LogiCORE™ IP Spartan-6 FPGA Integrated Endpoint Block v1.2 for PCI Express®

User Guide

UG654 September 16, 2009

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/24/09	1.0	Initial Xilinx release.
09/16/09	2.0	Updated core to v1.2 and ISE to v11.3. Added VHDL support.

Table of Contents

Preface: About This Guide

Chapter 1: Introduction

Chapter 2: Core Overview

Chapter 3: Licensing the Core

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: Programmed Input Output Example Design

Chapter 10: Root Port Model Test Bench

Chapter 11: Migration Considerations

Chapter 12: Known Restrictions

Chapter 13: Debugging Designs

Appendix A: Managing Receive-Buffer Space for Inbound Completions

Appendix B: PCIE_A1 Port Descriptions

Appendix C: PCIE_A1 Attribute Descriptions

Appendix D: PCIE_A1 Timing Parameter Descriptions

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Core Overview

Chapter 3: Licensing the Core

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: Programmed Input Output Example Design

Chapter 10: Root Port Model Test Bench

Chapter 11: Migration Considerations

Chapter 12: Known Restrictions

Chapter 13: Debugging Designs

Appendix A: Managing Receive-Buffer Space for Inbound Completions

Appendix B: PCIE_A1 Port Descriptions

Appendix C: PCIE_A1 Attribute Descriptions

Appendix D: PCIE_A1 Timing Parameter Descriptions

Schedule of Tables

Chapter 1: Introduction

Chapter 2: Core Overview

Chapter 3: Licensing the Core

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: Programmed Input Output Example Design

Chapter 10: Root Port Model Test Bench

Chapter 11: Migration Considerations

Chapter 12: Known Restrictions

Chapter 13: Debugging Designs

Appendix A: Managing Receive-Buffer Space for Inbound Completions

Appendix B: PCIE_A1 Port Descriptions

Appendix C: PCIE_A1 Attribute Descriptions

Appendix D: PCIE_A1 Timing Parameter Descriptions

Preface

About This Guide

The *Spartan®-6 FPGA Integrated Endpoint Block for PCI Express® User Guide* describes the function and operation of the Spartan-6 Integrated Endpoint Block for PCI Express (PCIe®) core, including how to design, customize, and implement the core.

Guide Contents

This manual contains the following chapters:

- Preface, "About this Guide" introduces the organization and purpose of this user guide and the conventions used in this document.
- Chapter 1, "Introduction," describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Core Overview,"describes the main components of the Integrated Endpoint Block core architecture.
- Chapter 3, "Licensing the Core," contains information about licensing 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 Endpoint Block using the Xilinx CORE Generator™ software.
- Chapter 6, "Designing with the Core,"provides instructions on how to design a device using the Integrated Endpoint Block core.
- Chapter 7, "Core Constraints," discusses the required and optional constraints for the Integrated Endpoint Block core.
- Chapter 8, "FPGA Configuration," discusses considerations for FPGA configuration and PCI Express.
- Chapter 9, "Programmed Input Output Example Design," describes the Programmed Input Output (PIO) example design for use with the core.
- Chapter 10, "Root Port Model Test Bench," describes the test bench environment, which provides a test program interface for use with the PIO example design.
- Chapter 11, "Migration Considerations," defines the differences in behaviors and options between the Integrated Endpoint Block and Endpoint PIPE core.
- Chapter 12, "Known Restrictions," describes any known restrictions for this core.
- Appendix A, "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.

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.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr`** ={**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr`** ={**`on`**|**`off`**}

Convention	Meaning or Use	Example

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis . . .	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-5 FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

Chapter 1

Introduction

This chapter introduces the Spartan®-6 FPGA Integrated Endpoint Block for PCI Express core and provides related information including system requirements, recommended design experience, additional core resources, technical support, and submitting feedback to Xilinx.

About the Core

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

The Spartan-6 FPGA Integrated Endpoint Block for PCI Express core 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 Integrated Endpoint Block for PCIe 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® 11.3

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

Recommended Design Experience

Although the Spartan-6 FPGA Integrated Endpoint 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 (UCF) is recommended.

Additional Core Resources

For detailed information and updates about the Integrated Endpoint Block core, see the following documents:

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

Additional information and resources related to the PCI Express technology are available from the following web sites:

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

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the Integrated Endpoint Block for PCIe core.

Xilinx will provide technical support for use of this product as described in this guide. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the core and the accompanying documentation.

Core

For comments or suggestions about the core, please submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, please submit a WebCase from www.xilinx.com/support. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

Chapter 2

Core Overview

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

Overview

Table 2-1 defines the Integrated Endpoint Block for PCIe solution.

Table 2-1: **Product Overview**

Product Name	FPGA Architecture	User Interface Width	Lane Widths Supported	Link Speeds Supported	PCI Express Base Specification Compliance
1-lane Integrated Endpoint Block	Spartan-6	32	x1	2.5 GT/s	v1.1

The Integrated Endpoint Block for PCIe core internally instances the Spartan-6 Integrated Endpoint Block for PCI Express. See Appendix A, "Managing Receive-Buffer Space for Inbound Completions," Appendix B, "PCIE_A1 Port Descriptions," and Appendix C, "PCIE_A1 Attribute Descriptions"for information about the software primitive, PCIE_A1, which represents the hardened-IP Integrated Endpoint Block. The Integrated Endpoint Block follows the *PCI Express Base Specification* layering model, which consists of the Physical, Data Link, and Transaction Layers.

Figure 2-1 illustrates the interfaces to the core, as defined below:

- System (SYS) interface
- PCI Express (PCI_EXP) interface
- Configuration (CFG) interface
- Transaction (TRN) 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.

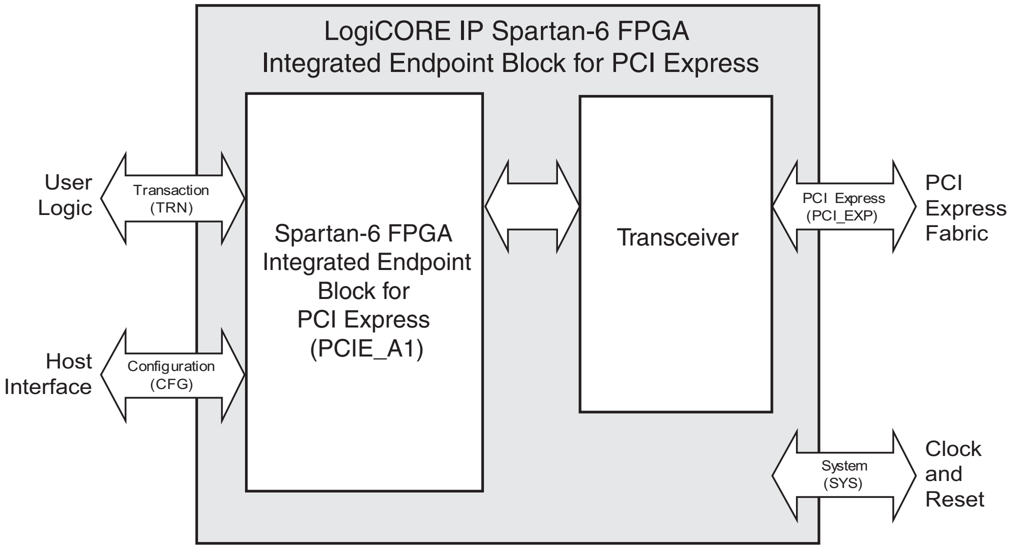


Figure 2-2: **Top-level Functional Blocks and Interfaces**

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, de-serialization and other circuitry for interface operation. Each layer is defined below.

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, IO, 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 is responsible for framing and de-framing of TLPs and DLLPs. It also implements the Link Training and Status State machine (LTSSM) which handles link initialization, training, and maintenance. Scrambling, de-scrambling 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 Polarity Inversion, as indicated in the *PCI Express Base Specification rev 1.1* requirement.

Configuration Management

The Configuration Management layer maintains the PCI Type0 Endpoint configuration space and supports the following features:

- Implements 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
- Implements the Device Serial Number Capability in the PCIe Extended Capability space

PCI Configuration Space

The configuration space consists of three primary parts, illustrated in Table 2-4. These include the following:

- Legacy PCI v3.0 Type 0 Configuration Header
- Legacy Extended Capability Items
    - PCIe Capability Item
    - Power Management Capability Item
    - Message Signaled Interrupt (MSI) Capability Item

- PCIe Extended Capabilities
    - Device Serial Number Extended Capability Structure (optional)

The core implements three legacy extended capability items. The remaining legacy extended capability space from address 0x6C to 0xFF is reserved. The core returns 0x00000000 when this address range is read.

The core also optionally implements one PCIe Extended Capability. The remaining PCIe Extended Capability space is reserved. If the Device Serial Number Capability is implemented, addresses from 0x10C to 0xFFF are reserved; otherwise addresses from 0x104 to 0xFFF are reserved. The core returns a Completion with Data of 0x00000000 if there is a configuration read to addresses in the reserved space range; writes are ignored.

Table 2-2: **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
	Base Address Register 0				010h
	Base Address Register 1				014h
	Base Address Register 2				018h
	Base Address Register 3				01Ch
	Base Address Register 4				020h
	Base Address Register 5				024h
	Cardbus CIS Pointer				028h
	Subsystem ID		Subsystem Vendor ID		02Ch
	Expansion ROM Base Address				030h
	Reserved			CapPtr	034h
	Reserved				038h
	Max Lat	Min Gnt	Intr Pin	Intr Line	03Ch
	PM Capability		NxtCap	PM Cap	040h
	Data	BSE	PMCSR		044h
	MSI Control		NxtCap	MSI Cap	048h
	Message Address (Lower)				04Ch
	Message Address (Upper)				050h
	Reserved		Message Data		054h
	PE Capability		NxtCap	PE Cap	058h
	PCI Express Device Capabilities				05Ch
	Device Status		Device Control		060h
	PCI Express Link Capabilities				064h
	Link Status		Link Control		068h
	Unimplemented Configuration Space (Returns 0x00000000)				06Ch-0FFh
Optional Returns 0 if not implemented	Next Cap	Capability	PCI Exp. Ext. Cap.(DSN)		100h
	PCI Express Device Serial Number (1st)				104h
	PCI Express Device Serial Number (2nd)				108h
	Reserved Extended Configuration Space (Returns Completion with 0x00000000)				10Ch-FFFh

Core Interfaces

The Integrated Endpoint 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`, the system clock signal, `sys_clk`, and a hot reset indicator, `received_hot_reset`, as described in Table 2-3.

Table 2-3: **System Interface Signals**

Function	Signal Name	Direction	Description
System Reset	sys_reset_n	Input	Asynchronous, active low signal.
System Clock	sys_clk	Input	Reference clock: 125 or 250 MHz.
Hot Reset	received_hot_reset	Output	The core received a hot reset.

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 either 125 MHz or 250 MHz, as selected in the CORE Generator GUI.

PCI Express Interface

The PCI Express (PCI_EXP) interface consists of differential transmit and receive pairs. 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. Transmit and receive signals of the PCI_EXP interface are defined in Tables 2-4.

Table 2-4: **PCI Express Interface Signals for the 1-lane Endpoint 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 (-)

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 Tables 2-5, 2-6, and 2-7.

Common TRN Interface

Table 2-5 defines and describes the common TRN interface signals.

Table 2-5: **Common Transaction Interface Signals**

Name	Direction	Description
trn_clk	Output	**Transaction Clock**: Transaction and Configuration interface operations are referenced-to and synchronous-with the rising edge of this clock. trn_clk is unavailable when the core sys_reset_n is held asserted. trn_clk is guaranteed to be stable at the nominal operating frequency only after trn_reset_n is de-asserted. The trn_clk clock output is a fixed frequency clock output. trn_clk does not change frequencies in case of link recovery. • 1-lane Integrated Endpoint Block Frequency: 62.5 MHz
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. trn_reset_n is deasserted synchronously with respect to trn_clk, trn_reset_n is asserted asynchronously with sys_reset_n assertion. Note that trn_reset_n is asserted for core in-band reset events like 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. trn_lnk_up_n is also deasserted when the core is driven to Hot Reset or Link Disable states by the link partner, and all TLPs stored in the core are lost.
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: 000 == receive buffer available space 001 == receive credits granted to the link partner 010 == receive credits consumed 100 == transmit user credits available 101 == transmit credit limit 110 == transmit credits consumed
trn_fc_ph[7:0]	Output	**Posted Header Flow Control Credits:** The number of Posted Header FC credits for the selected flow control type
trn_fc_pd[11:0]	Output	**Posted Data Flow Control Credits:** The number of Posted Data FC credits for the selected flow control type.

Table 2-5: **Common Transaction Interface Signals**

Name	Direction	Description
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.

Transmit TRN Interface

Table 2-6 defines the transmit (Tx) TRN interface signals.

Table 2-6: **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[31:0]	Input	**Transmit Data:** Packet data to be transmitted.
trn_tsrc_rdy_n	Input	**Transmit Source Ready**: Active low. Indicates that the User Application is presenting valid data on trn_td[31:0].
trn_tdst_rdy_n	Output	**Transmit Destination Ready**: Active low. Indicates that the core is ready to accept data on trn_td[31:0]. The simultaneous assertion of trn_tsrc_rdy_n and trn_tdst_rdy_n marks the successful transfer of one data beat on trn_td[31:0].
trn_tsrc_dsc_n	Input	**Transmit Source Discontinue**: Active low. Can be asserted any time starting on the first cycle after SOF to EOF, inclusive.
trn_tbuf_av [5:0]	Output	**Transmit Buffers Available**: Indicates the number of transmit buffers available in the core. Each free transmit buffer can accommodate one TLP up to the supported Maximum Payload Size. 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 may begin before the entire packet has been written to the core. Commonly referred to as transmit cut-through mode.
trn_tcfg_req_n	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	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 higher priority of transmission over core generated TLPs. trn_tcfg_req_n will be asserted once for each internally-generated packet. It may not deassert 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 may 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. trn_terrfwd_n must not be asserted if trn_tstr_n is asserted.

Receive TRN Interface

Table 2-7 defines the receive (Rx) TRN interface signals.

Table 2-7: **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[31:0]	Output	**Receive Data:** Packet data being received. Valid only if trn_rsrc_rdy_n is also asserted.
trn_rerrfwd_n	Output	**Receive Error Forward:** Active low. Marks the packet in progress as error poisoned. Asserted by the core for the entire length of the packet.
trn_rsrc_rdy_n	Output	**Receive Source Ready:** Active low. Indicates the core is presenting valid data on trn_rd[31:0]
trn_rdst_rdy_n	Input	**Receive Destination Ready:** Active low. Indicates the User Application is ready to accept data on trn_rd[31:0]. The simultaneous assertion of trn_rsrc_rdy_n and trn_rdst_rdy_n marks the successful transfer of one data beat on trn_td[31:0].
trn_rsrc_dsc_n	Output	**Receive Source Discontinue:** Active low. Indicates the core is aborting the current packet. Asserted when the physical link is going into reset. Active low.

Table 2-7: **Receive Transaction Interface Signals *(Cont'd)***

Name	Direction	Description
trn_rnp_ok_n	Input	**Receive Non-Posted OK:** Active low. The user application asserts this signal when it is ready to accept a Non-Posted Request TLP. trn_rnp_ok_n must be deasserted when the user application cannot process received Non-Posted TLPs, so that these may 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. 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. • trn_rbar_hit_n[0] => BAR0 • trn_rbar_hit_n[1] => BAR1 • trn_rbar_hit_n[2] => BAR2 • trn_rbar_hit_n[3] => BAR3 • trn_rbar_hit_n[4] => BAR4 • trn_rbar_hit_n[5] => BAR5 • trn_rbar_hit_n[6] => Expansion ROM Address. Note that if two BARs are configured into a single 64-bit address, both corresponding trn_rbar_hit_n bits are asserted.

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 double word (DWORD) registers. The endpoint returns the state of the selected register over the 32-bit data output port. Table 2-8 defines the Configuration interface signals. See "Accessing Configuration Space Registers," page 93 for usage.

Table 2-8: **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. • For a user configuration register read operation, the signal validates the cfg_do[31:0] data-bus value. • Writes to the configuration space are not supported.
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_rd_en_n	Input	**Configuration Read Enable**: Active low read-enable for configuration register access. ***Note:*** cfg_rd_en_n must be asserted for no more than 1 trn_clk cycle for each access.
cfg_interrupt_n	Input	**Configuration Interrupt**: Active-low interrupt-request signal. The User Application may assert this 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. **Value Message Type** 0 Assert 1 Deassert

Table 2-8: **Configuration Interface Signals** ***(Cont'd)***

Name	Direction	Description
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), the following list defines the type of message to be sent: **Value** **Legacy Interrupt** 00h INTA 01h INTB 02h INTC 03h INTD
cfg_interrupt_do[7:0]	Output	**Configuration Interrupt Data Out:** The value of the lowest 8 bits of the Message Data field in the endpoint's MSI capability structure. This value is not used and 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 may be used for cfg_interrupt_di[7:0]. cfg_interrupt_mmenable[2:0] values: • 000b, 0 bits • 001b, 1 bit • 010b, 2 bits • 011b, 3 bits • 100b, 4 bits • 101b, 5 bits
cfg_interrupt_msienable	Output	**Configuration Interrupt MSI Enabled**: Indicates that the Message Signaling Interrupt (MSI) messaging is enabled. If 0, then only Legacy (INTx) interrupts may be sent. If 1, only MSI interrupts may be sent.
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 Write packet is received.

Table 2-8: **Configuration Interface Signals** ***(Cont'd)***

Name	Direction	Description
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 Write 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 hard-wired to 000b.
cfg_status[15:0]	Output	**Configuration Status**: Status register from the Configuration Space Header.
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 Extended Capability Structure.
cfg_dcommand[15:0]	Output	**Configuration Device Command**: Device control register from the PCI Express Extended Capability Structure.
cfg_lstatus[15:0]	Output	**Configuration Link Status**: Link status register from the PCI Express Extended Capability Structure.
cfg_lcommand[15:0]	Output	**Configuration Link Command**: Link control register from the PCI Express Extended Capability Structure.
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 Integrated Endpoint core that it is safe to turn the power off.
cfg_pm_wake_n	Input	**Configuration Power Management Wake**: A one-clock cycle active low assertion signals 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 2-8: **Configuration Interface Signals *(Cont'd)***

Name	Direction	Description
cfg_pcie_link_state_n[2:0]	Output	**PCI Express Link State**: This encoded bus reports the PCIe Link State Information to the user. • 110b - PCI Express Link State is "L0" • 101b - PCI Express Link State is "L0s" • 011b - PCI Express Link State is "L1" • 111b - PCI Express Link State is "in transition"
cfg_trn_pending_n	Input	**User Transaction Pending**: If asserted, sets the Transactions Pending bit in the Device Status Register. **Note**: The user is required to assert this input if the User Application has not received a completion to an upstream request. Active low.
cfg_dsn[63:0]	Input	**Configuration Device Serial Number**: Serial Number Register fields of the Device Serial Number extended capability. Not used if DSN capability is disabled.

Error Reporting Signals

Table 2-9 defines the User Application error-reporting signals.

Table 2-9: **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	Input	**Configuration Error Completion Timeout**: Active low. The user can assert this signal to report a completion timed out. **Note**: The user should assert this signal only if the device power state is D0. Asserting this signal 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.1.1, Section 5.3.1.2.
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.

Table 2-9: **User Application Error-Reporting Signals** ***(Cont'd)***

Port Name	Direction	Description
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 which caused the error was a posted transaction.
cfg_err_cor_n	Input	**Configuration Error Correctable Error**: Active low. The user can assert this signal to report that a correctable error was detected.
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. The following information should be extracted from the received error TLP and presented in the format below: [47:41] Lower Address [40:29] Byte Count [28:26] TC [25:24] Attr [23:8] Requester ID [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. This signal is intended to be used 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. **Note**: When not in Legacy mode, the core will automatically return a Completion Locked, if appropriate.

Chapter 3

Licensing the Core

This chapter provided licensing options for the Spartan®-6 FPGA Integrated Endpoint Block for PCI Express core, which you must do before using the core in your designs. The core is provided under the terms of the Xilinx LogiCORE Site License Agreement, which conforms to the terms of the SignOnce IP License standard defined by the Common License Consortium.

Before you Begin

This chapter assumes you have installed the core using either the CORE Generator™ IP Software Update installer or by performing a manual installation after downloading the core from the web. For additional information about installing the core, see the Integrated Endpoint Block for PCIe product page.

License Options

After installing the required Xilinx ISE software and IP Service Packs, please refer to "Obtaining Your License Key" for instructions on obtaining a full license key.

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 Your License Key

To obtain a Full license key:

1. Navigate to the product page for this core:
   www.xilinx.com/products/ipcenter/S6_PCI_Express_Block.htm
2. Click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

After submitting your license key request, you will be sent an email with a full license key, along with instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Chapter 4

Getting Started Example Design

This chapter provides an overview of the Spartan®-6 FPGA Integrated Endpoint 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.

Overview

The example simulation design 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 Endpoint Block core and processed by the PIO example design. Figure 4-1 illustrates the simulation design provided with the Integrated Endpoint Block core. For more information about the Root Port Model, see Chapter 10, "Root Port Model Test Bench."

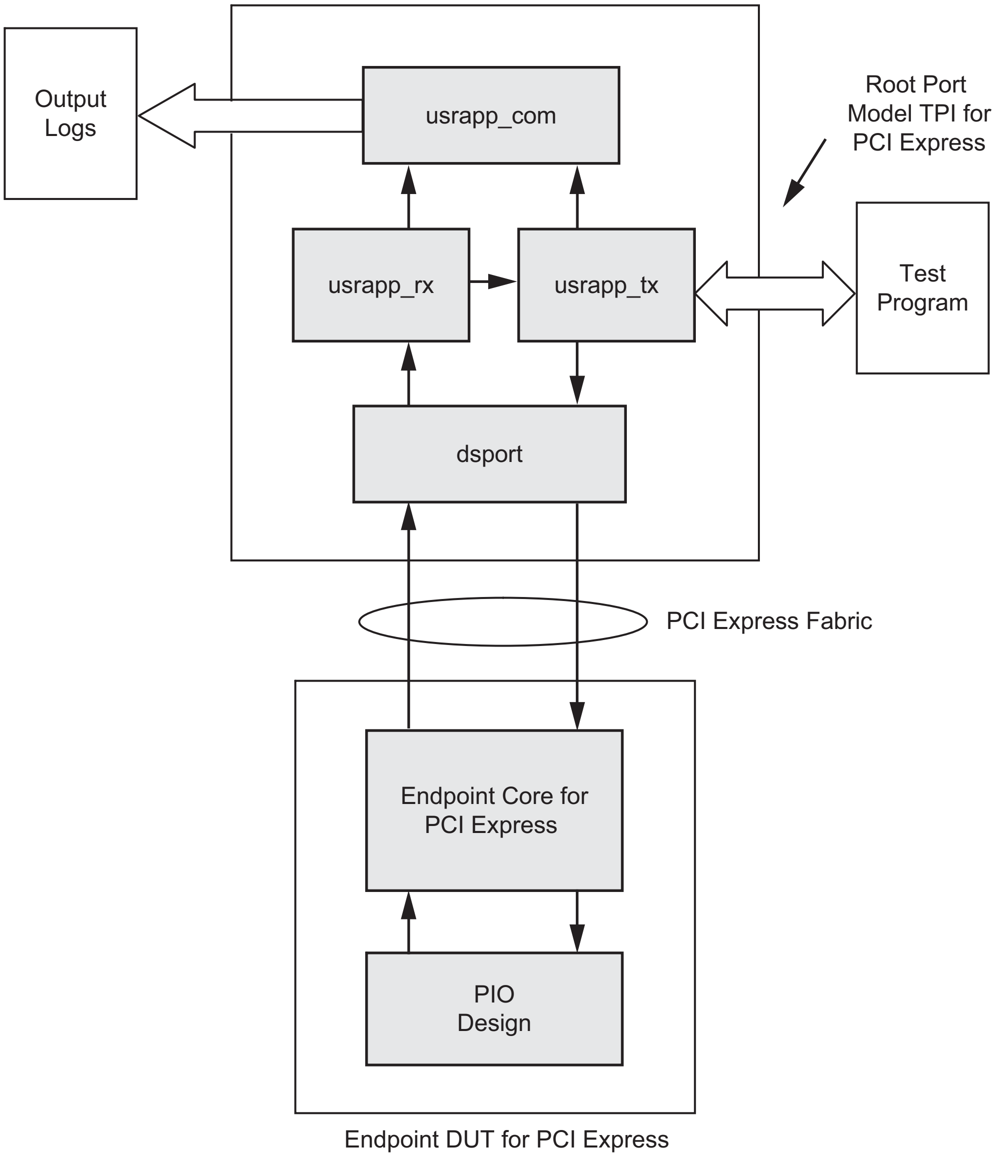


Figure 4-1: **Simulation Example Design Block Diagram**

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 Chapter 9, "Programmed Input Output Example Design."

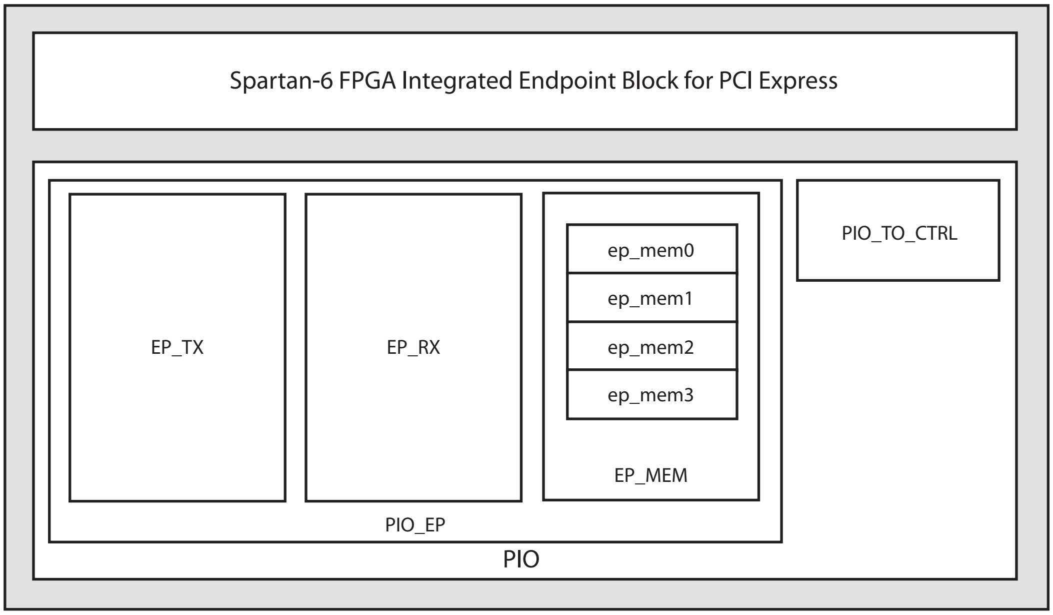


Figure 4-2: **Implementation Example Design Block Diagram**

Example Design Elements

The PIO example design elements include the following:

- 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 v11.3 and the following simulators:

- Mentor Graphics® ModelSim® v6.4b and above
- Cadence® IUS 8.1 -s009 and above
- Synopsys® 2008.09 and above

Note: Currently, the VHDL example design supports only ModelSim.

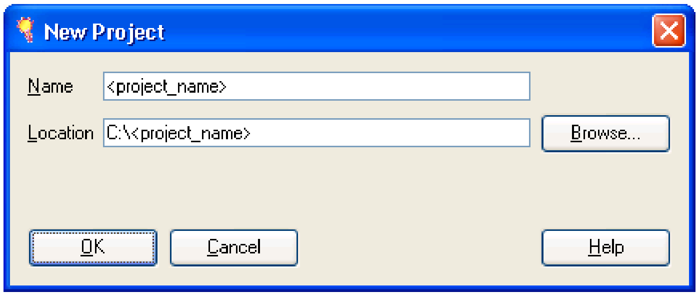
Generating the Core

To generate a core using the default values in the CORE Generator Graphical User Interface (GUI), do the following:

1. Start the CORE Generator.

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

2. Choose File > New Project.

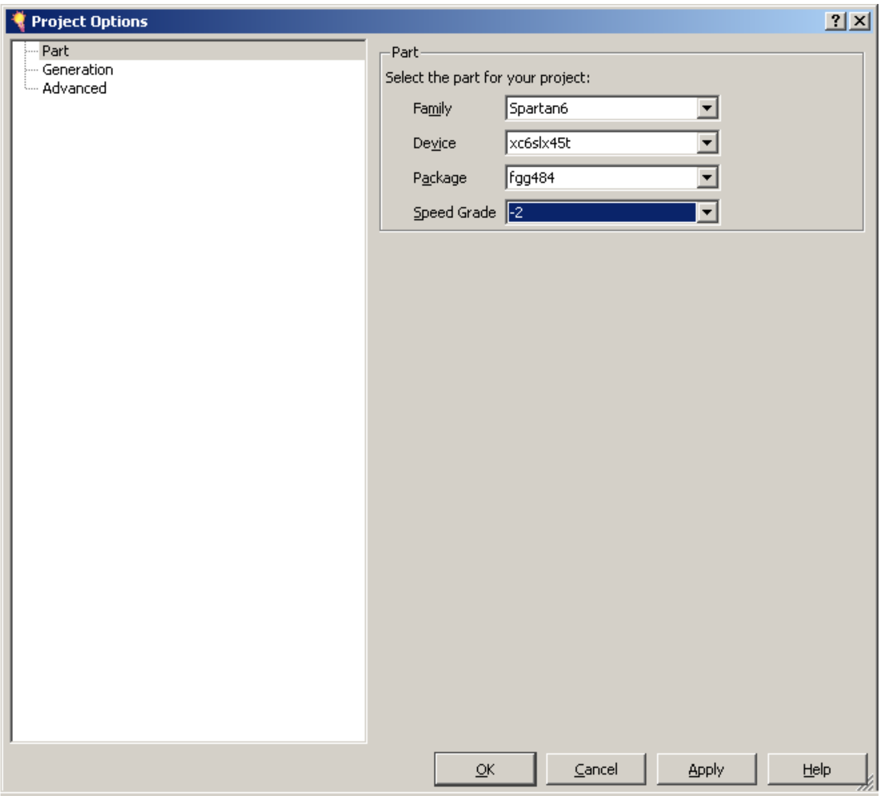


Name: <project_name>

Location: C:\<project_name>

Figure 4-3: **New Project Dialog Box**

3. Enter a project name and location, then click OK. `<project_dir>` is used in this example. The Project Options dialog box appears.



Select the part for your project:

Family: Spartan6

Device: xc6slx45t

Package: fgg484

Speed Grade: -2

Figure 4-4: **Project Options**

4. Set the project options:

   From the Part tab, select the following options:

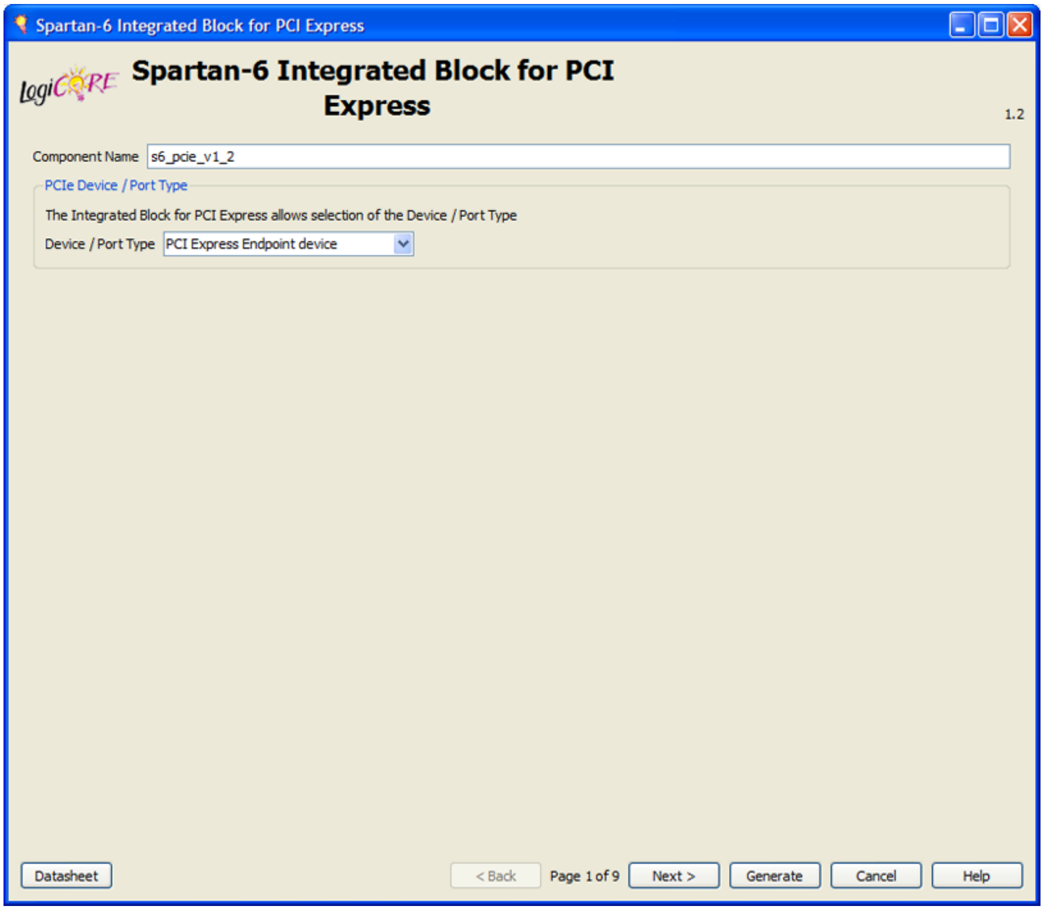
   - **Family**: Spartan6
   - **Device**: xc6slx45t
   - **Package**: fgg484
   - **Speed Grade**: -1

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

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

   - **Design Entry**. Select Verilog or VHDL.
   - **Vendor**. Select Synplicity® or ISE (for 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 Endpoint Block main screen.


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
PCIe Device / Port Type
Datasheet
< Back
Page 1 of 9
Next >
Generate
Cancel
Help

Component Name: s6_pcie_v1_2

The Integrated Block for PCI Express allows selection of the Device / Port Type

Device / Port Type: PCI Express Endpoint device

Figure 4-5: **Integrated Endpoint Block Main Screen**

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

7. Click Finish to generate the core using the default parameters. The core and its supporting files, including the PIO example design and Root Port 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 46.See the README file.

Simulating the Example Design

The example design provides a quick way to simulate and observe the behavior of the core. The simulation environment provided with the Integrated Endpoint Block core 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 Chapter 10, "Root Port Model Test Bench."

Setting up for Simulation

To run the gate-level simulation you must have the Xilinx Simulation Libraries compiled for your 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

Spartan-6 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Note for Cadence IUS users: The work construct must be manually inserted into your CDS.LIB file as shown below.

```
DEFINE WORK WORK
```

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 the following components:

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

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

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

2. Run the script that corresponds to your simulation tool using one of the following:
   - **ModelSim**: `vsim -do simulate_mti.do`
   - **VCS**: `>./simulate_vcs.sh`
   - **IUS**: `> ./simulate_ncsim.sh`

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 you 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 the following processes:

1. Removes data files from the previous runs.
2. Synthesizes the example design using XST or Synplify.
3. `ngdbuild`. Builds a Xilinx design database for the example design.

   Inputs:

   Part-Package-Speed Grade selection:

   For example, XC6SLX45T-FGG484-1

   Example design UCF:

   `xilinx_pcie_1_lane_ep_<device>.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 HDL or VHDL representation of the design and an SDF file for post-layout verification.
8. `bitgen`: Generates a bitstream file for programming the FPGA.

The following 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 Verilog or VHDL source files and results in a bitstream file. It is possible to use the Xilinx ISE GUI to implement the example design. However, the GUI flow is not presented in this document.

Directory Structure and File Contents

The Integrated Endpoint Block for PCIe 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
        Results directory, created after implementation scripts are run, and contains implement script results
    - <component name>/simulation
      - simulation/dsport
        Root Port Bus Functional Model
      - simulation/functional
        Functional simulation files
      - simulation/tests
        Test command files
    - <component name>/source
      Core source files

<project directory>

The project directory contains all the CORE Generator project files.

Table 4-1: **Project Directory**

Name	Description
`<project_dir>`	
`<component_name>.xco`	CORE Generator project-specific option file; can be used as an input to the CORE Generator.
`<component_name>_flist.txt`	List of files delivered with core.
`<component_name>.v[eo|ho]`	Verilog or VHDL instantiation template.

Back to Top

<project directory>/<component name>

The component name directory contains the release notes readme file provided with the core, which may includes tool requirements, last-minute changes, updates, and issue resolution.

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

Name	Description
`<project_dir>/<component_name>`	
`s6_pcie_readme.txt`	Readme 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`	
`s6_pcie_ug654.pdf`	*Spartan-6 FPGA Integrated Endpoint Block for PCI Express User Guide.*
`s6_pcie_ds718.pdf`	*Spartan-6 FPGA Integrated Endpoint 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: **Example Design Directory**

Name	Description
`<project_dir>/<component_name>/example_design`	
`xilinx_pcie_1_lane_ep_<device>.ucf`	Example design UCF. Filename varies by part, package, and speed-grade.
`xilinx_pcie_1_1_ep_s6.v[hd]`	Verilog top-level PIO example design files for 1-lane cores.

`pcie_app_s6.v[hd]`
`PIO_EP_MEM.v[hd]`
`PIO.v[hd]`
`PIO_EP.v[hd]`
`PIO_EP_MEM_ACCESS.v[hd]`
`PIO_TO_CTRL.v[hd]`
`PIO_32.v[hd]`
`PIO_32_RX_ENGINE.v[hd]`
`PIO_32_TX_ENGINE.v[hd]`
	PIO example design files.

Back to Top

<component name>/implement

The implement directory contains the core implementation script files.

Table 4-5: **Implement Directory**

Name	Description
`<project_dir>/<component_name>/implement`	
`xst.scr`	XST synthesis script.
`implement.bat` `implement.sh`	DOS and Linux implementation scripts.
`synplify.prj`	Synplify synthesis script.
`xst.prj`	XST project file for the example design.

Back to Top

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 4-6: **Results Directory**

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

Back to Top

<component name>/simulation

simulation/dsport

The dsport directory contains the Root Port Bus Functional model files provided with the core.

Table 4-7: **dsport Directory**

Name	Description
`<project_dir>/<component_name>/simulation/dsport`	

`gtx_tx_sync_rate_v6.v`
`gtx_wrapper_v6.v`
`pci_exp_expect_tasks.v`
`pci_exp_usrapp_cfg.v`
`pci_exp_usrapp_com.v`
`pci_exp_usrapp_pl.v`
`pci_exp_usrapp_rx.v`
`pci_exp_usrapp_tx.v`
`pcie_2_0_rport_v6.v`
`pcie_2_0_v6.v`
`pcie_bram_top_v6.v`
`pcie_bram_v6.v`
`pcie_brams_v6.v`
`pcie_clocking_v6.v`
`pcie_gtx_v6.v`
`pcie_pipe_lane_v6.v`
`pcie_pipe_misc_v6.v`
`pcie_pipe_v6.v`
`pcie_reset_delay_v6.v`
`xilinx_pcie_2_0_rport_v6.v`
	Root port model files.

Back to Top

simulation/functional

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

Table 4-8: **Functional Directory**

Name	Description
`<project_dir>/<component_name>/simulation/functional`	
`board_v.f` *and/or* `board_vhd.f`	List of files for RTL simulations.
`simulate_mti.do`	Simulation script for ModelSim.
`simulate_vcs.sh`	Simulation script for VCS.
`simulate_ncsim.sh`	Simulation script for IUS.
`board_common.v`	Contains test bench definitions.
`board.v[hd]`	Top-level simulation module.
`sys_clk_gen_ds.v[hd]`	System differential clock source.
`sys_clk_gen.v[hd]`	System clock source.

Back to Top

simulation/tests

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

Table 4-9: **Tests Directory**

Name	Description
`<project_dir>/<component_name>/simulation/tests`	
`sample_tests1.v` `tests.v`	Test definitions for example test bench.

Back to Top

<component name>/source

This directory contains the sources files for the core.

Table 4-10: **Source Directory**

Name	Description
`<project_dir>/<component_name>/source`	
`<component name>.v[hd]`	Verilog or VHDL top-level wrapper, which instantiates the Endpoint Block, Block RAMs, GT, and clocking resources.

`gtpa1_dual_wrapper_top.v[hd]`
`gtpa1_dual_wrapper_tile.v[hd]`
`gtpa1_dual_wrapper.v[hd]`
	Wrapper for the GTPA1, which configures the transceiver and presents the interfaces required for use with the Integrated Endpoint Block.

Table 4-10: **Source Directory *(Cont'd)***

Name	Description

`pcie_bram_top_s6.v[hd]`
`pcie_brams_s6.v[hd]`
`pcie_bram_s6.v[hd]`
	Configures and instantiates Block RAMs for use with the Integrated Endpoint Block

Back to Top

Chapter 5

Generating and Customizing the Core

The Integrated Endpoint Block for PCI Express Core is a fully configurable and highly customizable solution. The Integrated Endpoint Block for PCI Express is customized using the CORE Generator GUI.

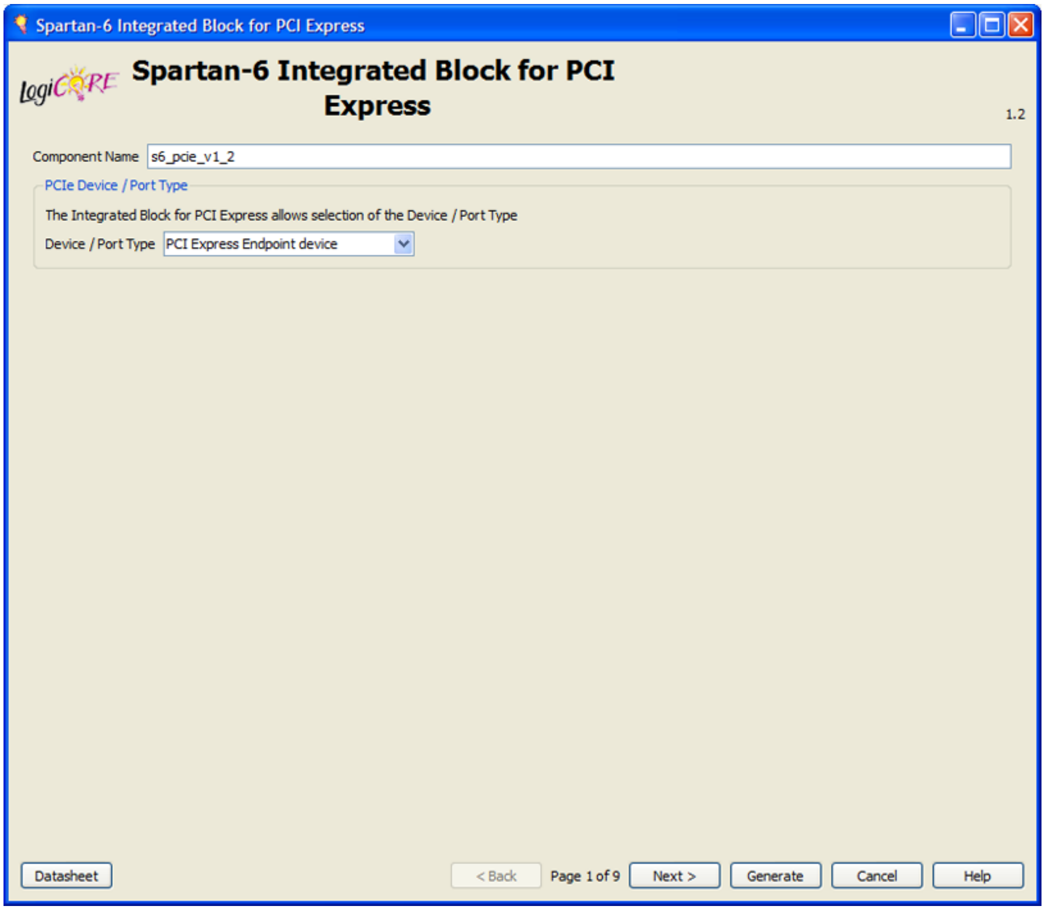
Customizing the Core through CORE Generator

The Integrated Block for PCI Express CORE Generator GUI consists of 9 screens:

- Screen 1: "Basic Parameter Settings"
- Screen 2: "Base Address Registers"
- Screen 3: "PCI Registers"
- Screens 4 and 5: "Configuration Register Settings"
- Screen 6: "Interrupt Capabilities"
- Screen 7: "Power Management Registers"
- Screen 8: "PCI Express Extended Capabilities"
- Screen 9: "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.



Spartan-6 Integrated Block for PCI Express

Component Name: s6_pcie_v1_2

PCIe Device / Port Type

The Integrated Block for PCI Express allows selection of the Device / Port Type

Device / Port Type: PCI Express Endpoint device

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

Component Name

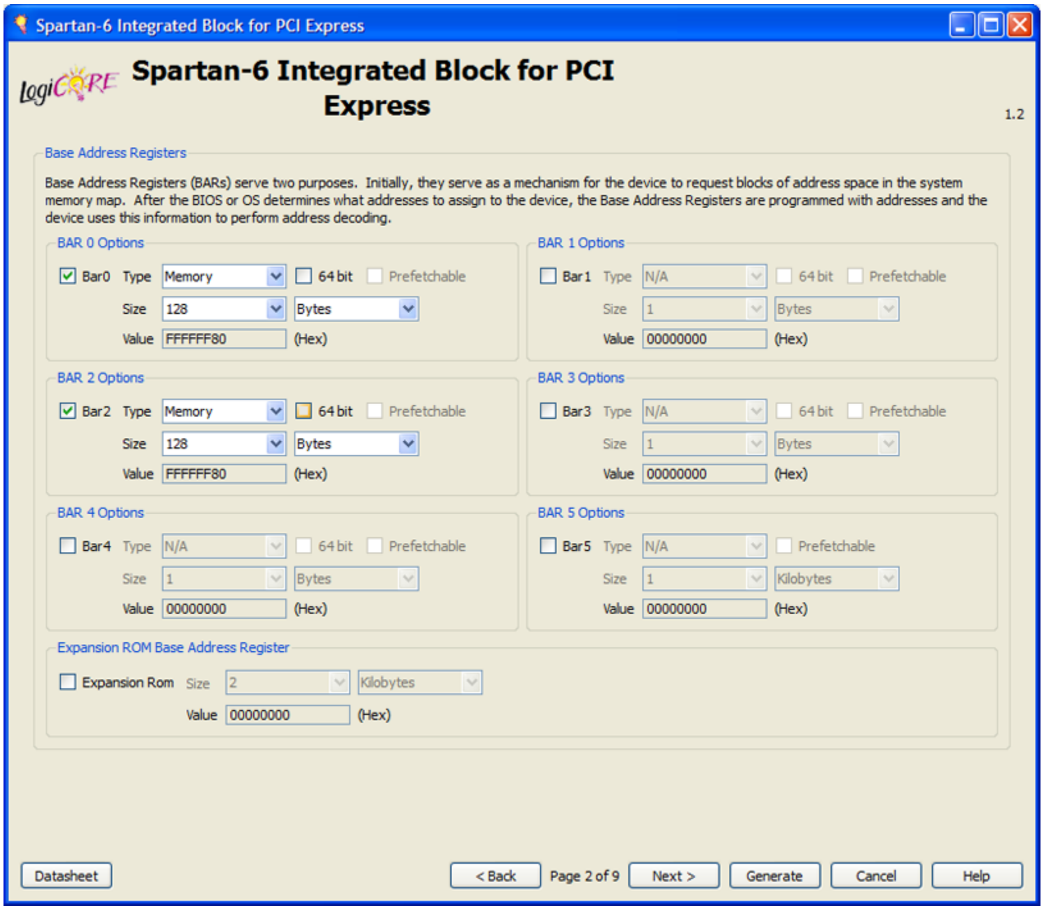
Base name of the output files generated for the core. The name must begin with a letter and can be composed of the following characters: a to z, 0 to 9, and "_."

PCIe Device / Port Type

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

Base Address Registers

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



Spartan-6 Integrated Block for PCI Express 1.2

Base Address Registers

Base Address Registers (BARs) serve two purposes. Initially, they serve as a mechanism for the device to request blocks of address space in the system memory map. After the BIOS or OS determines what addresses to assign to the device, the Base Address Registers are programmed with addresses and the device uses this information to perform address decoding.

BAR 0 Options
☑ Bar0 Type Memory ☐ 64 bit ☐ Prefetchable
Size 128 Bytes
Value FFFFFF80 (Hex)

BAR 1 Options
☐ Bar1 Type N/A ☐ 64 bit ☐ Prefetchable
Size 1 Bytes
Value 00000000 (Hex)

BAR 2 Options
☑ Bar2 Type Memory ☐ 64 bit ☐ Prefetchable
Size 128 Bytes
Value FFFFFF80 (Hex)

BAR 3 Options
☐ Bar3 Type N/A ☐ 64 bit ☐ Prefetchable
Size 1 Bytes
Value 00000000 (Hex)

BAR 4 Options
☐ Bar4 Type N/A ☐ 64 bit ☐ Prefetchable
Size 1 Bytes
Value 00000000 (Hex)

BAR 5 Options
☐ Bar5 Type N/A ☐ Prefetchable
Size 1 Kilobytes
Value 00000000 (Hex)

Expansion ROM Base Address Register
☐ Expansion Rom Size 2 Kilobytes
Value 00000000 (Hex)

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

Base Address Register Overview

The Endpoint for PCIe supports up to six 32-bit Base Address Registers (BARs) or three 64-bit BARs, and the Expansion ROM BAR. BARs may be one of two sizes:

- **32-bit BARs**: The address space can be as small as 128 bytes for Memory or 16 bytes for I/O, 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 the following 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 may only be 32-bit; the Prefetchable option does not apply to I/O BARs.

  - *Memory*: Memory BARs may be either 64-bit or 32-bit and may 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**
  - *Memory*: When Memory and 64-bit are not selected, the size can range from 128 bytes to 2 gigabytes. When Memory and 64-bit are selected, the size may range between 128 bytes and 8 exabytes.
  - *I/O*: When selected, the size can range from 16 bytes to 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 may be a value from 2 kilobytes to 4 gigabytes.

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 4Kbytes 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 will not be destroyed by reading, as from a RAM). Byte write operations can be merged into a single double-word write, when applicable.

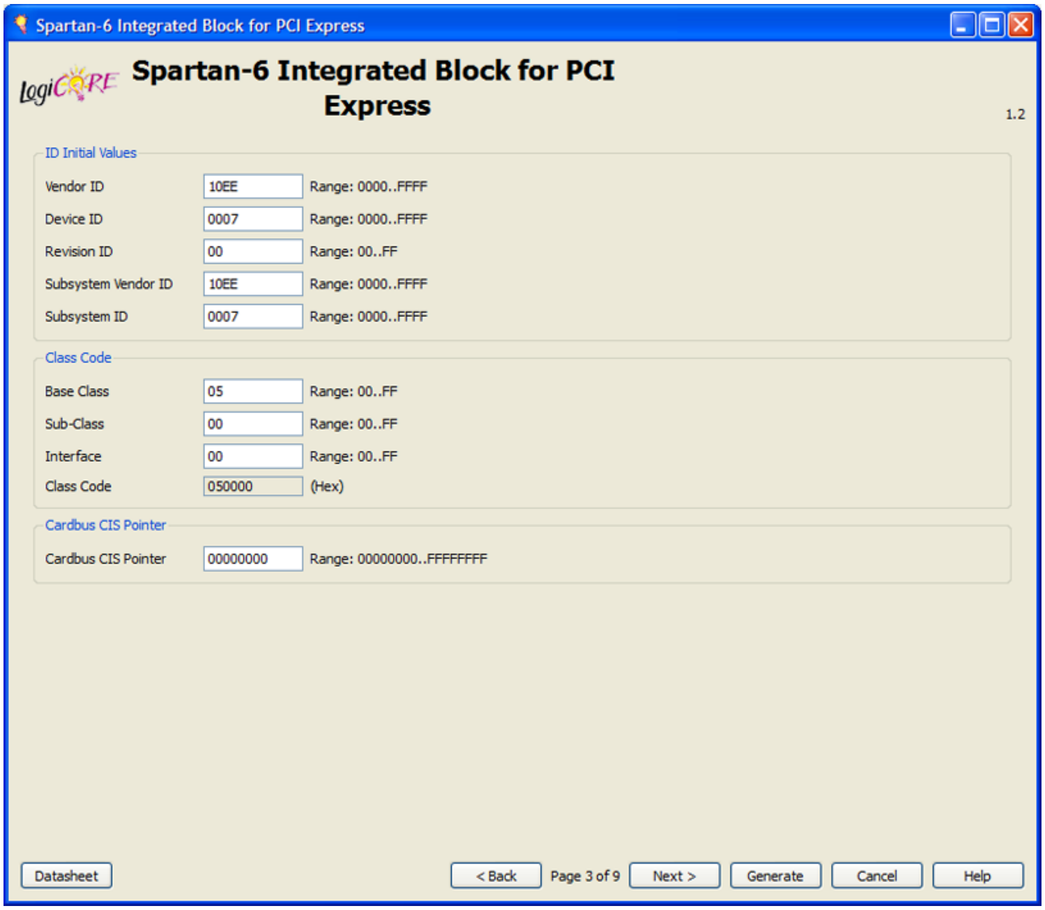
When configuring the core as an Endpoint for PCIe (non-Legacy), 64-bit addressing must be supported for all BARs (except BAR5) that have the prefetchable bit set. 32-bit addressing is permitted for all BARs that do not have the prefetchable bit set. The prefetchable bit related requirement does not apply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

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 5-3 is used to customize the IP initial values, class code and Cardbus CIS pointer information.


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
Datasheet
< Back
Page 3 of 9
Next >
Generate
Cancel
Help

ID Initial Values

Vendor ID	10EE	Range: 0000..FFFF
Device ID	0007	Range: 0000..FFFF
Revision ID	00	Range: 00..FF
Subsystem Vendor ID	10EE	Range: 0000..FFFF
Subsystem ID	0007	Range: 0000..FFFF

Class Code

Base Class	05	Range: 00..FF
Sub-Class	00	Range: 00..FF
Interface	00	Range: 00..FF
Class Code	050000	(Hex)

Cardbus CIS Pointer

Cardbus CIS Pointer	00000000	Range: 00000000..FFFFFFFF

Figure 5-3: **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 is 0007h. 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. Note that 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; default value is 0007h. Note that setting the value 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.

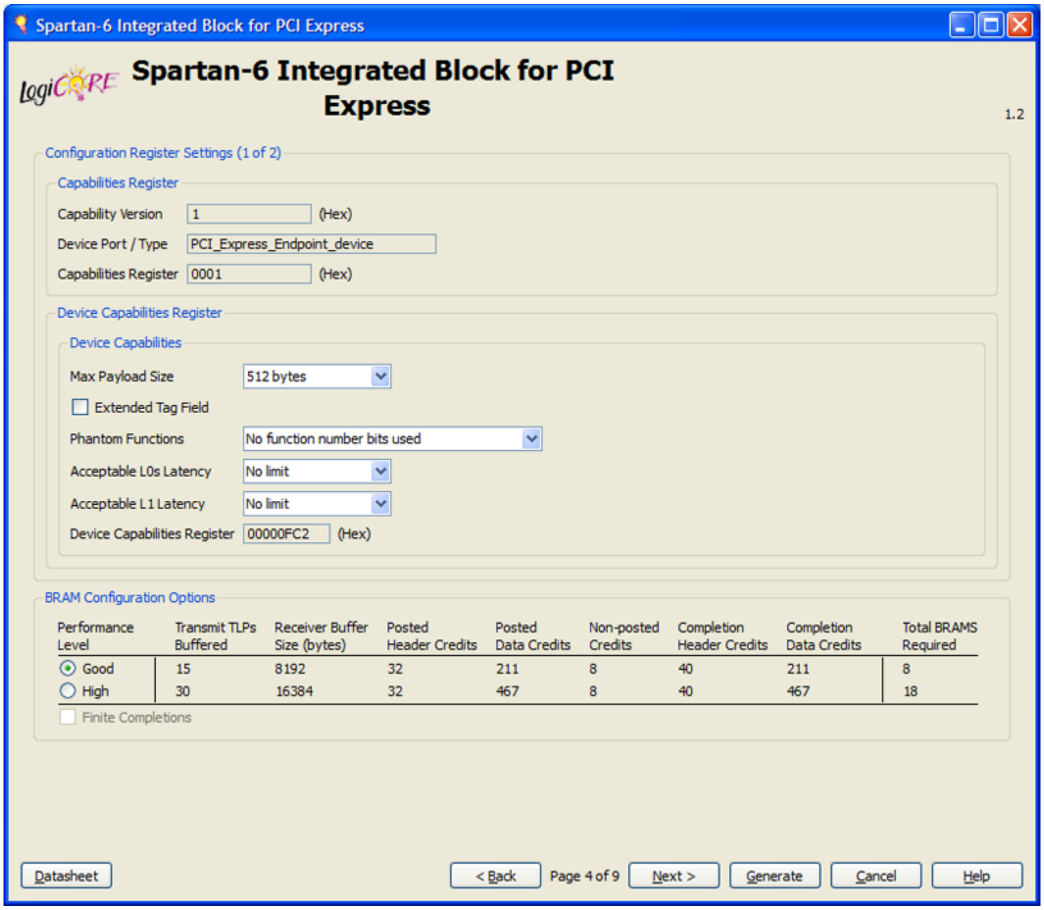
Class code encoding can be found at www.pcisig.com.

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; value range is 0000_0000h-FFFF_FFFFh.

Configuration Register Settings

The Configuration Registers screens shown in Figure 5-4 and Figure 5-5 show the options for the Device Capabilities and Registers, the Block RAM Configuration Options, the Link Capabilities Register, and the Link Status Register.


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
Datasheet
< Back
Page 4 of 9
Next >
Generate
Cancel
Help

Configuration Register Settings (1 of 2)

Capabilities Register

Capability Version: 1 (Hex)
Device Port / Type: PCI_Express_Endpoint_device
Capabilities Register: 0001 (Hex)

Device Capabilities Register

Device Capabilities

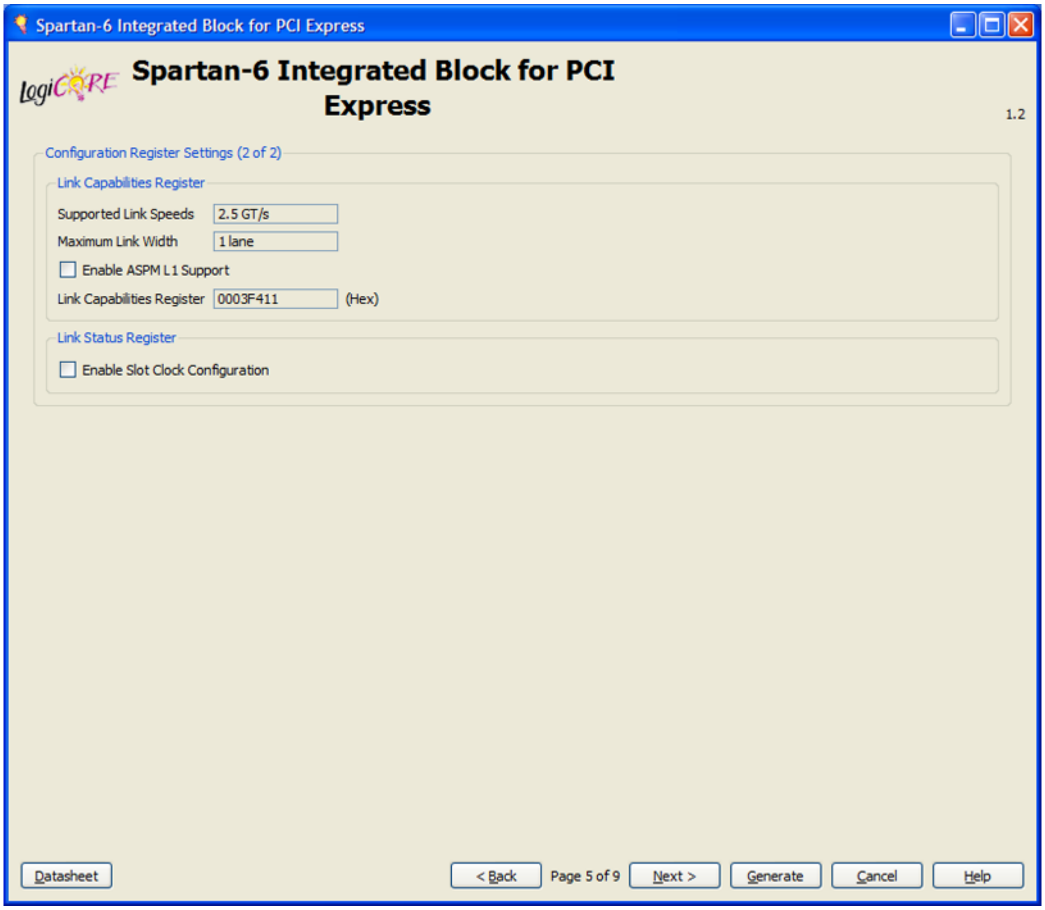
Max Payload Size: 512 bytes
☐ Extended Tag Field
Phantom Functions: No function number bits used
Acceptable L0s Latency: No limit
Acceptable L1 Latency: No limit
Device Capabilities Register: 00000FC2 (Hex)

BRAM Configuration Options

Performance Level	Transmit TLPs Buffered	Receiver Buffer Size (bytes)	Posted Header Credits	Posted Data Credits	Non-posted Credits	Completion Header Credits	Completion Data Credits	Total BRAMS Required
● Good	15	8192	32	211	8	40	211	8
○ High	30	16384	32	467	8	40	467	18

☐ Finite Completions

Figure 5-4: **Screen: Configuration Settings**


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
Datasheet
< Back
Page 5 of 9
Next >
Generate
Cancel
Help

Configuration Register Settings (2 of 2)

Link Capabilities Register

Supported Link Speeds: 2.5 GT/s

Maximum Link Width: 1 lane

☐ Enable ASPM L1 Support

Link Capabilities Register: 0003F411 (Hex)

Link Status Register

☐ Enable Slot Clock Configuration

Figure 5-5: **Screen 5: Configuration Settings**

Capabilities Register

- **Capability Version**: Indicates PCI-SIG defined PCI Express capability structure version number; this value cannot be changed.
- **Device Port Type**: Indicates the PCI Express logical device type.
- **Capabilities Register**: Displays the value of the Capabilities register presented by the Integrated Endpoint 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 version 1.1 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 Endpoint Block and is not editable.

Block RAM Configuration Options

- **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 version 1.1*, as endpoints are required to advertise an infinite amount of completion space. Finite completions are not supported in this release of the core.

Link Capabilities Register

This section is used to set the Link Capabilities register.

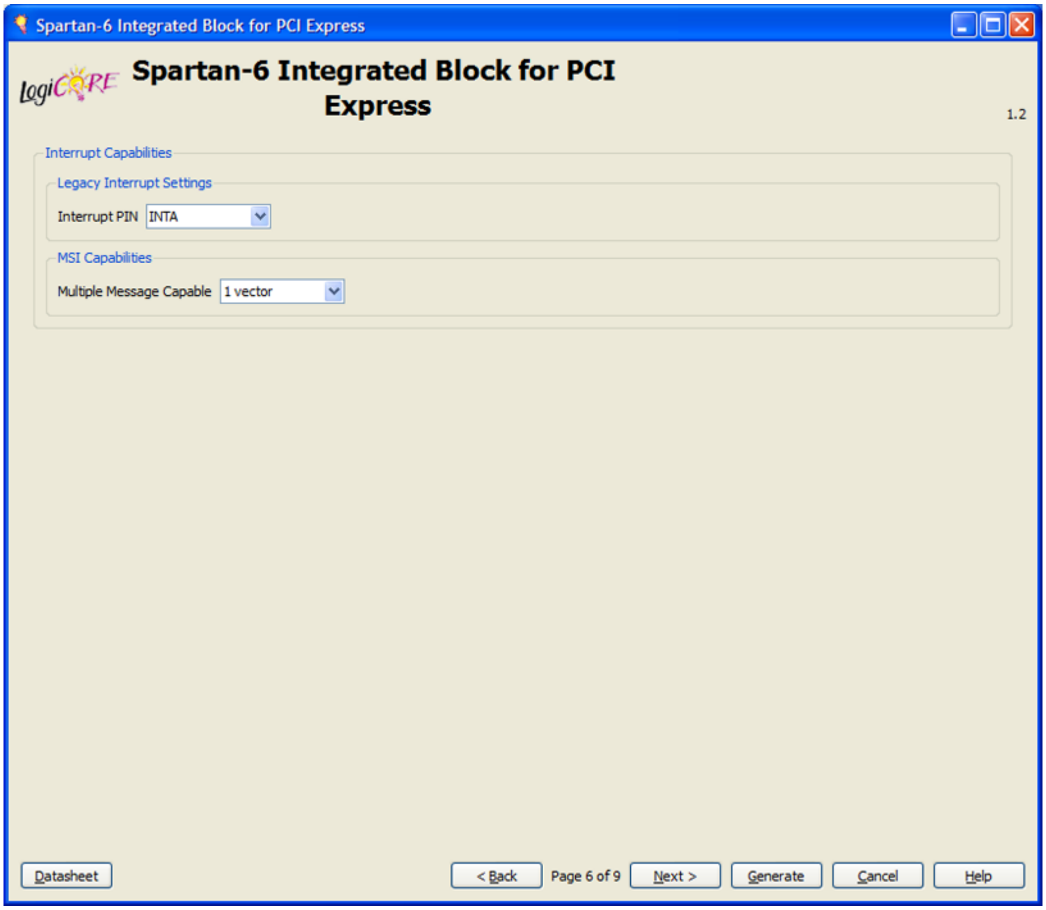
- **Maximum Link Speed**: Indicates the maximum link speed of the given PCI Express Link. This value is set to 2.5 Gbps and is not editable.
- **Maximum Link Width**: This value is set to 1 lane.
- **Enable ASPM L1 Support**: Indicates the level of ASPM supported on the given PCI Express Link. L0s is always supported by the Integrated Endpoint Block core; L1 support is optional and is enabled if this box is checked.
- **Link Capabilities Register**: Displays the value of the Link Capabilities register presented by the Endpoint 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.

Interrupt Capabilities

The Interrupt Settings screen shown in Figure 5-6 sets the Legacy Interrupt Settings and MSI Capabilities.



Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
Interrupt Capabilities
Legacy Interrupt Settings
MSI Capabilities
Datasheet
< Back
Page 6 of 9
Next >
Generate
Cancel
Help

Interrupt PIN: INTA

Multiple Message Capable: 1 vector

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

Legacy Interrupt Settings

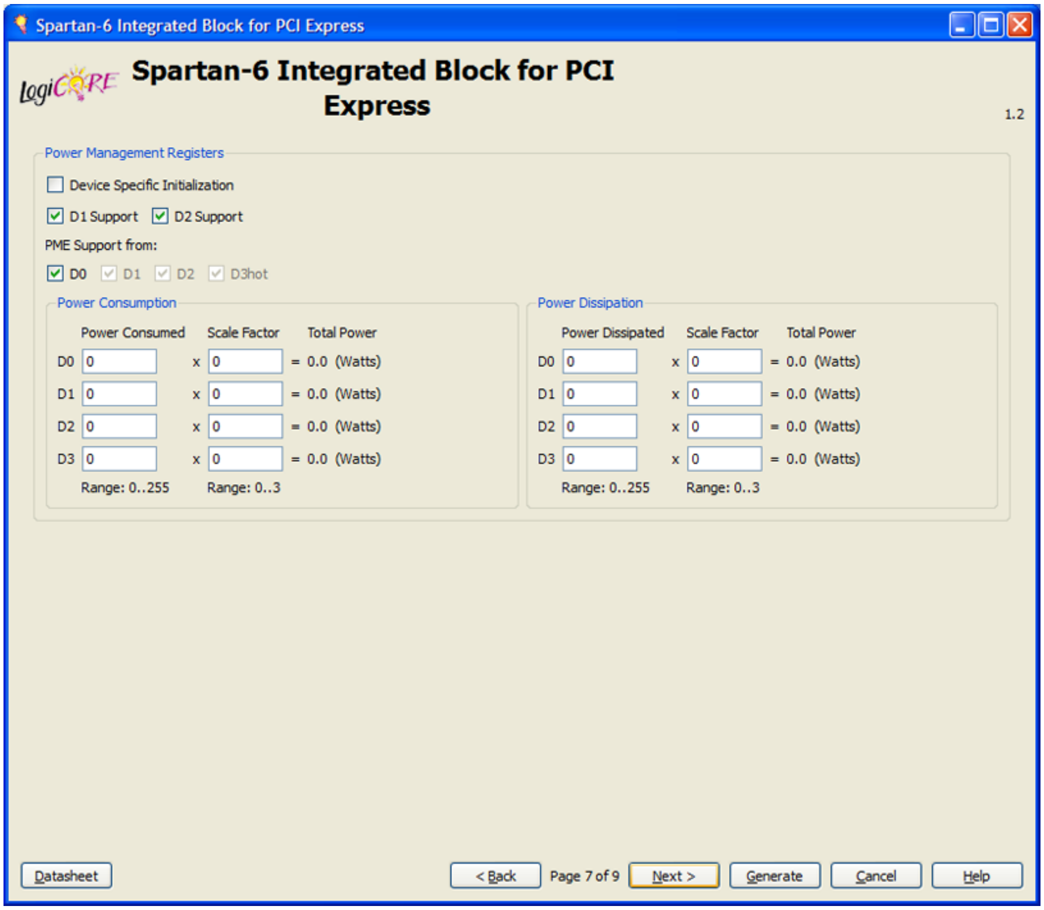
- **Interrupt PIN**: Indicates the mapping for Legacy Interrupt messages. A setting of "None" indicates no Legacy Interrupts are used.

MSI Capabilities

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

Power Management Registers

The Power Management Registers screen shown in Figure 5-7 includes settings for the Power Management Registers, power consumption and power dissipation options.


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
Datasheet
< Back
Page 7 of 9
Next >
Generate
Cancel
Help

Power Management Registers

☐ Device Specific Initialization

☑ D1 Support ☑ D2 Support

PME Support from:

☑ D0 ☑ D1 ☑ D2 ☑ D3hot

Power Consumption

	Power Consumed		Scale Factor	Total Power
D0	0	x	0	= 0.0 (Watts)
D1	0	x	0	= 0.0 (Watts)
D2	0	x	0	= 0.0 (Watts)
D3	0	x	0	= 0.0 (Watts)
	Range: 0..255		Range: 0..3	

Power Dissipation

	Power Dissipated		Scale Factor	Total Power
D0	0	x	0	= 0.0 (Watts)
D1	0	x	0	= 0.0 (Watts)
D2	0	x	0	= 0.0 (Watts)
D3	0	x	0	= 0.0 (Watts)
	Range: 0..255		Range: 0..3	

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

Power Management Registers

- **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, indicates the power states in which the function can assert PME#. See section 3.2.3 of the *PCI Bus Power Management Interface Specification Revision 1.2*.

Power Consumption

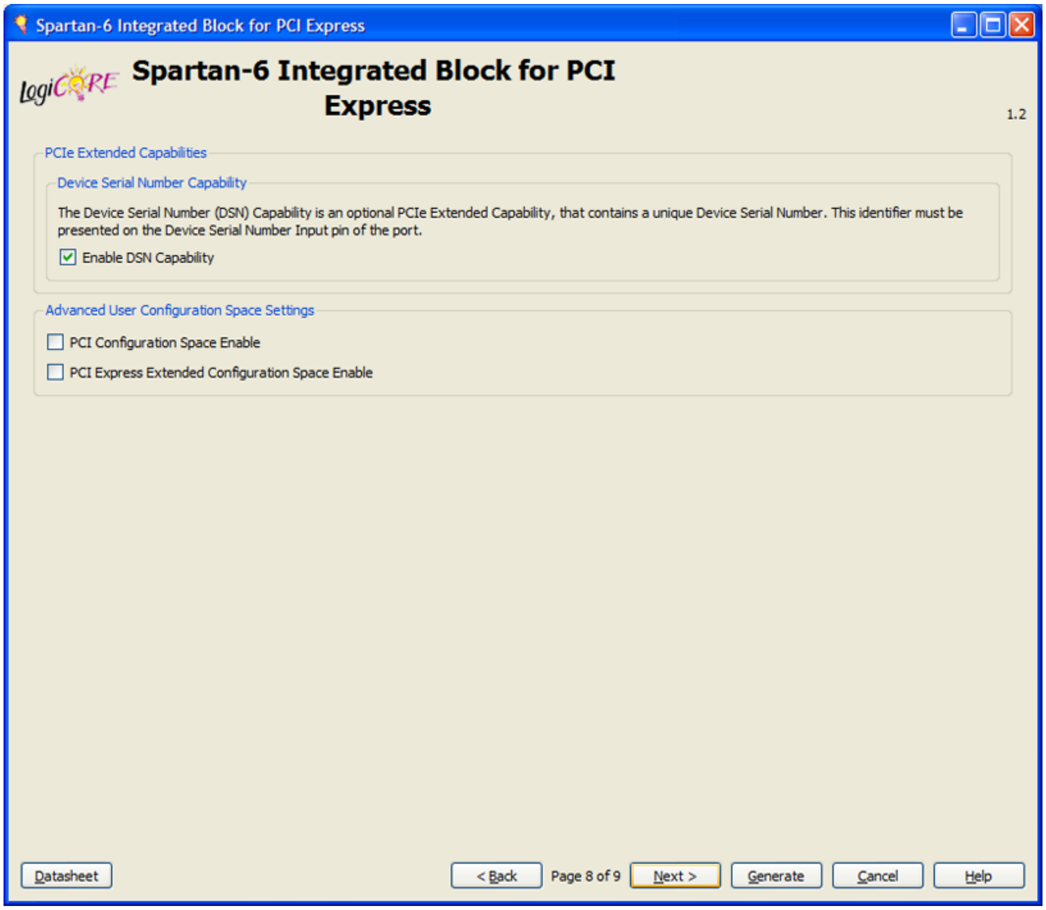
For information about power consumption, see section 3.2.6 of the PCI Bus Power Management Interface Specification Revision 1.2

Power Dissipated

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 5-8 includes settings for Device Serial Number Capability and optional user-defined Configuration capabilities.


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
PCIe Extended Capabilities
Device Serial Number Capability
Advanced User Configuration Space Settings
Datasheet
< Back
Page 8 of 9
Next >
Generate
Cancel
Help

The Device Serial Number (DSN) Capability is an optional PCIe Extended Capability, that contains a unique Device Serial Number. This identifier must be presented on the Device Serial Number Input pin of the port.

☑ Enable DSN Capability

☐ PCI Configuration Space Enable

☐ PCI Express Extended Configuration Space Enable

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

Device Serial Number Capability

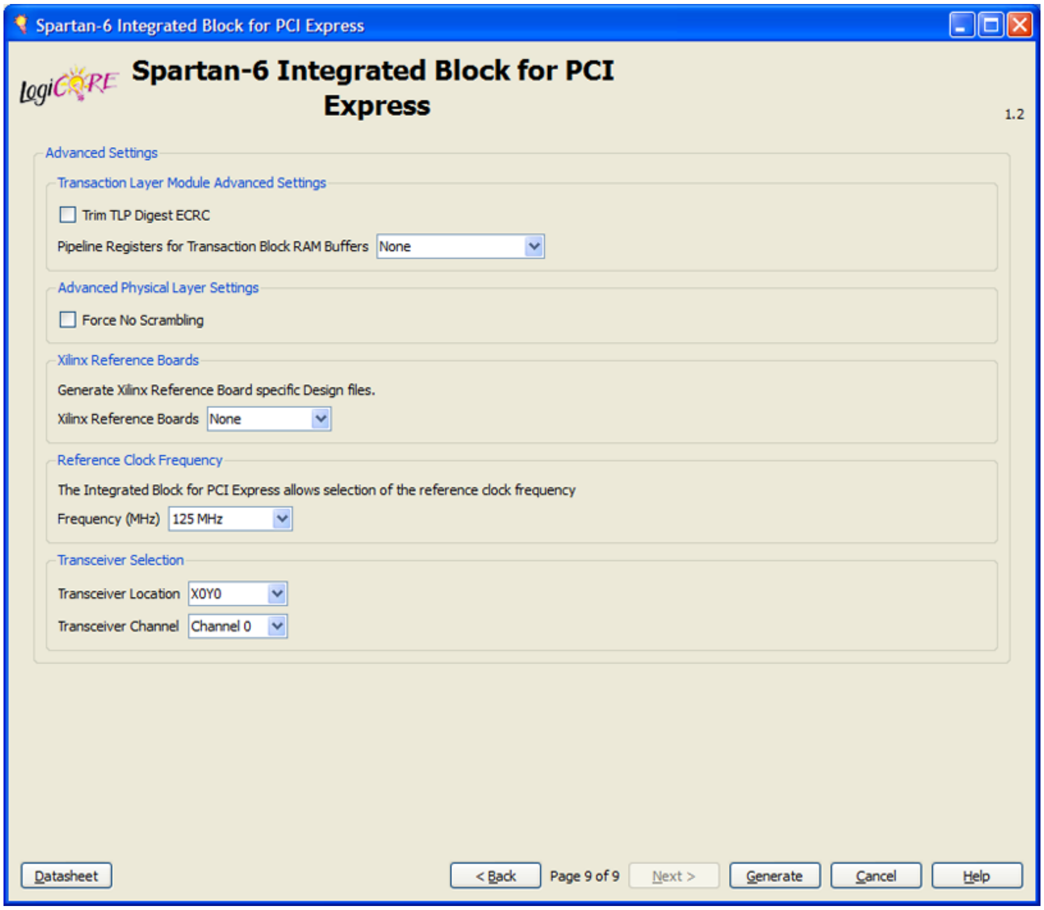
- **Device Serial Number Capability**: An optional PCIe Extended Capability containing a unique Device Serial Number. If enabled, the core presents the Device Serial Number Capability using the value presented on the Device Serial Number input pin of the port. If disabled, no Device Serial Number Extended Capability is presented.

User Defined Configuration Capabilities

- **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 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.

Advanced Settings

The Advanced Settings screen shown in Figure 5-9 includes settings for Transaction Layer, Physical Layer, Reference Clock Frequency and Xilinx Reference Boards options.


Spartan-6 Integrated Block for PCI Express
LogiCORE
Spartan-6 Integrated Block for PCI Express
1.2
Datasheet
< Back
Page 9 of 9
Next >
Generate
Cancel
Help

Advanced Settings

Transaction Layer Module Advanced Settings

☐ Trim TLP Digest ECRC

Pipeline Registers for Transaction Block RAM Buffers: None

Advanced Physical Layer Settings

☐ Force No Scrambling

Xilinx Reference Boards

Generate Xilinx Reference Board specific Design files.

Xilinx Reference Boards: None

Reference Clock Frequency

The Integrated Block for PCI Express allows selection of the reference clock frequency

Frequency (MHz): 125 MHz

Transceiver Selection

Transceiver Location: X0Y0

Transceiver Channel: Channel 0

Figure 5-9: **Screen 9: Advanced Settings 1**

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.

- **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.

Advanced Physical Layer

- **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.

Xilinx Reference Boards

Selecting this option enables the generation of Xilinx Reference Board specific design files. Selecting the SP605 board configures the Reference Clock Frequency, Transceiver Location and Transceiver Channel corresponding with the PCI Express edge connector on the reference board. It also sets the corresponding pin locations in the UCF file.

Reference Clock Frequency

Selects the frequency of the reference clock provided on sys_clk. For important information about clocking the Spartan-6 Integrated Block for PCI Express, see "Clocking and Reset of the Integrated Endpoint Block Core," page 111.

Transceiver Selection

- **Transceiver Location**: Selects the GTPA1_DUAL location for the PCI Express Link.
- **Transceiver Channel**: Selects the channel within the GTPA1_DUAL.

Chapter 6

Designing with the Core

This chapter provides design instructions for the Spartan®-6 FPGA Integrated Endpoint 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 v1.1*, Chapter 2, Transaction Layer Specification.

This chapter includes the following design guidelines:

- "Transmitting Outbound Packets"
- "Receiving Inbound Packets"
- "Accessing Configuration Space Registers"
- "Additional Packet Handling Requirements"
- "Power Management"
- "Generating Interrupt Requests"
- "Clocking and Reset of the Integrated Endpoint Block Core"

TLP Format on 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 6-1 represents a typical 32-bit addressable Memory Write Request TLP (as illustrated in Chapter 2 of the specification).

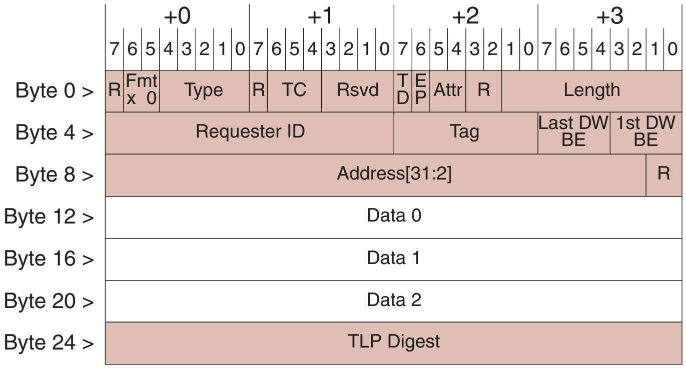


Figure 6-1: **PCI Express Base Specification Byte Order**

When using the 32-bit Transaction interface, packets are arranged on the 32-bit data path in the same order as shown in Figure 6-1. Byte 0 of the packet appears on trn_td[31:24] (outbound) or trn_rd[31:24] (inbound) of the first DWORD, byte 1 on trn_td[23:16] or trn_rd[23:16], and so forth. Byte 4 of the packet then appears on trn_td[31:24] or trn_rd[31:24] of the second DWORD. 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*.

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 DWORD 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 Endpoint Block does not support AER, users can still implement ECRC for custom solutions that do not require *PCI Express Base Specification* Compliance.

Transmitting Outbound Packets

Basic TLP Transmit Operation

The Endpoint for PCIe automatically transmits the following types of packets:

- Completions to a remote device in response to Configuration Space requests.
- Error-message responses to inbound requests 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 the following 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.

  Note: For important information about accessing user-implemented Configuration Space while in a low-power state, see "Power Management," page 106.

The Integrated Block for PCI Express Endpoint core notifies the User Application of pending internally generated TLPs that will be arbitrating for the transmit data path by asserting `trn_tcfg_req_n` (0b). The User Application can choose to give priority to core-generated TLPs by driving `trn_tcfg_gnt_n` asserted (0b) permanently, without regards to `trn_tcfg_req_n`. Doing so will prevent 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 holding `trn_tcfg_gnt_n` de-asserted (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, as this may cause a completion time-out in the Requester. See the PCI Express Base Specification for more information on the Completion Timeout Mechanism.

Table 2-9, page 35 defines the transmit-direction Transaction interface signals. To transmit a TLP, the User Application must perform the following 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 DWORD on `trn_td[31:0]` when it is ready to transmit data.
2. The User Application asserts `trn_tsrc_rdy_n` and presents the remainder of the TLP DWORDs on `trn_td[31:0]` 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 DWORD of data.
4. At the next clock cycle, the User Application deasserts `trn_tsrc_rdy_n` to signal the end of valid transfers on `trn_td[31:0]`.

Figure 6-2 illustrates a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request.

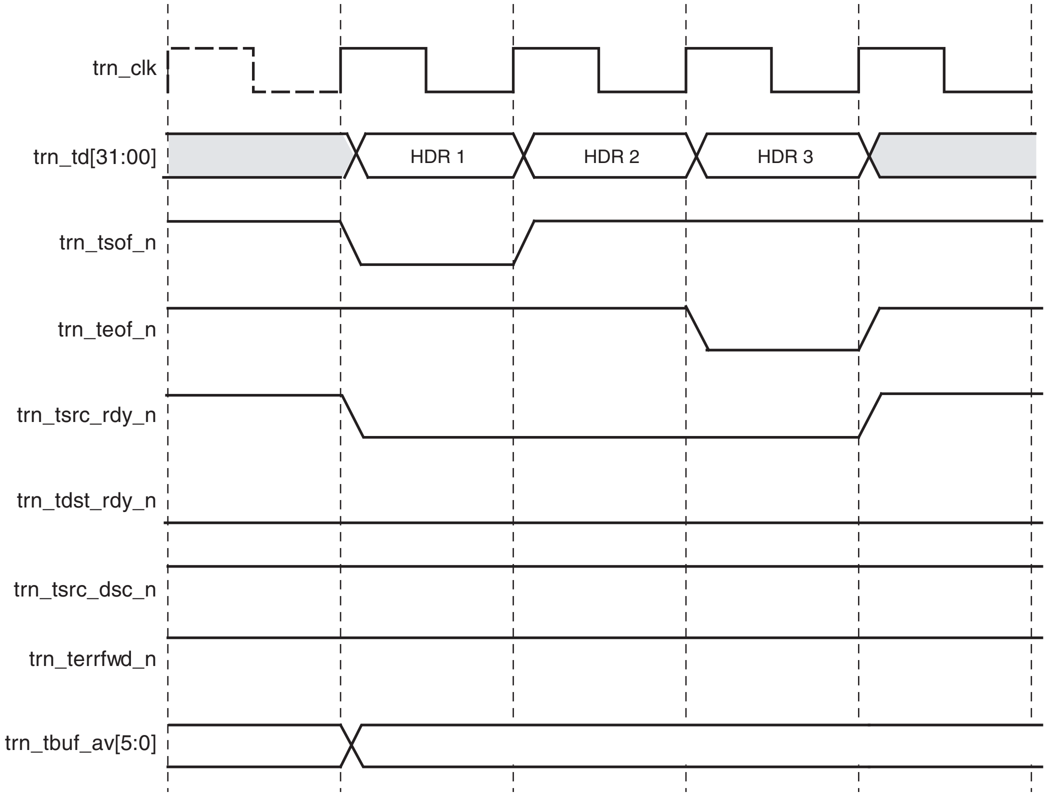


Figure 6-2: **TLP 3-DW Header without Payload**

Figure 6-3 illustrates a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request.

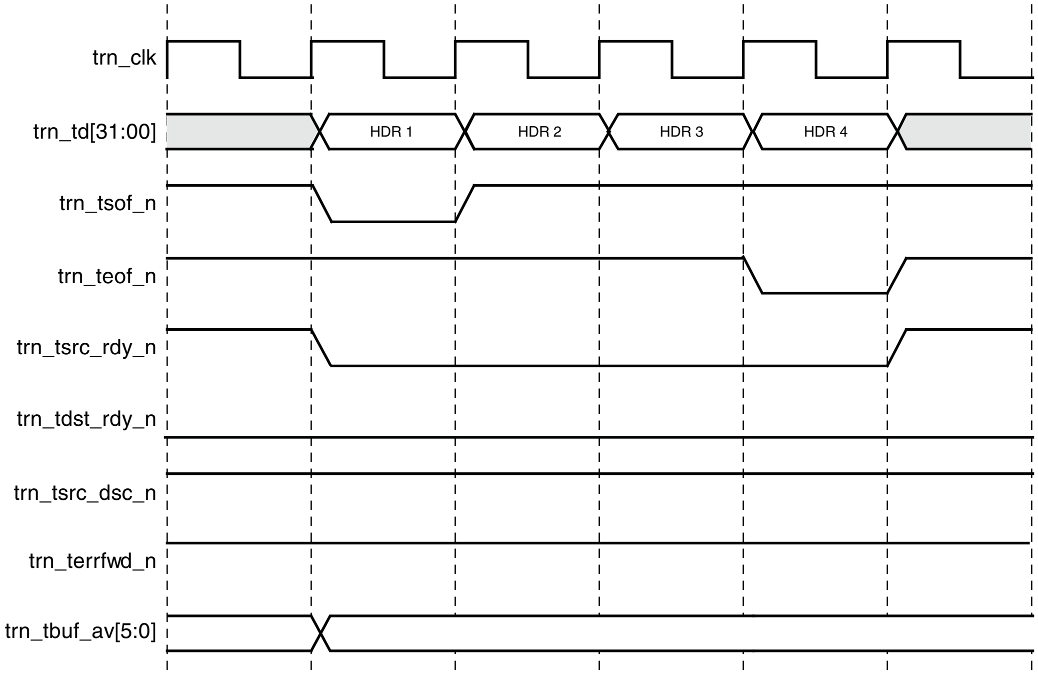


Figure 6-3: **TLP 4-DW Header without Payload**

Figure 6-4 illustrates a 3-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request.

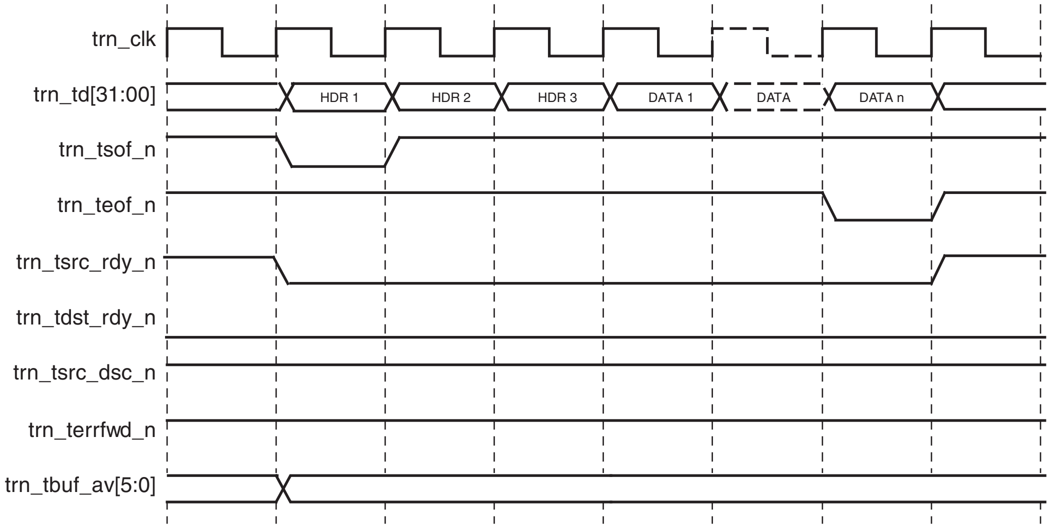


Figure 6-4: **TLP with 3-DW Header with Payload**

Figure 6-5 illustrates a 4-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request.

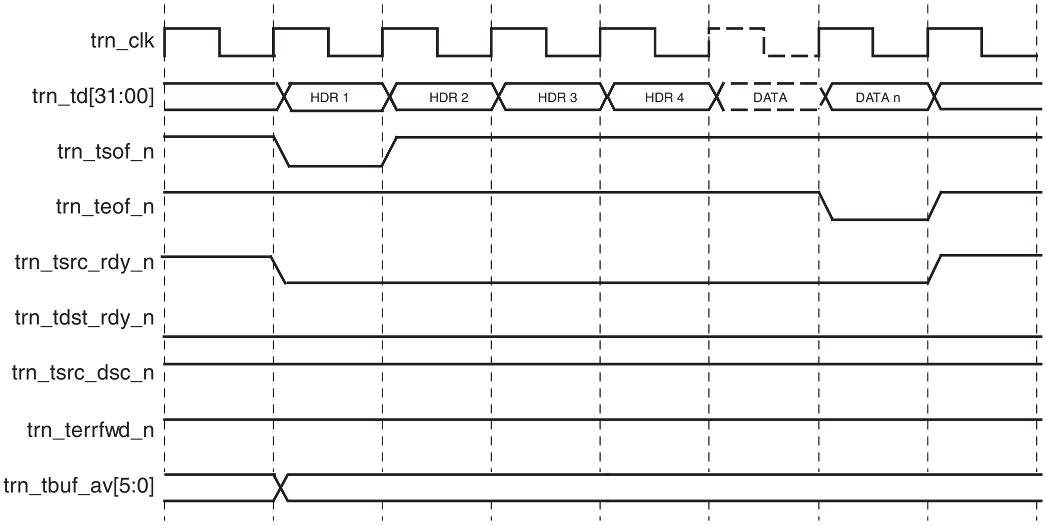


Figure 6-5: **TLP with 4-DW Header with Payload**

Presenting Back-to-Back Transactions on the Transmit Interface

The User Application may present back-to-back TLPs on the transmit Transaction interface to maximize bandwidth utilization. Figure 6-6 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.

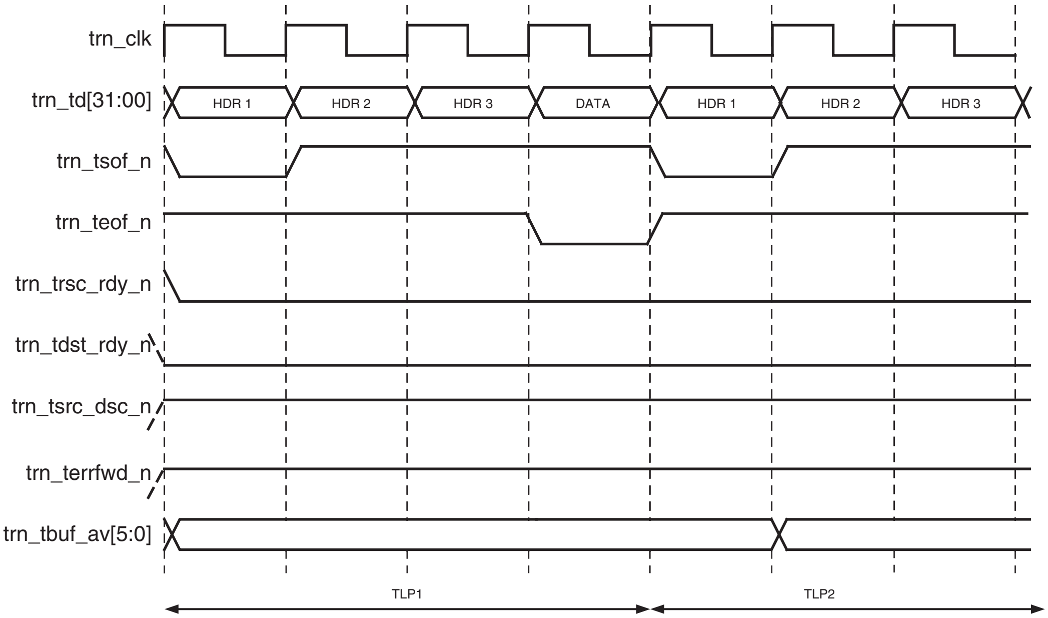


Figure 6-6: **Back-to-back Transaction on Transmit Interface**

Source Throttling on the Transmit Data Path

The Transaction interface lets the User Application throttle back if it has no data present on `trn_td[31:0]`. 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[31:0]`. Figure 6-7 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. The User Application should not assert `trn_tsrc_rdy_n` if `trn_tstr_n` is asserted.

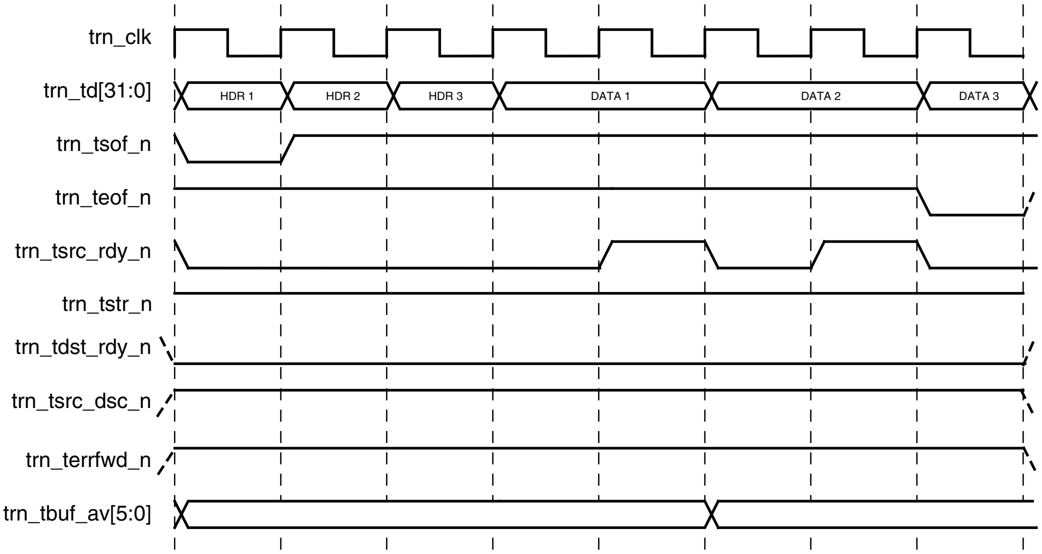


Figure 6-7: **Source Throttling on the Transmit Data Path**

Destination Throttling of the Transmit Data Path

The core Transaction interface throttles the User Application if there is no space left for a new TLP in its transmit buffer pool. This may 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 6-8 illustrates the deassertion of `trn_tdst_rdy_n` to throttle the User Application when the core's internal transmit buffers are full.

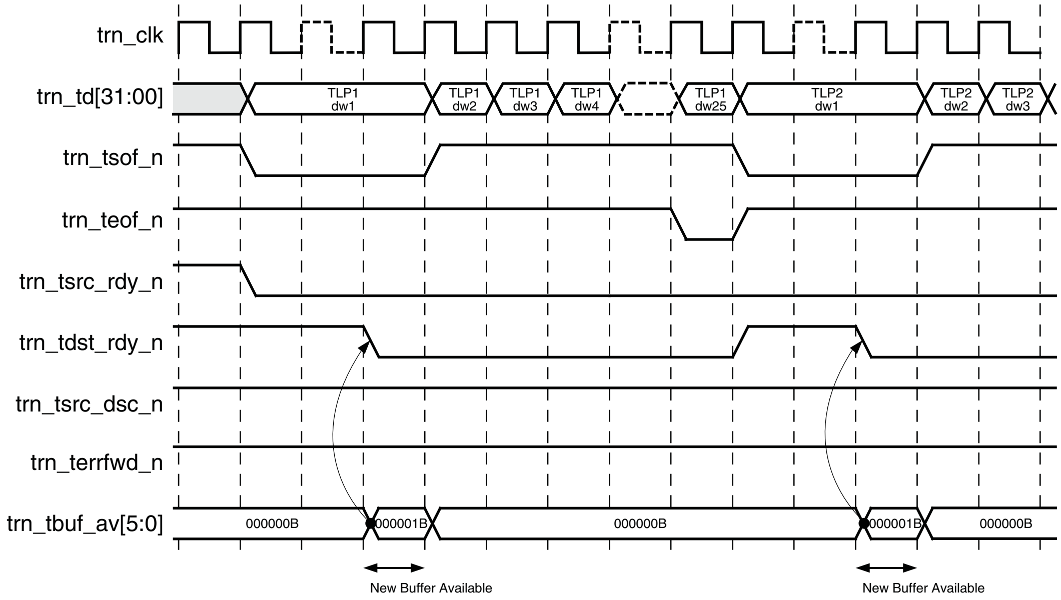


Figure 6-8: **Destination Throttling of the Endpoint Transmit Transaction Interface**

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 the following 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 data path 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 de-assertion 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 6-9 illustrates the User Application discontinuing a packet using `trn_tsrc_dsc_n`. Asserting `trn_teof_n` together with `trn_tsrc_dsc_n` is optional.

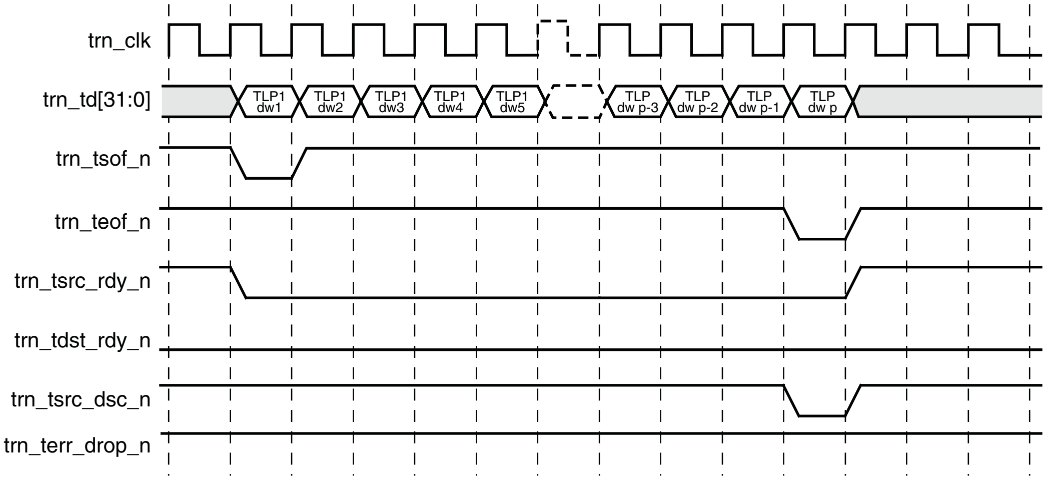


Figure 6-9: **Source Driven Transaction Discontinue on Transmit Interface**

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 left to the user 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 de-asserted 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 6-10 illustrates the core signaling that a packet was discarded using `trn_terr_drop_n` due to a length violation.

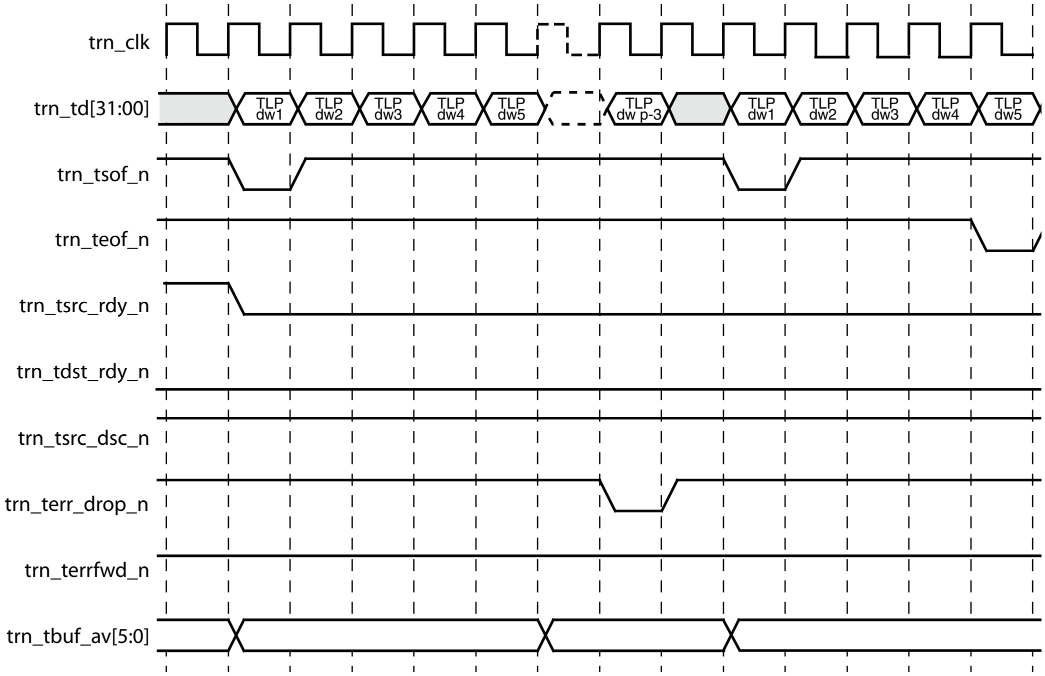


Figure 6-10: **Destination Driven Transaction Discontinue on Transmit Interface**

Packet Data Poisoning on the Transmit Transaction Interface

The User Application can use one of the following mechanisms 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 Transaction interface.
- Assert `trn_terr_fwd_n` for at least 1 valid data transfer cycle any time during the packet transmission, as shown in Figure 6-11. This causes the core to set EP=1 in the TLP header when it transmits the packet onto the PCI Express fabric. This mechanism may be used if the User Application does not know whether a packet may be poisoned at the start of packet transmission. Use of `trn_terr_fwd_n` is not supported for packets when `trn_tstr_n` is asserted (streamed transmit packets).

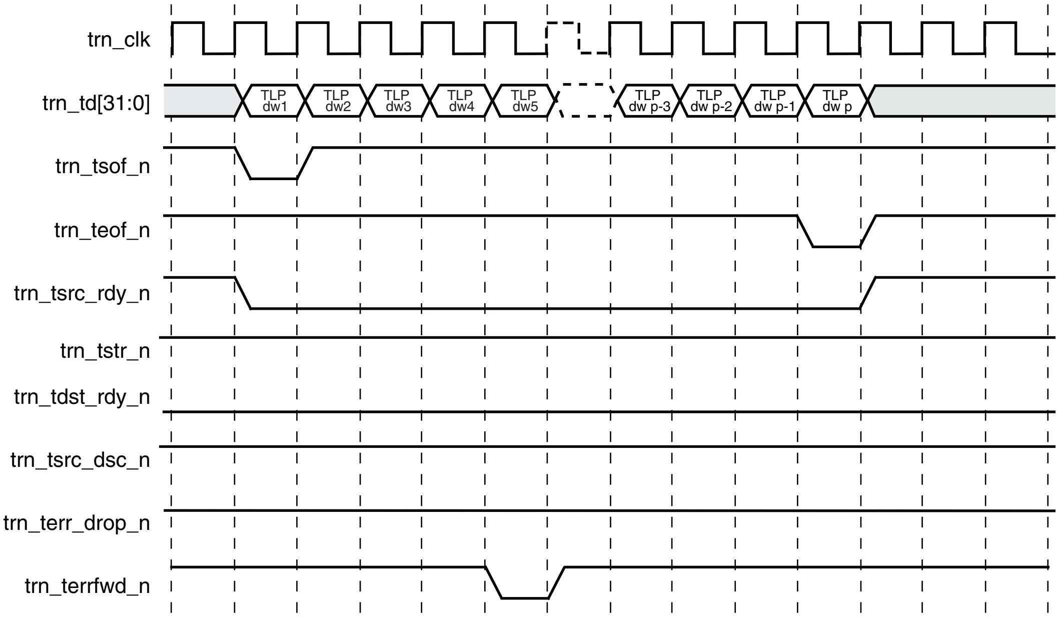


Figure 6-11: **Packet Data Poisoning on the Transmit Transaction Interface**

Streaming mode for Transactions on the Transmit Interface

The Integrated Endpoint 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 de-assert `trn_tsrc_rdy_n` for the duration of the presented TLP. Source throttling of the transaction while in streaming mode of operation will cause the transaction to be dropped (`trn_terr_drop_n` will be asserted) and a nullified TLP to be signaled on the PCI Express link. Figure 6-12 illustrates the streaming mode of operation, where the first TLP is streamed and the second TLP is dropped due to source throttling.

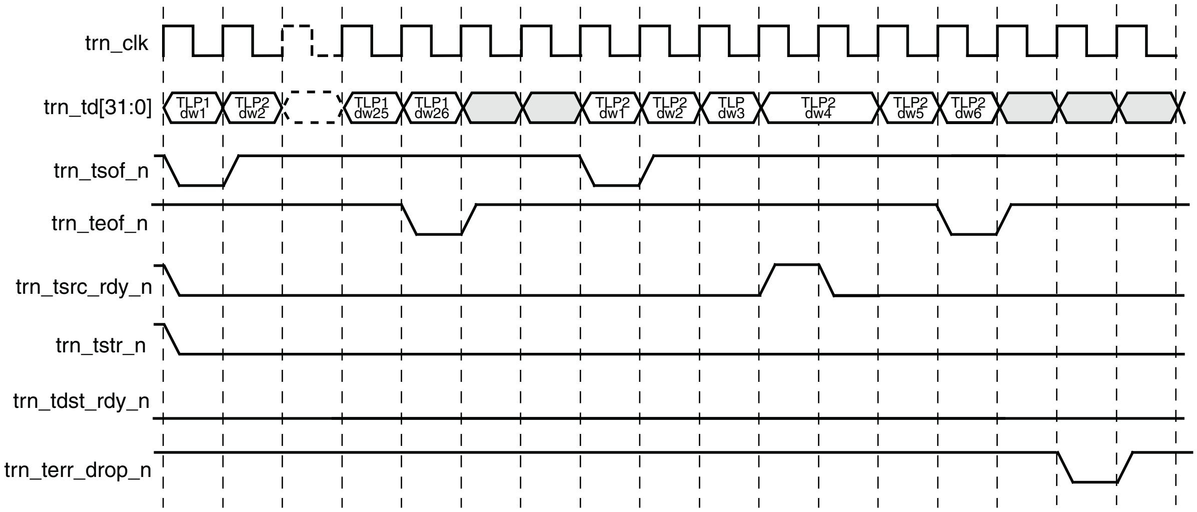


Figure 6-12: **Streaming Mode on the Transmit Interface**

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, for example Completions, Error Messages, and Interrupts, do not have a TLP Digest appended.

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, or 512 bytes, depending on the setting in the CORE Generator GUI. 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 “Accessing Configuration Space Registers,” page 93 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 6-1 defines the number of transmit buffers available and maximum supported payload size for a specific core.

Table 6-1: **Transmit Buffers Available**

Capability Max Payload Size (Bytes)	Performance Level[a]	
	Good (Minimize BRAM Usage)	High (Maximize Performance)
128	13	27
256	14	29
512	15	30

a. Performance level is set through a Core Generator GUI selection.

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. Note that 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 core's transmit buffer 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. Note that 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 may 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.

Receiving Inbound Packets

Basic TLP Receive Operation

Table 2-7, page 30 defines the receive Transaction interface signals. The following sequence of events must occur on the receive Transaction interface for the 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 DWORD on `trn_rd[31:0]`.
3. The core then deasserts `trn_rsof_n`, asserts `trn_rsrc_rdy_n`, and presents TLP DWORDs on `trn_rd[31:0]` for subsequent clock cycles, for which the User Application logic asserts `trn_rdst_rdy_n`.
4. The core then asserts `trn_reof_n` and presents either the last DWORD on `trn_td[31:0]`.
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[31:0]`.

Figure 6-13 illustrates a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request.

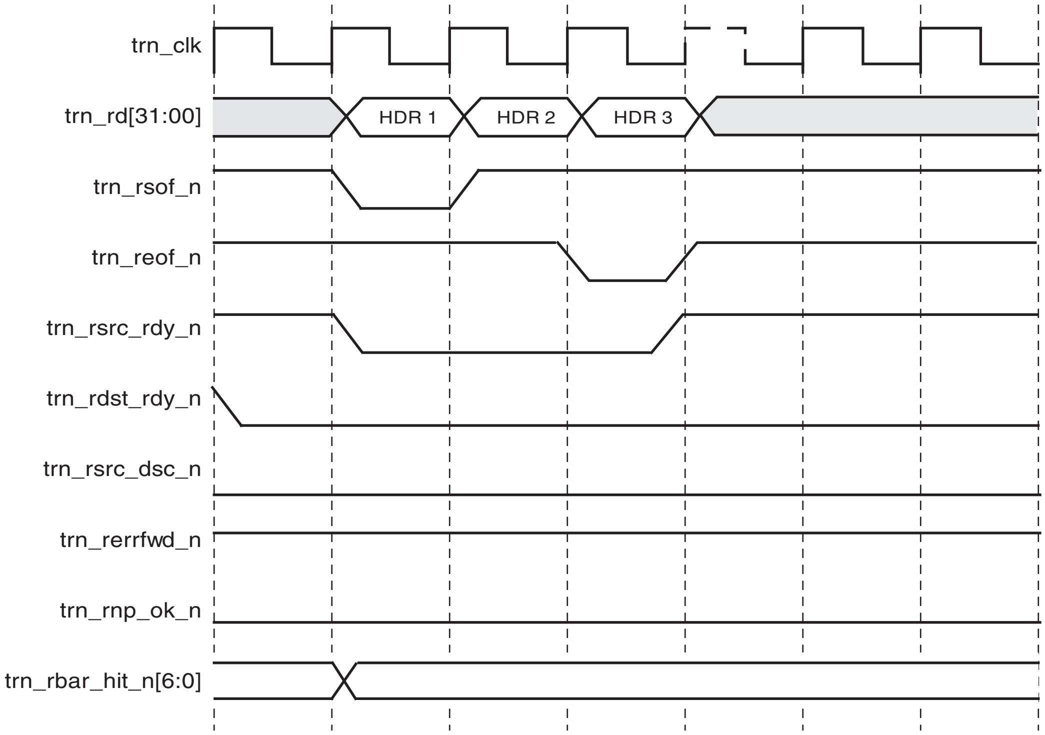


Figure 6-13: **TLP 3-DW Header without Payload**

Figure 6-14 illustrates a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request.

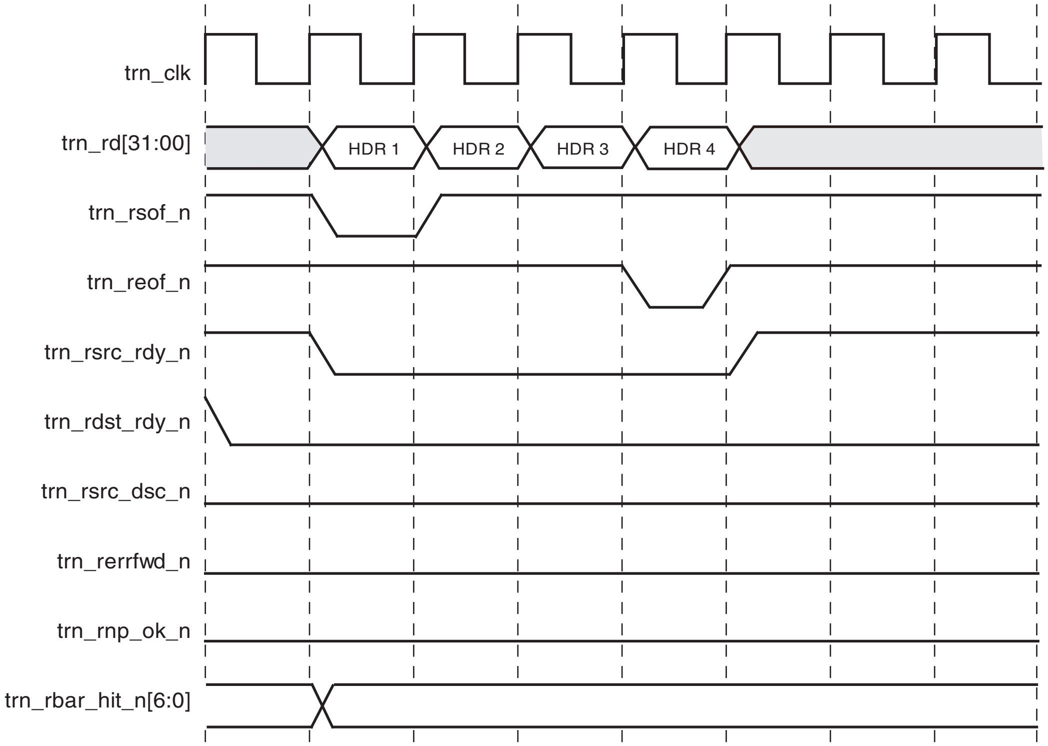


Figure 6-14: **TLP 4-DW Header without Payload**

Figure 6-15 illustrates a 3-DW TLP header with a data payload; an example is a 32-bit addressable Memory Write request.

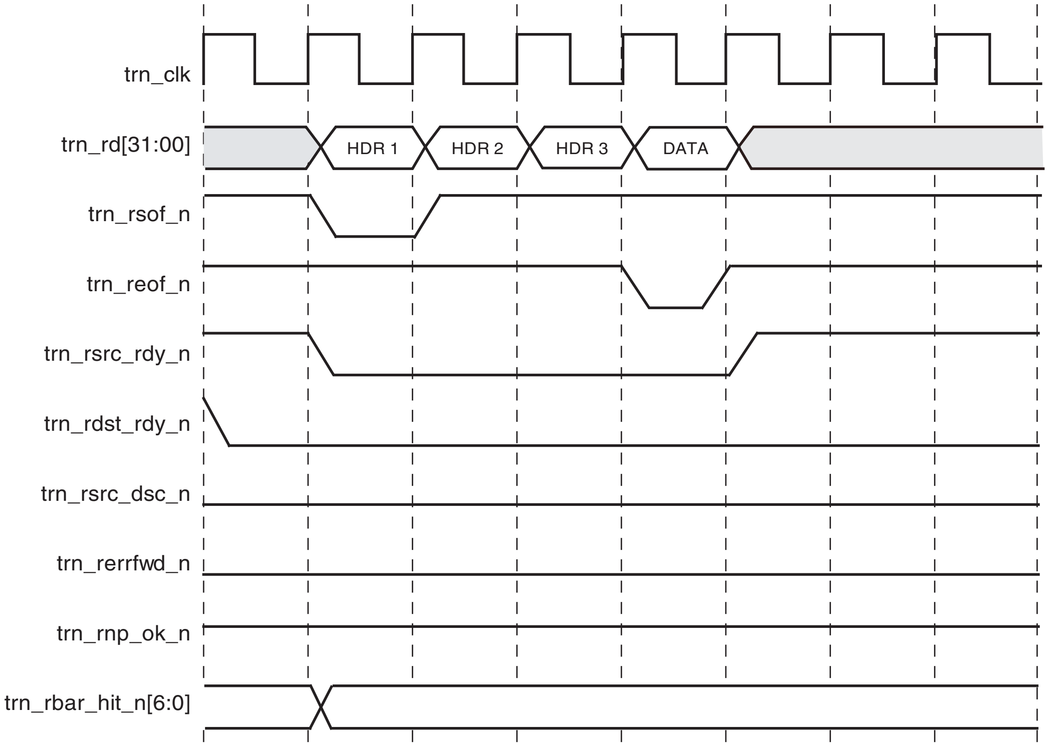


Figure 6-15: **TLP 3-DW Header with Payload**

Figure 6-16 illustrates a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request.

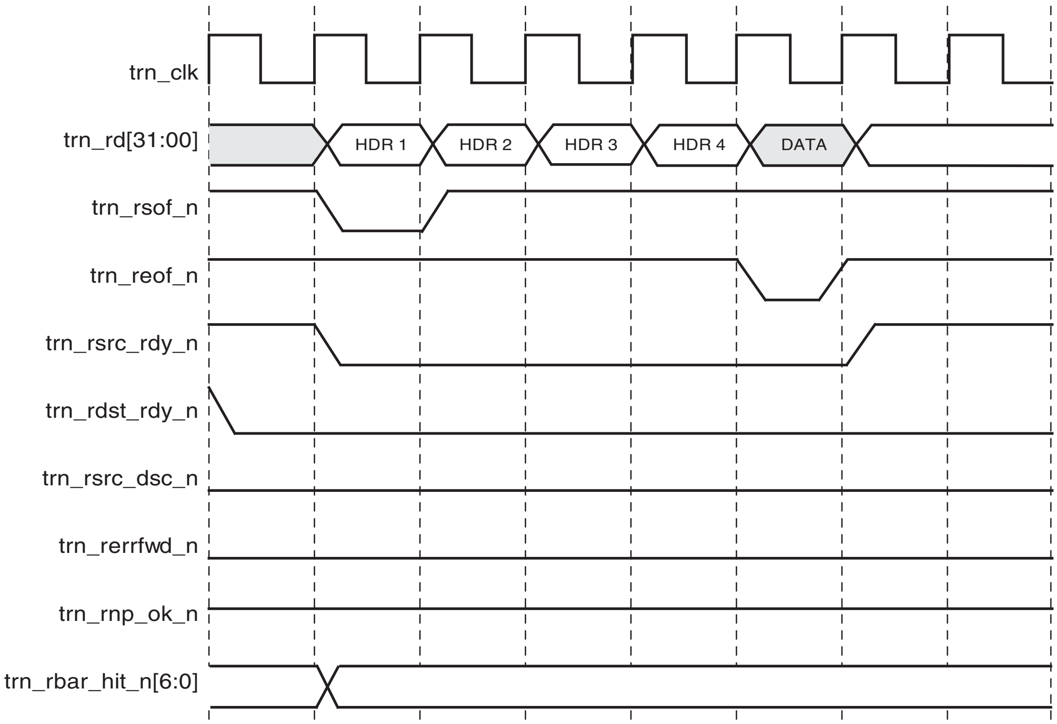


Figure 6-16: **TLP 4-DW Header with Payload**

Throttling the Data Path 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 DWORD on `trn_rd[31:0]`. 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[31:0]`. At that point, the core presents subsequent TLP DWORDs 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 will pause until the user is again ready to receive TLPs.

Figure 6-17 illustrates the core asserting `trn_rsrc_rdy_n` and `trn_rsof_n` along with presenting data on `trn_rd[31:0]`. The User Application logic inserts wait states by deasserting `trn_rdst_rdy_n`. The core will not present the next TLP DWORD until it detects `trn_rdst_rdy_n` assertion. The User Application logic may 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.

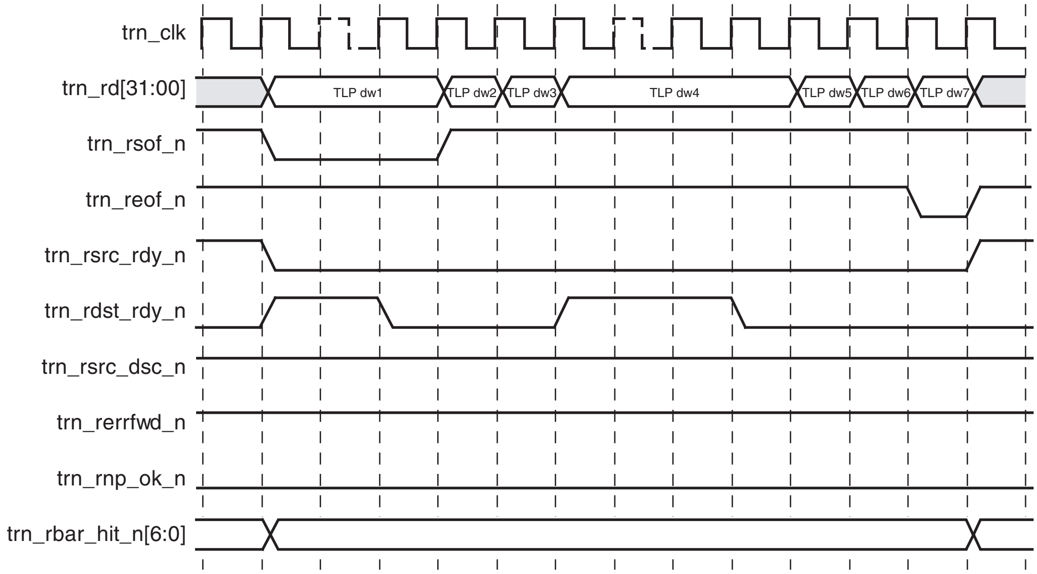


Figure 6-17: **User Application Throttling Receive TLP**

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 interface Transaction interface by the core. The core may assert `trn_rsof_n` for a new TLP at the clock cycle after `trn_reof_n` assertion for the previous TLP. Figure 6-18 illustrates back-to-back TLPs presented on the receive interface.

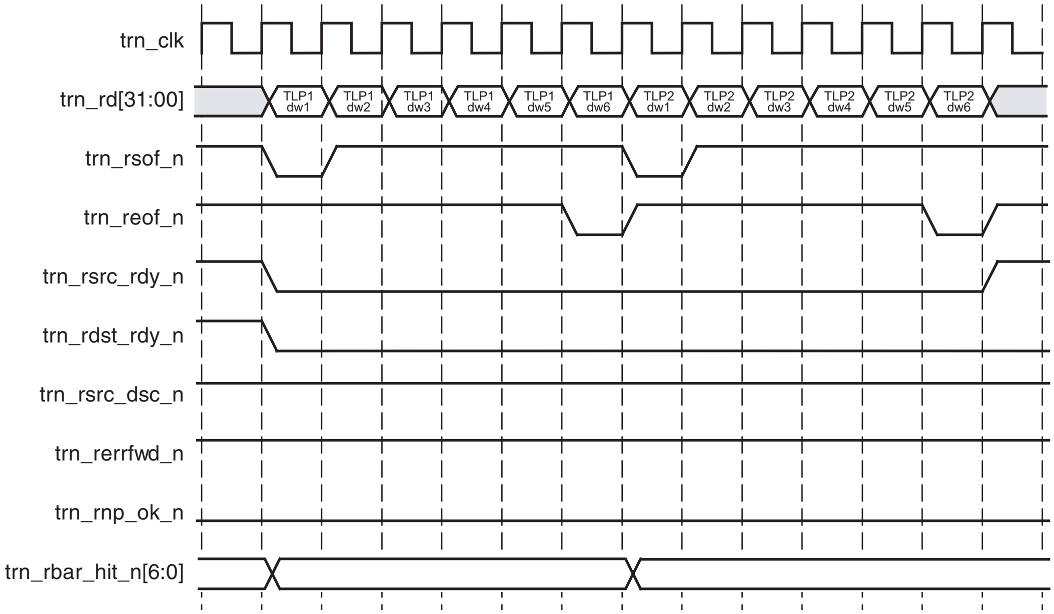


Figure 6-18: **Receive Back-To-Back Transactions**

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 previous section. Figure 6-19 shows an example of using `trn_rdst_rdy_n` to pause the acceptance of the second TLP.

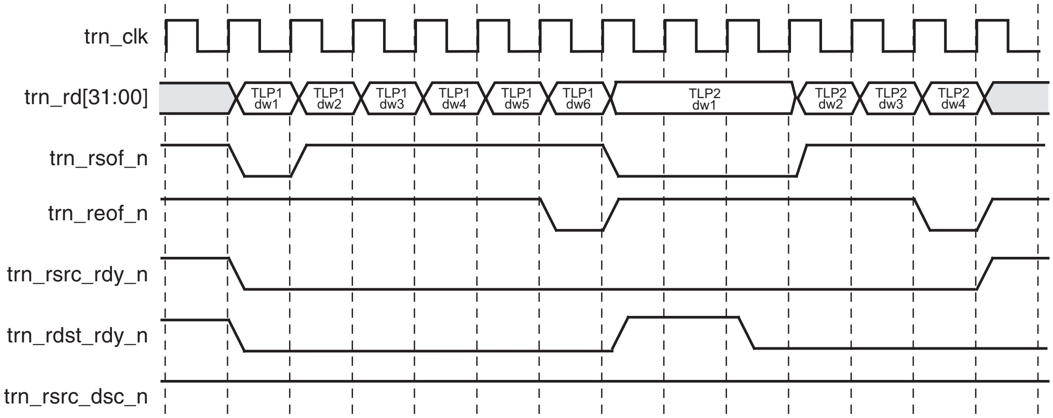


Figure 6-19: **User Application Throttling of Back-to-Back TLPs**

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 for Posted and Completion TLPs to bypass blocked Non-Posted TLPs.

The User Application may deassert `trn_rnp_ok_n` if it is not ready to accept Non-Posted Transactions from the core, (as shown in Figure 6-20) but can receive Posted and Completion Transactions. 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 packet 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.

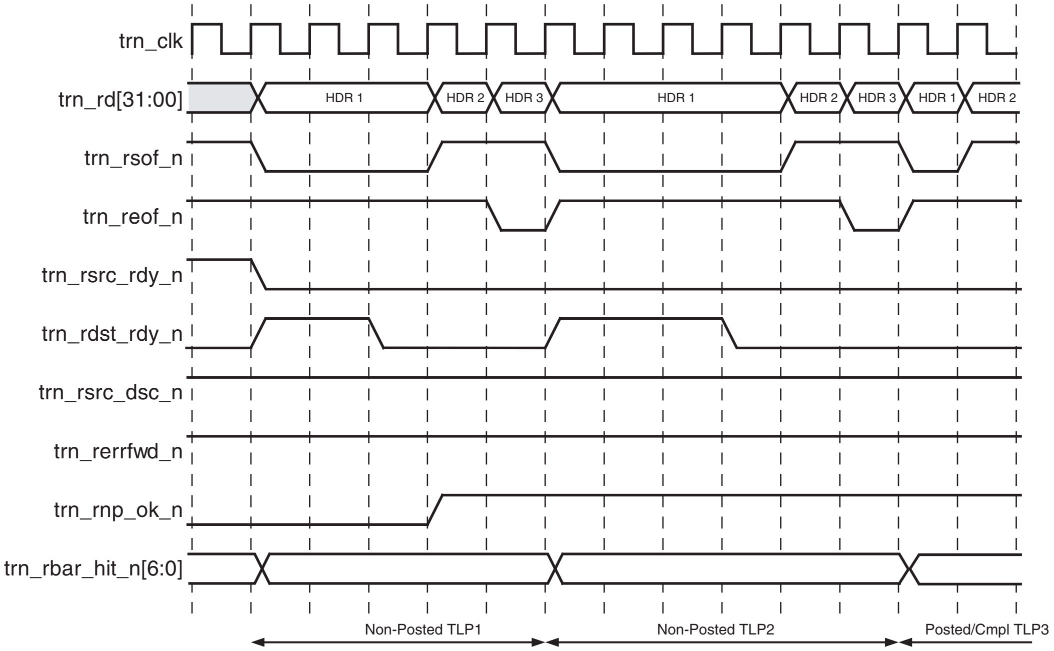


Figure 6-20: **Packet Re-ordering on Receive Transaction Interface**

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. The following 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 a 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 (Valid transaction Start-Of-Frame accepted by user application)  {
     Extract TLP Format and Type from the 1st TLP DW
     if (TLP type == Non Posted)  {
       if (Non-Posted_Buffers_Available <= 2)   // Accounts for the
current and possibly the next NP TLP
         Deassert trn_rnp_ok_n on the following clock cycle.
       else if (Other optional  user policies to stall Non-Posted
transactions)
         Deassert trn_rnp_ok_n on the following clock cycle.
       else // (Non-Posted_Buffers_Available > 2)
         Assert trn_rnp_ok_n on the following clock cycle.
       Decrement Non-Posted_Buffers_Available in User Application
      } else {  // Posted and Completion TLPs
        Process the received TLPs
      }
   }
}
```

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 6-21.

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 the following 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.

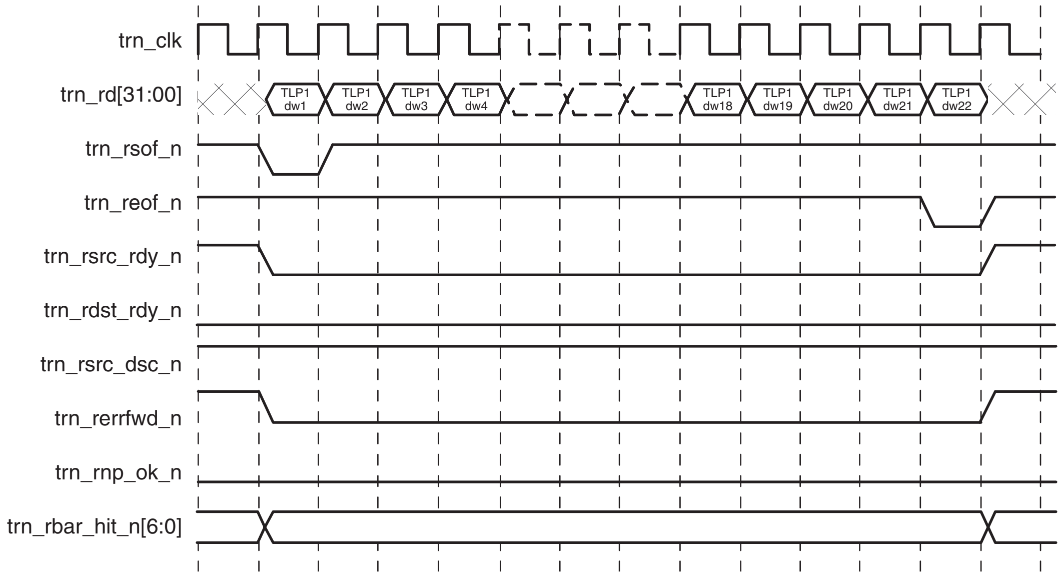


Figure 6-21: **Receive Transaction Data Poisoning**

Packet Base Address Register Hit on Receive Transaction Interface

The core decodes incoming Memory and IO 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 IO 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 IO 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 will drop it without presenting it to the user and it will automatically generate an Unsupported

Request message. Note that even if the core is configured for a 64-bit BAR, the system may not always allocate a 64-bit address, in which case only one `trn_rbar_hit_n[6:0]` signal will be asserted.

Table 6-2 illustrates mapping between `trn_rbar_hit_n[6:0]` and the BARs, and the corresponding byte offsets in the core Type0 configuration header.

Table 6-2: **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 IO 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 6-22. When receiving non-Memory and non-IO transactions, `trn_rbar_hit_n[6:0]` is undefined.

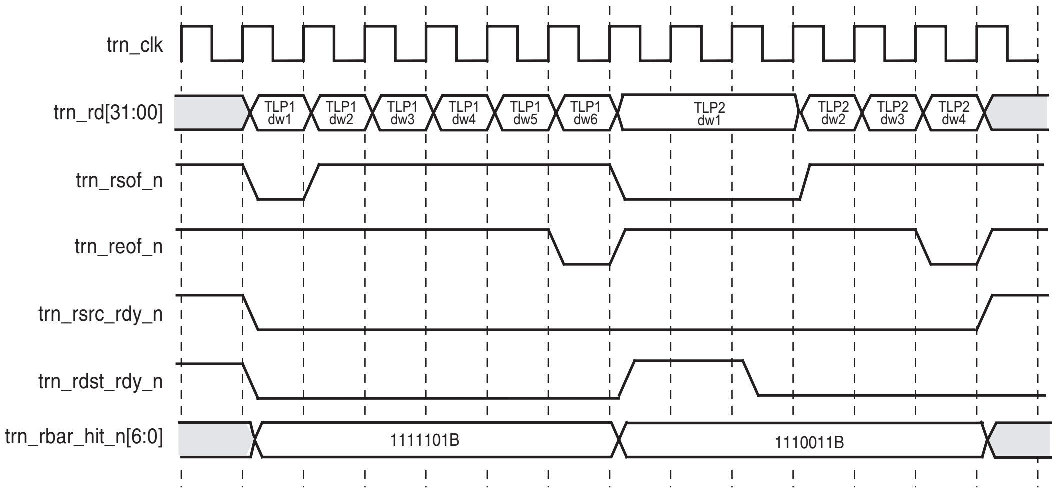


Figure 6-22: **BAR Target Determination using trn_rbar_hit**

The signal `trn_rbar_hit_n[6:0]` enables received Memory/IO Transactions to be directed to the appropriate destination Memory/IO apertures in the User Application. By utilizing `trn_rbar_hit_n[6:0]`, application logic may inspect only the lower order Memory/IO 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 6-23 illustrates packet transfer discontinue scenario.

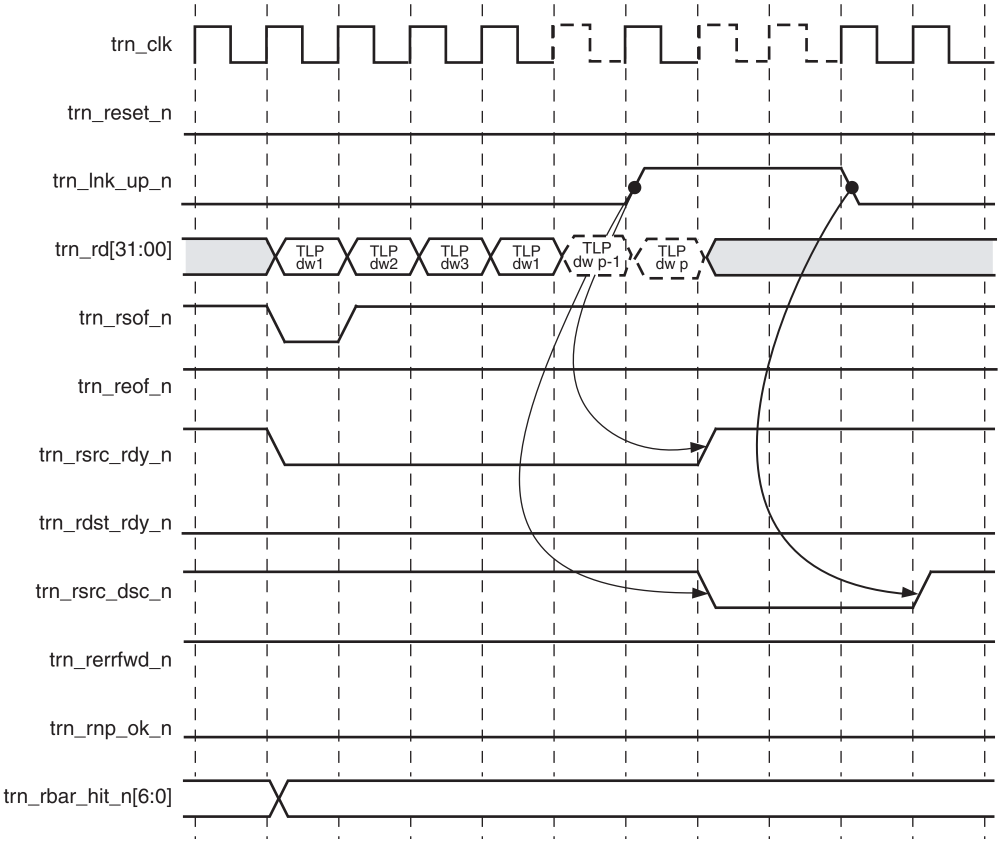


Figure 6-23: **Receive Transaction Discontinue**

Receiver Flow Control Credits Available

The Integrated Endpoint Block 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 6-3 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 will indicate a value of zero. Credits available return to initial values after the receiver has drained all TLPs.

Table 6-3: **Transaction Receiver Credits Available Initial Values**

Credit Category	Performance Level	128 byte Capability MPS	256 byte Capability MPS	512 byte Capability MPS
Non-Posted Header	Good	8		
	High			
Posted Header	Good	16	24	32
	High	30	32	32
Posted Data	Good	41	96	211
	High	89	211	467
Completion Header	Good	16	24	40
	High	30	40	40
Completion Data	Good	41	96	211
	High	89	211	467

The User Application may 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," page 103.

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 A, "Managing Receive-Buffer Space for Inbound Completions."

Accessing Configuration Space Registers

Registers Mapped Directly onto the Configuration Interface

The Integrated Endpoint Block core provides direct access to select command and status registers in its Configuration Space. Values in these registers are modified by Configuration Writes received from the Root Complex and cannot be modified by the User Application. Table 6-4 defines the command and status registers mapped to the configuration port.

Table 6-4: **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, hard wired to 000b.
cfg_status[15:0]	Output	Status Register: Status register from the Configuration Space Header.
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 Extended Capability Structure.
cfg_dcommand[15:0]	Output	Device Command Register: Device control register from the PCI Express Extended Capability Structure.
cfg_lstatus[15:0]	Output	Link Status Register: Link status register from the PCI Express Extended Capability Structure.
cfg_lcommand[15:0]	Output	Link Command Register: Link control register from the PCI Express Extended 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. Note that this core supports only one function; for this reason, the function number is hard wired to 000b.

cfg_status[15:0]

This bus allows the User Application to read the Status register in the PCI Configuration Space Header. Table 6-5 defines these bits. See the *PCI Express Base Specification* for detailed information.

Table 6-5: **Bit Mapping on Header Status Register**

Bit	Name
cfg_status[15]	Detected Parity Error
cfg_status[14]	Signaled System Error
cfg_status[13]	Received Master Abort
cfg_status[12]	Received Target Abort
cfg_status[11]	Signaled Target Abort
cfg_status[10:9]	DEVSEL Timing (hard-wired to 00b)
cfg_status[8]	Master Data Parity Error
cfg_status[7]	Fast Back-to-Back Transactions Capable (hard-wired to 0)
cfg_status[6]	Reserved
cfg_status[5]	66 MHz Capable (hard-wired to 0)
cfg_status[4]	Capabilities List Present (hard-wired to 1)
cfg_status[3]	Interrupt Status
cfg_status[2:0]	Reserved

cfg_command[15:0]

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

Table 6-6: **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
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

Table 6-6: **Bit Mapping on Header Command Register *(Cont'd)***

Bit	Name
cfg_command[1]	Memory Address Space Decoder Enable
cfg_command[0]	IO 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 Extended Capabilities Structure. Table 6-7 defines each bit in the `cfg_dstatus` bus. See the *PCI Express Base Specification* for detailed information.

Table 6-7: **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
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 Extended Capabilities Structure. Table 6-8 defines each bit in the `cfg_dcommand` bus. See the *PCI Express Base Specification* for detailed information.

Table 6-8: **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]	Max_Payload_Size
cfg_dcommand[4]	Enable Relaxed Ordering
cfg_dcommand[3]	Unsupported Request Reporting Enable
cfg_dcommand[2]	Fatal Error Reporting Enable

Table 6-8: **Bit Mapping of PCI Express Device Control Register** ***(Cont'd)***

Bit	Name
cfg_dcommand[1]	Non-Fatal Error Reporting Enable
cfg_dcommand[0]	Correctable Error Reporting Enable

cfg_lstatus[15:0]

This bus reflects the value stored in the Link Status register in the PCI Express Extended Capabilities Structure. Table 6-9 defines each bit in the `cfg_lstatus` bus. See the *PCI Express Base Specification* for details.

Table 6-9: **Bit Mapping of PCI Express Link Status Register**

Bit	Name
cfg_lstatus[15:13]	Reserved
cfg_lstatus[12]	Slot Clock Configuration
cfg_lstatus[11]	Reserved
cfg_lstatus[10]	Reserved
cfg_lstatus[9:4]	Negotiated Link Width
cfg_lstatus[3:0]	Link Speed

cfg_lcommand[15:0]

This bus reflects the value stored in the Link Control register of the PCI Express Extended Capabilities Structure. Table 6-10 provides the definition of each bit in `cfg_lcommand` bus. See the *PCI Express Base Specification* for more details.

Table 6-10: **Bit Mapping of PCI Express Link Control Register**

Bit	Name
cfg_lcommand[15:8]	Reserved
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

Accessing Additional 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 2-9, page 35.

The User Application must supply the read 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 the 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 6-24 illustrates an example with two consecutive reads from the Configuration Space.

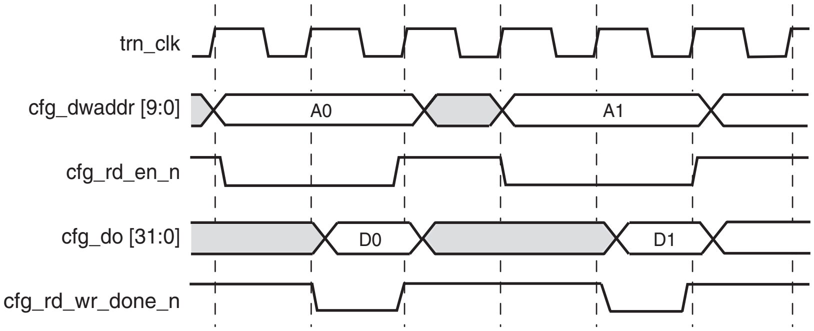


Figure 6-24: **Example Configuration Space Access**

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 Endpoint 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 Endpoint Block for PCIe 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 Endpoint Block core 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 A, "Managing Receive-Buffer Space for Inbound Completions" for more information.

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 2-9, page 35.

- 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 Figures 6-25, 6-26, and 6-27, and Tables 6-11 and 6-12.

Table 6-11: **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. 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. 1: A Completion with a "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 6-12: **Possible Error Conditions for TLPs Received by the User Application**

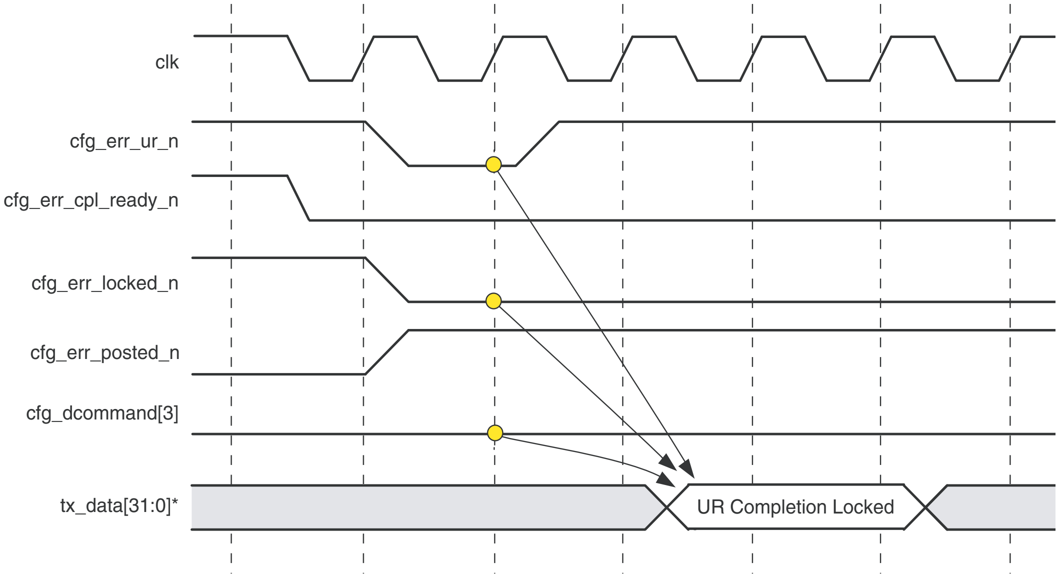
Received TLP Type		Possible Error Condition					Error Qualifying Signal Status	
		Unsupported Request (cfg_err_ur_n)	**Completion Abort** (cfg_err_cpl_abort_n)	**Correctable Error** (cfg_err_cor_n)	**ECRC Error** (cfg_err_ecrc_n)	**Unexpected Completion** (cfg_err_cpl_unexpect_n)	**Value to Drive** on (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

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 6-25. 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 following sections.

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 will result in a Non-Fatal Message to be sent, if enabled.

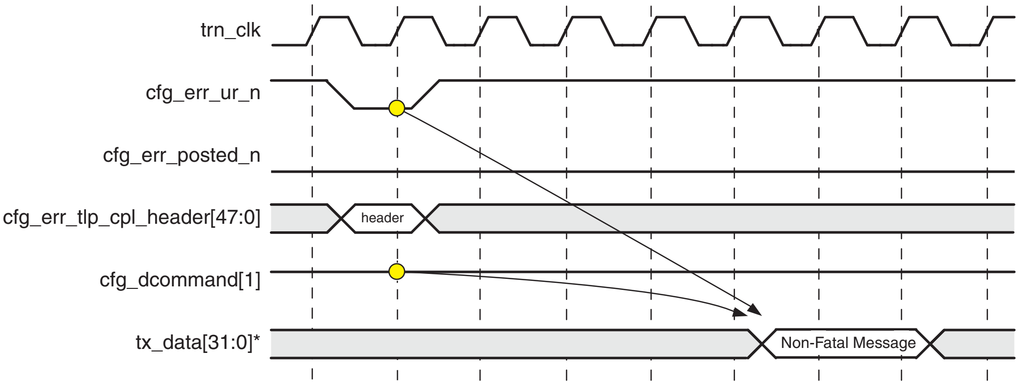
The core's ability to generate error messages can be disabled by the Root Complex issuing a configuration write to the Endpoint core's Device Control register and the PCI Command register setting the appropriate bits to 0. For more information about these registers, see Chapter 7 of the *PCI Express Base Specification*. However, error-reporting status bits are always set in the Configuration Space whether or not their Messages are disabled.

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` will deassert, and the user must cease signaling those types of errors on the same cycle. In addition, the user must not resume signaling those types of errors until `cfg_err_cpl_rdy_n` reasserts.



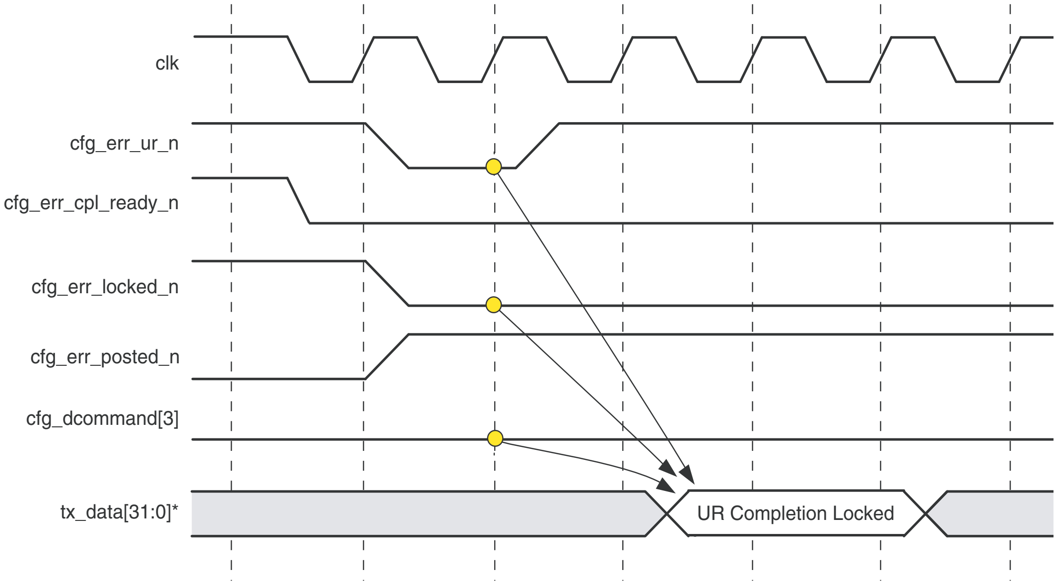
*Internal signal not appearing on the User Interface

Figure 6-25: **Signaling Unsupported Request for Non-Posted TLP**



* Internal signals not appearing on User Interface

Figure 6-26: **Signaling Unsupported Request for Posted TLP**



*Internal signal not appearing on the User Interface

Figure 6-27: **Signaling Locked Unsupported Request for Locked Non-Posted TLP**

Completion Timeouts

The Integrated Endpoint 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 Endpoint 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 will be 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 will be 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 the following:

- 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 Endpoint 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.

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 2-8, page 32 defines the Flow Control Credit signals. Credit status information is presented on the following 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 may 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 6-13.

Table 6-13: **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]` may 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. Figure 6-28 illustrates the timing of the Flow Control Credits signals.

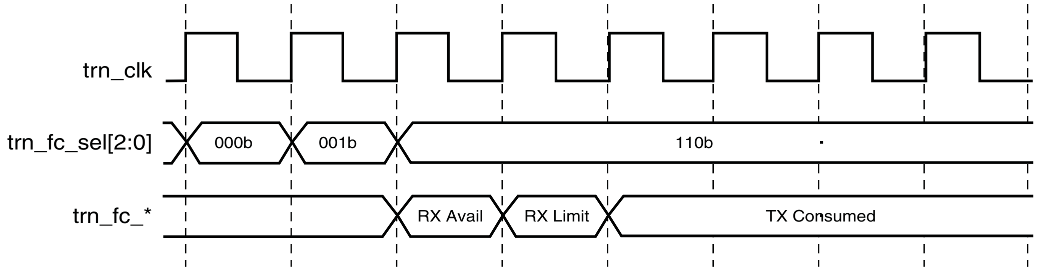


Figure 6-28: **Flow Control Credits**

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 6-14 defines the possible values presented on the `trn_fc_*` signals. Initial credit information will vary depending on the size of the receive buffers within the Integrated Block and the Link Partner.

Table 6-14: **trn_fc_* Value Definition**

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

[a] Only Transmit Credits Available Space will ever indicate Negative or Infinite credits available

Receive Credit Flow Control Information

Receive Credit Flow Control Information may 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 will indicate 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 A, "Managing Receive-Buffer Space for Inbound Completions" to avoid completion buffer overflow.

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

Receive Credits Limit show 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 will always indicate zero credits.

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

Receive Buffer Credits Consumed show 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 may 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 will indicate 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. Since the block does not allow posted packets to pass completions, a posted packet that is written will not be 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 will not be 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 will pass Non-Posted packets waiting for credits.

The Link Partner may advertise infinite credits for one or more of the three traffic types. Infinite credits will be indicated to the user by setting the Header and Data credit outputs to their maximum value as indicated in Table 6-14.

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 will always indicate 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*.

Power Management

The Integrated Endpoint Block core supports the following 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 sections 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.

Programmed Power Management

To achieve considerable power savings on the PCI Express hierarchy tree, the core supports the following link states of Programmed Power Management (PPM):

- L0: Active State (data exchange state)
- L1: Higher Latency, lower power standby state
- L3: Link Off State

All PPM messages are always initiated by an upstream link partner. Programming the core to a non-D0 state, results in PPM messages being exchanged with the upstream link-partner. The PCI Express Link transitions to a lower power state after completing a successful exchange.

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

The following 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 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.
3. The core exchanges appropriate power management messages with its link partner to successfully transition the link to a lower power PPM L1 state. This action is transparent to the user logic.
4. All user transactions are stalled for the duration of time when the device power state is non-D0.
5. The device power state is communicated to the user logic through the user configuration port interface. The user logic is responsible for performing a successful read operation to identify the device power state.
6. 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.
7. The user logic must poll the PME_Status bit of the PMCSR (via the Configuration Interface). If a PME message is not acknowledged by the host within 100 ms (+50%/-5%) by the host clearing the PME_Status bit, the Endpoint is required to re-transmit. This functionality is not provided by the Spartan-6 FPGA Integrated Block for PCI Express. For more information, see section 5.3.3.3.1 of the *PCI Express Base Specification v1.1*.

   Note: If the upstream link partner has not configured the device to allow the generation of PM_PME messages (PME_En bit of PMCSR = 0), the assertion of `cfg_pm_wake_n` will be ignored by the core.

PPM L3 State

The following steps outline the transition of the Integrated Endpoint for PCIe core to the PPM L3 state.

1. The core will negotiate 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 Endpoint core initiates a handshake with the user logic through cfg_to_turnoff_n (see Table 2-8, page 32) 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 Endpoint core to its upstream link partner.
4. The Endpoint core closes all its interfaces, disables the Physical/Data-Link/Transaction layers and is ready for removal of power to the core.

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 Endpoint PIPE for PCIe 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 or of `cfg_turnoff_ok_n`, the User Application has committed to being turned off.

4. The Endpoint core sends a PME_TO_Ack when it detects assertion of `cfg_turnoff_ok_n`.

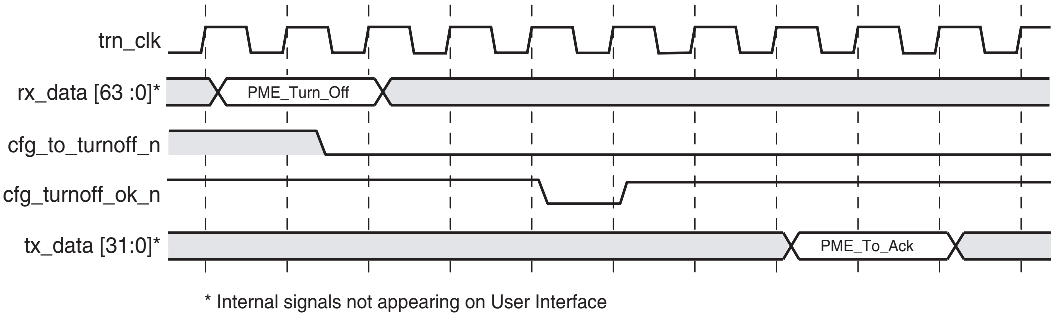


Figure 6-29: **Power Management Handshaking**

Generating Interrupt Requests

The Integrated Endpoint Block supports sending interrupt requests as either legacy interrupts or Message Signaled Interrupts (MSI). The mode is programmed using the MSI Enable bit in the Message Control Register of the MSI Capability Structure. For more information on the MSI capability structure please refer to section 6.8 of the PCI Local Base Specification v3.0. The state of the MSI Enable bit is reflected by the `cfg_interrupt_msienable` output:

- cfg_interrupt_msienable = 0: Legacy Interrupt (INTx) mode
- cfg_interrupt_msienable = 1: MSI mode

If the MSI Enable bit is set to a 1, the core will generate 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: interrupts enabled
- cfg_command[10] = 1: interrupts disabled (request are blocked by the core)

The User Application requests interrupt service in one of two ways, each of which are described below. The User Application must determine which method to use based on the value of the `cfg_interrupt_msienable` output. When 0, the Legacy Interrupt method must be used; when 1, the MSI method.

Note that the MSI Enable bit in the MSI 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. Regardless of the interrupt type used, the user initiates interrupt requests through the use of `cfg_interrupt_n` and `cfg_interrupt_rdy_n` as shown in Table 6-15

Table 6-15: **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.

MSI Mode

- As shown in Figure 6-30, 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 may 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 may 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, 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.

The following 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

Legacy Interrupt Mode

- As shown in Figure 6-30, 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, INTB, INTC, or INTD) using `cfg_interrupt_di[7:0]` as shown in Table 6-16.
- 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).

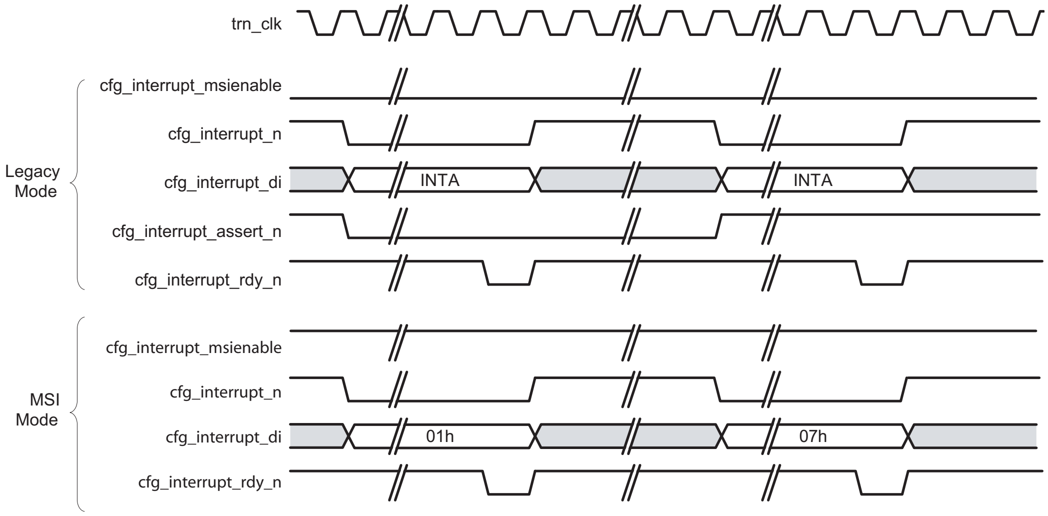


Figure 6-30: **Requesting Interrupt Service: MSI and Legacy Mode**

Table 6-16: **Legacy Interrupt Mapping**

cfg_interrupt_di[7:0] value	Legacy Interrupt
00h	INTA
01h	INTB
02h	INTC
03h	INTD

Clocking and Reset of the Integrated Endpoint Block Core

Reset

The Endpoint Integrated Endpoint 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 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 the following conditions:

- **Fundamental Reset**: Occurs (cold or warm) due to assertion of `sys_reset_n`.
- **PLL within the core**: Loses lock, indicating an issue with the stability of the clock input.
- **Loss of Transceiver PLL Lock:** The Lane 0 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 core input system clock signal is called `sys_clk`. The core requires a 125 or 250 MHz clock input. The clock frequency used must match the clock frequency selection in the CORE Generator 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. Note that 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.

  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. See the *Spartan-6 FPGA GTP Transceiver 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. When using the 125 or 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 6-31 and Figure 6-32. 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 mother board clocking schemes, synchronous clocking should still be used as shown in Figure 6-31.

Figure 6-31 and Figure 6-32 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.

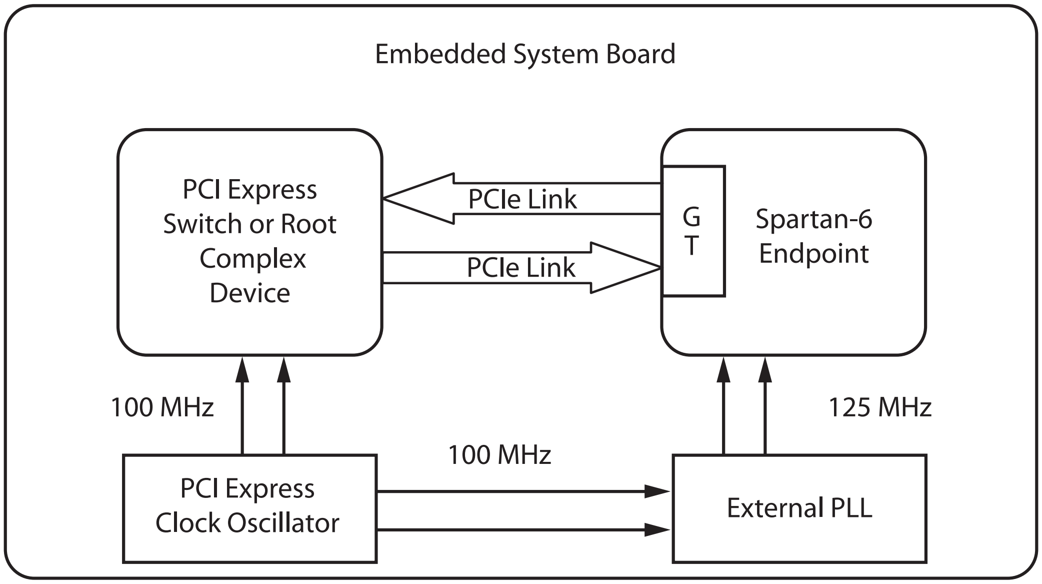


Figure 6-31: **Embedded System Using 125 MHz Reference Clock**

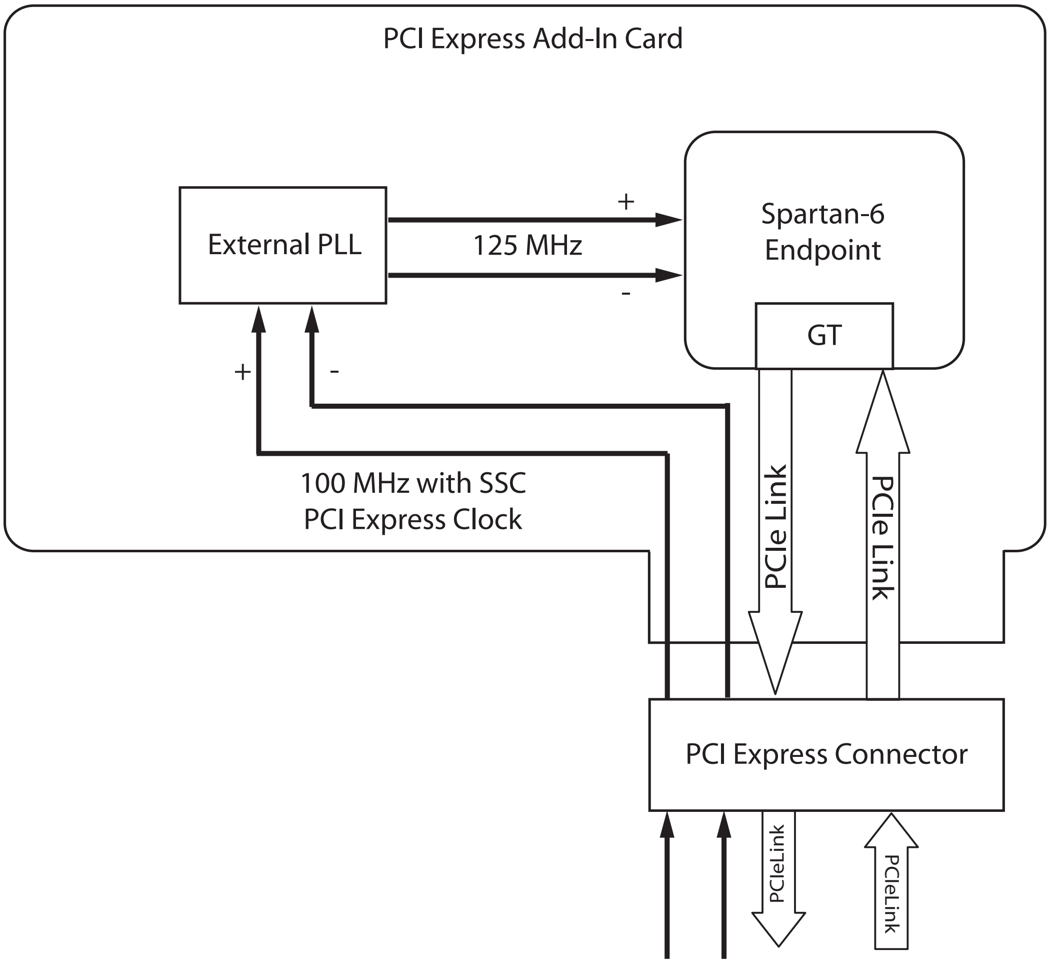


Figure 6-32: **Open System Add-In Card Using 125 MHz Reference Clock**

Chapter 7

Core Constraints

The Integrated Endpoint 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 Solution 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 the 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 Speedgrade

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 may be included. The device in the UCF reflects the device chosen in the CORE Generator 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

Some constraints provided in the UCF utilize hierarchical paths to elements within the Integrated Endpoint Block core. 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/mgt/GT_i/tile0_gtpa1_dual_wrapper_i/gtpa1_dual_i
LOC = GTPA1_DUAL_X0Y0
```

becomes

```
INST xilinx_pcie_ep/mgt/GT_i/tile0_gtpa1_dual_wrapper_i/gtpa1_dual_i
LOC = GTPA1_DUAL_X0Y0
```

The provided UCF includes a line specifying attributes for the sys_reset_n pin, but it is up to the user to un-comment that line and provide a pin location. In addition, the UCF includes blank sections for constraining user-implemented logic. While the constraints provided adequately constrain the Integrated Endpoint 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 Endpoint 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 = xc6slx45t-fgg484-2;
```

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_clk_p LOC = Y4;
```

```
NET sys_clk_n LOC = Y3;
```

See "Clocking and Reset of the Integrated Endpoint Block Core," page 111 for detailed information about reset and clock requirements.

Each GTPA1_DUAL tile contains two transceivers. Any transceiver along the top edge of the device may be used with the Integrated Block for PCI Express. For GTPA1_DUAL pinout information, see the *Spartan-6 GTP Transceiver 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. Note that the provided transceiver wrapper file consumes *both* transceivers in a tile even though only one is used.

For example:

```
INST core/mgt/GT_i/tile0_gtpa1_dual_wrapper_i/gtpa1_dual_i LOC =
GTPA1_DUAL_X0Y0;
```

Core Physical Constraints

Physical constraints may be included in the constraints file to control the location of clocking and memory elements. Specific physical constraints are chosen to match each supported device and package combination—it is very important to leave these constraints unmodified except for changing the hierarchical name, as described above.

Core Timing Constraints

Timing constraints are provided for all Integrated Endpoint 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 either 125 MHz or 250 MHz (selected in the CORE Generator GUI).

```
NET sys_clk_c PERIOD = 8ns;
```

Next, the internally generated clock net and period is specified, which can be 125 MHz or 250 MHz. (Note that *both* clock constraints must be specified as having the same period.)

```
NET core/gt_refclk_out TNM_NET = GT_REFCLK_OUT ;
```

```
TIMESPEC TS_GT_REFCLK_OUT = PERIOD GT_REFCLK_OUT 8ns HIGH 50 % ;
```

Relocating the Integrated Endpoint Block

While Xilinx does not provide technical support for designs whose system clock input, GTP 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 Endpoint Block, which enables meeting 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: **Spartan-6 LXT Pinout**

Package	Part	GTPA1_DUAL	Channel	sys_clk_n	sys_clk_p	pci_exp_txn	pci_exp_txp	pci_exp_rxn	pci_exp_rxp
CSG324	XC6SLX25T	X0Y0	0	A8	B8	A4	B4	C5	D5
	XC6SLX45T								
	XC6SLX25T		1	C9	D9	A6	B6	C7	D7
	XC6SLX45T								
	XC6SLX45T	X1Y0	0	A10	B10	A12	B12	C11	D11
	XC6SLX45T		1	E10	F10	A14	B14	C13	D13
FGG484	XC6SLX25T	X0Y0	0	B10	A10	A6	B6	C7	D7
	XC6SLX45T								
	XC6SLX25T		1	D11	C11	A8	B8	C9	D9
	XC6SLX45T								
	XC6SLX45T	X1Y0	0	B12	A12	A14	B14	C13	D13
	XC6SLX45T		1	F12	E12	A16	B16	C15	D15

Table 7-1: **Spartan-6 LXT Pinout *(Cont'd)***

Package	Part	GTPA1_DUAL	Channel	sys_clk_n	sys_clk_p	pci_exp_txn	pci_exp_txp	pci_exp_rxn	pci_exp_rxp
FGG484	XC6SLX100T	X0Y1	0	B10	A10	A6	B6	C7	D7
	XC6SLX150T								
	XC6SLX100T		1	D11	C11	A8	B8	C9	D9
	XC6SLX150T								
	XC6SLX100T	X1Y1	0	B12	A12	A14	B14	C13	D13
	XC6SLX150T								
	XC6SLX100T		1	F12	E12	A16	B16	C15	D15
	XC6SLX150T								
FGG676	XC6SLX75T	X0Y1	0	A10	B10	A6	B6	C7	D7
	XC6SLX100T								
	XC6SLX150T								
	XC6SLX75T		1	C11	D11	A8	B8	C9	D9
	XC6SLX100T								
	XC6SLX150T								
	XC6SLX75T	X1Y1	0	C15	D15	A18	B18	C17	D17
	XC6SLX100T								
	XC6SLX150T								
	XC6SLX75T		1	A16	B16	A20	B20	C19	D19
	XC6SLX100T								
	XC6SLX150T								
FGG900	XC6SLX100T	X0Y1	0	A13	B13	A9	B9	C10	D10
	XC6SLX150T								
	XC6SLX100T		1	C14	D14	A11	B11	C12	D12
	XC6SLX150T								
	XC6SLX100T	X1Y1	0	C18	D18	A21	B21	C20	D20
	XC6SLX150T								
	XC6SLX100T		1	A19	B19	A23	B23	C22	D22
	XC6SLX150T								

Chapter 8

FPGA Configuration

This chapter discusses how to configure the Spartan®-6 FPGA so that the device can link up and be recognized by the system. This information is provided so you can choose the correct FPGA configuration method for your 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 you read the actual specifications for detailed information. This chapter is divided into three sections:

- "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, the following 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*, rev 1.1 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 Endpoint 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#. (This figure is based on Figure 2-10 from section 2.1 of *PCI Express Card Electromechanical Specification*, rev 1.1.)

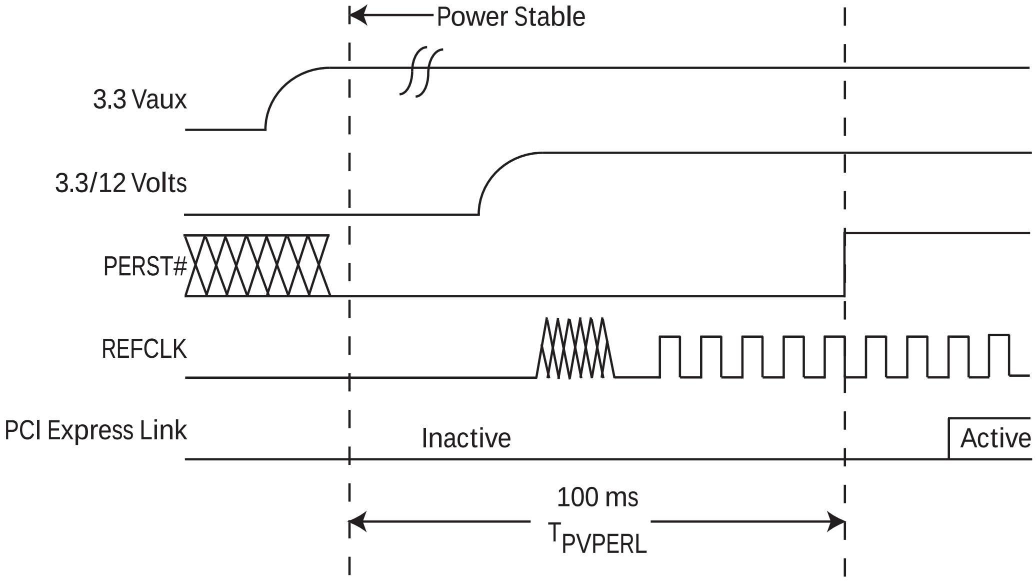


Figure 8-1: **Power Up**

Section 2.6.2 of the *PCI Express Card Electromechanical Specification* defines T_{PVPREL} as a minimum of 100 ms, indicating that from the time power is stable the system reset will be 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:

$$\text{FPGA Configuration Time} \le T_{PWRVLD} + T_{PVPERL}$$

Given that T_{PVPERL} is defined as 100 ms minimum, this becomes:

$$\text{FPGA Configuration Time} \le T_{PWRVLD} + 100\text{ ms}$$

Note: Although T_{PWRVLD} is included in the previous equation, is has yet to be defined in this discussion because it depends on the type of system in use. The next section, "Board Power in Real-world Systems" 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 you suspect the design is having problems due to FPGA configuration, 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` asserts high after some delay once the power supply has reached 95% of nominal.

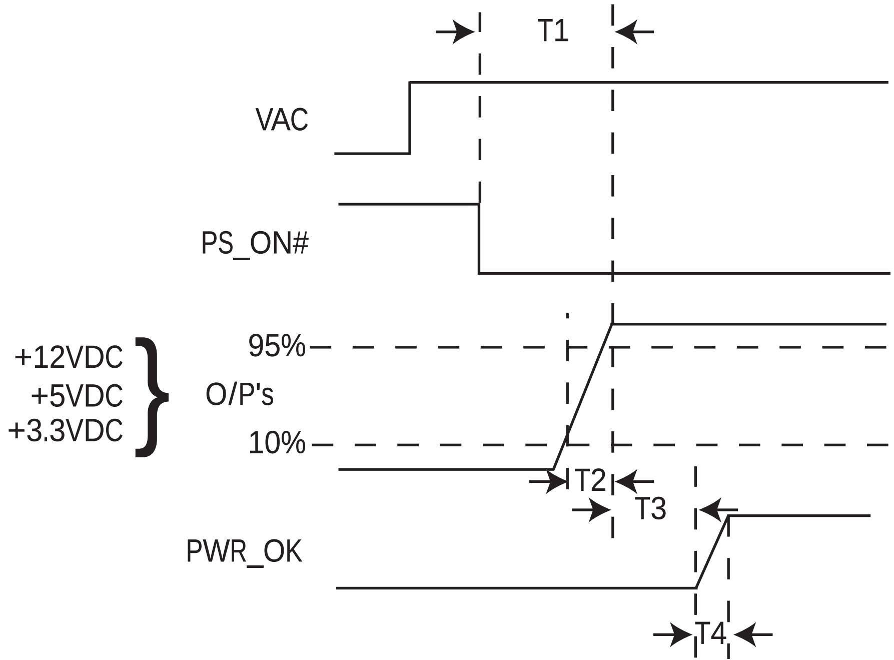

T1
VAC
PS_ON#
+12VDC
+5VDC
+3.3VDC
95%
O/P's
10%
T2
T3
PWR_OK
T4
T1 = Power On Time (T1 < 500 ms)
T2 = Risetime (0.1 ms <= T2 <= 20 ms)
T3 = PWR_OK Delay (100 ms < T3 < 500 ms)
T4 = PWR_OK risetime (T4 <= 10 ms)

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:

$$\text{FPGA Configuration Time} \leq T_{PWRVLD} + 100 \text{ ms}$$

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:

$$100 \text{ ms} \leq T_{PWRVLD} \leq 500 \text{ ms}$$

This provides the following requirement for FPGA configuration time in both ATX and non-ATX-based motherboards:

- FPGA Configuration Time ≤ 200 ms (for ATX based motherboard)
- FPGA Configuration Time ≤ 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}.

Recommendations

Xilinx recommends using a Quad-SPI Flash device in Master Serial/SPI mode with a CCLK frequency of 33 MHz, which allows time for the FPGA configuration of the Spartan-6 FPGA in ATX-based motherboards. Configuration options are shown as 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 may arise during FPGA configuration.

FPGA Configuration Times for Spartan-6 Devices

During power up, the FPGA configuration sequence is performed in four steps:

1. Wait for `POR` (Power on Reset) for all voltages (`VccInt`, `Vccaux`, and `VccO`) 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 ≤ 50ms
3. Wait for assertion of `DONE`, the actual time required for a bitstream to transfer, and depends on the following:
   - 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 *Spartan-6 FPGA Configuration User Guide*.

Table 8-2 and Table 8-3 show the comparative data for all Spartan-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 Spartan-6 FPGA Configuration Guide (UG191) 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. Choose a configuration method shaded in green when using ATX-based systems to ensure that the device is recognized.

Table 8-2: **Configuration Time Matrix (ATX Motherboards): Spartan-6 FPGA Bitstream Transfer Time in Milliseconds**

Spartan-6 FPGA	Bitstream (Bits)	SPIx4[a]	XCF128X[a] (Slave-SMAPx16)
XC6SLX25T	6,411,440	49	13
XC6SLX45T	11,875,104	90	23
XC6SLX75T	19,624,608	149	38
XC6SLX100T	26,543,136	202	51
XC6SLX150T	33,761,568	256	64
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			

[a] CCLK assumptions: 33 MHz

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.Choose a configuration method shaded in green when using non-ATX-based systems to ensure that the device is recognized.

It is also obvious that for some of the larger FPGAs, it may not be possible to configure within the 100 ms window. In these cases, evaluate your system 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): Spartan-6 FPGA Bitstream Transfer Time in Milliseconds**

Spartan-6 FPGA	Bitstream (Bits)	SPIx4[a]	XCF128X[a] (Slave-SMAPx16)
XC6SLX25T	6,411,440	49	13
XC6SLX45T	11,875,104	90	23
XC6SLX75T	19,624,608	149	38
XC6SLX100T	26,543,136	202	51
XC6SLX150T	33,761,568	256	64

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

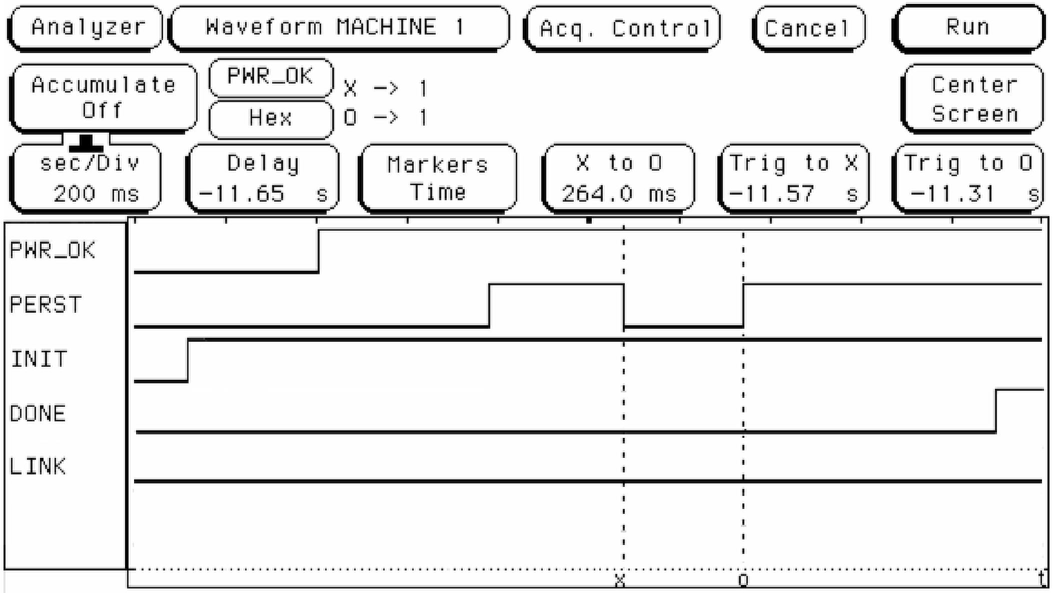
[a] Assumes CCLK=33 MHz

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.

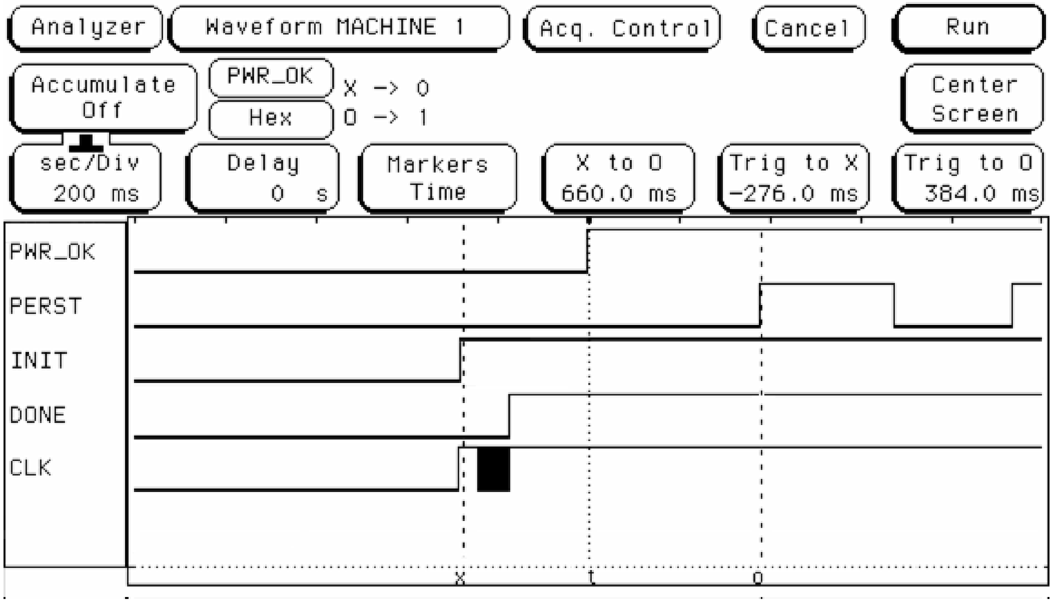


PWR_OK X -> 1
Hex O -> 1
sec/Div 200 ms
Delay -11.65 s
X to O 264.0 ms
Trig to X -11.57 s
Trig to O -11.31 s

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 ns before `PWR_OK`. DONE going high shows that the FPGA was configured even before `PWR_OK` was asserted.



Accumulate Off

PWR_OK X -> O

Hex O -> 1

sec/Div 200 ms

Delay 0 s

X to O 660.0 ms

Trig to X -276.0 ms

Trig to O 384.0 ms

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 may 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.

   $$\text{FPGA Configuration Time} \leq T_{PWRVLD} + T_{PERSTPERIOD} + 100\text{ ms}$$

2. In closed systems, it may 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 sub-system allowing the device to be recognized by the system.

Chapter 9

Programmed Input Output 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 IO Write transactions and respond to Memory and IO Read transactions with Completion with Data transactions.

The PIO example design (PIO design) is included with the Endpoint for PCI Express generated by the CORE Generator, 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 Spartan®-6 FPGA Integrated Endpoint 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 IO 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 and 64-bit TRN interfaces

Figure 9-1 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.

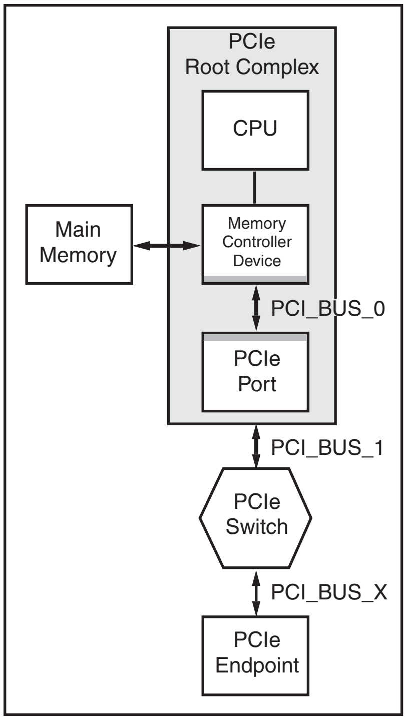


Figure 9-1: **System Overview**

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 IO Read, IO 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 IO Read TLP request presented to it by the core. In addition, the PIO design returns a completion without data with successful status for IO Write TLP request.

The PIO design processes a Memory or IO 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 produces a core configured to work with the PIO design defined in this section, consisting of the following:

- One 64-bit addressable Memory Space BAR
- One 32-bit Addressable Memory Space BAR

Users may change the default parameters used by the PIO design; however, in some cases they may need to change the back-end user application depending on their system. See "Changing CORE Generator Default BAR Settings" for information about changing the default CORE Generator parameters and the affect 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 9-1.

Table 9-1: **TLP Traffic Types**

Block RAM	TLP Transaction Type	Default BAR	trn_rbar_hit_n[6:0]
ep_io_mem	IO TLP transactions	Disabled	Disabled
ep_mem_32	32-bit address Memory TLP transactions	2	111_1011b
ep_mem64	64-bit address Memory TLP transactions	0-1	111_1100b
ep_mem_erom	32-bit address Memory TLP transactions destined for EROM	Exp. ROM	011_1111b

Changing CORE Generator Default BAR Settings

Users can change the CORE Generator parameters and continue to use the PIO design to create customized Verilog 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 parameters:

- The example design supports one IO 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 will be active–accesses to the other spaces will 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 IO space BAR, which by default is disabled, but can be changed if desired.

Although there are limitations to the PIO design, Verilog 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 138 and "Transmit Path," page 140.

The PIO design successfully processes single DWORD payload Memory Read and Write TLPs and IO 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/IO Write TLP Processing

When the Endpoint for PCIe receives a Memory or IO 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. Note that the PIO design handles Memory writes and IO TLP writes in different ways: the PIO design responds to *IO 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 IO 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 IO 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_io_mem (which by default represents 2 kB of IO 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. Note that 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/IO Read TLP Processing

When the Endpoint for PCIe receives a Memory or IO 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 IO 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. Note that 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 9-2 defines the PIO design file structure. Note that based on the specific core targeted, not all files delivered by CORE Generator are necessary, and some files may or may not be delivered. The major difference is that some of the Endpoint for PCIe solutions use a 32-bit user data path, others use a 64-bit data path, and the PIO design works with both. The width of the data path depends on the specific core being targeted.

Table 9-2: **PIO Design File Structure**

File	Description
PIO.[v | vhd]	Top-level design wrapper
PIO_EP.[v | vhd]	PIO application module
PIO_TO_CTRL.[v | vhd]	PIO turn-off controller module
PIO_32.v	32b interface macro define
PIO_64.v	64b macro define
PIO_32_RX_ENGINE.[v | vhd]	32b Receive engine
PIO_32_TX_ENGINE.[v | vhd]	32b Transmit engine
PIO_64_RX_ENGINE.[v | vhd]	64b Receive engine
PIO_64_TX_ENGINE.[v | vhd]	64b Transmit engine
PIO_EP_MEM_ACCESS.[v | vhd]	Endpoint memory access module
PIO_EP_MEM.[v | vhd]	Endpoint memory

Two configurations of the PIO Design are provided: PIO_32 and PIO_64, with 32 and 64-bit TRN interfaces, respectively. The PIO configuration generated depends on the selected endpoint type (that is, Spartan-6 Integrated Endpoint Block, PIPE, PCI Express, and Block Plus) as well as the number of PCI Express lanes selected by the user. Table 9-3 identifies the PIO configuration generated based on the user's selection.

Table 9-3: **PIO Configuration**

Core	x1	x2	x4	x8
Endpoint for PIPE	PIO_32	NA	NA	NA
Endpoint for PCI Express (Soft-IP)	PIO_32	NA	PIO_64	PIO_64
Endpoint for PCI Express Block Plus	PIO_64	NA	PIO_64	PIO_64
Spartan-6 Integrated Endpoint Block	PIO_32	NA	NA	NA
Virtex-6 Integrated Block	PIO_64	PIO_64	PIO_64	PIO_64

Figure 9-2 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.

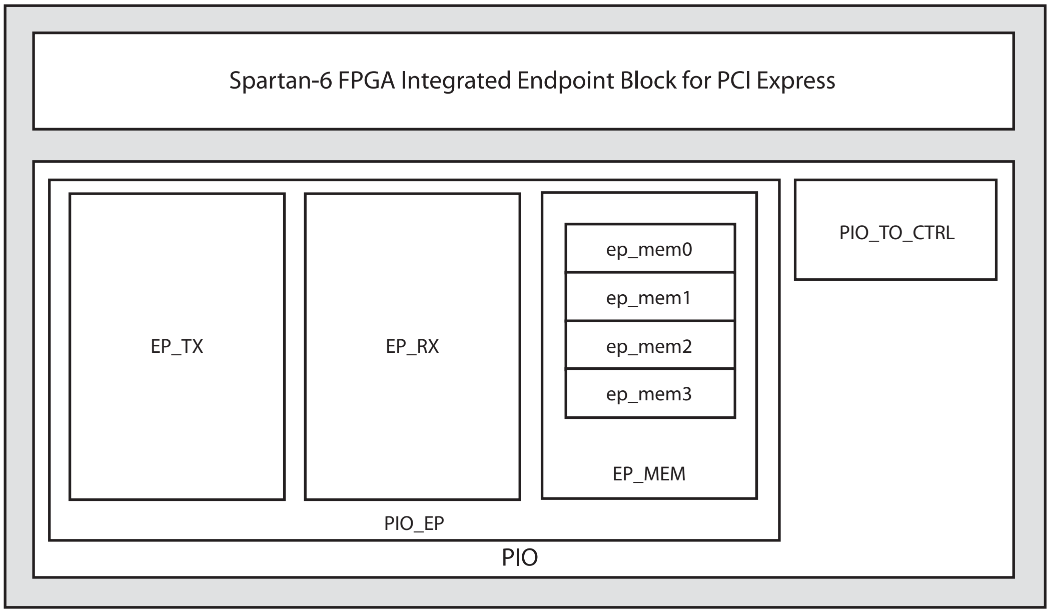


Figure 9-2: **PIO Design Components**

PIO Application

Figure 9-3 andFigure 9-4 depict 64-bit and 32-bit PIO application top-level connectivity, respectively. The data path 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 memory 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 acknowledgement.

The PIO_EP module connects to the Endpoint Transaction (TRN) and Configuration (CFG) interfaces.

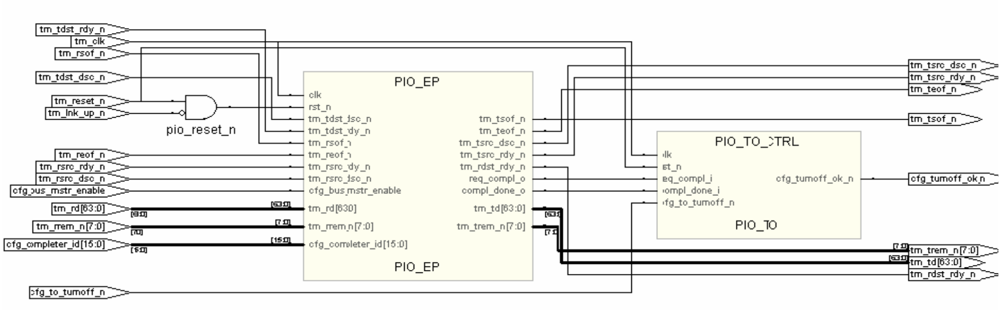


Figure 9-3: **PIO 64-bit Application**

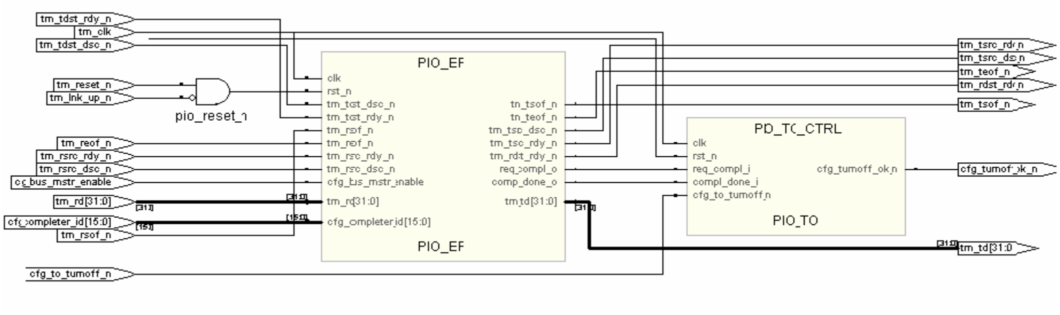


Figure 9-4: **PIO 32-bit Application**

Receive Path

Figure 9-5 illustrates the PIO_32_RX_ENGINE and PIO_64_RX_ENGINE modules. The data path of the module must match the data path of the core being used. These modules connect with Endpoint for PCIe Transaction Receive (trn_r*) interface.

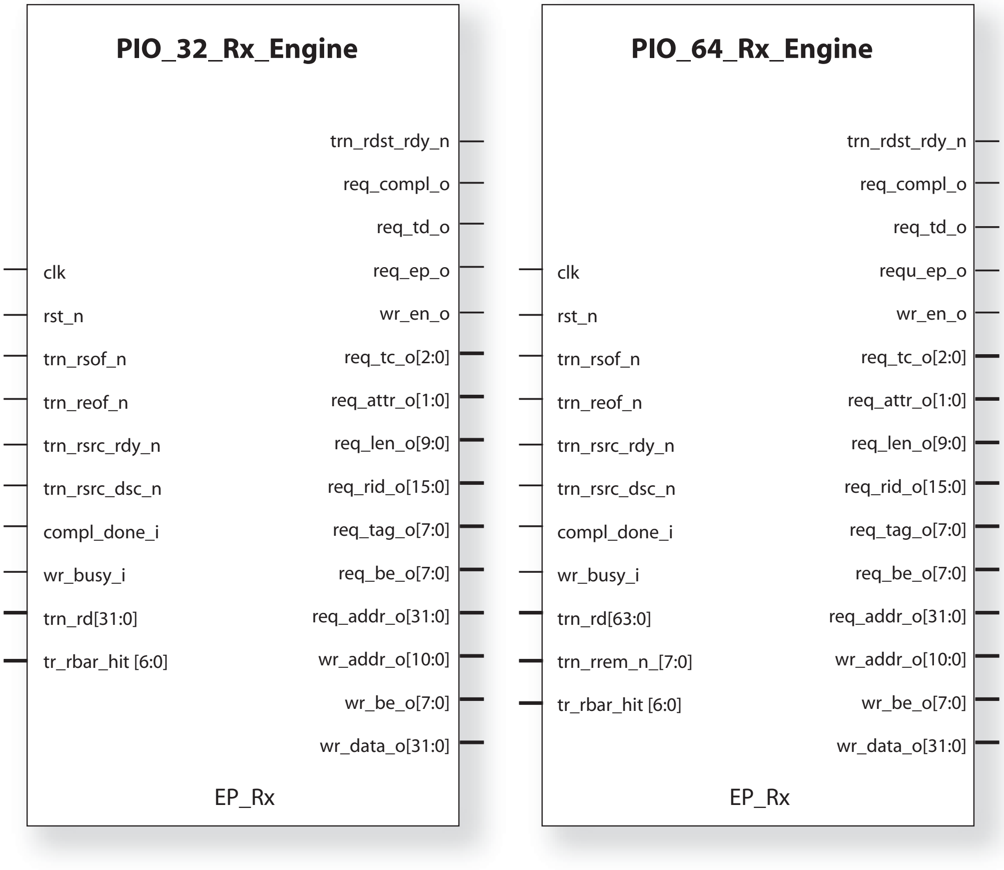


Figure 9-5: **Rx Engines**

The PIO_32_RX_ENGINE and PIO_64_RX_ENGINE modules receive and parse incoming read and write TLPs.

The RX engine parses 1 DWORD 32 and 64-bit addressable memory and IO read requests. The RX state machine extracts needed information from the TLP and passes it to the memory controller, as defined in Table 9-4.

Table 9-4: **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
req_attr_o[1:0]	Request Attributes

Table 9-4: **Rx Engine: Read Outputs *(Cont'd)***

Port	Description
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 IO write requests. The RX state machine extracts needed information from the TLP and passes it to the memory controller, as defined in Table 9-5.

Table 9-5: **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 data path 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 IO Read transaction, the module waits for `compl_done_i` input to be asserted before it accepts the next TLP, while an ongoing Memory or IO Write transaction is deemed complete after `wr_busy_i` is deasserted.

Transmit Path

Figure 9-6 shows the PIO_32_TX_ENGINE and PIO_64_TX_ENGINE modules. The data path of the module must match the data path of the core being used. These modules connect with the core Transaction Transmit (trn_r*) interface.

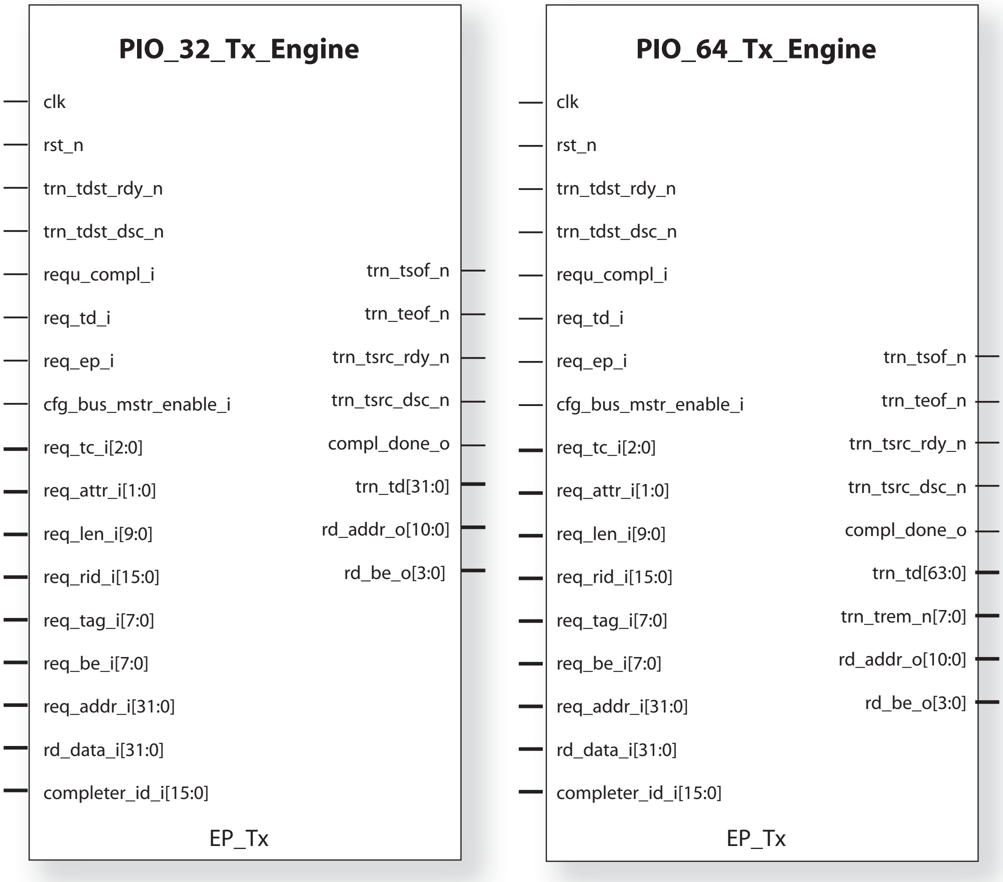


Figure 9-6: **Tx Engines**

The PIO_32_TX_ENGINE and PIO_64_TX_ENGINE modules generate completions for received memory and IO 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 and PIO_64_TX_ENGINE modules generate completions in response to 1 DWORD 32 and 64-bit addressable memory and IO read requests. Information necessary to generate the completion is passed to the TX Engine, as defined in Table 9-6.

Table 9-6: **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

Table 9-6: **Tx Engine Inputs *(Cont'd)***

Port	Description
req_tc_i[2:0]	Request Traffic Class
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 9-7 displays the PIO_EP_MEM_ACCESS module. This module contains the Endpoint memory space.

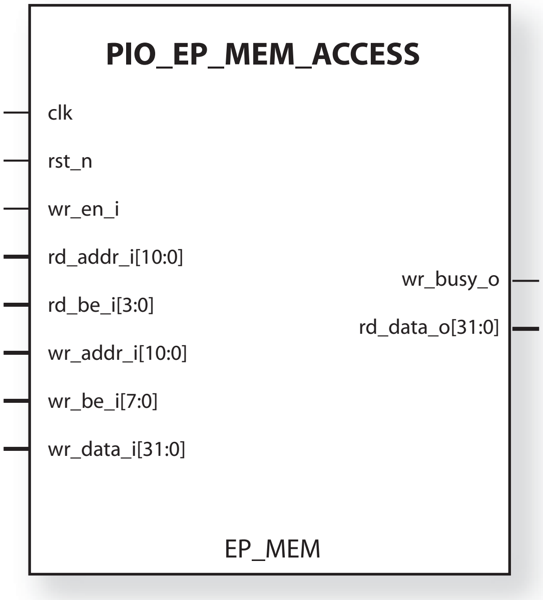


Figure 9-7: **EP Memory Access**

The PIO_EP_MEM_ACCESS module processes data written to the memory from incoming Memory and IO Write TLPs and provides data read from the memory in response to Memory and IO Read TLPs.

The EP_MEM module processes 1 DWORD 32- and 64-bit addressable Memory and IO Write requests based on the information received from the RX Engine, as defined in

Table 9-7. While the memory controller is processing the write, it asserts the `wr_busy_o` output indicating it is busy.

Table 9-7: **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 IO Read requests of one DWORD are processed based on the following inputs, as defined in Table 9-8. After the read request is processed, the data is returned on `rd_data_o[31:0]`.

Table 9-8: **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 9-8 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.

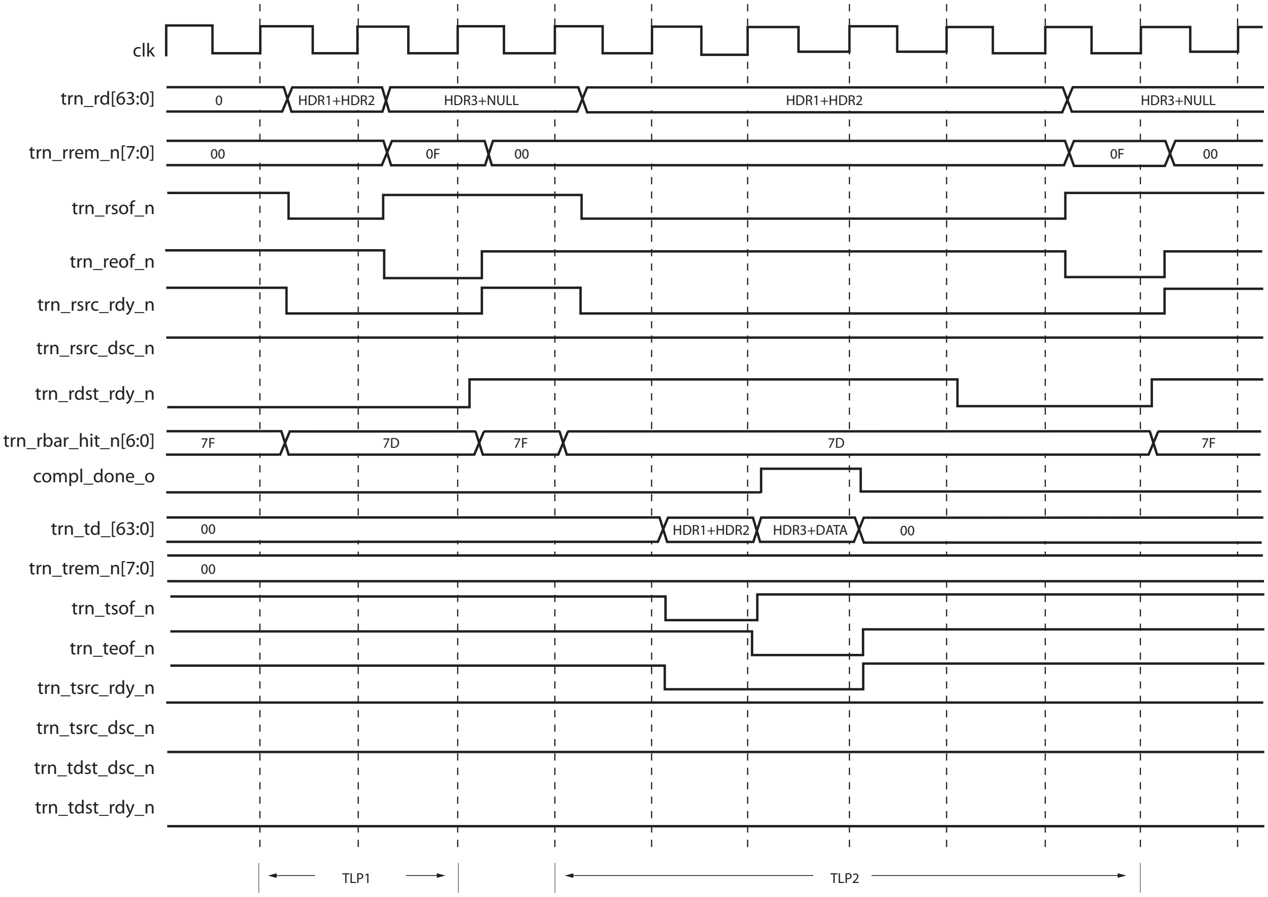


Figure 9-8: **Back-to-Back Read Transactions**

PIO Write Transaction

Figure 9-9 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.

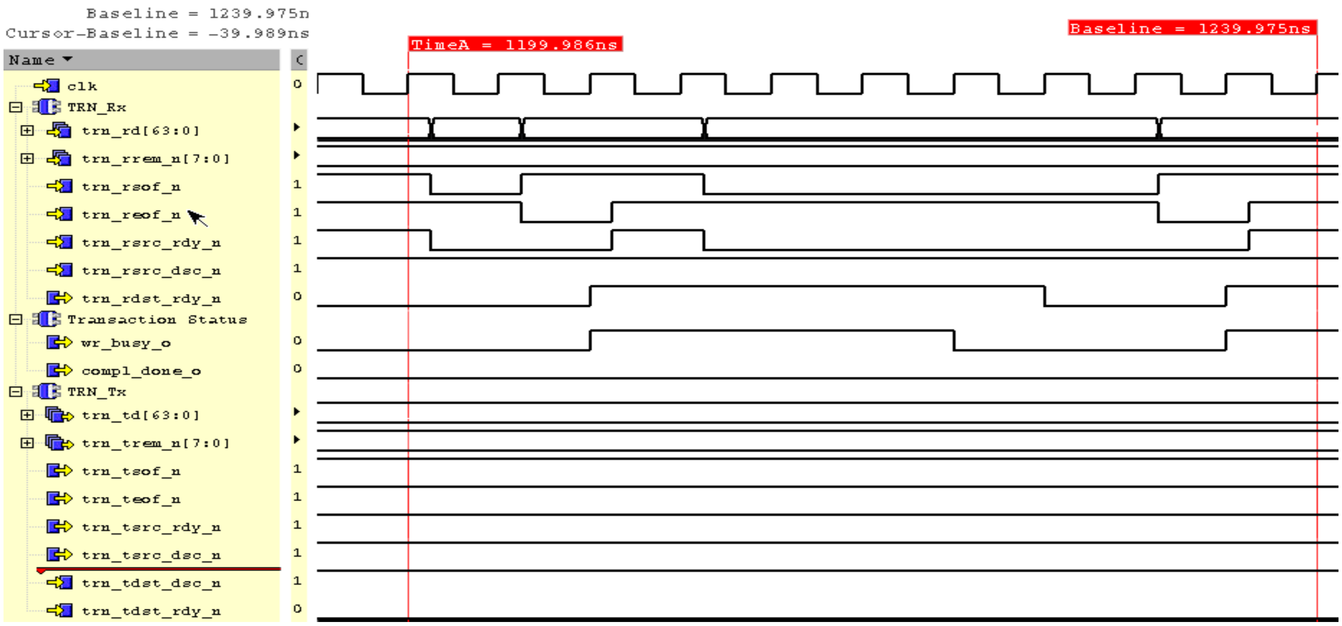


Figure 9-9: **Back-to-Back Write Transactions**

Device Utilization

Table 9-9 shows the PIO design FPGA resource utilization.

Table 9-9: **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 may leverage standard operating system utilities that enable generation of read and write transactions to the target space in the reference design.

Chapter 10

Root Port Model Test Bench

The Endpoint for 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 your own 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 Spartan®-6 FPGA Integrated Endpoint 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 (or Endpoint for PCIe).

Source code for the Root Port Model is included to provide the model for a starting point for your 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 you to dedicate your 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 the following:

- 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

Note: The Spartan-6 Integrated Endpoint Block only includes the Verilog source at this time.

Figure 10-1 illustrates the illustrates the Root Port Model coupled with the PIO design.

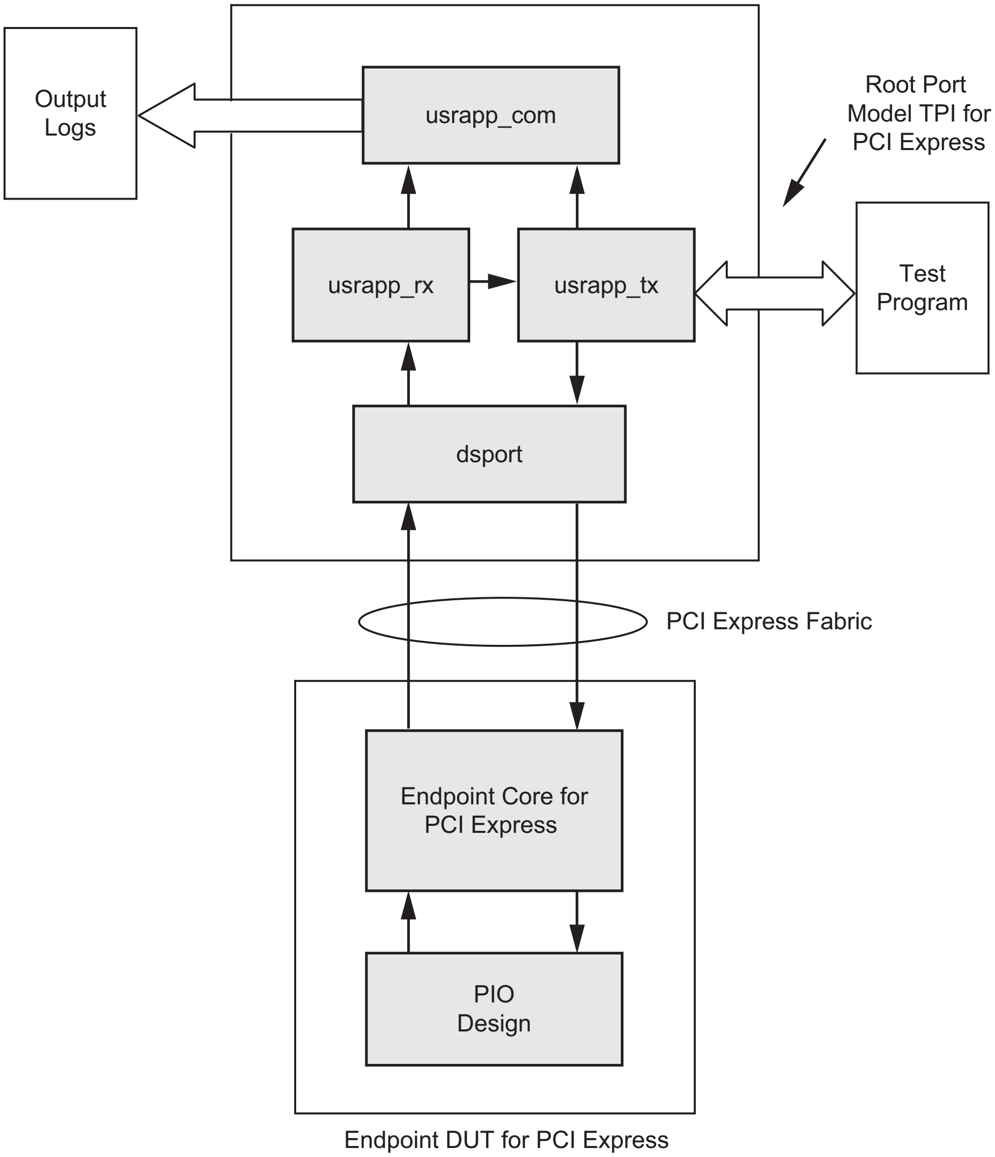


Figure 10-1: **Root Port Model and Top-level Endpoint**

Architecture

The Root Port Model consists of the following blocks, illustrated in Figure 10-1.

- `dsport` (downstream 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 the `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.

The Root Port Model has a 128-byte MPS capability in the receive direction and a 512-byte MPS capability in the transmit direction.

Simulating the Design

Three simulation script files are provided with the model to facilitate simulation with Synopsys® VCS, Cadence® IUS, and Mentor Graphics® ModelSim® simulators:

- `simulate_vcs.sh` (Verilog only)
- `simulate_ncsim.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 the *LogiCORE IP Endpoint for PCI Express Getting Started Guide*.

For IUS users, the work construct must be manually inserted into your CDS.LIB file:

```
DEFINE WORK WORK
```

Test Selection

Table 10-1 describes the tests provided with the Root Port Model, followed by specific sections for VHDL and Verilog test selection.

Table 10-1: **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.
pio_writeReadBack_test0	Verilog and VHDL	Transmits a 1 DWORD Write TLP followed by a 1 DWORD Read TLP to each of the example design's active BARs, and then waits for the Completion TLP and verifies that the write and read data match. The test will send the appropriate TLP to each BAR based on the BARs address type (for example, 32 bit or 64 bit) and space type (for example, IO or Memory).
pio_testByteEnables_test0	Verilog	Issues four sequential Write TLPs enabling a unique byte enable for each Write TLP, and then issues a 1 DWORD Read TLP to confirm that the data was correctly written to the example design. The test will send the appropriate TLP to each BAR based on the BARs address-type (for example, 32 bit or 64 bit) and space type (for example, IO or Memory).
pio_memTestDataBus	Verilog	Determines if the PIO design's FPGA block RAMs data bus interface is correctly connected by performing a 32-bit walking ones data test to the first available BAR in the example design.
pio_memTestAddrBus	Verilog	Determines whether the PIO design's FPGA block RAM's address bus interface is correctly connected by performing a walking ones address test. This test should only be called after successful completion of pio_memTestDataBus.
pio_memTestDevice	Verilog	Checks the integrity of each bit of the PIO design's FPGA block RAM by performing an increment/decrement test. This test should only be called after successful completion of pio_memTestAddrBus.

Table 10-1: **Root Port Model Provided Tests *(Cont'd)***

pio_timeoutFailureExpected	Verilog	Sends a Memory 32 Write TLP followed by Memory 32 Read TLP to an invalid address and waits for a Completion with data TLP. This test anticipates that waiting for the completion TLP times out and illustrates how the test programmer can gracefully handle this event.
pio_tlp_test0 (illustrative example only)	Verilog	Issues a sequence of Read and Write TLPs to the example design's RX interface. Some of the TLPs, for example, burst writes, are not supported by the PIO design.

VHDL Test Selection

Test selection is implemented in the VHDL Root 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. The default generic mapping is to override the `test_selector` with the test name `pio_writeReadBack_test0`. Currently there are two tests defined inside the tests entity, `sample_smoke_test0` and `pio_writeReadBack_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 you 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 IUS simulator explicitly specifies the test `sample_smoke_test0` to be run using the following command line syntax:

```
ncsim work.boardx01 +TESTNAME=sample_smoke_test0
```

You can change the test to be run by changing 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.

VHDL and Verilog Root Port Model Differences

The following sections 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, you may 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 10-2 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 10-2: **Simulator Dump File Format**

Simulator	Dump File Format
Synopsys VCS	.vpd
ModelSim	.vcd
Cadence IUS	.trn

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 IUS 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` (IUS). This command line can be found in the respective simulation script files `simulate_mti.do` and `simulate_ncsim.sh`.

ModelSim

The following 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
```

IUS

The following command line initiates waveform dumping for the IUS 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 IUS simulator) can indicate to the Root Port Model that the waveform should be saved to a file using the following 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 may 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 154 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.

Note that 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 may 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," page 154 displays the listing of a simple test program *pio_writeReadBack_test0*, written for use in conjunction with the PIO endpoint. This test program is located in the file `pio_tests.v`. As the test name implies, this test performs a one DWORD write operation *to* the PIO Design followed by a 1 DWORD read operation *from* the PIO Design, after which it compares the values to confirm that they are equal. The test is performed on the first location in each of the active Mem32 BARs of the PIO Design. For the default configuration, this test performs the write and read back to BAR2 and to the EROM space (BAR6) (Block Plus only). The following section outlines the steps performed by the test program.

- Line 1 of the sample program determines if the user has selected the test program pio_writeReadBack_test1 when invoking the Verilog simulator.
- Line 4 of the sample program invokes the TPI call TSK_SIMULATION_TIMEOUT which sets the master timeout value to be long enough for the test to complete.
- Line 5 of the sample program invokes the TPI call TSK_SYSTEM_INITIALIZATION. This task will cause the test program to wait for the system reset to deassert as well as the endpoint's trn_lnk_up_n signal to assert. This is an indication that the endpoint is ready to be configured by the test program via the Root Port Model.
- Line 6 of the sample program uses the TPI call TSK_BAR_INIT. This task will perform a series of Type 0 Configuration Writes and Reads to the Endpoint core's PCI Configuration Space, determine the memory and IO requirements of the endpoint, and then program the endpoint's Base Address Registers so that it is ready to receive TLPs from the Root Port Model.
- Lines 7, 8, and 9 of the sample program work together to cycle through all the endpoint's BARs and determine whether they are enabled, and if so to determine their type, for example, Mem32, Mem64, or IO).

Note that all PIO tests provided with the Root Port Model are written in a form that does not assume that a specific BAR is enabled or is of a specific type (for example, Mem32, Mem64, IO). These tests perform on-the-fly BAR determination and execute TLP transactions dependent on BAR types (that is, Memory32 TLPs to Memory32 Space, IO TLPs to IO Space, and so forth). This means that if a user reconfigures the BARs of the Endpoint, the PIO continues to work because it dynamically explores and configures the BARs. Users are not required to follow the form used and can create tests that assume their own specific BAR configuration.

- Line 7 sets a counter to increment through all of the endpoint's BARs.

- Line 8 determines whether the BAR is enabled by checking the global array BAR_INIT_P_BAR_ENABLED[]. A non-zero value indicates that the corresponding BAR is enabled. If the BAR is not enabled then test program flow will move on to check the next BAR. The previous call to TSK_BAR_INIT performed the necessary configuration TLP communication to the endpoint device and filled in the appropriate values into the BAR_INIT_P_BAR_ENABLED[] array.
- Line 9 performs a case statement on the same global array BAR_INIT_P_BAR_ENABLED[]. If the array element is enabled (that is, non-zero), the element's value indicates the BAR type. A value of 1, 2, and 3 indicates IO, Memory 32, and Memory 64 spaces, respectively.

  If the BAR type is either IO or Memory 64, then the test does not perform any TLP transactions. If the BAR type is Memory 32, program control continues to line 16 and starts transmitting Memory 32 TLPs.
- Lines 21-26 use the TPI call TSK_TX_MEMORY_WRITE_32 and transmits a Memory 32 Write TLP with the payload DWORD '01020304' to the PIO endpoint.
- Lines 32-33 use the TPI calls TSK_TX_MEMORY_READ_32 followed by TSK_WAIT_FOR_READ_DATA in order to transmit a Memory 32 Read TLP and then wait for the next Memory 32 Completion with Data TLP. In case the Root Port Model never receives the Completion with Data TLP, the TPI call TSK_WAIT_FOR_READ_DATA would locally timeout and display an error message.
- Line 34 compares the DWORD received from the Completion with Data TLP with the DWORD that was transmitted to the PIO endpoint and displays the appropriate success or failure message.

Test Program: pio_writeReadBack_test0

```
else if(testname == "pio_writeReadBack_test1"
begin
// This test performs a 32 bit write to a 32 bit Memory space and performs a read back
TSK_SIMULATION_TIMEOUT(10050);
TSK_SYSTEM_INITIALIZATION;
TSK_BAR_INIT;
for (ii = 0; ii <= 6; ii = ii + 1) begin
    if (BAR_INIT_P_BAR_ENABLED[ii] > 2'b00) // bar is enabled
     case(BAR_INIT_P_BAR_ENABLED[ii])
             2'b01 : // IO SPACE
            begin
                 $display("[%t] : NOTHING: to IO 32 Space BAR %x", $realtime, ii);
            end
             2'b10 : // MEM 32 SPACE
               begin
                $display("[%t] : Transmitting TLPs to Memory 32 Space BAR %x",
                             $realtime, ii);
        //-----------------------------------------------------------------------
        // Event : Memory Write 32 bit TLP
        //-----------------------------------------------------------------------
                  DATA_STORE[0] = 8'h04;
                  DATA_STORE[1] = 8'h03;
                  DATA_STORE[2] = 8'h02;
                  DATA_STORE[3] = 8'h01;
                  P_READ_DATA = 32'hffff_ffff; // make sure P_READ_DATA has known initial value
                  TSK_TX_MEMORY_WRITE_32(DEFAULT_TAG, DEFAULT_TC, 10'd1, BAR_INIT_P_BAR[ii][31:0] , 4'hF,
4'hF, 1'b0);
                  TSK_TX_CLK_EAT(10);
                  DEFAULT_TAG = DEFAULT_TAG + 1;
            //-----------------------------------------------------------------------
            // Event : Memory Read 32 bit TLP
            //-----------------------------------------------------------------------
                  TSK_TX_MEMORY_READ_32(DEFAULT_TAG, DEFAULT_TC, 10'd1, BAR_INIT_P_BAR[ii][31:0], 4'hF,
4'hF);
                  TSK_WAIT_FOR_READ_DATA;
                  if  (P_READ_DATA != {DATA_STORE[3], DATA_STORE[2], DATA_STORE[1], DATA_STORE[0] })
                    begin
                     $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);
                   end
                else
                  begin
                     $display("[%t] : Test PASSED --- Write Data: %x successfully received", $realtime,
P_READ_DATA);
                  end
                 TSK_TX_CLK_EAT(10);
                 DEFAULT_TAG = DEFAULT_TAG + 1;
              end
        2'b11 : // MEM 64 SPACE
              begin
                $display("[%t] : NOTHING: to Memory 64 Space BAR %x", $realtime, ii);
          end
        default : $display("Error case in usrapp_tx\n");
        endcase
      end
     $display("[%t] : Finished transmission of PCI-Express TLPs", $realtime);
     $finish;
            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. 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 IO 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 will cause 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 10-3 through 10-7.

- "Test Setup Tasks"
- "TLP Tasks"
- "BAR Initialization Tasks"
- "Example PIO Design Tasks"
- "Expectation Tasks"

Table 10-3: **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 10-4: **TLP Tasks**

Name	Input(s)		Description
TSK_TX_TYPE0_CONFIGURATION_READ	tag_ reg_addr_ first_dw_be_	7:0 11:0 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_ reg_addr_ first_dw_be_	7:0 11:0 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 Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
TSK_TX_TYPE0_CONFIGURATION_WRITE	tag_ reg_addr_ reg_data_ first_dw_be_	7:0 11:0 31:0 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. Cpl returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
TSK_TX_TYPE1_CONFIGURATION_WRITE	tag_ reg_addr_ reg_data_ first_dw_be_	7:0 11:0 31:0 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. Cpl returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
TSK_TX_MEMORY_READ_32	tag_ tc_ len_ addr_ last_dw_be_ first_dw_be_	7:0 2:0 9:0 31:0 3:0 3:0	Sends a PCI Express Memory Read TLP from downstream port to 32 bit memory address addr_ of Endpoint DUT. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
TSK_TX_MEMORY_READ_64	tag_ tc_ len_ addr_ last_dw_be_ first_dw_be_	7:0 2:0 9:0 63:0 3:0 3:0	Sends a PCI Express Memory Read TLP from Root Port Model to 64 bit memory address addr_ of Endpoint DUT. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.

Table 10-4: **TLP Tasks** ***(Cont'd)***

Name	Input(s)		Description
TSK_TX_MEMORY_WRITE_32	tag_	7:0	Sends a PCI Express Memory Write TLP from Root Port Model to 32 bit memory address addr_ of Endpoint DUT. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID. The global DATA_STORE byte array is used to pass write data to task.
	tc_	2:0	
	len_	9:0	
	addr_	31:0	
	last_dw_be_	3:0	
	first_dw_be_	3:0	
	ep_	–	
TSK_TX_MEMORY_WRITE_64	tag_	7:0	Sends a PCI Express Memory Write TLP from Root Port Model to 64 bit memory address addr_ of Endpoint DUT. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID. The global DATA_STORE byte array is used to pass write data to task.
	tc_	2:0	
	len_	9:0	
	addr_	63:0	
	last_dw_be_	3:0	
	first_dw_be_	3:0	
	ep_	–	
TSK_TX_COMPLETION	tag_	7:0	Sends a PCI Express Completion TLP from Root Port Model to Endpoint DUT using global COMPLETE_ID_CFG as completion ID.
	tc_	2:0	
	len_	9:0	
	comp_status_	2:0	
TSK_TX_COMPLETION_DATA	tag_	7:0	Sends a PCI Express Completion with Data TLP from Root Port Model to Endpoint DUT using global COMPLETE_ID_CFG as completion ID. The global DATA_STORE byte array is used to pass completion data to task.
	tc_	2:0	
	len_	9:0	
	byte_count	11:0	
	lower_addr	6:0	
	comp_status	2:0	
	ep_	–	
TSK_TX_MESSAGE	tag_	7:0	Sends a PCI Express Message TLP from Root Port Model to Endpoint DUT. Completion returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
	tc_	2:0	
	len_	9:0	
	data	63:0	
	mesage_rtg	2:0	
	message_code	7:0	
TSK_TX_MESSAGE_DATA	tag_	7:0	Sends a PCI Express Message with Data TLP from Root Port Model to Endpoint DUT. The global DATA_STORE byte array is used to pass message data to task. Completion returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
	tc_	2:0	
	len_	9:0	
	data	63:0	
	mesage_rtg	2:0	
	message_code	7:0	

Table 10-4: **TLP Tasks** ***(Cont'd)***

Name	Input(s)		Description
TSK_TX_IO_READ	tag_ addr_ first_dw_be_	7:0 31:0 3:0	Sends a PCI Express IO Read TLP from Root Port Model to IO address addr_[31:2] of Endpoint DUT. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
TSK_TX_IO_WRITE	tag_ addr_ first_dw_be_ data	7:0 31:0 3:0 31:0	Sends a PCI Express IO Write TLP from Root Port Model to IO address addr_[31:2] of Endpoint DUT. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.
TSK_TX_BAR_READ	bar_index byte_offset tag_ tc_	2:0 31:0 7:0 2:0	Sends a PCI Express 1 DWORD Memory 32, Memory 64, or IO 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. CplD returned from Endpoint DUT will use contents of global COMPLETE_ID_CFG as completion ID.

Table 10-4: **TLP Tasks** ***(Cont'd)***

Name	Input(s)		Description
TSK_TX_BAR_WRITE	bar_index byte_offset tag_ tc_ data_	2:0 31:0 7:0 2:0 31:0	Sends a PCI Express 1 DWORD Memory 32, Memory 64, or IO Write TLP from the Root Port to the target address corresponding to offset byte_offset from BAR bar_index of the Endpoint DUT. 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. 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 10-5: **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 IO space mapping calculations, and finally programs the Endpoint so that it is ready to be accessed. On completion, the user test program may begin memory and IO transactions to the device. This function displays to standard output a memory/IO 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 IO 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/IO mapping algorithm and allocates Memory 32, Memory 64, and IO 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 10-6: **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 IO 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 nBytes	2:0 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 IO 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 nBytes	2:0 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 10-7: **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. Note that 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 10-7: **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. Note that 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	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. Note that 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		
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. Note that 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	61:0		
TSK_EXPECT_IOWR	td	-	Expect status	Waits for an IO Write TLP with matching header fields. Returns a 1 on successful completion; 0 otherwise. Note that this task can only be used in conjunction with Bus Master designs.
	ep	-		
	requester_id	15:0		
	tag	7:0		
	first_dw_be	3:0		
	address	31:0		
	data	31:0		

Chapter 11

Migration Considerations

Migrating a design from a LogiCORE Endpoint PIPE for PCI Express to a LogiCORE IP Spartan-6 FPGA Integrated Endpoint BLOCK for PCI Express is a simple process. This chapter describes the changes in the interfaces and signals that are necessary when migrating from the Spartan-3 Endpoint PIPE core to a Spartan-6 Integrated Endpoint Block core.

Integrated PHY

The first and most notable change between the Spartan-3 Endpoint PIPE core and the Spartan-6 Integrated Endpoint Block core is that the external SerDes PHY chip that previously resided outside of the chip has been integrated into the Spartan-6 architecture. This means that all of the PXPIPE signals (33 ports) are replaced with the serial interface (4 ports).

System Clocking and Reset

For the Spartan-6 Integrated Endpoint Block, the `sys_clk` signal has been added.

In the Spartan-3 design, the system clock was provided by the external PHY on the port, RXCLK. For more information about `sys_clk` and how to set it up, please see "Clocking" in Chapter 6.

Interface Changes

Streaming Signal Added

A new signal, `trn_tstr_n`, to allow packets to be streamed in the transmit direction has been added. For more information on `trn_tstr_n`, see Table 2-6, page 29.

TRN Transmit Destination Discontinue Removed

The `trn_tdst_dsc_n` signal has been replaced with `trn_tx_terr_drop_n`. The signal serves the same purpose to signal when a packet has been dropped. However, the signal will now assert 1 to 2 clock cycles after end of the packet that was dropped. The User Application is not required to do anything in response to `trn_terr_drop_n`; it is intended for diagnosing problems when bringing up new designs. This makes timing significantly easier to meet.

TRN Buffer Available Size Change

The Spartan-3 Endpoint PIPE signal `trn_tbuf_av[4:0]` is now one bit wider. The signal is now `trn_tbuf_av[5:0]`. This change reflects the increased number of transmit buffers supported by the Spartan-6 core.

CMM Arbitration

The Spartan-6 Integrated Endpoint Block design has two new signals that allow for the user to control the arbitration between the CMM and the TRN interfaces for transmitted packets. These two new signals are `trn_tcfg_req_n` and `trn_tcfg_gnt_n`.

To maintain the same behavior as the Spartan-3 Endpoint PIPE, assign the signal `trn_tcfg_gnt_n` asserted (1'b0).

TRN Credit Busses Additional Functionality

The TRN credit buses have new names in the Spartan-6 Integrated Endpoint Block design. The letter 'r' has been removed from the signal names. Table 11-1 shows the old and new names.

Table 11-1: **Credit Bus Name Change from Spartan-3 to Spartan-6 Devices**

Spartan-3 FPGAs	Spartan-6 FPGAs
trn_rfc_nph	trn_fc_nph
trn_rfc_npd	trn_fc_npd
trn_rfc_ph	trn_fc_ph
trn_rfc_pd	trn_fc_pd
trn_rfc_cplh	trn_fc_cplh
trn_rfc_cpld	trn_fc_cpld

There is a also a new signal named `trn_fc_sel[2:0]` that controls what values are placed on the `trn_*` bus.

To maintain the same behavior as the Spartan-3 Endpoint PIPE, set the `trn_fc_sel[2:0]` signal to 3'b000. For more information, see "“Flow Control Credit Information” in Chapter 6.

Configuration Error Completion Ready

A new signal named `cfg_err_cpl_rdy_n` has been added for the Spartan-6 Integrated Endpoint Block. For more information, see Table 2-9, page 35.

Configuration Error Locked

A new signal named `cfg_err_locked_n` has been added for the Spartan-6 Integrated Endpoint Block. For more information, see Table 2-9, page 35.

Removed Configuration Signals

The following signals were removed because they were either unused or not needed:

- `cfg_di`
- `cfg_wr_en_n`
- `cfg_byte_en_n`
- `cfg_err_cpl_unexpected_n`

Hot Reset

A new signal named `received_hot_reset` has been added for the Spartan-6 Integrated Endpoint Block. For more information, see Table 2-3, page 26.

Block RAM Settings

The block RAM settings can now be customized. See the CORE Generator GUI for supported settings.

Signal Change Summary

The following signals have been added for the Spartan-6 Integrated Endpoint Block:

- `sys_clk`
- `trn_tstr_n`
- `trn_tcfg_req_n`
- `trn_tcfg_gnt_n`
- `cfg_err_cpl_rdy_n`
- `cfg_err_locked_n`
- `received_hot_reset`
- `trn_fc_sel[2:0]`

The following signals have been removed for the Spartan-6 Integrated Endpoint Block:

- `cfg_di`
- `cfg_wr_en_n`
- `cfg_byte_en_n`
- `cfg_err_cpl_unexpected_n`

Table 11-2 shows the signal name changes.

Table 11-2: **Signal Name Change from Spartan-3 to Spartan-6 Devices**

Spartan-3 FPGAs	Spartan-6 FPGAs
trn_rfc_nph	trn_fc_nph
trn_rfc_npd	trn_fc_npd
trn_rfc_ph	trn_fc_ph
trn_rfc_pd	trn_fc_pd
trn_rfc_cplh	trn_fc_cplh
trn_rfc_cpld	trn_fc_cpld

Table 11-2: **Signal Name Change from Spartan-3 to Spartan-6 Devices *(Cont'd)***

Spartan-3 FPGAs	Spartan-6 FPGAs
trn_tbuf_av[4:0]	trn_tbuf_av[5:0]
trn_tdst_dsn_n	trn_tx_terr_drop_n

Chapter 12

Known Restrictions

There are no known restrictions.

Chapter 13

Debugging Designs

This chapter 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 Spartan®-6 Integrated Block for PCI Express. This chapter uses flow diagrams to guide the user through the debug process.

The following information is found in this chapter:

- "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 Integrated Block for PCI Express, 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 Spartan-6 Integrated Endpoint Block, as shown in Table 13-1.

Table 13-1: **Spartan-6 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 **Spartan-6** from the **Device List** drop-down menu.
- This will sort all available Spartan-6 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 Spartan-6 Integrated Endpoint 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 Spartan-6 Integrated Endpoint Block for PCI Express, search for the phrase "Spartan-6 Integrated Endpoint 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 speedgrade
- 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 project

Additional files may 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 chapter 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 will appropriately handle 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 Chapter 9, "Programmed Input Output Example Design."

Chipscope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O software cores directly into your 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

Below is a description of 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 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 will display the extended configuration space of the device. It may be useful to read the extended configuration space on the root and look for the Advanced Error Reporting (AER) registers. These registers will provide more information on why the device has flagged an error (for example, it may show that a correctable error was issued because of a replay timer timeout).

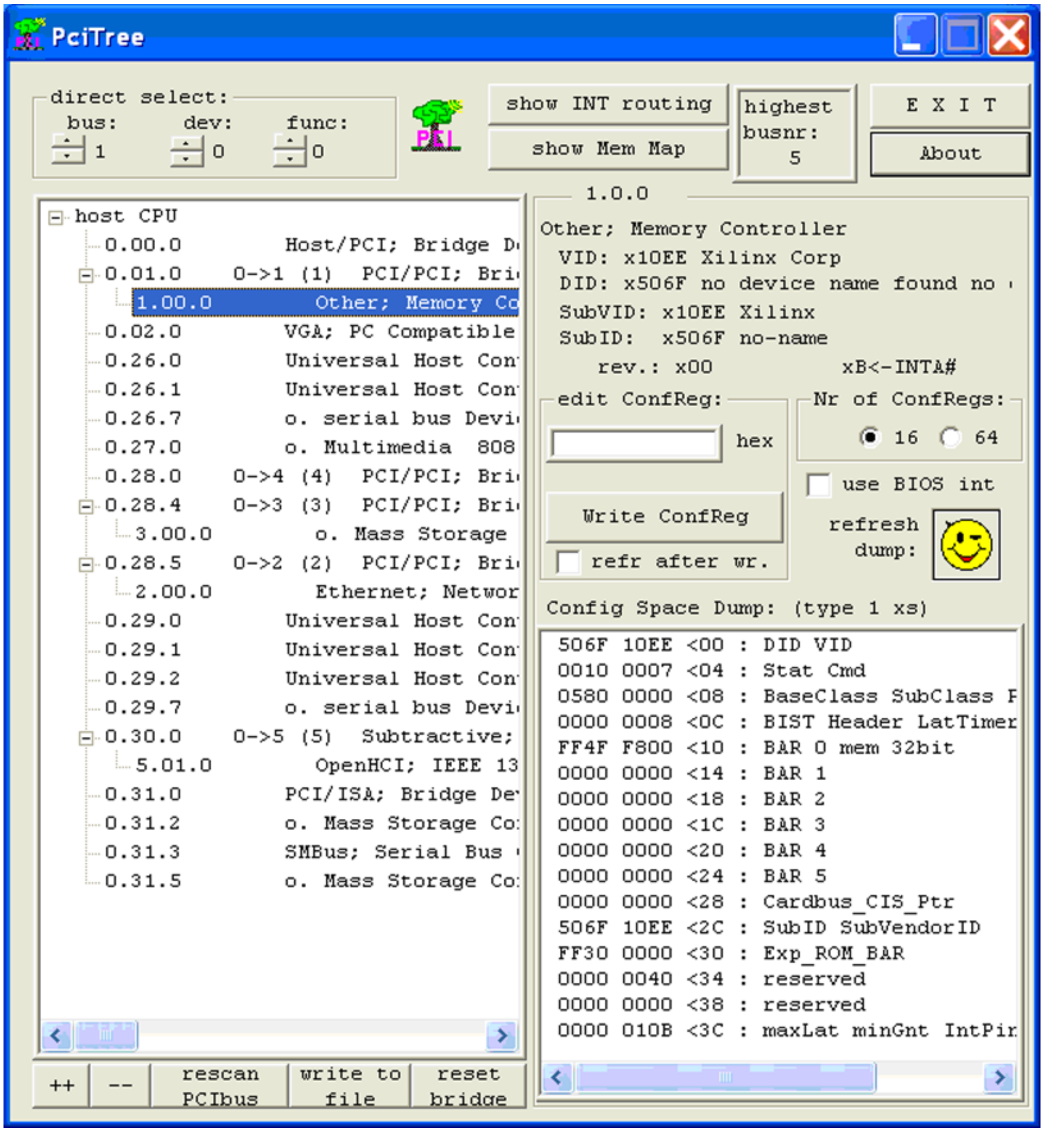
- `lspci -k`

  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 13-1.



```
host CPU
  0.00.0             Host/PCI; Bridge De
  0.01.0   0->1 (1)  PCI/PCI; Bri
    1.00.0             Other; Memory Co
  0.02.0             VGA; PC Compatible
  0.26.0             Universal Host Con
  0.26.1             Universal Host Con
  0.26.7             o. serial bus Devi
  0.27.0             o. Multimedia  808
  0.28.0   0->4 (4)  PCI/PCI; Bri
  0.28.4   0->3 (3)  PCI/PCI; Bri
    3.00.0             o. Mass Storage
  0.28.5   0->2 (2)  PCI/PCI; Bri
    2.00.0             Ethernet; Networ
  0.29.0             Universal Host Con
  0.29.1             Universal Host Con
  0.29.2             Universal Host Con
  0.29.7             o. serial bus Devi
  0.30.0   0->5 (5)  Subtractive;
    5.01.0             OpenHCI; IEEE 13
  0.31.0             PCI/ISA; Bridge De
  0.31.2             o. Mass Storage Co
  0.31.3             SMBus; Serial Bus
  0.31.5             o. Mass Storage Co
```

```
1.0.0
Other; Memory Controller
  VID: x10EE Xilinx Corp
  DID: x506F no device name found no
  SubVID: x10EE Xilinx
  SubID:  x506F no-name
    rev.: x00            xB<-INTA#
```

```
Config Space Dump: (type 1 xs)
506F 10EE <00 : DID VID
0010 0007 <04 : Stat Cmd
0580 0000 <08 : BaseClass SubClass P
0000 0008 <0C : BIST Header LatTimer
FF4F F800 <10 : BAR 0 mem 32bit
0000 0000 <14 : BAR 1
0000 0000 <18 : BAR 2
0000 0000 <1C : BAR 3
0000 0000 <20 : BAR 4
0000 0000 <24 : BAR 5
0000 0000 <28 : Cardbus_CIS_Ptr
506F 10EE <2C : SubID SubVendorID
FF30 0000 <30 : Exp_ROM_BAR
0000 0040 <34 : reserved
0000 0000 <38 : reserved
0000 010B <3C : maxLat minGnt IntPir
```

Figure 13-1: **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).

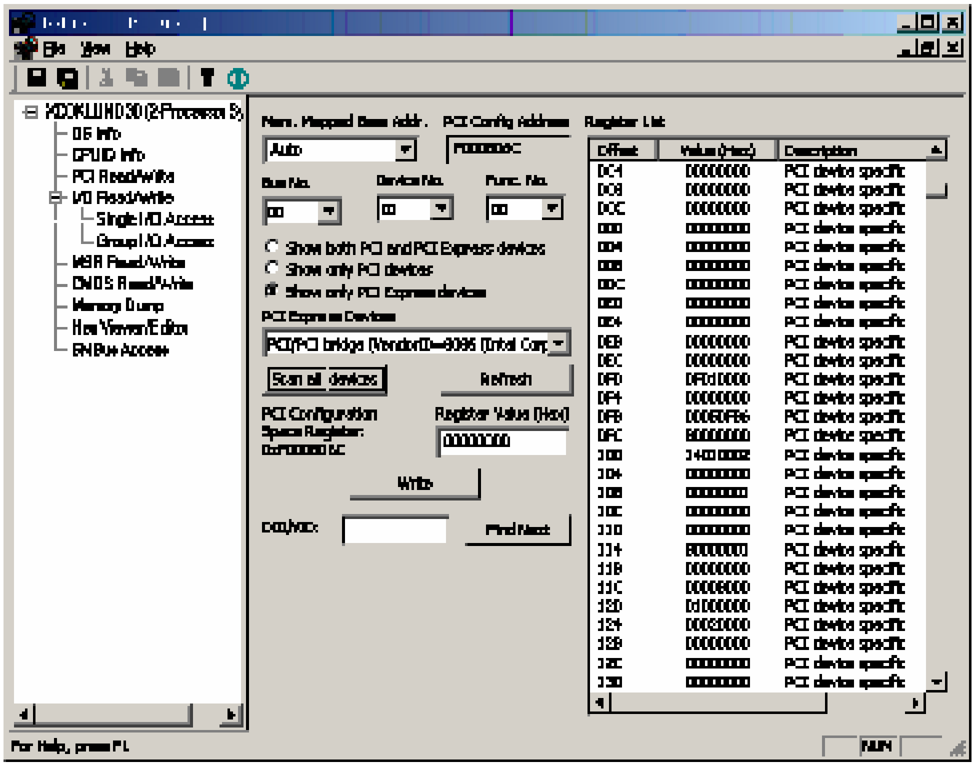


Figure 13-2: **HWDIRECT with Read of Configuration Space**

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.

Debug Ports

The Spartan-6 Integrated Block for PCI Express has debug ports described in Table x-x providing insight to why the different error conditions occur. The receiver may detect different problems that will result in either a Fatal, Non-fatal, or correctable error. Also, the receiver may detect an unsupported request. Four of the debug signals shown in Table 13-2 mirror the lower four bits of the PCI Express device status register. When one of these conditions occurs, another signal will assert for one clock cycle to show the reason causing the error.

Table 13-2: **Device Status Register Debug Ports**

Name	Device Status Bit
dbg_reg_detected_correctable	Bit 0 - correctable
dbg_reg_detected_non_fatal	Bit 1 - Non-Fatal

Table 13-2: **Device Status Register Debug Ports *(Cont'd)***

Name	Device Status Bit
dbg_reg_detected_fatal	Bit 2 - Fatal
dbg_reg_detected_unsupported	Bit 3 - Unsupported Request

Table 13-3 defines the debug port signals.

Table 13-3: **Spartan-6 Integrated Block for PCI Express Debug Ports**

Port	Direction	Clock Domain	Description
dbg_bad_dllp_status	Output	USERCLK	This signal pulses high for one USERCLK cycle when a DLLP CRC error is detected.
dbg_bad_dllp_status	Output	USERCLK	This signal pulses high for one USERCLK cycle when a TLP with an LCRC error is detected.
dbg_bad_tlp_lcrc	Output	USERCLK	This signal pulses high for one USERCLK cycle when a TLP with an invalid sequence number is detected.
dbg_bad_tlp_seq_num	Output	USERCLK	This signal pulses high for one USERCLK cycle when a bad TLP is detected, for reasons other than a bad LCRC or a bad sequence number.
dbg_bad_tlp_status	Output	USERCLK	This signal pulses high for one USERCLK cycle if an out-of-range ACK or NAK is received.
dbg_dl_protocol_status	Output	USERCLK	This signal pulses high for one USERCLK cycle if there is a protocol error with the received flow control updates.
dbg_fc_protocol_err_status	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a received TLP had a length that did not match what was in the TLP header.
dbg_mlfrmd_length	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a received TLP had a length in violation of the negotiated MPS.
dbg_mlfrmd_mps	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a received TLP had an invalid TC or VC value.
dbg_mlfrmd_tcvc	Output	USERCLK	This signal pulses high for one USERCLK cycle when a malformed TLP is received. See the other DBGMLFRMD* signals for further clarification. **Note:** There is skew between DBGMLFRMD* and DBGMLFRMDTLPSTATUS.
dbg_mlfrmd_unrec_type	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a received TLP had an invalid/unrecognized type field value.
dbg_poistlpstatus	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a TLP was received with the EP (poisoned) status bit set.

Table 13-3: **Spartan-6 Integrated Block for PCI Express Debug Ports *(Cont'd)***

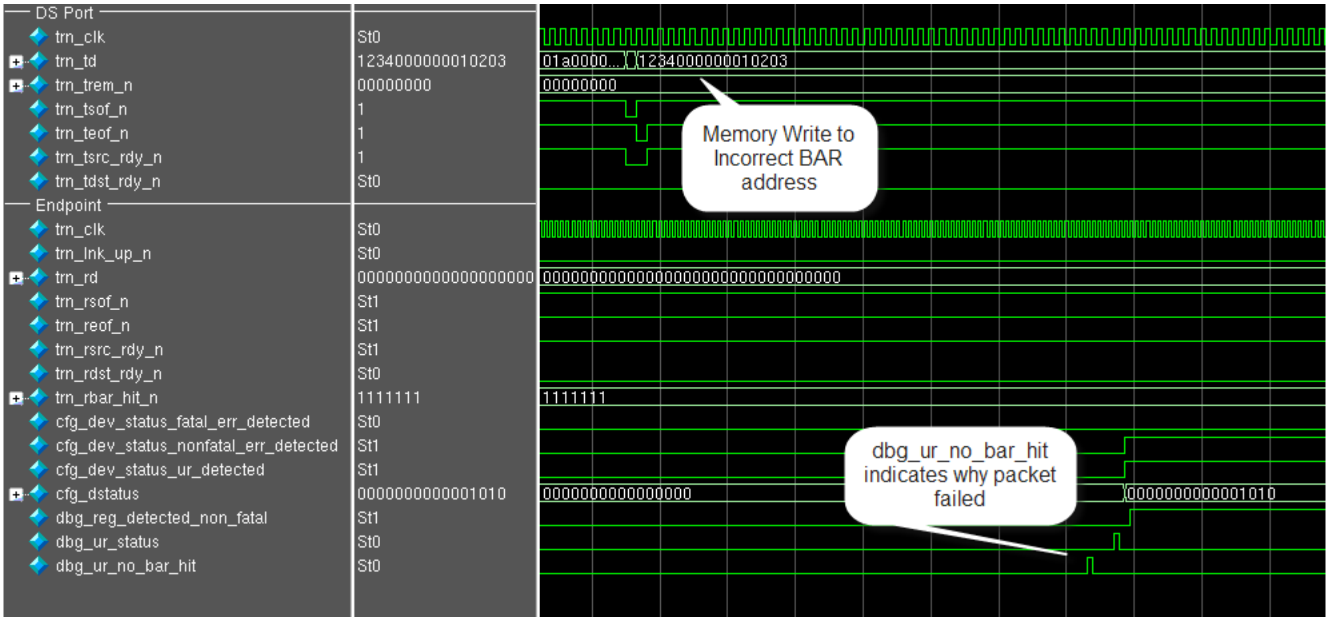
Port	Direction	Clock Domain	Description
dbg_mlfrmd_tlp_status	Output	USERCLK	This signal pulses high for one USERCLK cycle if a poisoned TLP is received.
dbg_rcvr_overflow_status	Output	USERCLK	This signal pulses high for one USERCLK cycle if a received TLP violates the advertised credit.
dbg_reg_detected_correctable	Output	USERCLK	This signal is a mirror of the internal signal used to indicate a correctable error is detected. The error is cleared upon a read by the Root Complex (RC).
dbg_reg_detected_fatal	Output	USERCLK	This signal is a mirror of the internal signal used to indicate that a fatal error has been detected. The error is cleared upon a read by the RC.
dbg_reg_detected_non_fatal	Output	USERCLK	This signal is a mirror of the internal signal used to indicate that a non-fatal error has been detected. The error is cleared upon a read by the RC.
dbg_reg_detected_unsupported	Output	USERCLK	This signal is a mirror of the internal signal used to indicate that an unsupported request has been detected. The error is cleared upon a read by the RC.
dbg_rply_rollover_status	Output	USERCLK	This signal pulses high for one USERCLK cycle when the rollover counter expires.
dbg_rply_timeout_status	Output	USERCLK	This signal pulses high for one USERCLK cycle when the replay time-out counter expires.
dbg_ur_no_bar_hit	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a received read or write request did not match any configured BAR.
dbg_ur_pois_cfg_wr	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that a CfgWr TLP with the Error/Poisoned bit (EP) = 1 was received.
dbg_ur_status	Output	USERCLK	This signal pulses high for one USERCLK cycle when an unsupported request is received. See the DBGUR* signals for further clarification. **Note:** There is skew between DBGUR* and DBGURSTATUS.
dbg_ur_unsup_msg	Output	USERCLK	This signal pulses high for one USERCLK cycle to indicate that an Msg or MsgD TLP with an unsupported type was received.

Using the Debug Ports

The debug ports are outputs on the integrated block and users can access them by opening the wrapper source file in the source directory. This file is named `<corename>.v[hd]` where *<corename>* represents core name entered in CORE Generator. Signals are defined for each of these ports as either wires in the verilog version or signals in the VHDL version.

The debug ports can be used in both simulation and in hardware to debug problems. In simulation, these signals can easily be added to the waveform viewer without any changes to the code since they are already defined in the wrapper file. In hardware, users may want to use ChipScope Pro to monitor these signals or probe them to external ports or add additional logic such as counters to enable more in depth analysis. Users may need to bring these signals to upper layers in the design and this can be done by modifying the port description and instantiations of the files.

Figure 13-3 shows a common problem faced by many users. This illustrates the behavior of the debug ports when a non-fatal error condition occurs. The scenario is a memory write is sent from the downstream port to the Spartan-6 endpoint. This memory write does not correctly target any of the available BARs in the design. This results in a non-fatal error condition. However, there are other reasons that will cause non-fatal error so monitoring the `cfg_dev_status_nonfatal_err_detected` output of the core may not be sufficient to debug the problem. By using the debug ports, the user can see that the non-fatal error was caused by a BAR miss due to `dbg_ur_no_bar_hit` asserts for one cycle.



DS Port	
trn_clk	St0
trn_td	1234000000010203
trn_trem_n	00000000
trn_tsof_n	1
trn_teof_n	1
trn_tsrc_rdy_n	1
trn_tdst_rdy_n	St0
Endpoint	
trn_clk	St0
trn_lnk_up_n	St0
trn_rd	00000000000000000000
trn_rsof_n	St1
trn_reof_n	St1
trn_rsrc_rdy_n	St1
trn_rdst_rdy_n	St0
trn_rbar_hit_n	1111111
cfg_dev_status_fatal_err_detected	St0
cfg_dev_status_nonfatal_err_detected	St1
cfg_dev_status_ur_detected	St1
cfg_dstatus	0000000000001010
dbg_reg_detected_non_fatal	St1
dbg_ur_status	St0
dbg_ur_no_bar_hit	St0

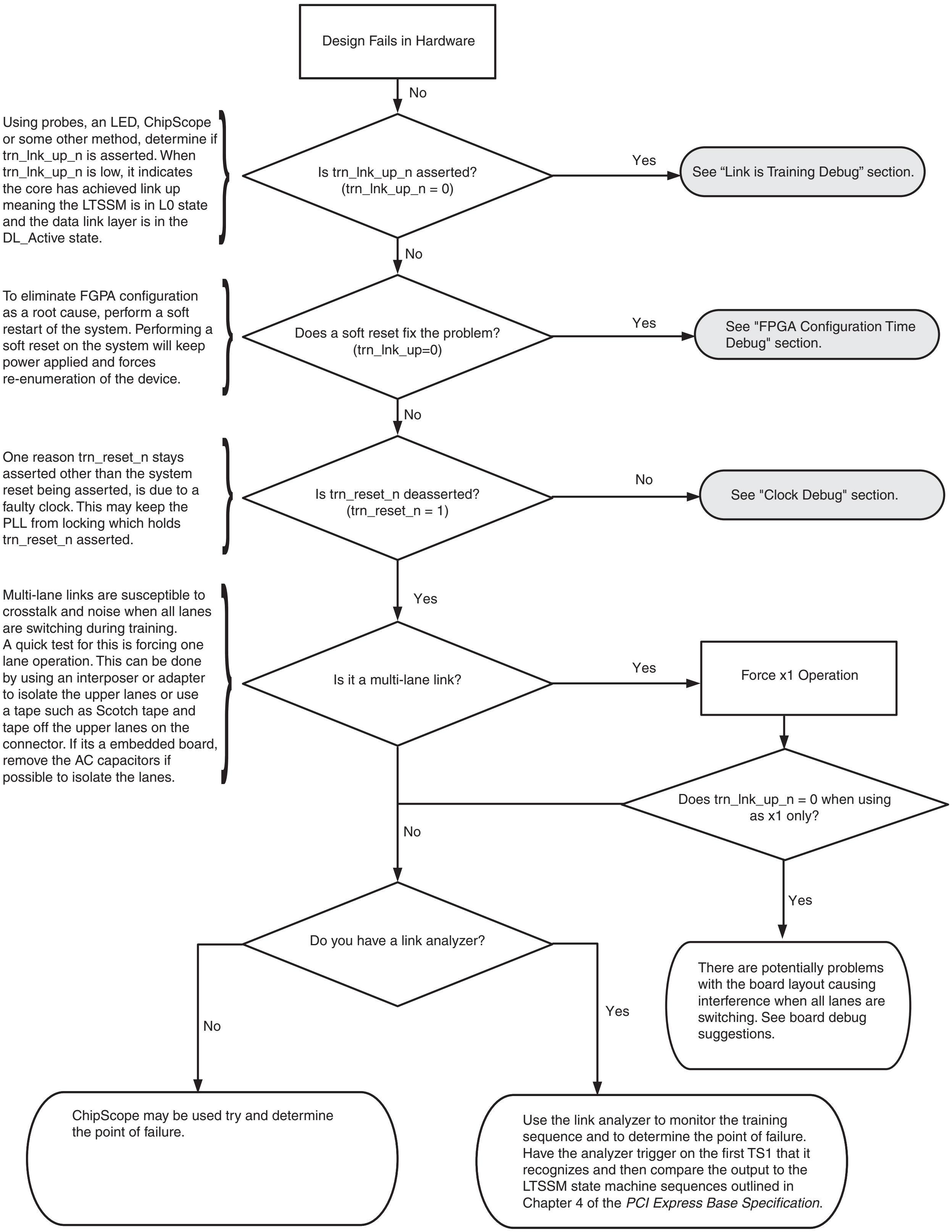
01a0000...
1234000000010203
00000000
Memory Write to Incorrect BAR address
1111111
dbg_ur_no_bar_hit indicates why packet failed
0000000000000000
0000000000001010

Figure 13-3: **Debug Wave Screenshot**

In ChipScope, the user will need to decide how best to trigger ChipScope to capture these problems. There are various ways to do this, but one suggestion would be to use the signals in Table 13-3 as triggers. At least one of these signals will assert. Once chipscope triggers, the rest of the signals can be analyzed to find out exactly what caused the error condition.

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 the figure.


Design Fails in Hardware
No
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
Does trn_lnk_up_n = 0 when using as x1 only?
No
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
Yes
ChipScope may be used try and determine the point of failure.
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.

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.

Figure 13-4: **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, v1.1* states two rules that may 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 there is a finite time in which the FPGA must be configured by, and not meeting these requirements could cause problems with link training and device recognition.

Configuration may be accomplished using an on-board 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 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 the device links up and is recognized after a soft reset is performed, then FPGA configuration is most likely the problem. Note that 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

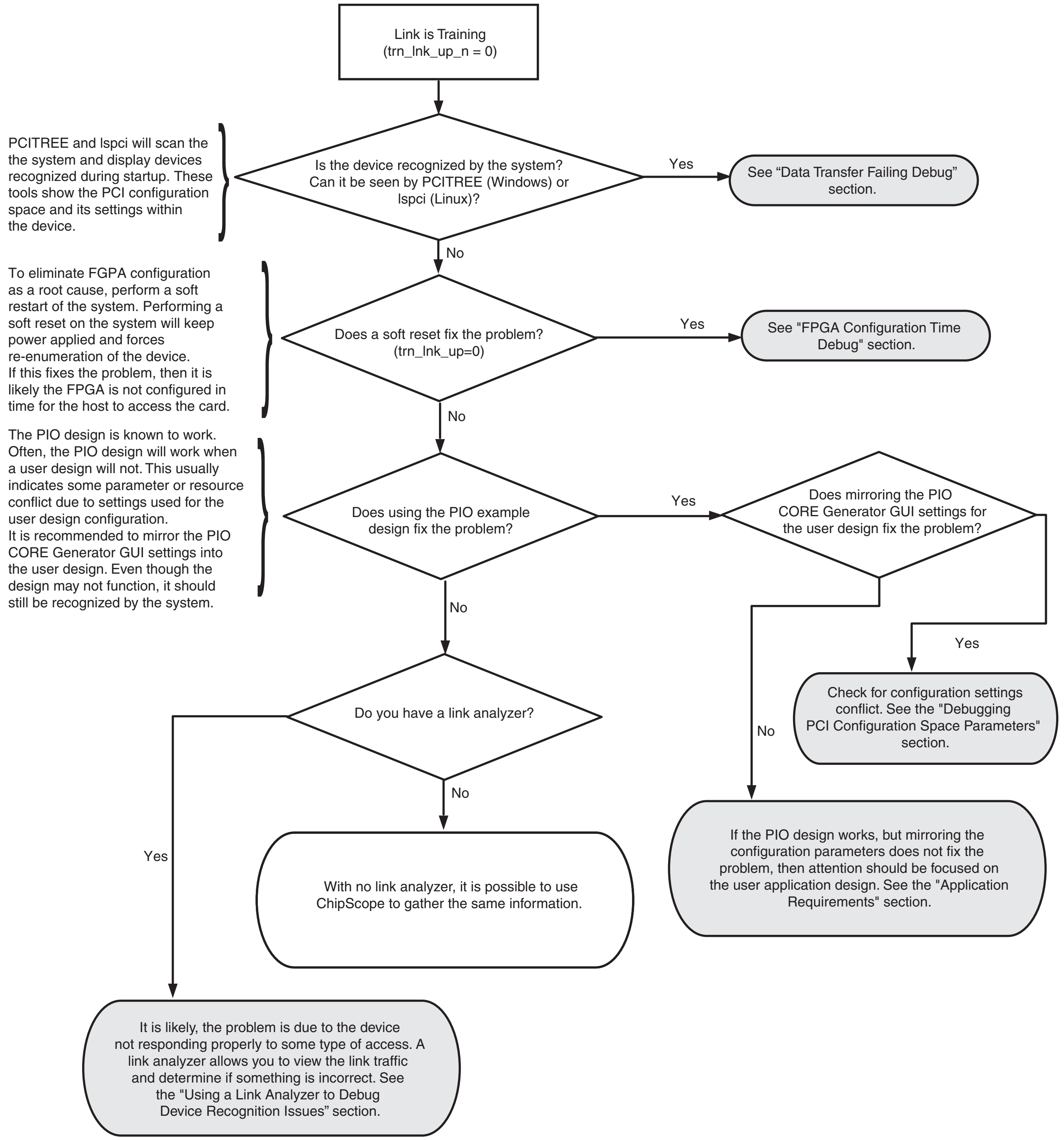

Link is Training
(trn_lnk_up_n = 0)
PCITREE and lspci will scan the the system and display devices recognized during startup. These tools show the PCI configuration space and its settings within the device.
Is the device recognized by the system? Can it be seen by PCITREE (Windows) or lspci (Linux)?
Yes
See "Data Transfer Failing Debug" section.
No
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.
Does a soft reset fix the problem? (trn_lnk_up=0)
Yes
See "FPGA Configuration Time Debug" section.
No
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.
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?
No
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.
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 13-5: **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, v2.0* states two rules that may 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 there is a finite time in which the FPGA must be configured by, and not meeting these requirements could cause problems with link training and device recognition.

Configuration may be accomplished using an on-board 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 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 the device links up and is recognized after a soft reset is performed, then FPGA configuration is most likely the problem. Note that 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 PCI Express solutions 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. IO BAR's are especially limited so configuring a large IO BAR will typically prevent the chipset from configuring the device. Generate a core that implements a small amount of Memory (approximately 2kB) to identify if this is the root cause.

The Class Code setting selected in the CORE Generator GUI can also affect configuration. The Class Code informs the Chipset as to what type of device the endpoint is. Chipsets may expect a certain type of device to be plugged into the PCI Express slot and configuration may fail if it reads an unexpected Class Code. The BIOS may be configurable to workaround 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 GUI. Trial and error may 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 IO 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.

Chipscope 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 Chipscope are defined in Table 13-4, page 186.

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 may 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

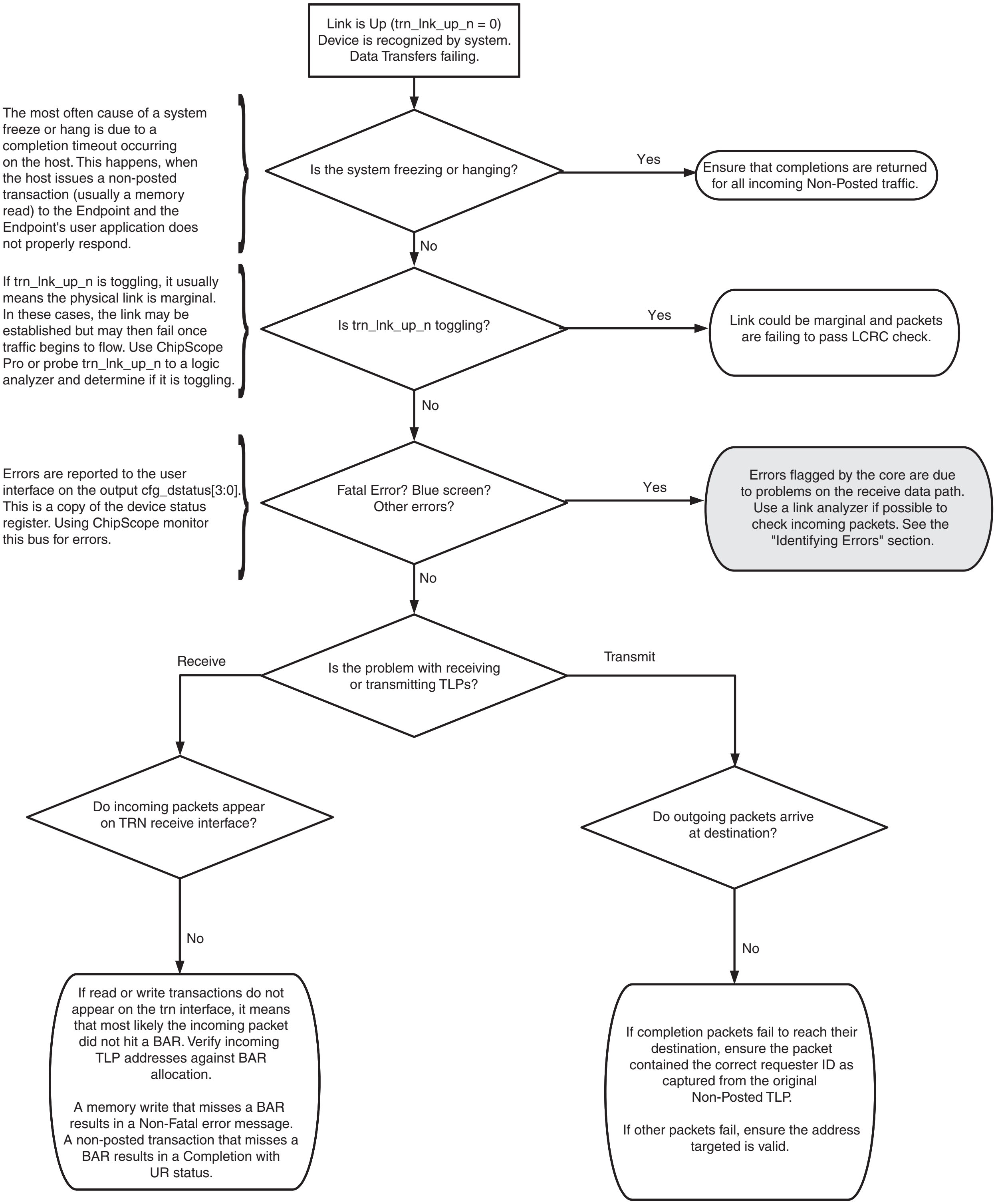

Link is Up (trn_lnk_up_n = 0)
Device is recognized by system.
Data Transfers failing.
The most often cause of a system freeze or hang is due to a completion timeout occurring on the host. This happens, when the host issues a non-posted transaction (usually a memory read) to the Endpoint and the Endpoint's user application does not properly respond.
Is the system freezing or hanging?
Yes
Ensure that completions are returned for all incoming Non-Posted traffic.
No
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.
Is trn_lnk_up_n toggling?
Yes
Link could be marginal and packets are failing to pass LCRC check.
No
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.
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
Receive
Is the problem with receiving or transmitting TLPs?
Transmit
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.
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 13-6: **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 v2.0* 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 Chipscope, 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 below A Root Complex may 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 v2.0* 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 will not report errors created by traffic on the transmit channel.

The signals shown in Table 13-4 should be monitored on the Transmit interface to verify all traffic being initiated is correct.

Table 13-4: **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: • 0 = packet data on all of trn_td[63:0] • 1 = packet data only on trn_td[63:32]

Table 13-4: **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]. 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 may occur if a TLP contains an address which 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]`. Debug ports are available to help users determine the exact cause of errors on the endpoint's receiver. See "Debug Ports," page 176 for information on these ports.

Table 13-5: **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. It is important to note that error messages will only be sent when error reporting is enabled in the Device Control register.

A fatal condition will be reported if any of the following 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 PCI Express solutions 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.

Debug ports are available to help users determine the exact cause of errors on the endpoint's receiver. See "Debug Ports," page 176 for information on these ports.

Non-Fatal Errors

Below are lists of conditions that are reported as Non-Fatal errors. See the *PCI Express Base Specification* 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 173, 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 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 173 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, ChipScope 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 VCD captures taken in the steps above.

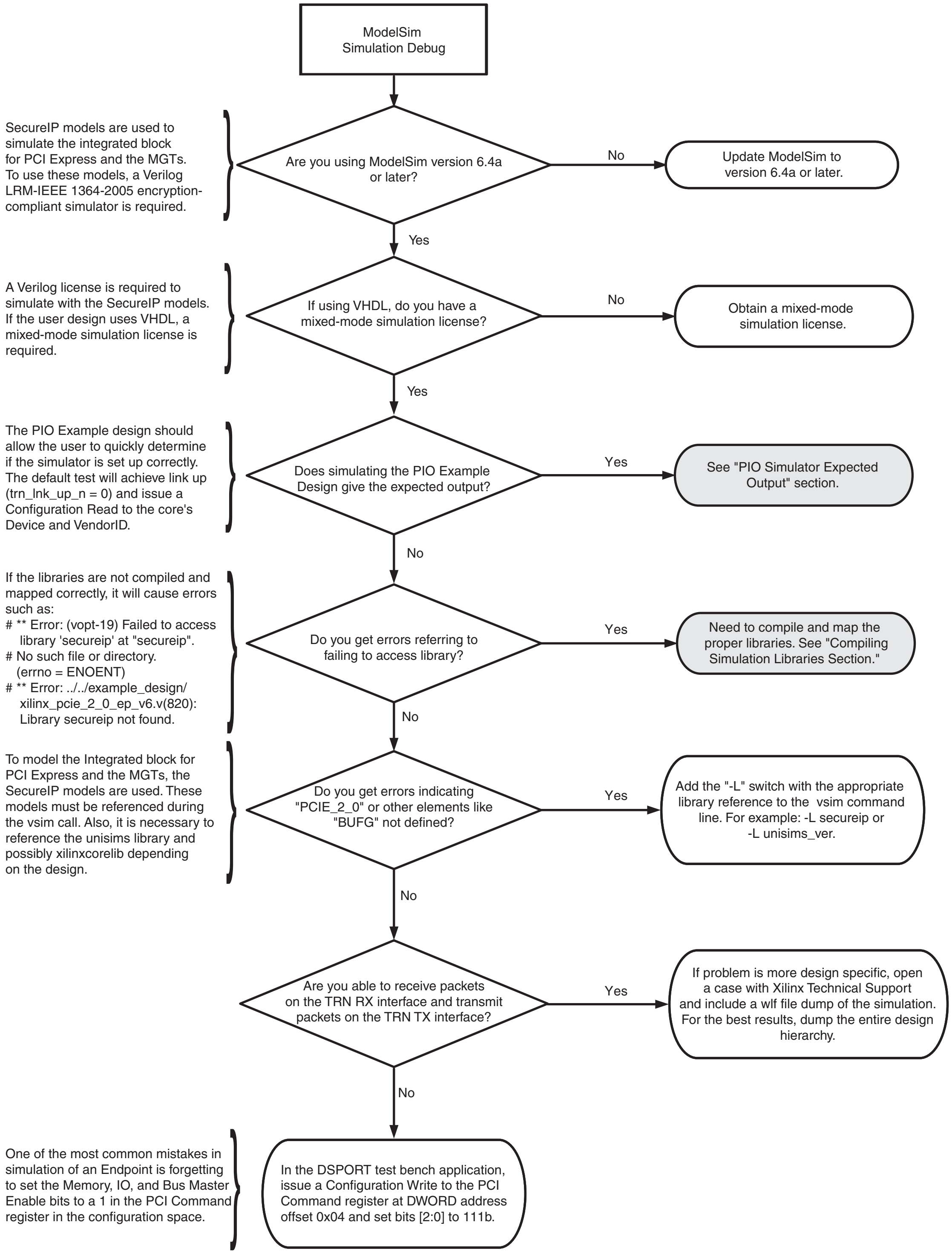
To discuss possible solutions, use the Xilinx User Community:

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 the figure.

ModelSim Debug


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.

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.

Figure 13-7: **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 the "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 spartan6 -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

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/

Appendix A

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 section 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 A-1 defines the completion space reserved in the receive buffer by the core. The values differ for different versions of the core, and also differ based on whether the designer chooses to have TLP Digests (ECRC) removed from the incoming packet stream. Values are credits, expressed in decimal.

Table A-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	8	64	16	128
256	16	128	32	256
512	32	256	32	256

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 A-2. If the User Application chooses not to read the Max_Request_Size value, it must use the default value of 128 bytes.

Table A-2: **Max Request Size Settings**

cfg_dcommand[14:12]	Max_Request_Size			
	Bytes	DW	QW	Credits
000b	128	32	16	8

Table A-2: **Max Request Size Settings**

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 A-3. If the User Application chooses not to read the RCB value, it must use the default value of 64 bytes.

Table A-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 may require; this is the same as the number of Completion Header credits required.

Methods of Managing Completion Space

A User Application may choose one of four methods to manage receive-buffer Completion space, as listed in Table B-3. For convenience, this discussion refers to these methods as LIMIT_FC, PACKET_FC, RCB_FC and DATA_FC. Each has advantages and disadvantages that the designer needs to consider when developing the User Application.

Table A-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

Table A-4: **Managing Receive Completion Space Methods**

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	Most complex user logic

The 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.

The 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 the following steps:

1. When the User Application needs to send an NP request determine the potential number of CplH and CplD credits it may 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 ≤ Total_CplH (from Table B-1)

   CPLD_PENDING + NP_CplD ≤ Total_CplD (from Table B-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 de-allocation granularity at the expense of more logic.

The 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 may 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 may cross more than one RCB. The number of RCB crossings may be calculated by:

   RCB_CROSSED = ceiling[((Lower_Address mod RCB) + Length) / RCB]

   Lower_Address and Length are fields that may 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.

The 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 may 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 ≤ 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 may be calculated by:

   RCB_CROSSED = ceiling[((Lower_Address mod RCB) + Length) / RCB]

   Lower_Address and Length are fields that may 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 mod 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.

PCIE_A1 Port Descriptions

This chapter describes the physical interfaces visible on the Spartan®-6 FPGA Integrated Endpoint Block's software primitive, `PCIE_A1`.

This chapter contains the following sections:

- "Clock and Reset Interface"
- "Transaction Layer Interface"
- "Block RAM Interface"
- "GTP Transceiver Interface"
- "Configuration Management Interface"
- "Debug Interface Ports"

Clock and Reset Interface

Table B-1 defines the ports in the Clock and Reset interface.

Table B-1: **Clock and Reset Interface Port Descriptions**

Port	Direction	Clock Domain	Description
CLOCKLOCKED	Input	USERCLK	LOCKED signal from the PLL.
MGTCLK	Input	MGTCLK	PIPE interface clock.
RECEIVEDHOTRST	Output	MGTCLK	Received hot reset. When asserted, this output indicates when an in-band hot reset has been received.
SYSRESETN	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.
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 B-2 defines the ports in the Transaction Layer interface.

Table B-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.
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: `000b`: Receive buffer available space `001b`: Receive credits granted to the link partner `010b`: Receive credits consumed `100b`: Transmit user credits available `101b`: Transmit credit limit `110b`: Transmit credits consumed
TRNLNKUPN	Output	USERCLK	Transaction Link Up (active Low). This output is asserted when the core and the connected upstream link partner port are ready and able to exchange data packets. It 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 the Hot Reset and Link Disable states by the link partner, TRNLNKUPN is deasserted and all TLPs stored in the core are lost.

Table B-2: **Transaction Layer Interface Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
TRNRBARHITN[6:0]	Output	USERCLK	Receive BAR Hit (active Low). This output indicates the BAR(s) targeted by the current receive transaction: TRNRBARHITN[0]: BAR0 TRNRBARHITN[1]: BAR1 TRNRBARHITN[2]: BAR2 TRNRBARHITN[3]: BAR3 TRNRBARHITN[4]: BAR4 TRNRBARHITN[5]: BAR5 TRNRBARHITN[6]: Expansion ROM Address If two BARs are configured into a single 64-bit address, both corresponding TRNRBARHITN bits are asserted.
TRNRD[31:0]	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.
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.
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.

Table B-2: **Transaction Layer Interface Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
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[31:0]	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.
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 only include packets longer than the supported maximum payload size and do not include 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. If TRNTSTRN is deasserted, TRNTERRFWDN can be asserted any time between start of frame (SOF) and end of frame (EOF), inclusive. If TRNTSTRN is asserted, TRNTERRFWDN can only be asserted at SOF.
TRNTSOFN	Input	USERCLK	Transmit Start-of-Frame (active Low). When asserted, this input indicates the start of a 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 B-3 defines the TX buffer and RX buffer ports for the Block RAM interface.

Table B-3: **Block RAM Interface Port Descriptions**

Port	Direction	Clock Domain	Description
MIMRXRADDR[11:0]	Output	USERCLK	RX buffer read address
MIMRXRDATA[34:0]	Input	USERCLK	RX buffer read data
MIMRXREN	Output	USERCLK	RX buffer read enable
MIMRXWADDR[11:0]	Output	USERCLK	RX buffer write address
MIMRXWDATA[34:0]	Output	USERCLK	RX buffer write data
MIMRXWEN	Output	USERCLK	RX buffer write enable
MIMTXRADDR[11:0]	Output	USERCLK	TX buffer read address
MIMTXRDATA[35:0]	Input	USERCLK	TX buffer read data
MIMTXREN	Output	USERCLK	TX buffer read enable
MIMTXWADDR[11:0]	Output	USERCLK	TX buffer write address
MIMTXWDATA[35:0]	Output	USERCLK	TX buffer write data
MIMTXWEN	Output	USERCLK	TX buffer write enable

GTP Transceiver Interface

Table B-4 defines the PIPE per Lane ports within the GTP Transceiver interface. There are two copies of the PIPE per Lane ports, one for each port (*n* = A or B). Depending on which GTP transceiver is used, the LogiCORE IP core for PCI Express selects the correct port to use for the design.

Table B-4: **PIPE per Lane Port Descriptions for the GTP Transceiver Interface**

Port	Direction	Clock Domain	Description
PIPEGTRESETDONE*n*	Input	MGTCLK	When asserted, this input indicates that the GTP transceiver has finished reset and is ready for use.
PIPEPHYSTATUS*n*	Input	MGTCLK	PIPEPHYSTATUS*n* is asserted for a single cycle to indicate completion of GTP transceiver functions such as Power Management state transitions and receiver detection on lane *n*.
PIPERXCHARISK*n*[1:0]	Input	MGTCLK	This output defines the control bit(s) for received data: `0b`: Data byte `1b`: Control byte The lower bit corresponds to the lower byte of PIPERXDATA*n*[7:0] while the upper bit describes of PIPERXDATA*n*[15:8].
PIPERXDATA*n*[15:0]	Input	MGTCLK	This input contains the received data.
PIPERXENTERELECIDLE*n*	Input	MGTCLK	This input indicates an electrical idle on the Receiver.

Table B-4: **PIPE per Lane Port Descriptions for the GTP Transceiver Interface *(Cont'd)***

Port	Direction	Clock Domain	Description
PIPERXPOLARITY*n*	Output	MGTCLK	When High, this output instructs the GTP transceiver to invert polarity (on the RX differential pair).
PIPERXRESET*n*	Output	MGTCLK	When asserted, this output resets the receive portion of the GTP transceiver.
PIPERXSTATUS*n*[2:0]	Input	MGTCLK	This input encodes the receiver status and error codes for the received data stream and receiver detection on lane *n*: `000b`: Data received OK `001b`: Reserved `010b`: Reserved `011b`: Receiver Detected `100b`: 8B/10B decode error `101b`: Elastic Buffer overflow `110b`: Elastic Buffer underflow `111b`: Receive disparity error
PIPETXCHARDISPMODE*n*[1:0]	Output	MGTCLK	PIPETXCHARDISPMODE and PIPETXCHARDISPVAL allow the 8B/10B disparity of outgoing data to be controlled when 8B/10B encoding is enabled. PIPETXCHARDISPMODE[1] corresponds to TXDATA[15:8] and PIPETXCHARDISPMODE[0] corresponds to PIPETXDATA[7:0]. For PCI Express operation, PIPETXCHARDISPMODE maps to the PIPE signal TXCOMPLIANCE given that PIPETXCHARDISPVAL is Low. When PIPETXCHARDISPMODE is High and PIPETXCHARDISPVAL is Low, the running disparity is set to negative. This functionality is used when transmitting the compliance pattern.
PIPETXCHARDISPVAL*n*[1:0]	Output	MGTCLK	PIPETXCHARDISPMODE and PIPETXCHARDISPVAL allow the 8B/10B disparity of outgoing data to be controlled when 8B/10B encoding is enabled. TXCHARDISPVAL[1] corresponds to TXDATA[15:8] and TXCHARDISPVAL[0] corresponds to TXDATA[7:0]. For PCI Express operation, PIPETXCHARDISPVAL should always be Low.
PIPETXCHARISK*n*[1:0]	Output	MGTCLK	This output determines the control bit(s) for received data: `0b`: Data byte `1b`: Control byte The lower bit corresponds to the lower byte of PIPETXDATA*n*[7:0] while the upper bit describes PIPETXDATA*n*[15:8].

Table B-4: **PIPE per Lane Port Descriptions for the GTP Transceiver Interface *(Cont'd)***

Port	Direction	Clock Domain	Description
PIPETXDATA*n*[15:0]	Output	MGTCLK	This output contains the transmit data.
PIPETXELECIDLE*n*	Output	MGTCLK	This output forces the transmit output to electrical idle in all power states.
PIPETXPOWERDOWN*n*[1:0]	Output	MGTCLK	This output is the Power Management signal for the transmitter for lane *n*: `00b`: P0 (Normal operation) `01b`: P0s (Low recovery time power-saving state) `10b`: P1 (Longer recovery time power state) `11b`: Reserved
PIPETXRCVRDET*n*	Output	MGTCLK	When asserted, this output enables the GTP transceiver to begin either a receiver detection operation or loopback.

Configuration Management Interface

The Configuration Management Interface contains the following signal groupings:

- "Management Interface Ports"
- "Error Reporting Ports"
- "Interrupt Generation and Status Ports"
- "Power Management Ports"
- "Configuration Specific Register Ports"
- "Miscellaneous Configuration Management Ports"

Management Interface Ports

Table B-5 defines the Management Interface ports within the Configuration Management interface. These ports are used when reading and writing the Configuration Space Registers.

Table B-5: **Management Interface Port Descriptions**

Port	Direction	Clock Domain	Description
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.
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. The integrated block does not support write operations.

Error Reporting Ports

Table B-6 defines the Error Reporting ports within the Configuration Management interface.

Table B-6: **Error Reporting Port Descriptions**

Port	Direction	Clock Domain	Description
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 Completion Ready (active low). When asserted, this signal indicates that the core can accept assertions on CFGERRURN and CFGERRCPLABORTN for Non-Posted Transactions. Assertions on CFGERRURN and CFGERRCPLABORTN are ignored when CFGERRCPLRDYN is deasserted.
CFGERRCPLTIMEOUTN	Input	USERCLK	Configuration Error Completion Time-out (active Low). The user asserts this signal to report a completion timed out.
CFGERRECRCN	Input	USERCLK	ECRC Error Report (active Low). The user asserts this signal to report an end-to-end CRC (ECRC) error.
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.

Table B-6: **Error Reporting Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
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. The following information should be extracted from the received error TLP and presented in the listed format: [47:41] Lower Address [40:29] Byte Count [28:26] TC [25:24] Attr [23:8] Requester ID [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 B-7 defines the Interrupt Generation and Status ports within the Configuration Management interface.

Table B-7: **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

Table B-7: **Interrupt Generation and Status Port Descriptions** ***(Cont'd)***

Port	Direction	Clock Domain	Description
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].
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. `0`: Only Legacy (INTx) interrupts can be sent `1`: The Message Signaling Interrupt (MSI) messaging is enabled
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.

Power Management Ports

Table B-8 defines the Power Management ports within the Configuration Management interface.

Table B-8: **Power Management Port Descriptions**

Port	Direction	Clock Domain	Description
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.
CFGTOTURNOFFN	Output	USERCLK	Configuration To Turnoff: This output signal notifies the user that a PME_TURN_Off message has been received, and the Configuration and Capabilities Module (CCM) starts polling the CFGTURNOFFOKN input coming in from the user. When CFGTURNOFFOKN is asserted, the CMM sends a PME_To_Ack message to the upstream device.
CFGTURNOFFOKN	Input	USERCLK	Configuration Turnoff OK (active low). This input is the power turn-off ready signal. The user application can assert this input to notify the Endpoint that it is safe for power to be turned off.

Configuration Specific Register Ports

Table B-9 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 B-9: **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. When this output is asserted, the user logic is allowed to issue Memory or I/O Requests (including MSI interrupts); otherwise, the user logic must not issue those requests.
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]. `0`: The integrated block filters these accesses and responds with a UR. `1`: Allows the device to receive I/O Space accesses.
CFGCOMMANDMEMENABLE	Output	USERCLK	Configuration Command, Memory Space Enable, Command[1]. `0`: The integrated block filters these accesses and responds with a UR. `1`: Allows the device to receive Memory Space accesses.
CFGCOMMANDSERREN	Output	USERCLK	Configuration Command, SERR Enable (active Low), Command[8]. 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.

Table B-9: **Configuration Specific Register Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
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.
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.
CFGDEVCONTROLFATALERRREPORTINGEN	Output	USERCLK	Configuration Device Control, Fatal Error Reporting Enable, DEVICECTRL[2]. This bit, in conjunction with other bits, controls sending ERRFATAL messages.
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. `000b`: 128-byte maximum payload size `001b`: 256-byte maximum payload size `010b`: 512-byte maximum payload size

Table B-9: **Configuration Specific Register Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
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. `000b`: 128-byte maximum Read Request size `001b`: 256-byte maximum Read Request size `010b`: 512-byte maximum Read Request size
CFGDEVCONTROLNONFATALREPORTINGEN	Output	USERCLK	Configuration Device Control, Non-Fatal Error Reporting Enable, DEVICECTRL[1]. This bit, in conjunction with other bits, controls sending ERRNONFATAL messages.
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.

Table B-9: **Configuration Specific Register Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
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.
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: `00b`: Disabled `01b`: L0s Entry Enabled `10b`: Not used `11b`: Not used

Table B-9: **Configuration Specific Register Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
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.
CFGLINKCONTROLRCB	Output	USERCLK	Configuration Link Control, RCB, LINKCTRL[3]. This output indicates the Read Completion Boundary value, where: `0`: 64B `1`: 128B
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 B-10 defines the Miscellaneous ports within the Configuration Management interface.

Table B-10: **Miscellaneous Configuration Management Port Descriptions**

Port	Direction	Clock Domain	Description
CFGBUSNUMBER[7:0]	Output	USERCLK	Configuration Bus Number. This 8-bit output provides the assigned bus number for the device. The user application must use this information in the Bus Number field of outgoing TLP requests. The default value after reset is `00h`. This output is refreshed whenever a Type 0 Configuration Write packet is received.
CFGDEVICENUMBER[4:0]	Output	USERCLK	Configuration Device Number: This 5-bit output provides the assigned device number for the device. The user application must use this information in the Device Number field of outgoing TLP requests. The default value after reset is `00000b`. This output is refreshed whenever a Type 0 Configuration Write packet is received.
CFGDEVID[15:0]	Input	USERCLK	Device ID value. This 16-bit input must be stable when SYSRESETN is deasserted.
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.
CFGFUNCTIONNUMBER[2:0]	Output	USERCLK	Configuration Function Number. This 3-bit output provides the function number for the device. The user application must use this information in the Function Number field of outgoing TLP requests. The function number is hardwired to `000b`.

Table B-10: **Miscellaneous Configuration Management Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
CFGLTSSMSTATE[4:0]	Output	MGTCLK	This 5-bit output is a mirror of the LTSSM state machine bits: `00000b`: Detect.Quiet `00001b`: Detect.Active `00010b`: Polling.Active `00011b`: Polling.Config `00100b`: Polling Compliance `00101b`: Configuration.Linkwidth.Start `00110b`: Configuration.Linkwidth.Start `00111b`: Configuration.Linkwidth.Accept `01000b`: Configuration.Linkwidth.Accept `01001b`: Configuration.Lanenum.Wait `01010b`: Configuration.Lanenum.Accept `01011b`: Configuration.Complete `01100b`: Configuration.Idle `01101b`: L0 `01110b`: L1.Entry `01111b`: L1.Entry `10000b`: L1.Entry `10001b`: L1.Idle `10010b`: L1.Exit-to-recovery `10011b`: Recovery.RcvrLock `10100b`: Recovery.RcvrCfg `10101b`: Recovery.Idle `10110b`: Hot Reset `10111b`: Disabled `11000b`: Disabled `11001b`: Disabled `11010b`: Disabled `11011b`: Detect.Quiet
CFGPCIELINKSTATEN[2:0]	Output	USERCLK	PCI Express Link State. This encoded bus reports the PCI Express Link State Information to the user: `110b`: L0 state `101b`: L0s state `011b`: L1 state `111b`: Under transition
CFGREVID[7:0]	Input	USERCLK	Revision ID Value. This input must be stable when SYSRESETN is deasserted.
CFGSUBSYSID[15:0]	Input	USERCLK	Subsystem ID Value. This input must be stable when SYSRESETN is deasserted.

Table B-10: **Miscellaneous Configuration Management Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
CFGSUBSYSVENID[15:0]	Input	USERCLK	Subsystem Vendor ID Reset Value. This input must be stable when SYSRESETN is deasserted.
CFGVENID[15:0]	Input	USERCLK	Vendor ID Value. This input must be stable when SYSRESETN is deasserted.

Debug Interface Ports

Table B-11 describes the Debug Interface ports.

Table B-11: **Debug Interface Port Descriptions**

Port	Direction	Clock Domain	Description
DBGBADDLLPSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle when a DLLP CRC error is detected.
DBGBADTLPLCRC	Output	USERCLK	This signal pulses High for one USERCLK cycle when a TLP with an LCRC error is detected.
DBGBADTLPSEQNUM	Output	USERCLK	This signal pulses High for one USERCLK cycle when a TLP with an invalid sequence number is detected.
DBGBADTLPSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle when a bad TLP is detected, for reasons other than a bad LCRC or a bad sequence number.
DBGDLPROTOCOLSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle if an out-of-range ACK or NAK is received.
DBGFCPROTOCOLERRSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle if there is a protocol error with the received flow control updates.
DBGMLFRMDLENGTH	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that a received TLP had a length that did not match what was in the TLP header.
DBGMLFRMDMPS	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that a received TLP had a length in violation of the negotiated MPS.
DBGMLFRMDTCVC	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that a received TLP had an invalid TC or VC value.
DBGMLFRMDTLPSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle when a malformed TLP is received. See the other DBGMLFRMD* signals for further clarification. **Note:** There is skew between DBGMLFRMD* and DBGMLFRMDTLPSTATUS.

Table B-11: **Debug Interface Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description
DBGMLFRMDUNRECTYPE	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that a received TLP had an invalid/unrecognized type field value.
DBGPOISTLPSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle if a poisoned TLP is received.
DBGRCVROVERFLOWSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle if a received TLP violates the advertised credit.
DBGREGDETECTEDCORRECTABLE	Output	USERCLK	This signal is a mirror of the internal signal used to indicate a correctable error is detected. The error is cleared upon a read by the Root Complex (RC).
DBGREGDETECTEDFATAL	Output	USERCLK	This signal is a mirror of the internal signal used to indicate that a fatal error has been detected. The error is cleared upon a read by the RC.
DBGREGDETECTEDNONFATAL	Output	USERCLK	This signal is a mirror of the internal signal used to indicate that a non-fatal error has been detected. The error is cleared upon a read by the RC.
DBGREGDETECTEDUNSUPPORTED	Output	USERCLK	This signal is a mirror of the internal signal used to indicate that an unsupported request has been detected. The error is cleared upon a read by the RC.
DBGRPLYROLLOVERSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle when the rollover counter expires.
DBGRPLYTIMEOUTSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle when the replay time-out counter expires.
DBGURNOBARHIT	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that a received read or write request did not match any configured BAR.
DBGURPOISCFGWR	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that a CfgWr TLP with the Error/Poisoned bit (EP) = 1 was received.
DBGURSTATUS	Output	USERCLK	This signal pulses High for one USERCLK cycle when an unsupported request is received. See the DBGUR* signals for further clarification. **Note:** There is skew between DBGUR* and DBGURSTATUS.
DBGURUNSUPMSG	Output	USERCLK	This signal pulses High for one USERCLK cycle to indicate that an Msg or MsgD TLP with an unsupported type was received.

Appendix C

PCIE_A1 Attribute Descriptions

Table C-1 defines the attributes on the `PCIE_A1` library primitive for the Spartan®-6 FPGA Integrated Endpoint Block for PCI Express® designs. All attributes are set in the LogiCORE™ IP; they are documented in this chapter for reference. Users should not change the attribute settings as set in the CORE Generator GUI for proper operation of the design.

Table C-1: **PCIE_A1 Attributes**

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: • Memory Space BAR: 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 {BAR1, BAR0} 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
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: • Memory Space BAR (not the upper bits of BAR0): 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 {BAR2, BAR1} 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 C-1: **PCIE_A1 Attributes *(Cont'd)***

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 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
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. For an Endpoint, bits are defined as follows: • Memory Space BAR (not upper bits of BAR2): 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 {BAR4, BAR3} 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 C-1: **PCIE_A1 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. For an Endpoint, bits are defined as follows: • Memory Space BAR (not upper bits of BAR3): 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 {BAR5, BAR4} 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
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. For an Endpoint, bits are defined as follows: • Memory Space BAR (not upper bits of BAR4): 0: Mem Space Indicator (set to `0`) [2:1]: Type field (`00` for 32-bit; BAR5 cannot be the lower part of a 64-bit BAR) 3: Prefetchable (`0` or `1`) [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 • 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
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.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
CLASS_CODE	24-bit Hex	Code identifying basic function, subclass, and applicable programming interface. This value is transferred to the Class Code Register.
DEV_CAP_ENDPOINT_L0S_LATENCY	3-bit Binary	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: `000b`: Less than 64 ns `001b`: 64 ns to 128 ns `010b`: 128 ns to 256 ns `011b`: 256 ns to 512 ns `100b`: 512 ns to 1 µs `101b`: 1 µs to 2 µs `110b`: 2 µs to 4 µs `111b`: More than 4 µs
DEV_CAP_ENDPOINT_L1_LATENCY	3-bit Binary	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: `000b`: Less than 1 µs `001b`: 1 µs to 2 µs `010b`: 2 µs to 4 µs `011b`: 4 µs to 8 µs `100b`: 8 µs to 16 µs `101b`: 16 µs to 32 µs `110b`: 32 µs to 64 µs `111b`: More than 64 µs
DEV_CAP_EXT_TAG_SUPPORTED	Boolean	Extended Tags support. FALSE: 5-bit tag TRUE: 8-bit tag
DEV_CAP_MAX_PAYLOAD_SUPPORTED	3-bit Binary	This attribute specifies the maximum payload supported. Valid (supported) settings are: `000b`: 128 bytes `001b`: 256 bytes `010b`: 512 bytes This value is transferred to the Device Capabilities Register.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
DEV_CAP_PHANTOM_FUNCTIONS_SUPPORT	2-bit Binary	Phantom Function Support. This attribute indicates the number of functions re-allocated as Tag bits. Valid settings are: `00b`: 0 `01b`: 1 `10b`: 2 `11b`: 3
DEV_CAP_ROLE_BASED_ERROR	Boolean	When this attribute is set to TRUE, compliant error reporting is supported.
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_SCRAMBLING	Boolean	When this attribute is TRUE, Scrambling of transmit data is turned 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.
EXPANSION_ROM	22-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 `22'h00000000`. Bits are defined as follows: 0: Expansion ROM implemented (set to `1` to implement ROM) [21:1]: Mask for writable bits of BAR. The uppermost 21:n bits are set to `1`, where 2^n = ROM aperture size in bytes
FAST_TRAIN	Boolean	When this attribute is set to TRUE, the timers in the LTSSM state machine are shortened to reduce simulation time. Specifically, the transition out of Polling.Active requires 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). This attribute must be set to FALSE for silicon designs.
GTP_SEL	Boolean	This attribute indicates which port interface is used: FALSE: Transceiver A port interface TRUE: Transceiver B port interface

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
LINK_CAP_ASPM_SUPPORT	2-bit Binary	Active State PM Support. This attribute indicates the level of active state power management supported by the selected PCI Express Link: `00b`: Reserved `01b`: L0s entry supported `10b`: Reserved `11b`: Reserved
LINK_CAP_L0S_EXIT_LATENCY	3-bit Binary	This attribute sets the exit latency from the L0s state to be applied (at 2.5 GT/s) where a common clock is used. This value is transferred to the Link Capabilities Register. Valid settings are: `000b`: Less than 64 ns `001b`: 64 ns to less than 128 ns `010b`: 128 ns to less than 256 ns `011b`: 256 ns to less than 512 ns `100b`: 512 ns to less than 1 µs `101b`: 1 µs to less than 2 µs `110b`: 2 µs to 4 µs `111b`: More than 4 µs
LINK_CAP_L1_EXIT_LATENCY	3-bit Binary	This attribute sets the exit latency from the L1 state to be applied (at 2.5 GT/s) where a common clock is used. This value is transferred to the Link Capabilities Register. Valid settings are: `000b`: Less than 1 µs `001b`: 1 µs to less than 2 µs `010b`: 2 µs to less than 4 µs `011b`: 4 µs to less than 8 µs `100b`: 8 µs to less than 16 µs `101b`: 16 µs to less than 32 µs `110b`: 32 µs to 64 µs `111b`: More than 64 µs
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.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
LL_ACK_TIMEOUT	15-bit Hex	This attribute sets an ACK time-out counter override value. The value is in increments of USERCLK periods. It should be set to 0 unless the user wishes to override the default (internal) setting.
LL_ACK_TIMEOUT_EN	Boolean	When set to TRUE, the value specified by LL_ACK_TIMEOUT is added to the internal value, increasing the ACK Timeout delay. When set to FALSE, the value provided on LL_ACK_TIMEOUT is subtracted from the internal value, decreasing the ACK Timeout delay
LL_REPLAY_TIMEOUT	15-bit Hex	This attribute sets a replay timer override value. The value is in increments of USERCLK periods. It should be set to 0 unless the user wishes to override the default (internal) setting.
LL_REPLAY_TIMEOUT_EN	Boolean	When set to TRUE, the value specified by LL_REPLAY_TIMEOUT is added to the internal value, increasing the Replay Timeout delay. When set to FALSE, the value provided on LL_REPLAY_TIMEOUT is subtracted from the internal value, decreasing the Replay Timeout delay
MSI_CAP_MULTIMSG_EXTENSION	Boolean	Multiple Message Capable Extension. When set to TRUE, this attribute allows 256 unique messages to be sent by the user regardless of the setting of MSI_CAP_MULTIMSGCAP). **Note:** Enabling this feature (TRUE) violates the PCI Express Base Specification and should only be used in closed systems.
MSI_CAP_MULTIMSGCAP	3-bit Binary	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: `000b`: 1 vector `001b`: 2 vectors `010b`: 4 vectors `011b`: 8 vectors `100b`: 16 vectors `101b`: 32 vectors `110b` - `111b`: Reserved
PCIE_CAP_CAPABILITY_VERSION	4-bit Hex	This attribute indicates the version number of the PCI-SIG defined PCI Express capability structure. It must be set to `0001b`.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
PCIE_CAP_DEVICE_PORT_TYPE	4-bit Hex	This attribute identifies the type of device/port. Valid settings are (all other values are unsupported): `0000b`: PCI Express Endpoint device `0001b`: Legacy PCI Express Endpoint device This value is transferred to the PCI Express Capabilities Register.
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_SLOT_IMPLEMENTED	Boolean	This attribute must be set to FALSE.
PCIE_GENERIC	12-bit Hex	The 12 bits are assigned as follows: 11: This bit must be 0. 10: ♦ `0`: Electrical idle is not received until an Electrical Idle Ordered Set (EIOS) is received, if no EIOS core enters the LTSSM RECOVERY state ♦ `1`: An electrical idle can occur without an EIOS (the EIOS is assumed). This is the default and recommended setting. [9:7]: These bits drive the Interrupt Pin Register in the PCI Configuration Space. A value of 0 indicates no Legacy interrupts are implemented. Values of 1, 2, 3, and 4 indicate INTA, INTB, INTC, and INTD, respectively. Other values are not permitted. 6: ♦ `0`: The DSN Extended Capability is not implemented ♦ `1`: The DSN Extended Capability is implemented 5: ♦ `0`: 8B/10B Not_in_table is not inferred ♦ `1`: 8B/10B Not_in_table from the GTP transceiver is inferred from RXSTATUS. This is the default and recommended setting. 4: ♦ `0`: A read to an unimplemented config space returns completion with data of zero. This is the default and recommended setting. ♦ `1`: A read to an unimplemented config space returns a UR (legacy behavior of PIPE) [3:0]: These bits drive nFTS[7:4]. The lower bits of nFTS are set to Fh. The default value is 0xF.
PLM_AUTO_CONFIG	Boolean	This attribute must be set to FALSE.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
PM_CAP_AUXCURRENT	3-bit Binary	AUX Current. Requested auxiliary current allocation. This value is transferred to the PM Capabilities Register, bits [24:22]. The integrated block does not support AUX power, so this field should be set to `000b`.
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_PME_CLOCK	Boolean	When this attribute is set to TRUE, a PCI™ clock is required for PME generation. This attribute must be set to FALSE per the specification. The value is transferred to the PM Capabilities Register, bit 19.
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_VERSION	3-bit Binary	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_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
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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `11b`: 0.001x

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `11b`: 0.001x
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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `11b`: 0.001x

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `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: `00b`: 1.0x `01b`: 0.1x `10b`: 0.01x `11b`: 0.001x
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.
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.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
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.
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. 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. 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. This attribute must be set to FALSE for Endpoints.
TL_RX_RAM_RADDR_LATENCY	Boolean	This attribute specifies the read address latency for RX RAMs in terms of USER_CLK cycles. FALSE: No fabric pipeline register is on the read address and enable block RAM inputs TRUE: A fabric pipeline register is on the read address and enable block RAM inputs
TL_RX_RAM_RDATA_LATENCY	2-bit Binary	This attribute specifies the read data latency for RX RAMs in terms of USER_CLK cycles. `01b`: The block RAM output register is disabled `10b`: The block RAM output register is enabled `11b`: The block RAM output register is enabled and a fabric pipeline register is added to the block RAM data output

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
TL_RX_RAM_WRITE_LATENCY	Boolean	This attribute specifies the write latency for RX RAMs in terms of cycles of USER_CLK. FALSE: No fabric pipeline register is on the write address and enable block RAM inputs TRUE: A fabric pipeline register is on the write address and enable block RAM inputs
TL_TFC_DISABLE	Boolean	When this attribute is set to TRUE, checking of flow control values and transmit packets in the order they were presented on the TRN TX interface is disabled.
TL_TX_CHECKS_DISABLE	Boolean	When this attribute is set to TRUE, all TLM checks of incoming data are disabled.
TL_TX_RAM_RADDR_LATENCY	Boolean	This attribute specifies the read address latency for TX RAMs in terms of USER_CLK cycles. FALSE: No fabric pipeline register on the read address and enable block RAM inputs TRUE: A fabric pipeline register is on the read address and enable block RAM inputs
TL_TX_RAM_RDATA_LATENCY	2-bit Binary	This attribute specifies the read data latency for TX RAMs in terms of USER_CLK cycles. `01b`: The block RAM output register is disabled `01b`: The block RAM output register is enabled `11b`: The block RAM output register is enabled and a fabric pipeline register is added to the block RAM data output
USR_CFG	Boolean	When this attribute is set to TRUE, the user application is permitted to add or implement PCI Legacy capability registers beyond address `BFh`. This option should be selected when the user application implements such a legacy capability configuration space, starting at `C0h`.
USR_EXT_CFG	Boolean	When this attribute is set to TRUE, the user application is permitted to add or implement PCI Express extended capability registers beyond address `1FFh`. This box should be checked when the user application implements such an extended capability configuration space starting at `200h`.
VC0_CPL_INFINITE	Boolean	When this attribute is set to TRUE, the block advertises infinite completions. **Note:** For Endpoints, this attribute must be set to TRUE for compliance.
VC0_RX_RAM_LIMIT	12-bit Hex	This attribute must be set to RX buffer bytes/4.

Table C-1: **PCIE_A1 Attributes *(Cont'd)***

Attribute Name	Type	Description
VC0_TOTAL_CREDITS_CD	11-bit Hex	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. The equation to calculate bytes advertised is: (ph * (rx_td_ecrc_trim ? 16 : 20)) + (pd * 16) + (nph * 20) + (ch * 16) + (cd * 16) The equation to calculate block RAM bytes available is: (vc0_rx_ram_limit + 1) * 4 See Table C-2, page 236 for valid settings.
VC0_TOTAL_CREDITS_CH	7-bit Hex	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. See Table C-2, page 236 for valid settings.
VC0_TOTAL_CREDITS_NPH	7-bit Hex	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. This attribute must be set to 8.
VC0_TOTAL_CREDITS_PD	11-bit Hex	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. The equation to calculate bytes advertised is: (ph * (rx_td_ecrc_trim ? 16 : 20)) + (pd * 16) + (nph * 20) + (ch * 16) + (cd * 16) The equation to calculate block RAM bytes available is: (vc0_rx_ram_limit + 1) * 4 See Table C-2, page 236 for valid settings.
VC0_TOTAL_CREDITS_PH	7-bit Hex	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.
VC0_TX_LASTPACKET	5-bit Hex	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. The equation is: ((TX buffer bytes) / (MPS_in_bytes + 20) - 1) See Table C-2 for valid settings.

Table C-2: **Valid Data Credit Combinations**

Parameter Name	Valid Combinations					
DEV_CAP_MAX_PAYLOAD_SUPPORTED	0	0	1	1	2	2
VC0_TOTAL_CREDITS_PD	36	92	92	204	204	716
VC0_TOTAL_CREDITS_CD	64	128	128	256	256	256
VC0_TOTAL_CREDITS_CH	8	16	16	32	32	32
VC0_TX_LAST_PACKET	12	26	13	28	14	29

Appendix D

PCIE_A1 Timing Parameter Descriptions

This chapter lists the timing parameter names and descriptions related to the Spartan®-6 FPGA Integrated Endpoint 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* [Ref 3].

The timing parameters on the integrated block consist of either Setup/Hold or Clock-to-Out parameters. Table D-1 lists the timing parameter names, descriptions, signal grouping, and related clock domain for a given parameter. In the table, parameter Tpcicck_XXX is a setup time (before the clock edge), and parameter Tpcickc_XXX is a hold time (after clock edge)

Table D-1: **PCIE_A1 Timing Parameters**

Name	Clock Domain	Signal Grouping
Sequential Setup and Hold Times for Integrated Block Inputs		
Tpcicck_CFG / Tpcickc_CFG	USERCLK	CFGDEVID[15:0]
		CFGDSN[63:0]
		CFGDWADDR[9:0]
		CFGERRCORN
		CFGERRCPLTIMEOUTN
		CFGERRECRCN
		CFGERRLOCKEDN
		CFGERRPOSTEDN
		CFGERRTLPCPLHEADER[47:0]
		CFGERRURN
		CFGINTERRUPTASSERTN
		CFGINTERRUPTDI[7:0]
		CFGINTERRUPTN
		CFGRDENN
		CFGREVID[7:0]
		CFGSUBSYSID[15:0]
		CFGSUBSYSVENID[15:0]
		CFGTRNPENDINGN
		CFGVENID[15:0]
		TRNTCFGGNTN
Tpcicck_ERR / Tpcickc_ERR	USERCLK	CFGERRCPLABORTN

Table D-1: **PCIE_A1 Timing Parameters *(Cont'd)***

Name	Clock Domain	Signal Grouping
Tpcicck_MGT / Tpcickc_MGT	MGTCLK	PIPEGTRESETDONEA
		PIPEGTRESETDONEB
		PIPEPHYSTATUSA
		PIPEPHYSTATUSB
		PIPERXCHARISKA[1:0]
		PIPERXCHARISKB[1:0]
		PIPERXDATAA[15:0]
		PIPERXDATAB[15:0]
		PIPERXENTERELECIDLEA
		PIPERXENTERELECIDLEB
		PIPERXSTATUSA[2:0]
		PIPERXSTATUSB[2:0]
Tpcicck_PWR / Tpcickc_PWR	USERCLK	CFGPMWAKEN
		CFGTURNOFFOKN
Tpcicck_SCAN / Tpcickc_SCAN	USERCLK	SCANEN
		SCANIN[4:0]
		SCANRESETMASK
Tpcidck_LOCKED / Tpcickd_LOCKED	USERCLK	CLOCKLOCKED
Tpcidck_RESET / Tpcickd_RESET	USERCLK	SYSRESETN
Tpcidck_RXRAM / Tpcickd_RXRAM	USERCLK	MIMRXRDATA[34:0]
Tpcidck_TRNFC / Tpcickd_TRNFC	USERCLK	TRNFCSEL[2:0]
Tpcidck_TRNRD / Tpcickd_TRNRD	USERCLK	TRNRDSTRDYN
Tpcidck_TRNRN / Tpcickd_TRNRN	USERCLK	TRNRNPOKN
Tpcidck_TRNTD / Tpcickd_TRNTD	USERCLK	TRNTD[31:0]
Tpcidck_TRNTE / Tpcickd_TRNTE	USERCLK	TRNTEOFN
		TRNTERRFWDN

Table D-1: **PCIE_A1 Timing Parameters *(Cont'd)***

Name	Clock Domain	Signal Grouping
Tpcidck_TRNTS / Tpcickd_TRNTS	USERCLK	TRNTSOFN
		TRNTSRCDSCN
		TRNTSRCRDYN
		TRNTSTRN
Tpcidck_TXRAM / Tpcickd_TXRAM	USERCLK	MIMTXRDATA[35:0]
Sequential Clock to Output Times for Integrated Block Outputs		
Tpcicko_CFG	USERCLK	CFGCOMMANDINTERRUPTDISABLE
		CFGDEVCONTROLMAXPAYLOAD[2:0]
		CFGDEVCONTROLMAXREADREQ[2:0]
Tpcicko_CFGBUS	USERCLK	CFGBUSNUMBER[7:0]
Tpcicko_CFGCOMMAND	USERCLK	CFGCOMMANDSERREN
Tpcicko_CFGDEV	USERCLK	CFGDEVCONTROLCORRERRREPORTINGEN
		CFGDEVCONTROLEXTTAGEN
		CFGDEVCONTROLFATALERRREPORTINGEN
		CFGDEVCONTROLNONFATALREPORTINGEN
		CFGDEVCONTROLNOSNOOPEN
		CFGDEVCONTROLPHANTOMEN
		CFGDEVCONTROLURERRREPORTINGEN
		CFGDEVICENUMBER[4:0]
		CFGDEVSTATUSCORRERRDETECTED
		CFGDEVSTATUSFATALERRDETECTED
		CFGDEVSTATUSNONFATALERRDETECTED
		CFGDEVSTATUSURDETECTED
Tpcicko_CFGDO	USERCLK	CFGDO[31:0]
Tpcicko_CFGDONE	USERCLK	CFGRDWRDONEN
Tpcicko_CFGERR	USERCLK	CFGERRCPLRDYN
Tpcicko_CFGFCN	USERCLK	CFGFUNCTIONNUMBER[2:0]
Tpcicko_CFGINT	USERCLK	CFGINTERRUPTDO[7:0]
		CFGINTERRUPTRDYN

Table D-1: **PCIE_A1 Timing Parameters *(Cont'd)***

Name	Clock Domain	Signal Grouping
Tpcicko_CFGLINK	USERCLK	CFGLINKCONTOLRCB
		CFGLINKCONTROLASPMCONTROL[1:0]
		CFGLINKCONTROLCOMMONCLOCK
		CFGLINKCONTROLEXTENDEDSYNC
Tpcicko_CFGOFF	USERCLK	CFGTOTURNOFFN
Tpcicko_CFGSTATE	MGTCLK	CFGLTSSMSTATE[4:0]
	USERCLK	CFGPCIELINKSTATEN[2:0]
Tpcicko_DBG	USERCLK	DBGBADTLPLCRC
		DBGBADTLPSEQNUM
		DBGMLFRMDLENGTH
		DBGMLFRMDMPS
		DBGMLFRMDTCVC
		DBGMLFRMDUNRECTYPE
		DBGREGDETECTEDCORRECTABLE
		DBGREGDETECTEDFATAL
		DBGREGDETECTEDNONFATAL
		DBGREGDETECTEDUNSUPPORTED
		DBGURNOBARHIT
		DBGURPOISCFGWR
		DBGURUNSUPMSG
Tpcicko_ENA	USERCLK	CFGCOMMANDBUSMASTERENABLE
		CFGCOMMANDIOENABLE
		CFGCOMMANDMEMENABLE
		CFGDEVCONTROLENABLERO
		CFGINTERRUPTMMENABLE[2:0]
Tpcicko_MSG	USERCLK	CFGINTERRUPTMSIENABLE

Table D-1: **PCIE_A1 Timing Parameters *(Cont'd)***

Name	Clock Domain	Signal Grouping
Tpcicko_PIPE	MGTCLK	PIPEGTTXELECIDLEA
		PIPEGTTXELECIDLEB
		PIPERXPOLARITYA
		PIPERXPOLARITYB
		PIPERXRESETA
		PIPERXRESETB
		PIPETXCHARDISPMODEA[1:0]
		PIPETXCHARDISPMODEB[1:0]
		PIPETXCHARDISPVALA[1:0]
		PIPETXCHARDISPVALB[1:0]
		PIPETXCHARISKA[1:0]
		PIPETXCHARISKB[1:0]
		PIPETXDATAA[15:0]
		PIPETXDATAB[15:0]
		PIPETXRCVRDETA
		PIPETXRCVRDETB
Tpcicko_PWR	MGTCLK	PIPEGTPOWERDOWNA[1:0]
		PIPEGTPOWERDOWNB[1:0]
	USERCLK	CFGDEVCONTROLAUXPOWEREN
Tpcicko_RXRAM	USERCLK	MIMRXRADDR[11:0]
		MIMRXREN
		MIMRXWADDR[11:0]
		MIMRXWDATA[34:0]
		MIMRXWEN
Tpcicko_SCANOUT	USERCLK	SCANOUT[4:0]

Table D-1: **PCIE_A1 Timing Parameters *(Cont'd)***

Name	Clock Domain	Signal Grouping
Tpcicko_STATUS	USERCLK	DBGBADDLLPSTATUS
		DBGBADTLPSTATUS
		DBGDLPROTOCOLSTATUS
		DBGFCPROTOCOLERRSTATUS
		DBGMLFRMDTLPSTATUS
		DBGPOISTLPSTATUS
		DBGRCVROVERFLOWSTATUS
		DBGRPLYROLLOVERSTATUS
		DBGRPLYTIMEOUTSTATUS
		DBGURSTATUS
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[31:0]
		TRNREOFN
		TRNRERRFWDN
		TRNRSOFN
		TRNRSRCDSCN
		TRNRSRCRDYN
		TRNTBUFAV[5:0]
		TRNTCFGREQN
		TRNTDSTRDYN
		TRNTERRDROPN
Tpcicko_TXRAM	USERCLK	MIMTXRADDR[11:0]
		MIMTXREN
		MIMTXWADDR[11:0]
		MIMTXWDATA[35:0]
		MIMTXWEN