconfiguration Port Interface
- Debug Interface Ports
- TL2 Interface Ports

## Clock and Reset Interface

Table G-1 defines the ports in the Clock and Reset interface.

*Table G-1:* **Clock and Reset Interface Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CMRSTN | Input | USERCLK | Configuration Management reset (active Low). This input resets the PCI™ Configuration Space of the integrated block. |
| CMSTICKYRSTN | Input | USERCLK | Sticky configuration reset (active Low). This input resets the sticky registers in the PCI Configuration Space of the integrated block. |
| DLRSTN | Input | USERCLK | Data Link Layer reset (active Low). This input resets the Data Link Layer (DLL) of the integrated block. |
| FUNCLVLRSTN | Input | USERCLK | Not supported. This input must be tied High. |
| PIPECLK | Input | PIPECLK | PIPE interface clock. |
| PLRECEIVEDHOTRST | Output | PIPECLK | Received hot reset. When asserted, this output indicates an in-band hot reset has been received. |
| PLRSTN | Input | PIPECLK | Physical Layer reset (active Low). This input resets the Physical Layer of the integrated block. |

*Table G-1:* **Clock and Reset Interface Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| PLTRANSMITHOTRST | Input | PIPECLK | Transmit hot reset. When asserted, this input directs the integrated block to transmit an in-band hot reset. |
| RECEIVEDFUNCLVLRSTN | Output | USERCLK | Not supported. |
| SYSRSTN | Input | NONE | Asynchronous system reset (active Low). When this input is asserted, the integrated block is held in reset until PLL LOCK; thus it can be used to reset the integrated block. |
| TLRSTN | Input | USERCLK | Transaction Layer reset (active Low). This input resets the Transaction Layer of the integrated block. |
| USERCLK | Input | USERCLK | User interface clock. |
| USERRSTN | Output | USERCLK | User interface reset (active Low). This output should be used to reset the user design logic (it is asserted when the integrated block is reset). |

# Transaction Layer Interface

Packets are presented to and received from the integrated block's Transaction Layer through the Transaction Layer interface. Table G-2 defines the ports in the Transaction Layer interface.

*Table G-2:* **Transaction Layer Interface Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| TRNFCCPLD[11:0] | Output | USERCLK | Completion Data Flow Control Credits. This output contains the number of Completion Data FC credits for the selected flow control type. |
| TRNFCCPLH[7:0] | Output | USERCLK | Completion Header Flow Control Credits. This output contains the number of Completion Header FC credits for the selected flow control type. |
| TRNFCNPD[11:0] | Output | USERCLK | Non-Posted Data Flow Control Credits. This output contains the number of Non-Posted Data FC credits for the selected flow control type. |
| TRNFCNPH[7:0] | Output | USERCLK | Non-Posted Header Flow Control Credits. This output contains the number of Non-Posted Header FC credits for the selected flow control type. |
| TRNFCPD[11:0] | Output | USERCLK | Posted Data Flow Control Credits. This output contains the number of Posted Data FC credits for the selected flow control type. |
| TRNFCPH[7:0] | Output | USERCLK | Posted Header Flow Control Credits. This output contains the number of Posted Header FC credits for the selected flow control type. |

*Table G-2:* **Transaction Layer Interface Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| TRNFCSEL[2:0] | Input | USERCLK | Flow Control Informational Select. This input selects the type of flow control information presented on the TRNFC* signals. Valid values are:<br>`000b`: Receive buffer available space<br>`001b`: Receive credits granted to the link partner<br>`010b`: Receive credits consumed<br>`100b`: Transmit user credits available<br>`101b`: Transmit credit limit<br>`110b`: Transmit credits consumed |
| TRNLNKUPN | Output | USERCLK | Link status output (active Low). When this output is asserted, the Data Link Control and Management State Machine (DLCMSM) is in the DLACTIVE state. |
| TRNRBARHITN[6:0] | Output | USERCLK | Receive BAR Hit (active Low). This output indicates the BAR(s) targeted by the current receive transaction:<br>TRNRBARHITN[0]: BAR0<br>TRNRBARHITN[1]: BAR1<br>TRNRBARHITN[2]: BAR2<br>TRNRBARHITN[3]: BAR3<br>TRNRBARHITN[4]: BAR4<br>TRNRBARHITN[5]: BAR5<br>TRNRBARHITN[6]: Expansion ROM Address<br>If two BARs are configured into a single 64-bit address, both corresponding TRNRBARHITN bits are asserted. |
| TRNRD[63:0] | Output | USERCLK | Receive Data. This bus contains the packet data being received. |
| TRNRDSTRDYN | Input | USERCLK | Receive Destination Ready (active Low). This input is asserted to indicate that the user application is ready to accept data on TRNRD. Simultaneous assertion of TRNRSRCRDYN and TRNRDSTRDYN marks the successful transfer of data on TRNRD. |
| TRNRECRCERRN | Output | USERCLK | Not used. |
| TRNREOFN | Output | USERCLK | Receive End-of-Frame (active Low). When asserted, this output indicates the end of a packet. |
| TRNRERRFWDN | Output | USERCLK | Receive Error Forward (active Low). This output marks the current packet in progress as error-poisoned. It is asserted by the integrated block for the entire length of the packet. |
| TRNRNPOKN | Input | USERCLK | Receive Non-Posted OK (active Low). The user application asserts this input whenever it is ready to accept a Non-Posted Request packet. This allows Posted and Completion packets to bypass Non-Posted packets in the inbound queue if necessitated by the user application. When the user application approaches a state where it is unable to service Non-Posted Requests, it must deassert TRNRNPOKN one clock cycle before the integrated block presents TRNREOFN of the last Non-Posted TLP the user application can accept. |

*Table G-2:* **Transaction Layer Interface Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| TRNRREMN | Output | USERCLK | Receive Data Remainder (active Low). This output is valid only when both TRNREOFN and TRNRDSTRDYN are asserted. Valid values are:<br><br>    `0`: Packet data on all of TRNRD[63:0]<br>    `1`: Packet data only on TRNRD[63:32] |
| TRNRSOFN | Output | USERCLK | Receive Start-of-Frame (active Low). When asserted, this output indicates the start of a packet. |
| TRNRSRCDSCN | Output | USERCLK | Receive Source Discontinue (active Low). When asserted, this output indicates that the integrated block is aborting the current packet transfer. It is asserted when the physical link is going into reset. |
| TRNRSRCRDYN | Output | USERCLK | Receive Source Ready (active Low). When asserted, this output indicates that the integrated block is presenting valid data on TRNRD. |
| TRNTBUFAV[5:0] | Output | USERCLK | Transmit Buffers Available. This output provides the number of transmit buffers available in the integrated block. The maximum number is 32. Each transmit buffer can accommodate one TLP up to the supported Maximum Payload Size. |
| TRNTCFGGNTN | Input | USERCLK | Transmit Configuration Grant (active Low). The user application asserts this input in response to TRNTCFGREQN, to allow the integrated block to transmit an internally generated TLP. If the user does not need to postpone internally generated TLPs, this signal can be continuously asserted. |
| TRNTCFGREQN | Output | USERCLK | Transmit Configuration Request (active Low). This output is asserted when the integrated block is ready to transmit a Configuration Completion or other internally generated TLP. |
| TRNTD[63:0] | Input | USERCLK | Transmit Data. This bus contains the packet data to be transmitted. |
| TRNTDSTRDYN | Output | USERCLK | Transmit Destination Ready (active Low). When asserted, this output indicates that the integrated block is ready to accept data on TRNTD. Simultaneous assertion of TRNTSRCRDYN and TRNTDSTRDYN marks a successful transfer of data on TRNTD. |
| TRNTECRCGENN | Input | USERCLK | Tie-off to `1`. |
| TRNTEOFN | Input | USERCLK | Transmit End-of-Frame (active Low). This input signals the end of a packet. |
| TRNTERRDROPN | Output | USERCLK | Transmit Error Drop (active Low). When asserted, this output indicates that the integrated block discarded a packet because of a length violation or, when streaming, data was not presented on consecutive clock cycles. Length violations include packets longer than supported or packets whose payload does not match the payload advertised in the TLP header length field. |
| TRNTERRFWDN | Input | USERCLK | Transmit Error Forward (active Low). This input marks the current packet in progress as error-poisoned. It can be asserted any time between SOF and EOF, inclusive. |

*Table G-2:* **Transaction Layer Interface Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| TRNTREMN | Input | USERCLK | Transmit Data Remainder (active Low).This input is valid only when both TRNTEOFN, TRNTSRCRDYN, and TRNTDSTRDYN are asserted. Valid values are:<br><br>    0: Packet data on all of TRNTD[63:0]<br>    1: Packet data only on TRNTD[63:32] |
| TRNTSOFN | Input | USERCLK | Transmit Start-of-Frame (active Low). When asserted, this input indicates the start of a packet. |
| TRNTSRCDSCN | Input | USERCLK | Transmit Source Discontinue (active Low). When asserted, this input indicates that the user application is aborting the current packet. |
| TRNTSRCRDYN | Input | USERCLK | Transmit Source Ready (active Low). When asserted, this input indicates that the user application is presenting valid data on TRNTD. |
| TRNTSTRN | Input | USERCLK | Transmit Streamed (active Low). When asserted, this input indicates a packet will be presented on consecutive clock cycles and transmission on the link can begin before the entire packet has been written to the integrated block. |

# Block RAM Interface

The Transmit (TX) and Receive (RX) buffers are implemented with block RAM. Table G-3 defines the TX buffer and RX buffer ports for the Block RAM interface.

*Table G-3:* **Block RAM Interface Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| MIMRXRADDR[12:0] | Output | USERCLK | RX buffer read address |
| MIMRXRCE | Output | USERCLK | RX buffer output and pipeline register clock enable |
| MIMRXRDATA[67:0] | Input | USERCLK | RX buffer read data |
| MIMRXREN | Output | USERCLK | RX buffer read enable |
| MIMRXWADDR[12:0] | Output | USERCLK | RX buffer write address |
| MIMRXWDATA[67:0] | Output | USERCLK | RX buffer write data |
| MIMRXWEN | Output | USERCLK | RX buffer write enable |
| MIMTXRADDR[12:0] | Output | USERCLK | TX buffer read address |
| MIMTXRCE | Output | USERCLK | TX buffer output and pipeline register clock enable |
| MIMTXRDATA[68:0] | Input | USERCLK | TX buffer read data |
| MIMTXREN | Output | USERCLK | TX buffer read enable |
| MIMTXWADDR[12:0] | Output | USERCLK | TX buffer write address |
| MIMTXWDATA[68:0] | Output | USERCLK | TX buffer write data |
| MIMTXWEN | Output | USERCLK | TX buffer write enable |

# GTX Transceiver Interface

The GTX Transceiver interface consists of these signal groupings:

- GTX Transceiver Ports
- PIPE per Lane Ports

## GTX Transceiver Ports

Table G-4 defines the transceiver ports within the GTX Transceiver interface.

*Table G-4:* **GTX Transceiver Port Descriptions**

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| PLSELLNKRATE | Output | PIPECLK | This output reports the current link rate (driven by a separate flip-flop to control the PIPECLK BUFGMUX): <br> `0b`: 2.5 Gb/s <br> `1b`: 5.0 Gb/s |
| PLSELLNKWIDTH[1:0] | Output | PIPECLK | This output reports the current link width: <br> `00b`: x1 <br> `01b`: x2 <br> `10b`: x4 <br> `11b`: x8 |
| PLLTSSMSTATE[5:0] | Output | PIPECLK | This output shows the current LTSSM state: <br> `000000b`: Det Quiet <br> `000001b`: Det Quiet Gen2 <br> `000010b`: Det Active <br> `000011b`: Det Active Second <br> `000100b`: Pol Active <br> `000101b`: Pol config <br> `000110b`: Pol Comp Pre Send Eios <br> `000111b`: Pol Comp Pre Time-out <br> `001000b`: Pol Comp Send Pattern <br> `001001b`: Pol Comp Post Send Eios <br> `001010b`: Pol Comp Post Time-out <br> `001011b`: Cfg Lwidth St0 <br> `001100b`: Cfg Lwidth St1 <br> `001101b`: Cfg Lwidth Ac0 <br> `001110b`: Cfg Lwidth Ac1 <br> `001111b`: Cfg Lnum Wait <br> `010000b`: Cfg Lnum Acpt <br> `010001b`: Cfg Complete1 <br> `010010b`: Cfg Complete2 <br> `010011b`: Cfg Complete4 <br> `010100b`: Cfg Complete8 <br> `010101b`: Cfg Idle |

*Table G-4:* **GTX Transceiver Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| PLLTSSMSTATE[5:0] *(Cont'd)* | Output | PIPECLK | This output shows the current LTSSM state:<br>`010110b`: L0<br>`010111b`: L1 Entry0<br>`011000b`: L1 Entry1<br>`011001b`: L1 Entry2<br>`011010b`: L1 Idle<br>`011011b`: L1 Exit<br>`011100b`: Rec Rcvrlock<br>`011101b`: Rec Rcvrcfg<br>`011110b`: Rec Speed 0<br>`011111b`: Rec Speed 1<br>`100000b`: Rec Idle<br>`100001b`: Hot Rst<br>`100010b`: Disabled Entry0<br>`100011b`: Disabled Entry1<br>`100100b`: Disabled Entry2<br>`100101b`: Disabled Idle<br>`100110b`: Dp Cfg Lwidth St0<br>`100111b`: Dp Cfg Lwidth St1<br>`101000b`: Dp Cfg Lwidth St2<br>`101001b`: Dp Cfg Lwidth Ac0<br>`101010b`: Dp Cfg Lwidth Ac1<br>`101011b`: Dp Cfg Lnum Wait<br>`101100b`: Dp Cfg Lnum Acpt<br>`101101b`: To 2 Detect<br>`101110b`: Lpbk Entry0<br>`101111b`: Lpbk Entry1<br>`110000b`: Lpbk Active0<br>`110001b`: Lpbk Exit0<br>`110010b`: Lpbk Exit1<br>`110011b`: Lpbkm Entry0<br>`110100b` - `111111b`: Reserved |
| PLLANEREVERSALMODE[1:0] | Output | PIPECLK | This output shows the current Lane Reversal mode:<br>`00b`: No reversal<br>`01b`: Lanes 1:0 reversed<br>`10b`: Lanes 3:0 reversed<br>`11b`: Lanes 7:0 reversed |
| PLPHYLNKUPN | Output | PIPECLK | This active-Low output indicates the Physical Layer link up status. |

*Table G-4:* **GTX Transceiver Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| PLDIRECTEDLINKCHANGE[1:0] | Input | PIPECLK | This input directs the LTSSM to initiate a link width and/or speed change:<br><br>00b: No change<br>01b: Force link width<br>10b: Force link speed<br>11b: Force link width and speed (level-triggered) |
| PLDIRECTEDLINKWIDTH[1:0] | Input | PIPECLK | This input specifies the target link width for a directed link change operation (it is only acted on when DIRECTEDLINKCHANGE[0] is 1b):<br><br>00b: x1<br>01b: x2<br>10b: x4<br>11b: x8 |
| PLDIRECTEDLINKSPEED | Input | PIPECLK | This input specifies the target link speed for a directed link change operation (only acted on when DIRECTEDLINKCHANGE[1] is 1b):<br><br>0b: 2.5 Gb/s<br>1b: 5.0 Gb/s |
| PLDIRECTEDLINKAUTON | Input | PIPECLK | This input specifies link reliability or autonomous for directed link change operation:<br><br>0b: Link reliability<br>1b: Autonomous |
| PLTXPMSTATE[2:0] | Output | PIPECLK | This output indicates the TX power management state:<br><br>000b: TXNOTINL0S<br>001b: TXL0SENTRY<br>010b: TXL0SIDLE<br>011b: TXL0SFTS<br>100b - 111b: Reserved |
| PLRXPMSTATE[1:0] | Output | PIPECLK | This output indicates the RX power management state:<br><br>00b: RXNOTINL0S<br>01b: RXL0SENTRY<br>10b: RXL0SIDLE<br>11b: RXL0SFTS |
| PLLINKUPCFGCAP | Output | PIPECLK | When this output is High, the link is upconfigure capable (the link partner advertised upconfigure capability [symbol 4, bit 6] in the TS2s while in the Config.Complete state, and the device is upconfigure capable). |

*Table G-4:* **GTX Transceiver Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| PLLINKGEN2CAP | Output | PIPECLK | A High on this output indicates that the link is 5.0 Gb/s capable (the link partner advertised a 5.0 Gb/s data rate during the last transition from Recovery.RcvrCfg or Config.Complete to the L0 state and the device is 5.0 Gb/s capable). |
| PLLINKPARTNERGEN2SUPPORTED | Output | PIPECLK | This output is driven High if the link partner supports a 5.0 Gb/s data rate (advertised at least once the 5.0 Gb/s data rate was detected since exiting, while transitioning from Recovery.RcvrCfg or Config.Complete to the L0 state). |
| PLINITIALLINKWIDTH[2:0] | Output | PIPECLK | This output specifies the initial negotiated link width (when the first entry to Config.Idle from detect was successfully completed).<br>`000b`: Link not trained yet<br>`001b`: x1<br>`010b`: x2<br>`011b`: x4<br>`100b`: x8 |
| PLUPSTREAMPREFERDEEMPH | Input | PIPECLK | This input indicates the preferred de-emphasis of an Endpoint. This input is used only when the UPSTREAM_FACING attribute is set to TRUE.<br>`0b`: –6 dB<br>`1b`: –3.5 dB |
| PLDOWNSTREAMDEEMPHSOURCE | Input | PIPECLK | The downstream Root Port selects the de-emphasis used on the link at 5.0 Gb/s.<br>`0b`: Use Upstream Link Partner preferred de-emphasis<br>`1b`: Use the Selectable De-Emphasis value from the Link Control 2 Register (only used when the UPSTREAM_FACING attribute is set to FALSE) |
| PIPETXRCVRDET | Output | PIPECLK | When asserted, this output either initiates a receiver detection operation (in power state P1) or begins loopback (in power state P0). |
| PIPETXRESET | Output | PIPECLK | When asserted, this output resets the PCS portion of the GTX transceiver. |
| PIPETXRATE | Output | PIPECLK | This output controls the link signaling rate (connects to the GTX transceiver):<br>`0b`: Use a 2.5 Gb/s signaling rate<br>`1b`: Use a 5.0 Gb/s signaling rate |

*Table G-4:* **GTX Transceiver Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| PIPETXDEEMPH | Output | PIPECLK | This output selects the transmitter de-emphasis:<br>`0b`: –6dB de-emphasis<br>`1b`: –3.5dB de-emphasis |
| PIPETXMARGIN[2:0] | Output | PIPECLK | This output selects the transmitter voltage levels:<br>`000b`: Normal operating range<br>`001b`: 1200 mV for full swing OR 400 - 700 mV for half swing<br>`010b`: Required and vendor defined<br>`011b`: Required and vendor defined<br>`100b`: Required and 200 - 400 mV for full swing OR 100 - 200 mV for half swing if the last value or vendor defined<br>`101b`: Optional and 200 - 400 mV for full swing OR 100 - 200 mV for half swing if the last value OR vendor defined OR Reserved if no other values supported<br>`110b`: Optional and 200 - 400 mV for full swing OR 100 - 200 mV for half swing<br>`111b`: Optional and 200 - 400 mV for full swing OR 100 - 200 mV for half swing if the last value OR Reserved if no other values supported |

## PIPE per Lane Ports

Table G-5 defines the PIPE per Lane ports within the GTX Transceiver interface. There are eight copies of the PIPE per lane ports, one for each lane ($n$ = 0 to 7).

*Table G-5:* **PIPE per Lane Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| PIPERX*n*CHANISALIGNED | Input | PIPECLK | When this input is asserted, the channel is properly aligned with the master transceiver according to the observed channel bonding sequences in the data stream. |
| PIPERX*n*CHARISK[1:0] | Input | PIPECLK | This input determines the control bit(s) for received data:<br>`0b`: Data byte<br>`1b`: Control byte<br>The lower bit corresponds to the lower byte of PIPERX*n*DATA[15:0] while the upper bit describes the upper byte. |
| PIPERX*n*DATA[15:0] | Input | PIPECLK | This input provides the received data. |
| PIPERX*n*ELECIDLE | Input | PIPECLK | This asynchronous input indicates electrical idle on the RX. |

*Table G-5:* **PIPE per Lane Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| PIPERX*n*PHYSTATUS | Input | PIPECLK | This input indicates completion of GTX transceiver functions, such as Power Management state transitions and receiver detection on lane *n*. The completion is indicated by a single cycle assertion of PIPERX*n*PHYSTATUS. |
| PIPERX*n*POLARITY | Output | PIPECLK | When High, this output instructs the GTX transceiver to invert polarity (on the RX differential pair). |
| PIPERX*n*STATUS[2:0] | Input | PIPECLK | This input encodes the receiver status and error codes for the received data stream and receiver detection on lane *n*:<br><br>000b: Data received OK<br>001b: 1 SKP added<br>010b: 1 SKP removed<br>011b: Receiver Detected<br>100b: 8B/10B decode error<br>101b: Elastic Buffer overflow<br>110b: Elastic Buffer underflow<br>111b: Receive disparity error |
| PIPERX*n*VALID | Input | PIPECLK | This input indicates the presence of symbol lock and valid data on PIPERX0DATA and PIPERX0CHARISK. |
| PIPETX*n*CHARISK[1:0] | Output | PIPECLK | This output defines the control bit(s) for transmit data:<br>0b: Data byte<br>1b: Control byte<br>The lower bit corresponds to the lower byte of PIPETX*n*DATA[15:0] while the upper bit describes the upper byte. |
| PIPETX*n*COMPLIANCE | Output | PIPECLK | When asserted, this output forces the running disparity to negative. It is used only when the compliance pattern is transmitted. |
| PIPETX*n*DATA[15:0] | Output | PIPECLK | This output contains the transmit data. |
| PIPETX*n*ELECIDLE | Output | PIPECLK | This output forces the transmit output to electrical idle in all power states. |
| PIPETX*n*POWERDOWN[1:0] | Output | PIPECLK | This output is the Power Management signal for the transmitter for lane *n*:<br><br>00b: P0 (Normal operation)<br>01b: P0s (Low recovery time power-saving state)<br>10b: P1 (Longer recovery time power state)<br>11b: Reserved |

# Configuration Management Interface

The Configuration Management Interface contains these signal groupings:

- Management Interface Ports
- Error Reporting Ports
- Interrupt Generation and Status Ports
- Root Port Specific Ports
- Received Message TLP Status Ports
- Power Management Ports
- Received Configuration TLP Status Ports
- Configuration Specific Register Ports
- Miscellaneous Configuration Management Ports

## Management Interface Ports

Table G-6 defines the Management Interface ports within the Configuration Management interface. These ports are used when reading and writing the Configuration Space Registers.

*Table G-6:* **Management Interface Port Descriptions**

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGBYTEENN[3:0] | Input | USERCLK | Management Access Byte Enable (active Low). This 4-bit input provides the byte enables for the configuration register access signal. |
| CFGDI[31:0] | Input | USERCLK | Management Data In. This 32-bit data input provides write data to the configuration space inside the integrated block. |
| CFGDO[31:0] | Output | USERCLK | Management Data Out. This 32-bit data output obtains read data from the configuration space inside the integrated block. |
| CFGDWADDR[9:0] | Input | USERCLK | Management DWORD Address. This 10-bit address input provides a configuration register DWORD address during configuration register accesses. |
| CFGRDENN | Input | USERCLK | Management Read Enable (active Low). This input is the read-enable for configuration register accesses. |

*Table G-6:* **Management Interface Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGRDWRDONEN | Output | USERCLK | Management Read or Write Done (active Low). The read-write done signal indicates successful completion of the user configuration register access operation. For a user configuration register read operation, this signal validates the value of the CFGDO[31:0] data bus. |
| CFGWRENN | Input | USERCLK | Management Write Enable (active Low). This input is the write-enable for configuration register accesses. |
| CFGWRREADONLYN | Input | USERCLK | Management Write Read-only Bits (active Low). When asserted, this input indicates the current write should treat a read-only (RO) bit as a read/write (RW) bit, not including bits set by attributes, reserved bits, and bits that reflect status. This permits the user to change RO bits (the bit remains RO for link-side accesses). |
| CFGWRRW1CASRWN | Input | USERCLK | Management Write RW1C Bit As RW (active Low). When asserted, this input indicates the current write should treat any RW1C bit as a RW bit. A RW1C bit is cleared by writing a 1 to it and can normally only be set by internal integrated block conditions. The user uses this signal to set the bit to 1. |

## Error Reporting Ports

Table G-7 defines the Error Reporting ports within the Configuration Management interface.

*Table G-7:* **Error Reporting Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGERRACSN | Input | USERCLK | Configuration Error Access Control Services (ACS) Violation (active Low). The user asserts this signal to report an ACS Violation. |
| CFGERRAERHEADERLOG[127:0] | Input | USERCLK | Configuration Error AER Header Log. This 128-bit input accepts the header information for the AER Header Log from the user when an error is signaled.<br>Tie-off to 0. |
| CFGERRAERHEADERLOGSETN | Output | USERCLK | Not used. |
| CFGERRCORN | Input | USERCLK | Configuration Error Correctable Error (active Low). The user asserts this signal to report a Correctable Error. |
| CFGERRCPLABORTN | Input | USERCLK | Configuration Error Completion Aborted (active Low). The user asserts this signal to report a completion was aborted. This signal is ignored if CFGERRCPLRDYN is deasserted. |
| CFGERRCPLRDYN | Output | USERCLK | Configuration Error TLP Completion Header FIFO Ready (active Low). When this output is asserted, the internal FIFO that buffers headers from CFGERRTLPCPLHEADER[47:0] can accept entries. When this output is deasserted, CFGERRURN and CFGERRCPLABORTN are ignored by the integrated block. |
| CFGERRCPLTIMEOUTN | Input | USERCLK | Configuration Error Completion Time-out (active Low). The user asserts this signal to report a completion timed out. |
| CFGERRCPLUNEXPECTN | Input | USERCLK | Configuration Error Completion Unexpected (active Low). The user asserts this signal to report that an unexpected completion was received. |
| CFGERRECRCN | Input | USERCLK | ECRC Error Report (active Low). The user asserts this signal to report an end-to-end CRC (ECRC) error. |

*Table G-7:* **Error Reporting Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGERRLOCKEDN | Input | USERCLK | Configuration Error Locked (active Low). This input is used to further qualify the CFGERRURN or CFGERRCPLABORTN input signal. When this input is asserted concurrently with one of those two signals, it indicates that the transaction that caused the error was an MRdLk transaction and not an MRd. The integrated block generates a CplLk instead of a Cpl if the appropriate response is to send a Completion. |
| CFGERRPOSTEDN | Input | USERCLK | Configuration Error Posted (active Low). This input is used to further qualify any of the CFGERR* input signals. When this input is asserted concurrently with one of the other signals, it indicates that the transaction that caused the error was a posted transaction. |
| CFGERRTLPCPLHEADER[47:0] | Input | USERCLK | Configuration Error TLP Completion Header. This 48-bit input accepts the header information from the user when an error is signaled. This information is required so that the integrated block can issue a correct completion, if required.<br><br>This information should be extracted from the received error TLP and presented in the listed format:<br>[47:41] Lower Address<br>[40:29] Byte Count<br>[28:26] TC<br>[25:24] Attr<br>[23:8] Requester ID<br>[7:0] Tag |
| CFGERRURN | Input | USERCLK | Configuration Error Unsupported Request (active Low). The user asserts this signal to report that an Unsupported Request (UR) was received. This signal is ignored if CFGERRCPLRDYN is deasserted. |

## Interrupt Generation and Status Ports

Table G-8 defines the Interrupt Generation and Status ports within the Configuration Management interface.

*Table G-8:* **Interrupt Generation and Status Port Descriptions**

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGINTERRUPTASSERTN | Input | USERCLK | Configuration Legacy Interrupt Assert/Deassert Select. This input selects between Assert and Deassert messages for Legacy interrupts when CFGINTERRUPTN is asserted. It is not used for MSI interrupts. Value Message Type: `0b`: Assert `1b`: Deassert |
| CFGINTERRUPTDI[7:0] | Input | USERCLK | Configuration Interrupt Data In. For Message Signaling Interrupts (MSI), this input provides the portion of the Message Data that the Endpoint must drive to indicate MSI vector number, if Multi-Vector Interrupts are enabled. The value indicated by CFGINTERRUPTMMENABLE[2:0] determines the number of lower-order bits of Message Data that the Endpoint provides; the remaining upper bits of CFGINTERRUPTDI[7:0] are not used. For Single-Vector Interrupts, CFGINTERRUPTDI[7:0] is not used. For Legacy Interrupt Messages (ASSERTINTX, DEASSERTINTX), this input indicates which message type is sent, where Value Legacy Interrupt is: `00h`: INTA `01h`: INTB `02h`: INTC `03h`: INTD |
| CFGINTERRUPTDO[7:0] | Output | USERCLK | Configuration Interrupt Data Out. This output is the value of the lowest eight bits of the Message Data field in the Endpoint's MSI capability structure. This value is used in conjunction with CFGINTERRUPTMMENABLE[2:0] to drive CFGINTERRUPTDI[7:0]. |

*Table G-8:* **Interrupt Generation and Status Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGINTERRUPTMMENABLE[2:0] | Output | USERCLK | Configuration Interrupt Multiple Message Enabled. This output has the value of the Multiple Message Enable field, where values range from `000b` to `101b`. A value of `000b` indicates that single vector MSI is enabled. Other values indicate the number of bits that can be used for multi-vector MSI. |
| CFGINTERRUPTMSIENABLE | Output | USERCLK | Configuration Interrupt MSI Enabled.<br><br>`0`: Only Legacy (INTx) interrupts can be sent<br><br>`1`: The Message Signaling Interrupt (MSI) messaging is enabled |
| CFGINTERRUPTMSIXENABLE | Output | USERCLK | Configuration Interrupt MSIX Enabled. When asserted, this output indicates that the Message Signaling Interrupt (MSI-X) messaging is enabled. |
| CFGINTERRUPTMSIXFM | Output | USERCLK | Configuration Interrupt MSIX Function Mask. This output indicates the state of the Function Mask bit in the MSI-X Message Control field. |
| CFGINTERRUPTN | Input | USERCLK | Configuration Interrupt Request (active Low). When asserted, this input causes the selected interrupt message type to be transmitted by the integrated block. The signal should be asserted until CFGINTERRUPTRDYN is asserted. |
| CFGINTERRUPTRDYN | Output | USERCLK | Configuration Interrupt Ready (active Low). This output is the interrupt grant signal. The simultaneous assertion of CFGINTERRUPTRDYN and CFGINTERRUPTN indicates that the integrated block has successfully transmitted the requested interrupt message. |

## Root Port Specific Ports

Table G-9 defines the Root Port Specific ports within the Configuration Management interface.

*Table G-9:* **Root Port Specific Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGDSBUSNUMBER[7:0] | Input | USERCLK | Configuration Downstream Bus Number. This 8-bit input provides the bus number portion of the Requester ID (RID) of the Root Port, which is used in TLPs generated inside the integrated block, such as UR Completions and Power-Management messages. It does not affect TLPs presented on the TRN interface.<br><br>Tie-off to `0` for Endpoints. |
| CFGDSDEVICENUMBER[4:0] | Input | USERCLK | Configuration Downstream Device Number. This 5-bit input provides the device number portion of the RID of the Root Port, which is used in TLPs generated inside the integrated block, such as UR Completions and Power-Management messages. It does not affect TLPs presented on the TRN interface.<br><br>Tie-off to `0` for Endpoints. |
| CFGDSFUNCTIONNUMBER[2:0] | Input | USERCLK | Configuration Downstream Function Number. This 3-bit input provides the function number portion of the RID of the Root Port. This is used in TLPs generated inside the integrated block, such as UR Completions and Power-Management messages. It does not affect TLPs presented on the TRN interface.<br><br>Tie-off to `0` for Endpoints. |
| CFGPORTNUMBER[7:0] | Input | USERCLK | Configuration Root Port Number. This 8-bit input provides the port number field in the Link Capabilities Register.<br><br>Tie-off to `0` for Endpoints. |

## Received Message TLP Status Ports

Table G-10 defines the Received Message TLP Status ports within the Configuration Management interface.

*Table G-10:* **Received Message TLP Status Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGMSGDATA[15:0] | Output | USERCLK | Message RID/Set Slot Data/Bus, Device, Function Number.<br><br>Endpoint:<br>• If CFGMSGRECEIVED = 0, this output has the captured Bus/Device/Function Number of an Endpoint.<br>• If CFGMSGRECEIVED = 1 & CFGMSGRECEIVEDSETSLOT POWERLIMIT = 1, this output has the Power Value and Scale fields.<br>• If CFGMSGRECEIVED = 1 & CFGMSGRECEIVEDSETSLOT POWERLIMIT = 0, this output has the RID of the message.<br><br>Root Port:<br>• If any CFGMSGRECEIVED* signal pulses, this output has the RID of the message.<br>• Otherwise, this output is undefined. |
| CFGMSGRECEIVED | Output | USERCLK | Configuration Received a Decodable Message. This output is only asserted if a message was received on the link. It is not asserted if an upstream-moving message was generated internally by a Root Port (although the appropriate CFGMSGRECEIVEDERR* signal is asserted). |
| CFGMSGRECEIVEDASSERTINTA | Output | USERCLK | This output will pulse once for every Assert INTA Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br>Not used for Endpoints. |
| CFGMSGRECEIVEDASSERTINTB | Output | USERCLK | This output will pulse once for every Assert INTB Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br>Not used for Endpoints. |

*Table G-10:* **Received Message TLP Status Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGMSGRECEIVEDASSERTINTC | Output | USERCLK | This output will pulse once for every Assert INTC Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDASSERTINTD | Output | USERCLK | This output will pulse once for every Assert INTD Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDDEASSERTINTA | Output | USERCLK | This output will pulse once for every Deassert INTA Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDDEASSERTINTB | Output | USERCLK | This output will pulse once for every Deassert INTB Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDDEASSERTINTC | Output | USERCLK | This output will pulse once for every Deassert INTC Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDDEASSERTINTD | Output | USERCLK | This output will pulse once for every Deassert INTD Message received on the link. The Requester ID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDERRCOR | Output | USERCLK | This output will pulse once for every Correctable Error Message received on the link or generated internally by the Root Port (with the intent of having the backend logic compose a message upstream). The RID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |
| CFGMSGRECEIVEDERRFATAL | Output | USERCLK | This output will pulse once for every Fatal Error Message received on the link or generated internally by a Downstream core (with the intent of having the backend logic compose a message upstream). The RID of the message will appear on cfg_msg_data.<br><br>Not used for Endpoints. |

*Table G-10:* **Received Message TLP Status Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGMSGRECEIVEDERRNONFATAL | Output | USERCLK | This output will pulse once for every Non-Fatal Error Message received on the link or generated internally by a Downstream core (with the intent of having the backend logic compose a message upstream). The RID of the message will appear on cfg_msg_data. Not used for Endpoints. |
| CFGMSGRECEIVEDPMASNAK | Output | USERCLK | Received Power Management Active-State NAK Message. This output pulses once for every PM AS NAK Message received on the link. The RID of the message appears on CFGMSGDATA. |
| CFGMSGRECEIVEDPMETO | Output | USERCLK | Received PM Turn Off Message. This output pulses once for every PM Turn Off Message received on the link. The RID of the message appears on CFGMSGDATA. |
| CFGMSGRECEIVEDPMETOACK | Output | USERCLK | This output will pulse once for every PM Turn Off Ack Message received on the link. The RID of the message will appear on cfg_msg_data. Not used for Endpoints. |
| CFGMSGRECEIVEDPMPME | Output | USERCLK | This output will pulse once for every Power Management Event Message received on the link. The RID of the message will appear on cfg_msg_data. Not used for Endpoint. |
| CFGMSGRECEIVEDSETSLOTPOWERLIMIT | Output | USERCLK | Received Set Slot Power Limit Message. This output pulses once for every Set Slot Power Limit Message received on the link. The data of this message (Value, Scale) appears on CFGMSGDATA. |
| CFGMSGRECEIVEDUNLOCK | Output | USERCLK | Received Unlock Message. This output pulses once for every Unlock Message received on the link. The RID of the message appears on CFGMSGDATA. |

## Power Management Ports

Table G-11 defines the Power Management ports within the Configuration Management interface.

*Table G-11:* **Power Management Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGPMDIRECTASPML1N | Input | USERCLK | Tie-off to 1. |
| CFGPMRCVASREQL1N | Output | USERCLK | Not used. |
| CFGPMRCVENTERL1N | Output | USERCLK | Not used. |
| CFGPMRCVENTERL23N | Output | USERCLK | This output will pulse for every PM_Enter_L23 DLLP received. PM_Enter_L23 DLLPs are received by a Root Port after it sends a PME_Turn_Off Message. The Root Port will automatically respond; no action is required of the user. Not used for Endpoint. |
| CFGPMRCVREQACKN | Output | USERCLK | Received PMREQUESTACK DLLP (active Low). When asserted, this output indicates that a PMREQUESTACK DLLP has been received by an Endpoint after it sends a PMENTERL1, a PMENTERL23, or a PM AS Req L1. The integrated block automatically responds; no action is required of the user. |
| CFGPMSENDPMACKN | Input | USERCLK | If a Root Port user asserts this active-Low input after a CFGPMRCVASREQL1N pulse, the core composes an outgoing PM Request ACK DLLP, and the other agent enters ASPM L1. If CFGPMSENDPMNAKN is also asserted, then the NAK has priority. This is not used for PPM L1 or L3; the core sends PM Request ACKs for these cases automatically. Tie-off to 1 for Endpoint. |
| CFGPMSENDPMETON | Input | USERCLK | Asserting this active-Low input causes the Root Port to send Turn Off Message. When the link partner responds with a Turn Off Ack, this will be reported on CFGMSGRECEIVEDPMETOACK, and the final transition to L3 Ready will be reported on cfg_pcie_link_state. Tie-off to 1 for Endpoint. |

*Table G-11:* **Power Management Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGPMSENDPMNAKN | Input | USERCLK | If a Root Port user asserts this active-Low input after a CFGPMRCVASREQL1N pulse, then the core will compose an outgoing PM Active State NAK Message, and the other agent will not enter ASPM L1.<br><br>Tie-off to 1 for Endpoint. |
| CFGPMTURNOFFOKN | Input | USERCLK | Configuration Turn off OK, PPM L3 (active Low). The user application can assert the active-Low power turn-off ready signal to notify the Endpoint that it is safe for power to be turned off. This input is sampled during or after the cycle in which CFGMSGRECEIVEDPMETO pulses. |
| CFGPMWAKEN | Input | USERCLK | Send PMPME Message (active Low). A one-clock cycle assertion of this input signals the integrated block to send a Power Management Wake Event (PMPME) Message TLP to the upstream link partner. |

## Received Configuration TLP Status Ports

Table G-12 defines the Received Configuration TLP Status ports within the Configuration Management interface.

*Table G-12:* **Received Configuration TLP Status Port Descriptions (Configuration Management Interface)**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGTRANSACTION | Output | USERCLK | Configuration Transaction Received. This output pulses when a valid Config read or write is received in the range of `0 - 7Fh` (DWORD# 0 to 127). |
| CFGTRANSACTIONADDR[6:0] | Output | USERCLK | Configuration Transaction Address. This 7-bit output contains the DWORD offset that was addressed (`0 - 7Fh`). This output is valid only when CFGTRANSACTION pulses. |
| CFGTRANSACTIONTYPE | Output | USERCLK | Configuration Transaction Type. This output indicates the type of Configuration transaction when CFGTRANSACTION pulses:<br><br>`0`: Read<br>`1`: Write |

## Configuration Specific Register Ports

Table G-13 defines the Configuration Specific Register ports within the Configuration Management interface. These ports directly mirror the contents of commonly used registers located within the PCI Express Configuration Space.

*Table G-13:*   **Configuration Specific Register Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGCOMMANDBUSMASTERENABLE | Output | USERCLK | Configuration Command, Bus Master Enable, Command[2]. The integrated block takes no action based on this setting; the user logic must. <br><br> Endpoints: <br> When this output is asserted, the user logic is allowed to issue Memory or I/O Requests (including MSI/X interrupts); otherwise, the user logic must not issue those requests. <br><br> Root Ports: <br> When this output is asserted, received Memory or I/O Requests can be forwarded upstream; otherwise these requests must be handled as URs. For Non-Posted Requests, a Completion with UR completion status must be returned. |
| CFGCOMMANDINTERRUPTDISABLE | Output | USERCLK | Configuration Command, Interrupt Disable, Command[10]. When this output is asserted, the integrated block is prevented from asserting INTx interrupts. |
| CFGCOMMANDIOENABLE | Output | USERCLK | Configuration Command, I/O Space Enable, Command[0]. <br><br> Endpoints: <br> 0: The integrated block filters these accesses and responds with a UR. <br> 1: Allows the device to receive I/O Space accesses. <br><br> Root Ports: <br> 0: The user logic must not generate TLPs downstream. <br> 1: The integrated block takes no action based on this setting. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGCOMMANDMEMENABLE | Output | USERCLK | Configuration Command, Memory Space Enable, Command[1].<br><br>Endpoints:<br><br>0: The integrated block filters these accesses and responds with a UR.<br><br>1: Allows the device to receive Memory Space accesses.<br><br>Root Ports:<br><br>0: The user logic must not generate TLPs downstream.<br><br>1: The integrated block takes no action based on this setting. |
| CFGCOMMANDSERREN | Output | USERCLK | Configuration Command, SERR Enable (active Low), Command[8].<br><br>When this output is asserted, reporting of Non-fatal and Fatal errors is enabled. If enabled, errors are reported either through this bit or through the PCI Express specific bits in the Device Control Register. In addition, for a Root Port application, this bit controls transmission by the primary interface of ERRNONFATAL and ERRFATAL Error messages forwarded from the secondary interface. |
| CFGDEVCONTROL2CPLTIMEOUTDIS | Output | USERCLK | Configuration Device Control 2, Completion Time-out Disable, DEVICECTRL2[4]. When asserted, this output should cause the user to disable the Completion Time-out counters. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGDEVCONTROL2CPLTIMEOUTVAL[3:0] | Output | USERCLK | Configuration Device Control 2, Completion Time-out Value, DEVICECTRL2[3:0]. This 4-bit output is the time range that the user logic should consider a Request's pending Completion as a Completion Time-out. The integrated block takes no action based on this setting.<br><br>`0000b`: 50 µs to 50 ms (default)<br>`0001b`: 50 µs to 100 µs<br>`0010b`: 1 ms to 10 ms<br>`0101b`: 16 ms to 55 ms<br>`0110b`: 65 ms to 210 ms<br>`1001b`: 260 ms to 900 ms<br>`1010b`: 1 s to 3.5 s<br>`1101b`: 4 s to 13 s<br>`1110b`: 17 s to 64 s |
| CFGDEVCONTROLAUXPOWEREN | Output | USERCLK | Not used. |
| CFGDEVCONTROLCORRERRREPORTINGEN | Output | USERCLK | Configuration Device Control, Correctable Error Reporting Enable, DEVICECTRL[0]. This bit, in conjunction with other bits, controls sending ERRCOR messages. For a Root Port, the reporting of correctable errors is internal to the root; no external ERRCOR message is generated. |
| CFGDEVCONTROLENABLERO | Output | USERCLK | Configuration Device Control, Enable Relaxed Ordering, DEVICECTRL[4]. When this output is asserted, the user logic is permitted to set the Relaxed Ordering bit in the Attributes field of transactions it initiates that do not require strong write ordering. |
| CFGDEVCONTROLEXTTAGEN | Output | USERCLK | Configuration Device Control, Tag Field Enable, DEVICECTRL[8]. When this output is asserted, the user logic can use an 8-bit Tag field as a Requester. When this output is deasserted, the user logic is restricted to a 5-bit Tag field. The integrated block does not enforce the number of Tag bits used, either in outgoing request TLPs or incoming Completions. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGDEVCONTROLFATALERRREPORTINGEN | Output | USERCLK | Configuration Device Control, Fatal Error Reporting Enable, DEVICECTRL[2]. This bit, in conjunction with other bits, controls sending ERRFATAL messages. For a Root Port, the reporting of correctable errors is internal to the root; no external ERRFATAL message is generated. |
| CFGDEVCONTROLMAXPAYLOAD[2:0] | Output | USERCLK | Configuration Device Control, MAXPAYLOADSIZE, DEVICECTRL[7:5]. This field sets the maximum TLP payload size. As a Receiver, the user logic must handle TLPs as large as the set value. As a Transmitter, the user logic must not generate TLPs exceeding the set value.<br><br>`000b`: 128-byte maximum payload size<br>`001b`: 256-byte maximum payload size<br>`010b`: 512-byte maximum payload size<br>`011b`: 1024-byte maximum payload size |
| CFGDEVCONTROLMAXREADREQ[2:0] | Output | USERCLK | Configuration Device Control, MAXREADREQUESTSIZE, DEVICECTRL[14:12]. This field sets the maximum Read Request size for the user logic as a Requester. The user logic must not generate Read Requests with size exceeding the set value.<br><br>`000b`: 128-byte maximum Read Request size<br>`001b`: 256-byte maximum Read Request size<br>`010b`: 512-byte maximum Read Request size<br>`011b`: 1024-byte maximum Read Request size<br>`100b`: 2048-byte maximum Read Request size<br>`101b`: 4096-byte maximum Read Request size |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGDEVCONTROLNONFATALREPORTINGEN | Output | USERCLK | Configuration Device Control, Non-Fatal Error Reporting Enable, DEVICECTRL[1]. This bit, in conjunction with other bits, controls sending ERRNONFATAL messages. For a Root Port, the reporting of correctable errors is internal to the root; no external ERRNONFATAL message is generated. |
| CFGDEVCONTROLNOSNOOPEN | Output | USERCLK | Configuration Device Control, Enable No Snoop, DEVICECTRL[11]. When this output is asserted, the user logic is permitted to set the No Snoop bit in TLPs it initiates that do not require hardware-enforced cache coherency. |
| CFGDEVCONTROLPHANTOMEN | Output | USERCLK | Configuration Device Control, Phantom Functions Enable, DEVICECTRL[9]. When this output is asserted, the user logic can use unclaimed Functions as Phantom Functions to extend the number of outstanding transaction identifiers. If this output is deasserted, the user logic is not allowed to use Phantom Functions. |
| CFGDEVCONTROLURERRREPORTINGEN | Output | USERCLK | Configuration Device Control, UR Reporting Enable, DEVICECTRL[3]. This bit, in conjunction with other bits, controls the signaling of URs by sending Error messages. |
| CFGDEVSTATUSCORRERRDETECTED | Output | USERCLK | Configuration Device Status, Correctable Error Detected, DEVICESTATUS[0]. This output indicates the status of correctable errors detected. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control Register. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGDEVSTATUSFATALERRDETECTED | Output | USERCLK | Configuration Device Status, Fatal Error Detected, DEVICESTATUS[2]. This output indicates the status of Fatal errors detected. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control Register. |
| CFGDEVSTATUSNONFATALERRDETECTED | Output | USERCLK | Configuration Device Status, Non-Fatal Error Detected, DEVICESTATUS[1]. This output indicates the status of Non-fatal errors detected. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control Register. |
| CFGDEVSTATUSURDETECTED | Output | USERCLK | Configuration Device Status, Unsupported Request Detected, DEVICESTATUS[3]. This output indicates that the integrated block received a UR. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control Register. |
| CFGLINKCONTROLASPMCONTROL[1:0] | Output | USERCLK | Configuration Link Control, ASPM Control, LINKCTRL[1:0]. This 2-bit output indicates the level of ASPM supported, where:<br>`00b`: Disabled<br>`01b`: L0s Entry Enabled<br>`10b`: Not used<br>`11b`: Not used |
| CFGLINKCONTROLAUTOBANDWIDTHINTEN | Output | USERCLK | Configuration Link Control, Link Autonomous Bandwidth Interrupt Enable, LINKCTRL[11]. When asserted active-Low, this bit enables the generation of an interrupt to indicate that the Link Autonomous Bandwidth Status bit has been set. The core will take no action based on the setting of this bit; user logic must create the interrupt.<br>Not used for Endpoint. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGLINKCONTROLBANDWIDTHINTEN | Output | USERCLK | Configuration Link Control, Link Bandwidth Management Interrupt Enable, LINKCTRL[10]. When asserted, active-Low, enables the generation of an interrupt to indicate that the Link Bandwidth Management Status bit has been set. The core will take no action based on the setting of this bit; user logic must create the interrupt. <br><br> Not used for Endpoint. |
| CFGLINKCONTROLCLOCKPMEN | Output | USERCLK | Configuration Link Control, Enable Clock Power Management, LINKCTRL[8]. For Endpoints that support a CLKREQ# mechanism: <br><br> 0b: Clock power management disabled <br><br> 1b: The device is permitted to use CLKREQ# <br><br> The integrated block takes no action based on the setting of this bit; this function must be implemented in external logic. |
| CFGLINKCONTROLCOMMONCLOCK | Output | USERCLK | Configuration Link Control, Common Clock Configuration, LINKCTRL[6]. When this output is asserted, this component and the component at the opposite end of this Link are operating with a distributed common reference clock. When this output is deasserted, the components are operating with an asynchronous reference clock. |
| CFGLINKCONTROLEXTENDEDSYNC | Output | USERCLK | Configuration Link Control, Extended Synch, LINKCTRL[7]. When this output is asserted, the transmission of additional ordered sets is forced when exiting the L0s state and when in the Recovery state. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGLINKCONTROLHWAUTOWIDTHDIS | Output | USERCLK | Configuration Link Control, Hardware Autonomous Width Disable, LINKCTRL[9]. When this output is asserted, the integrated block is disabled from changing the Link width for reasons other than attempting to correct an unreliable Link operation by reducing the Link width. |
| CFGLINKCONTROLLINKDISABLE | Output | USERCLK | Configuration Link Control, Link Disable, LINKCTRL[4]. When this output is asserted, indicates the Link is disabled and directs the LTSSM to the Disabled state. Not used for Endpoint. |
| CFGLINKCONTROLRCB | Output | USERCLK | Configuration Link Control, RCB, LINKCTRL[3]. This output indicates the Read Completion Boundary value, where: 0: 64B  1: 128B |
| CFGLINKCONTROLRETRAINLINK | Output | USERCLK | Configuration Link Control, Retrain Link, LINKCTRL[5]. A write of 1b to this bit to the Root Port Type 1 configuration space initiates Link retraining by directing the Physical Layer LTSSM to the Recovery state. Configuration Reads of this bit are always 0, but this signal will pulse for one cycle when a 1 is written to it. Not used for Endpoint. |
| CFGLINKSTATUSAUTOBANDWIDTHSTATUS | Output | USERCLK | Configuration Link Status, Link Autonomous Bandwidth Status, LINKSTATUS[15]. Indicates the core has autonomously changed Link speed or width, without the Port transitioning through DL_Down status, for reasons other than to attempt to correct unreliable Link operation. This bit must be set if the Physical Layer reports a speed or width change was initiated by the Downstream component that was indicated as an autonomous change. Not used for Endpoint. |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| CFGLINKSTATUSBANDWITHSTATUS | Output | USERCLK | Configuration Link Status, Link Bandwidth Management Status, LINKSTATUS[14]. This output indicates that either of the following has occurred without the Port transitioning through DL_Down status:<br><br>• A Link retraining has completed following a write of 1b to the Retrain Link bit. Note: This bit is Set following any write of 1b to the Retrain Link bit, including when the Link is in the process of retraining for some other reason.<br><br>• Hardware has changed Link speed or width to attempt to correct unreliable Link operation, either through an LTSSM timeout or a higher level process. This bit is set if the Physical Layer reports a speed or width change was initiated by the Downstream component that was not indicated as an autonomous change.<br><br>Not used for Endpoint. |
| CFGLINKSTATUSCURRENTSPEED[1:0] | Output | USERCLK | Configuration Link Status, Current Link Speed, LINKSTATUS[1:0]. This field indicates the negotiated Link speed of the given PCI Express Link:<br><br>`01b`: 2.5 Gb/s PCI Express Link<br>`10b`: 5.0 Gb/s PCI Express Link |
| CFGLINKSTATUSDLLACTIVE | Output | USERCLK | Not used. |
| CFGLINKSTATUSLINKTRAINING | Output | USERCLK | Not used. |
| CFGLINKSTATUSNEGOTIATEDWIDTH[3:0] | Output | USERCLK | Configuration Link Status, Negotiated Link Width, LINKSTATUS[7:4]. This output indicates the negotiated width of the given PCI Express Link (only widths up to x8 are displayed).<br><br>`0001b`: x1<br>`0010b`: x2<br>`0100b`: x4<br>`1000b`: x8 |

*Table G-13:* **Configuration Specific Register Port Descriptions** *(Cont'd)*

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGSLOTCONTROLELECTROMECHILCTLPULSE | Output | USERCLK | Not used. |
| CFGTRNPENDINGN | Input | USERCLK | User Transaction Pending (active Low). When asserted, this input sets the Transactions Pending bit in the Device Status Register (DEVICESTATUS[5]). **Note:** The user is required to assert this input if the User Application has not received a completion to a request. |

## Miscellaneous Configuration Management Ports

Table G-14 defines the Miscellaneous Configuration Management ports within the Configuration Management interface.

*Table G-14:* **Miscellaneous Configuration Management Port Descriptions**

| Port | Direction | Clock Domain | Description |
|---|---|---|---|
| CFGAERECRCCHECKEN | Output | USERCLK | Not used. |
| CFGAERECRCGENEN | Output | USERCLK | Not used. |
| CFGDSN[63:0] | Input | USERCLK | Configuration Device Serial Number. This 64-bit input indicates the value that should be transferred to the Device Serial Number Capability. Bits [31:0] are transferred to the first (Lower) DWORD (byte offset `0x4` of the Capability), and bits [63:32] are transferred to the second (Upper) DWORD (byte offset `0x8` of the Capability). |
| CFGPCIELINKSTATE[2:0] | Output | USERCLK | PCI Express Link State. This encoded bus reports the PCIe Link State Information to the user: `000b`: L0 state `001b`: PPM L1 state `010b`: PPM L2/L3Ready state `011b`: PMPME state `100b`: In or transitioning to/from the ASPM L0s state `101b`: Transitioning to/from the PPM L1 state `110b`: Transitioning to the PPM L2/L3Ready state `111b`: In or transitioning to/from the ASPM L1 state |

# Dynamic Reconfiguration Port Interface

Table G-15 describes the Dynamic Reconfiguration Port (DRP) ports.

*Table G-15:* **DRP Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| DRPCLK | Input | | DRP clock input |
| DRPDADDR[8:0] | Input | DRPCLK | DRP address bus |
| DRPDEN | Input | DRPCLK | DRP transaction enable |
| DRPDI[15:0] | Input | DRPCLK | DRP input data bus |
| DRPDO[15:0] | Output | DRPCLK | DRP data out |
| DRPDRDY | Output | DRPCLK | DRP transaction done |
| DRPDWE | Input | DRPCLK | DRP write enable |

# Debug Interface Ports

Table G-16 describes the Debug Interface ports.

*Table G-16:* **Debug Interface Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| DBGMODE[1:0] | Input | USERCLK | Debug Mode |
| DBGSCLRA | Output | USERCLK | Debug Scalar |
| DBGSCLRB | Output | USERCLK | Debug Scalar |
| DBGSCLRC | Output | USERCLK | Debug Scalar |
| DBGSCLRD | Output | USERCLK | Debug Scalar |
| DBGSCLRE | Output | USERCLK | Debug Scalar |
| DBGSCLRF | Output | USERCLK | Debug Scalar |
| DBGSCLRG | Output | USERCLK | Debug Scalar |
| DBGSCLRH | Output | USERCLK | Debug Scalar |
| DBGSCLRI | Output | USERCLK | Debug Scalar |
| DBGSCLRJ | Output | USERCLK | Debug Scalar |
| DBGSCLRK | Output | USERCLK | Debug Scalar |
| DBGSUBMODE | Input | USERCLK | Debug Sub Mode |
| DBGVECA[63:0] | Output | USERCLK | Debug Vector |
| DBGVECB[63:0] | Output | USERCLK | Debug Vector |
| DBGVECC[11:0] | Output | USERCLK | Debug Vector |
| PLDBGMODE[2:0] | Input | PIPECLK | PL Debug Mode |
| PLDBGVEC[11:0] | Output | PIPECLK | PL Debug Vector |

# TL2 Interface Ports

The TL2 interface is unused but documented for completeness (see Table G-17).

*Table G-17:* **TL2 Interface Port Descriptions**

| Port | Direction | Clock Domain | Description |
|------|-----------|--------------|-------------|
| LL2BADDLLPERRN | Output | USERCLK | Not used. |
| LL2BADTLPERRN | Output | USERCLK | Not used. |
| LL2PROTOCOLERRN | Output | USERCLK | Not used. |
| LL2REPLAYROERRN | Output | USERCLK | Not used. |
| LL2REPLAYTOERRN | Output | USERCLK | Not used. |
| LL2SENDASREQL1N | Input | USERCLK | Tie-off to 1. |
| LL2SENDENTERL1N | Input | USERCLK | Tie-off to 1. |
| LL2SENDENTERL23N | Input | USERCLK | Tie-off to 1. |
| LL2SUSPENDNOWN | Input | USERCLK | Tie-off to 1. |
| LL2SUSPENDOKN | Output | USERCLK | Not used. |
| LL2TFCINIT1SEQN | Output | USERCLK | Not used. |
| LL2TFCINIT2SEQN | Output | USERCLK | Not used. |
| LL2TLPRCVN | Input | USERCLK | Tie-off to 1. |
| PL2DIRECTEDLSTATE[4:0] | Input | USERCLK | Tie-off to 0. |
| PL2LINKUPN | Output | USERCLK | Not used. |
| PL2RECEIVERERRN | Output | USERCLK | Not used. |
| PL2RECOVERYN | Output | USERCLK | Not used. |
| PL2RXELECIDLE | Output | USERCLK | Not used. |
| PL2SUSPENDOK | Output | USERCLK | Not used. |
| TL2ASPMSUSPENDCREDITCHECKN | Input | USERCLK | Tie-off to 1. |
| TL2ASPMSUSPENDCREDITCHECKOKN | Output | USERCLK | Not used. |
| TL2ASPMSUSPENDREQN | Output | USERCLK | Not used. |
| TL2PPMSUSPENDOKN | Output | USERCLK | Not used. |
| TL2PPMSUSPENDREQN | Input | USERCLK | Tie-off to 1. |
| TRNRDLLPDATA[31:0] | Output | USERCLK | Not used. |
| TRNRDLLPSRCRDYN | Output | USERCLK | Not used. |
| TRNTDLLPDATA[31:0] | Input | USERCLK | Tie-off to 0. |
| TRNTDLLPDSTRDYN | Output | USERCLK | Not used. |
| TRNTDLLPSRCRDYN | Input | USERCLK | Tie-off to 1. |

# *PCIE_2_0 Attribute Descriptions*

Table H-1 defines the attributes on the PCIE_2_0 library primitive for the Virtex®-6 FPGA Integrated Block for PCI Express® designs. All attributes are set in the LogiCORE™ IP; they are documented in this appendix for reference. Users should not change the attribute settings as set in the LogiCORE IP GUI, because such changes might result in a faulty or non-compliant design.

*Table H-1:* **PCIE_2_0 Attributes**

| Attribute Name | Type | Description |
|---|---|---|
| AER_BASE_PTR | 12-bit Hex | Not supported. This attribute must be set to `128h`. |
| AER_CAP_ECRC_CHECK_CAPABLE | Boolean | Not supported. This attribute must be set to FALSE. |
| AER_CAP_ECRC_GEN_CAPABLE | Boolean | Not supported. This attribute must be set to FALSE. |
| AER_CAP_ID | 16-bit Hex | Not supported. This attribute must be set to `01h`. |
| AER_CAP_INT_MSG_NUM_MSI | 5-bit Hex | Not supported. This attribute must be set to `0Ah`. |
| AER_CAP_INT_MSG_NUM_MSIX | 5-bit Hex | Not supported. This attribute must be set to `15h`. |
| AER_CAP_NEXTPTR | 12-bit Hex | Not supported. This attribute must be set to `160h`. |
| AER_CAP_ON | Boolean | Not supported. This attribute must be set to FALSE. |
| AER_CAP_PERMIT_ROOTERR_UPDATE | Boolean | Not supported. This attribute must be set to TRUE. |
| AER_CAP_VERSION | 4-bit Hex | Not supported. This attribute must be set to `1h`. |
| ALLOW_X8_GEN2 | Boolean | Set to TRUE for x8 Gen 2 operation. Set to FALSE for all other lane width and data rate configurations. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| BAR0 | 32-bit Hex | This attribute specifies the mask/settings for Base Address Register (BAR) 0. If BAR is not to be implemented, this attribute is set to `32'h00000000`. Bits are defined as follows:<br><br>• Memory Space BAR:<br><br>0: Mem Space Indicator (set to `0`)<br><br>[2:1]: Type field (`10` for 64-bit, `00` for 32-bit)<br><br>3: Prefetchable (`0` or `1`)<br><br>[31:4]: Mask for writable bits of BAR. For a 32-bit BAR, the uppermost $31:n$ bits are set to `1`, where $2^n$ = memory aperture size in bytes. For a 64-bit BAR, the uppermost $63:n$ bits of {BAR1, BAR0} are set to `1`.<br><br>• I/O Space BAR:<br><br>0: I/O Space Indicator (set to `1`)<br><br>1: Reserved (set to `0`)<br><br>[31:2]: Mask for writable bits of BAR. The uppermost $31:n$ bits are set to `1`, where $2^n$ = I/O aperture size in bytes |
| BAR1 | 32-bit Hex | This attribute specifies the mask/settings for BAR1 if BAR0 is a 32-bit BAR, or the upper bits of {BAR1, BAR0} if BAR0 is a 64-bit BAR. If BAR is not to be implemented, this attribute is set to `32'h00000000`. See the BAR0 description if this attribute functions as the upper bits of a 64-bit BAR. Bits are defined as follows:<br><br>• Memory Space BAR (not the upper bits of BAR0):<br><br>0: Mem Space Indicator (set to `0`)<br><br>[2:1]: Type field (`10` for 64-bit, `00` for 32-bit)<br><br>3: Prefetchable (`0` or `1`)<br><br>[31:4]: Mask for writable bits of BAR. For a 32-bit BAR, the uppermost $31:n$ bits are set to `1`, where $2^n$ = memory aperture size in bytes. For a 64-bit BAR, the uppermost $63:n$ bits of {BAR2, BAR1} are set to `1`.<br><br>• I/O Space BAR:<br><br>0: I/O Space Indicator (set to `1`)<br><br>1: Reserved (set to `0`)<br><br>[31:2]: Mask for writable bits of BAR. The uppermost $31:n$ bits are set to `1`, where $2^n$ = I/O aperture size in bytes |

| Attribute Name | Type | Description |
|---|---|---|
| BAR2 | 32-bit Hex | For an Endpoint, this attribute specifies the mask/settings for BAR2 if BAR1 is a 32-bit BAR, or the upper bits of {BAR2, BAR1} if BAR1 is the lower part of a 64-bit BAR. If BAR is not to be implemented, this attribute is set to `32'h00000000`. See the BAR1 description if this attribute functions as the upper bits of a 64-bit BAR. |
| | | For a Root, this attribute must be set to `32'h00FFFFFF`. |
| | | For an Endpoint, bits are defined as follows: |
| | | • Memory Space BAR (not upper bits of BAR1): |
| | | 0: Mem Space Indicator (set to `0`) |
| | | [2:1]: Type field (`10` for 64-bit, `00` for 32-bit) |
| | | 3: Prefetchable (`0` or `1`) |
| | | [31:4]: Mask for writable bits of BAR. For a 32-bit BAR, the uppermost 31:$n$ bits are set to `1`, where $2^n$ = memory aperture size in bytes. For a 64-bit BAR, the uppermost 63:$n$ bits of {BAR3, BAR2} are set to `1`. |
| | | • I/O Space BAR: |
| | | 0: I/O Space Indicator (set to `1`) |
| | | 1: Reserved (set to `0`) |
| | | [31:2]: Mask for writable bits of BAR. The uppermost 31:$n$ bits are set to `1`, where $2^n$ = I/O aperture size in bytes |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| BAR3 | 32-bit Hex | For an Endpoint, this attribute specifies the mask/settings for BAR3 if BAR2 is a 32-bit BAR, or the upper bits of {BAR3, BAR2} if BAR2 is the lower part of a 64-bit BAR. If BAR is not to be implemented, this attribute is set to `32'h00000000`. See the BAR2 description if this functions as the upper bits of a 64-bit BAR.<br><br>For a Root, this attribute must be set to:<br><br>`32'hFFFF0000`: I/O Limit/Base Registers not implemented<br><br>`32'hFFFFF0F0`: I/O Limit/Base Registers use 16-bit decode<br><br>`32'hFFFFF1F1`: I/O Limit/Base Registers use 32-bit decode<br><br>For an Endpoint, bits are defined as follows:<br><br>• Memory Space BAR (not upper bits of BAR2):<br><br>0: Mem Space Indicator (set to `0`)<br><br>[2:1]: Type field (`10` for 64-bit, `00` for 32-bit)<br><br>3: Prefetchable (`0` or `1`)<br><br>[31:4]: Mask for writable bits of BAR. For a 32-bit BAR, the uppermost 31:$n$ bits are set to `1`, where $2^n$ = memory aperture size in bytes. For a 64-bit BAR, the uppermost 63:$n$ bits of {BAR4, BAR3} are set to `1`.<br><br>• I/O Space BAR:<br><br>0: I/O Space Indicator (set to `1`)<br><br>1: Reserved (set to `0`)<br><br>[31:2]: Mask for writable bits of BAR. The uppermost 31:$n$ bits are set to `1`, where $2^n$ = I/O aperture size in bytes |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| BAR4 | 32-bit Hex | For an Endpoint, this attribute specifies mask/settings for Base Address Register (BAR) 4 if BAR3 is a 32-bit BAR, or the upper bits of {BAR4, BAR3}, if BAR3 is the lower part of a 64-bit BAR. If BAR is not to be implemented, this attribute is set to `32'h00000000`. See the BAR3 description if this functions as the upper bits of a 64-bit BAR.<br><br>For a Root, this attribute must be set to `32'hFFF0FFF0`.<br><br>For an Endpoint, bits are defined as follows:<br>• Memory Space BAR (not upper bits of BAR3):<br>  0: Mem Space Indicator (set to `0`)<br>  [2:1]: Type field (`10` for 64-bit, `00` for 32-bit)<br>  3: Prefetchable (`0` or `1`)<br>  [31:4]: Mask for writable bits of BAR. For a 32-bit BAR, the uppermost 31:$n$ bits are set to `1`, where $2^n$ = memory aperture size in bytes. For a 64-bit BAR, the uppermost 63:$n$ bits of {BAR5, BAR4} to `1`.<br>• I/O Space BAR:<br>  0: I/O Space Indicator (set to `1`)<br>  1: Reserved (set to `0`)<br>  [31:2]: Mask for writable bits of BAR. The uppermost 31:$n$ bits are set to `1`, where $2^n$ = I/O aperture size in bytes. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| BAR5 | 32-bit Hex | For an Endpoint, this attribute specifies mask/settings for BAR5 if BAR4 is a 32-bit BAR or the upper bits of {BAR5, BAR4} if BAR4 is the lower part of a 64-bit BAR. If BAR is not to be implemented, this attribute is set to `32'h00000000`. See the BAR4 description if this functions as the upper bits of a 64-bit BAR.<br><br>For a Root, this attribute must be set to:<br>• `32'h00000000`: Memory Limit/Base Registers not implemented<br>• `32'hFFF1FFF1`: Prefetchable Memory Limit/Base Registers implemented<br><br>For an Endpoint, bits are defined as follows:<br>• Memory Space BAR (not upper bits of BAR4):<br>  0: Mem Space Indicator (set to `0`)<br>  [2:1]: Type field (`00` for 32-bit; BAR5 cannot be the lower part of a 64-bit BAR)<br>  3: Prefetchable (`0` or `1`)<br>  [31:4]: Mask for writable bits of BAR. The uppermost 31:$n$ bits are set to `1`, where $2^n$ = memory aperture size in bytes<br>• I/O Space BAR:<br>  0: I/O Space Indicator (set to `1`)<br>  1: Reserved (set to `0`)<br>  [31:2]: Mask for writable bits of BAR. The uppermost 31:$n$ bits are set to `1`, where $2^n$ = I/O aperture size in bytes |
| CAPABILITIES_PTR | 8-bit Hex | Pointer to the first capabilities structure location. This value is transferred to the Capabilities Pointer Register at offset `0x34`. |
| CARDBUS_CIS_POINTER | 32-bit Hex | Pointer to the Cardbus data structure. This value is transferred to the Cardbus CIS Pointer Register. It is set to `0` if the Cardbus pointer is not implemented. |
| CLASS_CODE | 24-bit Hex | Code identifying basic function, subclass, and applicable programming interface. This value is transferred to the Class Code Register. |
| CMD_INTX_IMPLEMENTED | Boolean | INTX Interrupt Generation Capable. If this attribute is FALSE, Command[10] is hardwired to `0`. |
| CPL_TIMEOUT_DISABLE_SUPPORTED | Boolean | When this attribute is TRUE, Completion Time-out Disable is supported. This value is required to be TRUE for Endpoints and either setting allowed for Root Ports. This attribute drives Device Capability 2 [4]. |

| Attribute Name | Type | Description |
|---|---|---|
| CPL_TIMEOUT_RANGES_SUPPORTED | 4-bit Hex | Supported range of completion time-outs. This attribute drives Device Capability 2 Register, bits [3:0]. |
| CRM_MODULE_RSTS | 7-bit Hex | Reserved. This attribute must be set to `00h`. |
| DEVICE_ID | 16-bit Hex | Unique Device ID. This value is transferred to the Device ID Register. |
| DEV_CAP_ENABLE_SLOT_PWR_LIMIT_SCALE | Boolean | This attribute permits captured Slot Power Limit Scale Messages to program the corresponding Device Capabilities Scale field (Endpoints only). If set to FALSE, this field is hardwired to `0`. If set to TRUE, this field is hardwired to `1`. |
| DEV_CAP_ENABLE_SLOT_PWR_LIMIT_VALUE | Boolean | This attribute permits captured Slot Power Limit Scale Messages to program the corresponding Device Capabilities Value field (Endpoints only). If set to FALSE, this field is hardwired to `0`. If set to TRUE, this field is hardwired to `1`. |
| DEV_CAP_ENDPOINT_L0S_LATENCY | Integer | Endpoint L0s Acceptable Latency. This attribute records the latency that the Endpoint can withstand on transitions from the L0s state to the L0 state. Valid settings are:<br><br>0: Less than 64 ns<br>1: 64 to 128 ns<br>2: 128 to 256 ns<br>3: 256 to 512 ns<br>4: 512 ns to 1 µs<br>5: 1 to 2 µs<br>6: 2 to 4 µs<br>7: More than 4 µs<br><br>For Endpoints only. This attribute must be set to 0 for other devices. |
| DEV_CAP_ENDPOINT_L1_LATENCY | Integer | Endpoint L1 Acceptable Latency. Records the latency that the endpoint can withstand on transitions from the L1 state to the L0 state (if the L1 state is supported). Valid settings are:<br><br>0: Less than 1 µs<br>1: 1 to 2 µs<br>2: 2 to 4 µs<br>3: 4 to 8 µs<br>4: 8 to 16 µs<br>5: 16 to 32 µs<br>6: 32 to 64 µs<br>7: More than 64 µs<br><br>For Endpoints only. This attribute must be set to 0 for other devices. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| DEV_CAP_EXT_TAG_SUPPORTED | Boolean | Extended Tags support.<br><br>FALSE: 5-bit tag<br><br>TRUE: 8-bit tag |
| DEV_CAP_FUNCTION_LEVEL_RESET_CAPABLE | Boolean | Must be set to FALSE. |
| DEV_CAP_MAX_PAYLOAD_SUPPORTED | Integer | This attribute specifies the maximum payload supported. Valid settings are:<br><br>0: 128 bytes<br><br>1: 256 bytes<br><br>2: 512 bytes<br><br>3: 1024 bytes<br><br>4: 2048 bytes (not supported)<br><br>5: 4096 bytes (not supported)<br><br>This value is transferred to the Device Capabilities Register. |
| DEV_CAP_PHANTOM_FUNCTIONS_SUPPORT | Integer | Phantom Function Support. This attribute indicates the number of functions re-allocated as Tag bits. Valid settings are: 0, 1, 2, 3. |
| DEV_CAP_ROLE_BASED_ERROR | Boolean | When this attribute is set to TRUE, compliant error reporting is supported. |
| DEV_CAP_RSVD_14_12 | Integer | Reserved bits [14:12] in the Device Capability Register. This attribute must be set to 0. |
| DEV_CAP_RSVD_17_16 | Integer | Reserved bits [17:16] in the Device Capability Register. This attribute must be set to 0. |
| DEV_CAP_RSVD_31_29 | Integer | Reserved bits [31:29] in the Device Capability Register. This attribute must be set to 0. |
| DEV_CONTROL_AUX_POWER_SUPPORTED | Boolean | When this attribute is set to TRUE, Device Control[10] is writable. |
| DISABLE_ASPM_L1_TIMER | Boolean | When this attribute is set to TRUE, the internal timer that causes an Endpoint to enter into ASPM L1 is disabled. |
| DISABLE_BAR_FILTERING | Boolean | When this attribute is set to TRUE, BAR filtering is disabled. This setting does not change the behavior of the BAR hit outputs. |
| DISABLE_ID_CHECK | Boolean | When this attribute is set to TRUE, checking for Requester ID of received completions is disabled. |
| DISABLE_LANE_REVERSAL | Boolean | When this attribute is set to FALSE, the device is capable of lane reversal. When this attribute is set to TRUE, lane reversal is disabled. |
| DISABLE_RX_TC_FILTER | Boolean | When this attribute is set to TRUE, TC filtering of received TLPs is disabled. |
| DISABLE_SCRAMBLING | Boolean | When this attribute is TRUE, Scrambling of transmit data is turned off. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| DNSTREAM_LINK_NUM | 8-bit Hex | Used in Root Port mode only. This attribute specifies the link number that this device advertises in TS1 and TS2 during link training.<br>This attribute must be set to `00h` for Endpoints. |
| DSN_BASE_PTR | 12-bit Hex | Byte address of the base of the Device Serial Number (DSN) Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address.<br>This attribute must be set to `100h`. |
| DSN_CAP_ID | 16-bit Hex | Capability ID for DSN capability.<br>This attribute must be set to `0003h`. |
| DSN_CAP_NEXTPTR | 12-bit Hex | Device Serial Number's Capability's Next Capability Offset pointer to the next item in the capabilities list, or `000h` if this is the final capability.<br>Valid settings are `000h`, `10Ch`, and `128h`. |
| DSN_CAP_ON | Boolean | This attribute indicates if the DSN structure exists. If this is FALSE, then the DSN structure cannot be accessed via either the link or the Configuration Management Interface. |
| DSN_CAP_VERSION | 4-bit Hex | This attribute indicates the Device Serial Number structure version number.<br>This attribute must be set to `1h`. |
| ENABLE_MSG_ROUTE | 11-bit Hex | This attribute enables the routing of message TLPs to the user through the TRN RX interface. A bit value of 1 enables routing of the message TLP to the user. Messages are always decoded by the message decoder.<br>    Bit 0: ERR COR<br>    Bit 1: ERR NONFATAL<br>    Bit 2: ERR FATAL<br>    Bit 3: INTA<br>    Bit 4: INTB<br>    Bit 5: INTC<br>    Bit 6: INTD<br>    Bit 7: PM_PME<br>    Bit 8: PME_TO_ACK<br>    Bit 9: Unlock<br>    Bit 10: PME_Turn_Off |
| ENABLE_RX_TD_ECRC_TRIM | Boolean | When this attribute is set to TRUE, received TLPs have their td bit set to `0` and the ECRC is removed. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| ENTER_RVRY_EI_L0 | Boolean | When this attribute is set to TRUE, the device enters recovery from the L0 state on one of two conditions:<br>• EI inference (if INFER_EI[0] is set to `1b`) or<br>• Analog electrical idle (if INFER_EI[0] is set to `0b`) without receiving EIOS<br>This attribute must be set to TRUE. |
| EXPANSION_ROM | 32-bit Hex | This attribute specifies the mask/settings for the Expansion ROM BAR. If the BAR is not to be implemented, this attribute is set to `32'h00000000`. Bits are defined as follows:<br>0: Expansion ROM implemented (set to `1` to implement ROM)<br>[10:1]: Reserved (set to `0`)<br>[31:11]: Mask for writable bits of BAR. The uppermost 31:n bits are set to `1`, where $2^n$ = ROM aperture size in bytes |
| EXT_CFG_CAP_PTR | 6-bit Hex | This attribute is used to initialize the pointer to user-defined configuration capabilities within the PCI™ compatible configuration space. Legacy configuration requests are routed with word addresses greater than or equal to the value to the user. When this attribute is set to `00h`, all legacy requests are routed to the user. When this attribute is set to `3Fh`, no Legacy configuration requests are routed to the user. |
| EXT_CFG_XP_CAP_PTR | 10-bit Hex | This attribute is used to initialize pointers to user-defined configuration capabilities within the PCI Express Extended configuration space. Extended configuration requests are routed with word addresses greater than or equal to the value to the user. When this attribute is set to `3FFh`, no extended configuration requests are routed to the user. |
| HEADER_TYPE | 8-bit Hex | This attribute specifies the values to be transferred to the Header Type Register. Bit 7 should be set to `0` indicating a single-function device. Bit 0 identifies the header as Type 0 or Type 1, with `0` indicating a Type 0 header. This attribute must be set to `00h` for Endpoints.<br>Valid settings are:<br>`00h`: Single-function, Type 0 header<br>`01h`: Single-function, Type 1 header |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| INFER_EI | 5-bit Hex | This attribute directs the Physical Layer to add the inferred electrical idle behavior to the specified LTSSM state when the respective bit is set to 1b.<br><br>Bit 0 for L0<br><br>Bit 1 for Recovery.RcvrCfg<br><br>Bit 2 for Recovery.Speed when successful_speed_negotiation = 1b<br><br>Bit 3 for Recovery.Speed when successful_speed_negotiation = 0b<br><br>Bit 4 for Loopback.Active (as slave) |
| INTERRUPT_PIN | 8-bit Hex | This attribute indicates the mapping for legacy interrupt messages. Valid settings are:<br><br>0: No legacy interrupt messages used<br><br>1: INTA<br><br>2: INTB<br><br>3: INTC<br><br>4: INTD |
| IS_SWITCH | Boolean | Not supported. This attribute must be set to FALSE. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LAST_CONFIG_DWORD | 10-bit Hex | Used to indicate which DWORD address (byte offset/4) is the last Configuration DWORD implemented in the integrated block in PCI Express Extended Configuration Space. Any Configuration transaction received by the integrated block and routed to the CMM (not the user) that is above this address causes a UR to be generated. All capability structures in PCI Express Extended Configuration Space should be mapped back-to-back (starting at byte offset `0x100`), so there are no gaps in between them. The valid settings for LAST_CONFIG_DWORD are: <br><br> 1. If EXT_CFG_XP_CAP_PTR is not active (= `0x3FF`), then it must be set as follows: <br><br>    a. If there are no PCI Express Extended Capabilities implemented, then LAST_CONFIG_DWORD should be set to `0x40` (byte offset `0x100`). Any access to byte offset `0x100` is successful, but returns `0`s, indicating there are no capabilities. Any access beyond `0x100` results in a UR. <br><br>    b. If there are PCI Express Extended Capabilities implemented, then LAST_CONFIG_DWORD should set to the DWORD address of the final entry of the last capability. For instance, if the VC cap (which is 7 DWORDs long) is the last capability and if it is mapped with VC_BASE_PTR = `0x10C`, then LAST_CONFIG_DWORD should be set to: (`0x10C` / 4) + 7 = `0x49`. Any access to byte offset range `0x100` to `0x124` (DWORD `0x49`) is successful, and any access to byte offset range `0x128` to `0x3FF` causes a UR to be returned. <br><br> 2. If EXT_CFG_XP_CAP_PTR is active (not `0x3FF`), causing CfgRds and CfgWrs greater than or equal to EXT_CFG_XP_CAP_PTR to be routed to the user, then LAST_CONFIG_DWORD must be set as follows: <br><br>    a. If EXT_CFG_XP_CAP_PTR is `0x100` (indicating that all PCI Express Extended Config accesses go to the user), then LAST_CONFIG_DWORD = `0x40`. The user must generate URs for invalid accesses to `0x100` to `0x3FF`, if appropriate. <br><br>    b. If EXT_CFG_XP_CAP_PTR is greater than `0x100` (indicating that the range [`0x100`, EXT_CFG_XP_CAP_PTR-4] goes to the CMM), then LAST_CONFIG_DWORD should be set to the DWORD address of the final entry of the last capability implemented in the integrated block (as explained in 1b). Any access to byte offset range `0x100` to LAST_CONFIG_DWORD*4 is successful, and any access to byte offset range EXT_CFG_XP_CAP_PTR to `0x3FF` requires the user to determine validity and generate URs on the TRN interface, if necessary. If EXT_CFG_XP_CAP_PTR > (LAST_CONFIG_DWORD*4+4), then there is an unimplemented gap in the integrated block, and the integrated block returns a UR for any access to the range [LAST_CONFIG_DWORD*4+4, EXT_CFG_XP_CAP_PTR-4]. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LINK_CAP_ASPM_SUPPORT | Integer | Active State PM Support. This attribute indicates the level of active state power management supported by the selected PCI Express Link:<br><br>0: Reserved<br>1: L0s entry supported<br>2: Reserved<br>3: Reserved |
| LINK_CAP_CLOCK_POWER_MANAGEMENT | Boolean | This attribute is set to TRUE if the Endpoint supports removal of reference clocks in L1 and L23. This value is transferred to the Link Capabilities Register.<br><br>TRUE: 1<br>FALSE: 0 |
| LINK_CAP_DLL_LINK_ACTIVE_REPORTING_CAP | Boolean | Data Link Layer Link Active status notification is supported. This value is transferred to the Link Capabilities Register.<br><br>TRUE: 1<br>FALSE: 0<br><br>This attribute must be set to FALSE for Endpoints. |
| LINK_CAP_L0S_EXIT_LATENCY_COMCLK_GEN1 | Integer | This attribute sets the exit latency from the L0s state to be applied (at 2.5 Gb/s) where a common clock is used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br><br>0: Less than 64 ns<br>1: 64 ns to less than 128 ns<br>2: 128 ns to less than 256 ns<br>3: 256 ns to less than 512 ns<br>4: 512 ns to less than 1 µs<br>5: 1 µs to less than 2 µs<br>6: 2 µs to 4 µs<br>7: More than 4 µs |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LINK_CAP_L0S_EXIT_LATENCY_COMCLK_GEN2 | Integer | This attributes sets the exit latency from the L0s state to be applied (at 5.0 Gb/s) where a common clock is used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br>0: Less than 64 ns<br>1: 64 ns to less than 128 ns<br>2: 128 ns to less than 256 ns<br>3: 256 ns to less than 512 ns<br>4: 512 ns to less than 1 μs<br>5: 1 μs to less than 2 μs<br>6: 2 μs to 4 μs<br>7: More than 4 μs |
| LINK_CAP_L0S_EXIT_LATENCY_GEN1 | Integer | This attribute sets the exit latency from the L0s state to be applied (at 2.5 Gb/s) where separate clocks are used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br>0: Less than 64 ns<br>1: 64 ns to less than 128 ns<br>2: 128 ns to less than 256 ns<br>3: 256 ns to less than 512 ns<br>4: 512 ns to less than 1 μs<br>5: 1 μs to less than 2 μs<br>6: 2 μs to 4 μs<br>7: More than 4 μs |
| LINK_CAP_L0S_EXIT_LATENCY_GEN2 | Integer | This attribute sets the exit latency from the L0s state to be applied (at 5.0 Gb/s) where separate clocks are used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br>0: Less than 64 ns<br>1: 64 ns to less than 128 ns<br>2: 128 ns to less than 256 ns<br>3: 256 ns to less than 512 ns<br>4: 512 ns to less than 1 μs<br>5: 1 μs to less than 2 μs<br>6: 2 μs to 4 μs<br>7: More than 4 μs |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LINK_CAP_L1_EXIT_LATENCY_COMCLK_GEN1 | Integer | This attribute sets the exit latency from the L1 state to be applied (at 2.5 Gb/s) where a common clock is used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br>0: Less than 1 µs<br>1: 1 µs to less than 2 µs<br>2: 2 µs to less than 4 µs<br>3: 4 µs to less than 8 µs<br>4: 8 µs to less than 16 µs<br>5: 16 µs to less than 32 µs<br>6: 32 µs to 64 µs<br>7: More than 64 µs |
| LINK_CAP_L1_EXIT_LATENCY_COMCLK_GEN2 | Integer | This attribute sets the exit latency from the L1 state to be applied (at 5.0 Gb/s) where a common clock is used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br>0: Less than 1 µs<br>1: 1 µs to less than 2 µs<br>2: 2 µs to less than 4 µs<br>3: 4 µs to less than 8 µs<br>4: 8 µs to less than 16 µs<br>5: 16 µs to less than 32 µs<br>6: 32 µs to 64 µs<br>7: More than 64 µs |
| LINK_CAP_L1_EXIT_LATENCY_GEN1 | Integer | This attribute sets the exit latency from the L1 state to be applied (at 2.5 Gb/s) where separate clocks are used. This value is transferred to the Link Capabilities Register.<br><br>Valid settings are:<br>0: Less than 1 µs<br>1: 1 µs to less than 2 µs<br>2: 2 µs to less than 4 µs<br>3: 4 µs to less than 8 µs<br>4: 8 µs to less than 16 µs<br>5: 16 µs to less than 32 µs<br>6: 32 µs to 64 µs<br>7: More than 64 µs |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LINK_CAP_L1_EXIT_LATENCY_GEN2 | Integer | This attribute sets the exit latency from the L1 state to be applied (at 5.0 Gb/s) where separate clocks are used. This value is transferred to the Link Capabilities Register. Valid settings are: 0: Less than 1 µs 1: 1 µs to less than 2 µs 2: 2 µs to less than 4 µs 3: 4 µs to less than 8 µs 4: 8 µs to less than 16 µs 5: 16 µs to less than 32 µs 6: 32 µs to 64 µs 7: More than 64 µs |
| LINK_CAP_LINK_BANDWIDTH_NOTIFICATION_CAP | Boolean | Link Bandwidth notification capability. This value is transferred to the Link Capabilities Register. When this attribute is TRUE, there is support for the link bandwidth notification status and interrupt mechanism. This setting is required for Root. This attribute must be set to FALSE for Endpoints. |
| LINK_CAP_MAX_LINK_SPEED | 4-bit Hex | Maximum Link Speed. This value is transferred to the Link Capabilities Register. Valid settings are: `0001b`: [2.5 Gb/s] `0010b`: [5.0 Gb/s and 2.5 Gb/s] |
| LINK_CAP_MAX_LINK_WIDTH | 6-bit Hex | Maximum Link Width. This value is transferred to the Link Capabilities Register. Valid settings are: `000001b`: x1 `000010b`: x2 `000100b`: x4 `001000b`: x8 |
| LINK_CAP_RSVD_23_22 | Integer | Reserved bits in Link Capability Register. This attribute must be set to 0. |
| LINK_CAP_SURPRISE_DOWN_ERROR_CAPABLE | Boolean | This attribute must be set to FALSE for Endpoints. |
| LINK_CONTROL_RCB | Integer | This value is transferred to the Link Control Register. It is non-zero only for a Root Port. 1: Implies read completion boundary is 128 bits 0: Implies 64 bits |

*Table H-1:*   **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LINK_CTRL2_DEEMPHASIS | Boolean | This attribute sets the de-emphasis level used by the upstream component in 5.0 Gb/s mode. This value is transferred to the Link Control 2 Register, bit 12.<br><br>FALSE: –6 dB<br>TRUE: –3.5 dB |
| LINK_CTRL2_HW_AUTONOMOUS_SPEED_DISABLE | Boolean | When this attribute is TRUE, hardware is disabled from changing the link speed for reasons other than reliability. This value is transferred to the Link Control 2 Register, bit 5. |
| LINK_CTRL2_TARGET_LINK_SPEED | 4-bit Hex | This attribute sets an upper limit on the speed advertised by the Upstream component (Root). This value is transferred to the Link Control 2[3:0] Register. Valid settings are:<br><br>1h: 2.5 Gb/s Target Link Speed<br>2h: 5.0 Gb/s Target Link Speed |
| LINK_STATUS_SLOT_CLOCK_CONFIG | Boolean | Slot Clock Configuration. This attribute indicates where the component uses the same physical reference clock that the platform provides on the connector. For a port that connects to the slot, this attribute indicates that it uses a clock with a common source to that used by the slot. For an adaptor inserted in the slot, this attribute indicates that it uses the same clock source as the slot, not a locally derived clock source. This value is transferred to the Link Status Register, bit 12. |
| LL_ACK_TIMEOUT | Integer | This attribute sets a user-defined time-out for the Ack/Nak Latency Timer to force any pending ACK or NAK DLLPs to be transmitted. Refer to LL_ACK_TIMEOUT_EN and LL_ACK_TIMEOUT_FUNC to see how this value is used. The timer runs in the link_clk domain. Thus the amount of time that this value corresponds to scales up or down depending on the link width and Gen 1 or Gen 2. A value of "1" unit on this attribute corresponds to:<br><br>32 ns (x1, Gen 1)<br>16 ns (x2, Gen 1 or x1, Gen 2)<br>8 ns (x4, Gen 1 or x2, Gen 2)<br>4 ns (x8, Gen 1 or x4, Gen 2)<br>2 ns (x8, Gen 2)<br><br>Should be set to 0h for compliant operation. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| LL_ACK_TIMEOUT_EN | Boolean | When set to TRUE, this attribute enables the Ack/Nak Latency Timer to use the user-defined LL_ACK_TIMEOUT value (or combined with the built-in value, depending on LL_ACK_TIMEOUT_FUNC). When this attribute is set to FALSE, the built-in value is used. |
| LL_ACK_TIMEOUT_FUNC | Integer | This attribute defines how LL_ACK_TIMEOUT is to be used, if enabled with LL_ACK_TIMEOUT_EN (otherwise, this attribute is not used).<br><br>0: Absolute Value<br><br>1: Add to the built-in table value<br><br>2: Subtract from the built-in table value<br><br>It is the user's responsibility to ensure that if "1" or "2" is chosen, the time-out value does not overflow or underflow the 15-bit field. |
| LL_REPLAY_TIMEOUT | Integer | This attribute sets a user-defined time-out for the Replay Timer to cause the retransmission of unacknowledged TLPs. Refer to LL_REPLAY_TIMEOUT_EN and LL_REPLAY_TIMEOUT_FUNC to see how this value is used. The timer runs in the link_clk domain; thus the amount of time that this value corresponds to scales up or down depending on the link width and Gen 1 or Gen 2. A value of "1" unit on this attribute corresponds to:<br><br>32 ns (x1, Gen 1)<br><br>16 ns (x2, Gen 1 or x1, Gen 2)<br><br>8 ns (x4, Gen 1 or x2, Gen 2)<br><br>4 ns (x8, Gen 1 or x4, Gen 2)<br><br>2 ns (x8, Gen 2)<br><br>Should be set to `0h` for compliant operation. |
| LL_REPLAY_TIMEOUT_EN | Boolean | When set to TRUE, this attribute enables the Replay Timer to use the user-defined LL_REPLAY_TIMEOUT value (or combined with the built-in value, depending on LL_REPLAY_TIMEOUT_FUNC). When this attribute is set to FALSE, the built-in value is used. |

| Attribute Name | Type | Description |
|---|---|---|
| LL_REPLAY_TIMEOUT_FUNC | Integer | This attribute defines how LL_REPLAY_TIMEOUT is to be used, if enabled with LL_REPLAY_TIMEOUT_EN (otherwise, this attribute is not used).<br><br>0: Absolute value<br><br>1: Add to the built-in table value<br><br>2: Subtract from the built-in table value<br><br>It is the user's responsibility to ensure that if "1" or "2" is chosen, the time-out value does not overflow or underflow the 15-bit field. |
| LTSSM_MAX_LINK_WIDTH | 6-bit Hex | This attribute is used by LTSSM to set the Maximum Link Width. Valid settings are:<br><br>`000001b`: [x1]<br><br>`000010b`: [x2]<br><br>`000100b`: [x4]<br><br>`001000b`: [x8] |
| MSIX_BASE_PTR | 8-bit Hex | Byte address of the base of the MSI-X Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address.<br><br>This attribute must be set to `9Ch`. |
| MSIX_CAP_ID | 8-bit Hex | The capability identifier of MSI-X capability. This value is transferred to the MSI-X Capabilities Register.<br><br>This attribute must be set to `11h`. |
| MSIX_CAP_NEXTPTR | 8-bit Hex | MSI-X Capability's Next Capability Offset pointer to the next item in the capabilities list, or `00h` if this is the final capability. |
| MSIX_CAP_ON | Boolean | When set to TRUE, this attribute indicates that the MSI-X structure exists. If this attribute is FALSE, the MSI-X structure cannot be accessed via either the link or the management port. |
| MSIX_CAP_PBA_BIR | Integer | MSI-X Pending Bit Array BIR. This value is transferred to the MSI-X PBA BIR field. It is set to `0` if MSI-X is not enabled.<br><br>Valid settings are 0–5. |
| MSIX_CAP_PBA_OFFSET | 29-bit Hex | MSI-X Pending Bit Array Offset This value is transferred to the MSI-X PBA Offset field. It is set to `0` if MSI-X is not enabled. |
| MSIX_CAP_TABLE_BIR | Integer | MSI-X Table BIR. This value is transferred to the MSI-X Table BIR field. It is set to `0` if MSI-X is not enabled.<br><br>Valid settings are 0–5. |
| MSIX_CAP_TABLE_OFFSET | 29-bit Hex | MSI-X Table Offset. This value is transferred to the MSI-X Table Offset field. It is set to 0 if MSI-X is not enabled. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| MSIX_CAP_TABLE_SIZE | 11-bit Hex | MSI-X Table Size. This value is transferred to the MSI-X Message Control[10:0] field. It is set to 0 if MSI-X is not enabled. The integrated block does not implement the table; it must be implemented in user logic. |
| MSI_BASE_PTR | 8-bit Hex | Byte address of the base of the Message Signaled Interrupt (MSI) Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address.<br><br>This attribute must be set to `48h`. |
| MSI_CAP_64_BIT_ADDR_CAPABLE | Boolean | MSI 64-bit Addressing Capable. This value is transferred to the MSI Control Register, bit 7. |
| MSI_CAP_ID | 8-bit Hex | The capability identifier of MSI capability. This value is transferred to the MSI Capabilities Register, bits [7:0].<br><br>This attribute must be set to `05h`. |
| MSI_CAP_MULTIMSGCAP | Integer | Multiple Message Capable. Each MSI function can request up to 32 unique messages. System software can read this field to determine the number of messages requested. The number of messages requested are encoded as follows:<br><br>`0h`: 1 vector<br>`1h`: 2 vectors<br>`2h`: 4 vectors<br>`3h`: 8 vectors<br>`4h`: 16 vectors<br>`5h`: 32 vectors<br>`6h` - `7h`: Reserved |
| MSI_CAP_MULTIMSG_EXTENSION | Integer | Multiple Message Capable Extension. When this attribute is set to 1, 256 unique messages can be sent by the user (regardless of the setting of MSI_CAP_MULTIMSGCAP).<br><br>Valid settings are 0 and 1. |
| MSI_CAP_NEXTPTR | 8-bit Hex | MSI Capability's Next Capability Offset pointer to the next item in the capabilities list, or `00h` if this is the final capability. |
| MSI_CAP_ON | Boolean | When set to TRUE, this attribute indicates that the MSI structures exist. If this attribute is FALSE, the MSI structure cannot be accessed via either the link or the management port. |
| MSI_CAP_PER_VECTOR_MASKING_CAPABLE | Boolean | MSI Per-Vector Masking Capable. This value is transferred to the MSI Control Register, bit 8. When this attribute is set to TRUE, the Mask and Pending DWORD are added to the Cap structure. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| N_FTS_COMCLK_GEN1 | Integer | This attribute sets the number of FTS sequences advertised in the TS1 ordered sets when the Link Configuration Register shows that a common clock source is selected (used for all lanes when operating at 2.5 Gb/s). <br><br> This attribute must be set to 255. |
| N_FTS_COMCLK_GEN2 | Integer | This attribute sets the number of FTS sequences advertised in the TS1 ordered sets when the Link Configuration Register shows that a common clock source is selected (used for all lanes when operating at 5.0 Gb/s). <br><br> This attribute must be set to 255. |
| N_FTS_GEN1 | Integer | This attribute sets the number of FTS sequences advertised in the TS1 ordered sets when the Link Configuration Register shows that a common clock source is not selected (used for all lanes when operating at 2.5 Gb/s). <br><br> This attribute must be set to 255. |
| N_FTS_GEN2 | Integer | This attribute sets the number of FTS sequences advertised in the TS1 ordered sets when the Link Configuration Register shows that a common clock source is not selected (used for all lanes when operating at 5.0 Gb/s). <br><br> This attribute must be set to 255. |
| PCIE_BASE_PTR | 8-bit Hex | Byte address of the base of the PCI Express (PCIE) Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address. <br><br> This attribute must be set to `60h`. |
| PCIE_CAP_CAPABILITY_ID | 8-bit Hex | Capability ID for Express capability. <br><br> This attribute must be set to `10h`. |
| PCIE_CAP_CAPABILITY_VERSION | 4-bit Hex | This attribute indicates the version number of the PCI-SIG defined PCI Express capability structure. |
| PCIE_CAP_DEVICE_PORT_TYPE | 4-bit Hex | This attribute identifies the type of device/port. Valid settings are: <br><br> `0000b`: PCI Express Endpoint device <br><br> `0001b`: Legacy PCI Express Endpoint device <br><br> `0100b`: Root Port of PCI Express Root Complex <br><br> This value is transferred to the PCI Express Capabilities Register. It must be consistent with UPSTREAM_FACING settings. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| PCIE_CAP_INT_MSG_NUM | 5-bit Hex | Interrupt Message Number. This value is transferred to the PCI Express Cap Register [13:9]. It is not used internally by the integrated block. |
| PCIE_CAP_NEXTPTR | 8-bit Hex | PCIe Capability's Next Capability Offset pointer to the next item in the capabilities list, or `00h` if this is the final capability. |
| PCIE_CAP_ON | Boolean | When set to TRUE, this attribute indicates that the PCIE structure exists. If this attribute is FALSE, the PCIE structure cannot be accessed via either the link or the management port. |
| PCIE_CAP_RSVD_15_14 | Integer | This attribute sets the Reserved bits [15:14] of the PCIE Capability Register.<br><br>This attribute must be set to 0. |
| PCIE_CAP_SLOT_IMPLEMENTED | Boolean | Slot Implemented. When this attribute is TRUE, the PCI Express Link associated with this Port is connected to a slot (rather than to an integrated component). This attribute is valid only for the Root Port of the Root Complex. This value is transferred to the PCI Express Capabilities Register. |
| PCIE_REVISION | Integer | A value of 2 specifies *PCI Express Specification, rev. 2.0* compliance.<br><br>This attribute must be set to 2. |
| PGL0_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver Link. A link is all PIPETX0* and PIPERX0* signals.<br><br>This attribute must be set to 0. |
| PGL1_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX1* and PIPERX1* signals.<br><br>This attribute must be set to 1. |
| PGL2_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX2* and PIPERX2* signals.<br><br>This attribute must be set to 2. |
| PGL3_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX3* and PIPERX3* signals.<br><br>This attribute must be set to 3. |

*Table H-1:*  **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| PGL4_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX4* and PIPERX4* signals. <br><br> This attribute must be set to 4. |
| PGL5_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX5* and PIPERX5* signals. <br><br> This attribute must be set to 5. |
| PGL6_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX6* and PIPERX6* signals. <br><br> This attribute must be set to 6. |
| PGL7_LANE | Integer | This attribute indicates which logical lane (not considering the lane reversal feature) uses this PCIe to GTX transceiver link. A link is all PIPETX7* and PIPERX7* signals. <br><br> This attribute must be set to 7. |
| PL_AUTO_CONFIG | Integer | Bypass link width negotiation in LTSSM Configuration states. Valid settings are: <br><br> 0 - 3: Link width negotiation is not bypassed <br> 4: Link configures as x1 <br> 5: Link configures as x2 <br> 6: Link configures as x4 <br> 7: Link configures as x8 |
| PL_FAST_TRAIN | Boolean | When this attribute is TRUE, the timers in the LTSSM state machine are shortened to reduce simulation time. Specifically, the transition out of Polling.Active is shortened to requiring sending 16 TS1s and receiving 8 TS1s. The LTSSM timer values of 1 ms, 2 ms, 12 ms, 24 ms, and 48 ms, are reduced to 3.9 µs, 7.81 µs, 46.8 µs, 93.75 µs, and 187.5 µs, respectively (reduced by a factor of 256). <br><br> This attribute must be set to FALSE for silicon designs. |
| PM_BASE_PTR | 8-bit Hex | Byte address of the base of the Power Management (PM) Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address. <br><br> This attribute must be set to `40h`. |
| PM_CAP_AUXCURRENT | Integer | AUX Current. Requested auxiliary current allocation. This value is transferred to the PM Capabilities Register, bits [24:22]. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| PM_CAP_D1SUPPORT | Boolean | D1 Support. This value is transferred to the PM Capabilities Register, bit 25. |
| PM_CAP_D2SUPPORT | Boolean | D2 Support. This value is transferred to the PM Capabilities Register, bit 26. |
| PM_CAP_DSI | Boolean | Device Specific Initialization (DSI). This value is transferred to the PM Capabilities Register, bit 21. |
| PM_CAP_ID | 8-bit Hex | The capability identifier of the power management capability. This value is transferred to the PM Capabilities Register, bits [7:0]. |
| PM_CAP_NEXTPTR | 8-bit Hex | PM Capability's Next Capability Offset pointer to the next item in the capabilities list, or `00h` if this is the final capability. |
| PM_CAP_ON | Boolean | When set to TRUE, this attribute indicates that the PM structure exists. If this attribute is FALSE, the PM structure cannot be accessed via either the link or the management port. |
| PM_CAP_PMESUPPORT | 5-bit Hex | PME Support. These five bits indicate support for D3cold, D3hot, D2, D1, and D0, respectively. This value is transferred to the PM Capabilities Register, bits [31:27]. |
| PM_CAP_PME_CLOCK | Boolean | When set to TRUE, this attribute indicates that a PCI clock is required for PME generation. This attribute must be set to FALSE per the specification. This value is transferred to the PM Capabilities Register, bit 19. |
| PM_CAP_RSVD_04 | Integer | Reserved bit 20 of the PM Capabilities Register. This attribute must be set to 0. |
| PM_CAP_VERSION | Integer | The version of Power Management specification followed. This value is transferred to the PM Capabilities Register, bits [18:16]. This attribute must be set to 3. |
| PM_CSR_B2B3 | Boolean | Power Management CSR, bit 22: B2/B3 bit. |
| PM_CSR_BPCCEN | Boolean | Power Management CSR, bit 23: Bus Power/Clock Control Enable bit. |
| PM_CSR_NOSOFTRST | Boolean | Power Management CSR, bit 3: No Soft Reset bit. This attribute must be set to TRUE. |
| PM_DATA0 | 8-bit Hex | Power Management Data Register 0 (D0 Power Consumed). This value appears in the Data field of the PM Status Register if the host has written the value `0000b` to the Data Select field of the PM Control Register. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| PM_DATA1 | 8-bit Hex | Power Management Data Register 1 (D1 Power Consumed). This value appears in the Data field of the PM Status Register if the host has written the value `0001b` to the Data Select field of the PM Control Register. |
| PM_DATA2 | 8-bit Hex | Power Management Data Register 2 (D2 Power Consumed). This value appears in the Data field of the PM Status Register if the host has written the value `0010b` the Data Select field of the PM Control Register. |
| PM_DATA3 | 8-bit Hex | Power Management Data Register 3 (D3 Power Consumed). This value appears in the Data field of the PM Status Register if the host has written the value `0011b` to the Data Select field of the PM Control Register. |
| PM_DATA4 | 8-bit Hex | Power Management Data Register 4 (D0 Power Dissipated). This value appears in the Data field of the PM Status Register if the host has written the value `0100b` to the Data Select field of the PM Control Register. |
| PM_DATA5 | 8-bit Hex | Power Management Data Register 5 (D1 Power Dissipated). This value appears in the Data field of the PM Status Register if the host has written the value `0101b` to the Data Select field of the PM Control Register. |
| PM_DATA6 | 8-bit Hex | Power Management Data Register 6 (D2 Power Dissipated). This value appears in the Data field of the PM Status Register if the host has written the value `0110b` to the Data Select field of the PM Control Register. |
| PM_DATA7 | 8-bit Hex | Power Management Data Register 7 (D3 Power Dissipated). This value appears in the Data field of the PM Status Register if the host has written the value `0111b` to the Data Select field of the PM Control Register. |
| PM_DATA_SCALE0 | 2-bit Hex | Power Management Data Scale Register 0. This attribute specifies the scale applied to PM_DATA0. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `11b`: 0.001x |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| PM_DATA_SCALE1 | 2-bit Hex | Power Management Data Scale Register 1. This attribute specifies the scale applied to PM_DATA1. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>`00b`: 1.0x<br>`01b`: 0.1x<br>`10b`: 0.01x<br>`11b`: 0.001x |
| PM_DATA_SCALE2 | 2-bit Hex | Power Management Data Scale Register 2. This attribute specifies the scale applied to PM_DATA2. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>`00b`: 1.0x<br>`01b`: 0.1x<br>`10b`: 0.01x<br>`11b`: 0.001x |
| PM_DATA_SCALE3 | 2-bit Hex | Power Management Data Scale Register 3. This attribute specifies the scale applied to PM_DATA3. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>`00b`: 1.0x<br>`01b`: 0.1x<br>`10b`: 0.01x<br>`11b`: 0.001x |
| PM_DATA_SCALE4 | 2-bit Hex | Power Management Data Scale Register 4. This attribute specifies the scale applied to PM_DATA4. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>`00b`: 1.0x<br>`01b`: 0.1x<br>`10b`: 0.01x<br>`11b`: 0.001x |

| Attribute Name | Type | Description |
|---|---|---|
| PM_DATA_SCALE5 | 2-bit Hex | Power Management Data Scale Register 5. This attribute specifies the scale applied to PM_DATA5. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>00b: 1.0x<br><br>01b: 0.1x<br><br>10b: 0.01x<br><br>11b: 0.001x |
| PM_DATA_SCALE6 | 2-bit Hex | Power Management Data Scale Register 6. This attribute specifies the scale applied to PM_DATA6. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>00b: 1.0x<br><br>01b: 0.1x<br><br>10b: 0.01x<br><br>11b: 0.001x |
| PM_DATA_SCALE7 | 2-bit Hex | Power Management Data Scale Register 7. This attribute specifies the scale applied to PM_DATA7. The power consumption of the device is determined by multiplying the contents of the Base Power Data Register field with the value corresponding to the encoding returned by this field. Defined encodings are:<br><br>00b: 1.0x<br><br>01b: 0.1x<br><br>10b: 0.01x<br><br>11b: 0.001x |
| RECRC_CHK | Integer | Not supported. This attribute must be set to 0. |
| RECRC_CHK_TRIM | Boolean | Not supported. This attribute must be set to FALSE. |
| REVISION_ID | 8-bit Hex | ID identifying revision of device. This value is transferred to the Revision ID Register. |
| ROOT_CAP_CRS_SW_VISIBILITY | Boolean | CRS SW Visibility. When this attribute is TRUE, the Root Port can return CRS to software. This value is transferred to the Root Capabilities Register. |
| SELECT_DLL_IF | Boolean | Not supported. This attribute must be set to FALSE. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| SLOT_CAP_ATT_BUTTON_PRESENT | Boolean | Attention Button Present. When this attribute is TRUE, an Attention Button is implemented on the chassis for this slot. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_ATT_INDICATOR_PRESENT | Boolean | Attention Indicator Present. When this attribute is TRUE, an Attention Indicator is implemented on the chassis for this slot. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_ELEC_INTERLOCK_PRESENT | Boolean | Electromechanical Interlock Present. When this attribute is TRUE, an Electromechanical Interlock is implemented on the chassis for this slot. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_HOTPLUG_CAPABLE | Boolean | Hot-Plug Capable. When this attribute is TRUE, this slot can support Hot-Plug operations. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_HOTPLUG_SURPRISE | Boolean | Hot-Plug Surprise. When this attribute is TRUE, a device present in this slot might be removed from the system without any prior notification. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_MRL_SENSOR_PRESENT | Boolean | MRL Sensor Present. When this attribute is TRUE, a Manually operated Retention Latch (MRL) sensor is implemented on the chassis for this slot. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_NO_CMD_COMPLETED_SUPPORT | Boolean | No Command Completed Support. When this attribute is TRUE, the Slot does not issue SW notification when the Hot-Plug Controller completes command. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |

*Table H-1:*   **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| SLOT_CAP_PHYSICAL_SLOT_NUM | 13-bit Hex | Physical Slot Number. This attribute contains the physical slot number attached to this port. It is required to be globally unique within the chassis. This field should be set to 0 when the port is connected to devices that are integrated on the system board. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to `0h` for Endpoints. |
| SLOT_CAP_POWER_CONTROLLER_PRESENT | Boolean | Power Controller Present. When this attribute is TRUE, a Power Controller is implemented for this slot. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_POWER_INDICATOR_PRESENT | Boolean | Power Indicator Present. When this attribute is TRUE, a Power Indicator is implemented on the chassis for this slot. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to FALSE for Endpoints. |
| SLOT_CAP_SLOT_POWER_LIMIT_SCALE | Integer | Slot Power Limit Scale. This attribute specifies the scale used for the Slot Power Limit Value. Valid settings are:<br><br>0: 1.0x<br>1: 0.1x<br>2: 0.01x<br>3: 0.001x<br><br>This attribute must be set to 0 for Endpoints.<br><br>This value is transferred to the Slot Capabilities Register. |
| SLOT_CAP_SLOT_POWER_LIMIT_VALUE | 8-bit Hex | Slot Power Limit Value. The upper limit on the power supplied to the slot (in watts) is given by multiplying this value by the value in the Slot Power Limit Scale field. This value is transferred to the Slot Capabilities Register.<br><br>This attribute must be set to `00h` for Endpoints. |
| SPARE_BIT0 | Integer | Must be set to 0. |
| SPARE_BIT1 | Integer | Must be set to 0. |
| SPARE_BIT2 | Integer | Must be set to 0. |
| SPARE_BIT3 | Integer | Must be set to 0. |
| SPARE_BIT4 | Integer | Must be set to 0. |
| SPARE_BIT5 | Integer | Must be set to 0. |
| SPARE_BIT6 | Integer | Must be set to 0. |
| SPARE_BIT7 | Integer | Must be set to 0. |
| SPARE_BIT8 | Integer | Must be set to 0. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| SPARE_BYTE0 | 8-bit Hex | Must be set to 0. |
| SPARE_BYTE1 | 8-bit Hex | Must be set to 0. |
| SPARE_BYTE2 | 8-bit Hex | Must be set to 0. |
| SPARE_BYTE3 | 8-bit Hex | Must be set to 0. |
| SPARE_WORD0 | 32-bit Hex | Must be set to 0. |
| SPARE_WORD1 | 32-bit Hex | Must be set to 0. |
| SPARE_WORD2 | 32-bit Hex | Must be set to 0. |
| SPARE_WORD3 | 32-bit Hex | Must be set to 0. |
| SUBSYSTEM_ID | 16-bit Hex | This ID can be used to provide additional device information to that provided by the Device ID. This value is transferred to the Subsystem ID Register. |
| SUBSYSTEM_VENDOR_ID | 16-bit Hex | This ID can be used to provide additional vendor information to that provided by the Vendor ID. This value is transferred to the Subsystem Vendor ID Register. |
| TL_RBYPASS | Boolean | Not supported. This attribute must be set to FALSE. |
| TL_RX_RAM_RADDR_LATENCY | Integer | This attribute specifies the read address latency for RX RAMs in terms of USER_CLK cycles.<br>0: No fabric pipeline register is on the read address and enable block RAM inputs<br>1: A fabric pipeline register is on the read address and enable block RAM inputs |
| TL_RX_RAM_RDATA_LATENCY | Integer | This attribute specifies the read data latency for RAMs in terms of USER_CLK cycles.<br>1: The block RAM output register is disabled<br>2: The block RAM output register is enabled<br>3: The block RAM output register is enabled and a fabric pipeline register is added to the block RAM data output |
| TL_RX_RAM_WRITE_LATENCY | Integer | This attribute specifies the write latency for RX RAMs in terms of USER_CLK cycles.<br>0: No fabric pipeline register is on the write address and enable block RAM inputs<br>1: A fabric pipeline register is on the write address and enable block RAM inputs |
| TL_TFC_DISABLE | Boolean | Not supported. This attribute must be set to FALSE. |
| TL_TX_CHECKS_DISABLE | Boolean | Not supported. This attribute must be set to FALSE. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| TL_TX_RAM_RADDR_LATENCY | Integer | This attribute specifies the read address latency for TX RAMs in terms of USER_CLK cycles.<br><br>0: No fabric pipeline register on the read address and enable block RAM inputs<br><br>1: A fabric pipeline register is on the read address and enable block RAM inputs |
| TL_TX_RAM_RDATA_LATENCY | Integer | This attribute specifies the read data latency for TX RAMs in terms of USER_CLK cycles.<br><br>1: The block RAM output register is disabled<br><br>2: The block RAM output register is enabled<br><br>3: The block RAM output register is enabled and a fabric pipeline register is added to the block RAM data output |
| TL_TX_RAM_WRITE_LATENCY | Integer | This attribute specifies the write latency for TX RAMs in terms of USER_CLK cycles.<br><br>0: No fabric pipeline register on the write address and enable block RAM inputs<br><br>1: A fabric pipeline register is on the write address and enable block RAM inputs |
| UPCONFIG_CAPABLE | Boolean | When set to TRUE, this attribute enables the upconfigure capability. When this attribute is set to FALSE, the upconfigure capability is disabled. |
| UPSTREAM_FACING | Boolean | TRUE specifies Endpoint. FALSE specifies Root Port. |
| UR_INV_REQ | Boolean | When this attribute is TRUE, received ATS Invalidate Request Messages are handled as Unsupported Requests (URs). When this attribute is FALSE, received ATS Invalidate Request Messages are passed to the user. |
| USER_CLK_FREQ | Integer | User clock frequency. Valid settings are:<br><br>0: 31.25 MHz<br><br>1: 62.5 MHz<br><br>2: 125 MHz<br><br>3: 250 MHz<br><br>4 - 7: Reserved |
| VC0_CPL_INFINITE | Boolean | When this attribute is set to TRUE, the block advertises infinite completions. |
| VC0_RX_RAM_LIMIT | 13-bit Hex | The receive buffer limit address default is two block RAMs.<br><br>This attribute must be set to RX buffer bytes/8. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| VC0_TOTAL_CREDITS_CD | Integer | Number of credits that should be advertised for Completion data received on Virtual Channel 0. The bytes advertised must be less than or equal to the block RAM bytes available.<br><br>The equation to calculate bytes advertised is:<br><br>(ph * (rx_td_ecrc_trim ? 16 : 24)) + (pd * 16) + (nph * 24) + (ch * 16) + (cd * 16)<br><br>The equation to calculate block RAM bytes available is:<br><br>(vc0_rx_ram_limit + 1) * 8<br><br>See Table H-2, page 372 for valid settings. |
| VC0_TOTAL_CREDITS_CH | Integer | Number of credits that should be advertised for Completion headers received on Virtual Channel 0. The sum of the posted, non-posted, and completion header credits must be $\leq$ 80.<br><br>This attribute must be set to 36. |
| VC0_TOTAL_CREDITS_NPH | Integer | Number of credits that should be advertised for Non-Posted headers received on Virtual Channel 0. The number of Non-Posted data credits advertised by the block is equal to the number of Non-Posted header credits. The sum of the Posted, Non-Posted, and Completion header credits must be $\leq$ 80.<br><br>This attribute must be set to 12. |
| VC0_TOTAL_CREDITS_PD | Integer | Number of credits that should be advertised for Posted data received on Virtual Channel 0. The bytes advertised must be less than or equal to the block RAM bytes available.<br><br>The equation to calculate bytes advertised is:<br><br>(ph * (rx_td_ecrc_trim ? 16 : 24)) + (pd * 16) + (nph * 24) + (ch * 16) + (cd * 16)<br><br>The equation to calculate block RAM bytes available is:<br><br>(vc0_rx_ram_limit + 1) * 8<br><br>See Table H-2, page 372 for valid settings. |
| VC0_TOTAL_CREDITS_PH | Integer | Number of credits that should be advertised for Posted headers received on Virtual Channel 0. The sum of the Posted, Non-Posted, and Completion header credits must be $\leq$ 80.<br><br>This attribute must be set to 32. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| VC0_TX_LASTPACKET | Integer | Index of the last packet buffer used by TX TLM (that is, the number of buffers – 1). This value is calculated from the maximum payload size supported and the number of block RAMs configured for transmit.<br>See Table H-2, page 372 for valid settings. |
| VC_BASE_PTR | 12-bit Hex | Byte address of the base of the Virtual Channel (VC) Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address.<br>Valid settings are `000h` or `10Ch`. |
| VC_CAP_ID | 16-bit Hex | The capability identifier of Virtual Channel capability. This value is transferred to the VC Capabilities Register [15:0].<br>The valid setting is `0002h`. |
| VC_CAP_NEXTPTR | 12-bit Hex | Virtual Channel Capability's Next Capability Offset pointer to the next item in the capabilities list, or `000h` if this is the final capability.<br>Valid settings are `000h` or `128h`. |
| VC_CAP_ON | Boolean | When set to TRUE, this attribute indicates that the VC structure exists. If this attribute is FALSE, the VC structure cannot be accessed via either the link or the management port. |
| VC_CAP_REJECT_SNOOP_TRANSACTIONS | Boolean | Reject Snoop Transaction. When this attribute is TRUE, any TLP with the No Snoop bit set can be rejected as a UR. This value is transferred to the VC Resource Capability Register. |
| VC_CAP_VERSION | 4-bit Hex | The version of Virtual Channel Capability followed. This value is transferred to the VC Capabilities Register [19:16]. |
| VENDOR_ID | 16-bit Hex | Unique Manufacturer ID. This value is transferred to the Vendor ID Register. |
| VSEC_BASE_PTR | 12-bit Hex | Byte address of the base of the Vendor-Specific Capability Structure. Any access to this structure (via either the link or the management port) is relative to this address.<br>Valid settings are `000h` or `128h`. |
| VSEC_CAP_HDR_ID | 16-bit Hex | The vendor-defined ID number of the Vendor-Specific Capability. This value is transferred to the VSEC Header Register [15:0]. |
| VSEC_CAP_HDR_LENGTH | 12-bit Hex | The length of the Vendor-Specific Capability in bytes, including the Cap header itself. This value is transferred to the VSEC Header Register [31:20]. |

*Table H-1:* **PCIE_2_0 Attributes** *(Cont'd)*

| Attribute Name | Type | Description |
|---|---|---|
| VSEC_CAP_HDR_REVISION | 4-bit Hex | The revision of the Vendor-Specific Capability followed. This value is transferred to the VSEC Header Register, bits [19:16]. |
| VSEC_CAP_ID | 16-bit Hex | The capability identifier of the Vendor-Specific Capability. This value is transferred to the VSEC Capabilities Register, bits [15:0]. This structure is used to implement the Xilinx specific loopback control registers.<br><br>This attribute must be set to `000Bh`. |
| VSEC_CAP_IS_LINK_VISIBLE | Boolean | The VSEC structure can be detected by link-side configuration accesses if TRUE. Otherwise, it is only user-side visible. |
| VSEC_CAP_NEXTPTR | 12-bit Hex | VSEC's Next Capability Offset pointer to the next item in the capabilities list, or `000h` if this is the final capability.<br><br>This attribute must be set to `000h`. |
| VSEC_CAP_ON | Boolean | When set to TRUE, this attribute indicates that the VSEC structure exists. If this value is FALSE, the VSEC structure cannot be accessed via either the link or the management port. |
| VSEC_CAP_VERSION | 4-bit Hex | The version of the Vendor-Specific Capability followed. This value is transferred to the VSEC Capabilities Register, bits [19:16]. |

*Table H-2:* **Valid Data Credit Combinations**

| Parameter | Valid Combinations | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| DEV_CAP_MAX_PAYLOAD_SUPPORTED | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| VC0_TOTAL_CREDITS_CD | 77 | 154 | 77 | 154 | 154 | 308 | 308 | 616 |
| VC0_TOTAL_CREDITS_PD | 77 | 154 | 77 | 154 | 154 | 308 | 308 | 616 |
| VC0_TX_LAST_PACKET | 25 | 31 | 13 | 28 | 14 | 29 | 14 | 30 |

![Xilinx logo]

*Appendix I*

# PCIE_2_0 Timing Parameter Descriptions

This appendix lists the timing parameter names and descriptions related to the Virtex®-6 FPGA Integrated Block for PCI Express® designs. This information is useful for debugging timing issues. Values for these timing parameters can be obtained by running the Speedprint tool. Usage of Speedprint is documented in the *Development System Reference Guide*.

The timing parameters on the integrated block consist of either Setup/Hold or Clock-to-Out parameters. Table I-1 lists the timing parameter names, descriptions, signal grouping, and related clock domain for a given parameter.

*Table I-1:* **PCIE_2_0 Timing Parameters**

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| **Sequential Setup and Hold Times for Integrated Block Inputs[1]** | | |
| Tpcicck_MGTn / Tpcickc_MGTn[2] | PIPECLK | PIPERX$n$CHANISALIGNED[2] |
| | | PIPERX$n$CHARISK[1:0] |
| | | PIPERX$n$DATA[15:0] |
| | | PIPERX$n$ELECIDLE |
| | | PIPERX$n$PHYSTATUS |
| | | PIPERX$n$STATUS[2:0] |
| | | PIPERX$n$VALID |
| Tpcicck_PWR / Tpcickc_PWR | USERCLK | CFGPMDIRECTASPML1N |
| | | CFGPMSENDPMACKN |
| | | CFGPMSENDPMETON |
| | | CFGPMSENDPMNAKN |
| | | CFGPMTURNOFFOKN |
| | | CFGPMWAKEN |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcidck_CFG / Tpcickd_CFG | USERCLK | CFGBYTEENN[3:0] |
| | | CFGDI[31:0] |
| | | CFGDSBUSNUMBER[7:0] |
| | | CFGDSDEVICENUMBER[4:0] |
| | | CFGDSFUNCTIONNUMBER[2:0] |
| | | CFGDSN[63:0] |
| | | CFGDWADDR[9:0] |
| | | CFGERRACSN |
| | | CFGERRAERHEADERLOG[127:0] |
| | | CFGERRCORN |
| | | CFGERRCPLABORTN |
| | | CFGERRCPLTIMEOUTN |
| | | CFGERRCPLUNEXPECTN |
| | | CFGERRECRCN |
| | | CFGERRLOCKEDN |
| | | CFGERRPOSTEDN |
| | | CFGERRTLPCPLHEADER[47:0] |
| | | CFGERRURN |
| | | CFGINTERRUPTASSERTN |
| | | CFGINTERRUPTDI[7:0] |
| | | CFGINTERRUPTN |
| | | CFGPORTNUMBER[7:0] |
| | | CFGRDENN |
| | | CFGTRNPENDINGN |
| | | CFGWRENN |
| | | CFGWRREADONLYN |
| | | CFGWRRW1CASRWN |
| Tpcidck_DBG / Tpcickd_DBG | PIPECLK | PLDBGMODE[2:0] |
| | USERCLK | DBGMODE[1:0] |
| | | DBGSUBMODE |
| Tpcidck_DRP / Tpcickd_DRP | DRPCLK | DRPDADDR[8:0] |
| | | DRPDEN |
| | | DRPDI[15:0] |
| | | DRPDWE |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|------|--------------|-----------------|
| Tpcidck_LL2 / Tpcickd_LL2 | USERCLK | LL2SENDASREQL1N |
| | | LL2SENDENTERL1N |
| | | LL2SENDENTERL23N |
| | | LL2SUSPENDNOWN |
| | | LL2TLPRCVN |
| Tpcidck_PL / Tpcickd_PL | PIPECLK | PLDIRECTEDLINKAUTON |
| | | PLDIRECTEDLINKCHANGE[1:0] |
| | | PLDIRECTEDLINKSPEED |
| | | PLDIRECTEDLINKWIDTH[1:0] |
| | | PLDOWNSTREAMDEEMPHSOURCE |
| | | PLUPSTREAMPREFERDEEMPH |
| Tpcidck_PL2 / Tpcickd_PL2 | USERCLK | PL2DIRECTEDLSTATE[4:0] |
| Tpcidck_RSTN / Tpcickd_RSTN | PIPECLK | PLRSTN |
| | | PLTRANSMITHOTRST |
| | USERCLK | CMRSTN |
| | | CMSTICKYRSTN |
| | | DLRSTN |
| | | FUNCLVLRSTN |
| | | TLRSTN |
| Tpcidck_RXRAM / Tpcickd_RXRAM | USERCLK | MIMRXRDATA[67:0] |
| Tpcidck_TL2 / Tpcickd_TL2 | USERCLK | TL2ASPMSUSPENDCREDITCHECKN |
| | | TL2PPMSUSPENDREQN |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcidck_TRN / Tpcickd_TRN | USERCLK | TRNFCSEL[2:0] |
| | | TRNRDSTRDYN |
| | | TRNRNPOKN |
| | | TRNTCFGGNTN |
| | | TRNTD[63:0] |
| | | TRNTDLLPDATA[31:0] |
| | | TRNTDLLPSRCRDYN |
| | | TRNTECRCGENN |
| | | TRNTEOFN |
| | | TRNTERRFWDN |
| | | TRNTREMN |
| | | TRNTSOFN |
| | | TRNTSRCDSCN |
| | | TRNTSRCRDYN |
| | | TRNTSTRN |
| Tpcidck_TXRAM / Tpcickd_TXRAM | USERCLK | MIMTXRDATA[68:0] |
| **Sequential Clock-to-Output Times for Integrated Block Outputs** | | |
| Tpcicko_CFG | USERCLK | CFGAERECRCCHECKEN |
| | | CFGAERECRCGENEN |
| | | CFGCOMMANDBUSMASTERENABLE |
| | | CFGCOMMANDINTERRUPTDISABLE |
| | | CFGCOMMANDIOENABLE |
| | | CFGCOMMANDMEMENABLE |
| | | CFGCOMMANDSERREN |
| | | CFGDEVCONTROL2CPLTIMEOUTDIS |
| | | CFGDEVCONTROL2CPLTIMEOUTVAL[3:0] |
| | | CFGDEVCONTROLAUXPOWEREN |
| | | CFGDEVCONTROLCORRERRREPORTINGEN |
| | | CFGDEVCONTROLENABLERO |
| | | CFGDEVCONTROLEXTTAGEN |
| | | CFGDEVCONTROLFATALERRREPORTINGEN |
| | | CFGDEVCONTROLMAXPAYLOAD[2:0] |
| | | CFGDEVCONTROLMAXREADREQ[2:0] |
| | | CFGDEVCONTROLNONFATALREPORTINGEN |
| | | CFGDEVCONTROLNOSNOOPEN |
| | | CFGDEVCONTROLPHANTOMEN |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcicko_CFG (*Cont'd*) | USERCLK | CFGDEVCONTROLURERRREPORTINGEN |
| | | CFGDEVSTATUSCORRERRDETECTED |
| | | CFGDEVSTATUSFATALERRDETECTED |
| | | CFGDEVSTATUSNONFATALERRDETECTED |
| | | CFGDEVSTATUSURDETECTED |
| | | CFGDO[31:0] |
| | | CFGERRAERHEADERLOGSETN |
| | | CFGERRCPLRDYN |
| | | CFGINTERRUPTDO[7:0] |
| | | CFGINTERRUPTMMENABLE[2:0] |
| | | CFGINTERRUPTRDYN |
| | | CFGLINKCONTROLASPMCONTROL[1:0] |
| | | CFGLINKCONTROLAUTOBANDWIDTHINTEN |
| | | CFGLINKCONTROLBANDWIDTHINTEN |
| | | CFGLINKCONTROLCOMMONCLOCK |
| | | CFGLINKCONTROLEXTENDEDSYNC |
| | | CFGLINKCONTROLHWAUTOWIDTHDIS |
| | | CFGLINKCONTROLLINKDISABLE |
| | | CFGLINKCONTROLRCB |
| | | CFGLINKCONTROLRETRAINLINK |
| | | CFGLINKSTATUSAUTOBANDWIDTHSTATUS |
| | | CFGLINKSTATUSBANDWITHSTATUS |
| | | CFGLINKSTATUSCURRENTSPEED[1:0] |
| | | CFGLINKSTATUSDLLACTIVE |
| | | CFGLINKSTATUSLINKTRAINING |
| | | CFGLINKSTATUSNEGOTIATEDWIDTH[3:0] |
| | | CFGMSGDATA[15:0] |
| | | CFGMSGRECEIVED |
| | | CFGMSGRECEIVEDERRCOR |
| | | CFGMSGRECEIVEDERRFATAL |
| | | CFGMSGRECEIVEDERRNONFATAL |
| | | CFGMSGRECEIVEDPMASNAK |
| | | CFGMSGRECEIVEDSETSLOTPOWERLIMIT |
| | | CFGMSGRECEIVEDUNLOCK |
| | | CFGPCIELINKSTATE[2:0] |
| | | CFGPMRCVASREQL1N |
| | | CFGPMRCVENTERL1N |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcicko_CFG *(Cont'd)* | USERCLK | CFGPMRCVENTERL23N |
| | | CFGPMRCVREQACKN |
| | | CFGRDWRDONEN |
| | | CFGSLOTCONTROLELECTROMECHILCTLPULSE |
| | | CFGTRANSACTION |
| | | CFGTRANSACTIONADDR[6:0] |
| | | CFGTRANSACTIONTYPE |
| | | CFGVCTCVCMAP[6:0] |
| Tpcicko_DBG | PIPECLK | PLDBGVEC[11:0] |
| | USERCLK | DBGSCLRA |
| | | DBGSCLRB |
| | | DBGSCLRC |
| | | DBGSCLRD |
| | | DBGSCLRE |
| | | DBGSCLRF |
| | | DBGSCLRG |
| | | DBGSCLRH |
| | | DBGSCLRI |
| | | DBGSCLRJ |
| | | DBGSCLRK |
| | | DBGVECA[63:0] |
| | | DBGVECB[63:0] |
| | | DBGVECC[11:0] |
| Tpcicko_DRP | DRPCLK | DRPDO[15:0] |
| | | DRPDRDY |
| Tpcicko_LL2 | USERCLK | LL2BADDLLPERRN |
| | | LL2BADTLPERRN |
| | | LL2PROTOCOLERRN |
| | | LL2REPLAYROERRN |
| | | LL2REPLAYTOERRN |
| | | LL2SUSPENDOKN |
| | | LL2TFCINIT1SEQN |
| | | LL2TFCINIT2SEQN |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcicko_MGT | PIPECLK | PIPETXDEEMPH |
| | | PIPETXMARGIN[2:0] |
| | | PIPETXRATE |
| | | PIPETXRCVRDET |
| | | PIPETXRESET |
| Tpcicko_MGTn[2] | PIPECLK | PIPERX$n$POLARITY[2] |
| | | PIPETX$n$CHARISK[1:0] |
| | | PIPETX$n$COMPLIANCE |
| | | PIPETX$n$DATA[15:0] |
| | | PIPETX$n$ELECIDLE |
| | | PIPETX$n$POWERDOWN[1:0] |
| Tpcicko_MSG | USERCLK | CFGINTERRUPTMSIENABLE |
| | | CFGINTERRUPTMSIXENABLE |
| | | CFGINTERRUPTMSIXFM |
| | | CFGMSGRECEIVEDASSERTINTA |
| | | CFGMSGRECEIVEDASSERTINTB |
| | | CFGMSGRECEIVEDASSERTINTC |
| | | CFGMSGRECEIVEDASSERTINTD |
| | | CFGMSGRECEIVEDDEASSERTINTA |
| | | CFGMSGRECEIVEDDEASSERTINTB |
| | | CFGMSGRECEIVEDDEASSERTINTC |
| | | CFGMSGRECEIVEDDEASSERTINTD |
| Tpcicko_PL | PIPECLK | PLINITIALLINKWIDTH[2:0] |
| | | PLLANEREVERSALMODE[1:0] |
| | | PLLINKGEN2CAP |
| | | PLLINKPARTNERGEN2SUPPORTED |
| | | PLLINKUPCFGCAP |
| | | PLLTSSMSTATE[5:0] |
| | | PLPHYLNKUPN |
| | | PLRECEIVEDHOTRST |
| | | PLRXPMSTATE[1:0] |
| | | PLSELLNKRATE |
| | | PLSELLNKWIDTH[1:0] |
| | | PLTXPMSTATE[2:0] |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcicko_PL2 | USERCLK | PL2LINKUPN |
| | | PL2RECEIVERERRN |
| | | PL2RECOVERYN |
| | | PL2RXELECIDLE |
| | | PL2SUSPENDOK |
| Tpcicko_PWR | USERCLK | CFGLINKCONTROLCLOCKPMEN |
| | | CFGMSGRECEIVEDPMETO |
| | | CFGMSGRECEIVEDPMETOACK |
| | | CFGMSGRECEIVEDPMPME |
| Tpcicko_RSTN | USERCLK | RECEIVEDFUNCLVLRSTN |
| | | USERRSTN |
| Tpcicko_RXRAM | USERCLK | MIMRXRADDR[12:0] |
| | | MIMRXRCE |
| | | MIMRXREN |
| | | MIMRXWADDR[12:0] |
| | | MIMRXWDATA[67:0] |
| | | MIMRXWEN |
| Tpcicko_TL2 | USERCLK | TL2ASPMSUSPENDCREDITCHECKOKN |
| | | TL2ASPMSUSPENDREQN |
| | | TL2PPMSUSPENDOKN |

*Table I-1:* **PCIE_2_0 Timing Parameters** *(Cont'd)*

| Name | Clock Domain | Signal Grouping |
|---|---|---|
| Tpcicko_TRN | USERCLK | TRNFCCPLD[11:0] |
| | | TRNFCCPLH[7:0] |
| | | TRNFCNPD[11:0] |
| | | TRNFCNPH[7:0] |
| | | TRNFCPD[11:0] |
| | | TRNFCPH[7:0] |
| | | TRNLNKUPN |
| | | TRNRBARHITN[6:0] |
| | | TRNRD[63:0] |
| | | TRNRDLLPDATA[31:0] |
| | | TRNRDLLPSRCRDYN |
| | | TRNRECRCERRN |
| | | TRNREOFN |
| | | TRNRERRFWDN |
| | | TRNRREMN |
| | | TRNRSOFN |
| | | TRNRSRCDSCN |
| | | TRNRSRCRDYN |
| | | TRNTBUFAV[5:0] |
| | | TRNTCFGREQN |
| | | TRNTDLLPDSTRDYN |
| | | TRNTDSTRDYN |
| | | TRNTERRDROPN |
| Tpcicko_TXRAM | USERCLK | MIMTXRADDR[12:0] |
| | | MIMTXRCE |
| | | MIMTXREN |
| | | MIMTXWADDR[12:0] |
| | | MIMTXWDATA[68:0] |
| | | MIMTXWEN |

**Notes:**

1. Tpcicck_XXX = Setup time (before the clock edge). Tpcickc_XXX = Hold time (after the clock edge).
2. There are eight copies of each of the following parameters/signals, one for each lane ($n$ = 0 to 7).