- THIS IS A DISCONTINUED IP CORE -

Virtex-5 LogiCORE Endpoint Block for PCI Express Designs

User Guide

UG350 (v1.6) March 24, 2008

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/01/07	1.1	Initial Xilinx release.
05/22/07	1.2	Miscellaneous typographical edits. Updated to core version 1.4.
08/15/07	1.3	Updated to core version 1.5.
10/10/07	1.4	Updated to core version 1.6.
03/24/08	1.5	Updated to core version 1.7.
03/24/08	1.6	Core version 1.7. Updated system requirements; removed support for Solaris. Miscellaneous typographical edits.

Table of Contents

Chapter 4: Generating and Customizing the Core

Chapter 5: Designing with the Core

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Licensing the Core

Chapter 3: Core Architecture

Chapter 4: Generating and Customizing the Core

Chapter 5: Designing with the Core

Schedule of Tables

XILINX®

Preface

About This Guide

This user guide provides information about the Virtex™-5 LogiCORE™ Endpoint block for PCI Express® (PCIe®) designs also known as the core. The core instantiates the integrated Endpoint block for PCIe designs found in Virtex-5 LXT and SXT devices.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Introduction" describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Licensing the Core" provides instructions for obtaining a license key for the core.
- Chapter 3, "Core Architecture" describes the main components of the core architecture.
- Chapter 4, "Generating and Customizing the Core" describes the menus for customizing the core.
- Chapter 5, "Designing with the Core" describes data transfers and timing
- Chapter 6, "Simulating with the Core" describes the setup of the simulation environment
- Chapter 7, "Implementing the Core" describes the timing and physical constraints provided in the user constraints file

Additional Documentation

The following documents are also available for download at http://www.xilinx.com/virtex5.

- Virtex-5 Family Overview

  The features and product selection of the Virtex-5 family are outlined in this overview.
- Virtex-5 Data Sheet: DC and Switching Characteristics

  This data sheet contains the DC and Switching Characteristic specifications for the Virtex-5 family.
- Virtex-5 Integrated Endpoint Block User Guide for PCI Express Designs

  This guide describes the integrated Endpoint blocks in the Virtex-5 LXT and SXT platform devices that are PCI Express® compliant.

- Virtex-5 User Guide

  Chapters in this guide cover the following topics: Clocking resources, Clock Management Technology (CMT), Phase-Locked Loops (PLLs), block RAM, Configurable Logic Blocks (CLBs), SelectIO™ resources, and SelectIO logic resources

- Virtex-5 RocketIO GTP Transceiver User Guide

  This guide describes the RocketIO™ GTP transceivers available in the Virtex-5 LXT and SXT platform devices.

- Virtex-5 RocketIO GTX Transceiver User Guide

  This guide describes the RocketIO GTX transceivers available in the Virtex-5 FXT platform devices.

- Virtex-5 Tri-Mode Ethernet Media Access Controller

  This guide describes the dedicated Tri-Mode Ethernet Media Access Controller available in the Virtex-5 LXT and SXT platform devices.

- XtremeDSP Design Considerations

  This guide describes the XtremeDSP™ slice and includes reference designs for using the DSP48E.

- Virtex-5 Configuration Guide

  This all-encompassing configuration guide includes chapters on configuration interfaces (serial and SelectMAP), bitstream encryption, Boundary-Scan and JTAG configuration, reconfiguration techniques, and readback through the SelectMAP and JTAG interfaces.

- Virtex-5 System Monitor User Guide

  The System Monitor functionality available in all the Virtex-5 devices is outlined in this guide.

- Virtex-5 Packaging Specifications

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

- Virtex-5 PCB Designer's Guide

  This guide provides information on PCB design for Virtex-5 devices, with a focus on strategies for making design decisions at the PCB and interface level.

Additional Support Resources

To search the database of silicon and software questions and answers, or to create a technical support case in WebCase, see the Xilinx website at: http://www.xilinx.com/support.

Typographical Conventions

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

Convention	Meaning or Use	Example
Italic font	References to other documents	See the *Virtex-5 Configuration Guide* for more information.
	Emphasis in text	The address (F) is asserted *after* clock event 2.
Underlined Text	Indicates a link to a web page.	http://www.xilinx.com/virtex5

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 Documentation" for details.
Red text	Cross-reference link to a location in another document	See Figure 5 in the *Virtex-5 Data Sheet*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest documentation.

Chapter 1

Introduction

This chapter introduces the Virtex™-5 LogiCORE™ Endpoint Block for PCIe™ designs and provides related information including system requirements, recommended design experience, additional core resources, technical support, and submitting feedback to Xilinx.

About the Core

LogiCORE IP solutions for PCIe are reliable high-bandwidth scalable serial interconnect intellectual property building blocks. The Virtex-5 LogiCORE Endpoint Block for PCIe wrapper instantiates the integrated Endpoint block found in Virtex-5 LXT, SXT, and FXT Platform devices and supports Verilog and VHDL.

The Endpoint Block for PCIe core is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see www.xilinx.com/products/ipcenter/V5_PCI_Express_Block.htm. For information about licensing options, see Chapter 2, "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

Tools

- ISE v10.1
- Mentor Graphics® ModelSim®: v6.3c
- Cadence® IUS v6.1
- Synopsys® vcs_mxY-2006.06-SP1

Recommended Design Experience

Although the 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 on the core, see the following documents:

- DS533, *Virtex-5 LogiCORE Endpoint Block for PCI Express Data Sheet*
- *Endpoint Block for PCI Express Release Notes* (available from the product lounge)
- UG197, *Virtex-5 Endpoint Block for PCI Express Designs Users Guide*

Additional information and resources related to the PCIe technology are available from the following websites:

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

Technical Support

Xilinx provides technical support for use of this product. For technical support, go to http://www.xilinx.com/support. Questions are routed to a team of engineers with expertise using the core.

Feedback

Xilinx welcomes comments and suggestions about the LogiCORE Endpoint Block for PCIe core and the accompanying documentation.

Core

For comments or suggestions about the core, please submit a WebCase from http://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 http://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

Licensing the Core

This chapter provides instructions for obtaining a license key for the Virtex™-5 LogiCORE™ Endpoint Block for PCIe™ designs, which you must do before using it in your designs. This 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 information about installing the core, see the Virtex-5 Endpoint Block for PCI Express product page.

License Options

The core provides two licensing options: a Simulation Only Evaluation license and a Full license. Both licensing options are available free of cost to the user. After installing the core, the Simulation Only Evaluation license is provided by default with the CORE Generator. To install the Full License, follow the directions in the "Full" section below.

Simulation Only

The Simulation Only Evaluation license is provided with the core from the Xilinx CORE Generator. This license lets you assess the core functionality with either the provided example design or alongside your own design and demonstrates the various interfaces to the core in simulation.

Full

The Full license provides full access to all core functionality both in simulation and in hardware. Full implementation support includes place and route and bitstream generation.

Obtaining Your License

Simulation Only Evaluation License

The Simulation Only Evaluation license is provided with the CORE Generator system and requires no license file.

Obtaining a Full License

To obtain a Full license, you must register for access to the *lounge*, a secured area of the Virtex-5 Endpoint Block for PCI Express product page.

- From the product page, click Register to register and request access to the lounge. Access to the lounge is automatic and granted immediately.
- After you receive confirmation of lounge access, click Access Lounge on the product page and log in.
- Click Access Lounge on the product lounge page and fill out the license request form linked from this location; then click Submit to automatically generate the license. An e-mail containing the license and installation instructions will be sent immediately to the e-mail address you specified.

Installing Your License File

If you select the Full License option, you will receive an e-mail that includes instructions for installing your license file. In addition, information about advanced licensing options and technical support is provided.

XILINX®

Chapter 3

Core Architecture

This chapter describes the main components of the Virtex™-5 LogiCORE™ Endpoint block for PCIe™ (the core).

Overview

The core internally instances the integrated Endpoint block. For information regarding the internal architecture of the block, see UG197: *Virtex-5 Integrated Endpoint Block for PCI Express Designs Users Guide*. The integrated block follows the *PCI Express Base Specification* layering model, which consists of the Physical, Data Link, and Transaction Layers.

Figure 3-1, page 22 illustrates the following interfaces to the core:

- Clock and Reset Interface (CLK_RST)
- Configuration and Status Interface (CFG_STAT)
- Power Management Interface (PWR_MGMT)
- Interface to PCI Express Partner (PCI_EXP)
- Management Interface (MGMT)
- Transaction Layer Interface (TLI)

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.

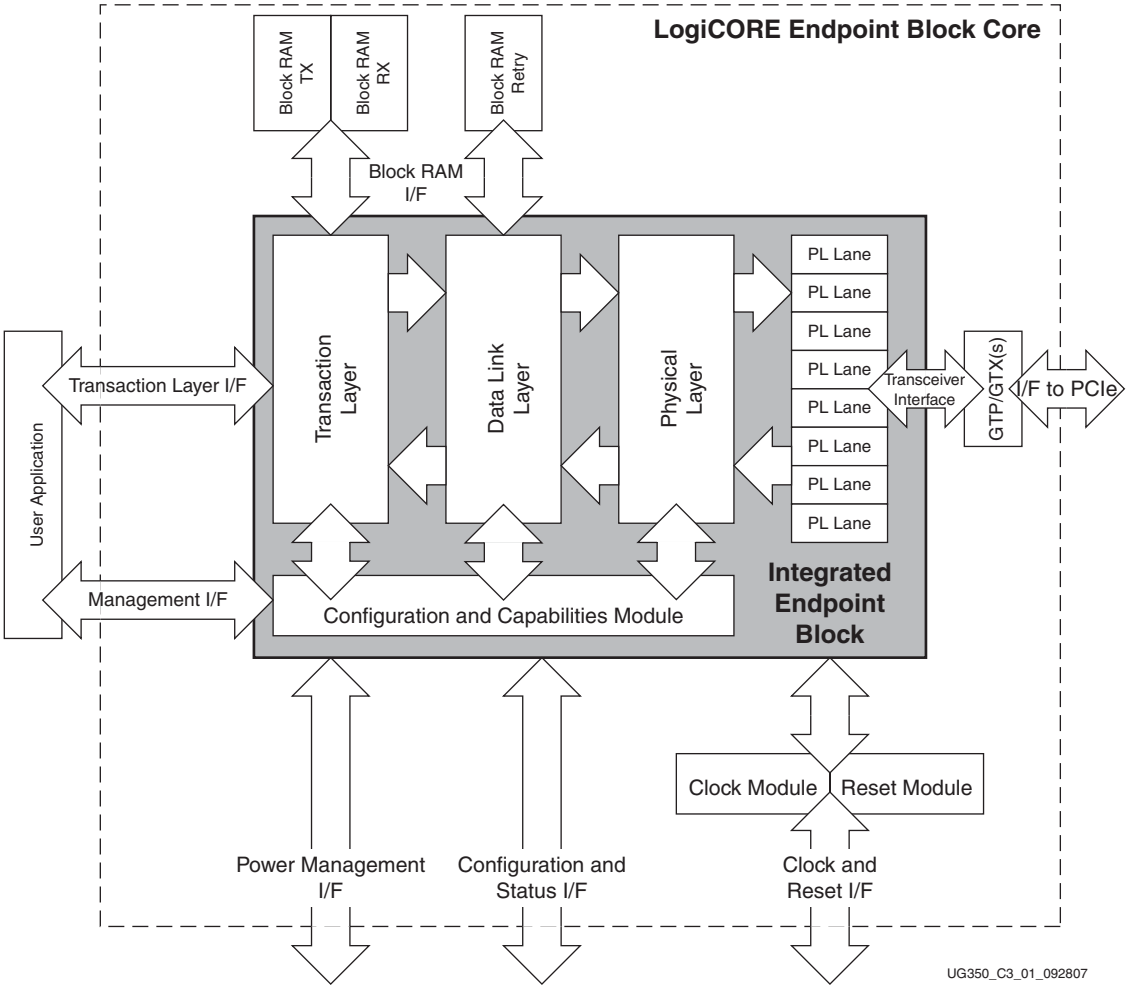


Figure 3-1: **Top-Level Functional Blocks and Interfaces**

Protocol Layers

The functions of the protocol layers, as defined by the *PCI Express Base Specification*, include generation and processing of Transaction Layer Packets (TLPs), flow control management, initialization and 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 as follows:

Transaction Layer

The Transaction Layer is the upper layer of the PCIe architecture, and its primary function is to accept, buffer, and disseminate Transaction Layer packets or TLPs. TLPs communicate information through the use of memory, I/O, configuration, and message transactions. To maximize the efficiency of communication between devices, the Transaction Layer enforces PCI Express 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 acknowledgements.

Physical Layer

The Physical Layer interfaces the Data Link Layer with signaling 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, descrambling and 8B/10B encoding and decoding of data are also performed in this sub-block. The Electrical sub-block defines the input and output buffer characteristics that interface to the device to the PCIe link.

Configuration Management

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

- Implements PCIe 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 INTx interrupt emulation
- Implements the Device Serial Number Capability in the PCIe Extended Capability space

PCI Express Configuration Space

The configuration space consists of three primary parts, illustrated in Table 3-1, page 24. 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 Capability Item
- PCIe Extended Capabilities
  - Device Serial Number Extended Capability Structure
  - Virtual Channel Capability Structure

Table 3-1: **PCI Express 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
Reserved Legacy Configuration Space (Returns 0x00000000)				058h-05Ch

Table 3-1: **PCI Express Configuration Space Header *(Continued)***

31	16	15	0	
PE Capability		NxtCap	PE Cap	060h
PCI Express Device Capabilities				064h
Device Status		Device Control		068h
PCI Express Link Capabilities				06Ch
Link Status		Link Control		070h
Reserved Legacy Configuration Space (Returns 0x00000000)				074h-0FFh
Next Cap	Cap. Ver.	PCI Exp. Capability		100h
PCI Express Device Serial Number (1st)				104h
PCI Express Device Serial Number (2nd)				108h
Next Cap	Cap Ver	PCI Exp Capability		10Ch-130h
Port VC Capability Register 1				
Port VC Capability Register 2				
Port VC Status Register		Port VC Control Register		
VC Resource Capability Register (0)				
VC Resource Control Register (0)				
VC Resource Status Reg (0)		RsvdP		
VC Resource Capability Register (1)				
VC Resource Control Register (1)				
VC Resource Status Reg (1)		RsvdP		
Reserved Extended Configuration Space (Returns 0x00000000)				134h-FFFh

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. Writes to this range are ignored. The core also implements two PCIe Extended Capabilities. The remaining PCIe Extended Capability space from addresses `0x134` to `0xFFF` is reserved. The core returns `0x00000000` when this address range is read. Writes to this range are ignored.

Core Interfaces

The core contains several interfaces which are grouped based on similar functionality. Some ports are specific to receive or transmit and some are common to both directions.

Clock and Reset Interface

The Clock and Reset Interface consists of the user controlled resets to reset various components of the core, monitors for PLLs and internally generated clocks. Table 3-2 shows the Clock and Reset Interface ports.

Table 3-2: **Clock and Reset Interface Ports**

Port	Direction	Clock Domain	Description
user_reset_n	Input	N/A	Asynchronous reset input which resets the entire core. User should tie this to the system reset function.
core_clk	Output	core_clk	Monitor output to track the core_clk generated in the core.
user_clk	Output	user_clk	Monitor output to track the user_clk generated in the core.
clock_lock	Output	N/A	Monitor output to track the clock lock in the PLL used in the clocking module in the wrapper. When High, indicates all clocks in the design are stable.
pllkdet_out[3:0]	Output	N/A	Monitor output to track the clock lock of the PLL in each GTP_DUAL/GTX_DUAL used in the design. 0: lanes 0, 1 1: lanes 2, 3 2: lanes 4, 5 3: lanes 6, 7
gt_reset	Input	core_clk	Used to reset all the GTP/GTX transceivers in the design.
gsr	Input	N/A	Asynchronous reset signal used to reset the core to ensure accurate simulation behavior with Xilinx Libraries. In a simulation environment, it should be tied to the glbl.GSR signal in Xilinx Libraries. In other cases, tie to 0.

Table 3-2: **Clock and Reset Interface Ports *(Continued)***

Port	Direction	Clock Domain	Description
crm_urst_n (1)	Input	core_clk	User reset (active-Low). When the RESETMODE attribute is set to FALSE, resets all the registers in the integrated block, except the sticky registers and the Management Interface registers. When the RESETMODE attribute is set to TRUE, resets the backend interface to the Transaction Layer (user_clk domain). Asynchronous, but the integrated block ensures internal synchronous deassertion with respect to core_clk. Tie High if not used in the user design or if the block is not used.
crm_nvrst_n (1)	Input	core_clk	Non-volatile reset (active-Low). When the RESETMODE attribute is set to FALSE, resets the sticky registers, and everything else in the integrated block except for the Management Interface registers. When the RESETMODE attribute is set to TRUE, resets the sticky registers only. Asynchronous, but the integrated block ensures internal synchronous deassertion with respect to core_clk. Tie High if not used in the user design or if the integrated block is not used.
crm_mgmt_rst_n (1)	Input	core_clk	Management interface reset (active-Low). Resets the registers in the block, including the management interface registers. The function of this signal does not depend on the RESETMODE attribute setting. Asynchronous, but the integrated block ensures internal synchronous deassertion with respect to core_clk. Tie High if not used in the user design or if the block is not used.
crm_user_cfg_rst_n (1)	Input	core_clk	User configuration reset (active-Low). Resets all the registers in the PCIe Configuration Space, except sticky registers. The function of this signal does not depend on the RESETMODE attribute setting. Asynchronous, but the integrated block ensures internal synchronous deassertion with respect to core_clk. Tie High if not used in the user design or if the block is not used.

Table 3-2: **Clock and Reset Interface Ports *(Continued)***

Port	Direction	Clock Domain	Description
crm_mac_rst_n (1)	Input	core_clk	MAC reset (active-Low). When the RESETMODE attribute is set to FALSE, CRMMACRSTN is not used and must be tied High. When the RESETMODE attribute is set to TRUE, CRMMACRSTN resets the MAC link and MAC lane logic (Physical Layer). Asynchronous, but the integrated block ensures internal synchronous deassertion with respect to core_clk. Tie High if not used in the user design or if the block is not used.
crm_link_rst_n (1)	Input	core_clk	Link reset (active-Low). When the RESETMODE attribute is set to FALSE, CRMLINKRSTN is not used and must be tied High. When the RESETMODE attribute is set to TRUE, CRMLINKRSTN resets the core_clk domain of the Transaction Layer, part of the Configuration module, and the Data Link Layer. Asynchronous, but the integrated block ensures internal synchronous deassertion with respect to core_clk. Tie High if not used in the user design or if the block is not used.
gtclk_bufg	Output	core_clk	Serves as a monitor for the clock output (TXOUTCLK) of the PLL in the lane 0 GTP/GTX transceiver. It also serves as the input clock to DCM-based clock generation in the clocking module.
reflckout_bufg	Output	core_clk	Serves as a monitor for the clock input to the lane 0 GTP/GTX transceiver. It also serves as the input clock to CMT PLL based clock generation in the clocking module.
resetdone[7:0]	Output	core_clk	Indicates if the reset sequence in each of the GTP/GTX transceivers is complete. Tracks the RESETDONE signal of the GTP/GTX transceiver.

Notes:

1. Use the reset logic in the core at all times. However, by setting the attribute USER_RESETS to `1`, the user can choose to control the individual resets of the integrated block by using these reset ports.

Configuration and Status Interface

The Configuration and Status Interface (CFG_STAT) contains signals that monitor information related to all layers of the core. Specific signals are provided to the user to set bits in the configuration space based on error conditions encountered in the user logic. The ports are listed in Table 3-3, page 29.

Table 3-3: **Configuration and Status Ports**

Port	Direction	Clock Domain	Description
compliance_avoid	Input	core_clk	Modifies the rules for entering POLLING.COMPLIANCE from POLLING.ACTIVE (see Section 4.2.6.2.1 of the *PCI Express Base Specification*). When 0, the block enters POLLING.COMPLIANCE if *any* lane that detected a receiver during Detect has not detected an exit from Electrical Idle since entering POLLING.ACTIVE. When 1, the block enters POLLING.COMPLIANCE if *all* the lanes that detected receivers have not detected exit from Electrical Idle since entering POLLING.ACTIVE. This option is provided to cope with broken lanes in the receive path.
l0_first_cfg_write_occurred	Output	user_clk	Asserted following the completion of the first configuration write after reset.
l0_cfg_loopback_master	Input	core_clk	Remote device loopback control, used to check the physical connectivity of a link. When asserted, causes the MAC to send training sequences, which put the device at the other end of the link into loopback mode. The remote device then loops back all packets sent by this device until l0_cfg_loopback_master is deasserted, causing the link to retrain.
l0_cfg_loopback_ack	Output	core_clk	Asserted after the block has entered the Loopback master state.
l0_rx_mac_link_error[1:0]	Output	core_clk	Used to report link errors. Bit 1 asserted indicates a receiver error. Bit 0 asserted indicates a link training error.
l0_mac_link_up	Output	core_clk	Driven High when link training has completed and the link is operational.
l0_mac_negotiated_link_width[3:0]	Output	core_clk	Link width selected after negotiation, as follows: 0001: One lane 0010: Two lanes 0100: Four lanes 1000: Eight lanes
l0_mac_link_training	Output	core_clk	Indicates that link training is in progress. Reset to logic 1. Deasserted Low when the link reaches the L0 state at the end of link training. If link training fails, the signal is pulsed Low for one clock cycle before the block reenters the detect state.

Table 3-3: **Configuration and Status Ports *(Continued)***

Port	Direction	Clock Domain	Description
l0_ltssm_state[3:0]	Output	core_clk	The state of the Link Training and Status State Machine, encoded as follows: 0000: Initial 0001: Detect 0010: Polling 0011: Configuration 0100: L0 0101: L0s TX 0110: L1 0111: L2 1000: Testmode Wait 1001: Loopback 1010: Hot Reset 1011: Disabled 1100: Recovery 1101: L0 to Recovery 1110: L0 to L0s TX 1111: L0 to L1/L2 *Note:* The encodings 1101, 1110, and 1111, corresponding to L0 transition states, identify where the device is still nominally in the L0 state but cannot transmit data.
l0_dl_up_down[7:0]	Output	core_clk	When operating as an Endpoint, a rising edge on this signal flags the transition from the InitFC1 phase of the initialization procedure to the InitFC2 phase. A falling edge indicates that the link has been broken and is used as a trigger for the link to be reset. There is a bit for each implemented VC. Bits [7:2] are never used.
l0_dll_error_vector[6:0]	Output	core_clk	Error signals relating to DLL data. Errors detected within the DLL result in the relevant bit in the vector being asserted for one clock period: bit 0: DLLBADTLP bit 1: DLLBADDLLP bit 2: DLLREPLAYTIMEOUT bit 3: DLLREPLAYROLLOVER bit 4: *Reserved* bit 5: RXTLPMISSING bit 6: DLLPROTOCOLERROR A missing TLP triggers both bit 5 and bit 0. The same error in consecutive DLLPs results in the associated bit being asserted for more than one clock period where the DLLPs are presented back-to-back to the Data Link Layer by the MAC.

Table 3-3: **Configuration and Status Ports *(Continued)***

Port	Direction	Clock Domain	Description
l0_completer_id[12:0]	Output	user_clk	Bus number and device number components of Completer ID. Append the 3-bit Function number, 0, to form the full 16-bit Completer ID. Also used as the Requester ID for Request TLPs.
l0_transactions_pending	Input	user_clk	Assert when there are outstanding transactions pending. Setting reflected in setting of the Transaction Pending bit in the Device Status register.
l0_set_completer_abort_error	Input	user_clk	When asserted, causes the relevant Completer Abort status bit(s) to be set to 1.
l0_set_detected_corr_error	Input	user_clk	When asserted, causes the relevant Correctable Error status bit(s) to be set to 1. If bit 0 of the Device Control Register is set (Correctable Error Reporting Enable), then a Correctable Error Message is also sent.
l0_set_detected_fatal_error	Input	user_clk	When asserted, causes the relevant Fatal Error status bit(s) to be set to 1. If bit 2 of the Device Control Register is set (Fatal Error Reporting Enable) or bit 8 of the Command Register is set (SERR Enable), then a Fatal Error Message is also sent.
l0_set_detected_nonfatal_error	Input	user_clk	When asserted, causes the relevant Nonfatal Error status bit(s) to be set to 1. If bit 1 of the Device Control Register is set (Non-Fatal Error Reporting Enable) or bit 8 of the Command Register is set (SERR Enable), then a Non-Fatal Error Message is also sent.
l0_set_user_detected_parity_error	Input	user_clk	When asserted, causes the relevant Parity Error status bit(s) to be set to 1.
l0_set_user_master_data_parity	Input	user_clk	When asserted, causes the relevant Master Data Parity status bit(s) to be set to 1.
l0_set_user_received_master_abort	Input	user_clk	When asserted, causes the relevant Master Abort status bit(s) to be set to 1.
l0_set_user_received_target_abort	Input	user_clk	When asserted, causes the relevant Target Abort status bit(s) to be set to 1.
l0_set_user_system_error	Input	user_clk	When asserted, causes the relevant System Error status bit(s) to be set to 1.
l0_set_user_signalled_target_abort	Input	user_clk	When asserted, causes the relevant Target Abort status bit(s) to be set to 1.
l0_set_completion_timeout_uncorr_error	Input	user_clk	Asserted to indicate that a requester has not seen a completion and has handled this as an Uncorrectable Error. Causes the relevant Completion Timeout status bit(s) to be set to 1.

Table 3-3: **Configuration and Status Ports *(Continued)***

Port	Direction	Clock Domain	Description
l0_set_completion_timeout_corr_error	Input	user_clk	Asserted to indicate that a requester has not seen a completion and has handled this as a Correctable Error. Causes the relevant Completion Timeout status bit(s) to be set to 1.
l0_set_unexpected_completion_uncorr_error	Input	user_clk	Asserted to indicate that a receiver has received an unexpected completion and has handled this as an Uncorrectable Error. Causes the relevant Unexpected Completion status bit(s) to be set to 1.
l0_set_unexpected_completion_corr_error	Input	user_clk	Asserted to indicate that a receiver has received an unexpected completion and has handled this as a Correctable Error. Causes the relevant Unexpected Completion status bit(s) to be set to 1.
l0_set_unsupported_request_nonposted_error	Input	user_clk	Asserted to indicate that a completer has received an unsupported non-posted request. Causes the relevant unsupported request status bit(s) to be set to 1.
l0_set_unsupported_request_other_error	Input	user_clk	Asserted to indicate that a completer has received some other kind of unsupported request (other than a non-posted request). Causes the relevant unsupported request status bit(s) to be set to 1.
l0_legacy_int_funct0	Input	user_clk	Drive High to request Legacy Interrupt on Function 0.
l0_msi_enable0	Output	user_clk	Asserted when MSI is enabled for Function 0.
l0_multi_msg_en0	Output	user_clk	Asserted when MSI multiple messages are enabled for Function 0.
l0_stats_dllp_received	Output	core_clk	Asserted for a single clock cycle when a DLLP is received.
l0_stats_dllp_transmitted	Output	core_clk	Asserted for a single clock cycle when a DLLP is transmitted.
l0_stats_os_received	Output	core_clk	Asserted for a single clock cycle when an ordered set is received.
l0_stats_os_transmitted	Output	core_clk	Asserted for a single clock cycle when an ordered set is transmitted.
l0_stats_tlp_received	Output	core_clk	Asserted for a single clock cycle when a TLP is received.
l0_stats_tlp_transmitted	Output	core_clk	Asserted for a single clock cycle when a TLP is transmitted.
l0_stats_cfg_received	Output	user_clk	Asserted for a single cycle of CRMUSERCLK when a configuration packet is received by the configuration block.

Table 3-3: **Configuration and Status Ports *(Continued)***

Port	Direction	Clock Domain	Description
l0_stats_cfg_transmitted	Output	user_clk	Asserted for a single cycle of CRMUSERCLK when a configuration packet is transmitted by the configuration block.
l0_stats_cfg_other_received	Output	user_clk	Asserted for a single cycle of CRMUSERCLK when a packet of any other type (e.g., a message packet or a posted memory write packet relating to MSI) is received by the configuration block.
l0_stats_cfg_other_transmitted	Output	user_clk	Asserted for a single cycle of CRMUSERCLK when one of these other types of packet is transmitted by the configuration block.
io_space_enable	Output	user_clk	I/O space enable. When 1, shows that response to I/O request packets has been enabled.
mem_space_enable	Output	user_clk	Memory space enable. When 1, response to memory request packets has been enabled.
max_payload_size[2:0]	Output	user_clk	Negotiated Maximum Payload size, as follows: 000: 128 bytes 001: 256 bytes 010: 512 bytes 011: 1024 bytes 100: 2048 bytes 101: 4096 bytes 110: *Reserved* 111: *Reserved*
max_read_request_size[2:0]	Output	user_clk	Negotiated Read request size, as follows: 000: 128 bytes 001: 256 bytes 010: 512 bytes 011: 1024 bytes 100: 2048 bytes 101: 4096 bytes 110: *Reserved* 111: *Reserved*
bus_master_enable	Output	user_clk	Bus Master Enable. When 0, Endpoint is prevented from issuing any memory or I/O requests.
parity_error_response	Output	user_clk	Parity Error Response. When 1, response to Master Data parity errors has been enabled.
serr_enable	Output	user_clk	SERR Enable. When 1, reporting of fatal and nonfatal errors has been enabled.

Table 3-3: **Configuration and Status Ports *(Continued)***

Port	Direction	Clock Domain	Description
interrupt_disable	Output	user_clk	Interrupt Disable. When 1, device is prevented from generating INTx interrupt messages.
ur_reporting_enable	Output	user_clk	Unsupported request reporting enable. When 1, reporting of unsupported requests has been enabled.

Power Management Interface

The Power Management Interface (PWR_MGMT) contains signals which monitor information related to the power management states of the core.

Table 3-4: **Power Management Interface Ports**

Port	Direction	Clock Domain	Description
l0_pwr_state0[1:0]	Output	user_clk	Indicates the current power state as follows: 00: D0 01: D1 10: D2 11: D3 Can be used to inhibit transfers when the core is in D1 or D2 state.
l0_pwr_l23_ready_state	Output	user_clk	Asserted when the link is in the L2/L3 power state.
l0_pwr_tx_l0s_state	Output	user_clk	Asserted when TX link is in the L0s state.
l0_pwr_turn_off_req	Output	user_clk	Asserted when the core has received a PME_Turn_Off message.
l0_mac_new_state_ack	Output	core_clk	Acknowledgement that the link has transitioned to the requested new link power state.
l0_mac_rx_l0s_state	Output	core_clk	Asserted when the RX link is in the L0s state.
l0_mac_entered_l0	Output	core_clk	Pulsed when the MAC transitions back into the L0 power state.

Interface to PCI Express Partner

The interface (PCI_EXP) consists of differential transmit and receive pairs organized in multiple lanes. A PCIe lane "m" consists of a pair of transmit differential signals {txp[m], txn[m]} and a pair of receive differential signals {rxp[m], rxn[m]}.

Table 3-5: **Interface Ports**

Port	Direction	Clock Domain	Description
REFCLK	Input	N/A	Reference Clock for the system. User can connect this to the reference clock on the board (100/125/250 MHz). Refer to Chapter 5, "Designing with the Core" for details on clocking.
txn[*n*-1:0](1)	Output	N/A	Transmit Negative: Serial differential signals for each lane.
txp[*n*-1:0](1)	Output	N/A	Transmit Positive: Serial differential signals for each lane.
rxn[*n*-1:0](1)	Input	N/A	Receive Negative: Serial differential signals for each lane.
rxp[*n*-1:0](1)	Input	N/A	Receive Positive: Serial differential signals for each lane.

Notes:

1. n is of a value of 1, 2, 4, or 8.

Management Interface

The Management interface (MGMT) is used to access various registers and signals in the core, including the PCIe Configuration Space, and various control and status registers. The Management Interface also contains output signals for statistics and monitoring and an interface to read flow control credit outputs.

Table 3-6 shows the ports of the Management interface.

Table 3-6: **Management Interface Ports**

Port	Direction	Clock Domain	Description
mgmt_rd_data[31:0]	Output	user_clk	Management Interface read data.
mgmt_wdata[31:0]	Input	user_clk	Management Interface write data.
mgmt_bwren[3:0]	Input	user_clk	Management Interface byte write enables.
mgmt_wren	Input	user_clk	Management Interface write enable.
mgmt_addr[10:0]	Input	user_clk	Management Interface address. See Registers section in Chapter 2 of UG197 for Management Interface address mapping.
mgmt_rden	Input	user_clk	Management Interface read enable.
mgmt_rddone	Output	user_clk	Indicates that the read transaction completed successfully.

Table 3-6: **Management Interface Ports *(Continued)***

Port	Direction	Clock Domain	Description
mgmt_wrdone	Output	user_clk	Indicates that the write transaction completed successfully.
mgmt_stats_credit[11:0]	Output	user_clk	Credit information as selected by MGMT_STATS_CREDIT_SEL[6:0].
mgmt_stats_credit_sel[6:0]	Input	user_clk	Channel select for credit information output to MGMT_STATS_CREDIT[11:0]. Bits[1:0] select the VC: 00: VC0 01: VC1 10: *Reserved* 11: *Reserved* Bits[4:2] select the channel. 000: Posted header (PH) 001: Non-posted header (NPH) 010: Completion header (CH) 011: Posted data (PD) 100: Non-posted data (NPD) 101: Completion data (CD) 110: *Reserved* 111: *Reserved* Bits[6:5] select the type of credit information. 00: credits consumed - TX credits used by transmitted packets 01: credit limit - TX credits received from link partner 10: credits allocated - RX credits issued to link partner 11: credits received - RX credits consumed by received packets
mgmt_pso[16]	Output	user_clk	Master Data parity error. Bit 8 of the Status register.
mgmt_pso[15]	Output	user_clk	Signaled Target abort. Bit 11 of the Status register.
mgmt_pso[14]	Output	user_clk	Received Target abort. Bit 12 of the Status register.
mgmt_pso[13]	Output	user_clk	Received Master abort. Bit 13 of the Status register.
mgmt_pso[12]	Output	user_clk	Signaled system error. Bit 14 of the Status register.
mgmt_pso[11]	Output	user_clk	Detected parity error (poisoned TLP). Bit 15 of the Status register.
mgmt_pso[10]	Output	user_clk	Correctable error detected. Bit 0 of the Device Status register.
mgmt_pso[9]	Output	user_clk	Nonfatal error detected. Bit 1 of the Device Status register.
mgmt_pso[8]	Output	user_clk	Fatal error detected. Bit 2 of the Device Status register.
mgmt_pso[7]	Output	user_clk	Unsupported request detected. Bit 3 of the Device Status register.
mgmt_pso[6]	Output	user_clk	Transactions Pending. Bit 5 of the Device Status register.
mgmt_pso[5:0]	Output	user_clk	*Reserved.*

Transaction Layer Interface

The Transaction Layer Interface (TLI) is used to send and receive packets between Transaction Layer and User Logic. The main Transaction Layer interface framing signals indicate the start of frame, the end of frame, destination ready, and source ready. Besides this, it contains signals which indicate status of the buffer space for all channels in both directions. These need to be monitored to manage traffic flow on the TLI Interface.

Table 3-7 shows the ports of the Transaction Layer interface.

Table 3-7: **Transaction Layer Interface Ports**

Port	Direction	Clock Domain	Description
llk_tc_status[7:0]	Output	user_clk	Report the status of the eight traffic classes: `1` implies initialized; `0` implies uninitialized.
llk_tx_data[63:0]	Input	user_clk	Transaction Layer interface transmit data.
llk_tx_src_rdy_n	Input	user_clk	Asserted (active-Low) if the transmit source has data available.
llk_tx_dst_rdy_n	Output	user_clk	Asserted (active-Low) if the transmit destination has space available on the selected channel.
llk_tx_chan_space[9:0]	Output	user_clk	Amount of free space in the TX FIFO as selected by llk_tx_ch_tc and llk_tx_ch_fifo. Bit [9] indicates if space is available for header: `1`: Space for one header `0`: No space for header Bit [8] indicates if space is available for data: `1`: Space for data `0`: No space for data Bits [7:0] indicate the number of data credits available: 1 .. 255: Number of credits available `0`: Meaning depends on bit [8] setting: – If bit [8] = 0, no credits are available. – If bit [8] = 1, at least 256 credits are available.
llk_tx_sof_n	Input	user_clk	Transaction Layer interface TX Start of Frame (active-Low).
llk_tx_eof_n	Input	user_clk	Transaction Layer interface TX End of Frame (active-Low).
llk_tx_enable_n[1:0]	Input	user_clk	Word enable for Transaction Layer interface Transmit bus (active-Low).
llk_tx_ch_tc[2:0]	Input	user_clk	Traffic class portion of Channel Select.
llk_tx_ch_fifo[1:0]	Input	user_clk	FIFO portion of Channel Select. `00`: Posted `01`: Non-posted `10`: Completion `11`: *Reserved*

Table 3-7: **Transaction Layer Interface Ports *(Continued)***

Port	Direction	Clock Domain	Description
llk_tx_ch_posted_ready_n[7:0]	Output	user_clk	Channel ready for posted packets TC7 – TC0 (active-Low).
llk_tx_ch_non_posted_ready_n[7:0]	Output	user_clk	Channel ready for non-posted packets TC7 – TC0 (active-Low).
llk_tx_ch_completion_ready_n[7:0]	Output	user_clk	Channel ready for completion packets TC7 – TC0 (active-Low).
llk_rx_data[63:0]	Output	user_clk	Transaction Layer interface receive data.
llk_rx_src_rdy_n	Output	user_clk	Asserted (active-Low) for one cycle if the receive source has data available on llk_rx_data in response to an earlier llk_rx_dst_req_n. Data must be captured by the user design during the cycle llk_rx_src_rdy_n is asserted. The llk_rx_src_rdy_n signal reflects the value of llk_rx_valid[1:0]. llk_rx_src_rdy_n =1 when llk_rx_valid_n = 11. llk_rx_src_rdy_n = 0 when either bit of llk_rx_valid_n = 0.
llk_rx_dst_req_n	Input	user_clk	Receive data destination request (active-Low). See llk_rx_src_last_req_n.
llk_rx_src_last_req_n	Output	user_clk	Asserted three cycles after the block has received the penultimate request for the current RX packet. If llk_rx_dst_req_n was asserted in either of the previous two cycles, then the block has received the final request for the current RX packet. No further requests should be issued (with llk_rx_dst_req_n, including on the current cycle) unless there are further packets available on the selected channel as indicated by the corresponding llk_rx_ch_*_available_n signal, where * is posted, nonposted, or completion (active-Low).
llk_rx_sof_n	Output	user_clk	Transaction Layer interface RX Start of Frame (active-Low).
llk_rx_eof_n	Output	user_clk	Transaction Layer interface RX End of Frame (active-Low).
llk_rx_valid_n[1:0]	Output	user_clk	Word enable for Transaction Layer interface receive bus (active-Low).
llk_rx_ch_tc[2:0]	Input	user_clk	Traffic class portion of Channel Select.
llk_rx_ch_fifo[1:0]	Input	user_clk	FIFO portion of Channel Select. 00: Posted 01: Non-posted 10: Completion 11: *Reserved*
llk_rx_ch_posted_available_n[7:0]	Output	user_clk	Traffic classes with complete posted packets available (active-Low).

Table 3-7: **Transaction Layer Interface Ports *(Continued)***

Port	Direction	Clock Domain	Description
llk_rx_ch_non_posted_available_n[7:0]	Output	user_clk	Traffic classes with complete non-posted packets available (active-Low).
llk_rx_ch_completion_available_n[7:0]	Output	user_clk	Traffic classes with complete completion packets available (active-Low).
llk_rx_preferred_type[15:0]	Output	user_clk	Used with llk_rx_ch_*_available_n to determine which queues have packets that can be read from the associated FIFO. The bits are interpreted in pairs with bits [1:0] allocated to TC0, bits [3:2] to TC1, and so on. Within those two bits: `00`: Posted `01`: Non-posted `10`: Completion `11`: *Reserved*
llk_rx_dst_cont_req_n	Input	user_clk	Asserted (active-Low) to allow continuous requests when receiving back-to-back packets. Should only be asserted in cases when additional packet(s) of the same type are waiting to be drained after the current one. When deasserted, the block ignores llk_rx_dst_req_n on the cycle that llk_rx_src_last_req_n is asserted and on the previous cycle (inserting two dead cycles between packets).
bar_hit[5:0]	Output	user_clk	Indicates which BAR (0 to 5), the received packet matched. When there is a match, the corresponding bit is asserted two clock cycles after llk_rx_sof_n is asserted and deasserted one clock cycle after llk_rx_eof_n is asserted. Note: The signals are meaningful only when the user selects the optional BAR monitoring logic to be generated through the GUI. In the case where the BAR monitoring logic is not selected all bits are tied to `0`.

XILINX®

Chapter 4

Generating and Customizing the Core

This chapter describes the menus for the CORE Generator GUI, which is used to generate the core. The sections include:

- "Endpoint Block Menu"
- "Configuration Space Header Fields Menu"
- "Base Address Register Configuration Menu"
- "PCI Express Capability Configuration Menu"
- "Extended Capability Configuration Menu"
- "Advanced Physical Layer Settings Menu"
- "Buffer Setup Menu"
- "Summary Window"
- "Directory Structure"

The GUI enables user access to all features of the integrated Endpoint block, including the advanced features.

Introduction to the GUI

The integrated Endpoint block is highly complex and customizable. The GUI is provided to customize and generate a subsystem using a simple set of menu options. The subsystem contains the integrated Endpoint block, GTP/GTX tiles, block RAM, clock module, and reset module, which are all automatically configured and connected. The options available in the GUI determine the correct attribute settings and tie off any unneeded ports. Clicking on the desired options in the GUI generates a completely customized core. The GUI performs the following functions:

- Sets the attributes of the integrated Endpoint block
- Ties off unneeded ports of the integrated Endpoint block
- Customizes, instantiates, and connects the desired number of GTP/GTX tiles to the integrated Endpoint block (collectively referred to as the GT wrapper)
- Customizes, instantiates, and connects the desired number of block RAMs to the integrated Endpoint block (collectively referred to as the block RAM wrapper)
- Customizes, instantiates, and connects a clock module to provide a complete clocking solution for the integrated Endpoint block, GTP/GTX tile, block RAM, and user application
- Customizes, instantiates, and connects a reset module to provide the necessary integrated Endpoint block resets

All of the above are contained in a single entity, the core or subsystem.

Running the GUI

To run the GUI:

1. Start the CORE Generator tool.

   For help starting and using the CORE Generator tools, see the documentation supplied with Xilinx Integrated Software Environment (ISE), including the CORE Generator Guide at: http://www.xilinx.com/support/software_manuals.htm

   Follow the links on the page based on the ISE software version.

2. Choose FILE → NEW PROJECT.
3. Set the project options:

   Set the *Part* options:

   For *Family*, select Virtex-5.

   Select the desired *Device*, *Package*, and *Speed Grade* options. The GUI does not appear in the taxonomy tree if an unsupported Virtex-5 device is selected.

   Set the *Generator* options:

   For *Design Entry*, select VERILOG or VHDL.

   For *Vendor*, select OTHER.

   Select *OK*, and a project is created.

4. After creating the project, locate the directory containing the GUI. The project appears in the *View By Function* mode as *Standard Bus Interfaces/PCI Express*.
5. Double click on the PCI EXPRESS ENDPOINT BLOCK icon.

   The GUI starts.

Customizing the Subsystem

This section describes how to customize a subsystem by walking through each of the windows and menus in the GUI with descriptions of the options. The descriptions also define the parameters affected by the menu choices and the values set based on these choices. The user can click on the NEXT and BACK buttons to navigate between the menus. A CANCEL button is provided to clear any modified contents in the menu. A FINISH button in each menu allows the user to quickly exit and generate the file without navigating to the last menu (called the "Summary Window").

Endpoint Block Menu

Figure 4-1 shows the first menu that appears when the customize link in the CORE Generator window is clicked. The four option categories are outlined in Table 4-1, page 44.

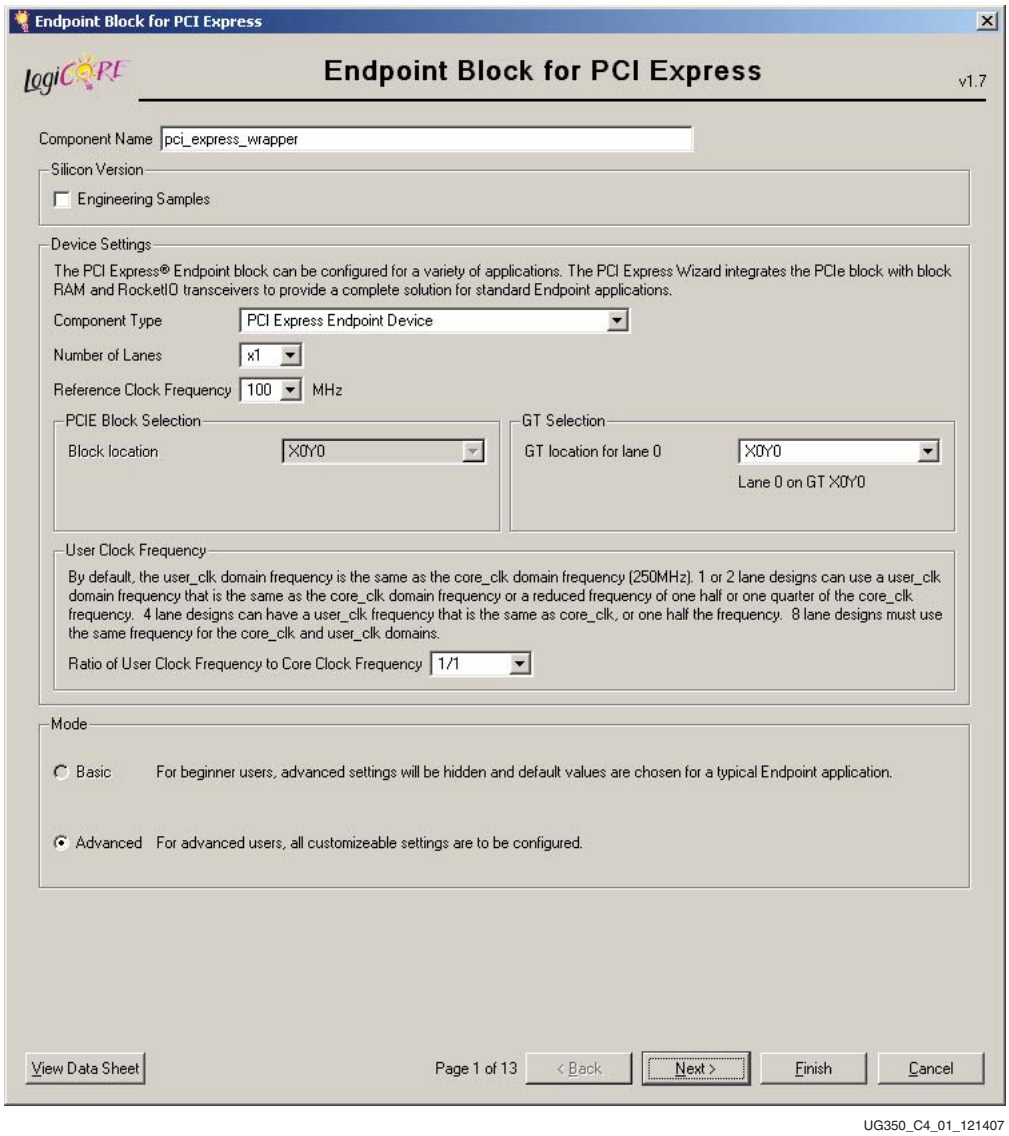


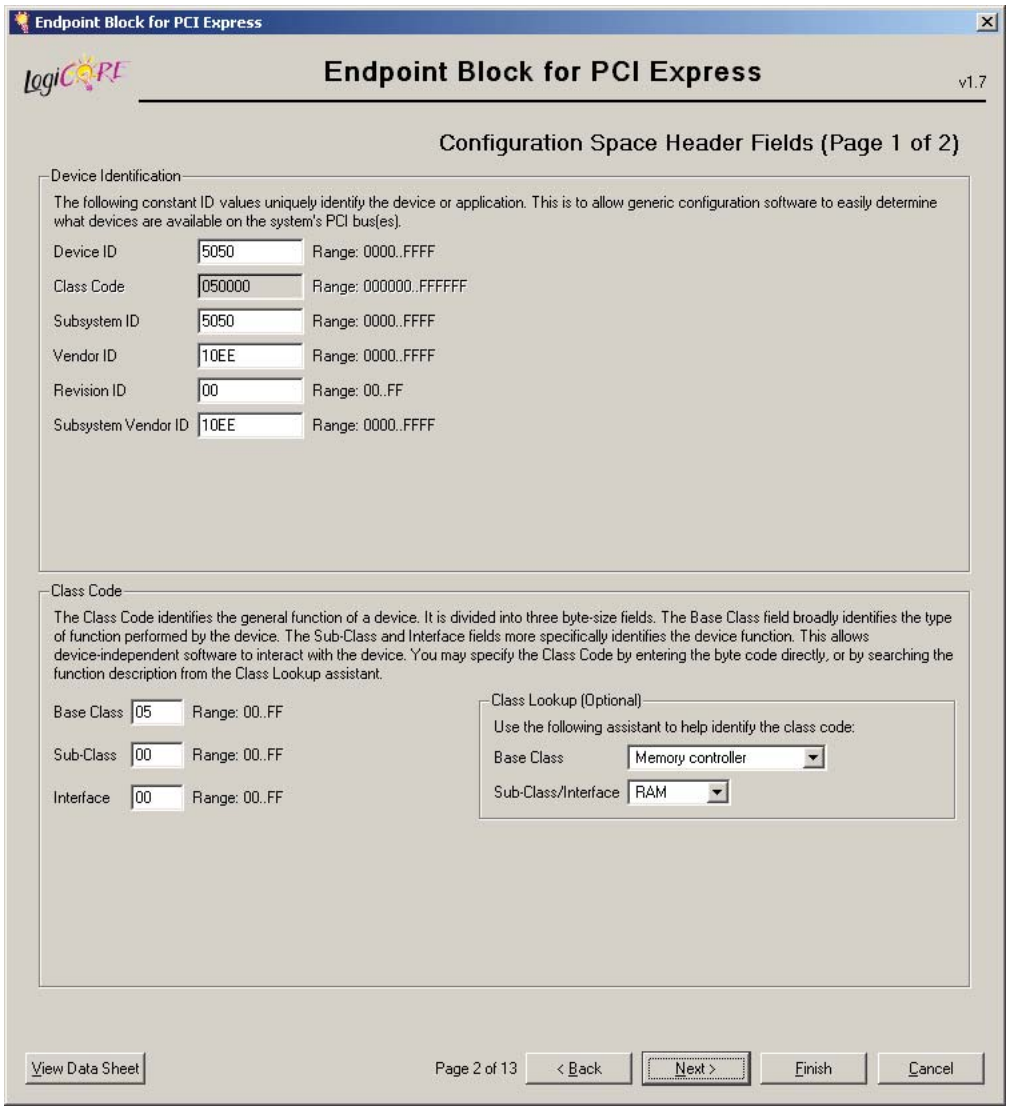
Figure 4-1: **Endpoint Block Menu**

Table 4-1: **Endpoint Block Menu**

Options	Possible Values	Default Value	Description
Component Name	Any ASCII text with no spaces or special characters	pcie_express_wrapper	The design is entered by the user. The name in the box becomes the top-level file name of the generated subsystem. By default, the name is pcie_express_wrapper and the top-level file is `pcie_express_wrapper.v`.
Component Type	PCI Express Endpoint Device or Legacy PCI Express Endpoint Device	PCI Express Endpoint Device	Determines whether the block is configured as a Legacy PCI Express Endpoint Device or a PCI Express Endpoint Device. Selected using the drop-down menu.
Number of Lanes	x1, x2, x4, x8	x1	Determines the number of lanes in the subsystem and the link width. Selected using the drop-down menu.
Clock Ratio	1 lane: 1/1, 1/2, 1/4 2 lanes: 1/1, 1/2, 1/4 4 lanes: 1/1, 1/2 8 lanes: 1/1	1/1	Determines the ratio of the user clock frequency to the core clock frequency. The core clock frequency must be 250 MHz, but the user clock frequency can be reduced for designs with 1, 2, or 4 lanes. Selected using the drop-down menu.
Mode	Advanced	Advanced	Only supported mode is Advanced.
Reference Clock Frequency	100 MHz, 125 MHz, 250 MHz	100 MHz	Determines the frequency of the input reference clock to the FPGA.

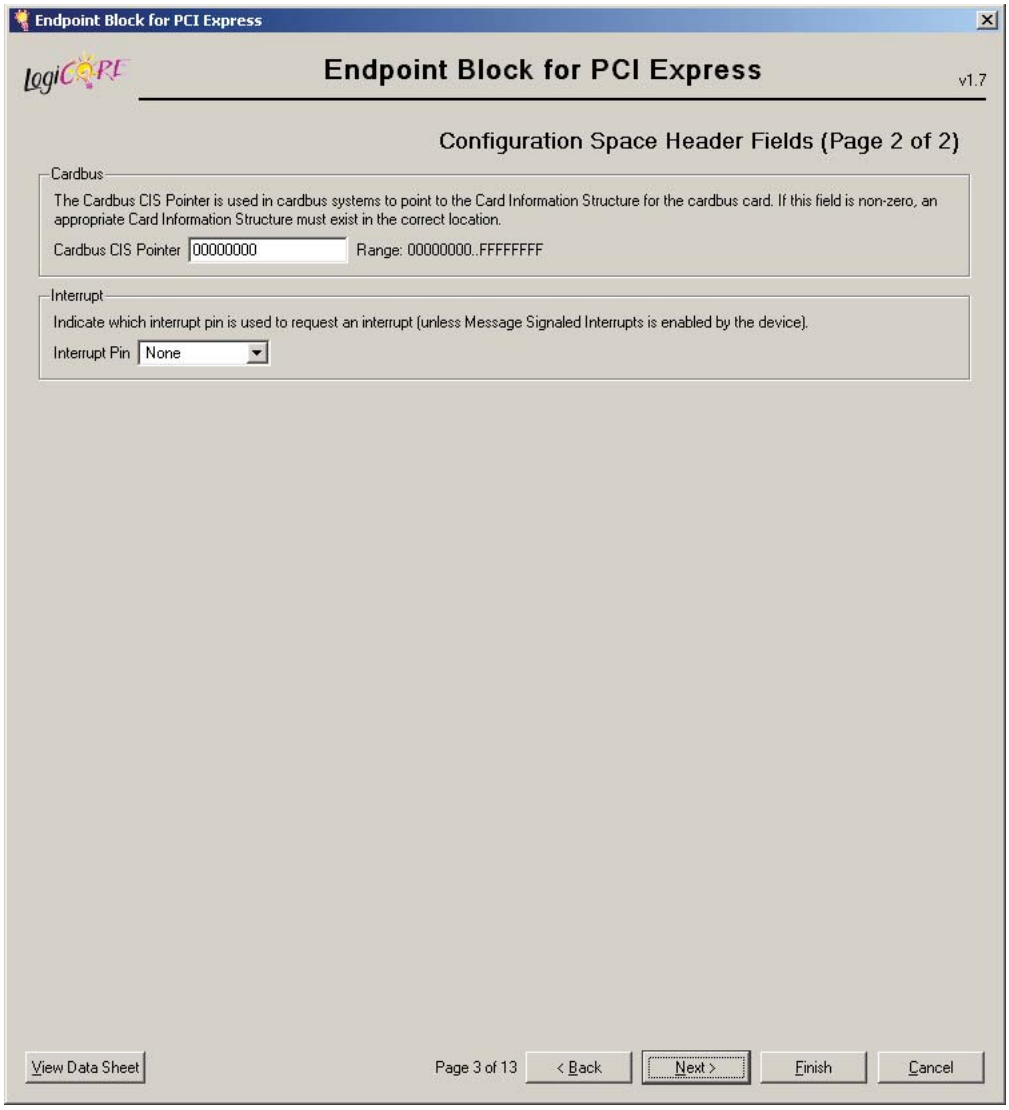
Configuration Space Header Fields Menu

Figure 4-2 and Figure 4-3, page 46 show the Configuration Space Header Fields menu. Table 4-2, page 47 lists the menu options.



UG350_C4_02_121407

Figure 4-2: **Configuration Space Header Fields Menu (Page 1 of 2)**



UG350_C4_03_121407

Figure 4-3: **Configuration Space Header Fields Menu (Page 2 of 2)**

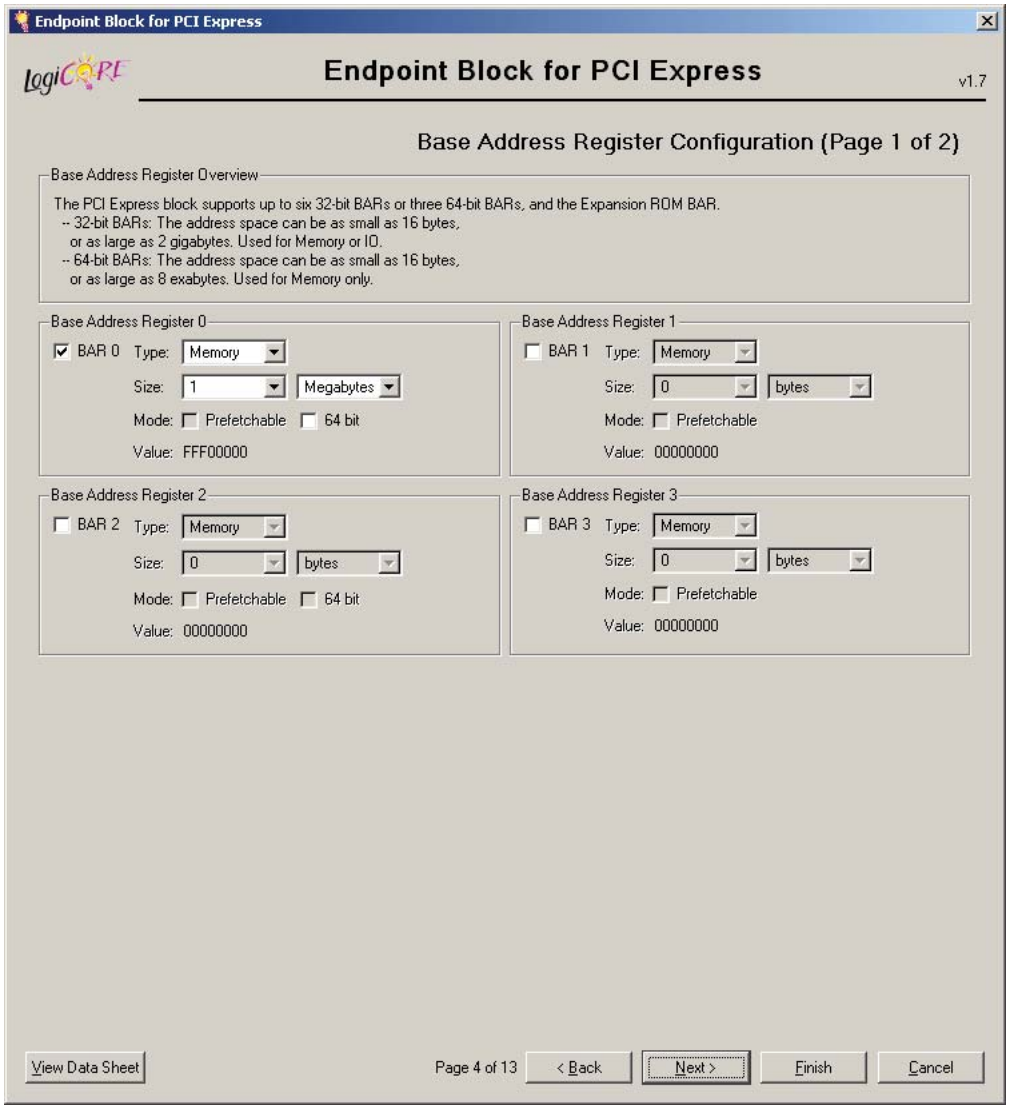
Table 4-2 lists the menu options for the Configuration Space Header Fields.

Table 4-2: **Configuration Space Header Fields Menu**

Options	Possible Values	Default Value	Description
Device ID	0000 .. FFFF	5050	Determines the device ID of the integrated Endpoint block. The user enters the value in the box.
Vendor ID	0000 .. FFFF	10EE	Determines the vendor ID of the integrated Endpoint block. The user enters the value in the box.
Class Code	000000 .. FFFFFF	058000	Determines the class code of the integrated Endpoint block. The user enters the value in the box. The class code can be selected by directly entering the code in the provided boxes OR using the class code look-up menu and selecting the appropriate description of the device. A box in the Device ID section reserved for class code displays the values based on the selections in the class code section. The box is not modifiable.
Revision ID	00 .. FF	00	Determines the revision ID of the integrated Endpoint block. The user enters the value in the box.
Subsystem ID	0000 .. FFFF	5050	Determines the subsystem ID of the integrated Endpoint block. The user enters the value in the box.
Subsystem Vendor ID	0000 .. FFFF	10EE	Determines the subsystem vendor ID of the v block. The user enters the value in the box.
Base Class	00 .. FF	05	Memory controller
Sub Class	00 .. FF	80	Other memory controller
Interface	00 .. FF	00	
Cardbus CIS Pointer	00000000 .. FFFFFFFF	00000000	Determines the cardbus CIS pointer of the integrated Endpoint block. Used in cardbus systems to point to the card information structure for the cardbus card.
Interrupt Pin	None, INTA	None	Determines the mapping for Legacy Interrupt Messages. When one of the pins is used, the MSI capability is also enabled.

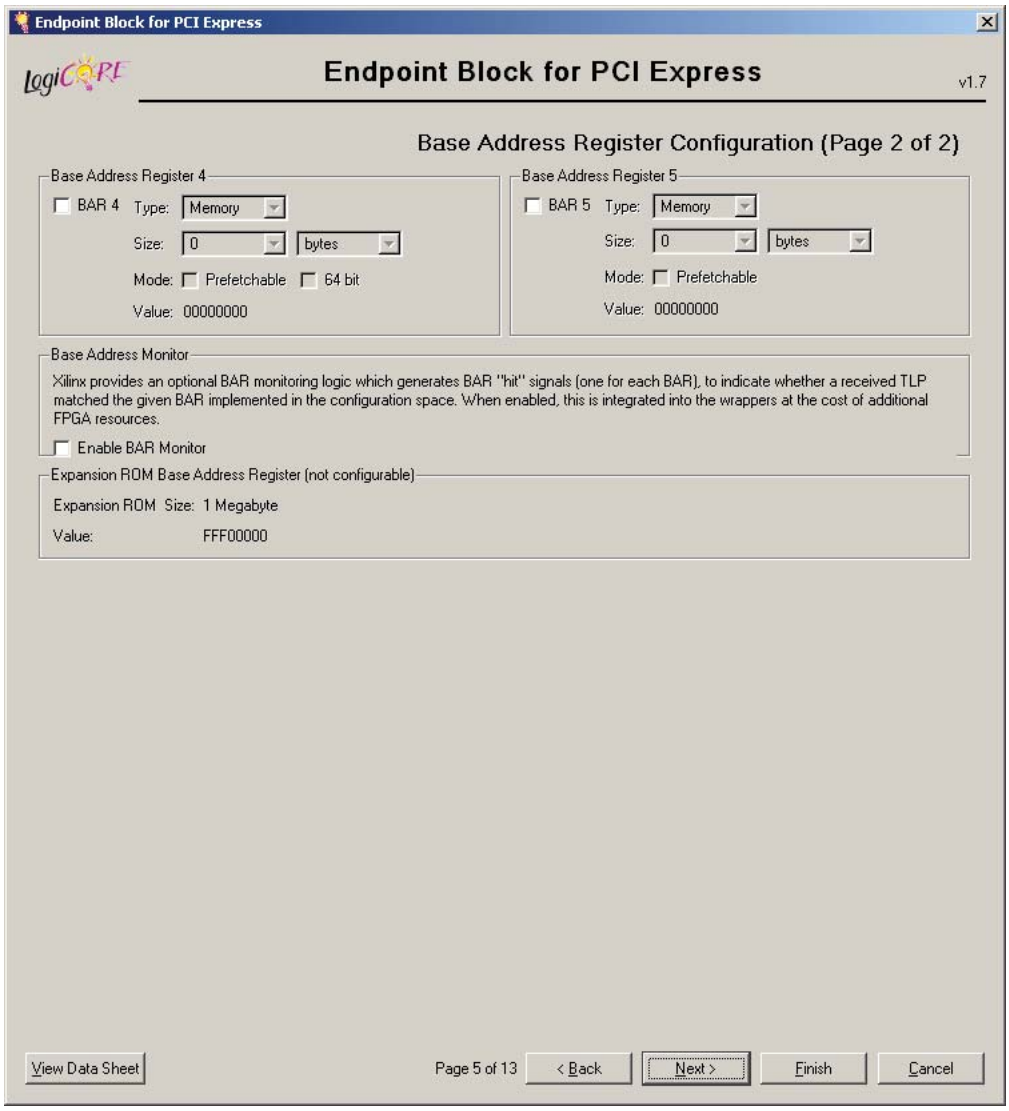
Base Address Register Configuration Menu

The base address register configuration menu (Figure 4-4 and Figure 4-5, page 49) is used to define the Base Address Registers (BARs) for the integrated Endpoint block. Integrated Endpoint blocks support up to six 32-bit BARs or three 64-bit BARs and one expansion ROM BAR. 64-bit BARs can be used for memory only, whereas 32-bit BARs can be used for both memory and I/O. Table 4-3, page 50 outlines the BAR0 through BAR6 menu options.



UG350_C4_04_121407

Figure 4-4: **Base Address Register Configuration Menu (Page 1 of 2)**



UG350_C4_05_121407

Figure 4-5: **Base Address Register Configuration Menu (Page 2 of 2)**

Table 4-3: **Base Address Register Configuration Menu**

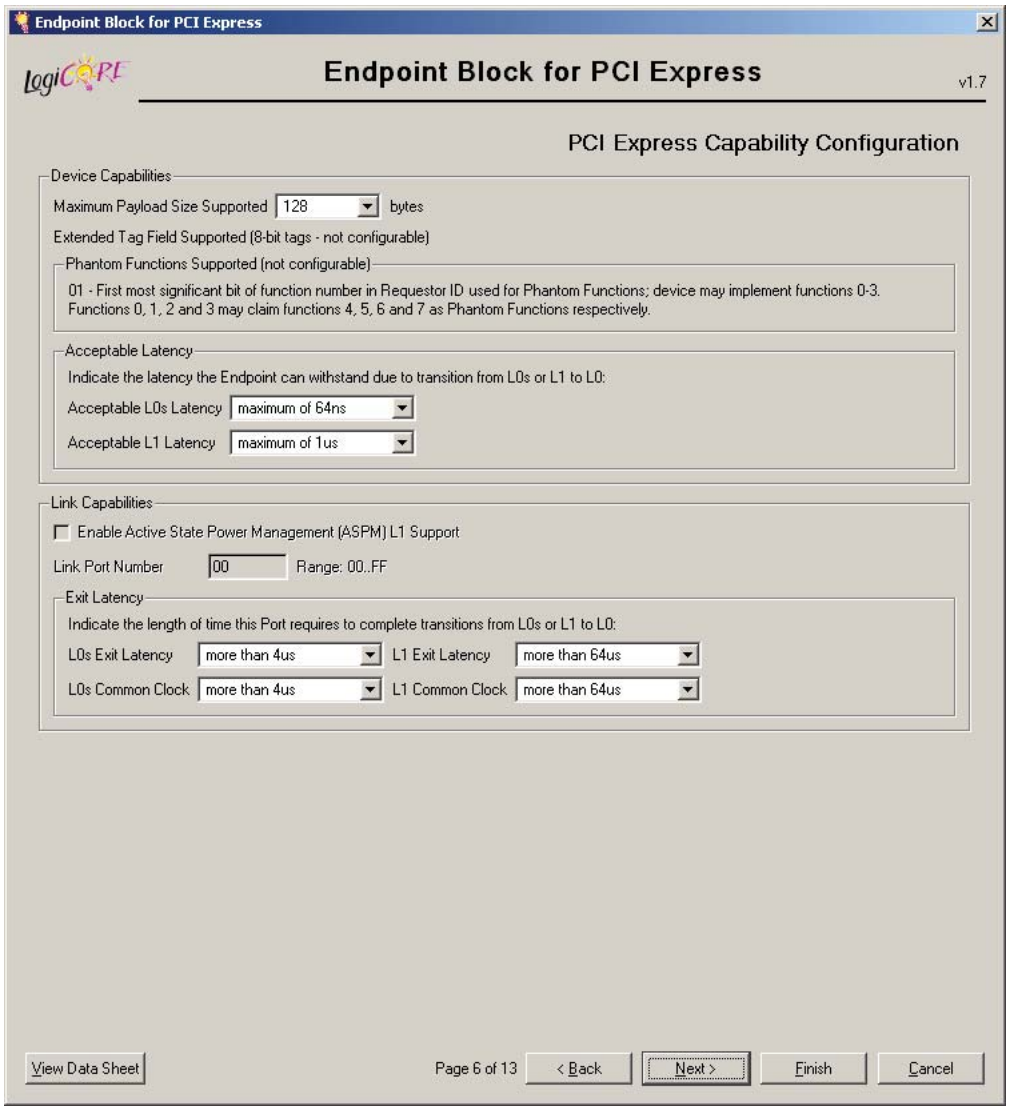
Options[1]	Possible Values	Default Value	Description[1]
BAR*x*	Checked or unchecked box	BAR0: Checked BAR1 - BAR5: Unchecked	Enables BAR*x*. The user checks the box to enable.
BAR Type	Memory, I/O	Memory	Determines if the BAR*x* type is Memory or I/O. Selected using the drop-down menu.
BAR Size	A 32-bit BAR can vary in size from 16 B to 2 GB. A 64-bit BAR can vary in size from 16 B to 8 EB[2].	16 bytes	Determines the size of BAR*x*. Selected using the drop-down menu.
BAR*x* Prefetchable	Checked or unchecked	Unchecked	Determines if BAR*x* is prefetchable. The user checks the box to enable. I/O BARs are not prefetchable.
BAR*x* 64 Bit	Checked or unchecked	Unchecked	If BAR*x* is set to 64 bits, then BAR*x* + 1 is automatically consumed and cannot be enabled. Only BAR0, BAR2, and BAR4 can be 64 bits.
Expansion ROM	1 MB	1 MB	The expansion ROM size is fixed to 1 MB and cannot be configured.
Enable BAR Monitor	Checked or Unchecked	Checked	Checking this option integrates BAR Monitoring logic into the wrappers. The logic generates BAR *hit* signals to indicate if a received TLP matched the given BAR implemented.

Notes:

1. The *x* in the BAR descriptions refers to the BAR number; *x* can take values from 0 to 5.
2. An EB is an exabyte, which equals 2^{60} bits.

PCI Express Capability Configuration Menu

Values related to the device capabilities and link capabilities registers of the PCI Express capability structure are set using the capability configuration menu (Figure 4-6). Table 4-4 outlines the options in this menu.



UG350_C4_06_121407

Figure 4-6: **PCIe Capability Configuration Menu**

Table 4-4: **PCIe Capability Configuration Menu**

Options	Possible Values	Default Value	Description
Maximum Payload Size	128, 256, 512, 1024, 2048, or 4096 bytes	128	Determines the maximum payload size for Transaction Layer Packets supported by the device.
L0s Acceptable Latency	Maximum of 64 ns, Maximum of 128 ns, Maximum of 256 ns, Maximum of 512 ns, Maximum of 1 µs, Maximum of 2 µs, Maximum of 4 µs, or No limit	Maximum of 64 ns	Determines the maximum latency that the integrated Endpoint block can withstand while transitioning from L0s to L0.
L1 Acceptable Latency	Maximum of 1 µs, Maximum of 2 µs, Maximum of 4 µs, Maximum of 8 µs, Maximum of 16 µs, Maximum of 32 µs, Maximum of 64 µs, or No limit	Maximum of 1 µs	Determines the maximum latency that the integrated Endpoint block can withstand while transitioning from L1 to L0.
ASPM L1 Support	Checked or unchecked	Unchecked	This parameter is not configurable. The integrated Endpoint block is configured to support L0s but not the L1 ASPM state.
Link Port Number	`00 .. FF`	`00`	Determines the link port number. *Not supported.*
L0s Exit Latency	Possible Values: < 64 ns 64 ns – 128 ns 128 ns – 256 ns 256 ns – 512 ns 512 ns – 1 µs 1 µs – 2 µs 2 µs – 4 µs > 4 µs	> 4 µs	Determines the L0s to L0 state transition latency reported by the integrated Endpoint block when the two ends of the link are using different clock sources. Whether or not the system uses a common clock is specified using the Advanced Physical Layer Settings menu. Selected using the drop-down menu.
L0s Exit Latency (Common Clock)	Possible Values: < 64 ns 64 ns – 128 ns 128 ns – 256 ns 256 ns – 512 ns 512 ns – 1 µs 1 µs – 2 µs 2 µs – 4 µs > 4 µs	> 4 µs	Determines the L0s to L0 state transition latency reported by the integrated Endpoint block when the two ends of the link are using the same clock source. Selected using the drop-down menu.

Table 4-4: **PCIe Capability Configuration Menu *(Continued)***

Options	Possible Values	Default Value	Description
L1 Exit Latency	Possible Values: < 1 µs 1 µ s – 2 µs 2 µs – 4 µs 4 µs – 8 µs 8 µs – 16 µs 16 µs – 32 µs 32 µs – 64 µs > 64 µs	> 64 µs	Determines the L1 to L0 state transition latency reported by the integrated Endpoint block when the two ends of the link are using different clock sources. Whether or not the system uses a common clock is specified on the Advanced Physical Layer Settings menu. Selected using the drop-down menu.
L1 Exit Latency (Common Clock)	Possible Values: < 1 µs 1 µ s – 2 µs 2 µs – 4 µs 4 µs – 8 µs 8 µs – 16 µs 16 µs – 32 µs 32 µs – 64 µs > 64 µs	> 64 µs	Determines the L1 to L0 state transition latency reported by the integrated Endpoint block when the two ends of the link are using the same clock source. Whether or not the system uses a common clock is specified on the Advanced Physical Layer Settings menu.

Notes:

1. When the GTP/GTX transceiver exits from Rx.L0s, six symbol times of valid data following the SKP symbol in the FTS-FTS-COM-SKP channel bonding sequence will be ignored by the block. This is due to a delay in the assertion of the RXCHANISALIGNED signal from the GTP/GTX. If the packets are replayed by the upstream port and correctly ACKed by the integrated block, then link will stay up. However, there is a chance that if the original missed data was an ACK, that the upstream component sends the ACK, goes back to L0s, wakes up to send the ACK then returns to L0s. If this repeats continuously, then this eventually triggers a link retrain. However, no data is lost and the link eventually recovers, causing minimal to no impact on safe operation.

Extended Capability Configuration Menu

Figure 4-7 and Figure 4-8, page 55 show the Extended Capability Configuration menu. Table 4-5, page 56 describes the menu options.

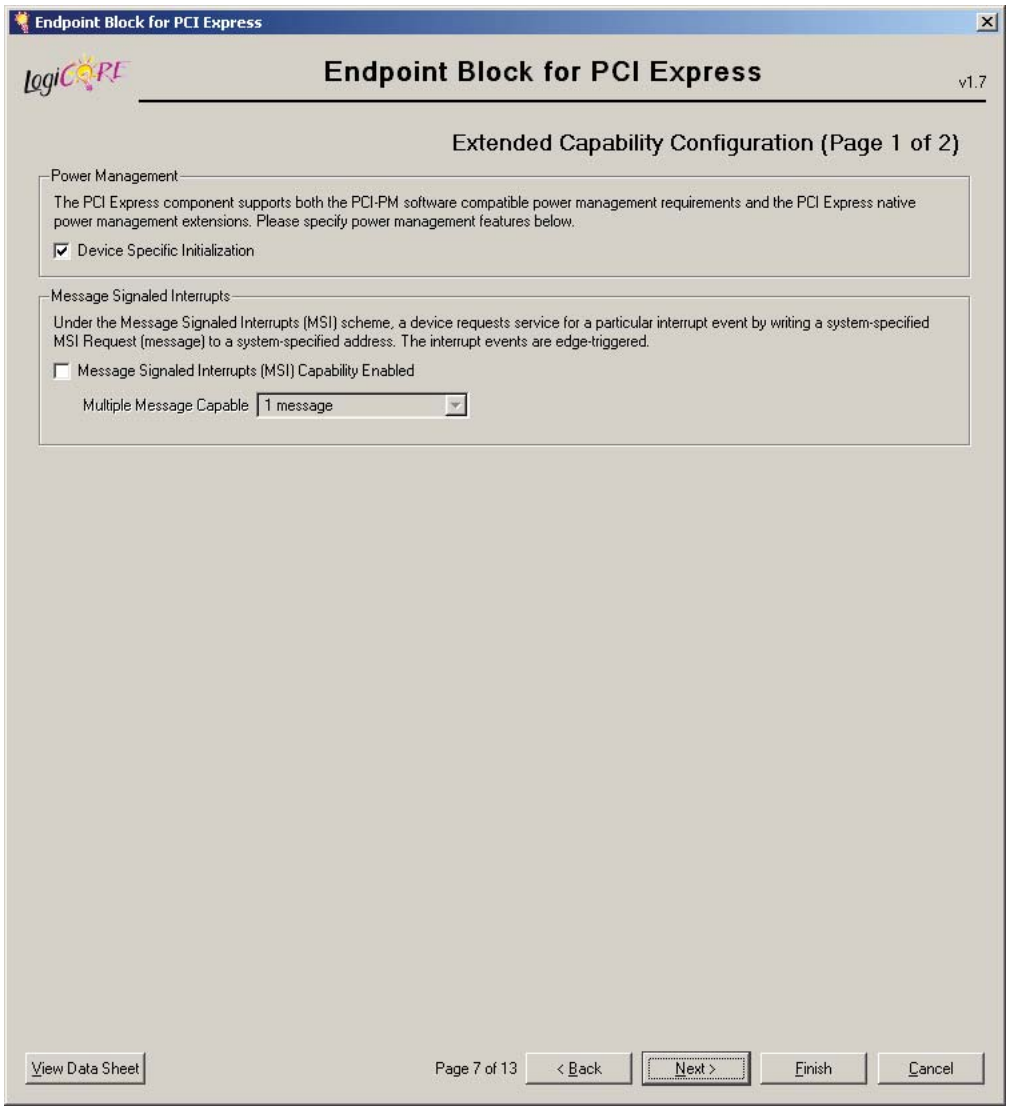


Figure 4-7: **Extended Capability Configuration Menu (Page 1 of 2)**

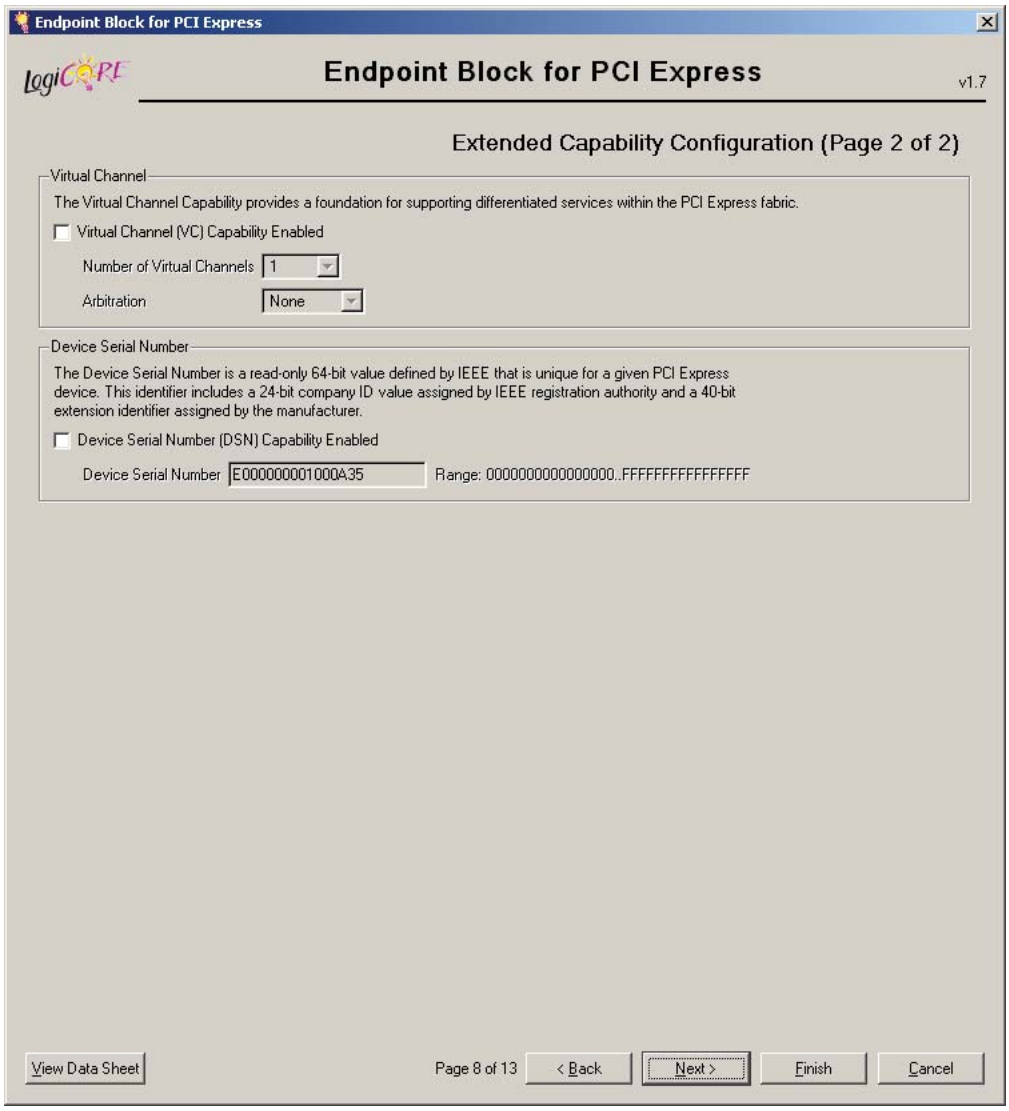


Figure 4-8: **Extended Capability Configuration Menu (Page 2 of 2)**

Table 4-5: **Extended Capability Configuration Menu**

Options	Possible Values	Default Value	Description
Device Specific Initialization	Checked or unchecked	Checked	Determines whether the DSI bit of the Power Management Capabilities Register is set.
Message Signaled Interrupts (MSI)	MSI enable: checked or unchecked 1, 2, or 4 messages for Multiple Message Capability	MSI enable: unchecked Multiple Message Capability: 1 message	Enables MSI and chooses the number of messages in the Multiple Message Capable drop down menu. When an interrupt pin is selected, by default the MSI capability is enabled.
VC Capability Enabled	Checked or unchecked	Unchecked	Determines whether the VC capability is enabled. The VC capability must be enabled to use two VCs or traffic classes other than TC0, even if only one VC is enabled.
Number of Virtual Channels	1 or 2	1	Enables and chooses one or two virtual channels. When two VCs are selected, an arbitration scheme can be selected.
Arbitration	Possible Values: • 1 VC selected: None • 2 VCs selected: Round Robin, Weighted Round Robin, or Strict Priority	Default Values: • 1 VC selected: None • 2 VCs selected: Round Robin	
Device Serial Number (DSN) Capability	Check or uncheck DSN enable box DSN value range 0000000000000000 .. FFFFFFFFFFFFFFFF	DSN enable: unchecked DSN: E000000001000A35	Enables the DSN capability and enters a number. The first 24 bits are preassigned by an IEEE registration authority. The remaining bits are chosen by the user.

Advanced Physical Layer Settings Menu

The Advanced Physical Layer Settings menu (Figure 4-9) sets the Physical Layer settings such as scrambling, reset domain, number of FTS in link training, and slot clock configuration. Table 4-6, page 58 describes the menu options.

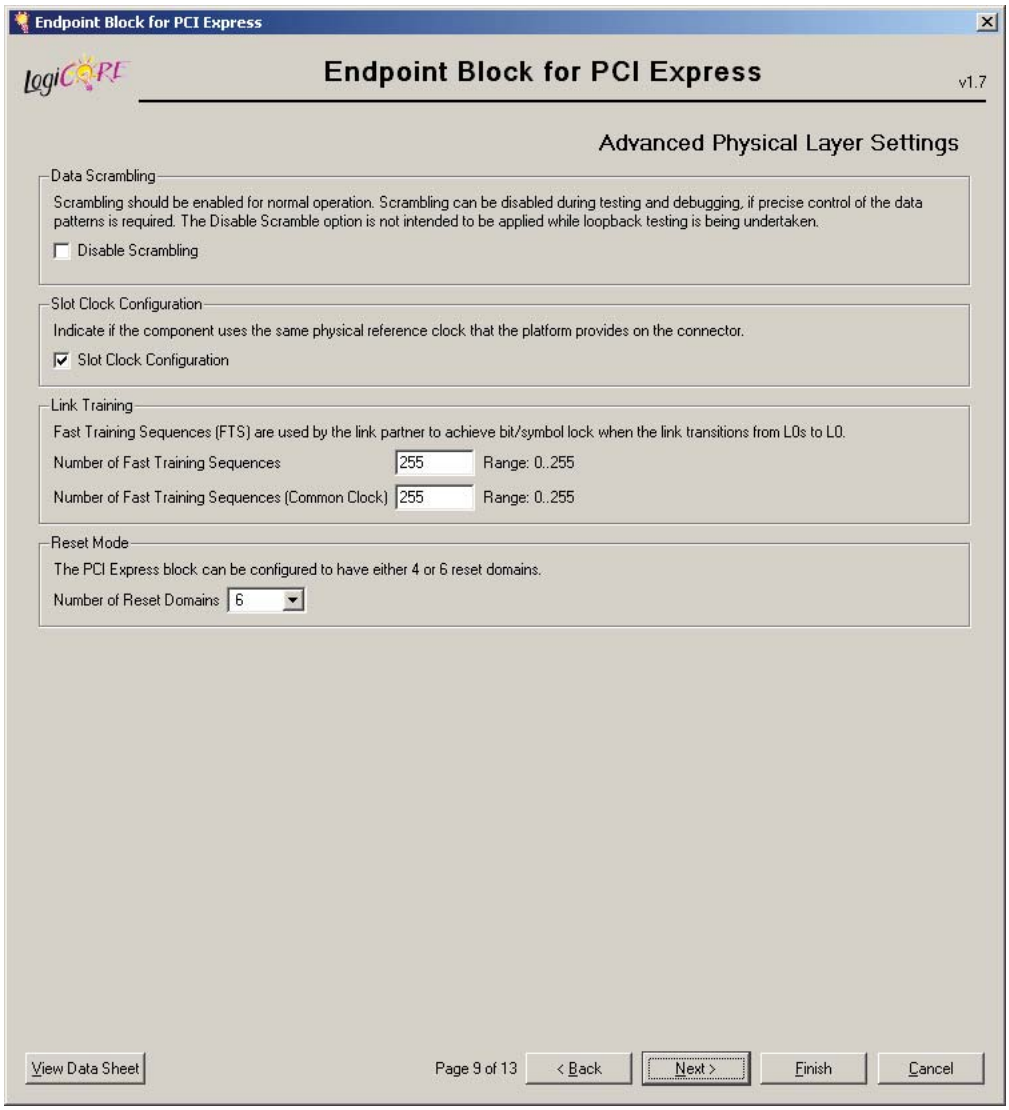


Figure 4-9: **Advanced Physical Layer Settings Menu**

Table 4-6: **Advanced Physical Layer Settings Menu**

Options	Possible Values	Default Value	Description
Data Scrambling	Checked or unchecked	Unchecked	Disable the scrambling of data in the Physical Layer by checking the Disable Scrambling Box.
Slot Clock Configuration	Checked or unchecked	Unchecked	Check this box when the device is using the same reference clock provided on the platform connector.
Fast Training Sequences (FTS)	1 .. 255	255	Enter the number of FTS required by the receiver to achieve bit/symbol lock. Specify separate numbers whether or not the two sides of the link use the same clock. Each scenario is available in the box. The assigned value is entered in the box.
Number of Reset Domains	4 or 6	4	The number of reset domains used inside the integrated Endpoint block. When four reset domains are chosen, the RESETMODE attribute is set to FALSE. When six reset domains are chosen, the RESETMODE attribute is set to TRUE. More information is available in "Clock and Reset Interface," page 26.

Buffer Setup Menu

The Buffer Setup Menu (Figure 4-10 and Figure 4-11, page 60) is used to configure the size of the three buffers: retry buffer, transmit buffer, and receive buffer. The transmit and receive buffers are further divided based on virtual channels and packet types.

Certain rules govern the sizes of the three buffers. The final size is determined by the core and depends on the user input, maximum payload size (set in the "PCI Express Capability Configuration Menu"), and the maximum number of block RAMs that the integrated Endpoint block can support. In addition, the total number of block RAMs per buffer is a power of two. Hence, rounding up to the nearest power of two is performed by the GUI to determine the final number of block RAMs. This menu also lists the initial flow control credits available for all the buffers. The initial flow control credits cannot be directly modified with the GUI. Table 4-7, page 61 describes the menu options.

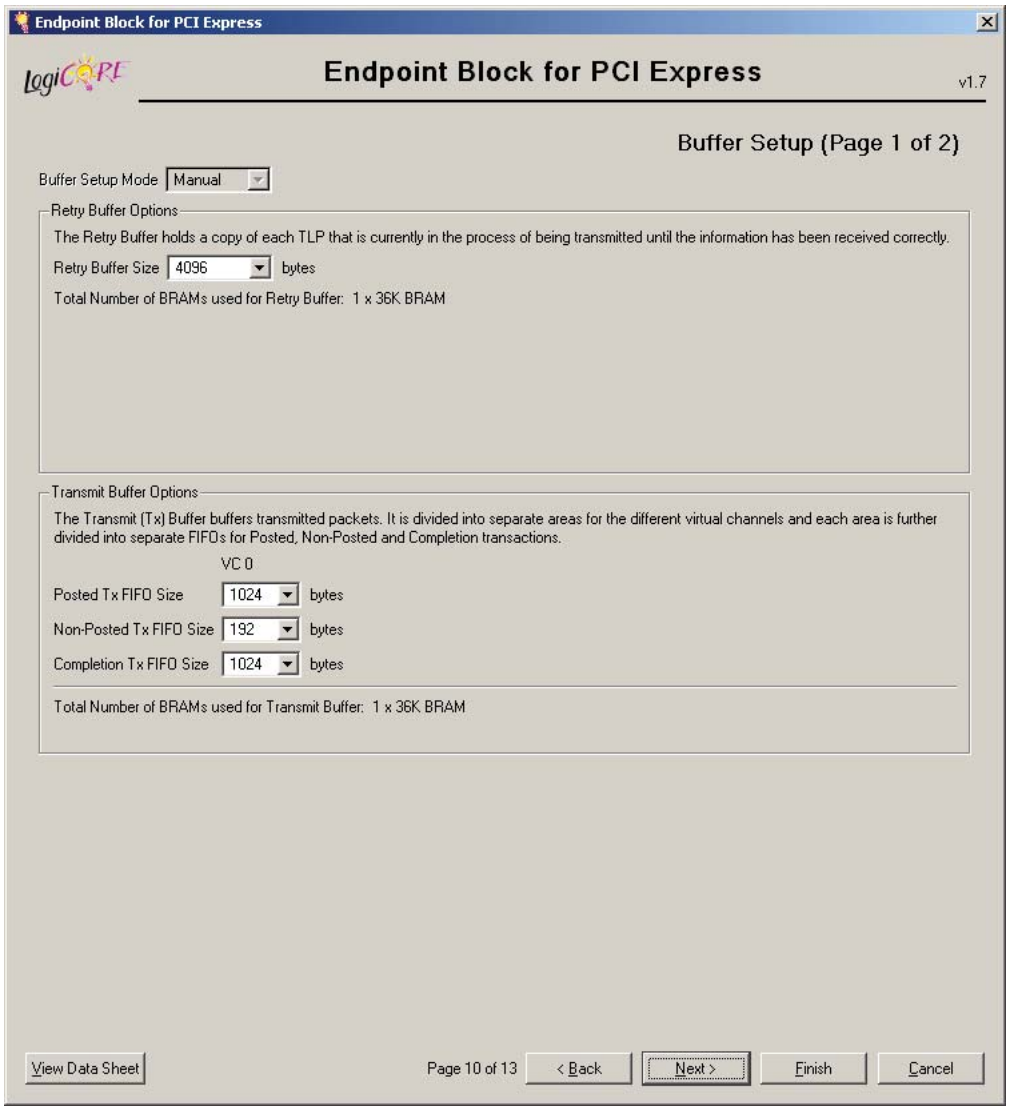


Figure 4-10: **Buffer Setup Menu (Page 1 of 2)**

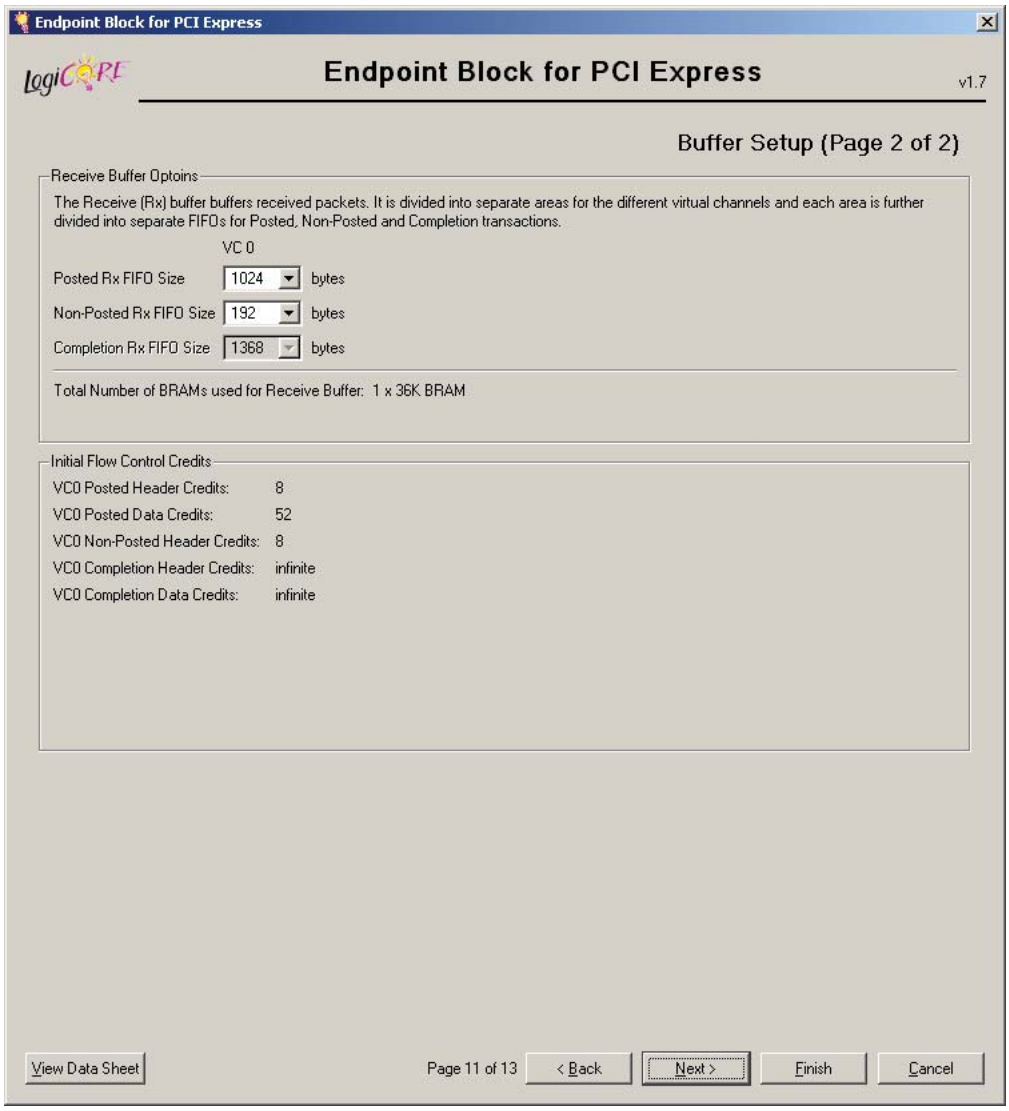


Figure 4-11: **Buffer Setup Menu (Page 2 of 2)**

Table 4-7: **Buffer Setup Menu**

Options	Possible Values	Default Value	Description
Retry Buffer Size	4096, 8192, 16284, or 32768	For maximum payload sizes of 256 .. 1024 bytes: 4096 For maximum payload size of 2048 bytes: 8192 For maximum payload size of 4096 bytes: 16384	The size is chosen from the drop-down menu. As soon as a selection is made, the number of block RAMs to be used is displayed to the right of the drop-down menu. A subset of the values depends on the value of maximum payload size selected in "PCI Express Capability Configuration Menu."
Transmit Buffer Size	Posted Buffers: 320 .. 32768 Non-Posted: 192 Completion: 256-32768	Posted Buffers: 1024 Non-Posted: 192 Completion: 1024	Six drop-down menus set the two virtual channels and the three buffers per virtual channel: posted, non-posted, and completion. The range of possible values is reduced depending on the maximum payload size and other FIFO sizes chosen.
Receive Buffer Size	Posted Buffers: 320 .. 32768 Non-Posted: 192 Completion: 1368 .. 18648	Posted Buffers: 1024 Non-Posted: 192 Completion: 1368	The range of possible values is reduced depending on the maximum payload size and other FIFO sizes chosen.

Notes:

1. The number of block RAMs allocated for the TX and RX buffers always snap to a power of 2. This may result in unallocated buffer space. It is recommended that the user resize the individual FIFOs to take advantage of the unallocated space whenever possible. Due to limitations on the maximum possible values of FIFO sizes based on the MAXPAYLOAD value chosen, it might not be possible to use the entire unallocated space available.

Summary Window

The Summary Window (Figure 4-12 and Figure 4-13, page 63) provides a snapshot of the key customization settings and the Configuration Space register view including a subset of Extended Capability registers. The Summary Window is used to review the desired customization.

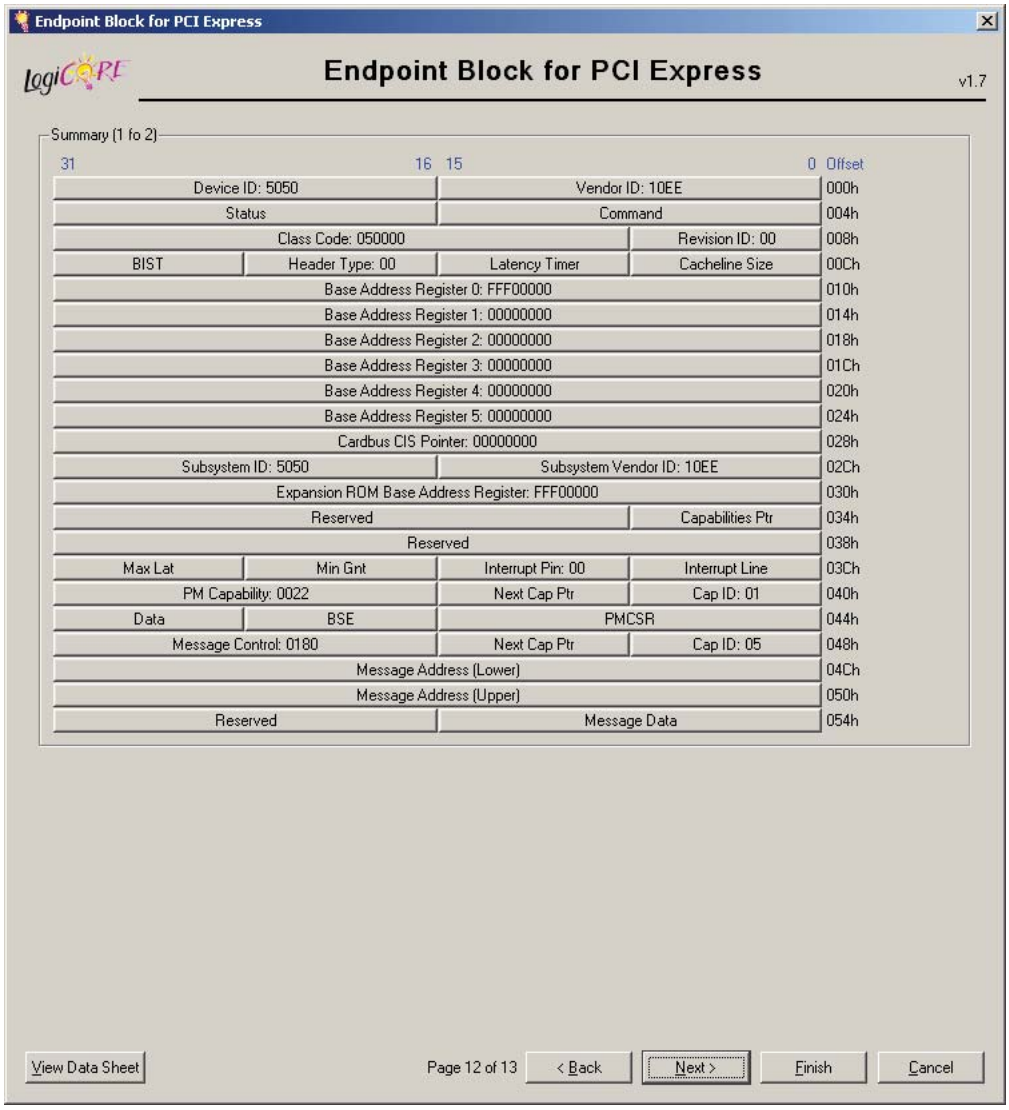
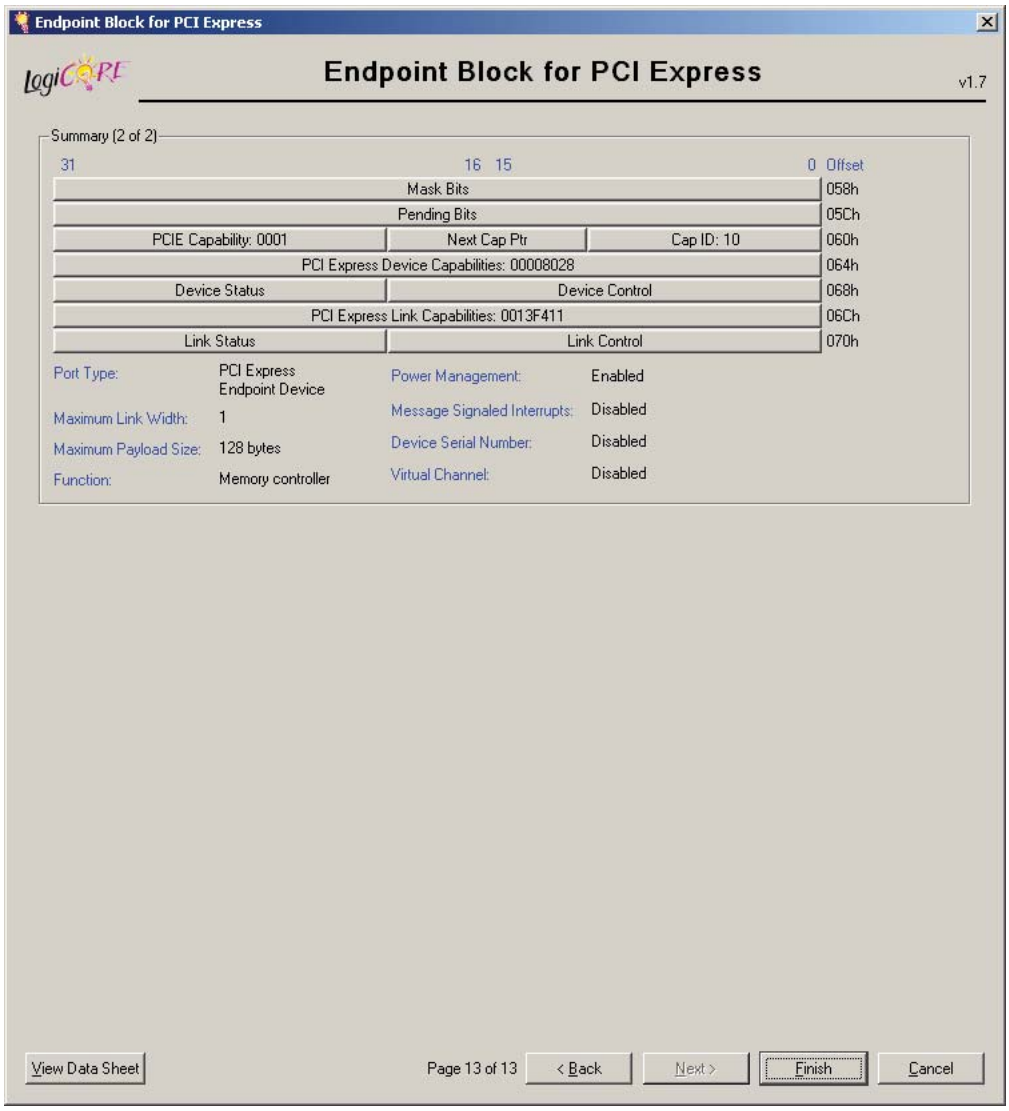


Figure 4-12: **Summary Window (Page 1 of 2)**



UG350_C4_13_121407

Figure 4-13: **Summary Window (Page 2 of 2)**

List of Parameters

Table 4-8 lists all the parameters that are exposed at the top-level file and are generated using the GUI. Some parameters are set automatically by the GUI. The few parameters which cannot be set by the GUI can be set manually if needed.

Table 4-8: **Top-Level Parameters**

Name	Possible Values	Default Value	Set by GUI	Description
G_SIM	0,1	0	No	Not supported
G_USER_RESETS	0,1	0	No	Can be used to bypass the custom reset logic provided
G_USE_DCM	0,1	0	No	Not supported
BARMONITORENABLE	0,1	0	Yes	Generates additional logic for BAR Monitoring
REFCLKFREQ	100,125,250	100	Yes	Reference clock frequency
CLKRATIO	1,2,4	1	Yes	Ratio of the frequency between the core_clk and user_clk
CLKDIVIDED	TRUE, FALSE	FALSE	Yes	Indicates if the user_clk is a derivative of the core_clk by division
MAXPAYLOADSIZE	128 - 4096 bytes	128	Yes	Maximum payload size

CORE Generator Output

The GUI generates a list of files after the user clicks on the Finish button. The key component in the generated files is a set of wrappers containing the customization settings for the integrated Endpoint block, GTP/GTX tile, block RAM, and clock and reset modules. The core is described in detail in this section, along with the directory structure and file organization.

The CORE Generator automatically generates wrappers for customizing and connecting GTP/GTX tiles and block RAM. Appropriate clock and reset modules are embedded in the top wrapper. These modules allow the user to focus on interfacing with the application, saving valuable time. Figure 4-14 and Figure 4-15, page 66 illustrate the benefits of using the wrappers. The effort in connecting the integrated Endpoint block to other blocks in the device is greatly reduced with the use of the core.

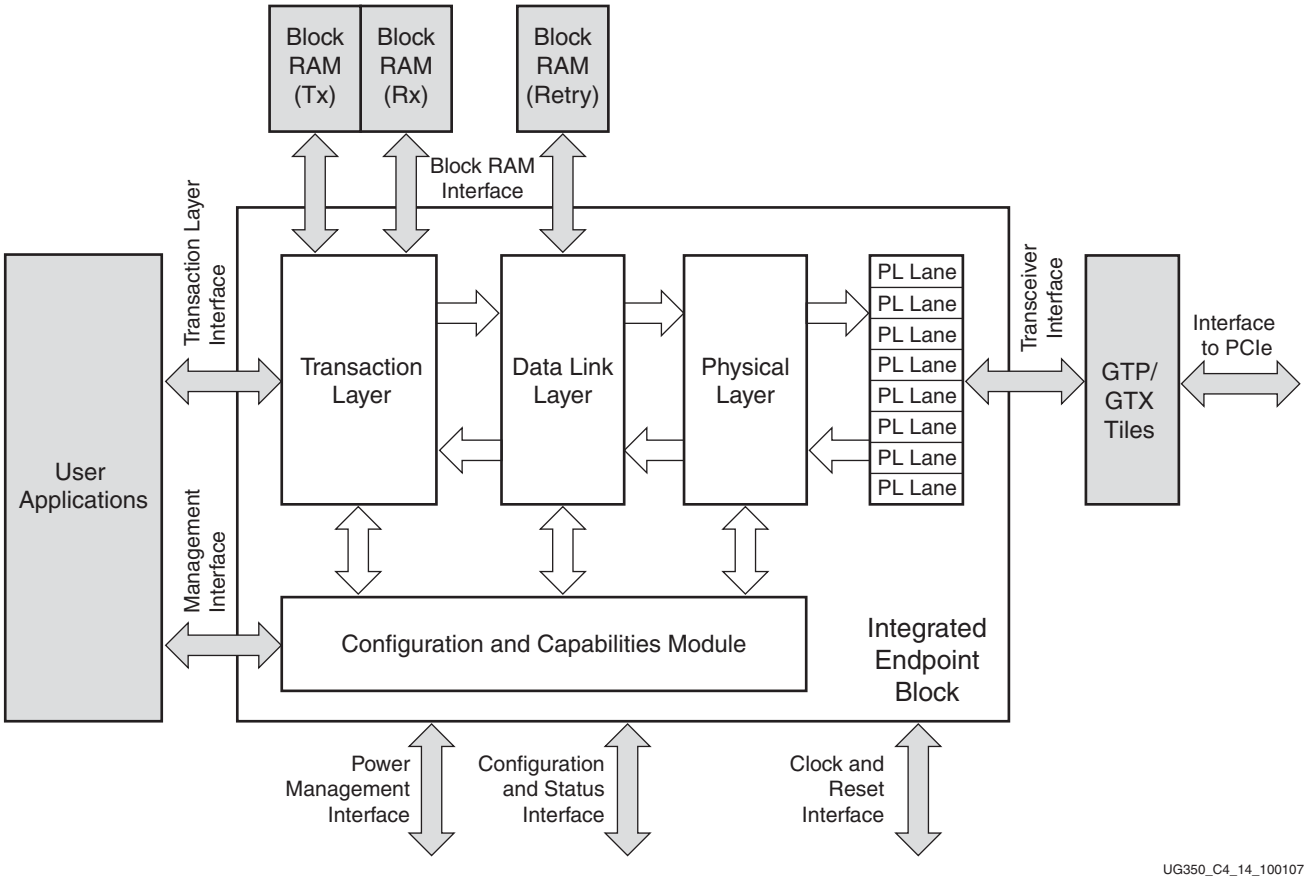


Figure 4-14: **Integrated Endpoint Block Connections/Customization Before Using the Core**

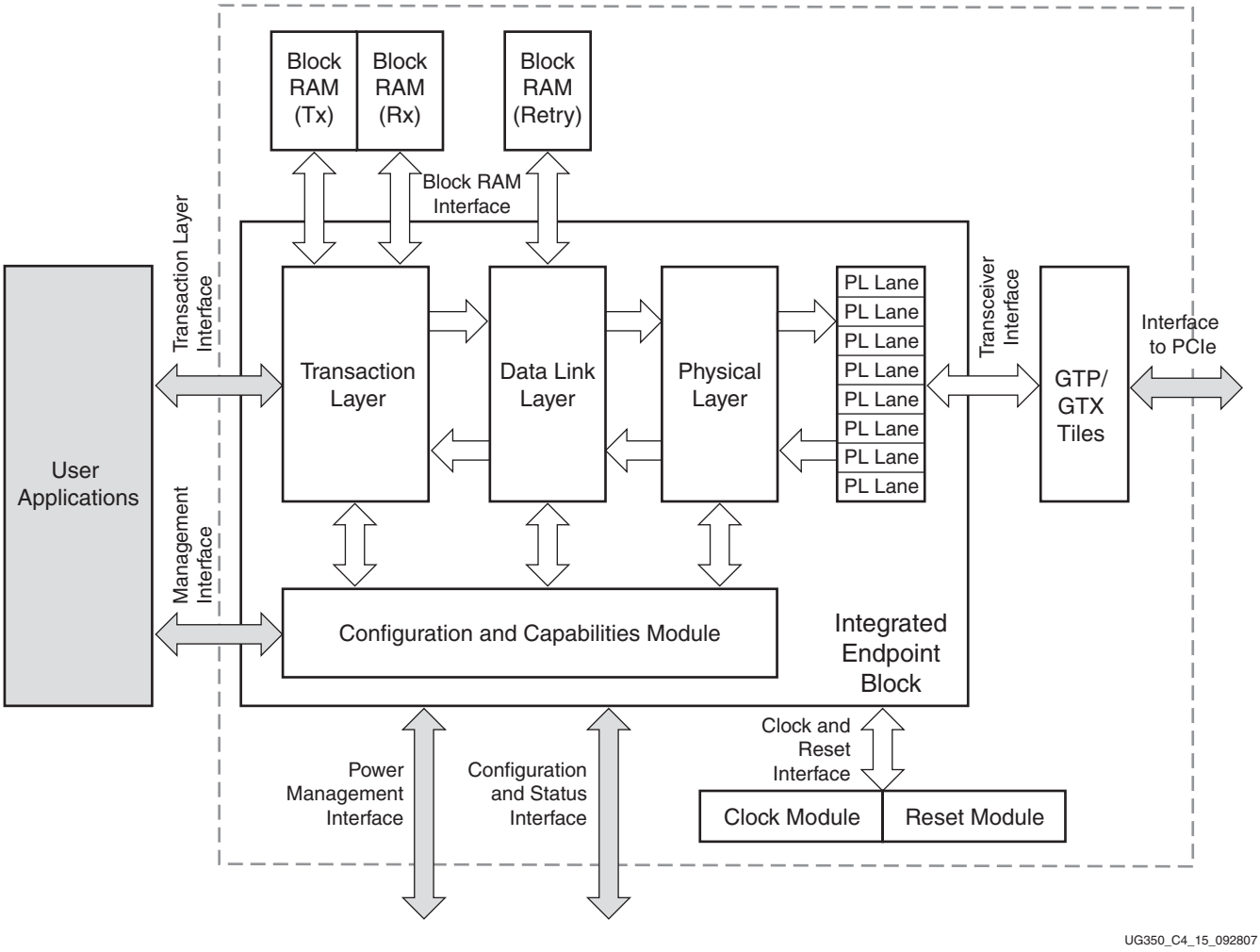


Figure 4-15: **Integrated Endpoint Block Connections/Customization After Using the Core**

Top Wrapper

The top wrapper (Figure 4-16) contains all the individual wrappers and modules. It serves as the top-level file. All ports relevant to the user and need to interface to the application are exposed at the top level. The rest of the ports are tied off to preset values in case of inputs and left unconnected in case of outputs. The user is responsible for the following:

- Connect a reference clock to the clock input of the core
- Connect the system-level reset signal to the reset input of the core
- Connect the serial pins of the GTP/GTX tile to the system-level serial pins
- Connect the user application to the Transaction Layer interface of the core
- Connect the management interface, power management interface, and/or configuration and status interface of the core to user logic

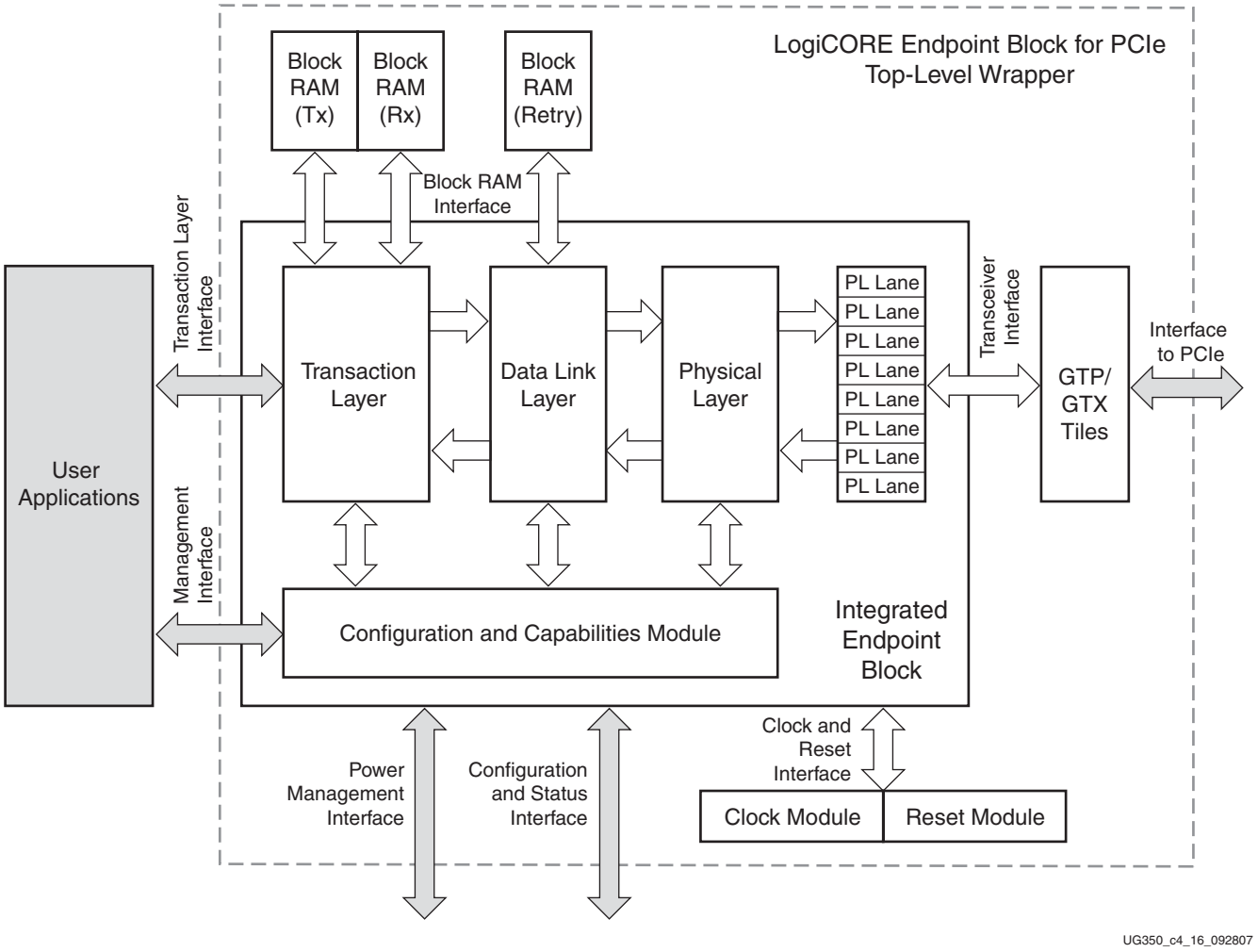


Figure 4-16: **LogiCORE Endpoint Block for PCIe Top-Level Wrapper**

GT Wrapper

The GT wrapper (Figure 4-17) connects the transceiver interface of integrated Endpoint block to the GTP/GTX tiles. Based on the user selection in the GUI, the wrapper instantiates the correct number of GTP/GTX tiles with the appropriate attribute settings to configure the GTP/GTX tiles. Multilane designs are automatically configured for channel bonding.

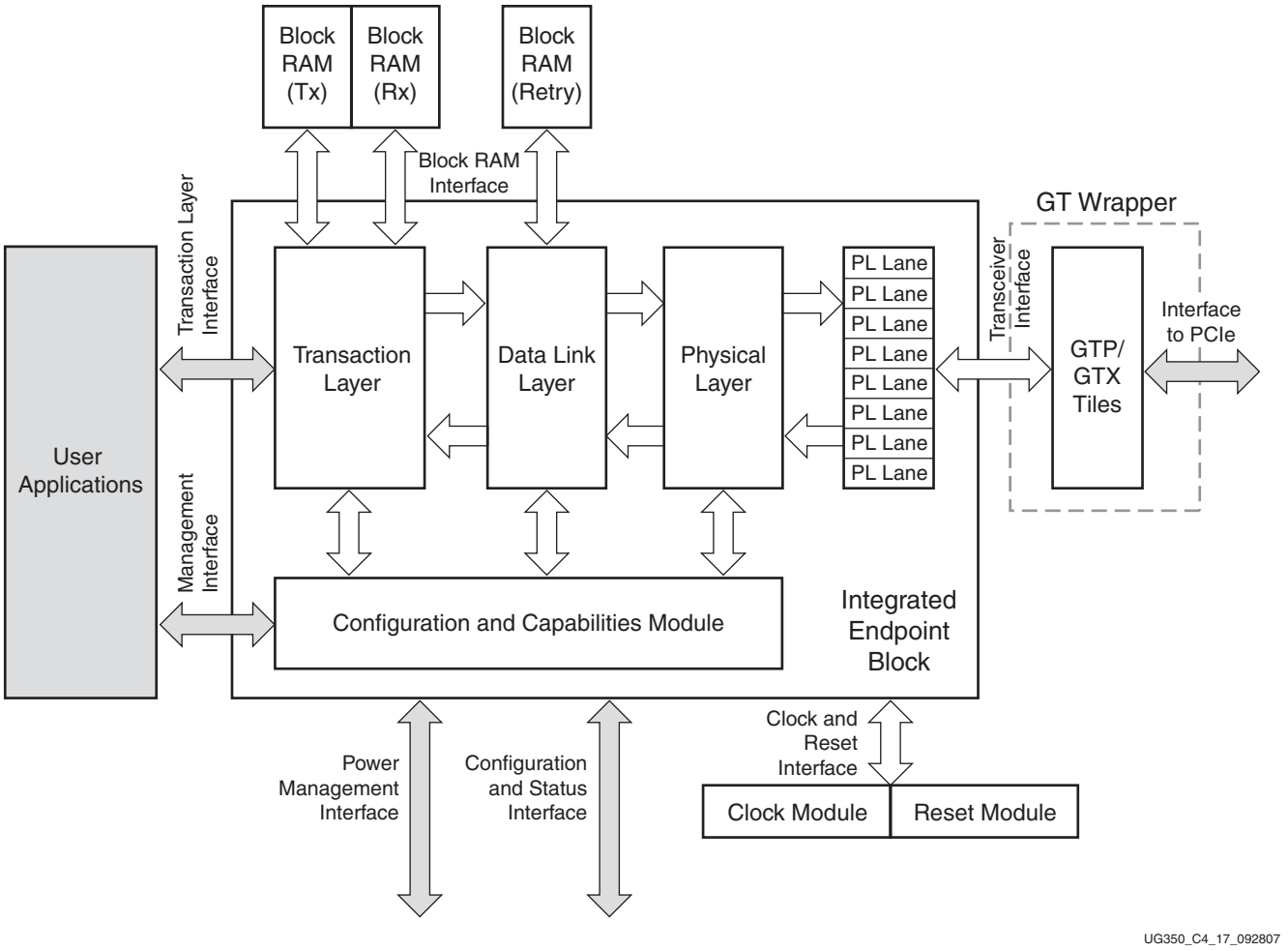


Figure 4-17: **GT Wrapper**

Note:

The GTP transceiver is required to signal an 8B/10B error by setting RXSTATUS[2:0] to `3'b100`, RXDATA[7:0] to `8'hFE` and RXCHARISK to `1`. The transceiver correctly sets RXSTATUS, but does not indicate the correct values on RXDATA and RXCHARISK. As a result, an 8B/10B error manifests itself as an LCRC error and the integrated block transmits a NAK DLLP. This does not cause any fatal errors, but results in a non-compliant response. Users can work around this by monitoring the GTP transceiver RXSTATUS output and using fabric logic to insert the correct values on the RXDATA and RXDATAK inputs to the integrated block.

Block RAM Wrapper

The block RAM wrapper (see Figure 4-18) connects the integrated Endpoint block to the required block RAMs through the block RAM interface. The buffer sizes and block RAM requirements are calculated based upon user input. The wrapper instantiates the correct number of block RAMs. The number of block RAMs in each buffer (TX, RX, Retry) is always rounded up to the nearest power of two. When multiple block RAMs are instantiated, they are connected in a *gang* mode. In *gang* mode, the memory bus width is distributed across the multiple block RAMs. All block RAMs that constitute a buffer are enabled simultaneously with the same address bits. For example, a buffer requirement of 16 kB uses four block RAMs by configuring each in a 2k x 16 mode with 11 address pins and 16 data pins. When additional pipelining is needed in the block RAM data/control path, pipelining stages must be added and the buffer latency attributes in the top wrapper must be set appropriately. See the Buffer Latency section in UG197 for more information on adding pipeline stages.

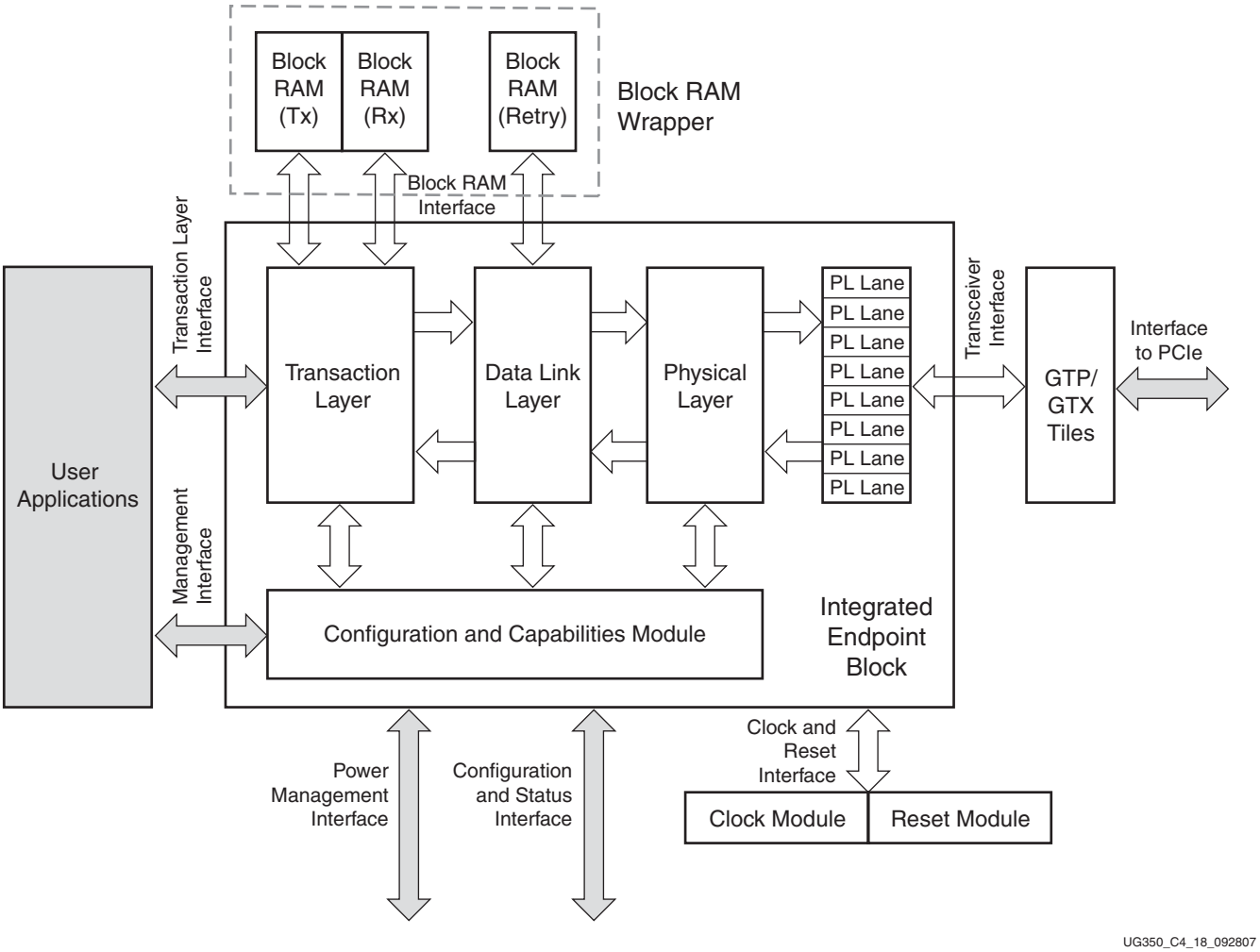


Figure 4-18: **Block RAM Wrapper**

Clock Module

The clocking requirements of the integrated Endpoint block change based on the number of lanes being used and the frequency of the user application. Based on the user selection, the clocking module (Figure 4-19) is automatically configured to generate the correct clock outputs with the desired frequency. The various clocking schemes are described in detail in UG197.

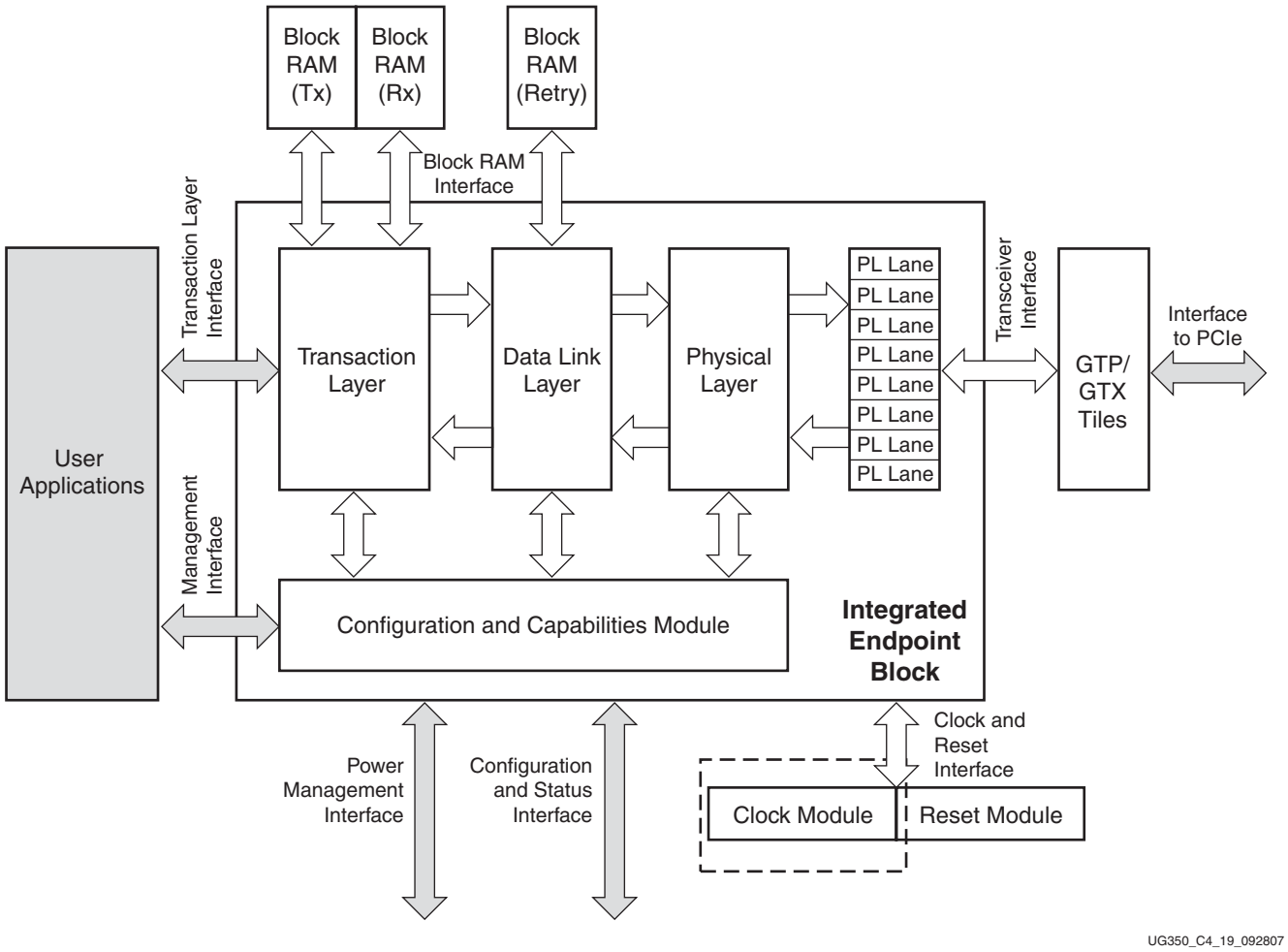


Figure 4-19: **Clock Module**

Reset Module

The integrated Endpoint block requires reset logic in the fabric for correct operation. The required reset logic depends on the RESETMODE attribute setting and the user's reset scheme. The reset module (see Figure 4-20, page 72) satisfies all the reset requirements of the integrated Endpoint block. The various reset domains and the reset inputs that control the domains in the integrated Endpoint block are explained in detail in UG197.

When the user selects a four reset or six reset domain, an internal attribute of the integrated Endpoint block, RESETMODE, is set to FALSE or TRUE, respectively. The reset module performs the following functions:

- Holds the integrated Endpoint block in reset (all registers are reset) during configuration of the FPGA.
- Holds the integrated Endpoint block in reset (all registers are reset) until the clocks generated by the clock module are stable.
- Holds a subset of the integrated Endpoint block in reset while the user master reset is active. The user can connect the user master reset input pin of the reset block to the desired system level resets to achieve the desired system level control.
- Asserts the `crm_user_cfg_rst_n` input of the integrated Endpoint block, whenever CRMPWRSOFTRESETN is asserted. This resets only the registers in the PCIe Configuration Space.
- When RESETMODE = FALSE, `the crm_urst_n` input resets all registers in the integrated Endpoint block except for sticky and management interface registers when:
  - L0_DL_UP_DOWN[0] transitions from 1 to 0

  or
  - The user master reset is asserted

  or
  - The LTSSM transitions from disabled, loopback, recovery, configuration, or hot rest to the detect state
- When RESETMODE = TRUE, asserts the appropriate reset inputs of the integrated Endpoint block when:
  - L0_DL_UP_DOWN[0] transitions from 1 to 0

  or
  - The user master reset is asserted

  or
  - CRMDOHOTRESETN is asserted

  Note: The crm_urst_n signal resets just the user_clk domain. Additional reset logic is provided in pcie_top.v to ensure the design can be reset when it is in a *train down* state (x4 design in an x8 slot, x1 design in an x4 or x8 slot).

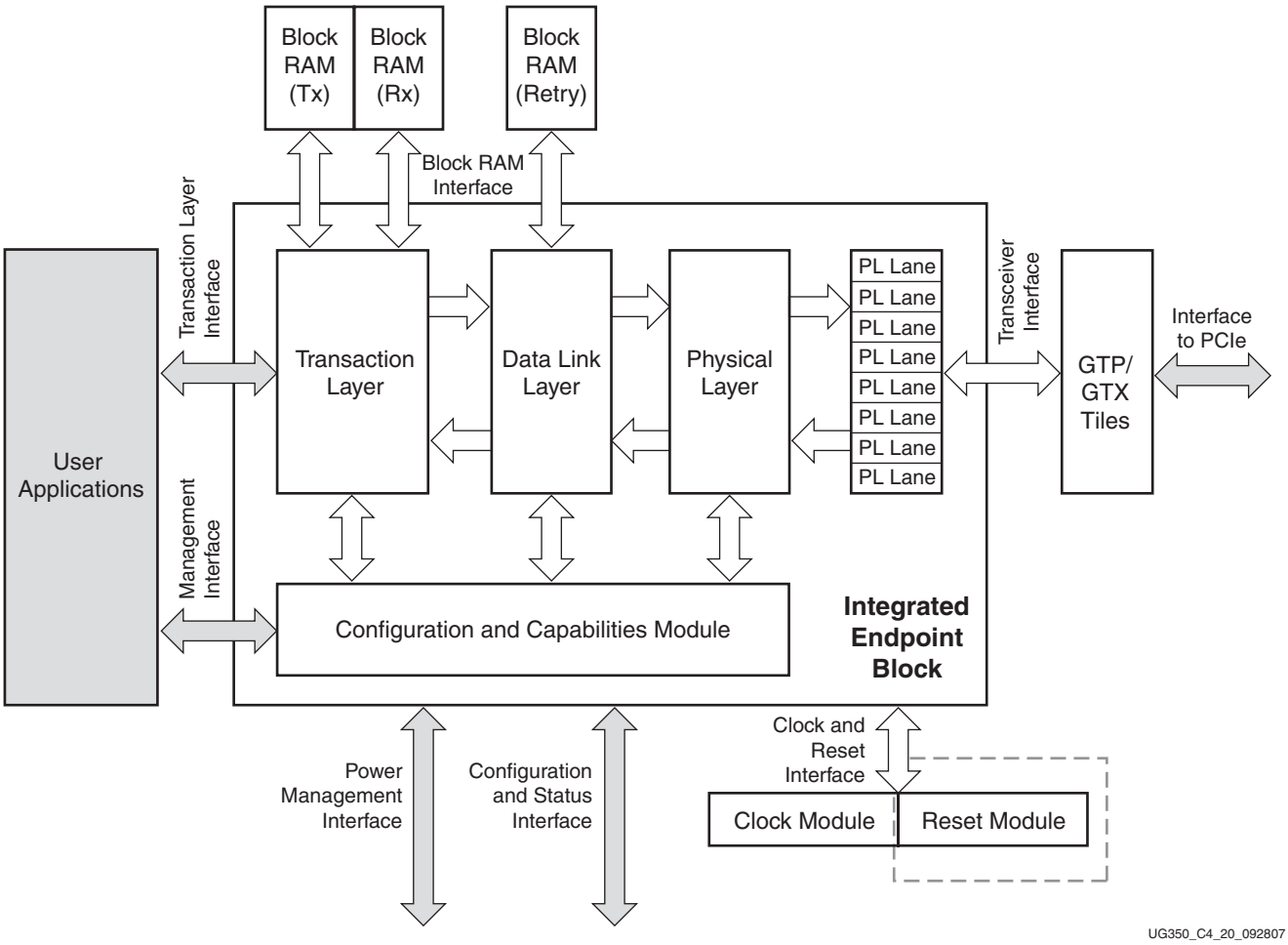


Figure 4-20: **Reset Module**

Directory Structure

This section defines the core directory structure and associated file contents. Table 4-9 provides a description of each file in the core directories.

Table 4-9: **Directory Files**

Name	Description
CORE Generator Project Files (`<project_dir>`)	
<component_name>.cgp	
<component_name>.xco	CORE Generator project-specific option file
*<component_name>*_flist.txt	List of files delivered with the core
Release Notes (`<project_dir>/<component_name>`)	
readme.txt	Release Notes

Table 4-9: **Directory Files *(Continued)***

Name	Description
SRC (<project_dir>/<component_name>/src)	
pci_express_wrapper.v(hd)	Top-level wrapper that contains parameter settings based on CORE generator GUI selections
pcie_top.v(hd)	Integrates clocking module, reset logic, GT wrapper, memory wrapper and integrated Endpoint block instantiation
pcie_clocking.v(hd)	Generates the clocks required for the integrated Endpoint block
pcie_reset_logic.v(hd)	Generates resets required for the integrated Endpoint block
pcie_mim_wrapper.v(hd)	Memory wrapper which integrates the block RAM configurations for TX, RX and Retry Buffers
bram_common.v(hd)	Generic module used to configure block RAMs
pcie_blk_cf_mgmt.v(hd) (1)	Contains polling logic to shadow certain registers in the integrated Endpoint block
pcie_cmm_decoder.v(hd) (1)	Compares, decodes and generates *bar hit* signals
Example Design (<project_dir>/<component_name>/example_design)	
pcie_top_pkg.vhd	Contains parameter calculations and settings for pcie_ep instantiation
mem_ep_app_top.v	Top-level memory endpoint application file
completer_mem_block_top.v	Top-level file for the completion logic
completer_mem_block.v	Contains the memory and related logic for the completion block
completer_mem_block_machine.v	Contains the completer state machine
mem_ep_app_<*lanes*>_<*board*>_<*part*>.ucf	UCF file necessary to run implementation

Table 4-9: **Directory Files *(Continued)***

Name	Description
Implementation Script Files (`<project_dir>/<component_name>/implement`)	
implement.sh, implement.bat	LINUX or DOS implementation script (User should type `implement.sh/implement.bat -board <ml523 OR ml555>` to execute the script)
mem_ep_app_top.scr	XST synthesis script for the top-level example design
mem_ep_app_top.prj	XST synthesis project file
gen_ucf.pl	UCF generation script that generates a synplify-friendly UCF file
Results Directory and Files (`<project_dir>/<component_name>/implement/results`)_<lanes>_<board>_<part>	
results_<lanes>_<board>_<part> directory created by implement script; results of implement script placed in results directory. The results directory is named based on user input for the board, while the values of *lane* and *part* are embedded in the `implement.sh` script.	
Functional Simulation (`<project_dir>/<component_name>/simulation/functional`)	
simulate_mti.sh	ModelSim simulation script for compiling, loading, and running the demonstration test bench
wave.do	ModelSim script for loading signals into the waveform viewer
simulate_ncsim.sh	NCSim simulation script for compiling, loading, and running the demonstration test bench
simulation_vcs.sh	VCS simulation script for compiling, loading, and running the demonstration test bench
simulate_isim.sh	ISim simulation script for compiling, loading and running the demonstration test bench
Test Bench (`<project_dir>/<component_name>/testbench`)	
tb.v	Top-level test bench that instantiates a *near end* device (limited downstream capability) and a *far end* device (endpoint device)
tb_def.v	Configuration information for the test bench
pcie_top_ne.v	Top-level file for the *near end* device
pcie_ne.v	Contains a PCIe primitive configured as a *near end* device with limited downstream functionality

Table 4-9: **Directory Files *(Continued)***

Name	Description
pcie_tasks.v	Set of tasks and functions to simulate the test bench

Notes:

1. These files are generated only when the Enable BAR Monitor Enable box is selected in the Base Address Register Configuration Menu

Chapter 5

Designing with the Core

This chapter provides design instructions for the Virtex™-5 LogiCORE™ Endpoint Block for PCIe™ produced by the CORE Generator™ GUI. The descriptions and waveforms assume knowledge of the PCI Express Transaction Layer Packet (TLP) header fields. Header fields are defined in the *PCI Express Base Specification v1.1*, Chapter 2, Transaction Layer Specification.

TLP Format on the User Application 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 5-1 represents a typical 32-bit addressable Memory Write Request TLP (as illustrated in Chapter 2 of the specification).

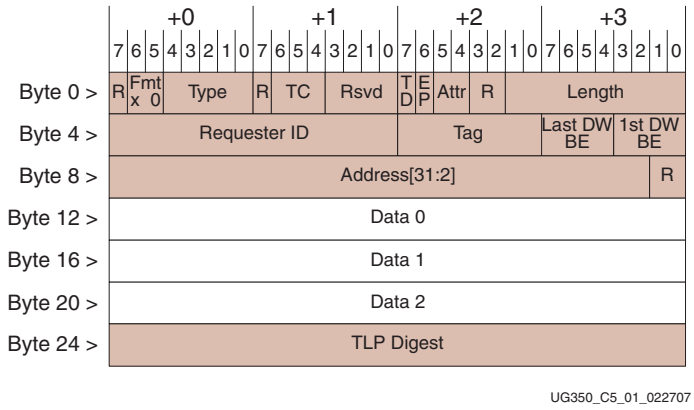


Figure 5-1: **PCI Express Base Specification Byte Order**

When using the transaction interface, packets are arranged on the entire 64-bit data path. Figure 5-2, page 78 shows the same example packet on the user application interface. Byte 0 of the packet appears on `llk_tx_data[63:56]` (outbound) or `llk_rx_data[63:56]` (inbound) of the first QWORD, byte 1 on `llk_tx_data[55:48]` or `llk_rx_data[55:48]`, and so forth. Byte 8 of the packet then appears on `llk_tx_data[63:56]` or `llk_rx_data[63:56]` of the second QWORD. The Header section of the packet consists of either three or four DWORDs, determined by the TLP format and type as described in section 2.2 of the *PCI Express Base Specification*.

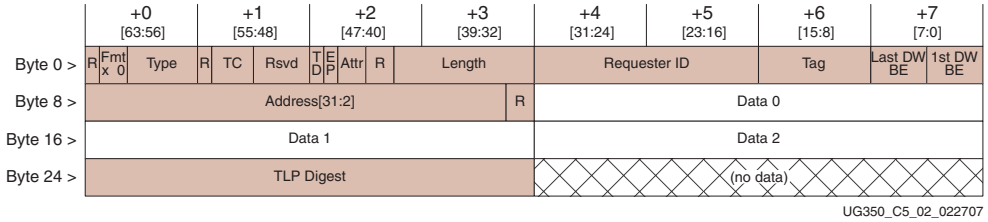


Figure 5-2: **PCI Express Endpoint Byte Order**

Packets sent to the core for transmission must follow the formatting rules for Transaction Layer Packets (TLPs) as specified in Chapter 2 of the *PCI Express Base Specification*. The user application is responsible for ensuring its packets' validity, as the core does not validate the packets. The exact fields of a given TLP vary depending on the type of packet being transmitted. Figure 5-3 and Figure 5-4 show the arrangement of both 3 and 4 DWORD packets on the transmit and receive data interface. The first DWORD of all TLPs has the same fields as shown in these figures and indicated by the Chapter 2 of the *PCI Express Base Specification*. Packets are shown with optional data payload and TLP Digest.

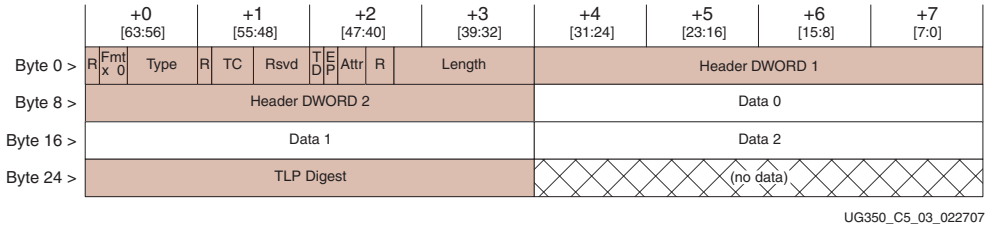


Figure 5-3: **Arrangement of 3 DWORD Packet on User Application Interface**

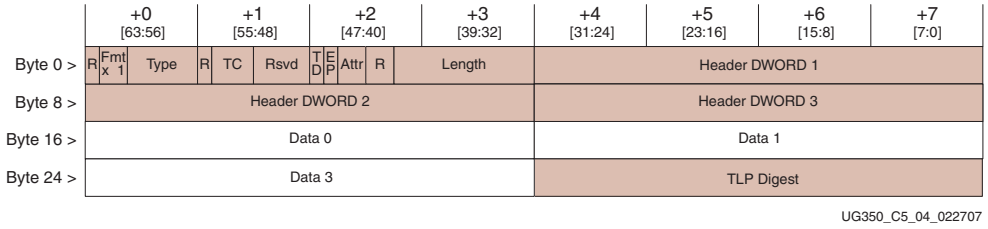


Figure 5-4: **Arrangement of 4 DWORD Packet on User Application Interface**

The presence of a TLP Digest or ECRC is indicated by the value of TD field in the TLP Header section. When TD=1, a correctly computed CRC32 remainder is expected to be presented as the last DWORD of the packet. The CRC32 remainder DWORD is not included in the length field of the TLP header. The user application must calculate and present the TLP Digest as part of the packet when transmitting packets. Upon receiving packets with a TLP Digest present, the user application must check the validity of the CRC32 based on the contents of the packet. The core does not check the TLP Digest for incoming packets.

Transmitting Outbound Packets

General TLP Transmit Information

The integrated Endpoint block 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

Transmit Framing

The user application uses `llk_tx_sof_n` and `llk_tx_eof_n` signals to mark the start and end of the TLP frames. The following conditions are framing errors and are not allowed:

- Two Start of Frames (SOFs) without an intervening End of Frame (EOF)
- Two EOFs without an intervening SOF
- An SOF and EOF in the same cycle

DWORD Enables

The user application transaction layer interface data bus is 64 bits wide allowing the user to transmit one QWORD of data into the core on a clock cycle. The PCI Express protocol allows DWORD or 32-bit alignment of the header and data. The `llk_tx_enable_n[1:0]` bus indicates which DWORD(s) contain valid header or data information. Bit 1 of `llk_tx_enable_n` maps to `llk_tx_data[63:32]` and bit 0 of `llk_tx_enable_n` maps to `llk_tx_data[31:0]`. A value of 0 indicates the DWORD is valid.

The core requires that the user to enable all 64-bits of `llk_tx_data` except on the last cycle of a TLP transfer indicated by `llk_tx_eof_n` = 0. For the last QWORD to be transferred into the core, it is possible that only `llk_tx_data[63:32]` is valid since the PCI Express protocol allows for the header and data to be DWORD aligned. This is denoted by `llk_tx_enable_n[1:0]` = 01 during `llk_tx_eof_n`. Otherwise, `llk_tx_enable_n[1:0]` = 00 is used for all other cycles during a TLP transfer.

TLP Transfer Operation

Table 3-7, page 37 defines the user application transmit signals. To transmit a TLP, the transmit user application must perform the following sequence of events on the Transaction transmit interface:

1. The user application logic asserts `llk_tx_ch_tc[2:0]` to identify a traffic class and `llk_tx_ch_fifo[1:0]` to select a channel, either posted, non-posted, or completion.
2. On the next cycle, the user application logic asserts `llk_tx_src_rdy_n` and `llk_tx_sof_n` and presents the first TLP QWORD on `llk_tx_data[63:0]`. The user also asserts `llk_tx_enable_n[1:0]` to 00 indicating all 64-bits on

`llk_tx_data[63:0]` contain valid information. If the core is asserting `llk_tx_dst_rdy_n`, the QWORD is accepted immediately; otherwise, the user application must keep the QWORD presented until the core asserts `llk_tx_dst_rdy_n`. The timing of llk_tx_dst_rdy_n depends on the amount of buffer space available for the selected channel. For more information, see "Channel Space Available," page 88.

3. The transmit user application asserts `llk_tx_src_rdy_n` and presents the remainder of the TLP QWORDs on `llk_tx_data[63:0]` for subsequent clock cycles (for which the Endpoint core asserts `llk_tx_dst_rdy_n`).
4. At any time, the user or the core can throttle the transmission of the packet by deasserting `llk_tx_src_rdy_n` or `llk_tx_dst_rdy_n`, respectively.
5. The transmit user application asserts `llk_tx_src_rdy_n` and `llk_tx_eof_n` together with the last QWORD data. If both DWORDS of the last transfer are valid, they are presented on `llk_tx_data[63:0]` and `llk_tx_enable_n[1:0]` is driven to 00; otherwise, the remaining DWORD is presented on `llk_tx_data[63:32]` and `llk_tx_enable_n[1:0]` is driven to 01.
6. At the next clock cycle, the transmit user application deasserts `llk_tx_src_rdy_n` to signal the end of valid transfers on `llk_tx_data[63:0]`.
7. After transmission of a packet, the next packet can be sent immediately if it belongs to the same channel (FIFO and TC). But, the transmit user application must wait an additional two cycles of `llk_tx_dst_rdy_n` assertion after the assertion of `llk_tx_eof_n` before switching `llk_tx_ch_fifo[1:0]` or `llk_tx_ch_tc[2:0]`.

Figure 5-5 illustrates a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request. The user application indicated this is a Non-Posted packet to be sent with a traffic class of two. When the user application asserts `llk_tx_eof_n`, it also places a value of `01` on `llk_tx_enable_n[1:0]` notifying the core that only `llk_tx_data[63:32]` contains valid data.

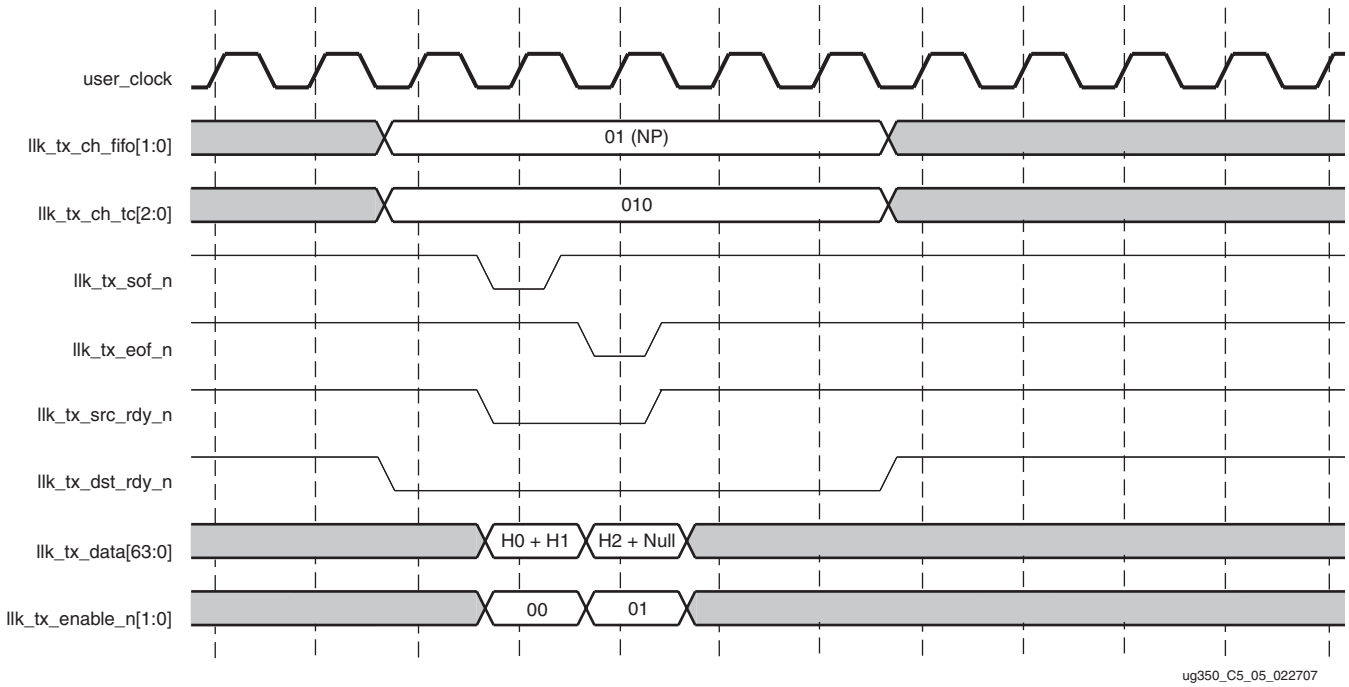


Figure 5-5: **TLP with 3-DW Header without Payload**

Figure 5-6 illustrates a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request. The user application indicated this is a Non-Posted packet to be sent with a traffic class of zero. When the user application asserts `llk_tx_eof_n`, it also places a value of 00 on `llk_tx_enable_n[1:0]` notifying the core that `llk_tx_data[63:0]` contains valid data.

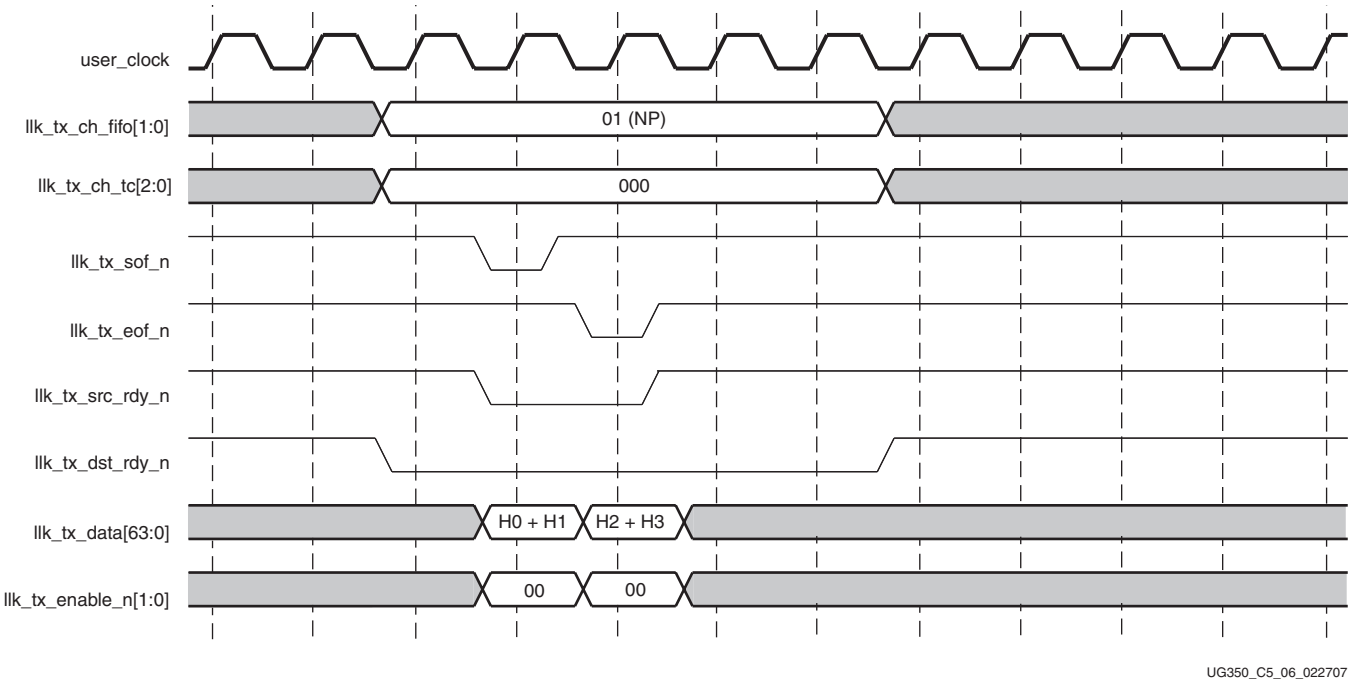


Figure 5-6: **TLP with 4-DW Header without Payload**

Figure 5-7 illustrates a 3-DW TLP header with a data payload; an example is a completion with data. The user application indicated this is a Completion packet to be sent with a traffic class of zero. When the user application asserts `llk_tx_eof_n`, it also places a value of 00 on `llk_tx_enable_n[1:0]` notifying the core that `llk_tx_data[63:0]` contains valid data.

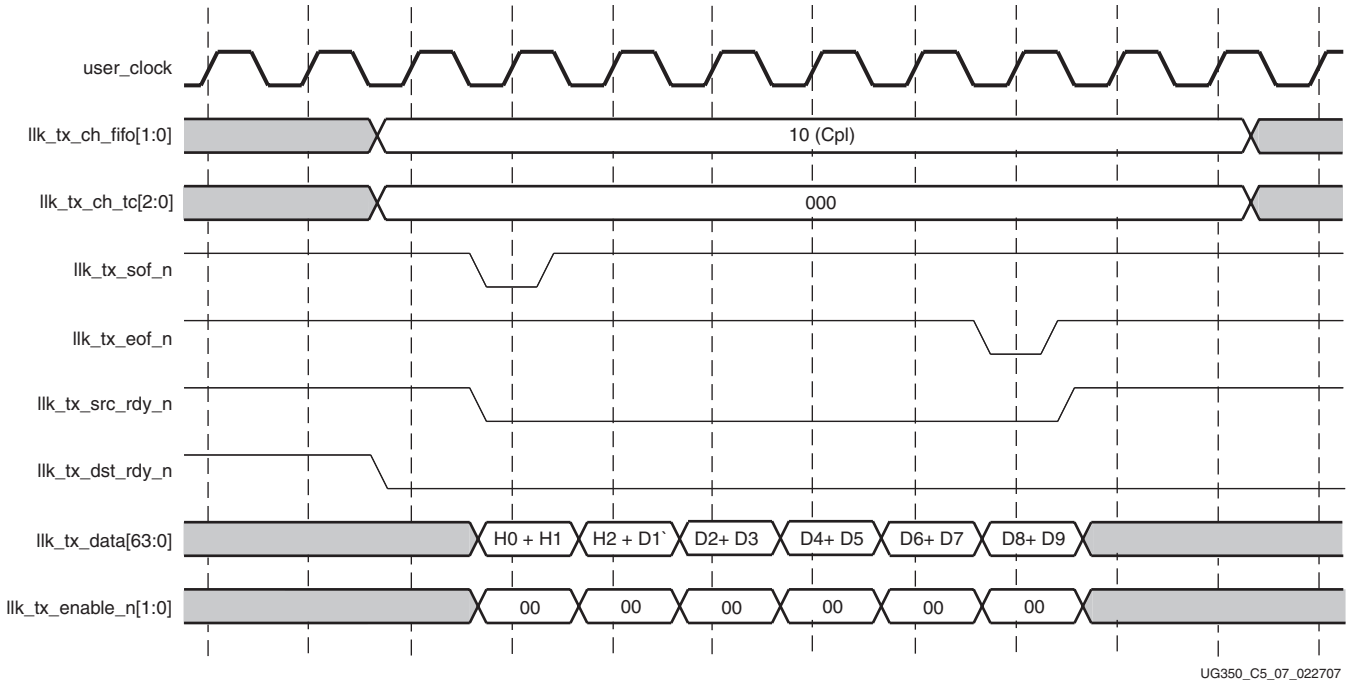


Figure 5-7: **TLP with 3-DW Header with Payload**

Figure 5-8 illustrates a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request. The user application indicated this is a Posted packet to be sent with a traffic class of zero When the user application asserts `llk_tx_eof_n`, it also places a value of 01 on `llk_tx_enable_n[1:0]` notifying the core that `llk_tx_data[63:32]` contains valid data.

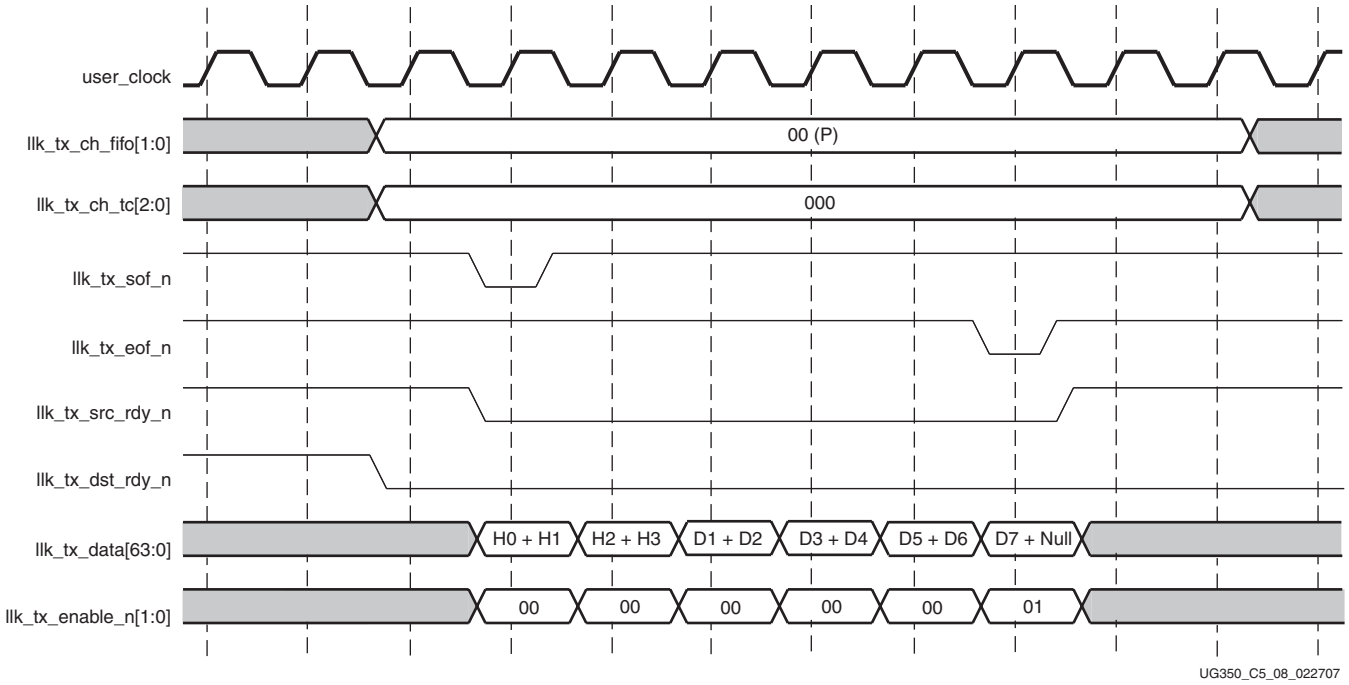


Figure 5-8: **TLP with 4-DW Header with Payload**

Source Throttling on the Transmit Data Path

The transmit interface lets the user application throttle back if it has no data to present on `llk_tx_data[63:0]`. When this condition occurs, the user application deasserts `llk_tx_src_rdy_n`, which instructs the transmit interface to disregard data presented on `llk_tx_data[63:0]`. Figure 5-9 illustrates the source throttling mechanism, where the user application does not have data to present every clock cycle, and for this reason must deassert `llk_tx_src_rdy_n` during these cycles.

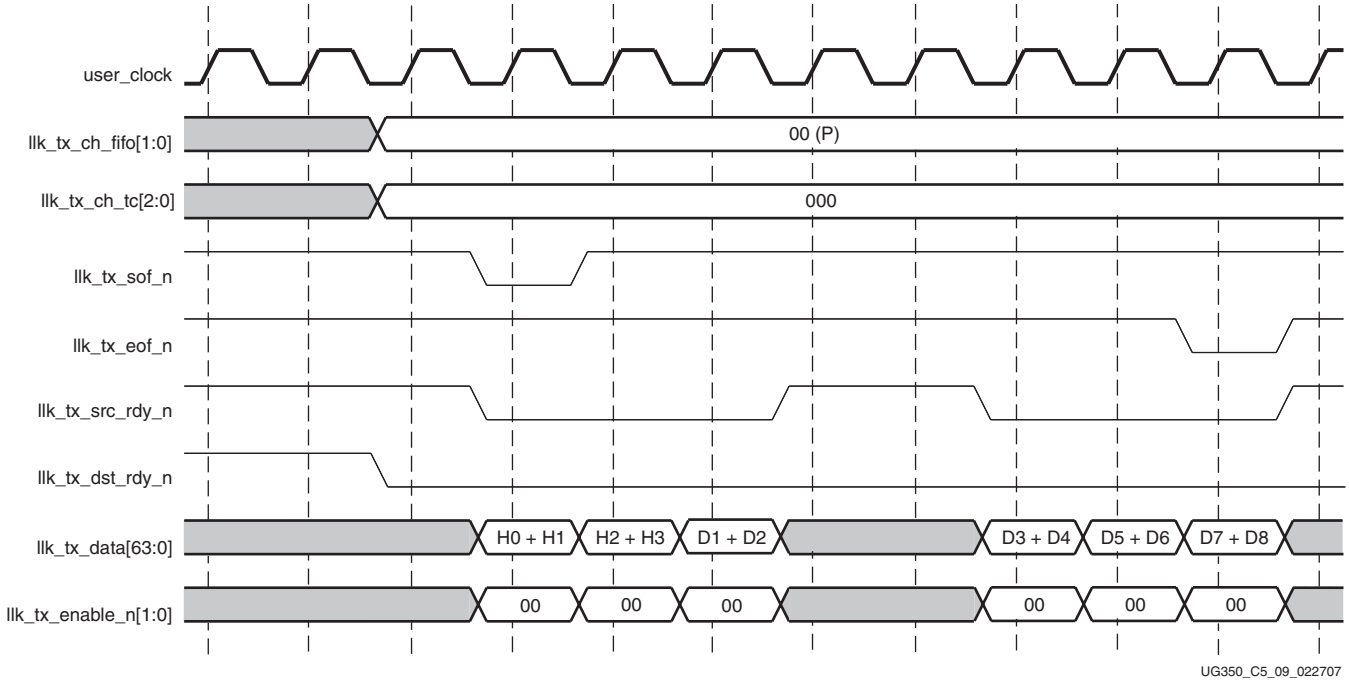


Figure 5-9: **Source Throttling by the User Application**

Destination Throttling

If the transmit buffer becomes full during the process of transferring a TLP from the user application into one of the TX FIFOs, the core deasserts `llk_tx_dst_rdy_n` and stalls data transfers on the Transaction Layer interface until space becomes available again as shown in Figure 5-10.

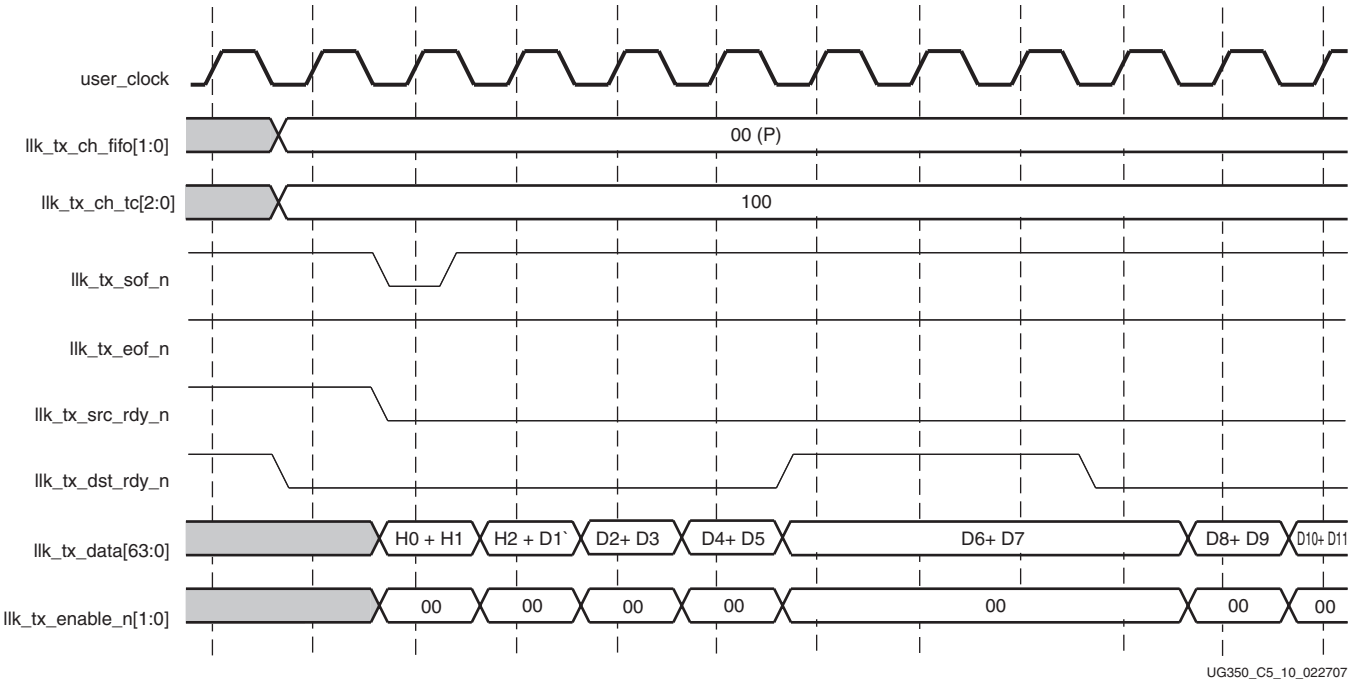


Figure 5-10: **Destination Throttling by the Core**

The core can throttle the data path at any time during the transfer including the beginning as shown in Figure 5-11. For this reason, the user application must check that `llk_tx_dst_rdy_n` is asserted on each clock cycle that `llk_tx_src_rdy_n` is asserted ensuring a valid transfer of data into the core.

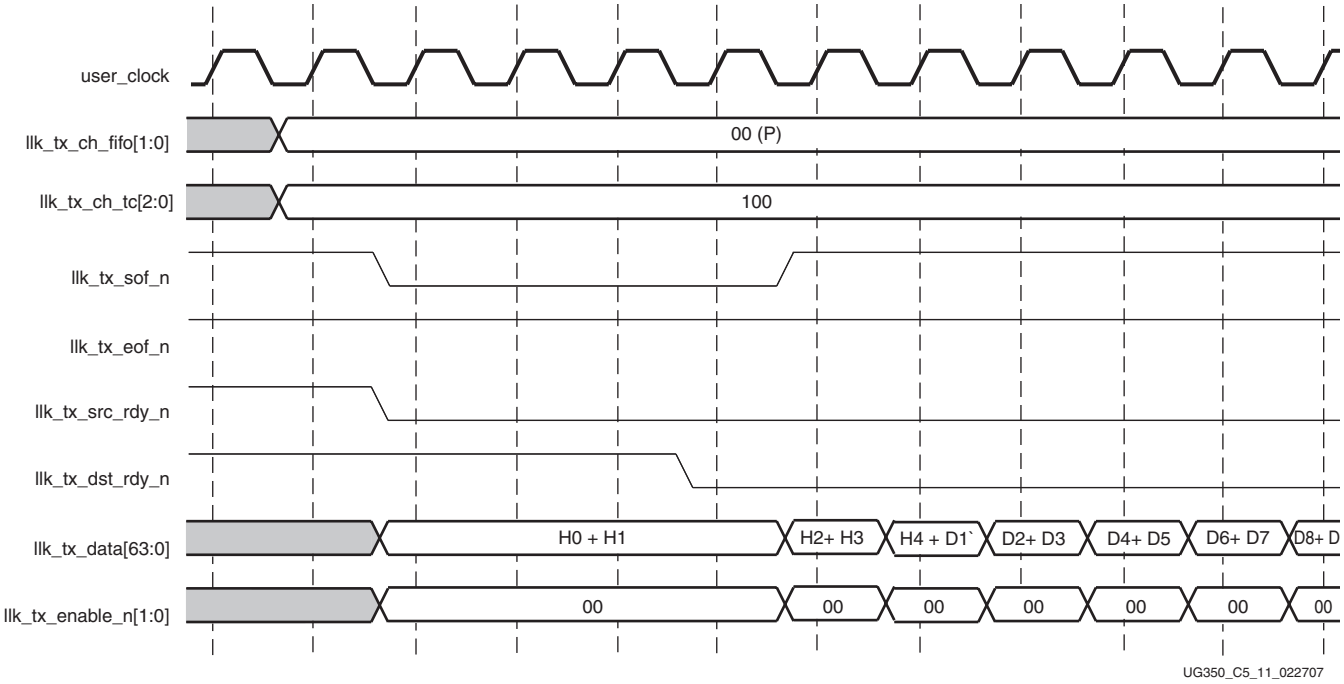


Figure 5-11: **Destination Throttling at Beginning of TLP Transfer**

Selecting the Appropriate Channel

The integrated Endpoint block supports up to eight traffic classes which are mapped to either one or two Virtual Channels (VCs). To enable multiple traffic classes or virtual channels, see "Advanced Physical Layer Settings Menu," page 57.

When the integrated Endpoint block is ready to accept a packet from the user application, it notifies the user that one or more channels are available. There is one channel ready signal per traffic class according to the type of packet to be transmitted as shown in Table 5-1. For example, if `llk_tx_ch_posted_ready_n[7:0]` is `11101100b`, it means that a posted packet with a traffic class of either 4, 1, or 0 could be accepted by the core. More than one channel can be available at the same time and the user application must indicate which channel to use for a given TLP. This selection criteria depends on the requirements of the user application and system design.

Table 5-1: **Channel Ready Signals**

Signal Name	Description
llk_tx_ch_posted_ready_n[7:0]	Channel Ready for Posted Packets
llk_tx_ch_non_posted_ready_n[7:0]	Channel Ready for Non-Posted Packets
llk_tx_ch_completion_ready_n[7:0]	Channel Ready for Completion Packets

The user application sets `llk_tx_ch_tc[2:0]` to select the traffic class and `llk_tx_ch_fifo[1:0]` to select the TX FIFO as shown in Table 5-2.

Table 5-2: **Mapping of llk_tx_ch_fifo[1:0] to TX FIFO Type**

Value	FIFO Selected
00	Posted TX FIFO
01	Non-Posted TX FIFO
10	Completion TX FIFO

The channel ready signals give the user a real time update as to if a particular channel is ready to accept a packet from the user. However, these signals do not guarantee the entire packet can be received without the core stalling the transfer while it waits for more room to be available in the buffer. See "Channel Space Available," page 88 for more information on how to determine if the entire packet can be received without destination throttling.

If the user application has multiple packets awaiting transmission, it can use the channel ready signals to decide which packet to send next assuming no particular packet ordering is required.

More importantly, the user application can avoid potential deadlock situations where posted or completion packets become blocked behind a non-posted packet stalled by the core. The core supports the transaction ordering rules specified by the *PCI Express Specification* 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 buffer space 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 `llk_tx_ch_non_posted_ready_n[7:0]` and `llk_tx_chan_space[9:0]` signals, the user application can ensure there is room for the Non-Posted TLP to be accepted before asserting `llk_tx_sof_n`. If no room is available for the Non-Posted TLP, the user application can transfer any pending posted or completion TLPs first as shown in Figure 5-12, page 88. Otherwise, once the user application asserts `llk_tx_sof_n` it would have to wait for the core to assert `llk_tx_dst_rdy_n` to accept the packet. During this time, the Completion or Posted TLP would be blocked.

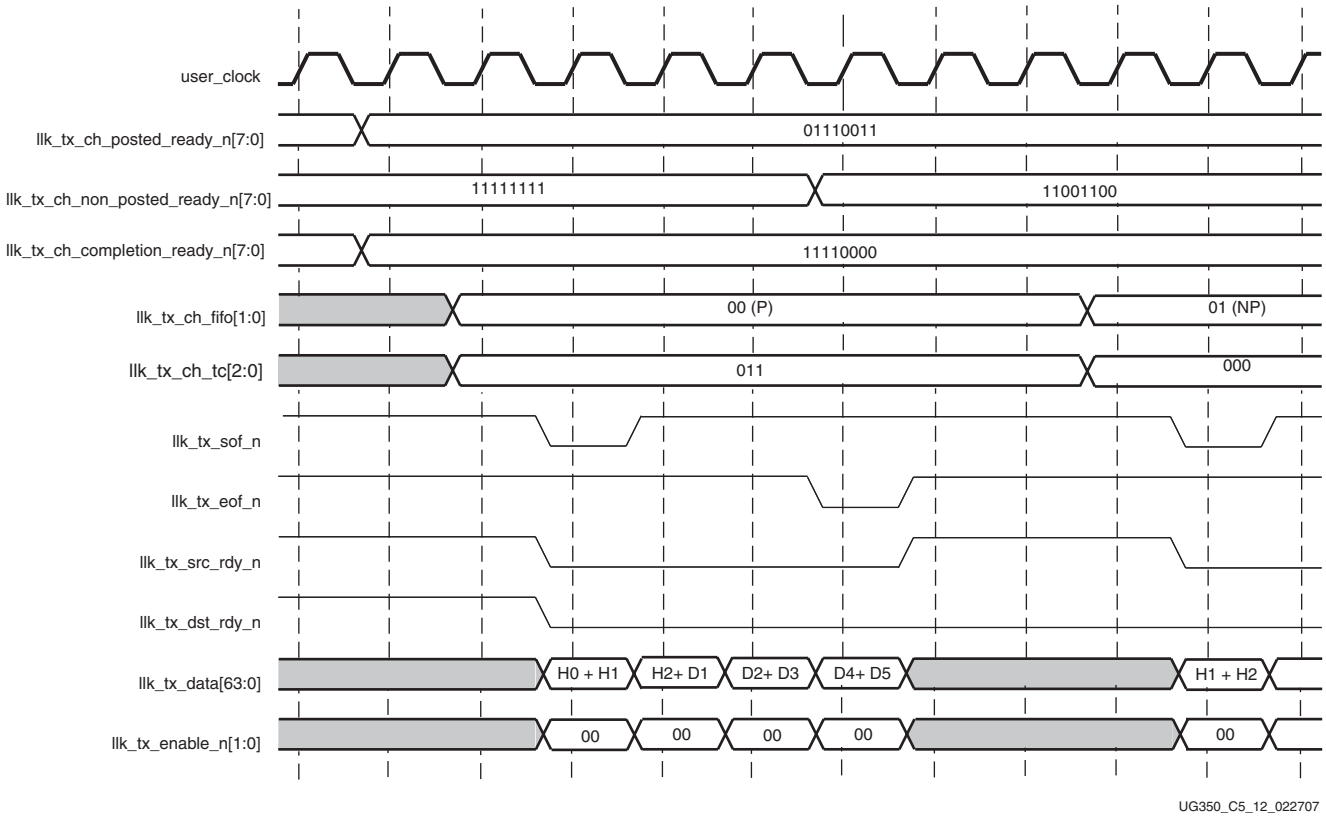


Figure 5-12: **Avoiding Blocking Posted and Completion TLPs**

The user application must be aware that channel ready signals give a real time indication of whether a channel is ready to start accepting a packet, however, this does not guarantee the core will accept the entire packet. Because of this, the packet being transferred could be stalled until more space becomes available. The `llk_tx_chan_space[9:0]` bus allows the user application to know how much space is available before asserting `llk_tx_sof_n`. See "Channel Space Available" for further information.

Channel Space Available

The user application may optionally choose not to begin the transfer of data until the TX FIFO for the chosen channel has enough room to receive the entire packet. If the TX FIFO is full when the user asserts `llk_tx_sof_n` and `llk_tx_src_rdy_n`, the core stalls the transfer as shown in Figure 5-11, page 86. Space becomes available as previously transferred packets from the user application are drained from the TX FIFOs as they are sent over the link. Outgoing TLPs are held in the core's transmit buffer until transmitted on the interface to the PCI Express partner. After a TLP is transmitted, it is stored in an internal retry buffer until the link partner acknowledges receipt of the packet. Optionally, the user can check the amount of space available in a channel using `llk_tx_chan_space[9:0]` and decide not to begin sending the packet over the Transaction Layer interface if insufficient space is available to send the packet without stalling.

After the user application starts transferring a packet it cannot disconnect the packet prematurely. The `llk_tx_chan_space[9:0]` bus allows the user to know how much room is available prior to asserting `llk_tx_sof_n`. This way the user application will know ahead of time if the core can accept the entire packet without stalling the transfer by deasserting `llk_tx_dst_rdy_n`. The user may use `llk_tx_chan_space[9:0]` to ensure a higher priority TLP does not get blocked while a prior TLP is in the middle of being transferred but momentarily stalled while it waits for more space to be available. The time in which the core may stall a transfer using `llk_tx_dst_rdy_n` depends on how fast prior packets are sent to the link partner.

The `llk_tx_chan_space[9:0]` bus indicates the amount of free space in the TX FIFO as selected by `llk_tx_ch_tc[2:0]` and `llk_tx_ch_fifo[1:0]` as shown in Figure 5-13. There is a delay of one cycle between the change in a channel select on `llk_tx_ch_tc[2:0]` and `llk_tx_ch_fifo[1:0]` before valid data is presented on `llk_tx_chan_space[9:0]` and `llk_tx_dst_rdy_n`.

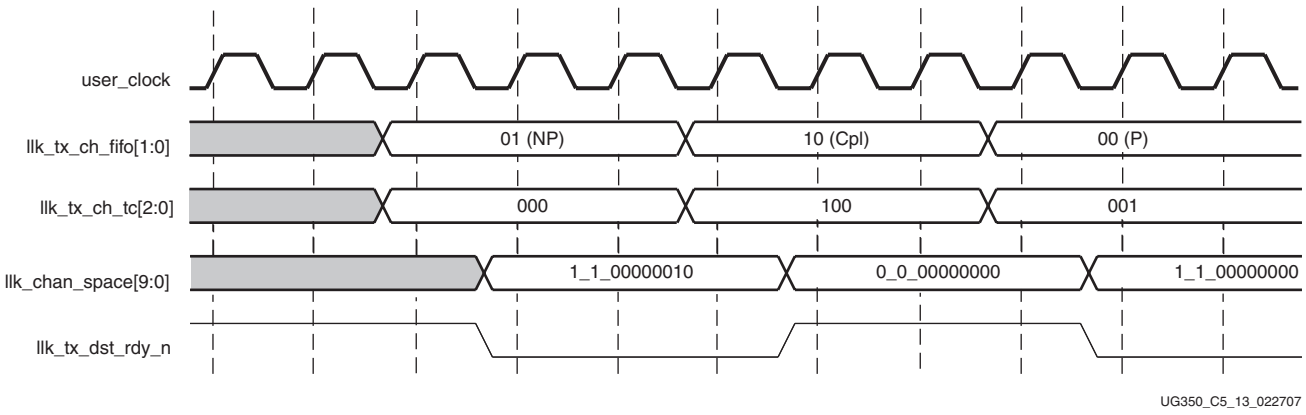


Figure 5-13: **Timing of llk_tx_chan_space[9:0]**

The `llk_tx_chan_space[9:0]` bus reports three pieces of information to the user application. It informs the user if there is space available for at least one TLP header, space for at least one data credit, and the amount of data credits available. Note that if the core is configured to support a maximum payload of 4096 bytes (256 credits), this is indicated by bits[7:0] = 00h and bit[8] = 1. So the user application must monitor both pieces of information to determine the amount of data credits available.

Bit[9] indicates if space is available for at least one TLP header

- 1: Space for at least one TLP header
- 0: No space available for a header

Bit[8] indicates if space is available for data

- 1: Space for at least 16 bytes of data or 1 data credit
- 0: No Space available for data

Bits[7:0] indicate the number of data credits available

- 1 .. 255: Number of credits available
- 0: Meaning depends on bit[8]
  - If bit[8] = 0, no credits are available
  - If bit[8] = 1, at least 256 credits are available

One header credit is equal to either a 3 or 4 DWORD header plus optional TLP digest. For example, if a posted or non-posted buffer is being evaluated then one header credit equals 20 bytes or a 4 DWORD header plus optional TLP Digest. The 4 DWORD header would be for a 64-bit addressable memory packet. If the completion buffer is being evaluated then 1 header credit equals 16 bytes or a 3 DWORD header plus optional TLP Digest.
Bit [9] essentially indicates if at least one TLP header can be accepted by the core. One data credit always equals 16 bytes.

Figure 5-13 shows the user application polling each transmit channel to find out how much space is available in each as follows:

- First, the non-posted channel is selected. One cycle later, `llk_tx_chan_space[9:0]` = 1_1_00000010. This indicates that at least there is enough room for one TLP header and two data credits or 32 bytes of data.
- Second, the completion channel is selected. One cycle later, `llk_tx_chan_space[9:0]` = 0_0_00000000. This indicates that currently no room is available to transfer a completion TLP.
- Third, the posted channel is selected. One cycle later, `llk_tx_chan_space[9:0]` = 1_1_00000000. This indicates that at least there is enough room for one TLP header and 256 data credits or 4096 bytes.

Two example scenarios in which `llk_tx_chan_space[9:0]` can be used to avoid problems are as follows:

In the first scenario the user application may need to transfer a large Memory Write TLP. However, after the channel and traffic class are chosen, `llk_tx_chan_space[9:0]` may report that only a fraction of the total TLP can be accepted at the current time. The user application could go ahead with the transfer but it can be stalled depending on how quickly more buffer space becomes available (Figure 5-14). Once the user application begins to transmit a packet it cannot stop the process until the entire packet is transferred into the core. This is indicated by the user application asserting `llk_tx_eof_n` once the packet is completely transferred to the core. Trying to stop the transfer early by asserting `llk_tx_eof_n` prematurely or trying to begin a new packet by asserting `llk_tx_sof_n` without properly finishing the previous packet by asserting `llk_tx_eof_n` will cause the integrated Endpoint block to go into an undefined state. Instead, the user application may choose to send other TLPs on different channels before beginning to send the pending Memory Write TLP.

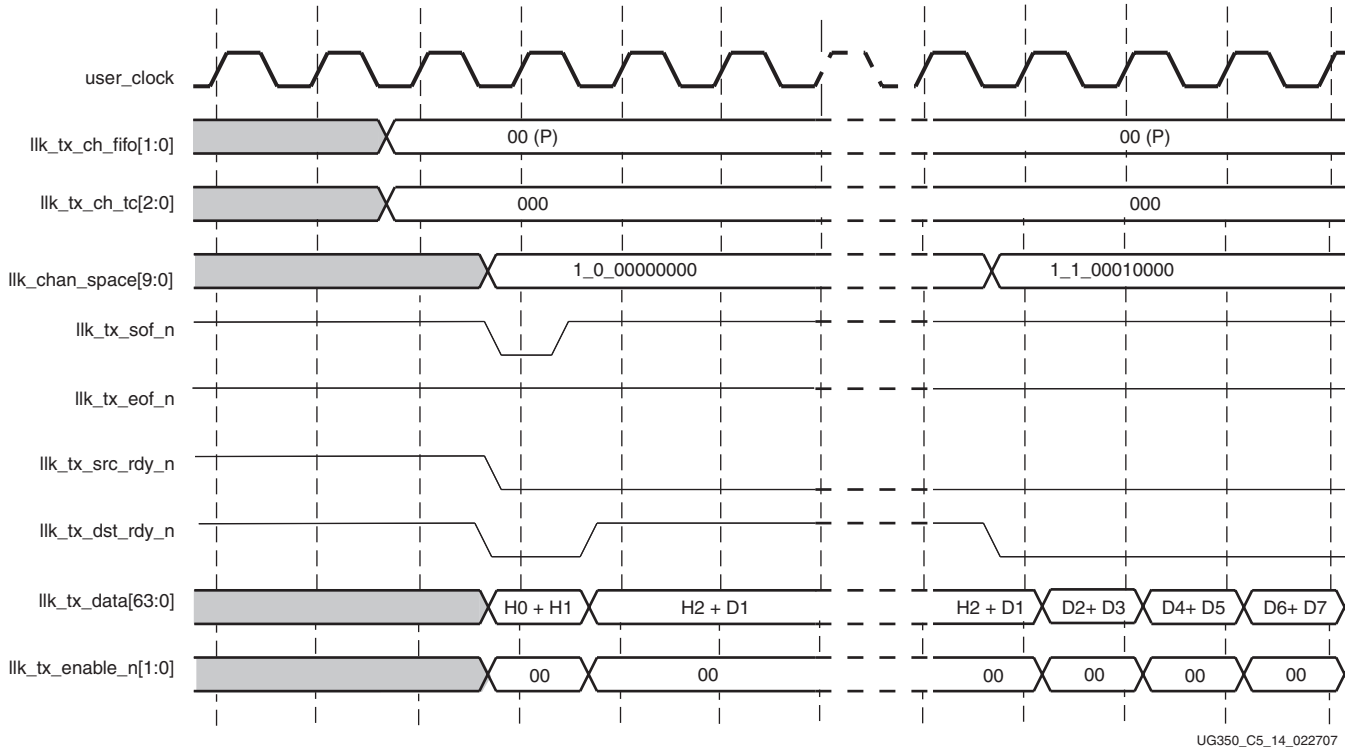


Figure 5-14: **Transferring Memory Write TLP**

The "Selecting the Appropriate Channel" section shows that the user can use `llk_tx_ch_non_posted_ready_n[7:0]` to avoid potential dead lock situations where the core cannot accept a started Non-Posted packet. In the second scenario the same result can be achieved using `llk_chan_space[9:0]` (Figure 5-15). However, there is more latency involved due to the one-cycle delay it takes for `llk_chan_space[9:0]` to update after a channel is selected. Both methods (using `llk_tx_ch_non_posted_ready_n[7:0]` or `llk_chan_space[9:0]`) achieve the same result, and the method used depends on the user application requirements.

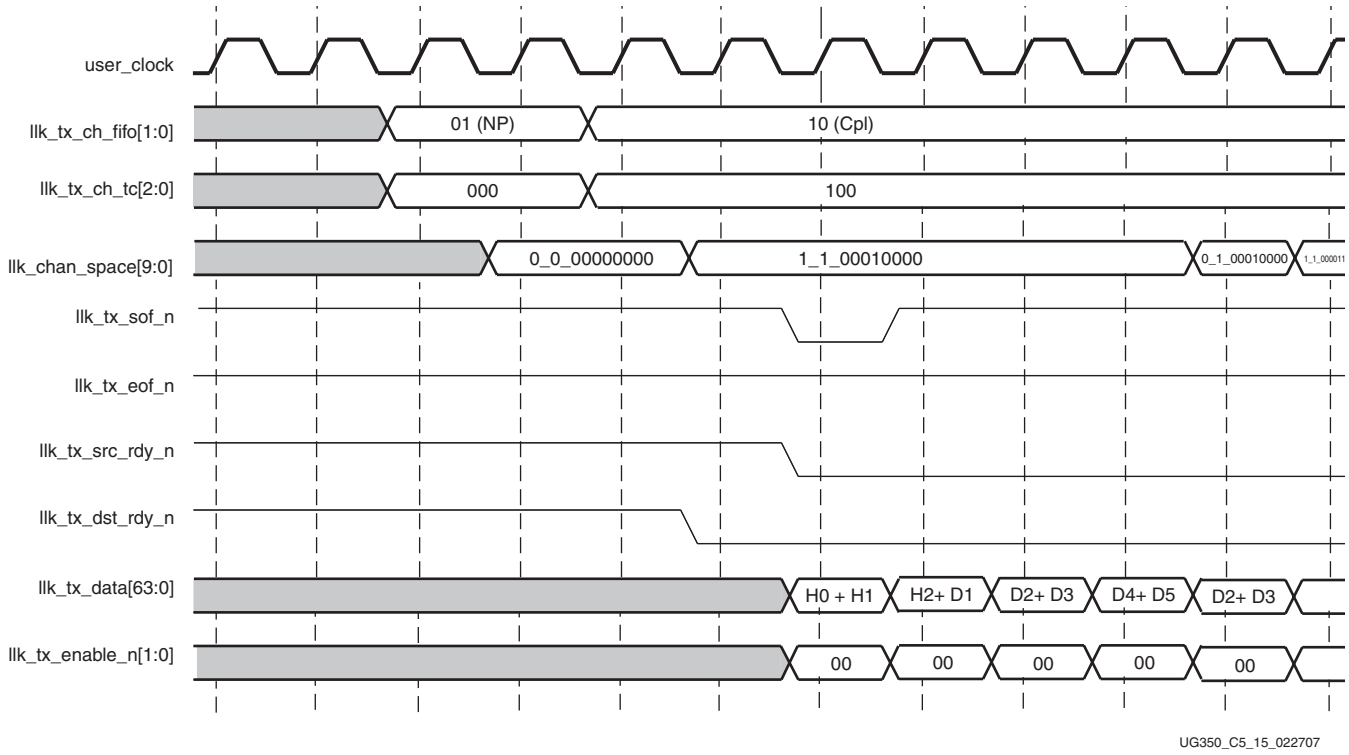


Figure 5-15: **Avoiding Blocking Posted and Completion TLPs**

It takes `llk_tx_chan_space[9:0]` four clock cycles to update as the transfer progresses. In other words, the information on `llk_tx_chan_space[9:0]` is four cycles behind the actual buffer status. Under most circumstances this is not a concern for typical applications. However, if the user application is attempting to be very granular with `llk_tx_chan_space[9:0]` it should be aware of this requirement.

Transferring Back-to-Back Packets and Channel Switching

The user can transfer packets back-to-back (`llk_tx_eof_n` on cycle *n* followed by `llk_tx_sof_n` on cycle *n*+1) as shown in Figure 5-16 as long as the same channel and traffic class is used for each TLP. Also the core must have enough buffer space available otherwise it stalls the transfer by deasserting `llk_tx_dst_rdy_n` as shown on the third TLP to be transferred in Figure 5-16. Notice that `llk_tx_dst_rdy_n` deasserted to stall the transfer of the third packet even though `llk_chan_space[9:0]` still indicated room was available for another packet. This is due to the four cycle delay in updating `llk_chan_space[9:0]` as discussed in "Channel Space Available." The user application must always monitor `llk_tx_dst_rdy_n` to know when the core can accept a transfer.

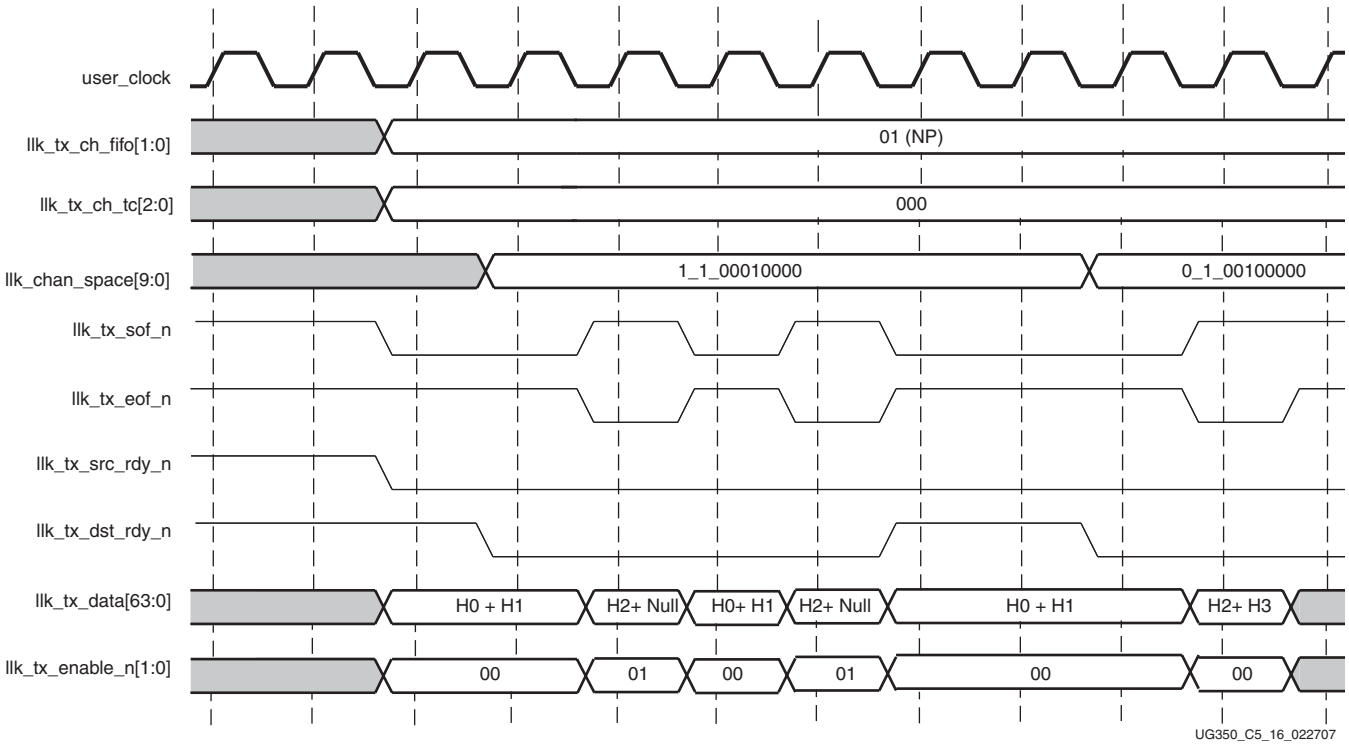


Figure 5-16: **Transferring Back-to-Back Packets on the Same Channel and Traffic Class**

Once the user application asserts `llk_tx_eof_n` marking the end of a TLP it must wait an additional two cycles of `llk_tx_dst_rdy_n` assertion before changing `llk_tx_ch_fifo[1:0]` or `llk_tx_ch_tc[2:0]`. Note that `llk_tx_dst_rdy_n` may deassert during this time as shown in Figure 5-17. Once the two cycle requirement is met, the user application can select a new channel and traffic class for the next TLP. It is not possible to transmit packets back-to-back without delay if a new channel or traffic class is selected.

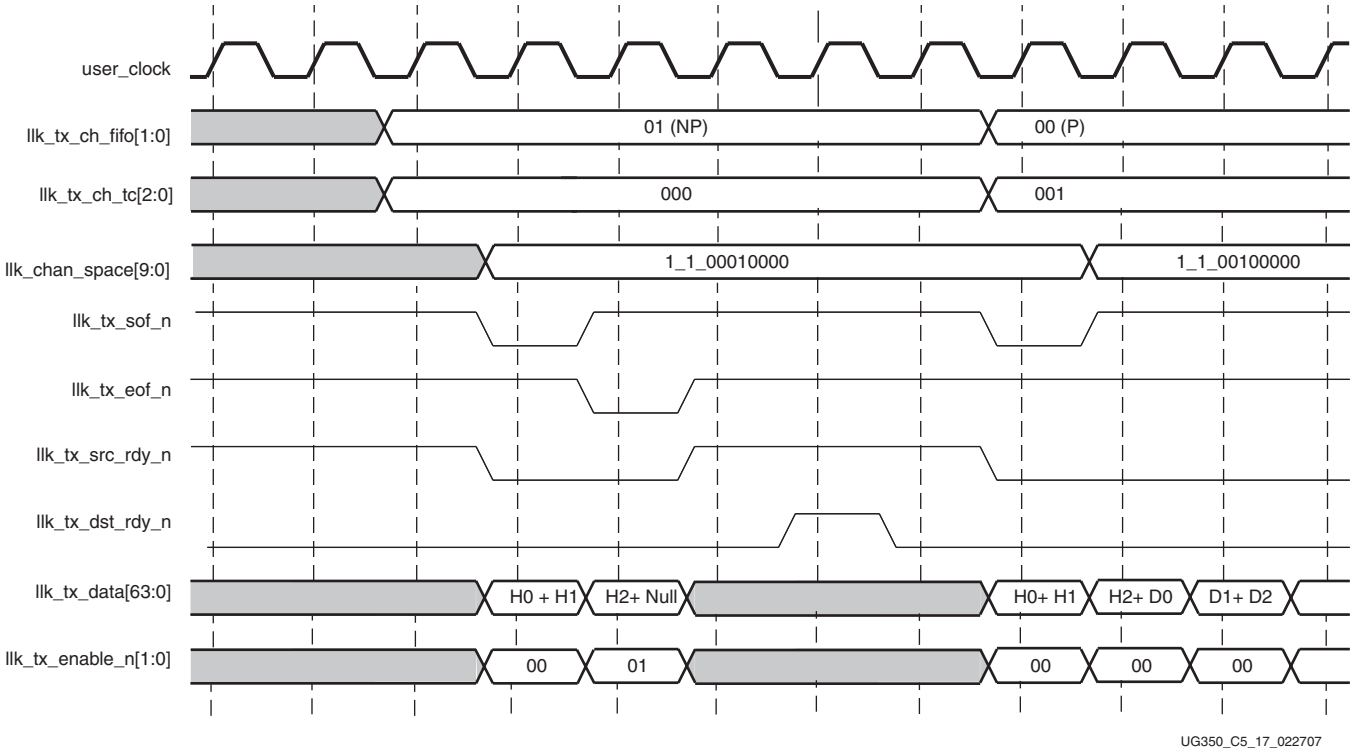


Figure 5-17: **Channel Switching on Transmit Interface**

Packet Data Poisoning Transaction Transmit Interface

To mark a TLP as poisoned, the user application, must set EP=1 in the TLP header. This can be done used if the payload is known to be poisoned when the first DWORD of the header is presented to the Endpoint core on the TRN interface.

Appending ECRCs (TLP Digest) 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 1-DWORD TLP Digest is properly computed and appended after the last valid TLP payload section (if applicable).

Receiving Inbound Packets

General TLP Receive Information

The integrated Endpoint block receiver logic is implemented as a store and forward type model, meaning the entire TLP is received by the core and stored in the receive buffer before being passed on to the user application.

The `llk_rx_preferred_type[15:0]` and `llk_rx_ch_*_available_n[7:0]` signals together indicate when incoming TLPs are available to the user application. When a packet has been received into the RX buffer and confirmed as valid, the core asserts the appropriate signal (Table 5-3) indicating a packet is ready to be delivered to the user application. Each signal shown in Table 5-3 is 8-bits, allowing for a one-hot encoded vector to inform the user the traffic class of the incoming packet. Along with the `llk_rx_ch_*_available_n[7:0]` signal assertion, the core asserts the appropriate bits on `llk_rx_preferred_type[15:0]` to further qualify the packet.

Table 5-3: **Channel Ready Signals**

Signal Name	Description
llk_rx_ch_posted_available_n[7:0]	Posted packet with traffic class available to user application.
llk_rx_ch_non_posted_available_n[7:0]	Non-Posted packet with traffic class available to user application.
llk_rx_ch_completion_available_n[7:0]	Completion packet with traffic class available to user application.

- A posted packet can be read when one of the `llk_rx_ch_posted_available_n[7:0]` bits is asserted
- A non-posted packet can be read when one of the `llk_rx_ch_non_posted_available_n[7:0]` bits is asserted and `llk_rx_preferred_type[15:0]` indicates the non-posted channel
- A completion packet can be read when one of the `llk_rx_ch_completion_available_n[7:0]` bits is asserted and `llk_rx_preferred_type[15:0]` indicates the completion channel or non-posted channel.

The `llk_rx_preferred_type[15:0]` bits are interpreted in pairs with bits [1:0] allocated to TC0, bits [3:2] to TC1 and so on as shown in Table 5-4.

Table 5-4: **Definition of llk_rx_preferred_type[15:0]**

Signal Name	Description
llk_rx_preferred_type[1:0]	TC 0
llk_rx_preferred_type[3:2]	TC 1
llk_rx_preferred_type[5:4]	TC 2
llk_rx_preferred_type[7:6]	TC 3
llk_rx_preferred_type[9:8]	TC 4
llk_rx_preferred_type[11:10]	TC 5

Table 5-4: **Definition of llk_rx_preferred_type[15:0] *(Continued)***

Signal Name	Description
llk_rx_preferred_type[13:12]	TC 6
llk_rx_preferred_type[15:14]	TC 7

Within each pair of bits the following definitions exist:

- 00: Posted
- 01: Non-posted
- 10: Completion
- 11: Reserved

The following are some examples of interpreting the `llk_rx_preferred_type[15:0]` and `llk_rx_ch_*_available_n[7:0]` signals.

- A Non-Posted TLP is available with traffic class of 011b when `llk_rx_ch_non_posted_available_n[3]` = 0 and `llk_rx_preferred_type[7:6]` = 01
- A Completion TLP is available with traffic class of `000b` when `llk_rx_ch_completion_available_n[0]` = 0 and `llk_rx_preferred_type[7:6]` = 01
- A posted packet is available when any bits of `llk_rx_ch_posted_available_n[7:0]` are asserted

The main purpose of `llk_rx_preferred_type[15:0]` is to ensure proper PCI Express transaction ordering rules are maintained by providing insight to the packets available in the core. The `llk_rx_preferred_type[15:0]` signal tells the user application which is the safest packet to drain. There might be multiple packet types available, but `llk_rx_preferred_type[15:0]` always indicates only one type of packet to drain from the core to ensure transaction ordering is not violated. For instance, if a posted packet has arrived earlier which was not drained, and a completion packet arrived later, then `llk_rx_preferred_type[15:0]` would indicate a posted packet. In such a scenario, the user application cannot drain the completion packet before draining the posted packet. But if the preferred type is shown as non-posted, then the user application can drain the completion packet. When packets are available in more than one traffic class, the user can choose to service the traffic classes in any order.

DWORD Enables

The user application receiver transaction layer interface data bus is 64 bits wide allowing the user to receive one QWORD of data from the core on a clock cycle. The PCI Express protocol allows DWORD or 32-bit alignment of the header and data. The `llk_rx_valid_n[1:0]` bus indicates which DWORD(s) contain valid header or data information. Bit 1 of `llk_rx_valid_n` maps `llk_rx_data[63:32]` and bit 0 of `llk_rx_valid_n` maps to `llk_rx_data[31:0]`. A value of 0 indicates the DWORD is valid.

The core transfers a complete QWORD of data to the user application on each clock cycle that `llk_rx_src_rdy_n` is asserted except on the last cycle of a TLP transfer indicated by `llk_rx_eof_n` = 0. For the last QWORD to be received into the user application, it is possible that only `llk_rx_data[63:32]` is valid since the PCI Express protocol allows for the header and data to be DWORD aligned. This is denoted by `llk_rx_valid_n[1:0]` = 01

during llk_rx_eof_n. Otherwise, llk_rx_valid_n[1:0] = 00 is used for all other cycles during a TLP transfer when llk_rx_src_rdy_n is asserted.

TLP Receiver Operation

Table 3-7, page 37 defines the user application receive signals. To receive a TLP, the receive user application must perform the following sequence of events on the Transaction receive interface:

1. The core, asserts the appropriate llk_rx_ch_posted_available_n[7:0], llk_rx_ch_non_posted_available_n[7:0], or llk_rx_completion_available_n[7:0] signal along with llk_rx_preferred_type[15:0] indicating a TLP is ready to be transferred to the user application.
2. The user application logic asserts llk_rx_ch_tc[2:0] to identify a traffic class and llk_rx_ch_fifo[1:0] to select a channel, either posted, non-posted, or completion based on what TLPs are waiting in the RX buffer.
3. The user asserts the llk_rx_dst_req_n signal to request data from the integrated Endpoint block.
4. When the core is ready to transfer data, the core asserts llk_rx_src_rdy_n with llk_rx_sof_n and presents the first complete TLP QWORD on llk_rx_data[63:0].
5. The core then deasserts llk_rx_sof_n, asserts llk_rx_src_rdy_n, and presents TLP QWORDs on llk_rx_data[63:0] for subsequent clock cycles, for which the user application logic asserts llk_rx_dst_req_n. For each clock where llk_rx_dst_req_n is asserted, the core asserts llk_rx_src_rdy_n for one clock after a minimum delay of 3 + TL_RAM_READ_LATENCY. The value of the TL_RAM_READ_LATENCY attribute is in the range [2 .. 6].
6. The core asserts llk_rx_src_last_req_n three clock cycles after it received the second to last request for the current RX packet on llk_rx_dst_req_n.
7. The core then asserts llk_rx_eof_n and presents either the last QWORD on llk_rx_data[63:0] and a value of 00h on llk_rx_valid_n[1:0] or the last DWORD on trn_td[63:32] and a value of 01h on llk_rx_valid_n[1:0].

Figure 5-18 illustrates a 3-DW TLP header without a data payload; an example is a 32-bit addressable Memory Read request. When the core asserts `llk_rx_eof_n`, it also places a value of `01` on `llk_rx_valid_n[1:0]` notifying the user application that only `llk_rx_data[63:32]` contains valid data.

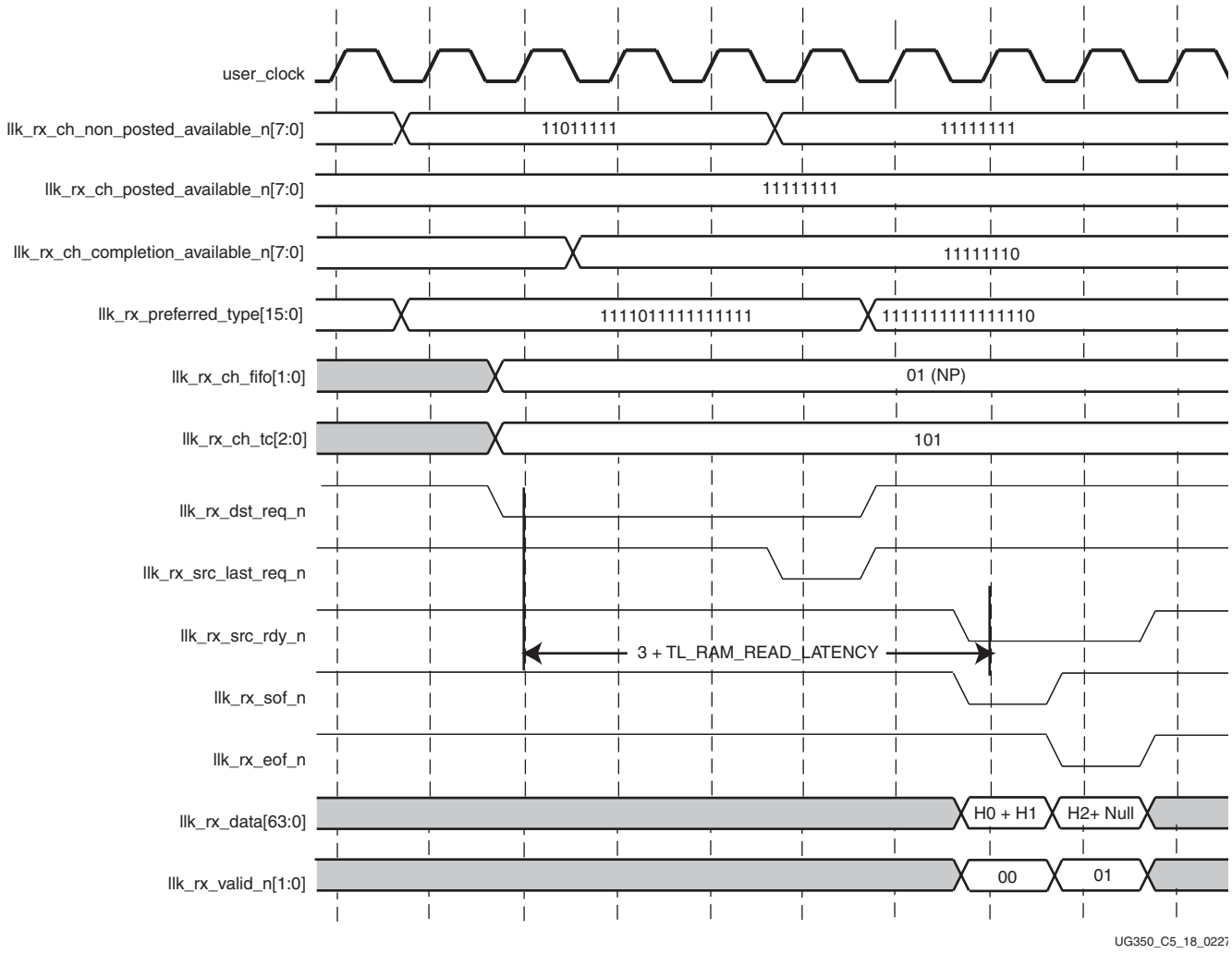


Figure 5-18: **TLP with 3-DW Header without Payload**

Figure 5-19 illustrates a 4-DW TLP header without a data payload; an example is a 64-bit addressable Memory Read request. When the core asserts llk_rx_eof_n, it also places a value of 00 on llk_rx_valid_n[1:0] notifying the core that llk_rx_data[63:0] contains valid data.

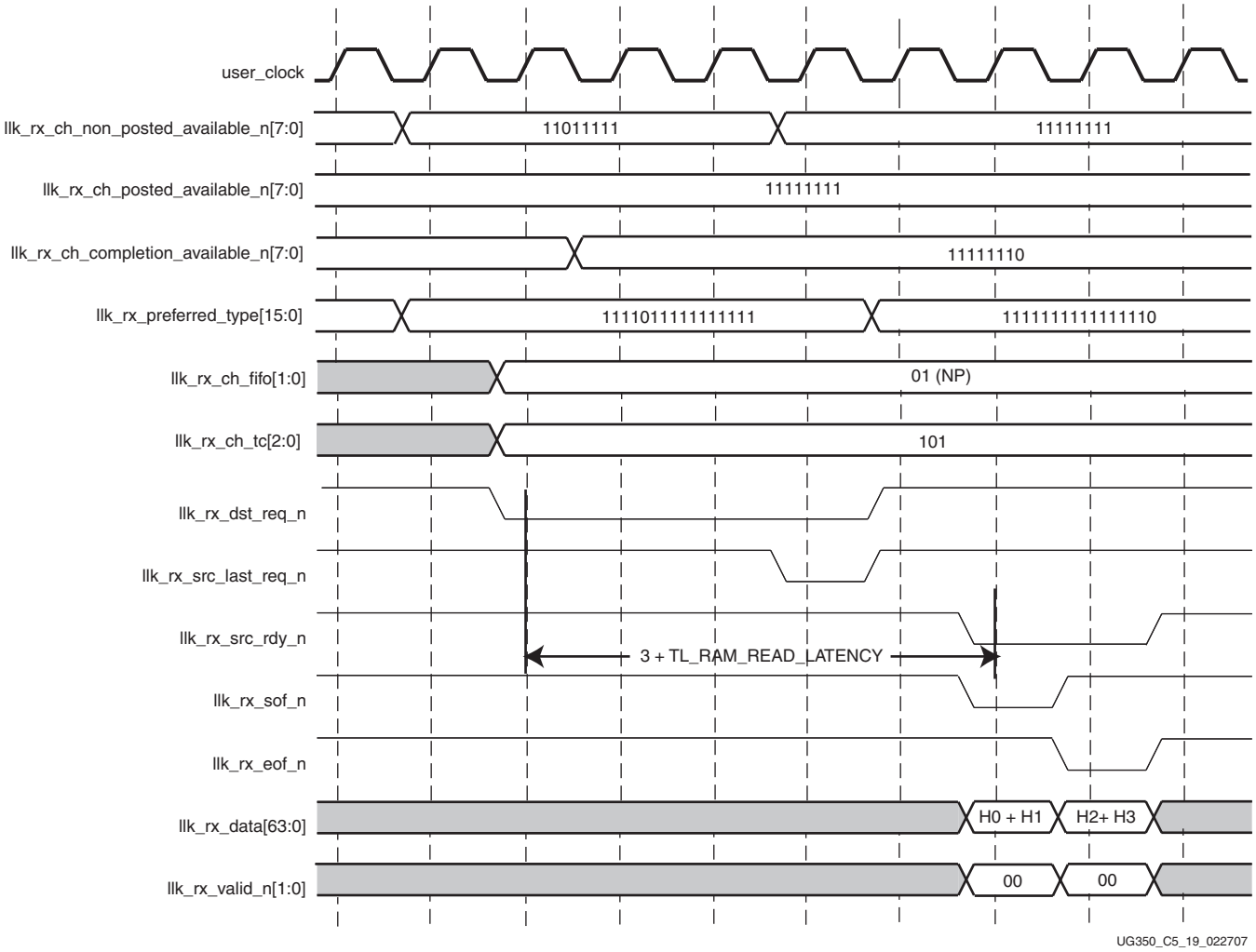


Figure 5-19: **TLP with 4-DW Header without Payload**

Figure 5-20 illustrates a 3-DW TLP header with a data payload; an example is a completion with data. When the core asserts llk_rx_eof_n, it also places a value of 00 on llk_rx_valid_n[1:0] notifying the core that llk_rx_data[63:0] contains valid data.

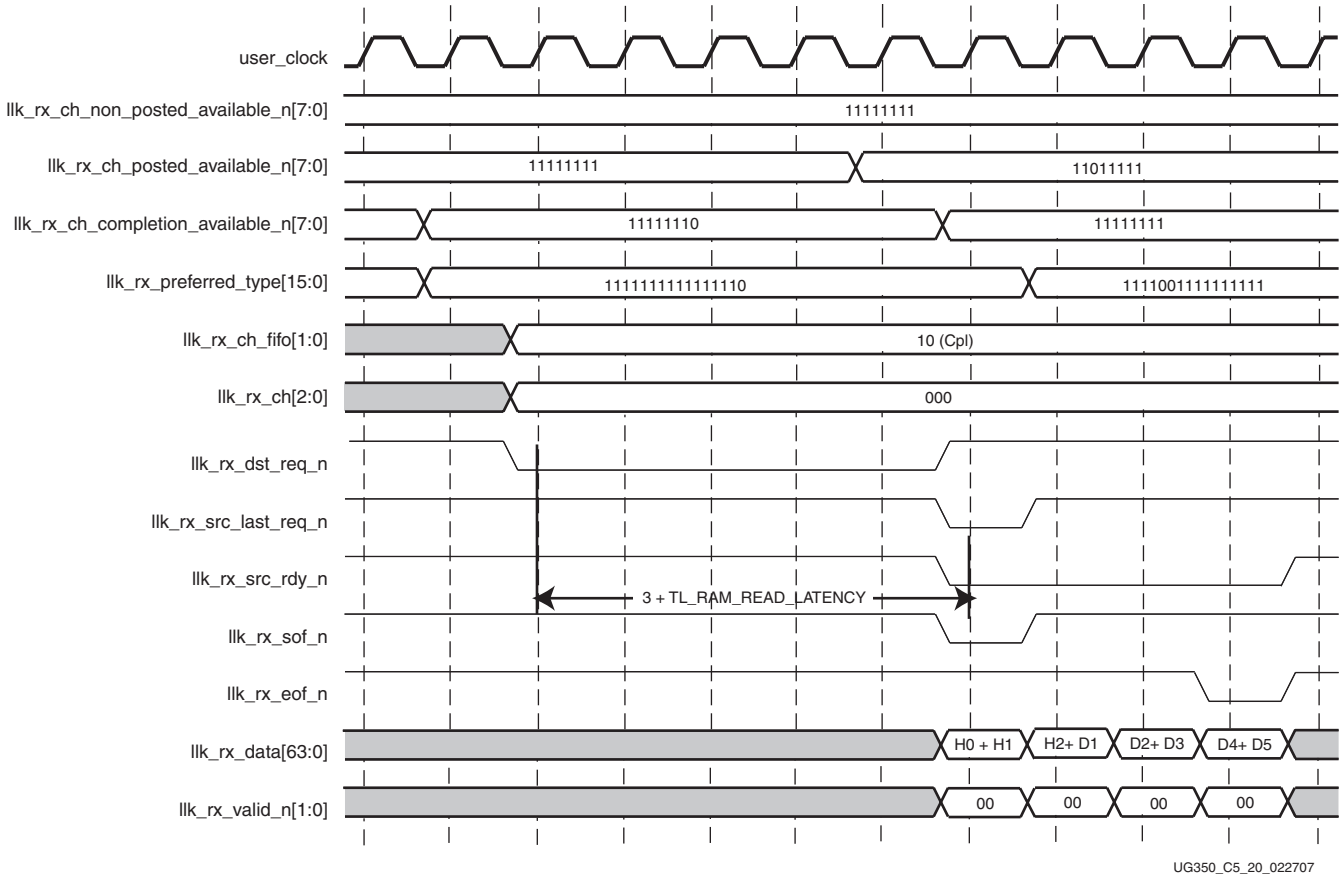


Figure 5-20: **TLP with 3-DW Header with Payload**

Figure 5-21 illustrates a 4-DW TLP header with a data payload; an example is a 64-bit addressable Memory Write request. Notice that in this figure both a posted and non-posted packet are available to the user application. Since the PCI Express ordering rules allow posted packets to bypass non-posted packets the user can choose to drain the posted packet before the non-posted even though `llk_rx_preffered_n[15:0]` indicates a non-posted packet is the first available per the PCIe ordering rules. When the core asserts `llk_rx_eof_n`, it also places a value of 01 on `llk_rx_valid_n[1:0]` notifying the core that `llk_rx_data[63:32]` contains valid data.

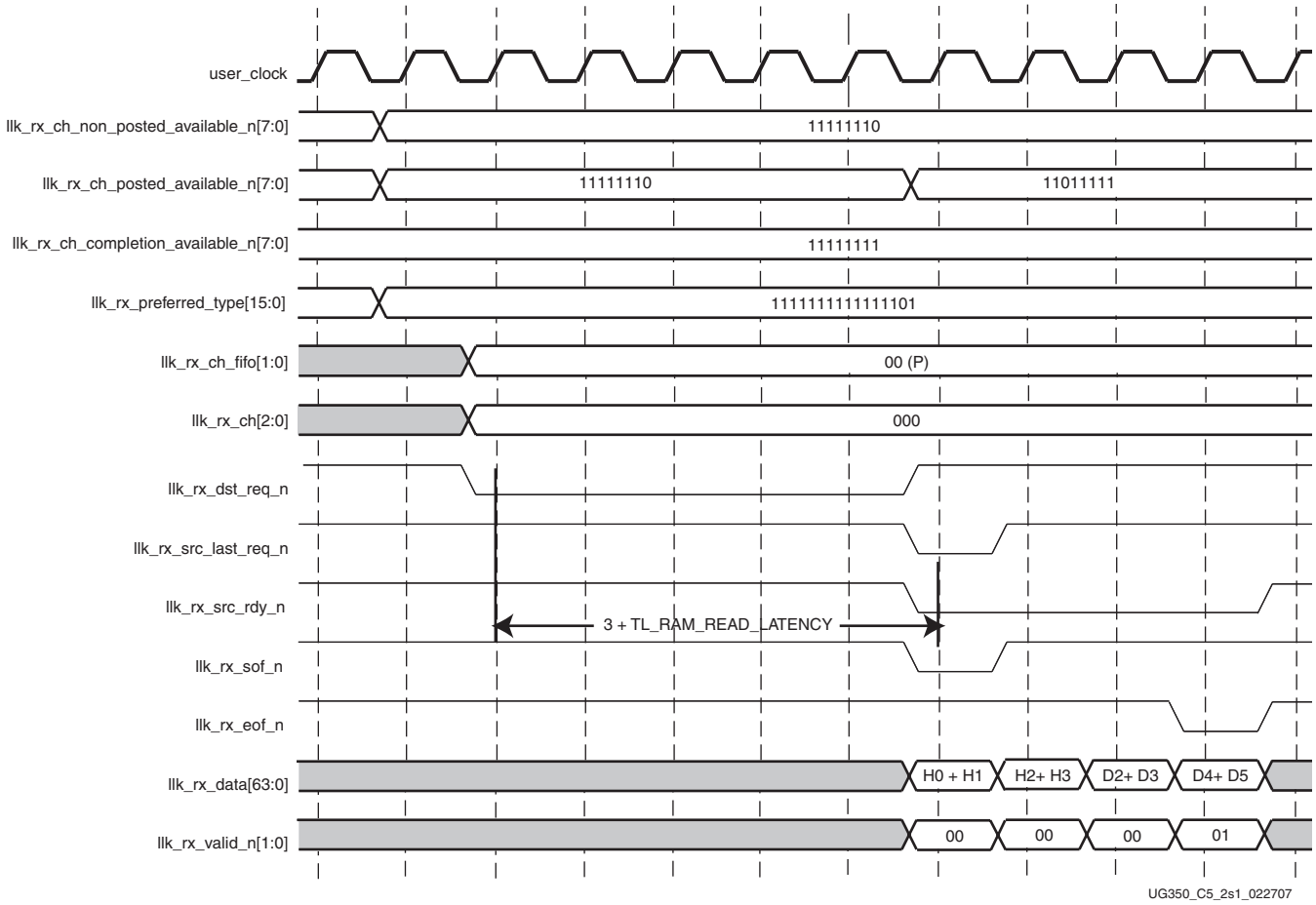


Figure 5-21: **TLP with 4-DW Header with Payload**

Requesting TLPs From the Core

After the user application determines a TLP is ready to be drained from the core based on the `llk_rx_ch_*_available[7:0]` and `llk_rx_preferred_type[15:0]` (as described in "General TLP Receive Information," page 95) the user must request the TLP from the core by asserting `llk_rx_dst_req_n`. For each clock where `llk_rx_dst_req_n` is asserted, the core asserts `llk_rx_src_rdy_n` for one clock after a minimum delay of 3 + TLRAMREADLATENCY as shown in Figure 5-18 through Figure 5-21. The value of the TLRAMREADLATENCY attribute is in the range [2 .. 6]. For more information on the TLRAMREADLATENCY attribute, see UG197.

The core outputs llk_rx_src_last_req_n three clock cycles after it has received the second-to-last request for the current RX packet via llk_rx_dst_req_n as shown in Figure 5-22. If llk_rx_dst_req_n is asserted in either of the two previous cycles before llk_rx_src_last_req_n, then the block received the final request for the current RX packet and no further requests should be issued unless there are further packets available on the selected channel as indicated by the corresponding llk_rx_ch_*_available[7:0] signal (where * is posted, non_posted, or completion).

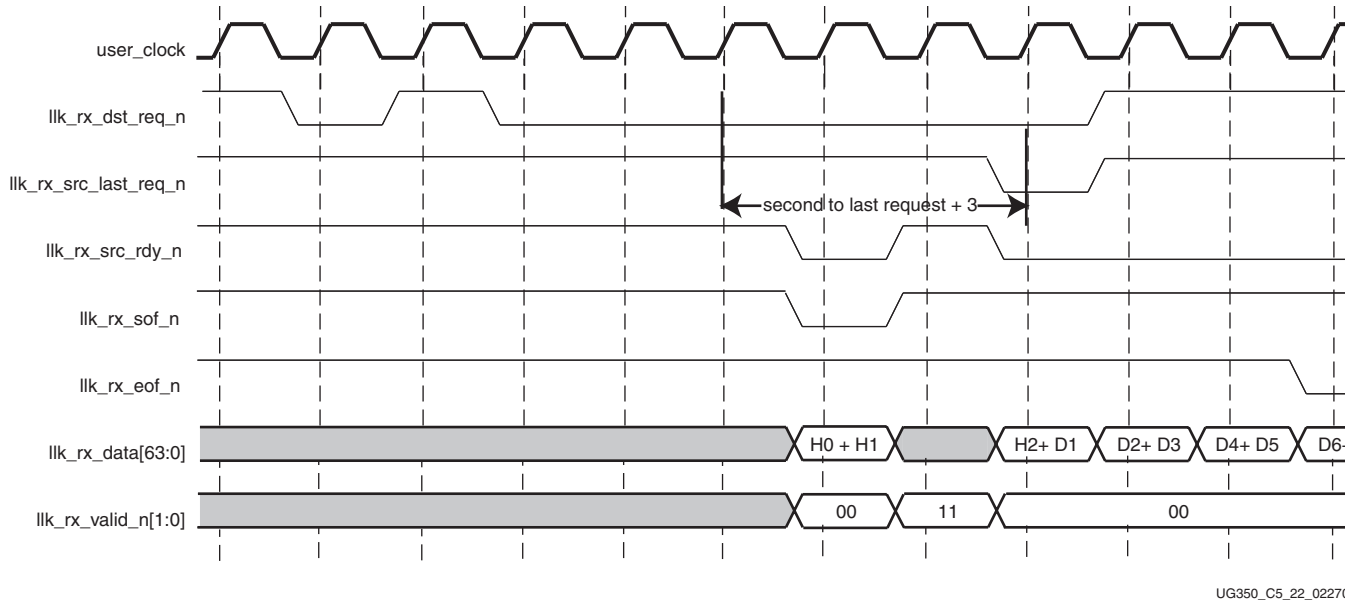


Figure 5-22: **Requesting Packets on the Transaction Receive Interface**

If llk_rx_dst_req_n is deasserted in both the previous two cycles before llk_rx_src_last_req_n is asserted, then one more request is required to complete reception of the current packet. To complete reception, the user must assert llk_rx_dst_req_n for one cycle, either in the same cycle as llk_rx_src_last_req_n is first asserted or in any subsequent cycle as shown in Figure 5-23. llk_rx_src_last_req_n remains asserted until three cycles after the core has received the final request for the current RX packet.

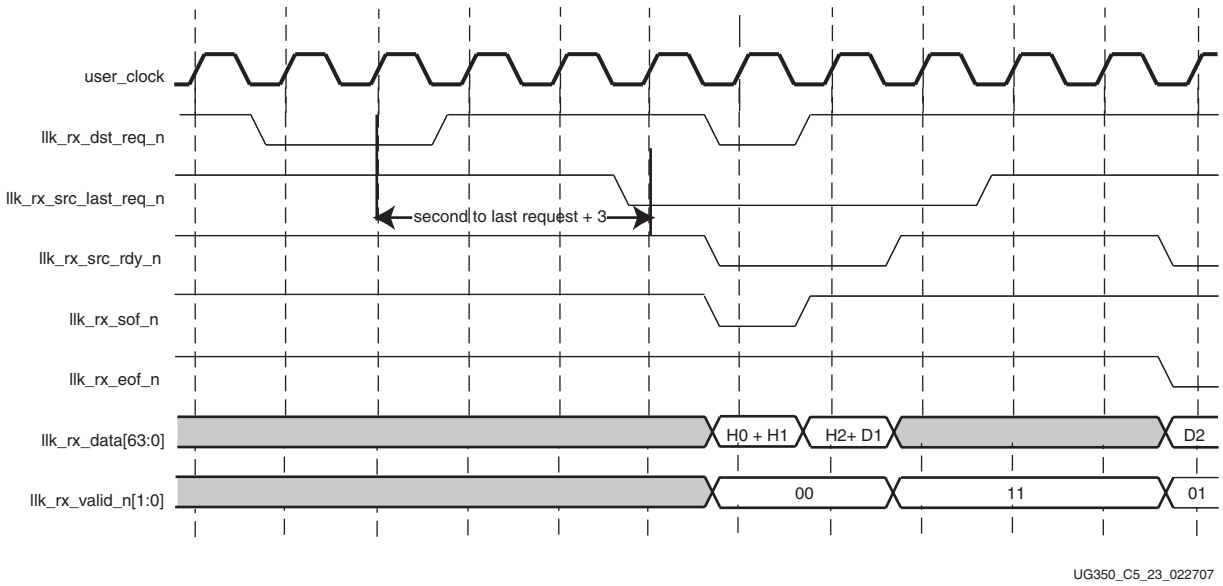


Figure 5-23: **Transaction Receive Interface Timing of a TLP with 3 DWORD Header and 2 DWORD Payload**

Note:

When the integrated block receives a TLP with an illegal payload length, it does not send an ERR_FATAL message. Instead, the block detects this as a bad LCRC and sends back a NAK. Occurrence of this case is very rare.

Throttling the Data Path on the Receiver Interface

The user application can stall the transfer of data from the core at any time by deasserting llk_rx_dst_req_n. By doing so the user application is pausing the transfer by not requesting more data. For each clock where llk_rx_dst_req_n is asserted, the core asserts llk_rx_src_rdy_n for one clock after a minimum delay of 3 + TLRAMREADLATENCY. The value of the TLRAMREADLATENCY attribute is in the range [2 .. 6]. Due to this latency, the user application should note that deasserting llk_rx_dst_req_n does not result in an immediate throttling of data on the receive data path. The resulting throttling of data occurs 3 + TLRAMREADLATENCY clock cycles after the cycle llk_rx_dst_req_n is deasserted. Figure 5-24 shows using llk_rx_dst_req_n to request data from the core.

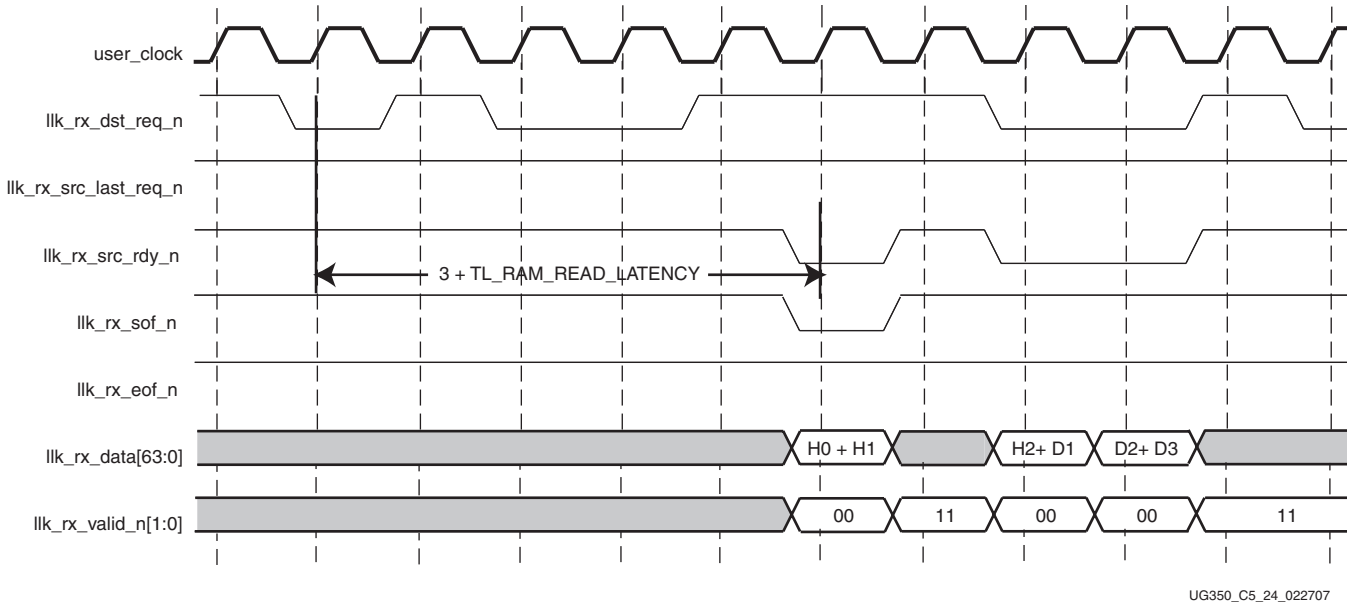


Figure 5-24: **Throttling the Receiver Data Path**

Managing Receiver Buffer Space for Inbound Completions

PCI Express Endpoints 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 PCIe Endpoint is required to advertise infinite storage credits for Completion Transactions in its receivers. The actual completion buffer inside the core is finite in size even though the endpoint is required to advertise infinite completion credits. The completion buffer must be sized to hold nine complete TLP completion packets. The number of bytes should be 9 * (24 + MPS) where MPS is Maximum Payload Size. Endpoints must internally manage Memory Read Requests transmitted upstream and not overflow the receiver when the corresponding Completions are received. The transmit user application must track Completions received for previously requested Non-Posted Transactions to modulate the rate and size of non-posted requests and stay within the instantaneous Completion space available in the Endpoint receiver.

Note: When MPS = 4096 bytes, the requirement of 9*(24+MPS) conflicts with the maximum possible size of the RX completion buffer. In this scenario, the GUI automatically sets the size to 8*(24+MPS) and enables Completion Flow Control (disables Infinite Completions mode). However, this results in a non-compliant core.

Accessing the Configuration Space Registers

The core allows the user to access the configuration space through the Management Port. This allows the user application to read and write information to the configuration space of the endpoint block. The PCI Express configuration header, capabilities, and extended capabilities are also included within the endpoint blocks configuration space. When reading and writing to the configuration registers, the DWORD address offsets must be used. For most of these registers, the value does not match the standard configuration space address defined by the PCIe specification. However, from the system's perspective the PCIe Configuration Space of the core does follow standard PCIe addressing and the necessary internal address translations are handled by the block when it receives configuration accesses from an upstream port.

Certain information contained within the configuration space is mapped directly to the user application. See Table 3-3, page 29 for more information.

The Management Interface address bus uses DWORD addressing. A typical processor bus has byte addressing, where the lower two bits of the address bus indicate which byte in a DWORD is accessed. To connect the Management Interface to a processor bus, the lower two bits of the processor address are decoded with user logic to generate the byte write enables for the Management Interface.

Done Signals and BAR Monitoring

The signals mgmt_rddone and mgmt_wrdone indicate when a management interface read or write completed successfully. The data is valid on the same cycle that the done signals go high. The latency between when a read or write is requested to the time the done signals assert vary depending on if BAR monitoring is enabled. See section Table 4-3, page 50 for how to enable BAR hit monitoring.

Bar Hit Monitoring Disabled

When BAR hit monitoring is disabled:

- mgmt_rddone will go high three clock cycles after the read request
- mgmt_wrdone will go high one clock cycle after the write request

Bar Hit Monitoring Enabled

When BAR hit monitoring is enabled a polling sequence continually operates on the management interface. When the user applications wants to make a request, it gets priority over the continuing polling sequence. The polling sequence is paused safely and resumed after completion of user request. The polling of single register takes three clock cycles. Depending on when the user makes a request:

- mgmt_rddone will go High anywhere between three to six clock cycles after request is made
- mgmt_wrdone will go High anywhere between one to four clock cycles after the request is made

Reading Configuration Registers

The interface has separate 32-bit data read and write buses. To read from the configuration space the user application performs the following steps.

1. The user application logic places a DWORD address on `mgmt_addr[10:0]` and asserts `mgmt_rden` for one `user_clk` cycle. If BAR LOGIC is enabled mgmt_addr should hold its value until mgmt_rddone.
2. Three to six cycles later the user application can capture valid data from `mgmt_rdata[31:0]` when mgmt_rddone asserts high.

Figure 5-25 shows the read timing on the management interface port.

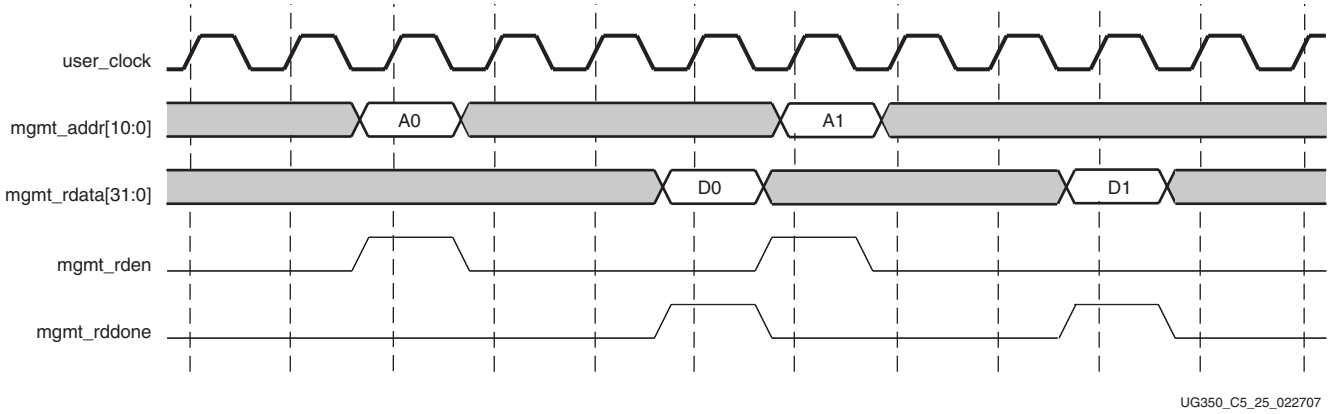


Figure 5-25: **Management Interface Read Timing**

Note:

The core should treat access to an unimplemented configuration space as an unsupported request per the *PCI Express Base Specification, v1.1*. The integrated Endpoint block responds with a successful completion which is non-compliant. However, as the upstream component is not expected to access unimplemented configuration access, this has no impact on safe operation.

Writing Configuration Registers

The user application can write to any register marked as R/W (read/writable) (see UG197). To use the Management interface to override an attribute (for example, DEVICEID), the integrated Endpoint block must be held in reset during and for at least four cycles after performing a Management write to the attribute address. This must be done to allow the new value to propagate within the integrated Endpoint block. The user application must keep user_reset_n asserted for an additional four user_clock cycles after the assertion of mgt_wren to allow the new value to propagate within the integrated Endpoint block. The following steps illustrate writing to a configuration space register.

1. The user application logic places a DWORD address on mgmt_addr[10:0] along with the appropriate byte enables on mgmt_bwren[3:0] and asserts mgmt_wren for one user_clk cycle. If BAR LOGIC is enabled, mgmt_wdata, mgmt_bwren, and mgmt_wren should hold their values until mgmt_wrdone.
2. Once the write has completed, mgmt_wrdone asserts.

Figure 5-26 shows the write timing on the management interface port.

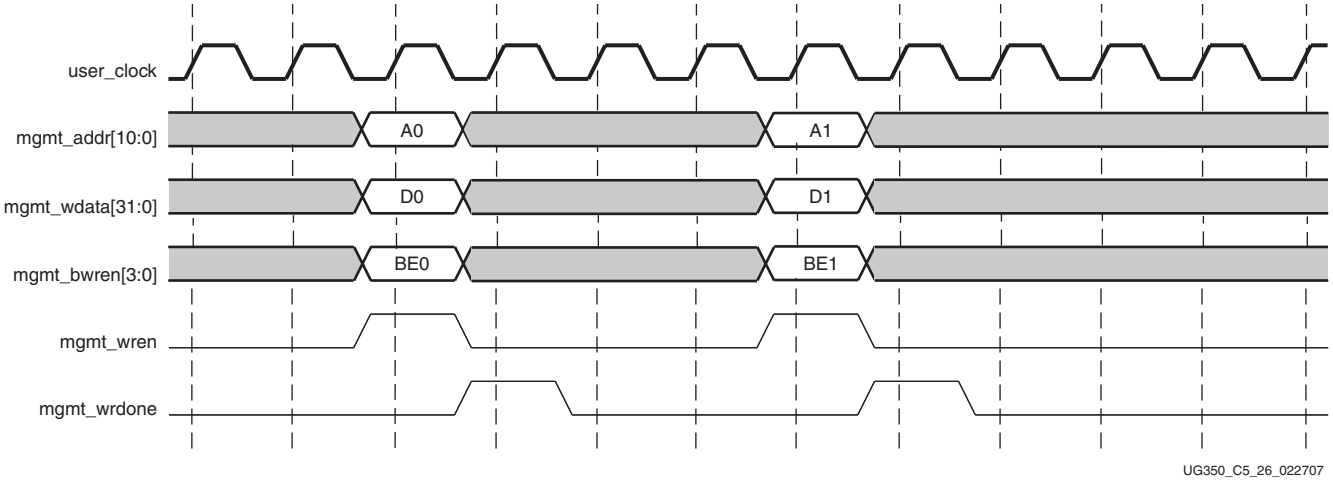


Figure 5-26: **Management Interface Write Timing**

The byte enable signal, mgmt_bwren[3:0], are active-High and used to enable a single byte within the write DWORD. The mapping is as follows:

```
mgmt_bwren[0] => mgmt_wrdata[7:0]
mgmt_bwren[1] => mgmt_wrdata[15:8]
mgmt_bwren[2] => mgmt_wrdata[23:16]
mgmt_bwren[3] => mgmt_wrdata[31:24]
```

Reporting User Error Conditions

The user application is responsible for detecting and reporting certain types of error conditions. For error handling required by the user application, refer to the Error Detection section in UG197.

The user application must report errors that occur during Completion handling using dedicated error signals on the Configuration and Status 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 by transmitting an error message on the PCI Express link and also sets the appropriate status bit(s) in the Configuration Space. The type of error-reporting messages transmitted depends on whether or not the error resulted from a Posted or Non-Posted Request. All signals dedicated to user-reported errors will cause Message packets 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 should be handled by the user application by assembling and transmitting completion packets with non-successful status on the Transaction Layer Interface which is sent to the original requester. In addition, the user application must also assert the appropriate error signals on the Configuration and Status Interface that is dedicated to setting the error bits in the configuration space and transmitting error messages.

The various error reporting signals are described in Table 3-3, page 29 in the "Configuration and Status Interface" section. Each signal, when used, should be asserted for a single clock cycle. The appropriate error messages will be sent and status bits will be set in response to these signals. Three general error reporting signals exist for cases which are not covered by other error reporting signals. These are:

- l0_set_detected_corr_error
- l0_set_detected_fatal_error
- l0_set_detected_nonfatal_error

Asserting any of these signals causes the appropriate bits to be set in the Device Status Register. In addition, the associated error message will also be sent assuming the message enable bits are set in the Device Control Register.

Generating Interrupt Requests

The core 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. If the MSI Enable bit is set to a `1`, then the user application may generate MSI request by creating and sending Memory Write TLPs on the cores transmit user interface. 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`:

- interrupt_disable core output = 0: interrupts enabled
- interrupt_disable core output = 1: interrupts disabled (requests are blocked by the core)

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. The user application must poll the MSI Enable bit to ensure the Root Complex has enabled the device to send MSI packets. See "Accessing the Configuration Space Registers," page 105 for more information on reading the core's internal configuration registers.

MSI Mode

The core supports up to four multiple MSI messages. Users can enable one, two, or four messages through the GUI. Please see Table 4-5, page 56 in the "Extended Capability Configuration Menu" section for more information on setting up multiple MSI messages. Once the system powers up the Root Complex will optionally enable up to four messages depending on the settings entered in the GUI. This information is contained in the MSI capability structure's Message Control register. The core supports the 64-bit Message Capability Structure as shown in Figure 5-27. Please see section 6.8 of the *PCI Local Bus Specification v3.0* for more information on the MSI capability structure.

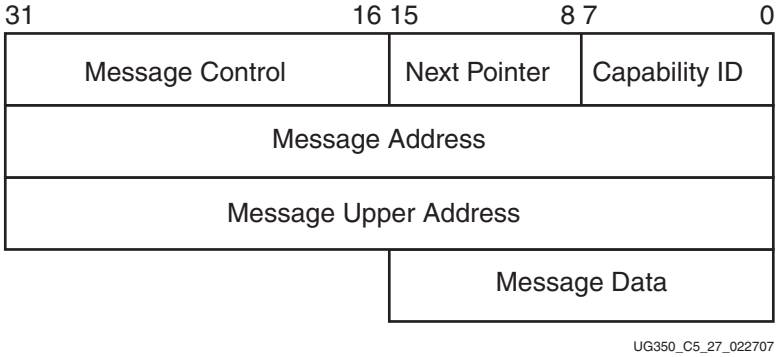


Figure 5-27: **MSI Capability Structure**

To send a MSI packet the user application must form the packet on the transmit user application interface. MSI packets are formed as either memory write 32-bit addressable packets or memory write 64-bit addressable packets. The user must choose either the 32-bit or 64-bit addressable packet depending on the contents of the Message Upper Address register of the MSI capability structure. If this register has a non-zero value then 64-bit addressable memory write is used, otherwise a 32-bit addressable memory write is used. MSI packets are identical to normal memory write packets except the *PCI Express Base Specification* specifies that the attribute bits of the transaction description field must be set to 00b. This imposes default ordering and hardware enforced cache coherency on the packet. Please see section 2.2.6.4, 2.2.6.5 and 6.1 of the *PCI Express Base Specification v1.1* for more information on MSI packets.

To send the MSI packet the user must poll the MSI capability structure using the Management Interface to obtain the necessary information of the packet. Once retrieved the packet is transmitted like any normal memory write packet as described in the section **"Transmitting Outbound Packets," page 79**.

Legacy Interrupt Mode

The core supports legacy interrupt mode if MSI interrupt mode is not being used. Legacy interrupts use inband messages to assert and deassert virtual interrupt lines on the link to emulate the legacy PCI interrupt pins. Please see section 6.1 of the *PCI Express Base Specification v1.1* for more details. To assert an interrupt the ASSERT_INTx message is sent and to deassert the interrupt the DEASSERT_INTx message is sent. The core supports only legacy interrupt pin A or INTA which is common practice in endpoint applications.

The core input l0_legacy_int_funct0 is used to send the ASSERT_INTA and DEASSERT_INTA messages. To send the ASSERT_INTA message the user drives l0_legacy_int_funct0 from low to high for at least one clock cycle. Once the user application is ready to send the DEASSERT_INTA message, the user drives the l0_legacy_int_funct0 signal from high to low as shown in Figure 5-28.

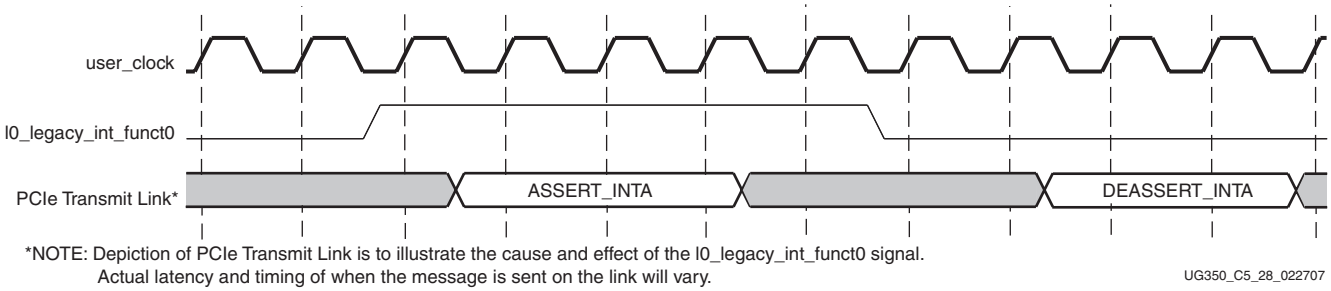


Figure 5-28: **Sending Legacy Interrupts**

Link Training: 2-Lane, 4-Lane, and 8-Lane Cores

The 2-lane, 4-lane and 8-lane cores are capable of operating at less than the maximum lane width as required by the *PCI Express Base Specification*. Two cases that cause the core to operate at less than its specified maximum lane width are described below.

Upstream Partner Supports Fewer Lanes

When the 2-lane or 4-lane core is connected to an upstream device that implements 1 lane, the 2-lane or 4-lane core trains and operates as a 1-lane device using lane 0, as shown in Figure 5-29. Similarly, if the 4-lane core is connected to a 2-lane upstream device, the core trains and operates as a 2-lane device using lanes 0 and 1.

When the 8-lane core is connected to an upstream device that only implements 4 lanes, it trains and operates as a 4-lane device using lanes 0-3. Additionally, if the upstream device only implements 1 or 2 lanes, the 8-lane core trains and operates as a 1- or 2-lane device.

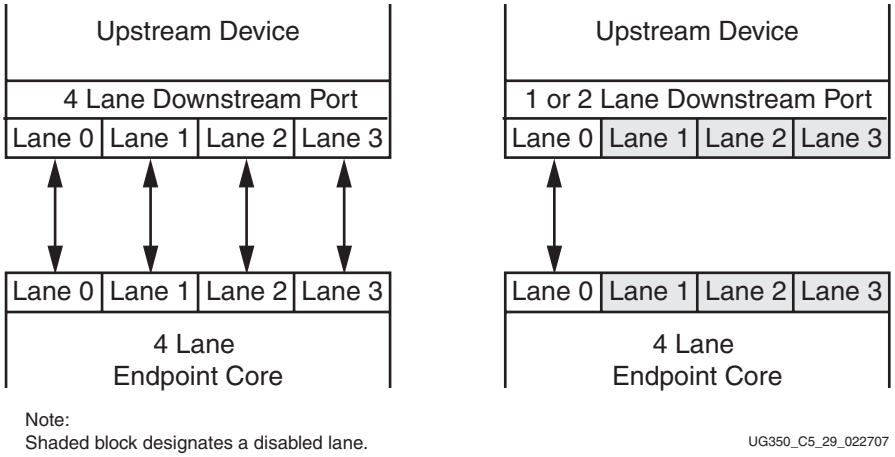


Figure 5-29: **Scaling of 4-lane Endpoint Core from 4-lane to 1-lane Operation**

Lane Becomes Faulty

If a link becomes faulty after training to the maximum lane width supported by the core and the link partner device, the core attempts to recover and train to a lower lane width, if available. If lane 0 becomes faulty, the link is irrecoverably lost. If any or all of lanes 1-7 become faulty, the link goes into recovery and attempts to recover the largest viable link with whichever lanes are still operational.

For example, when using the 8-lane core, loss of lane 1 yields a recovery to 1-lane operation on lane 0, whereas the loss of lane 6 yields a recovery to 4-lane operation on lanes 0-3. After recovery occurs, if the failed lane(s) becomes alive again, the core does not attempt to recover to a wider link width. The only way a wider link width can occur is if the link actually goes down and it attempts to retrain from scratch.

Clocking and Reset of the Core

Reset

The core uses `user_reset_n` to reset the system, an asynchronous, active-low reset signal asserted during Fundamental Resets. 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 user_reset_n. For endpoint applications that do not have a sideband system reset signal, the initial hardware reset should be generated locally. Three PCI Express reset events can occur:

- **Cold Reset** - A fundamental reset that occurs at the application of power. The `user_reset_n` signal is asserted to cause the cold reset of the integrated Endpoint block.
- **Warm Reset** - A fundamental reset triggered by hardware without the removal and re-application of power. The `user_reset_n` signal is asserted to cause the warm reset to the integrated Endpoint block.
- **Hot Reset** - In-band propagation of a reset across the PCIe link through the protocol. The `user_reset_n` signal is not used in this case.

The integrated Endpoint block requires reset logic in the user application for correct operation. The required reset logic depends on the RESETMODE attribute setting. The core contains a reset module file, `pcie_reset_logic.v[hd]`. See "Reset Module," page 71 for more information on this module. The reset module logic provided in the core implements the required reset control logic for the integrated Endpoint block. For more information on how the reset domains operate, refer to the Clocking and Reset Interface section of UG197.

Note: Systems designed to the PCI Express electro-mechanical specification provide a sideband reset signal, which uses 3.3V signaling levels—read the FPGA device data sheet carefully to understand the requirements for interfacing to such signals.

Clocking

The core requires a 100 MHz, 125 MHz or 250 MHz system reference clock input. The clock frequency used must match the clock frequency selection in the GUI.

The integrated Endpoint block has two synchronous clock domains: core_clk and user_clk. Each clock domain is controlled by different inputs on the integrated Endpoint block as discussed in the Clocking and Reset Interface section of UG197. The user application must provide the clock connections as required in UG197 for the integrated Endpoint block.

The core contains a clock module file, pcie_clocking.v[hd]. See "Clock Module," page 70 for more information on this module. The clock module logic provided by the core creates the required clocks based on the input reference clock frequency setting in the CORE Generator software. The clocks are then connected to the integrated Endpoint block in the `pcie_top.v[hd]` file also contained in the core. The user does not need to modify these files under most circumstances, however, source code for each is provided.

Synchronous and Non-Synchronous Clocking

There are two ways to clock the core:

- 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: Xilinx recommends that designers use synchronous clocking when using core. All add-in card designs must use synchronous clocking due the characteristics of the provided reference clock. See UG196: *Virtex-5 GTP Transceiver User Guide*, UG198: *Virtex-5 GTX Transceiver User Guide*, and the 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 convert the 100 MHz provided clock to 125 MHz or 250 MHz. 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.

Chapter 6

Simulating with the Core

This chapter describes how to simulate the core and provides an example on how to set the system environment to run simulations

Simulation Environment

The simulation of designs containing the integrated Endpoint block has specific prerequisites that the simulation environment must fulfill.

The Synthesis and Simulation Design Guide explains how to setup the simulation environment for various Xilinx supported simulators. This design guide can be downloaded from the Xilinx website at:

http://toolbox.xilinx.com/docsan/xilinx82/books/docs/sim/sim.pdf

Prerequisites for the simulating designs containing the integrated Endpoint block:

- Using a simulator with a SWIFT interface to support SmartModels
- Installing SmartModels for the integrated Endpoint block, the GTP_DUAL tiles, and the GTX_DUAL tiles
- Setting the environment variable to point to the SmartModel installation directory
- Modifying simulator setup to use the SWIFT interface (initialization file, and environment variables)
- Compiling SmartModel wrapper files into the UNISIM and SIMPRIM libraries using compxlib facility (Compiling Xilinx Libraries)

Before proceeding, make sure all the prerequisites are met. The user guide of the simulator and the Synthesis and Simulation Design Guide provide a detailed description on setting up the various simulators for SmartModel support.

Simulation Setup for ModelSim SE Simulator

This section provides an example on how to set up the system environment to run a simulation on the Linux operating system using ModelSim SE simulator, the HDL simulator from Mentor Graphics.

The example uses ModelSim SE 6.2d and ISE 9.2i (Service Pack 2) run on a Linux platform.

1. The following environment variables are set using **`setenv`**:

- XILINX: location of the installed Xilinx ISE software
- MODEL_TECH: location of the installed ModelSim simulator version 6.2d
- MODELSIM: $MODEL_TECH/modelsim.ini
- LMC_HOME: $XILINX/smartmodel/lin/installed_lin

- LMC_CONFIG: $LMC_HOME/data/linux.lmc
- LD_LIBRARY_PATH: $LMC_HOME/lib/linux.lib:$LD_LIBRARY_PATH

If you are not a system administrator and do not have write permissions to the above directories, set the environment variables as follows

- XILINX: location of the installed Xilinx ISE software
- MODEL_TECH: location of the installed ModelSim simulator version 6.2d
- MODELSIM: copy the modelsim.ini file $MODEL_TECH/modelsim.ini to a local location Example:/home/user/modelsim and set variable MODELSIM to: /home/user/modelsim
- LMC_HOME: set variable to a local location Example:/home/user/modelsim/sim_libs
- LMC_CONFIG: $LMC_HOME/data/linux.lmc
- LD_LIBRARY_PATH: $LMC_HOME/lib/linux.lib:$LD_LIBRARY_PATH

2. The initialization file (modelsim.ini) is modified to support the SWIFT interface:

- The simulator resolution is set to 1 ps
- The Path Separator setting for simulator commands is removed
- Variable **`Veriuser`** is set to $MODEL_TECH/libswiftpli.sl to allow dynamic loading of objects for Verilog PLI (Programming Language Interface) applications
- Variable **`libsm = $MODEL_TECH/libsm.sl`** and variable **`libswift = $LMC_HOME/lib/linux.lib/libswift.so`** are set to load logic modeling the SWIFT SmartModel software

3. The simulation libraries (UNISIM, UNISIMS_VER, SIMPRIM, SIMPRIMS_VER, XILINXCORELIB, and XILINXCORELIB_VER) and SmartModel binaries are compiled using COMPXLIB tool provide by XILINX as a part of ISE software. Before running the compxlib command, make sure variables $XILINX, $MODELSIM and $LMC_HOME are properly set. A user with administrator permissions can run COMPXLIB with the following options:

- **`compxlib -s mti_se -l all  -arch virtex5 -smartmodel_setup`**

These options compile Virtex-5 device family libraries for all languages supported by ModelSim SE 6.2d HDL simulator. The above example also updates the modelsim.ini file to include the paths to all the simulation libraries generated. By default, the output directory created for storing the libraries is `$XILINX/language/target_simulator`.

- VHDL libraries are stored in `$XILINX/vhdl/mti_se`
- Verilog libraries are stored in `$XILINX/verilog/mti_se`

A user with out administrator permissions can run compxlib with a **`-dir`** option to specify a local directory (The location can be the same as $LMC_HOME) in which the libraries will be compiled

- **`compxlib -s mti_se  -arch virtex5 -lib all -l all -dir <user_specified_directory> -log compxlib.log  -smartmodel_setup`**

The simulation libraries will compiled in the following location

- VHDL UNISIM library: `<user_specified_directory>/unisim`
- VHDL SIMPRIM library: `<user_specified_directory>simprim`
- VHDL XILINXCORELIB: `<user_specified_directory>/XilinxCoreLib`

- Verilog UNISIM: `<user_specified_directory>/unisims_ver`
- Verilog SIMPRIM: `<user_specified_directory>/simprims_ver`
- Verilog XILINXCORELIB: `<user_specified_directory>/XilinxCoreLib_ver`
- SmartModels are installed in `$LMC_HOME`

Simulating the Memory Endpoint Reference Design

This section describes the basic functionality of the Memory Endpoint Reference design, which is delivered along with the core. The core and a Memory Endpoint application together form the Memory Endpoint Reference Design. The test bench provided exercises this reference design and serves as an example of how to simulate the core using Modelsim 6.2d simulator.

In order to demonstrate the integrated Endpoint block in operation, the CORE Generator tool generates a Memory Endpoint Reference Design which includes the core. The Memory Endpoint application primarily contains a memory bank and a completer state machine. The application communicates with the core through the Transaction Layer Interface (TLI). The main function of the application is to process memory read/write requests and return completions for read requests.

The completer continuously polls the TLI to determine if there are any TLP packets to be processed. If TLP packets are available it sends a request to the RX interface of the core to drain TLPs. The core acknowledges the request and follows it with a TLP packet (Refer to Chapter 2, section Transaction Layer Interface of *PCI Express Endpoint Block User Guide* for further details). If the TLP is a memory write request, the payload of the TLP is drained from the RX interface and is stored in the memory bank of the application. If the TLP is a memory read request, the payload is fetched from the specified memory location in the memory bank, and a completion packet is constructed and sent out on the TX interface of the integrated Endpoint block.

The memory endpoint reference design is limited in its functionality as follows:

- Does not support split transactions (the completer cannot break a read request into two or more separate completions)
- Can only process packets in TC0/VC0
- Does not initiate traffic (memory read or memory write requests)

A test bench is provided to exercise the functionality of the core and the reference design. The files corresponding to the test bench and the tasks it invokes are located in `<project_dir>/<component_name>/test_bench`.

The test bench generates a clock with a frequency equal to the reference clock frequency (100/125/250 MHz) chosen in the GUI. The test bench generates clocks that are used to clock the "Near-End" partner with limited downstream port capabilities and the reference design.

The user has the flexibility to change some parameters that drive the core while simulating the design. The changes can override the selections made in the GUI. The parameters are listed in Table 6-1.

Table 6-1: **User Parameters**

Parameter	Description
REFCLKFREQ	Allowed values: 100 MHz, 125 MHz, or 250 MHz
NO_OF_LANES	Allowed values: 1, 2, 4, or 8

Table 6-1: **User Parameters *(Continued)***

Parameter	Description
MAXPAYLOAD	Allowed values: 128, 256, 512, 1024, 2048, or 4096. The test bench passes this value to the downstream and upstream ports. It also initiates configuration writes to PCI Express Device status register to update the register value with the same
G_TC_NUMBER	Which traffic class to use to send data. Allowed values are 0–7. This parameter applies only to the Memory Endpoint Application.

Task Descriptions

The test bench provides tasks to initiate traffic from the downstream port to the core. Table 6-2 describes the various tasks invoked by the test bench.

Table 6-2: **Test Bench Tasks**

Task	Description
find_capabilities	Reports the capabilities (Power Management capability, PCI Express capability etc.) enabled in the integrated Endpoint block. This task does not have any input parameters.
ne_config_read	Initiates configuration reads to the near end device (downstream port). Input parameters to the task: • Length of the payload in 32-bit Data Words (DW) • Start Address • Byte Enables on the First DW • Byte Enables on the Last DW The data returned is stored in rx_data_buffer array and logged in the log file.
ne_config_write	Initiates configuration writes to the near end device (downstream port). Input parameters to the task: • Length of the payload in 32-bit Data Words (DW) • Start Address • Byte Enables on the First DW • Byte Enables on the Last DW The tx_data_to_send array should be populated with the payload data before invoking the task.
config0_read	Initiates type 0 configuration reads from the near end device to the core. Input parameters to the task: • Length of the payload in 32-bit Data Words (DW) • Start Address • Byte Enables on the First DW • Byte Enables on the Last DW The data returned is stored in rx_data_buffer array and logged in the log file.

Table 6-2: **Test Bench Tasks *(Continued)***

Task	Description
config0_write	Initiates type 0 configuration writes from the near end device to the core. Input parameters to the task: • Length of the payload in 32-bit Data Words (DW) • Start Address • Byte Enables on the First DW • Byte Enables on the Last DW The tx_data_to_send array should be populated with the payload data before invoking the task.
memory_read	Initiates memory reads from the near end device to the core. Input parameters to the task: • Length of the payload in 32-bit Data Words (DW) • Start Address • Traffic Class • Byte Enables on the First DW • Byte Enables on the Last DW The expected_data array should be populated with the expected payload before invoking the task. The returned data in rx_data_buffer is checked against the expected data.
memory_write	Initiates memory writes from the near end device to the core. Input parameters to the task: • Length of the payload in 32-bit Data Words (DW) • Start Address • Traffic Class • Byte Enables on the First DW • Byte Enables on the Last DW The tx_data_to_send array should be populated with the payload data before invoking the task.

Test Results

For Memory Writes, there is no data to compare, hence the success of the test is determined by issuing a memory read request to the same address location. For memory read transactions the payload portion of the completion packet returned from the core is checked against the expected data. The simulation will report errors if the data is incorrect.

Config Writes are followed by Config Reads to the same address. The user should verify if the data returned is correct. All test results are delivered through statements logged in log files.

Simulation Scripts

To simulate the test bench on ModelSim, a script is provided in the `<project_dir>/<component_name>/simulation/functional` directory

To run the simulation:

- Open a ModelSim window (VSIM).
- Change directory to `<project_dir>/<component_name>/simulation/functional`
- At the command prompt:
  - For Linux systems, type **`./simulate_mti.sh`**
  - For Windows operating systems, type **`simulate_mti.bat`**

The downstream port model and tasks in the test bench verify the basic functionality of the core by initiating several memory reads and writes to the Memory Endpoint reference design. PCIe BFMs available in the market may provide a more robust downstream model and enhanced testing capabilities. User should contact www.xilinx.com for further assistance on Xilinx approved BFM vendors.

XILINX®

Chapter 7

Implementing the Core

Core Constraints

The LogiCORE™ based solutions require the specification of timing and other physical implementation constraints to meet specified PCI Express™ performance requirements. These constraints are specified in a User Constraints File (UCF) and the wrapper files generated through the GUI.

Constraints provided with core have been fully tested in hardware. Constraints can be modified, but modifications should only be made with a thorough understanding of the effect of each constraint.

For details on the definition and use of a UCF or specific constraints, refer to Constraints Guide http://toolbox.xilinx.com/docsan/xilinx82/books/docs/cgd/cgd.pdf.

Although the UCF delivered with each core shares the same overall structure and sequence of information, the content of each core's UCF varies based on the user choices in the GUI, device, and the platform board used. The GUI delivers UCF files for all Virtex-5 SXT/LXT devices and packages.

The Memory Endpoint reference design delivered through the GUI is tested on following Xilinx platforms:

- ML505
- ML523
- ML525
- ML555

The I/O constraints in the UCF are customized to work with these boards and packages. Users must modify the I/O constraints in the UCF file for other custom boards.

I/O Constraints

The device selection, that is the part and package, govern the I/O locations and the Virtex-5 integrated block locations.

GTP_DUAL/GTX_DUAL PCIE_EP

The UCF files provide the most optimal recommended locations for the GTP/GTX tiles. The choice is based on the physical proximity of the GTP/GTX locations to the integrated Endpoint block. This gives the best relative positioning to meet timing requirements. Users can modify the locations to suit their design requirements. Users should use the Xilinx Software Tools to ensure that the choice of GTP/GTX locations does not result in timing errors before freezing their board layouts. As an alternative solution, the user can add pipelining stages between integrated Endpoint block and GTP/GTX tiles to solve timing issues due to non-optimal GTP/GTX location choices. In general, using more than four to five pipelining stages is not recommended.

For example, the ML555 board uses the GTP_DUAL locations shown below.

For the XC5VLX50T:

```
INST  "*/GTD[6].*GT_i" LOC = GTP_DUAL_X0Y0; # (MGT_122)
INST  "*/GTD[4].*GT_i" LOC = GTP_DUAL_X0Y3; # (MGT_112)
INST  "*/GTD[2].*GT_i" LOC = GTP_DUAL_X0Y1; # (MGT_118)
INST  "*/GTD[0].*GT_i" LOC = GTP_DUAL_X0Y2; # (MGT_114)
```

For the XC5VFX100T:

```
INST  "*/GTD[6].*GT_i" LOC = GTP_DUAL_X0Y1; # (MGT_122)
INST  "*/GTD[4].*GT_i" LOC = GTP_DUAL_X0Y4; # (MGT_112)
INST  "*/GTD[2].*GT_i" LOC = GTP_DUAL_X0Y2; # (MGT_118)
INST  "*/GTD[0].*GT_i" LOC = GTP_DUAL_X0Y3; # (MGT_114)GT
```

PCIE_EP

Users can select the location of the Integrated Endpoint block (PCIE_EP). Users can also select the PCIE_EP and GTX_DUAL locations to suit their design requirements.

For the XC5VFX100T:

```
INST "*dut_b/pcie_top_inst/pcie_ep_i" LOC = PCIE_X0Y0;
```

Reference Clock

The reference clock location constraints are tied to the choice of the GTP/GTX locations. Reference clock needs to arrive at one of the GTP/GTX tiles chosen in the design and routed to the other GTP/GTX tiles. The reference clock arriving at one GTP/GTX tile can be routed to any other GTP/GTX tile that is up to three GTP/GTX tiles away in either the north or south direction. The GTP/GTX tile used for Lane 0 is chosen to receive the reference clock. The location constraints for the ML555 example are shown below.

For the XC5VFX70T device or the XC5VFX100T device, (package type FF1136) the location constraints on the REFCLK are:

```
NET "REFCLK_P"          LOC = Y4;
NET "REFCLK_N"          LOC = Y3;
```

Locations Y4, Y3 are bound to the reference clock pins tied to MGT_114 in the package.

Users modifying the GTP/GTX tile locations should choose an appropriate REFCLK location.

The combination of board, device and package type determines the I/O locations such as clock pins, LEDs, pushbuttons, headers, dip switches, and so on.

For example if the package type selected is an FF1738 the location constraints on the REFCLK should be modified to:

```
NET "REFCLK_P"       LOC = AD4;
NET "REFCLK_N"       LOC = AD3;
```

Locations AD4, AD3 are bound to the reference clock pins tied to MGT_114. The GUI does not deliver a UCF file for this package. The user must modify the UCF file to use this package.

LEDs, Pushbuttons, Headers

The platform selected also affects the choice of I/O locations. For example, when using an ML523 board, the location constraints for the LEDs, pushbuttons, and headers should be as follows:

```
# HEADER PINS for ML523
# 250 MHz TXCLKOUT0 from the GT used for Lane0
NET "TXCLKOUT" LOC = T33;

# REFCLKOUT from the GT used for Lane0. The frequency is equal to the
# frequency of reference clock selected
NET "REFCLKOUT" LOC = R33;

# PLLLKDET from the GT used for Lane0. Indicates if the GTP PLL is
locked
NET "PLLLKDETOUT" LOC = R34;

# 250 MHz Clock
NET "CORECLK" LOC = R32;

# Frequency of the USERCLK depends on the CLK RATIO selected.
# 1:1 - 250 MHz ; 1:2 - 125 MHz ; 1:4 - 62.5 MHz
NET "USERCLK" LOC = AH34;

# LEDs
# indicates that the PCIe endpoint has successfully completed link
training with the downstream #port connected to it
NET "LINKUP" LOC = AE27;

# Indicates the PLL used to generate the core clk and user clk is
#locked
NET "CLKLOCK" LOC = AE26;

# PUSHBUTTONS
# Reset to the PCIe block
NET RST_N LOC = AF14;
NET RST_N PULLUP;

# Reset to the GTP_DUAL/GTX_DUAL tile
NET GTRESET_N LOC = AE22;
NET GTRESET_N PULLUP;
```

For an ML555 board, the location constraints for the LEDs, pushbuttons, and headers are not the same as the ML523 and should be as follows:

```
# HEADER PINS for ML555
# 250 MHz TXCLKOUT0 from the GT used for Lane0
NET "TXCLKOUT"        LOC = M8;

# REFCLKOUT from the GT used for Lane0. The frequency is equal to the
# frequency of reference clock selected
NET "REFCLKOUT"       LOC = H17;

# PLLLKDET from the GT used for Lane0. Indicates if the GTP PLL is
locked
NET "PLLLKDETOUT"     LOC = K8;

# 250 MHz Clock
NET "CORECLK"         LOC = L9;

# Frequency of the USERCLK depends on the CLK RATIO selected.
# 1:1 - 250 MHz ; 1:2 - 125 MHz ; 1:4 - 62.5 MHz
NET "USERCLK"         LOC = E12;

# LEDs
# Indicates that the PCIe endpoint has successfully completed link
training
# with the downstream port connected to it
NET "LINKUP"    LOC = H8;

# Indicates the PLL used to generate the core clk and user clk is locked
NET "CLKLOCK"   LOC = G8;

# PUSH BUTTONS
# Reset to the PCIe block
NET RST_N               LOC = AF20;
NET RST_N PULLUP;

# Reset to the GTP_DUAL/GTX_DUAL tile
NET GTRESET_N          LOC = AF21;
NET GTRESET_N PULLUP;
```

The UCF constrains some of the output pins to LEDs and headers. These are used for debugging and status and are not mandatory constraints. The only required I/Os are the reference clock pins REFCLK_P and REFCLK_N and the reset pins RST_N and GTRESET_N.

Note: The UCF constraints for LEDs, pushbuttons, and headers are not generated automatically. Templates and placeholders are generated for each constraint in the UCF file. However, if the user appends the board name (ml555, ml523, ml525, or ml505) to the component name in the Endpoint block menu (see Figure 4-1, page 43), the constraints are automatically generated. For example, if the user selects a component name of **`pci_express_wrapper_ml555`**, then the UCF file generated will contain all constraints related to LEDs, pushbuttons, and headers for the ML555 board.

Timing Constraints

Timing constraints should be specified for the core to operate at the desired frequency. The core has two clock domains: `user_clk` and `core_clk`. The `core_clk` is constrained to always run at 250 MHz, whereas the timing constraint on the `user_clk` is determined by the CLKRATIO setting of the core. The timing constraint in the UCF file should reflect the choices selected in the GUI and match parameter settings in the core.

For example, in a x4 design with the CLKRATIO set to 1:2 the timing constraints are

```
NET "core_clk"  PERIOD = 4ns;
NET "user_clk"  PERIOD = 8ns;
```

For a x8 design with the CLKRATIO set to 1:1 the user clock and the core clock are equal. The timing constraint in this case would be:

```
NET "*_clk"             PERIOD = 4ns;
```

For the design to pass through implementation successfully, the physical locations of the I/Os, flip-flops, and Virtex-5 integrated blocks and the timing constraints on the signals should be correctly set.

Implementation Scripts

The GUI delivers an implementation script to run the Memory Endpoint Reference Design through XST or Synplify synthesis, NGD build, MAP, PAR, TRACE and BITGEN. For more information on these tools refer to the Development System Reference Guide: http://toolbox.xilinx.com/docsan/xilinx82/books/docs/dev/dev.pdf.

The script is provided in the `<project_dir>/<component_name>/implement` directory.

To implement the design:

- Change directory to **`<project_dir>/<component_name>/implement`**
- At the command prompt:
  - For Linux systems, type **`./implement.sh`**
  - For Windows operating systems, type **`implement.bat`**

The **`-board`** option selects the UCF file corresponding to the specified board. If the board is not specified it defaults to ML555.

The resultant files from running NGD build, MAP, PAR, TRACE, and BITGEN are located in the `<project_dir>/<component_name>/implement/results_<lane width>_<board>_<part>` directory.

The examples and sample UCFs delivered along with the core are proven in simulation, timing, and hardware. Users can change the constraints if needed, but should verify the validity of the changes by ensuring that timing is met.