PCI Express x1/x2/x4 Endpoint IP Core

User Guide

FPGA-IPUG-02009 Version 1.7

October 2016

Contents

Figures

Tables

1. Introduction

PCI Express is a high performance, fully scalable, well defined standard for a wide variety of computing and communications platforms. It has been defined to provide software compatibility with existing PCI drivers and operating systems. Being a packet based serial technology, PCI Express greatly reduces the number of required pins and simplifies board routing and manufacturing. PCI Express is a point-to-point technology, as opposed to the multidrop bus in PCI. Each PCI Express device has the advantage of full duplex communication with its link partner to greatly increase overall system bandwidth. The basic data rate for a single lane is double that of the 32 bit/33 MHz PCI bus. A four lane link has eight times the data rate in each direction of a conventional bus.

Lattice's PCI Express core provides a x1, x2 or x4 endpoint solution from the electrical SERDES interface to the transaction layer. This solution supports the LatticeECP3™, ECP5™ and ECP5-5G™device families. When used with the LatticeECP3, ECP5 and ECP5-5G family of devices, the PCI Express core is implemented using an extremely economical and high value FPGA platform.

This users guide covers the following versions of the Lattice PCI Express Endpoint core:

PCI Express (2.5G) IP Core

- The Native x4 Core targets the LatticeECP3 and ECP5 family of devices.
- The x4 Downgraded x1 Core also targets the LatticeECP3 and ECP5 family. The x4 Downgraded x1 core is a x4 core that uses one channel of SERDES/PCS and a 64-bit data path for x1 link width.
- The x4 Downgraded x2 Core also targets the LatticeECP3 and ECP5 family. The x4 Downgraded x2 core is a x4 core that uses two channels of SERDES/PCS and a 64-bit data path for x2 link width.
- The Native x1 Core targets the LatticeECP3 and ECP5 family of devices. This is a reduced LUT count x1 core with a 16-bit data path.

PCI Express 5G IP Core

- The Native x2 Core targets the Lattice ECP5-5G device. The x2 core uses 2 channels of SERDES/PCS and a 64-bit data path for x2 link width.
- The x2 Downgraded x1 Core targets the Lattice ECP5-5G device. The x2 Downgraded x1 core is a x2 core that uses 1 channel of SERDES/PCS and a 64-bit data path for x1 link width.

1.1. Quick Facts

Table 1.1 provides quick facts about the Lattice PCI Express (2.5G) x1/x2/x4 IP Core.

Table 1.1. PCI Express (2.5G) IP Core Quick Facts

		PCI Express IP Configuration					
		Native x4		Native x1		Downgraded x2	
IP Requirements	FPGA Families Supported	LatticeECP3 and ECP5					
	Minimal Device Needed	LFE317E-7FN484C	LFE5UM-45F-7BG756CES	LFE3-17E-7FN484C	LFE5UM-25F-7BG381C	LFE317E-7FN484C	LFE5UM-45F-7BG756CES
Resource Utilization	Targeted Device	LFE395E-7FPBGA1156C	LFE5UM-85F-7BG756CES	LFE3-95E-7FPBGA1156C	LFE5UM-85F-7BG756CES	LFE395E-7FPBGA1156C	LFE5UM-85F-7BG756CES
	Data Path Width	64	64	16	16	64	64
	LUTs	12200	13900	6040	6207	12900	12200
	sysMEM™ EBRs	11	11	4	4	11	11
	Registers	9746	9763	4027	4188	8899	9746
Design Tool Support	Lattice Implementation	Lattice Diamond® 3.8					
	Synthesis	Synopsys® Synplify Pro® for Lattice I-2013.09L-SP1					
	Simulation	Aldec® Active-HDLTM 9.3 (Windows only, Verilog and VHDL)					
		Mentor Graphics® ModelSim® SE 10.2C (Verilog Only)					

Table 1.2 provides quick facts about the Lattice PCI Express 5G x1/x2 IP Core.

Table 1.2. PCI Express 5G IP Core Quick Facts

		PCI Express 5G IP Configuration	
		Native x2	**Downgraded x1**
IP Requirements	FPGA Families Supported	**Lattice ECP5-5G**	
Resource Utilization	Targeted Device	LFE5UM5G-85F-8BG756C	LFE5UM5G-85F-8BG756C
	Data Path Width	64	64
	LUTs	14909	13378
	sysMEM™ EBRs	7	7
	Registers	10928	9354
Design Tool Support	Lattice Implementation	Lattice Diamond® 3.8	
	Synthesis	Synopsys® Synplify Pro® for Lattice L-2016.03L-1	
	Simulation	Aldec® Active-HDL 10.3 (Windows only, Verilog and VHDL)	
		Mentor Graphics® ModelSim® SE 10.2C (Verilog Only)	

1.2. Features

The Lattice PCI Express IP core supports the following features:

1.2.1. PHY Layer

- 2.5 Gbps or 5.0 Gbps CML electrical interface
- PCI Express 2.0 electrical compliance
- Serialization and de-serialization
- 8b10b symbol encoding/decoding
- Data scrambling and de-scrambling
- Link state machine for symbol alignment
- Clock tolerance compensation supports +/- 300 ppm
- Framing and application of symbols to lanes
- Lane-to-lane de-skew
- Link Training and Status State Machine (LTSSM)
    - Electrical idle generation
    - Receiver detection
    - TS1/TS2 generation/detection
    - Lane polarity inversion
    - Link width negotiation
    - Higher layer control to jump to defined states

1.2.2. Data Link Layer

- Data link control and management state machine
- Flow control initialization
- Ack/Nak DLLP generation/termination
- Power management DLLP generation/termination through simple user interface
- LCRC generation/checking
- Sequence number generation/checking
- Retry buffer and management

1.2.3. Transaction Layer

- Supports all types of TLPs (memory, I/O, configuration and message)
- Power management user interface to easily send and receive power messages
- Optional ECRC generation/checking
- 128, 256, 512, 1 k, 2 k, or 4 Kbyte maximum payload size

1.2.4. Configuration Space Support

- PCI-compatible Type 0 Configuration Space Registers contained inside the core (0x0-0x3C)
- PCI Express Capability Structure Registers contained inside the core
- Power Management Capability Structure Registers contained inside the core
- MSI Capability Structure Registers contained inside the core
- Device Serial Number Capability Structure contained inside the core
- Advanced Error Reporting Capability Structure contained inside the core

1.2.5. Top Level IP Support

- 125 MHz user interface
  For 2.5G IP core:
    - Native x4 and Downgraded x1/x2 support a 64-bit datapath
    - Native x1 supports a 16-bit datapath

  For 5G IP core:
    - Native x2 and Downgraded x1 support a 64-bit datapath
- In transmit, user creates TLPs without ECRC, LCRC, or sequence number
- In receive, user receives valid TLPs without ECRC, LCRC, or sequence number
- Credit interface for transmit and receive for PH, PD, NPH, NPD, CPLH, CPLD credit types
- Upstream/downstream, single function endpoint topology
- Higher layer control of LTSSM via ports
- Access to select configuration space information via ports

2. Functional Descriptions

This chapter provides a functional description of the Lattice PCI Express Endpoint IP core.

2.1. Overview

The PCI Express core is implemented in several different FPGA technologies. These technologies include soft FPGA fabric elements such as LUTs, registers, embedded block RAMs (EBRs), embedded hard elements with the PCS/SERDES.

The Clarity Designer design tool is used to customize and create a complete IP module for the user to instantiate in a design. Inside the module created by the Clarity Designer tool are several blocks implemented in heterogeneous technologies. All of the connectivity is provided, allowing the user to interact at the top level of the IP core.

Figure 2.1 provides a high-level block diagram to illustrate the main functional blocks and the technology used to implement PCI Express functions.

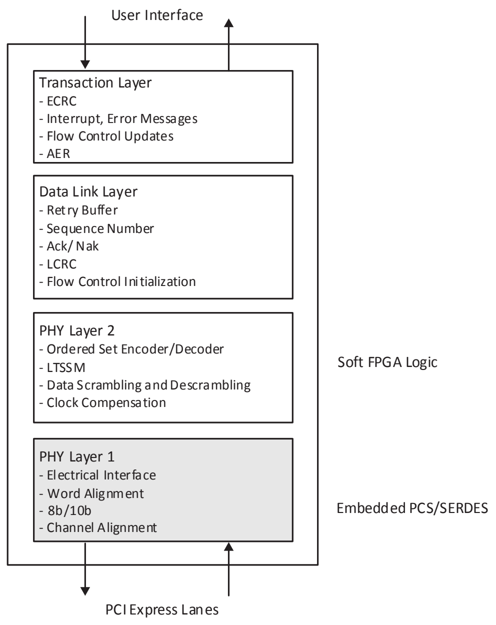


Figure 2.1. PCI Express IP Core Technology and Functions

As the PCI Express core proceeds through the Diamond software design flow specific technologies are targeted to their specific locations on the device. Figure 2.2 provides implementation representations of the LFE5UM devices with a PCI Express core.

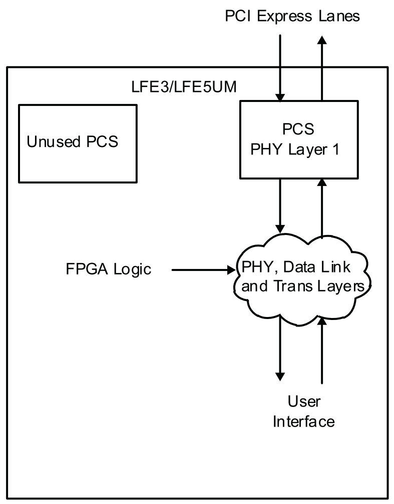


Figure 2.2. PCI Express Core Implementation in LatticeECP3, ECP5 and EPC5-5G

As shown, the data flow moves in and out of the heterogeneous FPGA technology. The user is responsible for selecting the location of the hard blocks (this topic will be discussed later in this document). The FPGA logic placement and routing is the job of the Diamond design tools to select regions nearby the hard blocks to achieve the timing goals.

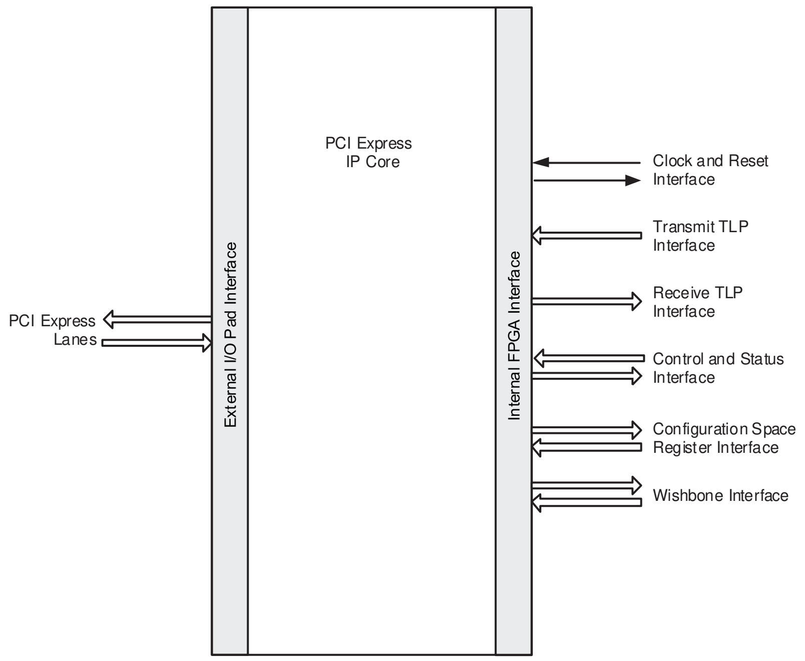


Figure 2.3. PCI Express Interfaces

Table 2.1 provides the list of ports and descriptions for the PCI Express IP core.

Table 2.1. PCI Express IP Core Port List

Port Name	Direction	Clock	Function Description
Clock and Reset Interface			
refclk[p,n]	Input		For 2.5G IP core: 100 MHz PCI Express differential reference clock used to generate the 2.5 Gbps data. For 5G IP core: 200 MHz PCI Express differential reference clock used to generate 5.0 Gbps data.
sys_clk_125	Output		125 MHz clock derived from refclk to be used in the user application.
rst_n	Input		Active-low asynchronous data path and state machine reset.
PCI Express Lanes			
hdin[p,n]_[0,1,2,3]	Input		PCI Express 2.5 or 5.0 Gbps CML inputs for lanes 0, 1, 2, and 3. For 5G IP core, up to lanes 0 and 1 only. hdin[p,n]_0 - PCI Express Lane 0 hdin[p,n]_1 - PCI Express Lane 1 hdin[p,n]_2 - PCI Express Lane 2 hdin[p,n]_3 - PCI Express Lane 3
hdout[p,n]_[0,1,2,3]	Output		PCI Express 2.5 or 5.0 Gbps CML inputs for lanes 0, 1, 2, and 3. For 5G IP core, up to lanes 0 and 1 only. hdout[p,n]_0 - PCI Express Lane 0 hdout[p,n]_1 - PCI Express Lane 1 hdout[p,n]_2 - PCI Express Lane 2 hdout[p,n]_3 - PCI Express Lane 3
Transmit TLP Interface			
tx_data_vc0[n:0]	Input	sys_clk_125	Transmit data bus. For 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core: [63:56] Byte N [55:48] Byte N+1 [47:40] Byte N+2 [39:32] Byte N+3 [31:24] Byte N+4 [23:16] Byte N+5 [15: 8] Byte N+6 [7: 0] Byte N+7 For 2.5G IP core Native x1: [15:8] Byte N [7:0] Byte N+1
tx_req_vc0	Input	sys_clk_125	Active high transmit request. This port is asserted when the user wants to send a TLP. If several TLPs will be provided in a burst, this port can remain high until all TLPs have been sent.
tx_rdy_vc0	Output	sys_clk_125	Active high transmit ready indicator. Tx_st should be provided next clock cycle after tx_rdy is high. This port will go low between TLPs.
tx_st_vc0	Input	sys_clk_125	Active high transmit start of TLP indicator.
tx_end_vc0	Input	sys_clk_125	Active high transmit end of TLP indicator. This signal must go

Port Name	Direction	Clock	Function Description
			low at the end of the TLP.
tx_nlfy_vc0	Input	sys_clk_125	Active high transmit nullify TLP. Can occur anywhere during the TLP. If tx_nlfy_vc0 is asserted to nullify a TLP the tx_end_vc0 port should not be asserted. The tx_nlfy_vc0 terminates the TLP.
tx_dwen_vc0	Input	sys_clk_125	Active high transmit 32-bit word indicator. Used if only bits [63:32] provide valid data. This port is only available for 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core.
tx_val	Output	sys_clk_125	Active high transmit clock enable. When a Native x4 or x2 is downgraded, this signal is used as the clock enable to downshift the transmit bandwidth. This port is only available for 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core.
tx_ca_[ph,nph,cplh]_vc0[8:0]	Output	sys_clk_125	Transmit Interface header credit available bus. This port will decrement as TLPs are sent and increment as UpdateFCs are received. Ph - Posted header Nph - Non-posted header Cplh - Completion header This credit interface is only updated when an UpdateFC DLLP is received from the PCI Express line. [8] - This bit indicates the receiver has infinite credits. If this bit is high then bits [7:0] should be ignored. [7:0] – The amount of credits available at the receiver.
tx_ca_[pd,npd,cpld]_vc0[12:0]	Output	sys_clk_125	Transmit Interface data credit available bus. This port will decrement as TLPs are sent and increment as UpdateFCs are received. pd - posted data npd - non-posted data cpld - completion data [12] - This bit indicates the receiver has infinite credits. If this bit is high, then bits [11:0] should be ignored. [11:0] - The amount of credits available at the receiver.
tx_ca_p_recheck_vc0	Output	sys_clk_125	Active high signal that indicates the core sent a Posted TLP which changed the tx_ca_p[h,d]_vc0 port. This might require a recheck of the credits available if the user has asserted tx_req_vc0 and is waiting for tx_rdy_vc0 to send a Posted TLP.
tx_ca_cpl_recheck_vc0	Output	sys_clk_125	Active high signal that indicates the core sent a Completion TLP which changed the tx_ca_cpl[h,d]_vc0 port. This might require a recheck of the credits available if the user has asserted tx_req_vc0 and is waiting for tx_rdy_vc0 to send a Completion TLP.
Receive TLP Interface			
rx_data_vc0[n:0]	Output	sys_clk_125	Receive data bus. For 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core: [63:56] Byte N [55:48] Byte N+1 [47:40] Byte N+2 [39:32] Byte N+3

Port Name	Direction	Clock	Function Description
			[31:24] Byte N+4 [23:16] Byte N+5 [15: 8] Byte N+6 [7: 0] Byte N+7 For 2.5G IP core Native x1: [15:8] Byte N [7:0] Byte N+1
rx_st_vc0	Output	sys_clk_125	Active high receive start of TLP indicator.
rx_end_vc0	Output	sys_clk_125	Active high receive end of TLP indicator.
rx_dwen_vc0	Output	sys_clk_125	Active high 32-bit word indicator. Used if only bits [63:32] contain valid data. This port is only available for 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core.
rx_ecrc_err_vc0	Output	sys_clk_125	Active high ECRC error indicator. Indicates an ECRC error in the current TLP. Only available if ECRC is enabled in the IP configuration GUI.
rx_us_req_vc0	Output	sys_clk_125	Active high unsupported request indicator. Asserted if any of the following TLP types are received: — Memory Read Request-Locked — The TLP is still passed to the user where the user will need to terminate the TLP with an Unsupported Request Completion.
rx_malf_tlp_vc0	Output	sys_clk_125	Active high malformed TLP indicator. Indicates a problem with the current TLPs length or format.
rx_bar_hit[6:0]	Output	sys_clk_125	Active high BAR indicator for the current TLP. If this bit is high, the current TLP on the receive interface is in the address range of the defined BAR. [6] - Expansion ROM [5] - BAR5 [4] - BAR4 [3] - BAR3 [2] - BAR2 [1] - BAR1 [0] - BAR0 For 64-bit BARs, a BAR hit will be indicated on the lower BAR number. The rx_bar_hit changes along with the rx_st_vc0 signal.
ur_np_ext	Input	sys_clk_125	Active high indicator for unsupported non-posted request reception.
ur_p_ext	Input	sys_clk_125	Active high indicator for unsupported posted request reception.
[ph,pd, nph,npd] _buf_status_vc0	Input	sys_clk_125	Active high user buffer full status indicator. When asserted, an UpdateFC will be sent for the type specified as soon as possible without waiting for the UpdateFC timer to expire.
[ph,nph]_processed_vc0	Input	sys_clk_125	Active high indicator to inform the IP core of how many credits have been processed. Each clock cycle high counts as one credit processed. The core will generate the required UpdateFC DLLP when either the UpdateFC timer expires or enough credits have been processed.
[pd,npd]_processed_vc0	Input	sys_clk_125	Active high enable for [pd, npd]_num_vc0 port. The user should place the number of data credits processed on the [pd, npd]_num_vc0 port and then assert [pd, npd]_processed_vc0 for one clock cycle. The core will generate the required UpdateFC DLLP when either the

Port Name	Direction	Clock	Function Description
			UpdateFC timer expires or enough credits have been processed.
[pd,npd]_num_vc0[7:0]	Input	sys_clk_125	This port provides the number of PD or NPD credits processed. It is enabled by the [pd, npd]_processed_vc0 port.
Control and Status			
		PHYSICAL LAYER	
no_pcie_train	Input	Async	Active high signal disables LTSSM training and forces the LTSSM to L0. This is intended to be used in simulation only to force the LTSSM into the L0 state.
force_lsm_active	Input	Async	Forces the Link State Machine for all of the channels to the linked state.
force_rec_ei	Input	Async	Forces the detection of a received electrical idle.
force_phy_status	Input	Async	Forces the detection of a receiver during the LTSSM Detect state on all of the channels.
force_disable_scr	Input	Async	Disables the PCI Express TLP scrambler.
hl_snd_beacon	Input	sys_clk_125	Active high request to send a beacon.
hl_disable_scr	Input	Async	Active high to set the disable scrambling bit in the TS1/TS2 sequence.
hl_gto_dis	Input	Async	Active high request to go to Disable state when LTSSM is in Config or Recovery.
hl_gto_det	Input	sys_clk_125	Active high request to go to Detect state when LTSSM is in L2 or Disable.
hl_gto_hrst	Input	sync	Active high request to go to Hot Reset when LTSSM is in Recovery.
hl_gto_l0stx	Input	sys_clk_125	Active high request to go to L0s when LTSSM is in L0.
hl_gto_l0stxfts	Input	sys_clk_125	Active high request to go to L0s and transmit FTS when LTSSM is in L0s.
hl_gto_l1	Input	sys_clk_125	Active high request to go to L1 when LTSSM is in L0.
hl_gto_l2	Input	sys_clk_125	Active high request to go to L2 when LTSSM is in L0.
hl_gto_lbk[3:0]	Input	sys_clk_125	Active high request to go to Loopback when LTSSM is in Config or Recovery.
hl_gto_rcvry	Input	sys_clk_125	Active high request to go to Recovery when LTSSM is in L0, L0s or L1.
hl_gto_cfg	Input	sys_clk_125	Active high request to go to Config when LTSSM is in Recovery.
phy_ltssm_state[3:0]	Output	sys_clk_125	PHY Layer LTSSM current state 0000 - Detect 0001 - Polling 0010 - Config 0011 - L0 0100 - L0s 0101 - L1 0110 - L2 0111 - Recovery 1000 - Loopback 1001 - Hot Reset 1010 - Disabled
phy_cfgln[n:0]	Output	sys_clk_125	Active high LTSSM Config state link status. An active bit indicates the channel is included in the configuration link width negotiation. [3:0] - 2.5G IP core Native x4, Downgraded x1/x2

Port Name	Direction	Clock	Function Description
			[1:0] - 5G IP core Native x2, Downgraded x1 [0] - PCI Express Lane 3 [1] - PCI Express Lane 2 [2] - PCI Express Lane 1 [3] - PCI Express Lane 0
phy_cfgln_sum[2:0]	Output	sys_clk_125	Link Width. This port is only available for 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core. 000 - No link defined 001 - Link width = 1 010 - Link width = 2 100 - Link width = 4
phy_pol_compliance	Output	sys_clk_125	Active high indicator that the LTSSM is in the Polling. Compliance state.
tx_lbk_rdy	Output	sys_clk_250	This output port is used to enable the transmit master loop-back data. This port is only available if the Master Loop-back feature is enabled in the IP configuration GUI.
tx_lbk_kcntl[7/1:0]	Input	sys_clk_250	This input port is used to indicate a K control word is being sent on tx_lbk_data port. This port is only available if the Master Loopback feature is enabled in the IP configuration GUI. For 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core: [7:6]- K control on tx_lbk_data[63:48] [5:4] - K control on tx_lbk_data[47:32] [3:2] - K control on tx_lbk_data[31:16] [1:0] - K control on tx_lbk_data[15:0] For 2.5G IP core Native x1: [1:0] - K control on rx_lbk_data[15:0]
tx_lbk_data[63/15:0]	Input	sys_clk_250	This input port is used to send 64-bit data for the master loopback. This port is only available if the Master Loopback feature is enabled in the IP configuration GUI. For 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core: [63:48] - Lane 3 data [47:32] - Lane 2 data [31:16] - Lane 1 data [15:0] - Lane 0 data For 2.5G IP core Native x1: [15:0] - Lane 0 data
rx_lbk_kcntl[7/1:0]	Output	sys_clk_250	This output port is used to indicate a K control word is being received on rx_lbk_data port. This port is only avail-able if the Master Loopback feature is enabled in the IP configuration GUI. For 2.5G IP core Native x4, Downgraded x1/x2 or 5G IP core: [7:6] - K control on rx_lbk_data[63:48] [5:4] - K control on rx_lbk_data[47:32] [3:2] - K control on rx_lbk_data[31:16]

Port Name	Direction	Clock	Function Description
			[1:0] - K control on rx_lbk_data[15:0] For 2.5G IP core Native x1: [1:0] - K control on rx_lbk_data[15:0]
rx_lbk_data[63:0]	Output	sys_clk_250	This output port is used to receive 64/16-bit data for the master loopback. This port is only available if the Master Loopback feature is enabled in the IP configuration GUI. For 2.5G IP core Native x4, Downgraded x1/x2 and 5G IP core: [63:48] - Lane 3 data [47:32] - Lane 2 data [31:16] - Lane 1 data [15:0] - Lane 0 data For 2.5G IP core Native x1: [15:0]- Lane 0 data
DATA LINK LAYER			
dl_inactive	Output	sys_clk_125	Data Link Layer is the DL_Inactive state.
dl_init	Output	sys_clk_125	Data Link Layer is in the DL_Init state.
dl_active	Output	sys_clk_125	Data Link Layer is in the DL_Active state.
dl_up	Output	sys_clk_125	Data Link Layer is in the DL_Active state and is now providing TLPs to the Transaction Layer.
tx_dllp_val	Input	sys_clk_125	Active high power message send command 00 - Nothing to send 01 - Send DLLP using tx_pmtype DLLP 10 - Send DLLP using tx_vsd_data Vendor Defined DLLP 11 - Not used
tx_pmtype[2:0]	Input	sys_clk_125	Transmit power message type 000 - PM Enter L1 001 - PM Enter L2 011 - PM Active State Request L1 100 - PM Request Ack
tx_vsd_data[23:0]	Input	sys_clk_125	Vendor-defined data to send in DLLP
tx_dllp_sent	Output	sys_clk_125	Requested DLLP was sent.
rxdp_pmd_type[2:0]	Output	sys_clk_125	Receive power message type 000 - PM Enter L1 001 - PM Enter L2 011 - PM Active State Request L1 100 - PM Request Ack
rxdp_vsd_data[23:0]	Output	sys_clk_125	Vendor-defined DLLP data received
rxdp_dllp_val	Output	sys_clk_125	Active high power message received
tx_rbuf_empty	Output	sys_clk_125	Transmit retry buffer is empty. (Used for ASPM implementation outside the core.)
tx_dllp_pend	Output	sys_clk_125	A DLLP is pending to be transmitted. (Used for ASPM implementation outside the core.)
rx_tlp_rcvd	Output	sys_clk_125	A TLP was received. (Used for ASPM implementation outside the core.)
TRANSACTION LAYER			
ecrc_gen_enb	Output	sys_clk_125	If AER and ECRC are enabled then this port is an output and indicates when ECRC generation is enabled by the PCI Express IP core.

Port Name	Direction	Clock	Function Description
			If ECRC generation is turned on then the TD bit in the transmit TLP header must be set to provide room in the TLP for the insertion of the ECRC.
ecrc_chk_enb	Output	sys_clk_125	If AER and ECRC are enabled then this port is an output and indicates when ECRC checking is enabled by the PCI Express IP core.
cmpln_tout	Input	sys_clk_125	Completion Timeout Indicator for posted request. Used to force non-fatal error message generation and also set appropriate bit in AER.
cmpltr_abort_np	Input	sys_clk_125	Complete or Abort Indicator for non-posed request. Used to force non-fatal error message generation and also set appropriate bit in AER.
cmpltr_abort_p	Input	sys_clk_125	Complete or Abort Indicator. Used to force non-fatal error message generation and also set appropriate bit in AER.
unexp_cmpln	Input	sys_clk_125	Unexpected Completion Indicator. Used to force non-fatal error message generation and also set appropriate bit in AER.
np_req_pend	Input	sys_clk_125	Sets device Status[5] indicating that a Non-Posted transaction is pending.
err_tlp_header[127:0]	Input	sys_clk_125	Advanced Error Reporting errored TLP header. This port is used to provide the TLP header for the TLP associated with an unexp_cmpln or cmpltr_abort_np/cmpltr_abort_p. The header data should be provided on the same clock cycle as the unexp_cmpln or cmpltr_abort_np/cmpltr_abort_p.
CONFIGURATION REGISTERS			
bus_num[7:0]	Output	sys_clk_125	Bus Number supplied with configuration write.
dev_num[4:0]	Output	sys_clk_125	Device Number supplied with configuration write.
func_num[2:0]	Output	sys_clk_125	Function Number supplied with configuration write.
cmd_reg_out[5:0]	Output	sys_clk_125	PCI Type0 Command Register bits [5] - Interrupt Disable [4] - SERR# Enable [3] - Parity Error Response [2] - Bus Master [1] - Memory Space [0] - IO Space
dev_cntl_out[14:0]	Output	sys_clk_125	PCI Express Capabilities Device Control Register bits [14:0].
lnk_cntl_out[7:0]	Output	sys_clk_125	PCI Express Capabilities Link Control Register bits [7:0].
inta_n	Input	sys_clk_125	Legacy INTx interrupt request. Falling edge will produce an ASSERT_INTx message and set the Interrupt Status bit to a 1. Rising edge will produce a DEASSERT_INTx message and clear the Interrupt Status bit. The Interrupt Disable bit will disable the message to be sent, but the status bit will operate as normal. The inta_n port has a requirement for how close an assert or de-assert event can be to the previous assert or de-assert event. For the 2.5G IP core native x4 and x2/x1 downgraded cores, this is two sys_clk_125 clock cycles. For the native x1 core this is eight sys_clk_125 clock cycles. If the inta_n port is low indicating an ASSERT_INTx and the Interrupt Disable bit is driven low by the system, then the inta_n port needs to be pulled high to send a

Port Name	Direction	Clock	Function Description
			DEASSERT_INTx message. This can be automatically performed by using a logic OR between the inta_n and cmd_reg_out[5] port.
msi[7:0]	Input	sys_clk_125	MSI interrupt request. Rising edge on a bit will produce a MemWr TLP for a MSI interrupt for the provided address and data by the root complex. [7] - MSI 8 [6] - MSI 7 [5] - MSI 6 [4] - MSI 5 [3] - MSI 4 [2] - MSI 3 [1] - MSI 2 [0] - MSI 1
flr_rdy_in	Input	sys_clk_125	Ready from user logic to perform Functional Level Reset
initiate_flr	Output	sys_clk_125	Initiate Functional Level Reset for user logic
dev_cntl_2_out	Output	sys_clk_125	PCI Express Capabilities Device Control 2 Register Bits [4:0]
mm_enable[2:0]	Output	sys_clk_125	Multiple MSI interrupts are supported by the root complex. This indicates how many messages the root complex will accept.
msi_enable	Output	sys_clk_125	MSI interrupts are enabled by the root complex. When this port is high MSI interrupts are to be used. The inta_n port is disabled.
pme_status	Input	sys_clk_125	Active high input to the Power Management Capability Structure PME_Status bit. Indicates that a Power Management Event has occurred on the endpoint.
pme_en	Output	sys_clk_125	PME_En bit in the Power Management Capability Structure. Active high signal to allow the endpoint to send PME messages.
pm_power_state[1:0]	Output	sys_clk_125	Power State in the Power Management Capability Structure. Software sets this state to place the endpoint in a particular state. 00 - D0 01 - D1 10 - D2 11 - D3
load_id	Input	sys_clk_125	This port is only present when the “Load IDs from Ports” checkbox is enabled in the IP configuration GUI. When this port is low, the core will send Configuration Request Retry Status for all Configuration Requests. When this port is high, the core will send normal Successful Completions for Configuration Requests. On the rising edge of load_id the vectors on vendor_id[15:0], device_id[15:0], rev_id[7:0], class_code[23:0], subsys_ven_id[15:0], and subsys_id[15:0] will be loaded into the proper configuration registers.
vendor_id[15:0]	Input	sys_clk_125	This port is only present when the “Load IDs from Ports” checkbox is enabled in the IP configuration GUI. This port will load the vendor ID for the core on the rising edge of load_id.
device_id[15:0]	Input	sys_clk_125	This port is only present when the “Load IDs from Ports” checkbox is enabled in the IP configuration GUI. This port will load the device ID for the core on the rising edge of load_id.
rev_id[7:0]	Input	sys_clk_125	This port is only present when the “Load IDs from Ports”

Port Name	Direction	Clock	Function Description
			checkbox is enabled in the IP configuration GUI. This port will load the revision ID for the core on the rising edge of load_id.
class_code[23:0]	Input	sys_clk_125	This port is only present when the “Load IDs from Ports” checkbox is enabled in the IP configuration GUI. This port will load the class code for the core on the rising edge of load_id.
subsys_ven_id[15:0]	Input	sys_clk_125	This port is only present when the “Load IDs from Ports” checkbox is enabled in the IP configuration GUI. This port will load the subsystem vendor ID for the core on the rising edge of load_id.
subsys_id[15:0]	Input	sys_clk_125	This port is only present when the “Load IDs from Ports” checkbox is enabled in the IP configuration GUI. This port will load the subsystem device ID for the core on the rising edge of load_id.
Wishbone Interface*			
CLK_I	Input		Wishbone interface clock.
RST_I	Input	CLK_I	Asynchronous reset.
SEL_I [3:0]	Input	CLK_I	Data valid indicator [3] - DAT_I[31:24] [2] - DAT_I[23:16] [1] - DAT_I[15:8] [0] - DAT_i[7:0]
WE_I	Input	CLK_I	Write enable 1 - write 0 - read
STB_I	Input	CLK_I	Strobe input.
CYC_I	Input	CLK_I	Cycle input.
DAT_I[31:0]	Input	CLK_I	Data input.
ADR_I[12:0]	Input	CLK_I	Address input.
CHAIN_RDAT_in[31:0]	Input	CLK_I	Daisy chain read data. If using a read chain for the wish-bone interface, this would be the read data from the previous slave. If not using a chain, then this port should be tied low.
CHAIN_ACK_in	Input	CLK_I	Daisy chain ack. If using a read chain for the wishbone interface, this would be the ack from the previous slave. If not using a chain, then this port should be tied low.
ACK_O	Output	CLK_I	Ack output.
IRQ_O	Output	CLK_I	Interrupt output. This port is not used (always 0).
DAT_O[31:0]	Output	CLK_I	Data output.

***Note**: Complete information on the Wishbone interface specification can be found at www.opencores.org in the WISHBONE System-on-Chip (SOC) Interconnection Architecture for Portable IP Cores specification.

2.2. Interface Description

This section describes the datapath user interfaces of the IP core. The transmit and receive interfaces both use the TLP as the data structure. The lower layers attach the start, end, sequence number and crc.

2.2.1. Transmit TLP Interface

In the transmit direction, the user must first check the credits available on the far end before sending the TLP. This information is found on the tx_ca_[ph,pd,nph,npd]_vc0 bus. There must be enough credits available for the entire TLP to be sent.

The user must then check that the core is ready to send the TLP. This is done by asserting the tx_req_vc0 port and waiting for the assertion of tx_rdy_vc0. While waiting for tx_rdy_vc0, if tx_ca_p/cpl_recheck is asserted, then the user must check available credit again. If there is enough credit, the user can proceed with the sending data based on tx_rdy_vc0. If the credit becomes insufficient, tx_req_vc0 must be deasserted on the next clock until enough credit is available. When tx_rdy_vc0 is asserted the next clock cycle will provide the first 64-bit word of the TLP and assert tx_st_vc0.

Tx_rdy_vc0 will remain high until one clock cycle before the last clock cycle of TLP data (based on the length field of the TLP). This allows the tx_rdy_vc0 to be used as the read enable of a non-pipelined FIFO.

2.2.1.1. Transmit TLP Interface Waveforms for 2.5G IP Core Native x4, Downgraded x1/x2 and 5G IP Core Native x2, Downgraded x1

Figure 2.4 through Figure 2.12 provide timing diagrams for the tx interface signals with a 64-bit datapath.

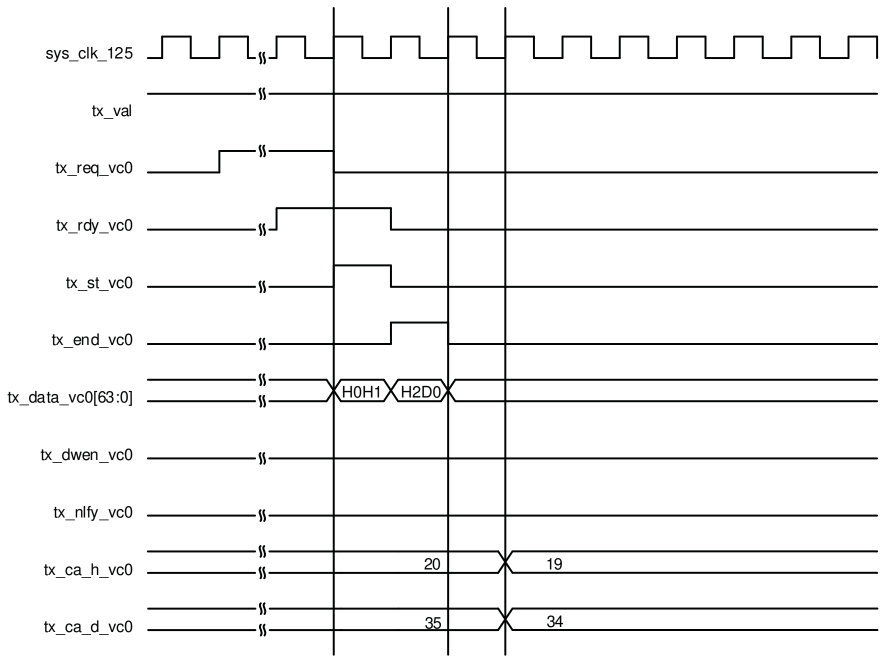


Figure 2.4. Transmit Interface of 2.5G IP core Native x4 or 5G IP core Native x2, 3DW Header, 1 DW Data

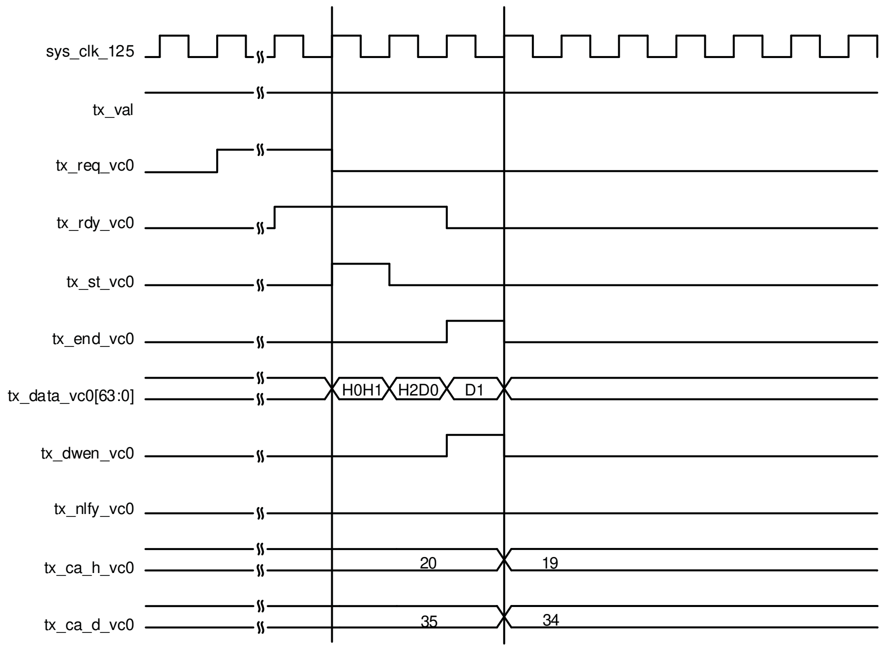


Figure 2.5. Transmit Interface 2.5G IP core Native x4 or 5G IP core Native x2, 3DW Header, 2 DW Data

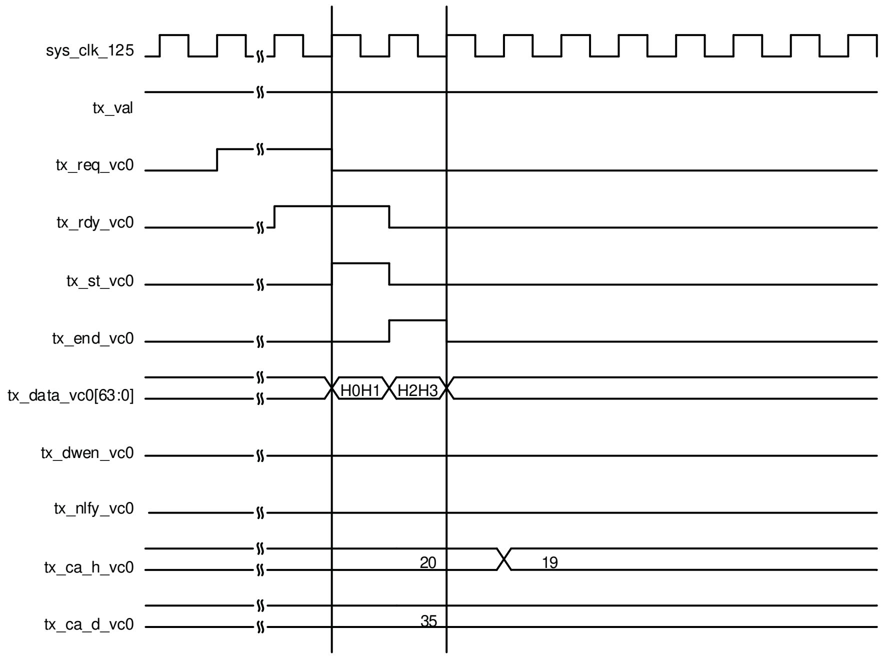


Figure 2.6. Transmit Interface 2.5G IP core Native x4 or 5G IP core Native x2, 4DW Header, 0 DW

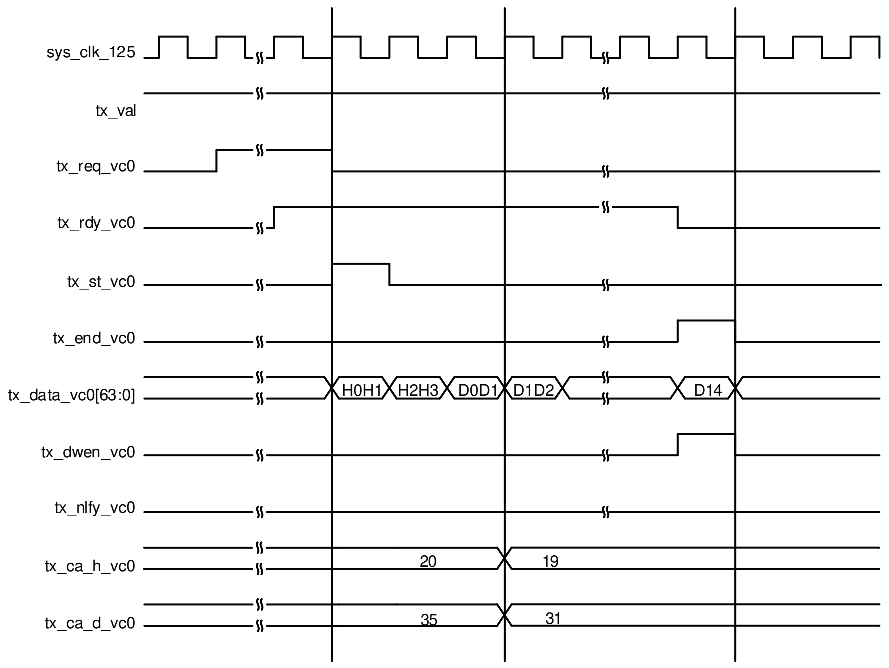


Figure 2.7. Transmit Interface 2.5G IP core Native x4 or 5G IP core Native x2, 4DW Header, Odd Number of DWs

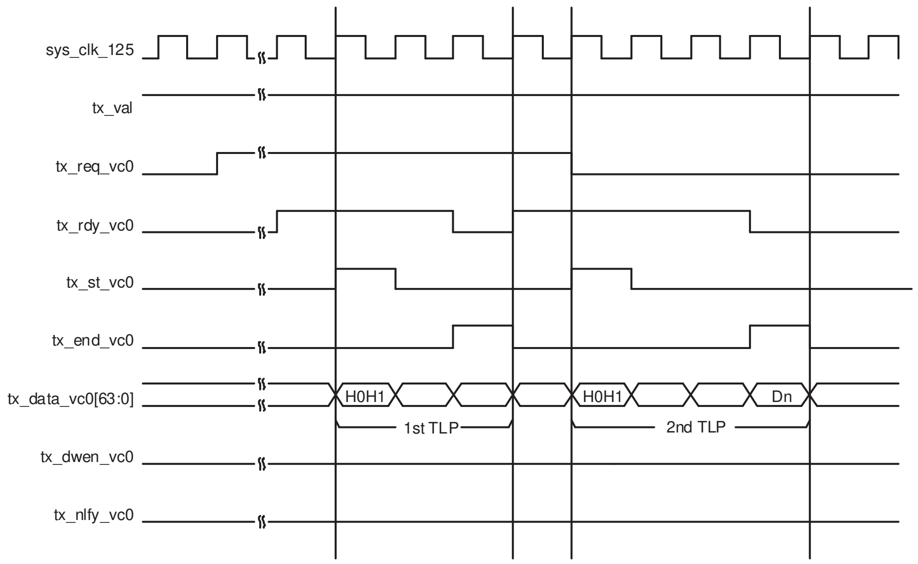


Figure 2.8. Transmit Interface 2.5G IP core Native x4 or 5G IP core Native x2, Burst of Two TLPs

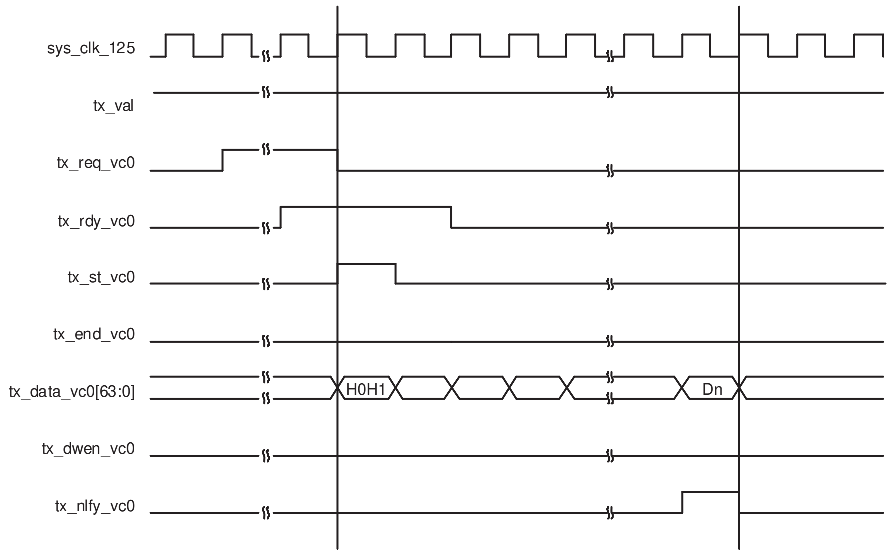


Figure 2.9. Transmit Interface 2.5 IP core Native x4 or 5G IP core Native x2, Nullified TLP

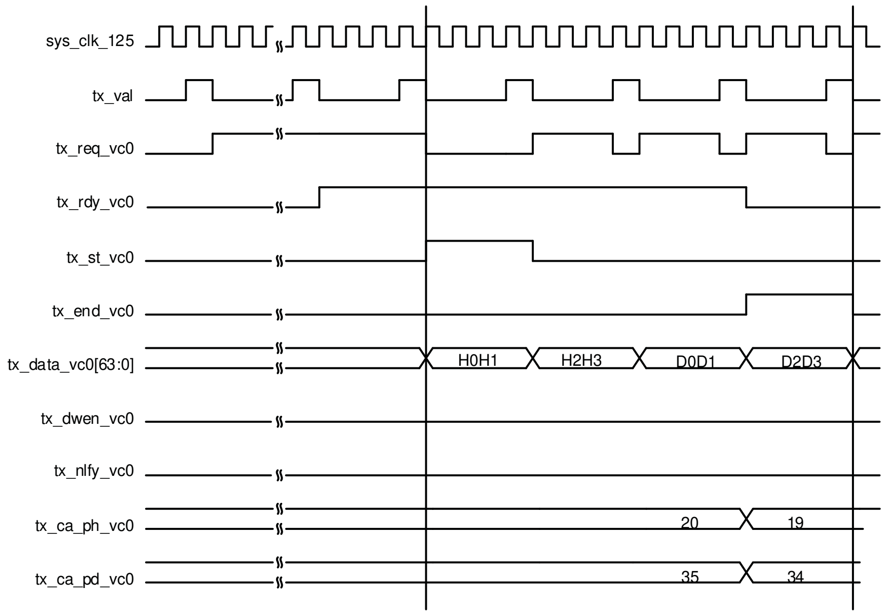


Figure 2.10. Transmit Interface 2.5G IP Core Downgraded x1 or 5G IP core Downgraded x1 at Gen1 speed

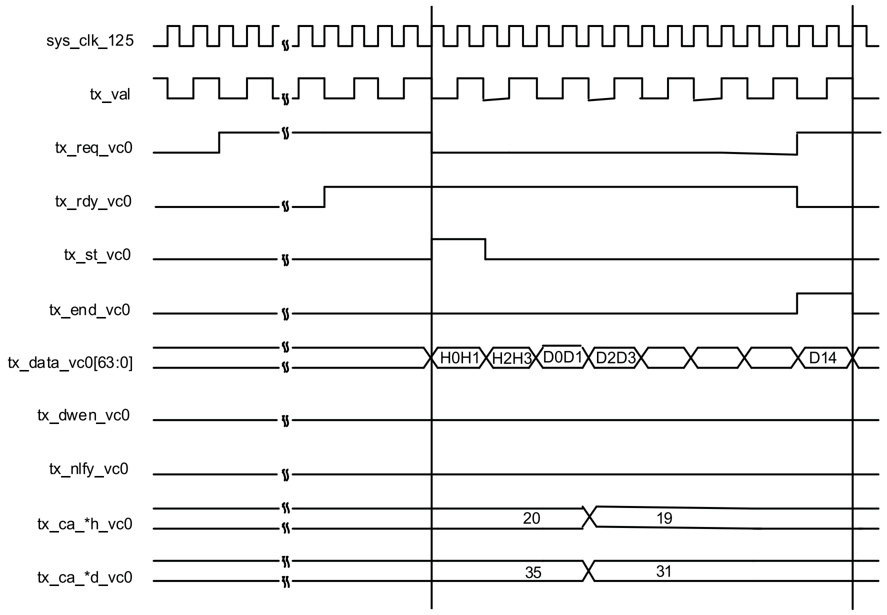


Figure 2.11. Transmit Interface 2.5G IP core Downgraded x2, 5G IP core Native x2 at Gen 1 speed or Downgraded x1 at Gen2 speed

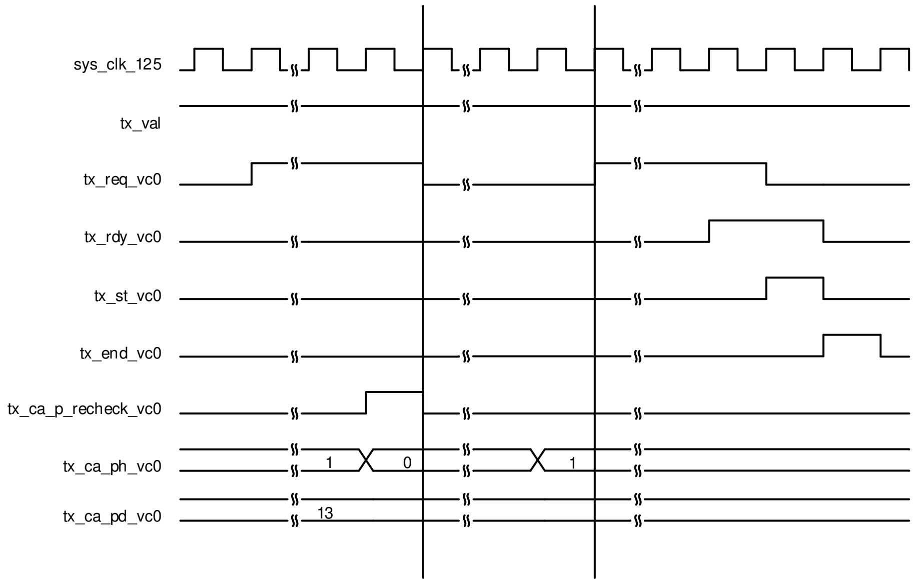


Figure 2.12. Transmit Interface 2.5G IP core Native x4 or 5G IP core Native x2, Posted Request with tx_ca_p-recheck Assertion

2.2.1.2. Transmit TLP Interface Waveforms for 2.5G IP Core Native x1

Figure 2.13 through Figure 2.16 provide timing diagrams for the transmit interface signals with a 16-bit datapath.

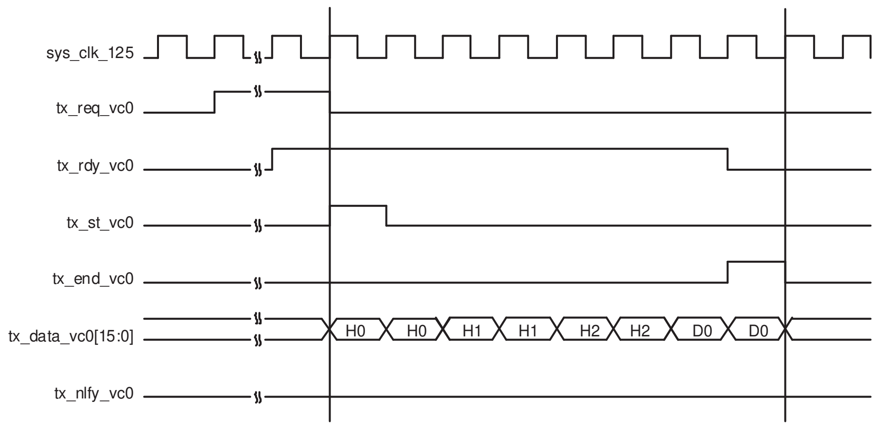


Figure 2.13. Transmit Interface Native x1, 3DW Header, 1 DW Data

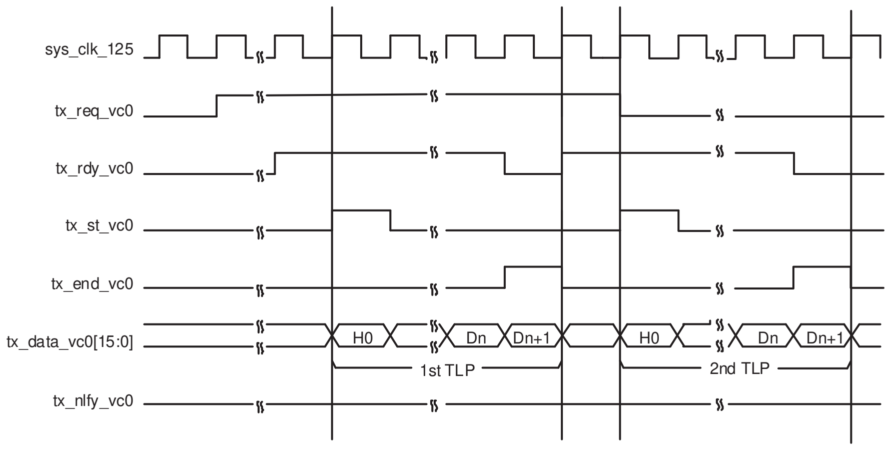


Figure 2.14. Transmit Interface Native x1, Burst of Two TLPs

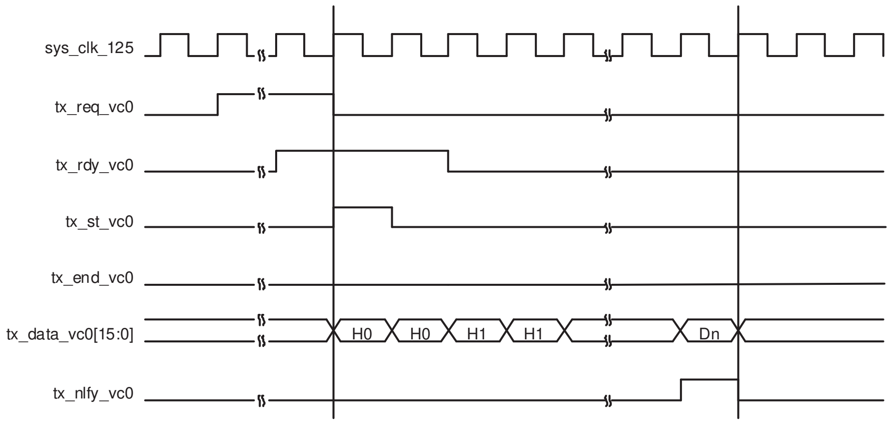


Figure 2.15. Transmit Interface Native x1, Nullified TLP

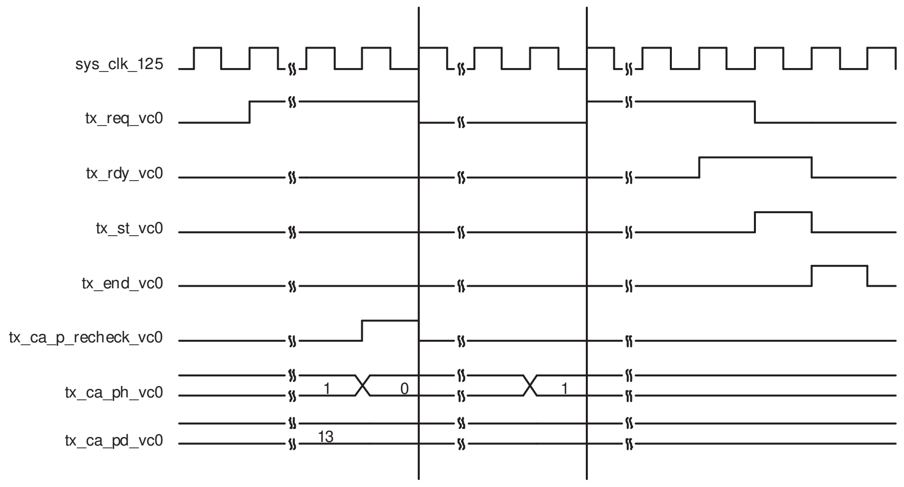


Figure 2.16. Transmit Interface Native x1 Posted Request with tx_ca_p-recheck Assertion

2.2.2. Receive TLP Interface

In the receive direction, TLPs will come from the core as they are received on the PCI Express lanes. Config read and config write TLPs to registers inside the core will be terminated inside the core. All other TLPs will be provided to the user. Also, if the core enables any of the BARs the TLP will go through a BAR check to make sure the TLPs address is in the range of any programmed BARs. If a BAR is accessed, the specific BAR is indicated by the rx_bar_hit[6:0] bus.

When a TLP is sent to the user the rx_st_vc0 signal will be asserted with the first word of the TLP. The remaining TLP data will be provided on consecutive clock cycles until the last word with rx_end_vc0 asserted. If the TLP contains a ECRC error the rx_ecrc_err_vc0 signal is asserted at the end of the TLP. If the TLP has a length problem, the rx_malf_tlp_vc0 will be asserted at any time during the TLP. Figure 2.17 through Figure 2.20 provide timing diagrams of the receive interface.

TLPs come from the receive interface only as fast as they come from the PCI Express lanes. There will always be at least one clock cycle between rx_end_vc0 and the next rx_st_vc0.

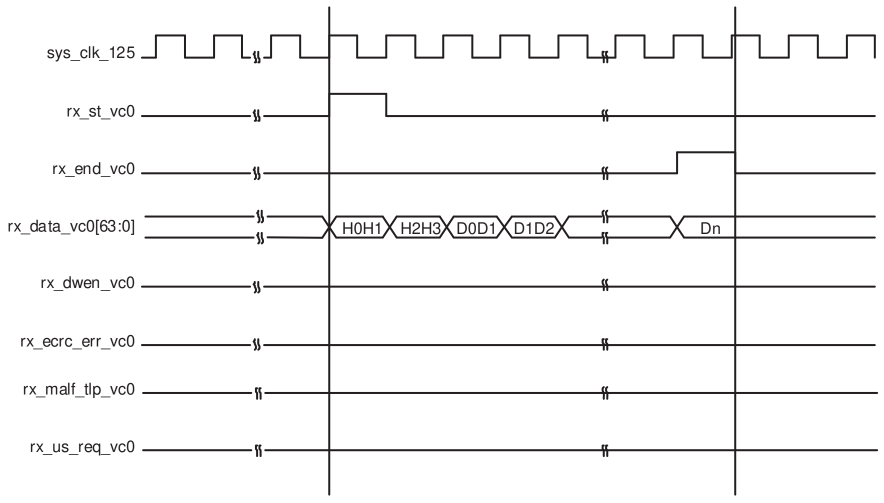


Figure 2.17. Receive Interface, Clean TLP

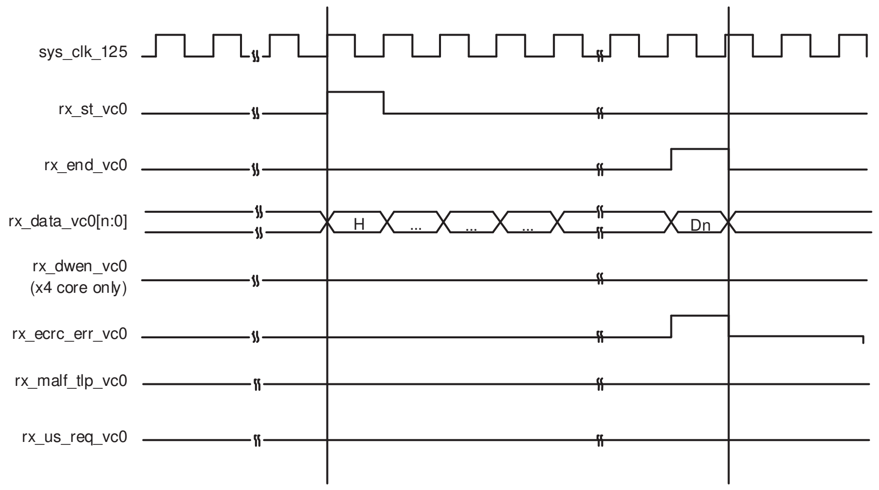


Figure 2.18. Receive Interface, ECRC Errored TLP

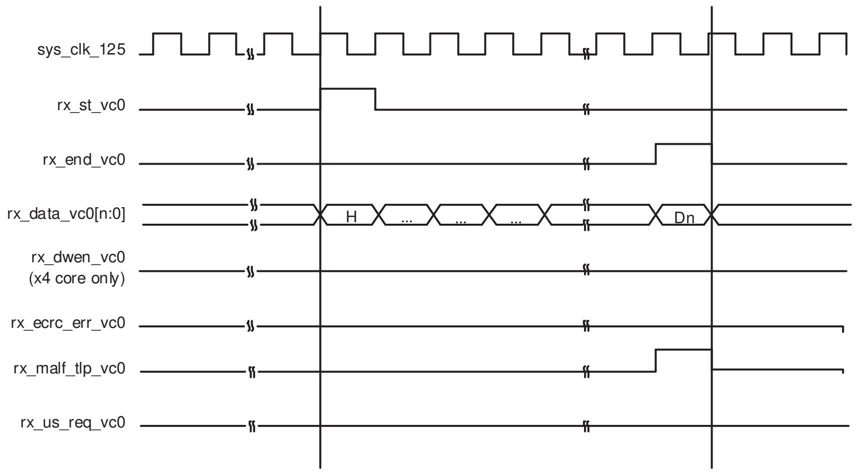


Figure 2.19. Receive Interface, Malformed TLP

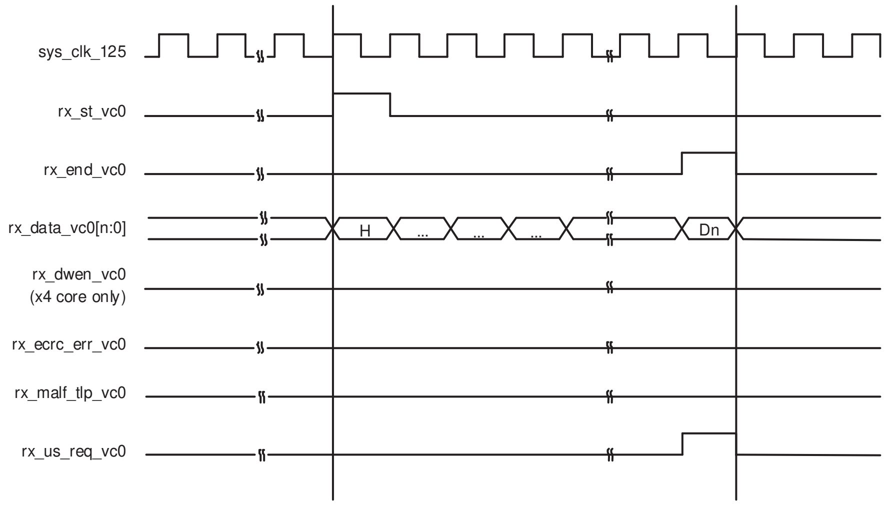


Figure 2.20. Receive Interface, Unsupported Request TLP

2.3. Using the Transmit and Receive Interfaces

There are two ways a PCI Express endpoint can interact with a root complex. As a completer, the endpoint will respond to accesses made by the root complex. As an initiator, the endpoint will perform accesses to the root complex.

The following sections will discuss how to use transmit and receive TLP interfaces for both of these types of interactions.

When the “Terminate All Config TLPs” option is checked in the IP configuration GUI, the IP core will handle all configuration requests. This includes responding as well as credit handling.

2.3.1. As a Completer

In order to be accessed by a root complex at least one of the enabled BARs will need to be programmed. The BIOS or OS will enumerate the configuration space of the endpoint. The BARs initial value (loaded via the GUI) is read to understand the memory requirements of the endpoint. Each enabled BAR is then provided a base address to be used.

When a memory request is received the PCI Express core will perform a BAR check. The address contained in the memory request is checked against all of the enabled BARs. If the address is located in one of the BARs' address range the rx_bar_hit[6:0] port will indicate which BAR is currently being accessed.

At this time the rx_st_vc0 will be asserted with rx_data_vc0 providing the first eight bytes of the TLP. The memory request will terminate with the rx_end_vc0 port asserting. The user must now terminate the received TLP by release credit returns and completions for a non-posted request. The user logic must decode release credits for all received TLPs except for error TLP. If the core finds any errors in the current TLP, the error will be indicated on the rx_ecrc_err_vc0 or rx_malf_tlp_vc0 port. Refer to the Error Handling section for additional details on credit handling and completion rules for receive errored TLP.

If the TLP is a 32-bit MWr TLP (rx_data_vc0[63:56]= 0x40) or 64-bit MWr TLP (rx_data_vc0[63:56]=0x30) the address and data need to be extracted and written to the appropriate memory space. Once the TLP is processed the posted credits for the MWr TLP must be released to the far end. This is done using the ph_processed_vc0, pd_processed_vc0, and pd_num_vc0 ports. Each MWr TLP takes 1 header credit. There is one data credit used per four DWs of data. The length field (rx_data_vc0[41:32]) provides the number of DWs used in the TLP. If the TLP length is on 4DW boundary (rx_data_vc0[33:32]=0x0), the number of credits is the TLP length divided by 4 (rx_data_vc0[41:34]). If the TLP length is not on 4DW boundary (rx_data_vc0[33:32] >0) the number of credits is rx_data_vc0[41:34] + 1 (round up by 1). The number of credits used should then be placed on pd_num_vc0[7:0]. Assert ph_processed_vc0 and pd_processed_vc0 for 1 clock cycle to latch in the pd_num_vc0 port and release credits.

If the TLP is a 32-bit MRd TLP (rx_data_vc0[63:56]= 0x00) or 64-bit MRd TLP (rx_data_vc0[63:56]=0x20) the address needs to be read creating a completion TLP with the data. A CplD TLP (Completion with Data) will need to be created using the same Tag from the MRd. This Tag field allows the far end device to associate the completion with a read request. The completion must also not violate the read completion boundary of the far end requestor. The read completion boundary of the requestor can be found in the Link Control Register of the PCI Express capability structure. This information can be found from the IP core using the lnk_cntl_out[3]. If this bit is 0 then the read completion boundary is 64 bytes. If this bit is a 1 then the read completion boundary is 128 bytes. The read completion boundary tells the completer how to segment the CplDs required to terminate the read request. A completion must not cross a read completion boundary and must not exceed the maximum payload size. The Lower Address field of the CplD informs the far end the lower address of the current CplD allow the far end to piece the entire read data back together.

Once the CplD TLP is assembled the TLP needs to be sent and the credits for the MRd need to be released. To release the credits the port nph_processed_vc0 needs to be asserted for 1 clock cycle. This will release the 1 Non-Posted header credit used by a MRd.

The CplD TLP can be sent immediately without checking for completion credits. If a requestor requests data then it is necessary for the requestor to have enough credits to handle the request. If the user still wants to check for credits before sending then the credits for a completion should be checked against the tx_ca_cplh and tx_ca_cpld ports.

2.3.2. As a Requestor

As a requestor the endpoint will issue memory requests to the far end. In order to access memory on the far end device the physical memory address will need to be known. The physical memory address is the address used in the MWr and MRd TLP.

To send a MWr TLP the user must assemble the MWr TLP and then check to see if the credits are available to send the TLP. The credits consumed by a MWr TLP is the length field divided by 4. This value should be compared against the tx_ca_pd port value. If tx_ca_pd[12] is high, this indicates the far end has infinite credits available. The TLP can be sent regardless of the size. An MWr TLP takes 1 Posted header credit. This value can be compared against the tx_ca_ph port. Again, if tx_ca_ph[8] is high, this indicates the far end has infinite credits available.

To send a MRd TLP the user must assemble the MRd TLP and then check to see if the credits are available to send the TLP. The credits consumed by an MRd TLP is 1 Non-Posted header credit. This value should be compared against the tx_ca_nph port value. If tx_ca_nph[8] is high, this indicates the far end has infinite credits available. After a Non-Posted TLP is sent the np_req_pend port should be asserted until all Non-Posted requests are terminated.

In response to a MRd TLP the far end will send a CplD TLP. At this time the rx_st_vc0 will be asserted with rx_data_vc0 providing the first 8 bytes of the TLP. The completion will terminate with the rx_end_vc0 port asserting. The user must now terminate the received CplD. If the core found any errors in the current TLP the error will be indicated on the rx_ecrc_err_vc0 or rx_malf_tlp_vc0 port. An errored TLP does not need to be terminated or release credits. The core will not provide a NAK even if the rx_ecrc_err_vc0 or rx_malf_tlp_vc0 are asserted.

If the TLP is a CplD TLP (rx_data_vc0[63:56]= 0x4A) the data needs to be extracted stored until all CplDs associated with the current Tag are received.

2.4. Unsupported Request Generation

The user ultimately is responsible for sending an Unsupported Request completion based on the capabilities of the user's design. For example, if the user's design only works with memory transactions and not I/O transactions, then I/O transactions are unsupported. These types of transactions require an Unsupported Request completion. There are several instances in which an Unsupported Request must be generated by the user. These conditions are listed below.

- rx_us_req port goes high with rx_st indicating a Memory Read Locked, Completion Locked, or Vendor Defined Message.
- Type of TLP is not supported by the user's design (I/O or memory request)

Table 2.2 shows the types of unsupported TLPs which can be received by the IP core and the user interaction.

Table 2.2. Unsupported TLPs Which Can be Received by the IP

		Unsupported Event Request	"rx_us_req" Port Goes High	User Logic Needs to Send UR Completion	User Needs to Release Credit
"Terminate All Config TLP" is enabled	1	Configuration Read/Write Type*	No	No	No
	2	Memory Read Request - Locked	Yes	Yes	Yes
	3	Locked Completions	Yes	No (Because EP Ignores Locked Completions)	No
	4	Vendor Defined Message Type 0 and Type1	No	Yes	Yes
	Unsup-ported by User Design	TLP with invalid BAR Address	No	Yes (With UR Status)	Yes
		MRd with Inconsistent TLP Type	No	Yes (With UR Status)	Yes
		MWr with Inconsistent TLP Type	No	No (Since MWr is the Posted Req)	Yes
		I/ORd with Inconsistent TLP Type	No	Yes (With UR Status)	Yes
		I/OWr with Inconsistent TLP Type	No	Yes (With UR Status)	Yes
		Msg with Inconsistent TLP Type	No	No (Since Msg is the Posted Req)	Yes
		MsgD with Inconsistent TLP Type	No	No (Since Msg is the Posted Req)	Yes

***Note:** For unsupported by user design events, "inconsistent TLP type" means, for example, MRd request came in for the BAR that only supports I/O, and the other way around.

2.5. Configuration Space

The PCI Express IP core includes the required PCI configuration registers and several optional capabilities. The section will define which registers are included inside the core and how they are utilized.

2.5.1. Base Configuration Type0 Registers

This base configuration Type0 registers are the legacy PCI configuration registers from 0x0-0x3F. The user sets appropriate bits in these registers using the IPexpress GUI. The user is provided with the cmd_reg_out[3:0] to monitor certain bits in the base configuration space.

2.5.2. Power Management Capability Structure

The Power Management Capability Structure is required for PCI Express. The base of this capability structure is located at 0x50. The user sets appropriate bits in these registers using the IPexpress GUI. The user is provided with the pme_status, pme_enable, and pm_power_state ports to monitor and control the Power Management of the endpoint.

2.5.3. MSI Capability Structure

The Message Signaled Interrupt Capability Structure is optional and is included in the IP core. The base of this capability structure is located at 0x70. The number of MSIs is selected in the IPexpress GUI. The user is provided with the msi, mm_enable, and msi_enable ports to utilize MSI.

2.5.4. How to Enable/Disable MSI

The user can enable or disable MSI by setting or resetting bit16 of PCI Express configuration registers (address 70h). This bit value is also shown on “msi_enable” on IP core ports.

This status of MSI is also reflected at bit 16 of the first Dword (Dword 0) of MSI Capability Register Set (which is bit 0 of Message Control Register), and this value is also shown on “msi_enable” port.

2.5.5. How to issue MSI

Up to eight MSI interrupts can be issued. The user can use any bit. Assertion to any of bit 0 to 7 of MSI issues an interrupt of the corresponding MSI number. The IP issues the interrupt at the rising edge of MSI input signal.

2.5.6. PCI Express Capability Structure

The PCI Express Capability Structure is required for PCI Express. The base of this capability structure is located at 0x90. The user sets appropriate bits in these registers using the IPexpress GUI. The user is provided with the dev_cntl_out and lnk_cntl_out ports to monitor certain registers in the design.

2.5.7. Device Serial Number Capability Structure

The Device Serial Number Capability Structure is optional and is included in the IP core. The base of this capability is located at 0x100 which is in the extended register space. The user sets the 64-bit Device Serial Number in the IPexpress GUI.

2.5.8. Advanced Error Reporting Capability Structure

The Advanced Error Reporting Capability Structure is optional and is included in the IP core. The base of this capability is located at 0x1A0 which is in the extended register space. The user is provided the cmpln_tout, cmpltr_abort_np/cmpltr_abort_p, unexp_cmpln, and err_tlp_header ports to provide error conditions to the AER.

2.5.9. Handling of Configuration Requests

Table 2.3 provides the Configuration Space memory map.

Table 2.3. Unsupported TLPs Which Can Be Received by the IP

Port Name	Direction	Function Description
0x0 - 0x3C	Type 0	0 - F
0x40 - 0x4F	Empty	
0x50 - 0x57	Power Management CS	14 - 15
0x58 - 0x6F	Empty	
0x70 - 0x7F	MSI CS	1C - 1D
0x80 - 0x8F	Empty	
0x90 - 0xC3	PCI Express CS	24 - 30
0xA4 - 0xFF	Empty	
0x100 - 0x10B	Device Serial Number CS	Extended 0 - 2
0x10C - 0x17B	Reserved	
0x17C - 0x19F	Empty	
0x1A0 - 0x1C8	AER CS	Extended 128 - 132

The PCI Express core might optionally terminate all configuration requests registers identified in the table. By default, configuration requests to registers that are marked as empty will not be terminated by the core and passed to the user through the receive TLP interface. If the user wishes to implement further capability structures not implemented by the core, or implement PCI-SIG ECNs this could be implemented in the user design. If the user does not want to handle any configuration requests there is an option in the IPexpress/Clarity Designer GUI to have the core terminate all configuration requests. When this is selected, the user will never see a configuration request on the receive TLP interface.

2.6. Wishbone Interface

The optional wishbone interface provides the user with access into the configuration space and select status and control registers. This interface is useful for designs which require knowledge of the entire configuration space, not only those provided as port to the user. When a Wishbone access to the PCI express configuration register space occurs along with configuration access from PCI express link, the later takes the precedence and Wishbone cycle termination will be delayed.

2.6.1. Wishbone Byte/Bit Ordering

The write byte order for Wishbone is:

DAT_I = {upper byte of N+2, lower byte of N+2, upper byte of N, lower byte of N}

The read byte order for Wishbone is different depending on the address range.

1. For an address range of 0x0000-0x0FFF accessing the PCI Express configuration space, the read byte ordering is: DAT_O = {lower byte of N, upper byte of N, lower byte of N+2, upper byte of N+2}
2. For an address range of 0x1000-101F accessing control and status registers inside the PCI Express IP core, the read byte ordering is:

   DAT_O = {upper byte of N+2, lower byte of N+2, upper byte of N, ,lower byte of N}

The bit ordering within a byte is always 7:0.

The memory map for the Wishbone interface is provided in Table 2.4.

Table 2.4. Wishbone Interface Memory Map

Type	Address (hex)	Bits*		Defaults	Description
Status	1008-100B	31:24	R	0	Reserved
		23:20			PHY LSM Status. For X1 Core [23:21] are Reserved.
		19:16	COW		PHY Connection Status / Result of Receiver Detection. For X1 Core [19:17] are Reserved.
		15:12			PHY Receive/Rx Electrical Idle. For x1 Core [15:13] are Reserved.
		11:7	R		LTSSM State: 0 – DETECT,1 – POLLING, 2 – CONFIG, 3 - L0, 4 - L0s, 5 - L1, 6 - L2, 7 – RECOVERY, 8 – LOOPBACK, 9 – HOTRST, 10 - DISABLED*
		6:3			DLL/Link Control SM Status [6] - DL Inactive State [5] - DL Init State [4] - DL Active State [3] - DL Up State
		2:0			Reserved
	100C-100F	31:22	RW	0	Reserved
		21:18			LTSSM goto Loopback For x1 Core [21:19] are Reserved
		17			TLP Debug Mode: TLP bypasses DLL & TRNC check.
		16			PHY/LTSSM Send Beacon
		15			Force LSM Status active
		14			Force Received Electrical Idle
		13			Force PHY Connection Status
		12			Force Disable Scrambler (to PCS)
		11			Disable scrambling bit in TS1/TS2
		10			LTSSM go to Disable
		9			LTSSM go to Detect
		8			LTSSM go to HotReset
		7			LTSSM go to L0s
		6			LTSSM go to L1
		5			LTSSM go to L2
		4			LTSSM go to L0s and Tx FTS

		3			Reserved
		2			LTSSM go to Recovery
		1			LTSSM go to Config
		0			LTSSM no training
	1010-101	31:30	R/W	GUI	Reserved
		29:16			ACK/NAK Latency Timer
		15			Reserved
		14:10			Number of FTS
		9:0			SKP Insert Counter
	1014-1017	31:18	R/W	Set in IP	Reserved
		17:11			Update Frequency for Posted Header
		10:0			Update Frequency for Posted Data
	1018-101B	31:18	R/W	Set in IP	Reserved
		17:11			Update Frequency for Non-Posted Header
		10:0			Update Frequency for Non-Posted Data
	101C-101F	31:18	R/W	Set in IP	Reserved
		17:11			Update Frequency for Completion Header
		10:0			Update Frequency for Completion Data
	1020-1023	31:12	R/W	Set in IP	Reserved
		11:0			FC Update Timer
	1023-1027	31:8	R/W	0	Reserved
		7:0			Link Number

***Note:** R - Read Only; R/W - Read and Write; COW - Clear On Write

2.7. Error Handling

The IP Core handles DLL errors. User logic does not need to control any of DLL errors. When the IP receives NAK, the IP does not report outside of IP and retransmit TLP in retry buffer.

Table 2.5 lists physical layer error messages, error type, and how the IP responds to the error. Table 2.5 corresponds to Table 6-2 of the PCI Express Base Specification Version 1.1. The IP automatically responds to all Physical Layer errors. No user logic interaction is required.

Table 2.5. Physical Layer Error List

Error Name	Error Type	IP Action	IP Handling from User Logic	References
Receiver Error	Correctable	Send ERR_COR to root complex.	Nothing is required.	Section 4.2.1.3 Section 4.2.2.1 Section 4.2.4.4 Section 4.2.6

Table 2.6 lists data link layer error messages, error type, and how the IP responds to the errors. Table 2-6 corresponds to Table 6-3 of the PCI Express Base Specification Version 1.1. The IP automatically responds to all Data Link Layer errors. No user logic interaction is required.

Table 2.6. Data Link Layer Error List

Error Name	Error Type	IP Action	IP Handling from User Logic	References
Bad TLP	Correctable	Send ERR_COR to root complex.	Nothing is required.	Section 3.5.2.1
Bad DLLP	Correctable	Send ERR_COR to root complex.	Nothing is required.	Section 3.5.2.1
Replay Timeout	Correctable	Send ERR_COR to root complex.	Nothing is required.	Section 3.5.2.1
REPLAY NUM	Correctable	Send ERR_COR to root complex.	Nothing is required.	Section 3.5.2.1

Rollover				
Data Link Layer Protocol Error	Uncorrectable (fatal)	- If the severity level is fatal, send ERR_FATAL to root complex. - If the severity level is non-fatal, send ERR_NONFATAL to root complex.	Nothing is required.	Section 3.5.2.1
Surprise Down	Uncorrectable (fatal)	Not required to implement for endpoint, per section 7.8.6, page 380	Nothing is required.	Section 3.5.2.1

Table 2.7 lists transaction layer error messages, error type, how the IP responds to the errors, and any IP handling that requires user logic interaction. The table corresponds to Table 6-4 of the PCI Express Base Specification Version 1.1.

Table 2.7. Transaction Layer Error List

Error Name	Error Type	IP Action	IP Handling from User Logic	References
Poisoned TLP Received	Uncorrectable (non-fatal)	**Automatically:** - If the severity level is non-fatal, send ERR_COR for the advisory non-fatal error to root croot complex. - If the severity level is fatal, send ERR_FATAL to root complex. - Log the header of the poisoned TLP. - Release credit.	This is a special case where the TLP is passed to the user, but the core handles the error message automatically. The data is passed to the user logic for handling, per section 2.7.2.2.	Section 2.7.2.2
ECRC Check Failed	Uncorrectable (non-fatal)	**Automatically (if ECRC checking is supported):** - If the severity level is non-fatal, send the non-fatal error to root complex. - If the severity level is fatal, send ERR_FATAL to root complex. - Log the header of the TLP that encountered the ECRC error. - Assert rx_ecrc_err_vc0 to userlogic side.	- Packet is passed to the user for discard. - No completion for non-posted is returned because anything can be wrong in the packet. - Release credit.	Section 2.7.1
Unsupported Request (UR) Configuration Type 1	Uncorrectable (non-fatal)	**Automatically:** - If the severity level is non-fatal, send ERR_COR for the advisory non-fatal error to root complex. - If the severity level is fatal, send	Nothing is required.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3

		ERR_FATAL to root complex. - Release credit. - Send UR completion. - Log the header of the TLP.		Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2
Unsupported Request (UR) MrdLK	Uncorrectable (non-fatal)	**Automatically:** - If the severity level is non-fatal, send ERR_COR for the advisory non-fatal error to root complex. - If the severity level is fatal, send ERR_FATAL to root complex. - Assert rx_ur_req_vc0 to userlogic side. - Log the header of the TLP.	- Release credit. - Send UR completion.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2
Unsupported Request (UR) Vector Defined Message Type 0	Uncorrectable (non-fatal)	**Based on assertion of ur_p_ext input:** - If the severity level is non-fatal, send ERR_NONFATAL because this is an unsupported posted request which is not advisory non-fatal. - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	- Assert ur_p_ext input. - Assert err_tlp_header[127:0] inputs. - Release credit. - Send completion with UR status for vendor define Type0 if not sup- ported.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2
Unsupported Request (UR) Locked Completion	Uncorrectable (non-fatal)	**Based on assertion of ur_np_ext/ur_p_ext input:** - If the severity level is non-fatal and the request type is non-posted, send ERR_COR for the advisory non-fatal error to root complex. - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	Assert cmpln_tout input.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2

Unsupported Request (UR) Request type not supported	Uncorrectable (non-fatal)	**Based on assertion of ur_np_ext/ur_p_ext input:** - If the severity level is non-fatal and request type is non-posted, send ERR_COR for the advisory non-fatal error to root complex. Send ERR_NONFATAL for posted request which is not advisory non-fatal. - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	- Assert ur_np_ext/ur_p_ext input. - Assert err_tlp_header[127:0] inputs. - Release credit. - Send UR completion if request is non-posted.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2
Unsupported Request (UR) Request not referencing address space mapped within the device	Uncorrectable (non-fatal)	**Based on assertion of ur_np_ext/ur_p_ext input:** - If the severity level is non-fatal and request type is non-posted, send ERR_COR for the advisory non-fatal error to root complex. Send ERR_NONFATAL for posted request which is not advisory non-fatal. - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	- Assert ur_np_ext/ur_p_ext input. - Assert err_tlp_header[127:0] input. - Release credit. - Send UR completion if request is non-posted.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2
Unsupported Request (UR) Message code with unsupported or undefined message code	Uncorrectable (non-fatal)	**Based on assertion of ur_p_ext input:** - If the severity level is non-fatal, send ERR_NONFATAL (Unsupported Posted request is not advisory non-fatal). - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	- Assert ur_p_ext input. - Assert err_tlp_header[127:0] inputs. - Release credit.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2
Completion Timeout	Uncorrectable (non-fatal)	**Based on assertion of cmpln_tout input:**	Assert cmpln_tout input.	Section 2.8

		- If the severity level is non-fatal, send ERR_COR for the advisory non-fatal error to root complex. - If the severity level is fatal, send ERR_FATAL to root complex.		
Completer Abort	Uncorrectable (non-fatal)	**Based on assertion of cmpltr_abort_p/cmpltr_abort_np input:** - If the severity level is non-fatal, send ERR_COR for the advisory non-fatal error to root complex. - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	- Assert cmpltr_abort_p/cmpltr_ abort_np input. input. - Assert err_tlp_header[127:0] input.	Section 2.3.1
Unexpected Completion	Uncorrectable (non-fatal)	**Based on assertion of unexp_cmpln:** - If the severity level is non-fatal, send ERR_COR for the advisory non-fatal error to root complex. - If the severity level is fatal, send ERR_FATAL to root complex. **Based on assertion of err_tlp_header[127:0] input:** - Log the header of the TLP.	- Assert unexp_cmpln input. - Assert err_tlp_header[127:0] input.	Section 2.3.2
Receiver Overflow	Uncorrectable (fatal)	**Automatically:** - If the severity level is fatal, send ERR_FATAL to root complex. - If the severity level is non-fatal,send ERR_NONFATAL to root complex.	Nothing is required.	Section 2.6.1.2
Flow Control Protocol Error	Uncorrectable (fatal)	**Automatically:** - If the severity level is fatal, send ERR_FATAL to root complex. - If the severity level is non-fatal,	Nothing is required.	Section 2.6.1

		send ERR_NONFATAL to root complex.		
Malformed TLP	Uncorrectable (fatal)	**Automatically:** - If the severity level is fatal, send ERR_FATAL to root complex. - If the severity level is non-fatal, send ERR_NONFATAL to root complex. - Assert rx_malf_tlp_vc0. - Log the header of the TLP.	Nothing is required.	Section 2.2.8.6 Section 2.3.1 Section 2.3.2 Section 2.7.2.2 Section 2.9.1 Section 5.3.1 Section 6.2.3 Section 6.2.6 Section 6.2.8.1 Section 6.5.7 Section 7.3.1 Section 7.3.3 Section 7.5.1.1 Section 7.5.1.2

3. Parameter Settings

The Clarity Designer tool is used to create IP and architectural modules in the Diamond software. Refer to the IP Core Generation and Evaluation section for a description on how to generate the IP.

Table 3.1 provides the list of user configurable parameters for the PCI Express IP core. The parameter settings are specified using the PCI Express IP core Configuration GUI in Clarity Designer. The numerous PCI Express parameter options are partitioned across multiple GUI tabs as shown in this chapter.

Table 3.1. IP Core Parameters

Parameters	2.5G IP Core Range/Options	5G IP Core Range/Options	Default
General			
PCI Express Link Configuration	Native x4, x4 Downgraded 1x or x2, Native x1	Native x1, x2 Downgraded x1	Native x1 for 2.5G IP core x2 Downgraded x1 for 5G IP core
Endpoint Type	PCI Express Endpoint, Legacy Endpoint	PCI Express Endpoint	PCI Express Endpoint
Include Master Loop back data path	Yes/No	Yes/No	No
Include Wishbone interface	Yes/No	Yes/No	No
Configuration Registers not required	Yes/No	Yes/No	No
PCS Pipe Options			
Config			
Configuration	x1, x2, x4	x1, x2	X1
Data Width	8-16	16	8 for 2.5G IP Core 16 for 5G IP core
DCU (for ECP5 and ECP-5G)	DCU0, DCU1	DCU0 (ECP5-5G only)	DCU0
Channel (for LatticeECP3)	Ch0, Ch1	*Not Supported*	Ch0
Flow Control			
Update Flow Control Generation Control			
Number of PH credits between UpdateFC P	1-127	1-127	8
Number of PD credits between UpdateFC P	1-2047	1-2047	255
Number of NPH credits between UpdateFC NP	1-127	1-127	8
Number of NPD credits between UpdateFC NP	1-2047	1-2047	255
Worst case number of 125 MHz clock cycles between UpdateFC	3650-4095	3650-4095	4095
Initial Receive Credits			
Infinite PH Credits	Yes/No	Yes/No	No
Initial PH credits available	1-127	1-127	0
Infinite PD Credits	Yes/No	Yes/No	No
Initial PD credits available	8-255	8-255	0
Initial NPH credits available	1-32	1-32	0
Initial NPD credits available	8-2047	8-2047	0
Configuration Space			
Type0 Config Space			
Device ID	0000-ffff	0000-ffff	0000

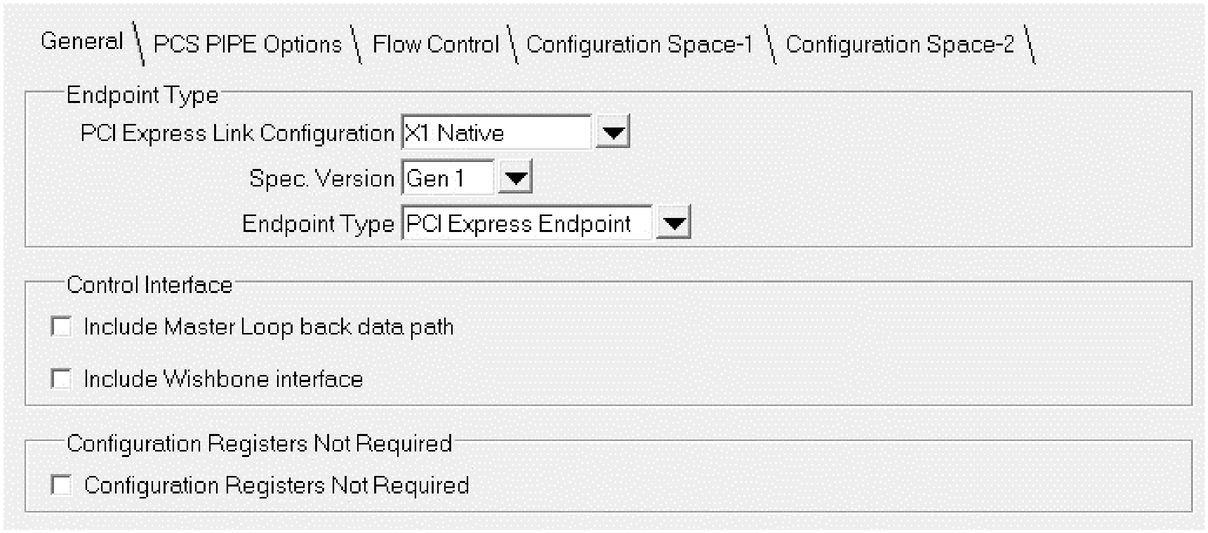
Vendor ID	0000-ffff	0000-ffff	0000
Class Code	000000-ffffff	000000-ffffff	000000
Rev ID	00-ff	00-ff	00
BIST	00-ff	00-ff	00
Header Type	00-ff	00-ff	00
Bar0	00000000-ffffffff	00000000-ffffffff	00000000
Bar0 Enable	Yes/No	Yes/No	No
Bar1	00000000-ffffffff	00000000-ffffffff	00000000
Bar1Enable	Yes/No	Yes/No	No
Bar2	00000000-ffffffff	00000000-ffffffff	00000000
Bar2 Enable	Yes/No	Yes/No	No
Bar3	00000000-ffffffff	00000000-ffffffff	00000000
Bar3 Enable	Yes/No	Yes/No	No
Bar4	00000000-ffffffff	00000000-ffffffff	00000000
Bar4 Enable	Yes/No	Yes/No	No
Bar5	00000000-ffffffff	00000000-ffffffff	00000000
Bar5 Enable	Yes/No	Yes/No	No
CardBus CIS Pointer	00000000-ffffffff	00000000-ffffffff	00000000
Subsystem ID	0000-ffff	0000-ffff	0000
Subsystem Vendor ID	0000-ffff	0000-ffff	0000
ExpROM Base Addr	00000000-ffffffff	00000000-ffffffff	00000000
Expansion ROM Enable	Yes/No	Yes/No	No
Load IDs from Ports	Yes/No	Yes/No	No
Power Management Capability Structure			
Power Management Cap Reg [31-16]	0000-ffff	0000-ffff	0003
Date Scale Multiplier	0-3	0-3	0
Power Consumed in D0 (Watts)	00-ff	00-ff	00
Power Consumed in D0 (Watts)	00-ff	00-ff	00
Power Consumed in D1 (Watts)	00-ff	00-ff	00
Power Consumed in D2 (Watts)	00-ff	00-ff	00
Power Consumed in D3 (Watts)	00-ff	00-ff	00
Power Dissipated in D0 (Watts)	00-ff	00-ff	00
Power Dissipated in D1 (Watts)	00-ff	00-ff	00
Power Dissipated in D2 (Watts)	00-ff	00-ff	00
Power Dissipated in D3 (Watts)	00-ff	00-ff	00
Power Dissipated in D4 (Watts)	00-ff	00-ff	00
MSI Capability Structure			
Use Message Signaled Interrupts	Yes/No	Yes/No	Yes
Number of Messages Requested	1-8	1-8	1
PCI Capability Structure			
Next Capability Pointer	00-ff	00-ff	00
PCIe Capability Version	1 or 2	1 or 2	1
Max Payload Size Bytes	128, 256, 512, 1024, 2048, 4096	128, 256, 512, 1024, 2048, 4096	128
Device Capabilities Register [28:3]	0000000-fffffff	0000000-fffffff	0000000
Enable Relaxed Ordering	Yes/No	Yes/No	Yes
Maximum Link Width	1, 2, 4	1, 2, 4	4
Device Capabilities 2 Register [4:0]	00-1f	00-1f	11

Device Serial Number			
Device Serial Number Version	1	1	1
Device Serial Number	0000000000000000-ffffffffffffffff	0000000000000000-ffffffffffffffff	0000000000000000
Advanced Error Reporting			
Use Advanced Error Reporting	Yes/No	Yes/No	No
Advanced Error Reporting Version	1	1	Disabled
Include ECRC support	Yes/No	Yes/No	No
Terminate All Config TLPs			
Terminate All Config TLPs	Yes/No	Yes/No	Yes
User Extended Capability Structure	000-fff	000-fff	Disabled

The default values shown in the following pages are those used for the PCI Express reference design. IP core options for each tab are discussed in further detail.

3.1. General Tab

Figure 3-1 shows the contents of the General tab.



Endpoint Type

PCI Express Link Configuration: X1 Native

Spec. Version: Gen 1

Endpoint Type: PCI Express Endpoint

Control Interface

☐ Include Master Loop back data path

☐ Include Wishbone interface

Configuration Registers Not Required

☐ Configuration Registers Not Required

Figure 3.1. PCI Express IP Core General Options

The General tab consists of the following parameters:

3.1.1. PCI Express Link Configuration

Specifies the link width and type of core to be used.

25G IP core:

- Native x4 - This is a x4 link width using a 64-bit data path. This configuration can dynamically downgraded to a x2 or x1 link width.
- Downgraded x1 - This is a x1 link width using a 64-bit datapath.
- Downgraded x2 - This is a x2 link width using a 64-bit datapath.
- Native x1 - This is a x1 link width only using a 16-bit datapath.

5G IP core:

- Native x2 and Downgraded x1 using 64-bit datapath.

3.1.2. Spec. Version

Specifies the version of the PCI express specification.

- Gen1 – Supports version 1.1 of the PCI express specification.
- Gen 2 – Supports version 2.1 of the PCI express specification for 2.5.Gbps and 5.0 Gbps.

3.1.3. Endpoint Type

This option allows the user to choose between PCI Express Endpoint or Legacy Endpoint.

- Legacy Endpoint is permitted to support locked access.
- PCI Express Endpoint does not support locked access. PCI Express Endpoint must treat a MRdLk request as an unsupported request. Refer to the PCI Express Base Specification Version 1.1, sections 6.5.6 and 6.5.7, for more details.

3.1.4. Include Master Loopback Data Path

This option includes additional transmit and receive data path ports to the IP, if the device needs to be used as a loopback master in Loopback state of the LTSSM. In Table 2.1, refer to following I/O ports: tx_lbk_rdy, tx_lbk_kcntl, tx_lbk_data, rx_lbk_kcntl and rx_lbk_data

3.1.5. Include Wishbone Interface

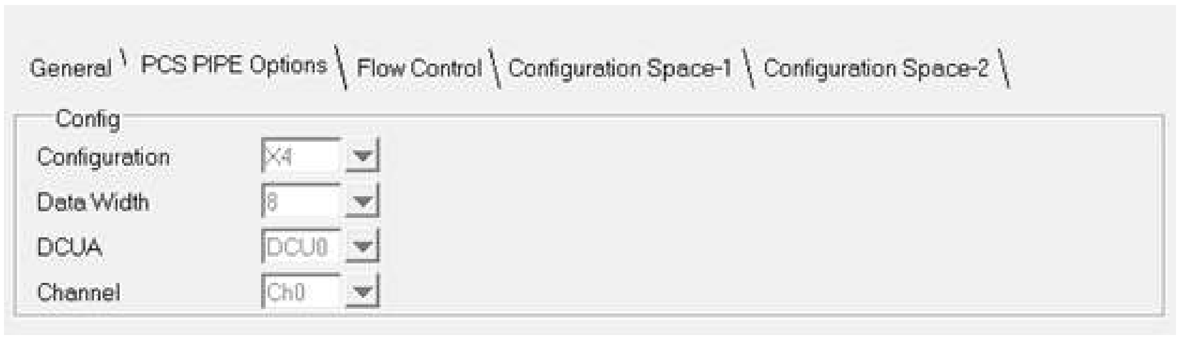
This option includes a Wishbone interface to access certain features of the PCI Express core (see Table 2.1).

3.1.6. Configuration Registers Not Required

This option excludes implementation of configuration registers space in the PCI Express core.

3.1.7. PCS Pipe Options Tab

Figure 3.2 showS the contents of the PCS Pipe Options tab. This is not supported in PCI Express 5G IP core.



Configuration: X4
Data Width: 8
DCUA: DCU0
Channel: Ch0

Figure 3.2. PCI Express IP Core PCS Pipe Options

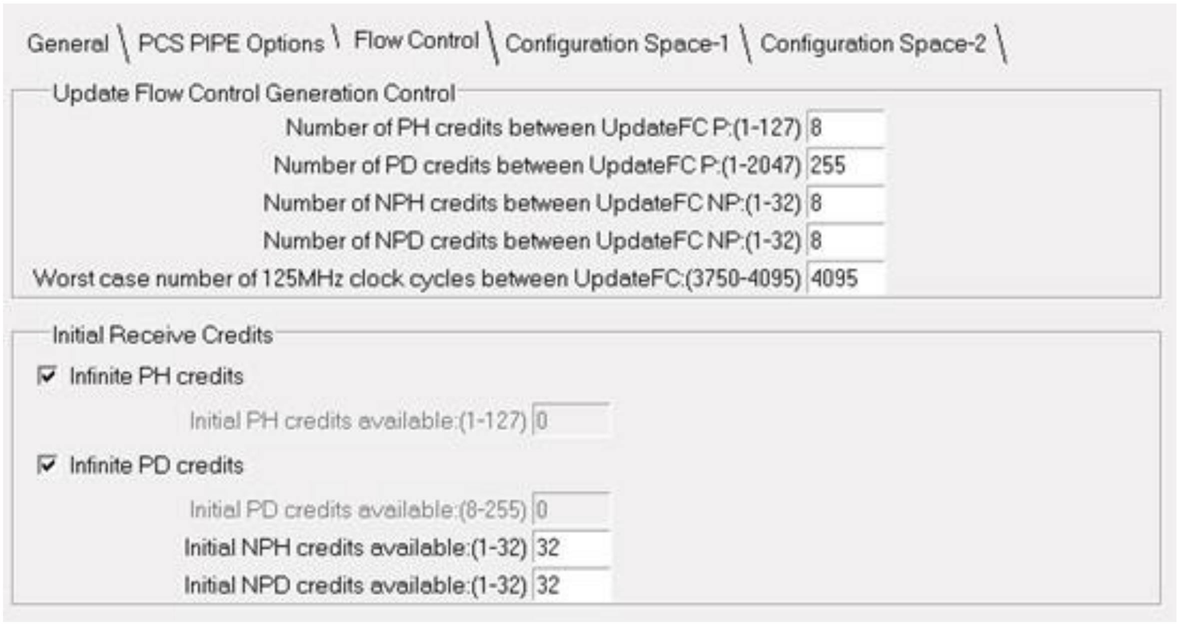
3.1.8. Config

The options under this changes based on the selection of PCI Express Link Configuration drop down list in the General tab selection. This tab in not available for 5G IP core. To select DCUA, use Clarity Designer.

- If Native x4, these options cannot be modified.
- If Native x2, DCUA (DCU0/DCU1) of SERDES can be selected.
- If Native x1, DCUA (DCU0/DCU1) and Channel (Ch0/Ch1) of a SERDES can be selected.
- If Downgraded x1, DCUA (DCU0/DCU1) of SERDES can be selected.
- If Downgraded x2, DCUA (DCU0/DCU1) of SERDES can be selected.

3.2. Flow Control Tab

Figure 3.3 shows the contents of the Flow Control tab.



Update Flow Control Generation Control

Number of PH credits between UpdateFC P:(1-127) 8
Number of PD credits between UpdateFC P:(1-2047) 255
Number of NPH credits between UpdateFC NP:(1-32) 8
Number of NPD credits between UpdateFC NP:(1-32) 8
Worst case number of 125MHz clock cycles between UpdateFC:(3750-4095) 4095

Initial Receive Credits

☑ Infinite PH credits
Initial PH credits available:(1-127) 0
☑ Infinite PD credits
Initial PD credits available:(8-255) 0
Initial NPH credits available:(1-32) 32
Initial NPD credits available:(1-32) 32

Figure 3.3. PCI Express IP Core Flow Control Options

3.2.1. Update Flow Control Generation Control

There are two times when an UpdateFC DLLP will be sent by the IP core. The first is based on the number of TLPs (header and data) that were processed. The second is based on a timer.

For both controls a larger number will reduce the amount of UpdateFC DLLPs in the transmit path resulting in more throughput for the transmit TLPs. However, a larger number will also increase the latency of releasing credits for the far end to transmit more data to the endpoint. A smaller number will increase the amount of UpdateFC DLLPs in the transmit path. But, the far end will see credits available more quickly.

3.2.2. Number of P TLPs Between UpdateFC

This control sets the number of Posted Header TLPs that have been processed before sending an UpdateFC-P.

3.2.3. Number of PD TLPs Between UpdateFC

This control sets the number of Posted Data TLPs (credits) that have been processed before sending an UpdateFC-P.

3.2.4. Number of NP TLPs Between UpdateFC

This control sets the number of Non-Posted Header TLPs that have been processed before sending an UpdateFC- NP.

3.2.5. Number of NPD TLPs Between UpdateFC

This control sets the number of Non-Posted Data TLPs (credits) that have been processed before sending an UpdateFC-NP.

3.2.6. Worst Case Number of 125 MHz Clock Cycles Between UpdateFC

This is the timer control that is used to send UpdateFC DLLPs. The core will send UpdateFC DLLPs for all three types when this timer expires regardless of the number of credits released.

3.2.7. Initial Receive Credits

During the Data Link Layer Initialization InitFC1 and InitFC2 DLLPs are transmitted and received. This function is to allow both ends of the link to advertise the amount of credits available. The following controls are used to set the amount of credits available that the IP core will advertise during this process.

3.2.8. Infinite PH Credits

This option is used if the endpoint will have an infinite buffer for PH credits. This is typically used if the endpoint will terminate any PH TLP immediately.

3.2.9. Initial PH Credits Available

If PH infinite credits are not used then this control allows the user to set an initial credit value. This will be based on the receive buffering that exists in the user's design connected to the receive interface.

3.2.10. Infinite PD Credits

This option is used if the endpoint will have an infinite buffer for PD credits. This is typically used if the endpoint will terminate any PD TLP immediately.

3.2.11. Initial PD Credits Available

If PD infinite credits are not used then this control allows the user to set an initial credit value. This will be based on the receive buffering that exists in the user's design connected to the receive interface.

3.2.12. Initial NPH Credits Available

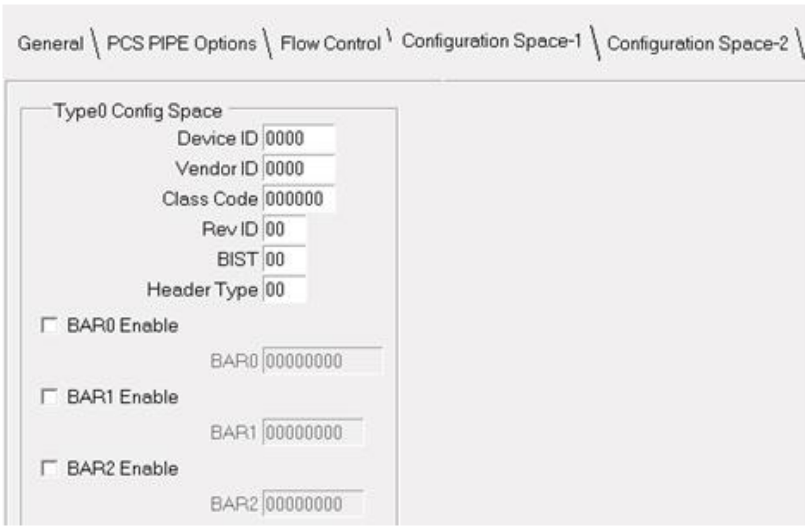
This option allows the user to set an initial credit value. This will be based on the receive buffering that exists in the user's design connected to the receive interface.

3.2.13. Initial NPD Credits Available

This option allows the user to set an initial credit value. This will be based on the receive buffering that exists in the user's design connected to the receive interface.

3.3. Configuration Space - 1 Tab

Figure 3.4 shows the contents of the Configuration Space - 1 tab.



Type0 Config Space

Device ID 0000
Vendor ID 0000
Class Code 000000
Rev ID 00
BIST 00
Header Type 00
☐ BAR0 Enable
BAR0 00000000
☐ BAR1 Enable
BAR1 00000000
☐ BAR2 Enable
BAR2 00000000

Figure 3.4. PCI Express IP Core Configuration Space - 1 Options

3.3.1. Type 0 Config Space

This section provides relevant PCI Express settings for the legacy Type0 space.

3.3.2. Device ID

This 16-bit read only register is assigned by the manufacturer to identify the type of function of the endpoint.

3.3.3. Vendor ID

This 16-bit read only register assigned by the SIG to identify the manufacturer of the endpoint.

3.3.4. Class Code

This 24-bit read only register is used to allow the OS to identify the type of endpoint.

3.3.5. Revision ID

This 8-bit read only register is assigned by the manufacturer and identifies the revision number of the endpoint.

3.3.6. BIST

This 8-bit read only register indicates if a Built-In Self-Test is implemented by the function.

3.3.7. Header Type

This 8-bit read only register identifies the Header Type used by the function.

3.3.8. BAR Enable

This option enables the use of the particular Base Address Register (BAR).

3.3.9. BAR

This field allows the user to program the BAR to request a specific amount of address space from the system. If using 64-bit BARs then the BARs will be paired. BARs 0 and 1 will be paired with BAR0 for the LSBs and BAR1 for the MSBs. BARs 2 and 3 will be paired with BAR2 for the LSBs and BAR3 for the MSBs. BARs 4 and 5 will be paired with BAR4 for the LSBs and BAR5 for the MSBs.

For more details on BAR register bits, refer to the Configuration Space section of the PCI Local Bus Specification Revision 3.0. The following section provides an example for requesting address space by setting BAR registers. Bit [0] – 0 for memory space request. 1 for I/O space request. Bits [2:1] – 00 for 32-bit memory address space. 10 for 64-bit memory address space. Bit [3] – 1 for prefetchable memory. 0 for non-prefetchable memory. Bits [31:4] – Indicate the size of required address space by resetting least significant bits. Example 1: 32'hFFFF_F000 requests for memory space(bit[0]=0),32-bit address space(bit[2:1]=00), non-prefetchable memory(bit[3]=0) and 4KB address space (bits[31:4]=FFFF_F00) Example 2: 32'hFFF0_0000 requests for memory space(bit[0]=0),32-bit address space(bit[2:1]=00), non-prefetchable memory(bit[3]=0) and 1MB address space (bits[31:4]=FFF0_000)

3.3.10. CardBus CIS Pointer

This is an optional register used in card bus system architecture and points to the Card Information Structure (CIS) on the card bus device. Refer to PCI Local Bus Specification Revision 3.0 for further details.

3.3.11. Subsystem ID

This 16-bit read only register assigned by the manufacturer to identify the type of function of the endpoint.

3.3.12. Subsystem Vendor ID

This 16-bit read only register assigned by SIG to identify the manufacturer of the endpoint.

3.3.13. Expansion ROM Enable

This option enables the Expansion ROM to be used.

3.3.14. Expansion ROM Enable

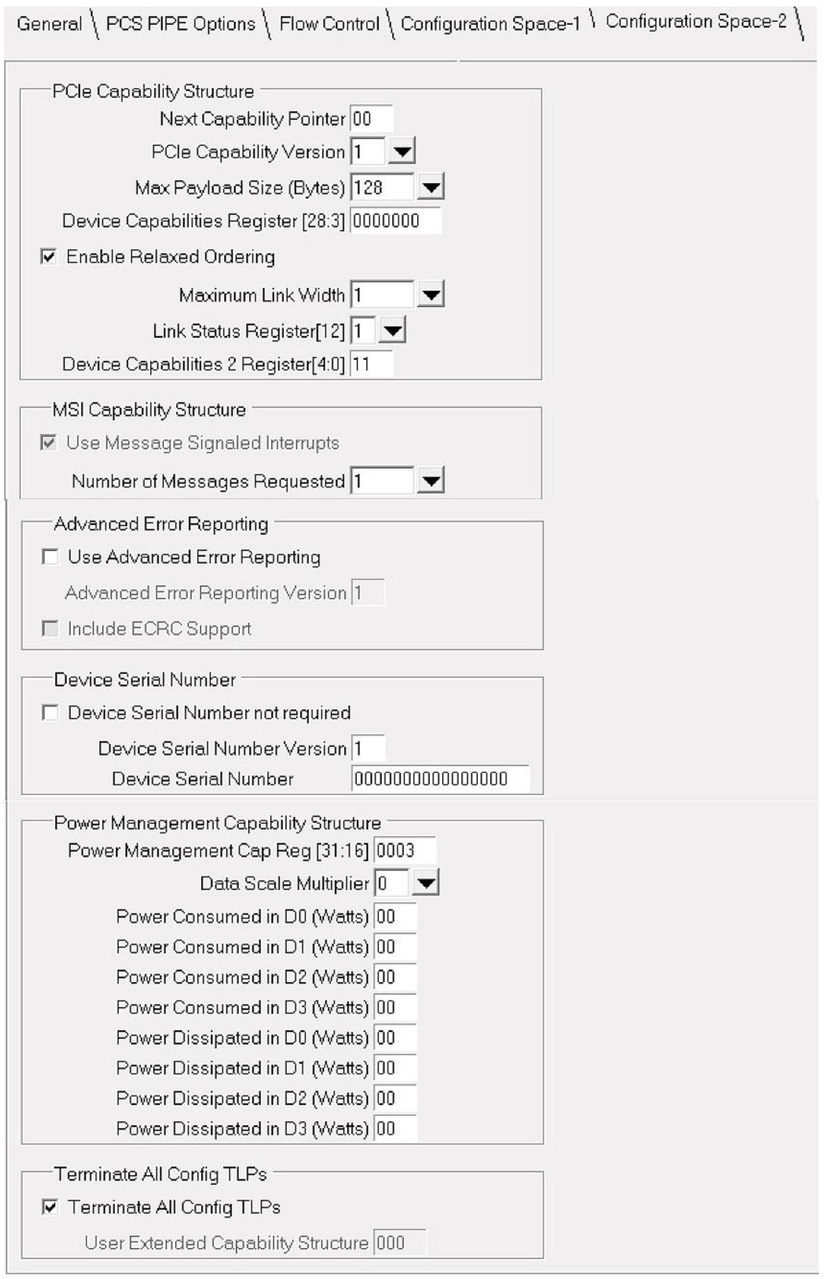
The Expansion ROM base address if one is used in the solution.

3.3.15. Load IDs From Ports

This option provides ports for the user to set the Device ID, Vendor ID, Class Code, Rev ID, Subsystem ID, and Subsystem Vendor ID from the top level of the core. This is useful for designs which use the same hardware with different software drivers.

3.4. Configuration Space - 2 Tab

Figure 3.5 shows the contents of the Configuration Space tab.



PCIe Capability Structure

Next Capability Pointer 00
PCIe Capability Version 1
Max Payload Size (Bytes) 128
Device Capabilities Register [28:3] 0000000
☑ Enable Relaxed Ordering
Maximum Link Width 1
Link Status Register[12] 1
Device Capabilities 2 Register[4:0] 11

MSI Capability Structure

☑ Use Message Signaled Interrupts
Number of Messages Requested 1

Advanced Error Reporting

☐ Use Advanced Error Reporting
Advanced Error Reporting Version 1
☐ Include ECRC Support

Device Serial Number

☐ Device Serial Number not required
Device Serial Number Version 1
Device Serial Number 0000000000000000

Power Management Capability Structure

Power Management Cap Reg [31:16] 0003
Data Scale Multiplier 0
Power Consumed in D0 (Watts) 00
Power Consumed in D1 (Watts) 00
Power Consumed in D2 (Watts) 00
Power Consumed in D3 (Watts) 00
Power Dissipated in D0 (Watts) 00
Power Dissipated in D1 (Watts) 00
Power Dissipated in D2 (Watts) 00
Power Dissipated in D3 (Watts) 00

Terminate All Config TLPs

☑ Terminate All Config TLPs
User Extended Capability Structure 000

Figure 3.5. PCI Express IP Core Configuration Space - 2 Options

3.5. PCI Express Capability Structure Options

These controls allow the user to control the PCI Express Capability Structure.

3.5.1. Next Capability Pointer

This control defines the pointer to additional non-extended capability implemented in the user application design. The default is 0 to indicate that there are no user implemented non-extended capability. If the pointer is set to a non-zero value, “Terminate All Config TLPs” must not be selected.

3.5.2. PCI Express Capability Version

Indicates the version of the PCI Express Capability Register. This number must be set to 2 for PCI Express version 2.0.

3.5.3. Max Payload Size

This option allows the endpoint to advertise the max payload size supported by the endpoint, and is used to size the Retry Buffer contained in the Data Link Layer. The user should select the largest size payload size that will be used in the application. The option 512B retry buffer size should also be selected for payload size of 128B and 256B. The retry buffer uses Embedded Block RAM (EBR) and will be sized accordingly. Table 3.2 provides a total EBR count for the core based on Max Payload Size.

Table 3.2. Total EBR Count Based on Max Payload Size (2.5G IP Core)

Max Payload Size	LatticeECP3/ECP5 Native x1	LatticeECP3/ECP5 Native x4	LatticeECP3/ECP5 Downgraded x2/x1
512B	4	11	11
1KB	5	11	11
2KB	9	13	13
4KB	15	20	20

3.5.4. Device Capabilities Register (27:3)

This 25-bit field sets the Device Capabilities Register bits 27:3.

3.5.5. Enable Relaxed Ordering

Relaxed ordering is the default setting for PCI Express. If the PCI Express link does not support relaxed ordering then this checkbox should be cleared. This feature does not change the behavior of the core, only the setting of this bit in the PCI Express capability structure. The user will be required to ensure strict ordering is enforced by the transmitter.

3.5.6. Maximum Link Width

This option sets the maximum link width advertised by the endpoint. This control should match the intended link width of the endpoint.

3.5.7. Device Capabilities 2 Register (4:0)

This 5-bit field sets the Device Capabilities Register bits 4:0.

3.5.8. MSI Capability Structure Options

These controls allow the user to include MSI and request a certain number of interrupts

3.5.9. Use Message Signaled Interrupts

This option includes MSI support in the IP core.

3.5.10. Number of Messages Requested

This number specifies how many MSIs will be requested by the endpoint of the system. The system will respond with how many interrupts have been provided. The number of interrupts provided can be found on the mm_enable port of the IP core.

3.5.11. Advanced Error Reporting Options

These controls allow the user to include AER.

3.5.12. Use Advanced Error Reporting

This control will include AER in the IP core. AER is used to provide detailed information on the status of the PCI Express link and error TLPs.

3.5.13. Advanced Error Reporting Version

Indicates the version of the Advanced Error Reporting Capability. This number must always be set to 1 for v1.1.

3.5.14. Device Serial Number Version

Indicates the version of the Device Serial Number Capability. This number must always be set to 1 for v1.1.

3.5.15. Device Serial Number

This 64-bit value is provided in the IP core through the Device Serial Number Capability Structure.

3.5.16. Power Management Capability Structure

This section includes options for the Power Management Capability Structure. This structure is used to pass power information from the endpoint to the system. If power management is not going to be used by the solution then all fields can remain in the default state.

3.5.17. Power Management Cap Reg (31:16)

This field sets the Power Management Capabilities (PMC) register bits 31:16.

3.5.18. Data Scale Multiplier

This control sets the Data Scale Multiplier used by system to multiplier the power numbers provided by the end- point.

3.5.19. Power Consumed in D0, D1, D2, D3

These controls allow the user to specify the power consumed by the endpoint in each power state D0, D1, D2, and D3. The user specifies Watts as an 8-bit hex number.

3.5.20. Power Dissipated in D0, D1, D2, D3

These controls allow the user to specify the power dissipated by the endpoint in each power state D0, D1, D2, and D3. The user specifies Watts as an 8-bit hex number.

3.5.21. Terminate All Configuration TLPs

If enabled, this control will allow the core to terminate all Configuration requests. The user will not need to handle any configuration requests in the user's design.

3.5.22. User Extended Capability Structure

This control defines the pointer to additional non-extended capability implemented in the user application design. The default is 0 to indicate there is no user implemented non-extended capability. If the pointer is set to a non-zero value, “Terminate All Config TLPs” must not be selected.

4. IP Core Generation and Evaluation

This chapter provides information on how to generate the PCI Express IP core using the Clarity Designer in Diamond software. Additional information on how to use the Clarity Designer tool to create designs using multiple IP Cores is given in Clarity Designer User Manual.

4.1. Licensing the IP Core

An IP license is required to enable full, unrestricted use of the PCI Express IP core in a complete, top-level design for the ECP5 and ECP5-5G families.

4.1.1. Licensing Requirements for ECP5 and ECP5-5G

An IP license that specifies the IP core (PCI Express), device family and configuration (x1 or x2 or x4) is required to enable full use of the PCI Express IP core in LatticeECP3, ECP5 and ECP5-5G devices. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/Products/DesignSoftwareAndIP.aspx

Users may download and generate the PCI Express IP core for ECP5 and ECP5-5G and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The PCI Express IP core for LatticeECP3, ECP5 and ECP5-5G also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license (see the Hardware Evaluation section for further details). However, a license is required to enable timing simulation, to open the design in the Diamond tool, and to generate bit streams that do not include the hard- ware evaluation timeout limitation.

Note that there are no specific IP licensing requirements associated with a x4 core that functionally supports the ability to downgrade to a x1/x2 configuration. Such a core is licensed as a x4 configuration.

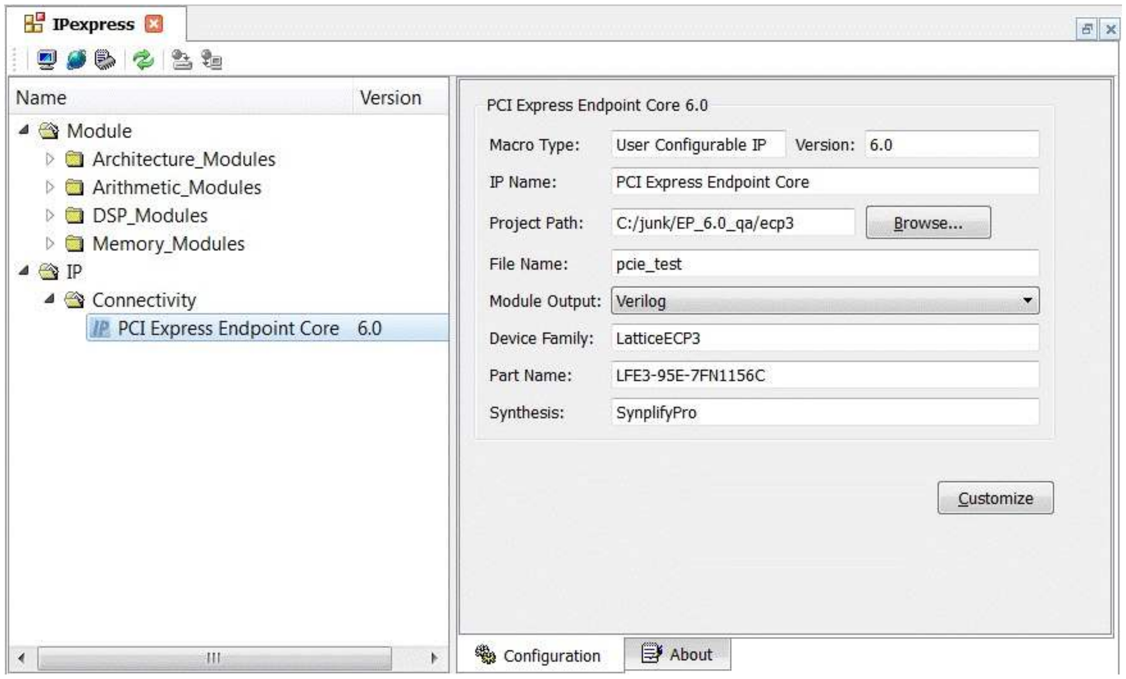
4.2. IPexpress Flow for LatticeECP3 Devices

4.2.1. Getting Started

The PCI Express IP core is available for download from the Lattice IP server using the IPexpress tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4.1.

To generate a specific IP core configuration specify:

- Project Path – Path to the directory where the generated IP files will be located.
- File Name – "username" designation given to the generated IP core and corresponding folders and files.
- Module Output – Verilog or VHDL.
- Device Family – Device family to which IP is to be targeted (e.g. LatticeECP3, ECP5 etc.). Only families that support the particular IP core are listed.
- Part Name – Specific targeted part within the selected device family.



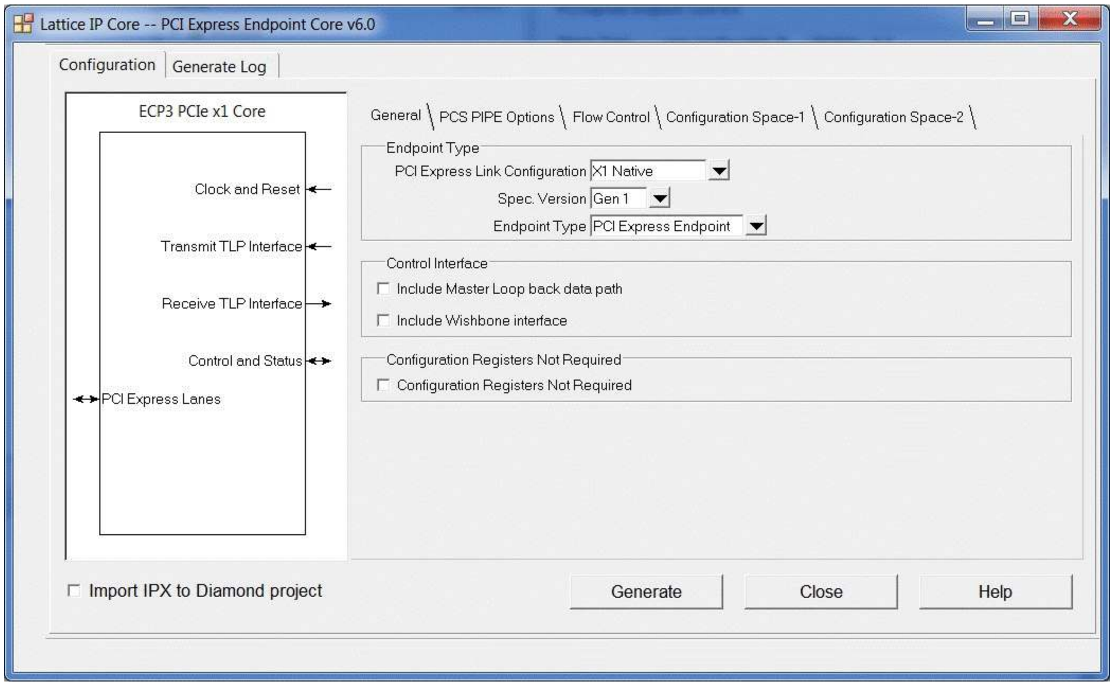
PCI Express Endpoint Core 6.0

Macro Type: User Configurable IP Version: 6.0
IP Name: PCI Express Endpoint Core
Project Path: C:/junk/EP_6.0_qa/ecp3
File Name: pcie_test
Module Output: Verilog
Device Family: LatticeECP3
Part Name: LFE3-95E-7FN1156C
Synthesis: SynplifyPro

Figure 4.1. IPexpress Tool Dialog Box

Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output, Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, click the **Customize** button in the IPexpress tool dialog box to display the PCI Express IP core Configuration GUI, as shown in Figure 4.2. From this dialog box, the user can select the IP parameter options specific to their application. Refer to Parameter Settings for more information on the PCI Express parameter settings. Additional information and known issues about the PCI Express IP core are provided in a ReadMe document that may be opened by clicking on the Help button in the Configuration GUI.



Endpoint Type
PCI Express Link Configuration X1 Native
Spec. Version Gen 1
Endpoint Type PCI Express Endpoint

Control Interface
☐ Include Master Loop back data path
☐ Include Wishbone interface

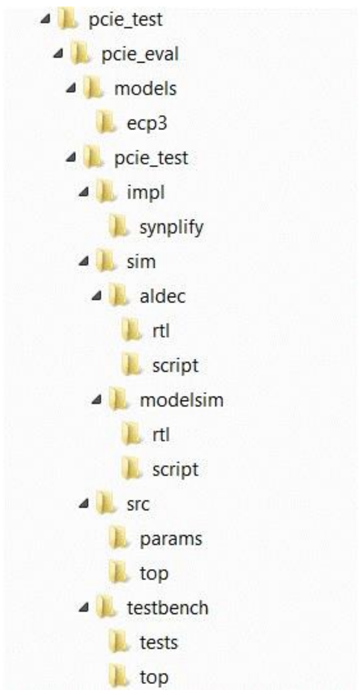
Configuration Registers Not Required
☐ Configuration Registers Not Required

☐ Import IPX to Diamond project

Figure 4.2. IPexpress Configuration GUI

4.2.2. IPexpress-Created Files and Top Level Directory Structure

When the user clicks the Generate button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified “Project Path” directory. The directory structure of the generated files is shown in Figure 4.3.



```
pcie_test
  pcie_eval
    models
      ecp3
    pcie_test
      impl
        synplify
      sim
        aldec
          rtl
          script
        modelsim
          rtl
          script
      src
        params
        top
      testbench
        tests
        top
```

Figure 4.3. LatticeECP3 PCI Express Core Directory Structure

The design flow for IP created with the IPexpress tool uses a post-synthesized module (NGO) for synthesis and a protected model for simulation. The post-synthesized module is customized and created during the IPexpress tool generation. The protected simulation model is not customized during the IPexpress tool process, and relies on parameters provided to customize behavior during simulation.

Table 4.1 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the files created are customized to the user’s module name specified in the IPexpress tool.

Table 4.1. File List

File	Sim	Synthesis	Description
<username>.v	Yes		This file provides the PCI Express core for simulation. This file provides a module which instantiates the PCI Express core and the PIPE interface.
<username>_core.v	Yes		This file provides the PCI Express core for simulation.
<username>_core_bb.v		Yes	This file provides the synthesis black box for the PCI express core.
<username>_phy_bb.v		Yes	This file provides the synthesis black box for the PCI express PIPE interface wrapper of SERDES/PCS.
<username>_beh.v	Yes		This file provides the front-end simulation library for the PCI Express core. This file is located in the pcie_eval/<user_name>/src/top directory.
pci_exp_params.v	Yes		This file provides the user options of the IP for the simulation model.
pci_exp_ddefines.v	Yes		This file provides parameters necessary for the simulation.
<username>_core/phy.ngo		Yes	This file provides the synthesized IP core used by the Diamond software. This file needs to be pointed to by the Build step by using the search path property.
<username>.lpc			This file contains the configuration options used to recreate or modify

			the core in the IPexpress tool.
username>.ipx			The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
pmi_*.ngo			These files contains the memories used by the IP core. These files need to be pointed to by the Build step by using the search path property.

Most of the files required to use the PCI Express IP core in a user's design reside in the root directory created by the IPexpress tool. This includes the synthesis black box, simulation model, and example preference file.

The `\pcie_eval` and subtending directories provide files supporting PCI Express IP core evaluation. The `\pcie_eval` directory contains files/folders with content that is constant for all configurations of the PCI Express IP core. The `\<username>` subfolder `(\pcie_test` in this example) contains files/folders with content specific to the `<username>` configuration.

The PCI Express ReadMe document is also provided in the `\pcie_eval` directory.

For example information and known issues on this core, see the Lattice PCI Express ReadMe document. This file is available when the core is installed in the Diamond software. The document provides information on creating an evaluation version of the core for use in Diamond and simulation.

The `\pcie_eval` directory is created by the IPexpress tool the first time the core is generated and updated each time the core is regenerated. A `\<username>` directory is created by the IPexpress tool each time the core is generated and regenerated each time the core with the same file name is regenerated. A separate `\<username>` directory is generated for cores with different names, e.g. `\<my_core_0>, \<my_core_1>`, etc.

The `\pcie_eval` directory provides an evaluation design which can be used to determine the size of the IP core and a design which can be pushed through the Diamond software including front-end and timing simulations. The models directory provides the library element for the PCS and PIPE interface for LatticeECP3.

The `\<username>` directory contains the sample design for the configuration specified by the customer. The `\<username>\impl` directory provides project files supporting Synplify synthesis flows. The sample design pulls the user ports out to external pins. This design and associated project files can be used to determine the size of the core and to push it through the mechanics of the Diamond software design flow.

The `\<username>\sim` directory provides project files supporting RTL simulation for both the Active- HDL and ModelSim simulators. The `\<username>\src` directory provides the top-level source code for the evaluation design. The \testbench directory provides a top-level testbench and test case files.

4.2.3. Instantiating the Core

The generated PCI Express IP core package includes <username>.v that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file that can be used as an instantiation template for the IP core is provided in `\<project_dir>\pcie_eval\<username>\src\top`. Users may also use this top- level reference as the starting template for the top-level for their complete design.

4.2.4. Running Functional Simulation

Simulation support for the PCI Express IP core is provided for Aldec and ModelSim simulators. The PCI Express core simulation model is generated from the IPexpress tool with the name <username>.v. This file calls <user- name>_core.v which contains the obfuscated simulation model(<username_beh.v). An obfuscated simulation model is Lattice's unique IP protection technique which scrambles the Verilog HDL while maintaining logical equivalence. VHDL users will use the same Verilog model for simulation.

When compiling the PCI Express IP core the following files must be compiled with the model.

- pci_exp_params.v
- pci_exp_ddefines.v

These files provide "define constants" that are necessary for the simulation model.

The ModelSim environment is located in `\<project_dir>\pcie_eval\<username>\sim\modelsim`. You can run the ModelSim simulation by performing the following steps:

1. Open ModelSim.
2. Under the File tab, select Change Directory and choose folder `\<project_dir>\pcie_eval\<username>\sim\modelsim`.
3. Under the Tools tab, select **Tcl > Execute Macro** and execute one of the ModelSim "do" scripts shown, depending on which version of ModelSim is used (ModelSim SE or the Lattice OEM version).

The Aldec Active-HDL environment is located in `\<project_dir>\pcie_eval\<username>\sim\aldec`. You can run the Aldec evaluation simulation by performing the following steps:

1. Open Active-HDL.
2. Under the Tools tab, select **Execute Macro**.
3. Browse to the directory `\<project_dir>\pcie_eval\<username>\sim\aldec` and execute the Active-HDL "do" script shown.

4.2.5. Synthesizing and Implementing the Core in a Top-Level Design

The PCI Express IP core itself is synthesized and provided in NGO format when the core is generated through the IPexpress tool. You can combine the core in your own top-level design by instantiating the core in your top level file as described in the Instantiating the Core section and then synthesizing the entire design with Synplify RTL Synthesis.

The top-level file `<username>_eval_top.v` provided in `\<project_dir>\pcie_eval\<user-name>\src\top` supports the ability to implement the PCI Express core in isolation. Push-button implementation of this top-level design with either Synplify RTL Synthesis is supported via the project files `<user- name>_eval.ldf` located in the directory `\<project_dir>\pcie_eval\<username>\impl\synplify`.

To use this project file in Diamond:

1. Choose **File > Open > Project**.
2. Browse to `\<project_dir>\pcie_eval\<username>\impl\synplify` in the Open Project dialog box.
3. Select and open <username>.ldf. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the Process tab in the left-hand GUI window.
5. Implement the complete design via the standard Diamond GUI flow.

To use this project file in ispLEVER:

1. Choose **File > Open Project**.
2. Browse to `\<project_dir>\pcie_eval\<username>\impl\synplify` in the Open Project dialog box.
3. Select and open <username>.syn. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the device top-level entry in the left-hand GUI window.
5. Implement the complete design via the standard ispLEVER GUI flow.

4.2.6. Hardware Evaluation

The PCI Express IP core supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of IP cores that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase on an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

4.2.7. Enabling Hardware Evaluation in Diamond

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

4.2.8. Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including: device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

4.2.9. Regenerating an IP Core in Diamond

To regenerate an IP core in Diamond:

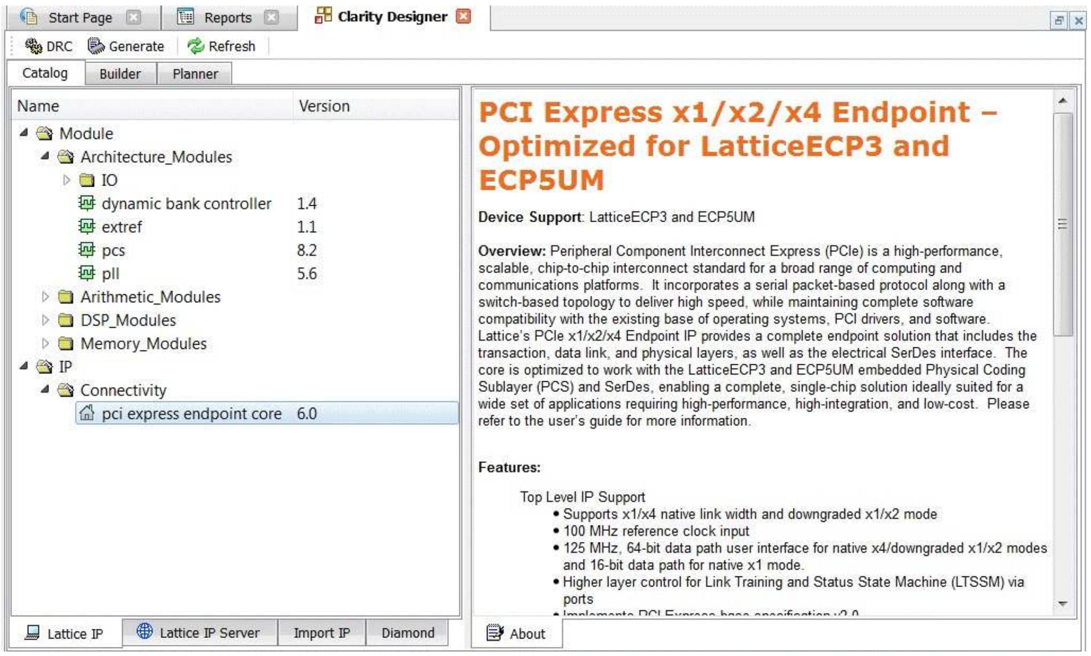
1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the Tar- get box.
4. To generate a new set of files in a new location, set the new location in the IPX Target File box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.
5. Click **Regenerate**. The dialog box of the module opens, showing the current option settings.
6. In the dialog box, choose the desired options. To get information about the options, click **Help**. Also, check the About tab in IPexpress for links to technical notes and user guides. IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.
7. To import the module into your project, if it's not already there, select Import **IPX to Diamond Project** (not available in standalone mode).
8. Click **Generate**.
9. Click the **Generate Log** tab to check for warnings and error messages.
10. Click **Close**.

The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above

4.3. Clarity Designer Flow for ECP5 and ECP5-5G Devices

4.3.1. Getting Started

The PCI Express IP core is available for download from the Lattice IP Server using the Diamond Clarity Designer tool for ECP5 and ECP5-5G devices. The IP files are automatically installed using InstallShield® technology in any customer-specified directory. After the IP core has been installed, the IP core is listed in the Available Modules tab of the Clarity Designer GUI as shown in Figure 4.4.



Name	Version
Module	
Architecture_Modules	
IO	
dynamic bank controller	1.4
extref	1.1
pcs	8.2
pll	5.6
Arithmetic_Modules	
DSP_Modules	
Memory_Modules	
IP	
Connectivity	
pci express endpoint core	6.0

PCI Express x1/x2/x4 Endpoint – Optimized for LatticeECP3 and ECP5UM

Device Support: LatticeECP3 and ECP5UM

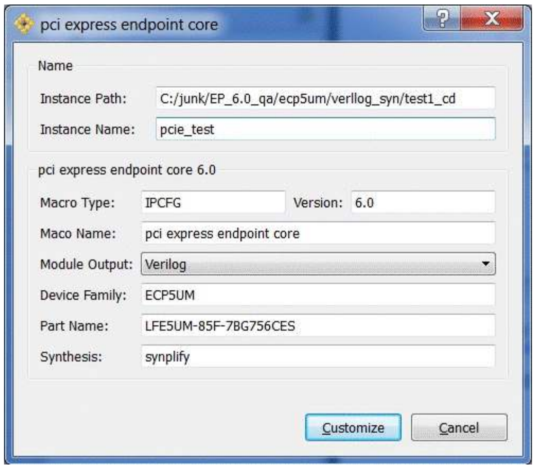
Overview: Peripheral Component Interconnect Express (PCIe) is a high-performance, scalable, chip-to-chip interconnect standard for a broad range of computing and communications platforms. It incorporates a serial packet-based protocol along with a switch-based topology to deliver high speed, while maintaining complete software compatibility with the existing base of operating systems, PCI drivers, and software. Lattice's PCIe x1/x2/x4 Endpoint IP provides a complete endpoint solution that includes the transaction, data link, and physical layers, as well as the electrical SerDes interface. The core is optimized to work with the LatticeECP3 and ECP5UM embedded Physical Coding Sublayer (PCS) and SerDes, enabling a complete, single-chip solution ideally suited for a wide set of applications requiring high-performance, high-integration, and low-cost. Please refer to the user's guide for more information.

Features:

Top Level IP Support
- Supports x1/x4 native link width and downgraded x1/x2 mode
- 100 MHz reference clock input
- 125 MHz, 64-bit data path user interface for native x4/downgraded x1/x2 modes and 16-bit data path for native x1 mode.
- Higher layer control for Link Training and Status State Machine (LTSSM) via ports

Figure 4.4. Clarity Designer GUI (pci express endpoint core)

In the Catalog tab, double clicking the **PCI Express endpoint** entry opens the PCI Express GUI dialog box shown in Figure 4.5.



Name

Instance Path:	C:/junk/EP_6.0_qa/ecp5um/verilog_syn/test1_cd
Instance Name:	pcie_test

pci express endpoint core 6.0

Macro Type:	IPCFG
Version:	6.0
Maco Name:	pci express endpoint core
Module Output:	Verilog
Device Family:	ECP5UM
Part Name:	LFE5UM-85F-7BG756CES
Synthesis:	synplify

Figure 4.5. PCI Express IP GUI Dialog Box

Note: Macro Type, Version and Macro Name are fixed for the specific IP core selected. Instance Path, Device Family and Part Name default to the specified parameters. SynplifyPro is the only synthesis tool presently supported for IP generation.

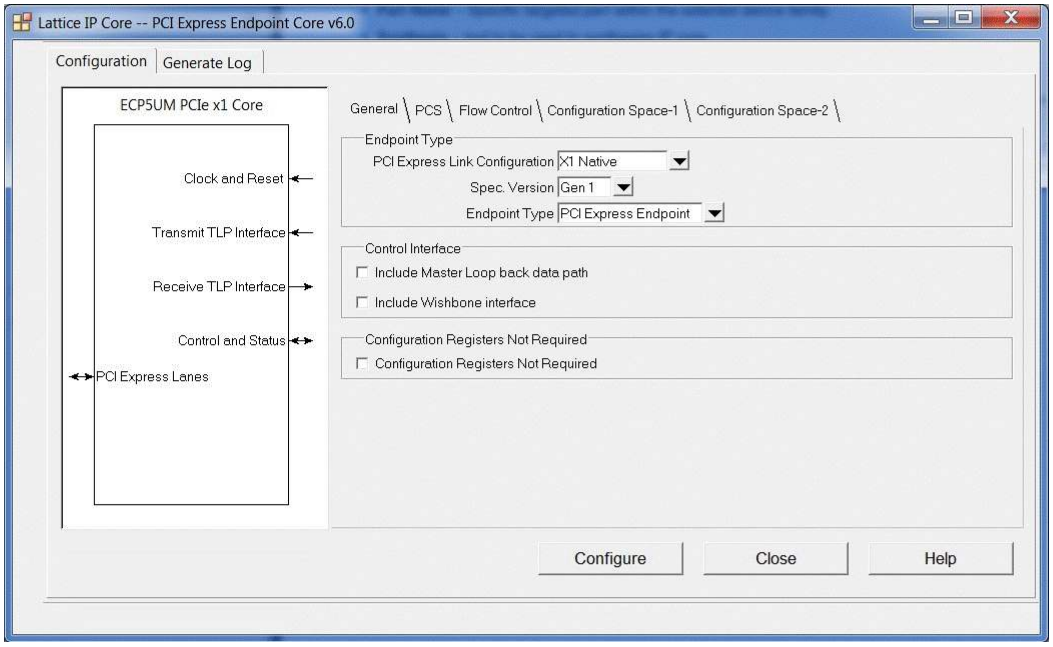
To generate a specific IP core configuration the user specifies:

- **Instance Path** – Path to the directory where the generated IP files will be located.
- **Instance Name** – “username” designation given to the generated IP core and corresponding folders and file.
- **Macro Type** – IPCFG (configurable IP) for PCI Express Endpoint
- **Version** – IP version number
- **Macro Name** – Name of IP Core
- **Module Output** – Verilog or VHDL.
- **Device Family** – Device family to which IP is to be targeted

- **Part Name** – Specific targeted part within the selected device family.
- **Synthesis** – tool to be used to synthesize IP core.

To configure the PCI Express IP:

1. Click the **Customize** button in the Clarity Designer tool dialog box to display the PCI Express IP core configuration GUI, as shown in Figure 4.6.
2. Select the IP parameter options specific to your application. Refer to Parameter Settings for more information on the PCI Express Endpoint IP core parameter settings. Additional information and known issues about the PCI Express IP core are provided in a ReadMe document that may be opened by clicking on the Help button in the Configuration GUI.



Endpoint Type
PCI Express Link Configuration: X1 Native
Spec. Version: Gen 1
Endpoint Type: PCI Express Endpoint

Control Interface
☐ Include Master Loop back data path
☐ Include Wishbone interface

Configuration Registers Not Required
☐ Configuration Registers Not Required

Figure 4.6. PCI Express Endpoint IP Core Configuration GUI

4.3.2. Configuring and Placing the IP Core

To configure and place the IP core:

1. After specifying the appropriate settings, click the **Configure** button and then **Close** button at the bottom of the IP core configuration GUI. At this point the data files specifying the configuration of this IP core instance are created in the user's project directory. An entry for this IP core instance is also included in the Planner tab of the Clarity Designer GUI, as shown in Figure 4.7.

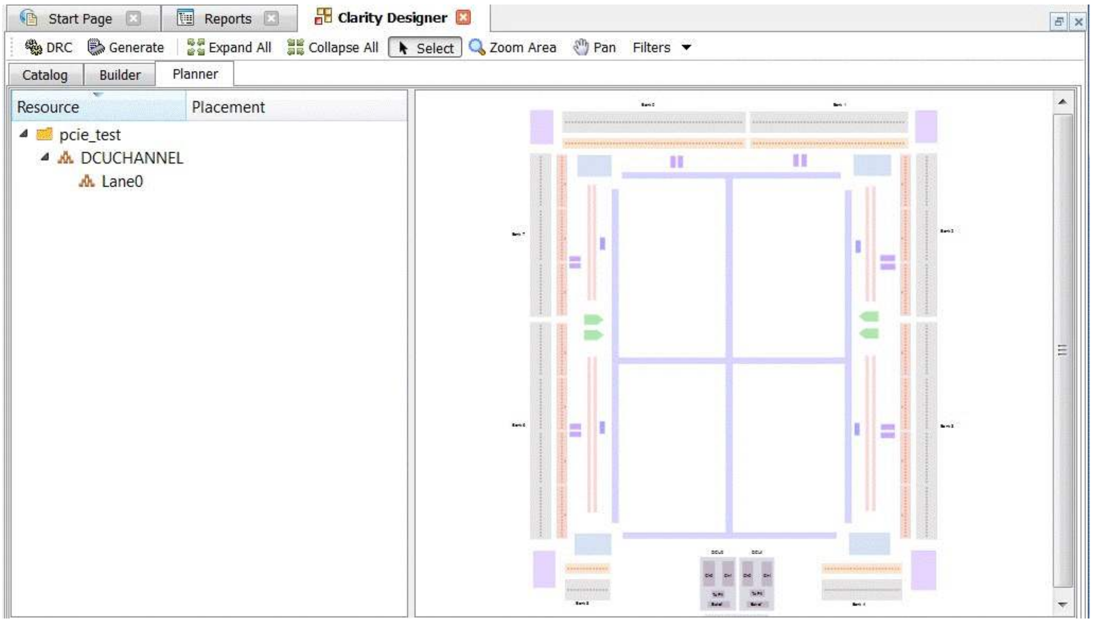


Figure 4.7. PCI Express Endpoint IP Core Clarity Designer GUI

2. The IP core instance may now be placed in the desired location in the ECP5 DCU. Drag the instance of associated IP SERDES lanes entry from the Planner tab to the Clarity Designer Placement tab. Once placed, the specific IP core placement is shown in the Configured Modules tab and highlighted in the Placement tab as shown in Figure 4.8.

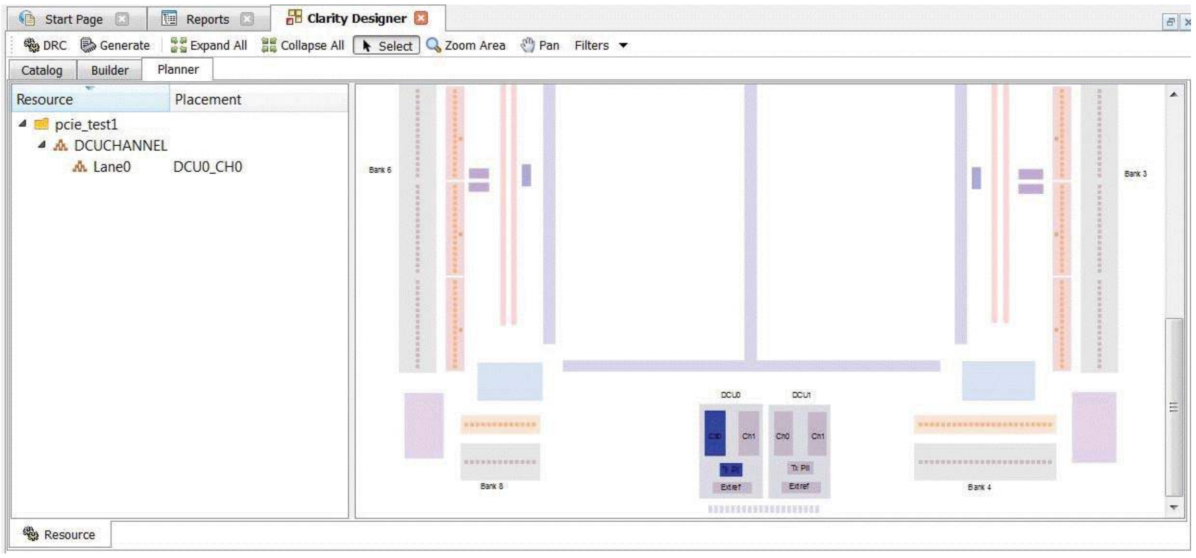
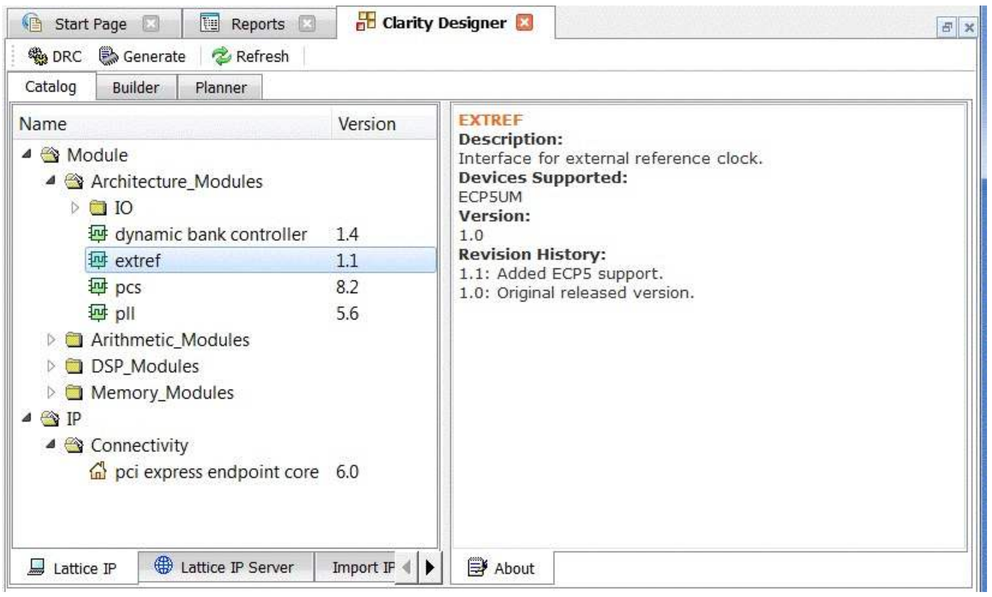


Figure 4.8. Clarity Designer Placed Module

Note that the PCI Express endpoint core may be configured to support 1, 2 or 4 SERDES channels. In x4 mode PCIe lanes 0, 1, 2 and 3 are mapped to both the DCUs, lanes 0,1 to DC0 and lanes 2,3 to DCU1. In x2 mode PCIe lanes 0, 1 can be mapped to either of DCUs. Similarly in X1 mode PCIe lanes 0 can be mapped to any channels of any DCU.

Note also that PCI express endpoint IP needs an EXTREF module to be connected for reference clocks which can be shared across multiple IPs. So it has to be generated outside the PCI express endpoint IP in the Clarity Designer tool and connected. Similar flow as the generation of PCI express endpoint core can be followed to generate extref module from the catalog tab as shown in the following Figure 4.9.

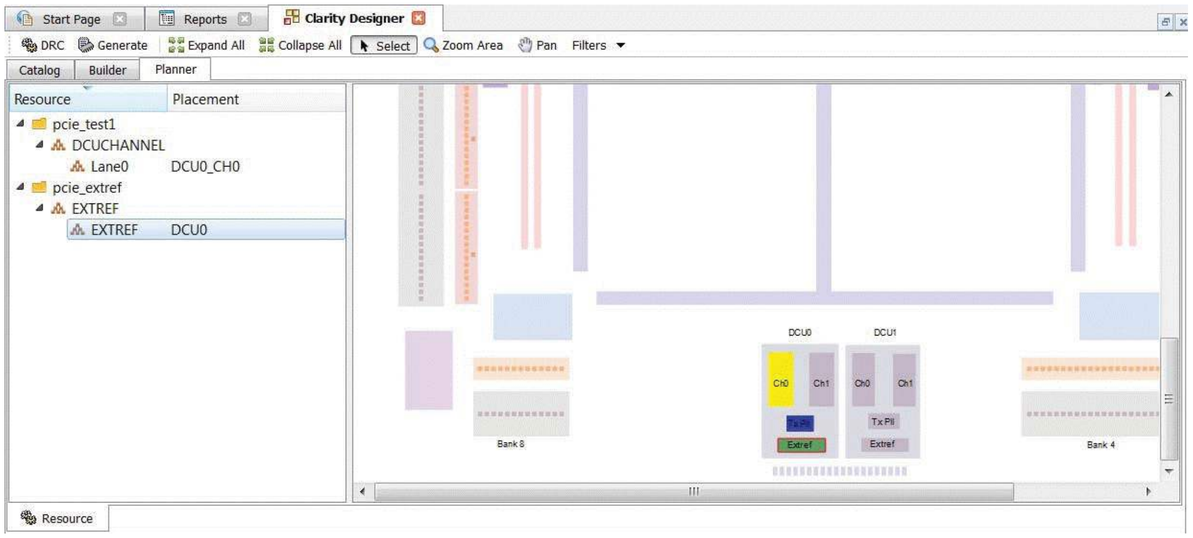


Name	Version
Module	
Architecture_Modules	
IO	
dynamic bank controller	1.4
extref	1.1
pcs	8.2
pll	5.6
Arithmetic_Modules	
DSP_Modules	
Memory_Modules	
IP	
Connectivity	
pci express endpoint core	6.0

EXTREF
Description:
Interface for external reference clock.
Devices Supported:
ECP5UM
Version:
1.0
Revision History:
1.1: Added ECP5 support.
1.0: Original released version.

Figure 4.9. Clarity Designer GUI (extref)

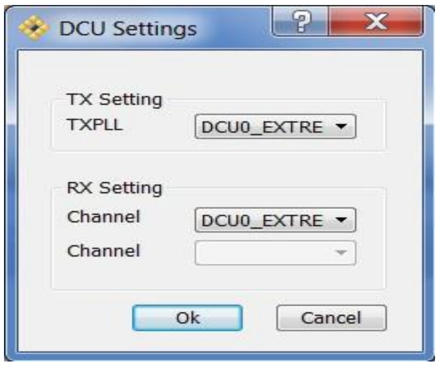
Both PCI express endpoint core and extref instances can be dragged and dropped from the Planner tab to the Clarity Designer Placement tab. Once placed, the specific IP core placement is shown in the Configured Modules tab and highlighted in the Placement tab as shown in Figure 4.10.



Resource	Placement
pcie_test1	
DCUCHANNEL	
Lane0	DCU0_CH0
pcie_extref	
EXTREF	
EXTREF	DCU0

Figure 4.10. Clarity Designer Placed Modules (pci express endpoint and extref)

3. After placing both PCI express endpoint core and extref instances, double-click the selected channel placement GUI. This opens a pop up DCU settings window as shown in Figure 4.11.


DCU Settings

TX Setting
TXPLL: DCU0_EXTRE

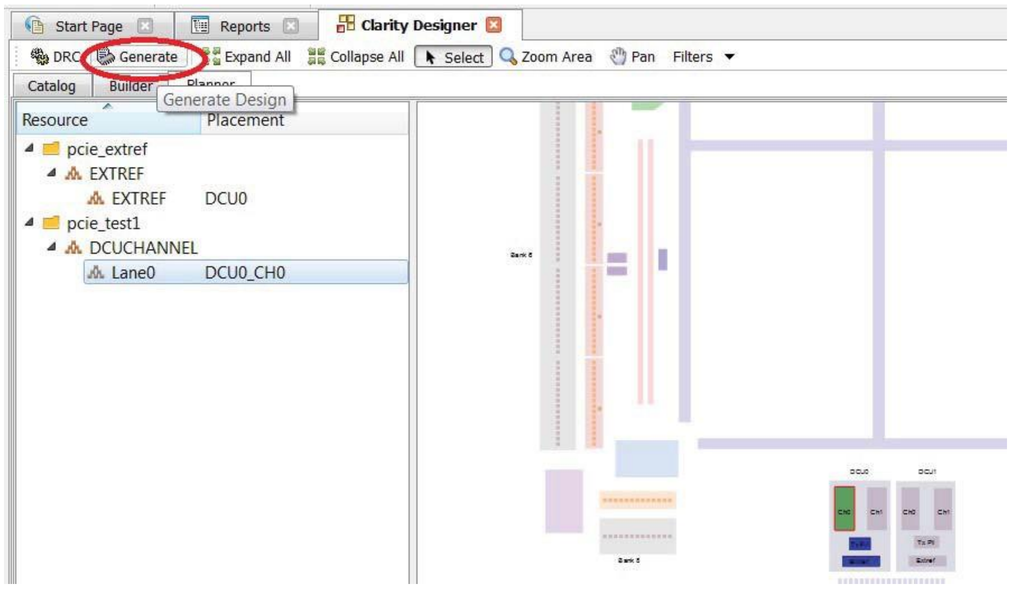
RX Setting
Channel: DCU0_EXTRE
Channel:

Figure 4.11. Clarity Designer DCU Settings

4. Select **DCU0_EXTREF (DCU1_EXTREF if DCU1)** as source for TXPLL and channel.
5. Click **OK**.

4.3.3. Generating the IP Core

After the IP core has been configured and placed, it may be generated by clicking on the Generate icon in the Clarity Designer GUI, as shown in Figure 4.12. When Generate is selected, all of the configured and placed IP cores shown in the Configured Modules tab and the associated supporting files are re-generated with corresponding placement information in the “Instance Path” directories.



Resource	Placement
pcie_extref	
EXTREF	
EXTREF	DCU0
pcie_test1	
DCUCHANNEL	
Lane0	DCU0_CH0

Figure 4.12. Generating the IP Core

4.3.4. Clarity Designer-Created Files and Directory Structure

The directory structure of the files created when the PCI express endpoint core is created is shown in Figure 4.13. Table 4.2 provides a list of key files and directories created by the Clarity Designer tool and how they are used. The Clarity Designer tool creates several files that are used throughout the design cycle. The names of many of the created files are customized to the user-specified instance name.

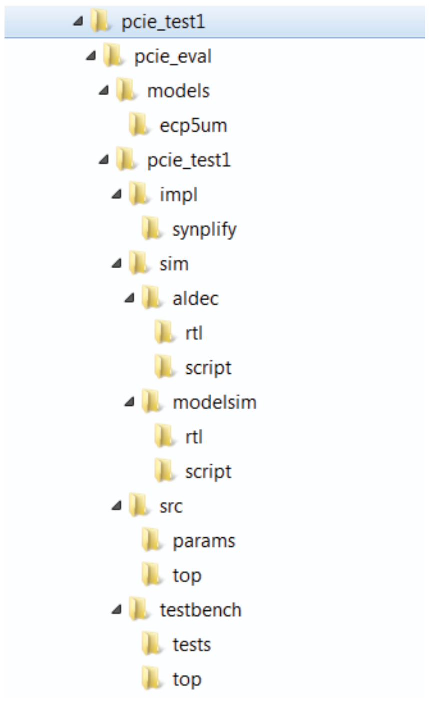


Figure 4.13. Directory Structure

Table 4.2. File List

File	Sim	Synthesis	Description
<username>.v	Yes		This file provides the PCI Express core for simulation. This file pro- vides a module which instantiates the PCI Express core and the PIPE interface
<username>_core_bb.v		Yes	This file provides the synthesis black box for the PCI express core.
<username>_beh.v	Yes		This file provides the front-end simulation library for the PCI Express core. This file is located in the pcie_eval/<user_name>/src/top directory.
pci_exp_params.v	Yes		This file provides the user options of the IP for the simulation model. This file is located in the pcie_eval/<user_name>/src/params directory.
pci_exp_ddefines.v	Yes		This file provides parameters necessary for the simulation. This file is located in the pcie_eval/<user_name>/src/params directory.
<username>_core.ngo		Yes	This file provides the synthesized IP core used by the Diamond software. This file needs to be pointed to by the Build step by using the search path property.
<username>.lpc			This file contains the configuration options used to recreate or modify the core in the Clarity Designer tool.
pmi_*.ngo			These files contains the memories used by the IP core. These files need to be pointed to by the Build step by using the search path property.

Most of the files required to use the PCI Express IP core in a user's design reside in the root directory created by the Clarity Designer tool. This includes the synthesis black box, simulation model, and example preference file. The `\pcie_eval` and subtending directories provide files supporting PCI Express IP core evaluation. The `\pcie_eval` directory contains files/folders with content that is constant for all configurations of the PCI Express IP core.

The `\<username>` subfolder (`\pcie_core0` in this example) contains files/folders with content specific to the <username> configuration.

The \pcie_eval directory is created by the Clarity Designer tool the first time the core is generated and updated each time the core is regenerated. A `\<username>` directory is created by the Clarity Designer tool each time the core is generated and regenerated each time the core with the same file name is regenerated. The `\pcie_eval` directory provides an evaluation design which can be used to determine the size of the IP core and a design which can be pushed through the Diamond software including front-end simulations. The models directory provides the library element for the PCS and PIPE interface.

The `\<username>` directory contains the sample design for the configuration specified by the customer. The `\<username>\impl` directory provides project files supporting Synplify synthesis flows. The sample design pulls the user ports out to external pins. This design and associated project files can be used to determine the size of the core and to push it through the mechanics of the Diamond software design flow. The `\<username>\sim` directory provides project files supporting RTL and timing simulation for both the Active- HDL and ModelSim simulators. The `\<username>\src` directory provides the top-level source code for the evaluation design. The \testbench directory provides a top-level testbench and test case files.

4.3.5. Instantiating the Core

The generated PCI express endpoint core package includes black-box (<username>_core_bb.v, <user- name>_phy.v) and instance (<username>.v) templates that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file that can be used as an instantiation template for the IP core is provided at `<project_dir>\pcie_eval\<username>\src\top\(<username>_eval_top.v`. Users may also use this top-level reference as the starting template for the top-level for their complete design.

4.3.6. Running Functional Simulation

Simulation support for the PCI Express IP core is provided for Aldec and ModelSim simulators. The PCI Express core simulation model is generated from the Clarity Designer tool with the name `<project_dir>\pcie_eval\<username>\src\top\<username>_beh.v` which contains the obfuscated simulation model. An obfuscated simulation model is Lattice's unique IP protection technique which scrambles the Verilog HDL while maintaining logical equivalence.

The ModelSim environment is located in the following directory:

`\<project_dir>\<user_name>\pcie_eval\<user_name>\sim\modelsim`.

To run the ModelSim simulation:

1. Open ModelSim.
2. Choose **File > Change Directory**.
3. Set the directory to `\<project_dir>\<user_name>\pcie_eval\<user_name>\sim\modelsim\rtl`.
4. Click **OK**.
5. Choose **Tools > TCL > Execute Macro**.
6. Select the simulation do file under the `modelsim\scripts` directory.

Note: When the simulation completes, a pop-up window appears with the prompt "Are you sure you want to finish?" Click **No** to analyze the results (clicking **Yes** closes ModelSim).

The Aldec Active-HDL environment is located in the following directory:

`\<project_dir>\<user_name>\pcie_eval\<user_name>\sim\aldec`.

To run the Aldec evaluation simulation:

1. Open Aldec.
2. Choose **Tools > Execute Macro**.
3. Set the directory to `\<project_dir>\<user_name>\pcie_eval\<user_name>\sim\aldec. rtl`.
4. Select **OK.**
5. Select simulation do file.

Note: When the simulation completes, a pop-up window appears stating "Simulation has finished. There are no more vectors to simulate."

4.3.7. Synthesizing and Implementing the Core in a Top-Level Design

The PCI Express endpoint IP core is synthesized and provided in NGO format when the core is generated through the Clarity Designer tool. You can combine the core in your own top-level design by instantiating the core in your top level file as described in section "Instantiating the Core" and then synthesizing the entire design with either Synplify. The top-level file `<username>_eval_top.v` provided in `\<project_dir>\pcie_eval\<username>\src\top` supports the ability to implement the PCI Express core in isolation. Push-button implementation of this top-level design with either Synplify is supported via the project files <username>_eval.ldf located in the `\<project_dir>\pcie_eval\<username>\impl\synplify` directory.

To use the .ldf project file in Diamond:

1. Choose File > Open > Project.
2. Browse to `\<project_dir>\<username>\pcie_eval\<username>\impl\synplify` in the Open Project dialog box.
3. Select and `open <username>_eval.ldf`. At this point, all of the files needed to support top-level synthesis and implementation are imported to the project.
4. Select the Process tab in the left-hand GUI window.
5. Implement the complete design via the standard Diamond GUI flow. At this point, all of the files needed to support top-level synthesis and implementation are imported to the project.

4.3.8. Hardware Evaluation

The PCI Express IP core supports Lattice's IP hardware evaluation capability, which makes it possible to create IP cores that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase on an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

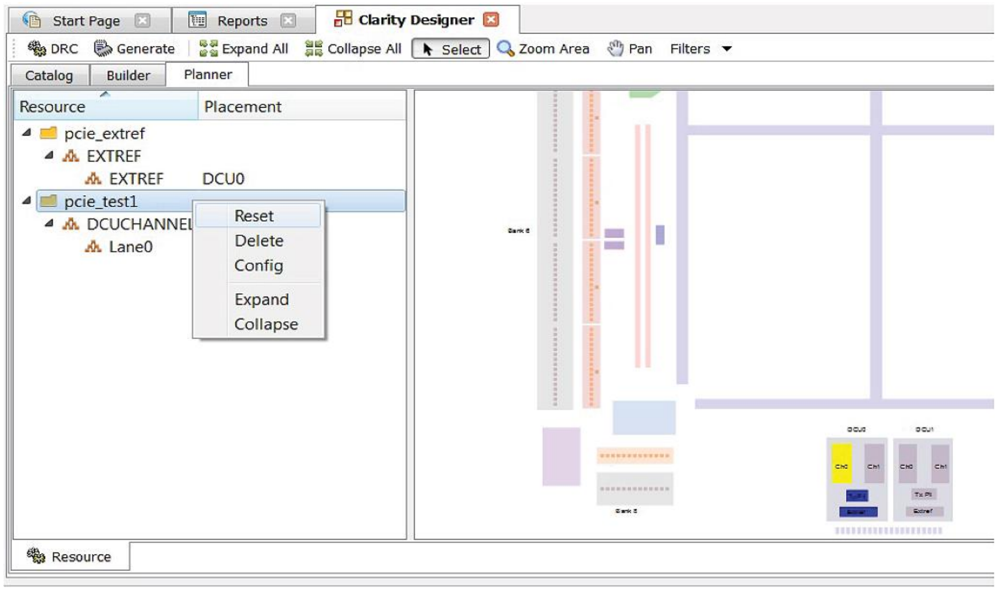
To Enable Hardware Evaluation in Diamond, choose Project > Active Strategy > Translate Design Settings. The hardware evaluation capability may be Enabled/Disabled in the Strategy dialog box. It is enabled by default.

4.3.9. Updating/Regenerating the IP Core

It is possible to remove, reconfigure and change the placement of an existing IP core instance using Clarity Designer tool. When the user right clicks on a generated IP core entry in the Planner tab, the selection options shown in Figure 4.14 are displayed. These options support the following capabilities:

- Reset – the present IP core placement is cleared and the IP core may be re-placed at any available site.
- Delete – the IP core instance is completely deleted from the project.
- Config – the IP core GUI is displayed and IP settings may be modified.
- Expand – Expands the view to show Placement information for IP core resource.
- Collapse – Collapses the view to with no placement information for IP resource.

After re-configuring or changing the placement of an IP core, the user must click Generate to implement the changes in the project design files.



Resource	Placement
pcie_extref	
EXTREF	
EXTREF	DCU0
pcie_test1	
DCUCHANNEL	
Lane0	

Figure 4.14. Reset, Delete, Config, Expand and Collapse Placement of the IP Core

5. Using the IP Core

This chapter provides supporting information on how to use the PCI Express IP core in complete designs. Topics discussed include IP simulation and verification, FPGA design implementation and board-level implementation.

5.1. Simulation and Verification

This section discusses strategies and alternative approaches for verifying the proper functionality of the PCI Express core through simulation.

5.1.1. Simulation Strategies

Included with the core from the Clarity tool is the evaluation test bench located in the <username> directory. The intent of the evaluation test bench is to show the core performing in simulation, as well as to provide timing simulations post place and route. Many communication cores work in a loopback format to simplify the data generation process and to meet the simple objectives of this evaluation test bench. A loopback format has been used in this case as well.

In a real system, however, PCI Express requires that an upstream port connect to a downstream port. In the simple-to-use, Lattice-supplied evaluation test bench, a few force commands are used to force an L0 state as a x4 link. Other force commands are also used to kick off the credit processing correctly.

Once a link is established via a loopback with the core, a few TLPs are sent through the link to show transmit and receive interface. This is the extent of the evaluation test bench.

Figure 5.1 illustrates the evaluation testbench process.

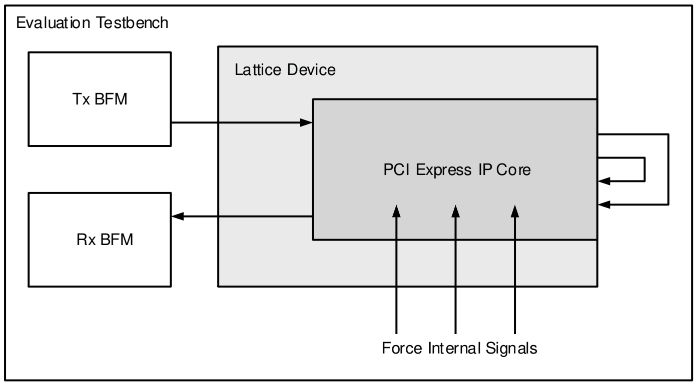


Figure 5.1. PCI Express x4 Core Evaluation Testbench Block Diagram

This testbench scheme works for its intent, but it is not easily extendible for the purposes of a Bus Functional Model (BFM) to emulate a real user system. Users can take the testbench provided and modify it to build in their own specific tests.

Sometimes the testbench is oriented differently than users anticipate. Users might wish to interface to the PCI Express core via the serial lanes. As an endpoint solution developer the verification should be performed at the endpoint of the system from the root complex device.

Refer to the Alternative Testbench Approach section for more information on setting up a testbench.

Users simulating a multi-lane core at the serial level should give consideration to lane ordering. Lane ordering is dependent on the layout of the chip on a board.

5.1.2. Alternative Testbench Approach

In order to create a testbench which meets the user's needs, the data must be sourced across the serial PCI Express lanes. The user must also have the ability to create the source traffic that will be pushed over the PCI Express link. This solution can be created by the user using the Lattice core.

Figure 5.2 shows a block diagram that illustrates a new testbench orientation which can be created by the user.

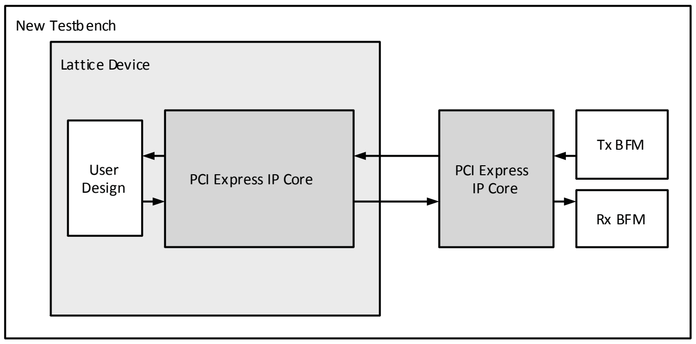


Figure 5.2. PCI Express x4 Core Testbench Using Two Cores

Use two PCI Express cores. The first PCI Express core is used in the design and the second PCI Express core is used as a driver. The user needs to use the no_pcie_train command to force the L0 state of the LTSSM in both cores.

When IP Core does not use the Wishbone bus, the bench must force no_pcie_train port on the IP to “1” to set LTSSM to L0 status.

When the Wishbone bus is implemented, there is no no_pcie_train port on the IP Core. Therefore, the bench must set the “LTSSM no training” register to force LTSSM to L0 status.

Whether or not the Wishbone bus is implemented, the bench must force LTSSM to L0 after both LTSSM state machines of transmitter and receiver are moved to Configuration status (4’d2).

As a result, the second core can then be used as a traffic separator. The second core is created to be the opposite of the design core. Thus an upstream port will talk with a downstream port and vice versa. The second core is used as a traffic generator. User-written BFMs can be created to source and sink PCI Express TLPs to exercise the design.

An issue associated with this test bench solution is that the run time tends to be long since the test bench will now include two PCS/SERDES cores. There is a large number of functions contained in both of the IP blocks which will slow down the simulation. Another issue is that the Lattice PCI Express solution is being used to verify the Lattice PCI Express solution. This risk is mitigated by the fact that Lattice is PCI-SIG compliant (see the Integrator's list at www.pci-sig.com) and a third party verification IP was used during the development process.

It should also be noted that this approach does not allow for PCI Express layer error insertion.

5.1.3. Third Party Verification IP

The ideal solution for system verification is to use a third party verification IP. These solutions are built specifically for the user’s needs and supply the BFMs and provide easy to use interfaces to create TLP traffic. Also, models are behavioral, gate level, or even RTL to increase the simulation speed.

Lattice has chosen the Synopsys PCI Express verification IP for development of the PCI Express core, as shown in Figure 5.3. There are other third party vendors for PCI Express including Denali® and Cadence®.

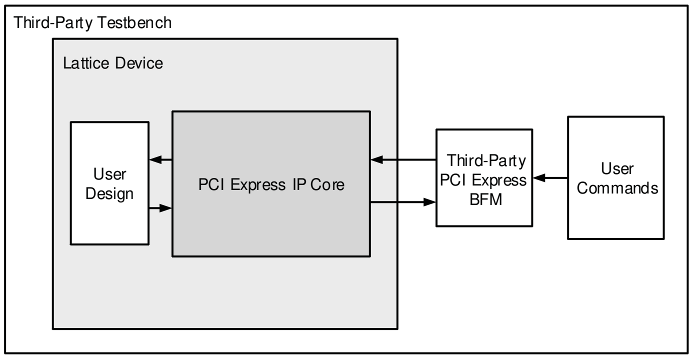


Figure 5.3. PCI Express x4 Core Testbench with Third-Party VIP

If desired, an independent Bus Functional Model can be modified to emulate a user's environment. This option is highly recommended.

5.2. FPGA Design Implementation for LatticeECP3 Devices

This section provides information on implementing the PCI Express IP core in a complete FPGA design. Topics covered include how to set up the IP core for various link width combinations, clocking schemes and physically locating the IP core within the FPGA.

5.2.1. Setting Up the Core

This section describes how to set up the PCI Express core for various link width combinations. The user must pro- vide a different PCS/SERDES autoconfig file based on the link width and the flipping of the lanes. The PCS/SERDES memory map is initially configured during bit stream loading using the autoconfig file generated with the IPexpress tool.

Note that transactions shown display data in hexadecimal format with bit 0 as the MSb.

5.2.2. Setting Up for Native x4 (No Flip)

This is the default condition that is created from the IPexpress tool. Simply use the autoconfig file to setup the channels. The flip_lanes port should be tied low.

5.2.3. Setting Up for Native x4 (Flipped)

No changes required. Simply use the pcs_pcie_8b_x4.txt file generated from the IPexpress tool.

5.2.4. Setting Up for Downgraded x1 (No Flip)

If the design will be using only a single channel and it is not flipped then Channels 1, 2, and 3 need to be powered down. Change the following lines from the pcs_pipe_x4.txt file.

```
CH0_MODE "GROP1"
CH1_MODE "GROUP1"
CH2_MODE "GROUP1"
CH3_MODE "GROUP1"
```

to

```
CH0_MODE "GROUP1"
CH1_MODE "DISABLE"
CH2_MODE "DISABLE"
```

```
CH3_MODE "DISABLE"
```

The flip_lanes port should be tied low.

5.2.5. Setting Up for Downgraded x1 (Flipped)

If the design will be using only a single channel and it is flipped then Channel 3 becomes the master channel and Channels 0, 1, and 2 to be powered down using the autoconfig file.

Change the following lines from the pcs_pcie_8b_x4.txt file.

```
CH0_MODE"GROUP1"
CH1_MODE"GROUP1"
CH2_MODE"GROUP1"
CH3_MODE"GROUP1"
```

to

```
CH0_MODE"DISABLE"
CH1_MODE"DISABLE"
CH2_MODE"DISABLE"
CH3_MODE"GROUP1"
```

The flip_lanes port should be tied high.

5.2.6. Setting Design Constraints

There are several design constraints that are required for the IP core. These constraints must be placed as preferences in the .lpf file. These preferences can be entered in the .lpf file through the Preference Editing View in Diamond or directly in the text based .lpf file.

Refer to .lpf file at directory `\<project_dir>\pcie_eval\<username>\impl\synplify` for design constraints required by the IP.

5.2.7. Clocking Scheme

A PCI Express link is typically provided with a 100 MHz reference clock from which the 2.5 Gbps data rate is achieved. The user interface for the PCI Express IP core is clocked using a 125 MHz clock (sys_clk_125).

Figure 5.4 and Figure 5.5 provide the internal clocking structures of the IP core in the LatticeECP3 and ECP5 family

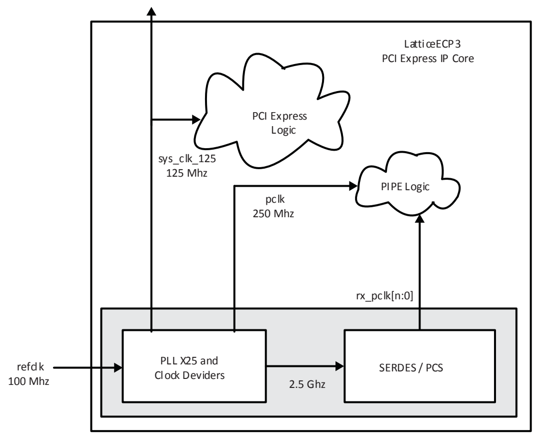


Figure 5.4. LatticeECP3 PCI Express Clocking Scheme

The LatticeECP3 clocking solution uses the 100 MHz differential refclk provided from the PCI Express link connected directly to the REFCLKP/N of the SERDES. The 100 Ω differential termination is included inside the SERDES so external resistors are not required on the board. It is recommended that both the sys_clk_125 and pclk clock nets are routed using primary clock routing.

Inside the SERDES, a PLL creates the 2.5 Gbps rate from which a transmit 250 MHz clock (pclk) and recovered clock(s) (ff_rx_fclk_[n:0]) are derived. The Lattice PCI Express core then performs a clock domain change to the sys_clk_125 125 MHz clock for the user interface.

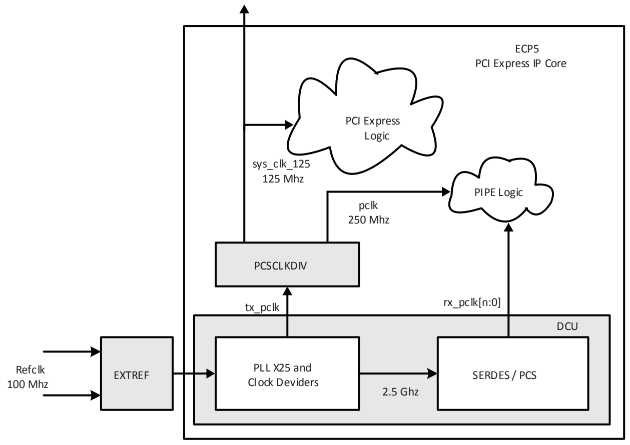


Figure 5.5. ECP5 PCI Express Clocking Scheme

The ECP5 clocking solution uses the 100 MHz differential refclk provided from the PCI Express link connected directly to the REFCLKP/N of the EXTREF component of the device. The 100 Ω differential termination is included in the device so external resistors are not required on the board. It is recommended that both the sys_clk_125 and pclk clock nets are routed using primary clock routing.

Inside the SERDES, a PLL creates the 2.5 Gbps rate from which a transmit 250 MHz clock (pclk) and recovered clock(s) (ff_rx_fclk_[n:0]) are derived. The Lattice PCI Express core then performs a clock domain change to the 125 MHz clock(sys_clk_125) for the user interface.

5.2.8. Locating the IP

The PCI Express core uses a mixture of hard and soft IP blocks to create the full design. This mixture of hard and soft IP requires the user to locate, or place, the core in a defined location on the device array. The hard blocks' fixed locations will drive the location of the IP.

Figure 5.6 provides a block diagram with placement positions of the PCS/SERDES quads in the LatticeECP3 devices.

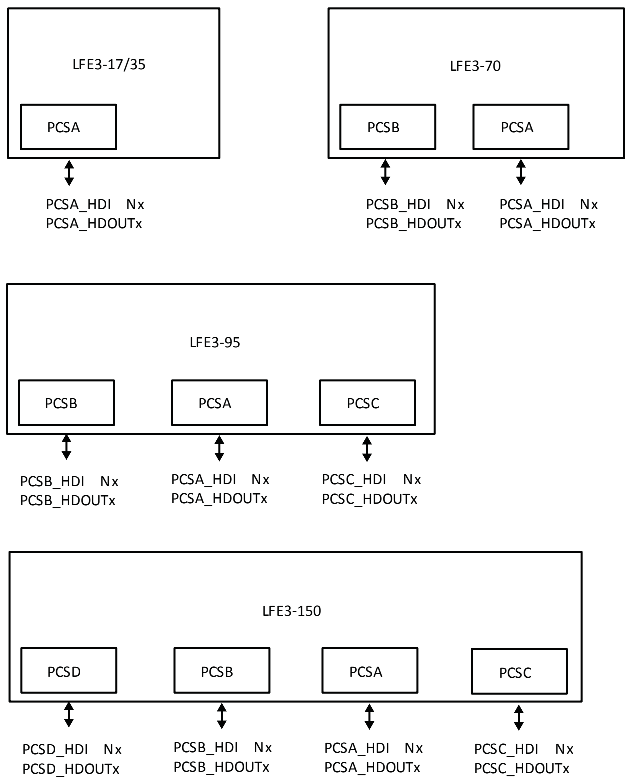


Figure 5.6. LatticeECP3 Device Arrays with PCS/SERDES

Figure 5.7 provides a block diagram with placement positions of the PCS/SERDES duals in the ECP5 devices.

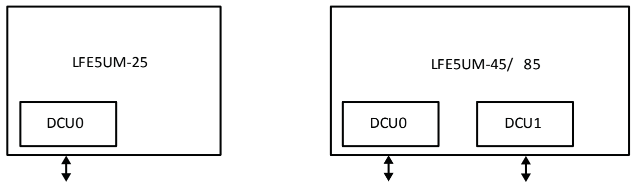


Figure 5.7. ECP5 Device Arrays with PCS/SERDES

5.3. Board-Level Implementation Information

This section provides circuit board-level requirements and constraints associated with using the PCI Express IP core.

5.3.1. PCI Express Power-Up

The PCI Express specification provides aggressive requirements for Power Up. As with all FPGA devices Power Up is a concern when working with tight specifications. The PCI Express specification provides the specification for the release of the fundamental reset (PERST#) in the connector specification. The PERST# release time (TPVPERL) of 100 ms is used for the PCI Express Card Electromechanical Specification for Add-in Cards.

From the point of power stable to at least 100 ms the PERST# must remain asserted. Different PCI Express systems will hold PERST# longer than 100 ms, but the minimum time is 100 ms. Shown below in Figure 5.8 is a best case timing diagram of the Lattice device with respect to PERST#.

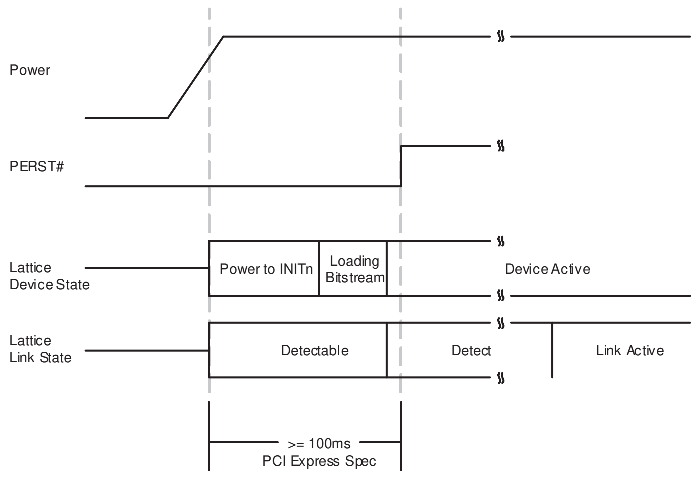


Figure 5.8. Best Case Timing Diagram, Lattice Device with Respect to PERST#

If the Lattice device has finished loading the bitstream prior to the PERST# release, then the PCI Express link will proceed through the remainder of the LTSSM as normal.

In some Lattice devices the device will not finish loading the bitstream until after the PERST# has been released. Figure 5.9 shows a worst case timing diagram of the Lattice device with respect to PERST#.

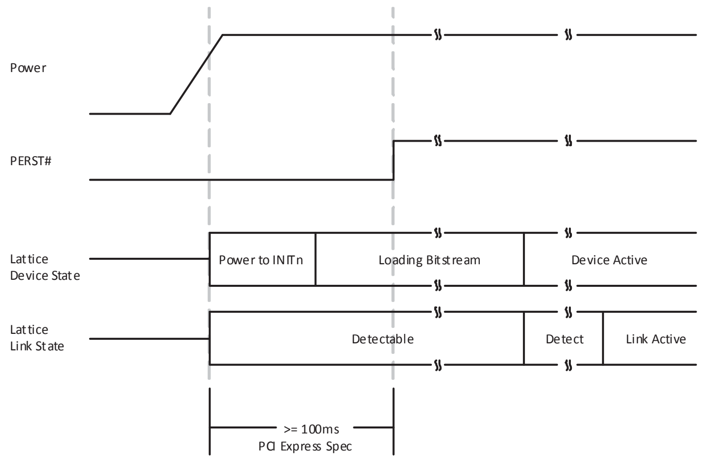


Figure 5.9. Worst Case Timing Diagram, Lattice Device with Respect to PERST#

If the Lattice device does not finish loading the bit stream until after the release of PERST#, then the link will still be established. The Lattice device turns on the 100 Ω differential resistor on the receiver data lines when power is applied. This 100 Ω differential resistance will allow the device to be detected by the link partner. This state is shown above as "Detectable". If the device is detected the link partner will proceed to the Polling state of the LTSSM. When the Lattice device goes through Detect and then enters the Polling state the link partner and Lattice device will now cycle through the remainder of the LTSSM.

In order to implement a power-up strategy using Lattice devices, Table 5.1 and Table 5.2 contain the relative numbers for the LatticeECP3 and ECP5 family.

Table 5.1. LatticeECP3 Power Up Timing Specifications

Specification	ECP3-17	ECP3-35	ECP3-70	ECP3-95	ECP3-150	Units
Power to INITn	23	23	23	23	23	ms

Worst-case Programming Time (SPI at 33 MHz)	136	249	682	682	1082	ms
Worst-case Programming Time (Parallel Flash with CPLD)*	17	31	85	85	135	ms

Note: 8-bit wide Flash and external CPLD interfacing to LatticeECP3 at 33 MHz SLAVE PARALLEL mode.

Table 5.2. ECP5 Power Up Timing Specifications

Specification	ECP5-25	ECP5-45	P5UMECP5-85	Units
Power to INITn	33	33	33	ms
Worst-case Programming Time (SPI at 33 MHz)	164	295	556	ms
Worst-case Programming Time (Parallel Flash with CPLD)1	20	36	69	ms

Note: 8-bit wide Flash and external CPLD interfacing to LatticeECP3 at 33 MHz SLAVE PARALLEL mode.

These warnings inform the user that a SLICE is programmed in DPRAM mode which allows a constant write to the RAM. This is an expected implementation of the RAM which is used in the PCI Express design.

To reduce the bit stream loading time of the Lattice device a parallel Flash device and CPLD device can be used. The use of parallel Flash devices and Lattice devices is documented in AN8077, Parallel Flash Programming and FPGA Configuration.

During initialization the PROGRAM and GSR inputs to the FPGA can be used to hold off bit stream programming. These should not be connected to PERST# as this will delay the bit stream programming of the Lattice device.

5.4. Board Layout Concerns for Add-in Cards

The PCI Express Add-in card connector edge finger is physically designed for a particular orientation of lanes. The device package pinout also has a defined orientation of pins for the SERDES channels. The board lay- out will connect the PCI Express edge fingers to the SERDES channels. For multi-lane implementations there might be a layout concern in making this connection. On some packages lane 0 of the edge fingers will align with lane 0 of the SERDES and likewise for channels 1, 2 and 3. However, in other packages lane 0 of the edge fingers will need to cross lanes 1, 2 and 3 to connect to lane 0 of the SERDES. It will not be possible to follow best practice layout rules and cross SERDES lanes in the physical board design. Figure 5.10 provides an example of the board layout concern.

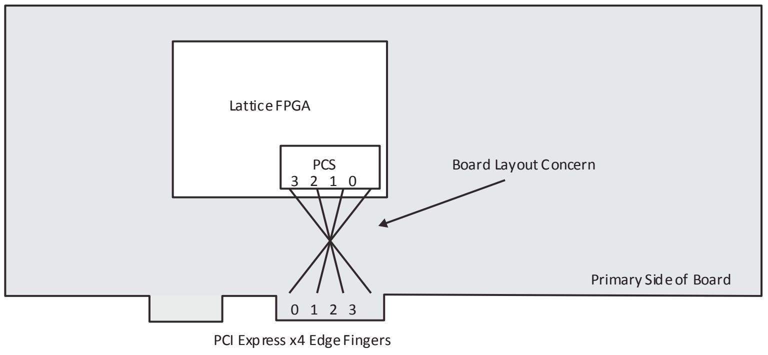


Figure 5.10. Example of Board Layout Concern with x4 Link

To accommodate this layout dilemma, the Lattice PCI Express solution provides an option to reverse the order of the SERDES lanes to the LTSSM block of the PCI Express core. This allows the board layout to connect edge finger lane 0 to SERDES lane 3, edge finger lane 1 to SERDES lane 2, edge finger lane 2 to SERDES lane 1, and edge finger lane 3 to SERDES lane 0. The PCI Express core will then perform a reverse order connection so the PCI Express edge finger lane 0 always connects to the logical LTSSM lane 0. This lane connection feature is controlled using the flip_lanes port. When high, this port will connect the SERDES channels to the PCI Express core in the reversed orientation. The user must be aware when routing the high speed serial lines that this change has taken place. PCI Express lane 0 will need to connect to SERDES channel 3, etc. Figure 5.11 provides a diagram of a normal and a reversed IP core implementation.

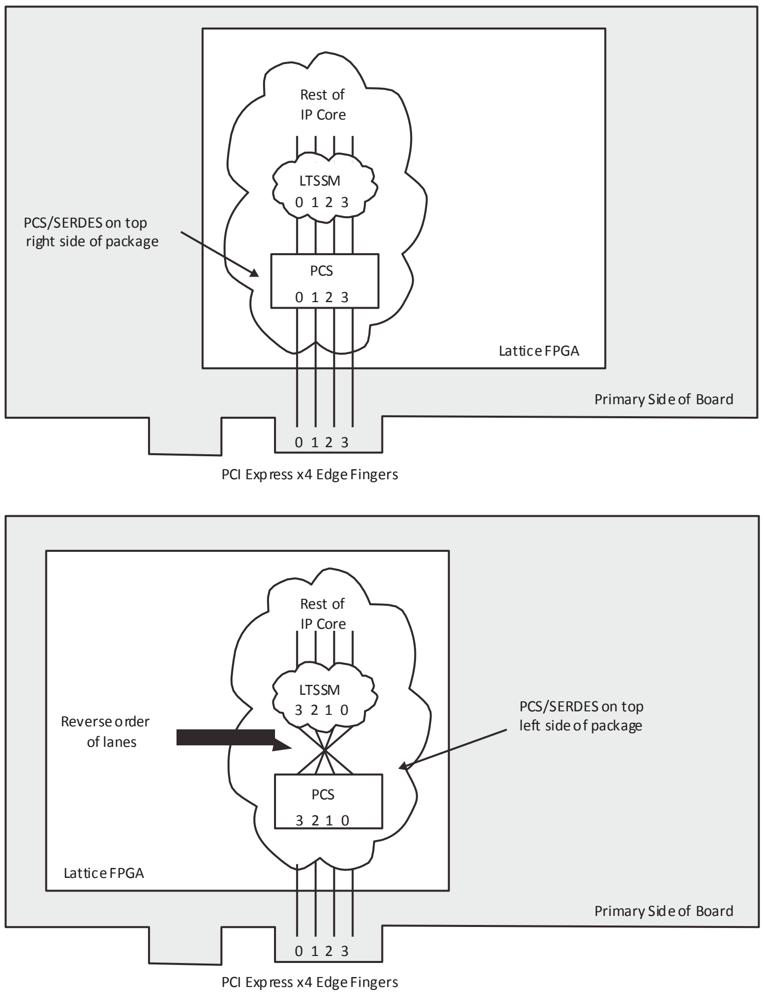


Figure 5.11. Implementation of x4 IP Core to Edge Fingers

As shown in Figure 5.11, this board layout condition will exist on SERDES that are located on the top left side of the package. When using a SERDES quad located on the top left side of the package the user should reverse the order of the lanes inside the IP core.

Figure 5.12 provides a diagram of a x1 IP core to illustrate the recommended solution in the board layout. Provides a diagram of a x1 IP core to illustrate the recommended solution in the board layout.

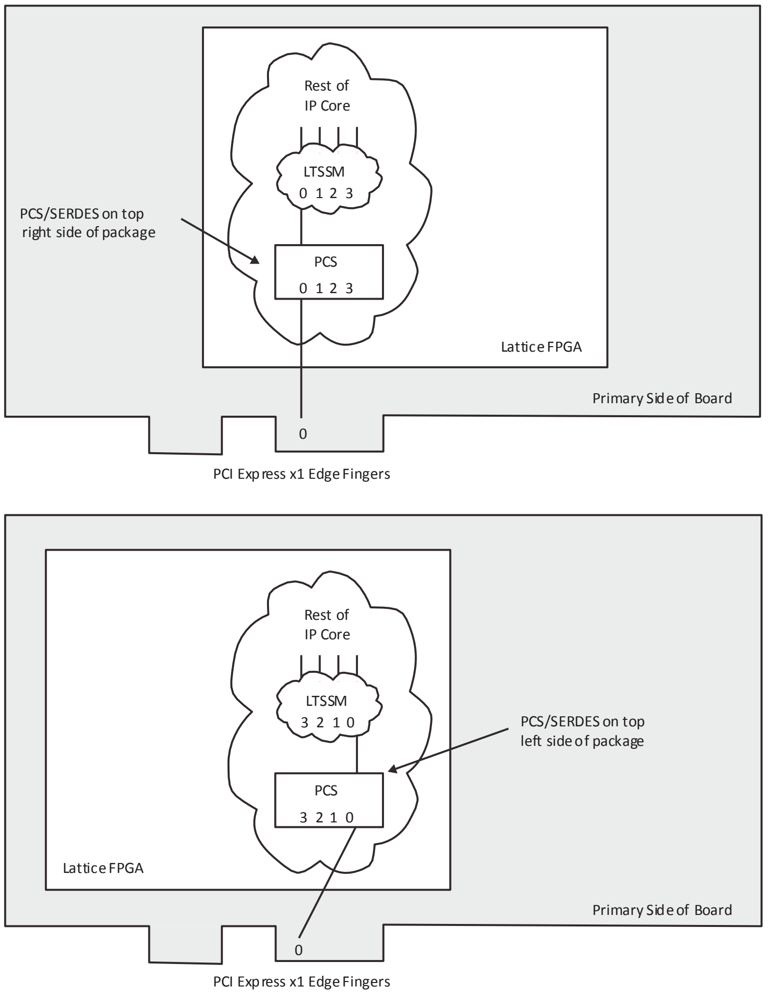


Figure 5.12. Implementation of x1 IP Core to Edge Fingers

5.5. Adapter Card Concerns

A PCI Express adapter card allows a multi-lane PCI Express endpoint to be plugged into a PCI Express slot that supports less lanes. For example, a x16 endpoint add in card could use an adapter card to plug into a x1 slot. Adapter cards simply plug onto the edge fingers and only supply connections to those on the edge fingers of the adapter card. Figure 5.13 provides the stack up of an endpoint add in card with an adapter card.

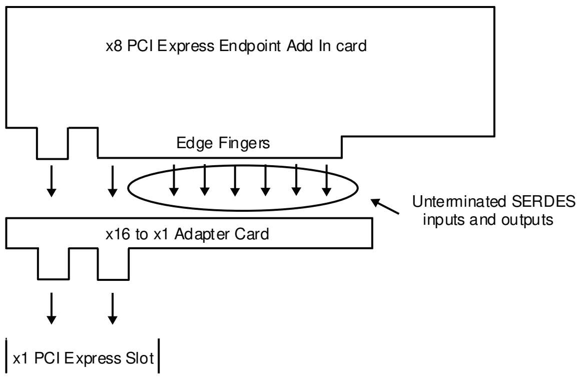


Figure 5.13. PCI Express Endpoint Add In Card

An adapter card simply connects edge fingers to edge fingers. Any of the lanes that are not used by the adapter card are sitting in the adapter card slot. They are unterminated. In Lattice devices, all SERDES channels that are powered up need to be terminated. When using an adapter card the unused channels must be powered down. This can be accomplished by simply editing the autoconfig file for the PCS and not powering up the unused channels. This will provide a bitstream that is suitable for adapter cards.

5.5.1. LatticeECP3 and ECP5 IP Simulation

The ECP5 PCI Express simulation uses the PIPE module. This simulation model is found in the `<user_name>/pcie_eval/models/<ecp5um/ecp3>/<user_name>_phy.v` file. The same directory contains few other files required for `pcs_pipe_top` module.

Refer to `pcie_eval/<username>/sim/<aldec/modelsim>/script/eval_beh_rtl<_se>`.

5.6. Simulation Behavior

When setting the SIMULATE variable for the simulation model of the PCI Express core several of the LTSSM counters are reduced. Table 5.3 provides the new values for each of the LTSSM counters when the SIMULATE variable is defined.

Table 5.3. LTSSM Counters

Counter	Normal Value	SIMULATE Value	Description
CNT_1MS	1 ms	800 ns	Electrical Order set received to Electrical Idle condition detected by Loop- back Slave
CNT_1024T1	1024 TS1	48 TS1	Number of TS1s transmitted in Polling.Active
CNT_2MS	2 ms	1200 ns	Configuration.Idle (CFG_IDLE)
CNT_12MS	12 ms	800 ns	Detect.Quiet (DET_QUIET)
CNT_24MS	24 ms	1600 ns	Polling.Active (POL_ACTIVE), Configuration.Linkwidth. Start (CFG_LINK_WIDTH_ST), Recovery.RcvrLock (RCVRY_RCVRLK)
CNT_48MS	48 ms	3200 ns	Polling.Configuration (POL_CONFIG), Recovery.RcvrCfg (RCVRY_RCVRCFG)

5.7. Troubleshooting

Table 5.4 provides some troubleshooting tips for the user when the core does not work as expected.

Table 5.4. Troubleshooting

Symptom	Possible Reason	Troubleshooting
LTSSM does not transition to L0 state	The PCI Express slot does not support the advertised link width.	Some PC systems do not support all possible link width configurations in their x16 slots. Try using the slot as a x1 and working up to the x4 link width.
Board is not recognized by the PC	Driver not installed or did not bind to the board.	Check to make sure the driver is installed and the DeviceID and VendorID match the drivers IDs defined in the .inf file.
Software application locks up	Endpoint is stalled and can- not send TLPs.	Check to make sure the amount of credits requested is correct. If the endpoint cannot complete a transaction started by the application soft- ware, the software will "hang" waiting on the endpoint.
PC crashes	Endpoint might have violated the available credits of the root complex.	A system crash usually implies a hardware failure. If the endpoint violates the number of credits available, the root complex can throw an exception which can crash the machine.
CNT_24MS	Endpoint might have created a NAK or forced a retrain.	Certain motherboards are forgiving of a NAK or LTSSM retrain while others are not. A retrain can be identified by monitoring the phy_ltssm_state vector from the PCI Express core to see if the link falls

6. Core Verification

The functionality of the Lattice PCI Express Endpoint IP core has been verified via simulation and hardware testing in a variety of environments, including:

- Simulation environment verifying proper PCI Express endpoint functionality when testing with a Synopsys DesignWare behavioral model in root complex mode.
- PCI-SIG certification via hardware validation of the IP implemented on LatticeECP3 FPGA evaluation boards. Specific testing has included:
    - Verifying proper protocol functionality (transaction layer, data link layer and DUT response to certain error conditions) when testing with the Agilent E2969A Protocol Test Card (PTC) and PTC test suite.
      Note: PTC was used for both in-house testing and testing at PCI-SIG workshops.
    - Verifying proper protocol functionality with the PCI-SIG configuration test suite.
    - Verifying electrical compliance.
      Note: Electrical compliance testing has been verified at PCI-SIG and also in-house by the Lattice PDE group.
- Interop testing with multiple machines at PCI-SIG workshops and in-house.
- Using the Agilent E2960A PCI Express Protocol Tester and Analyzer for analyzing and debugging PCI Express bus protocol. The Tester is used for sending and responding to PCI Express traffic from the DUT.

6.1. Core Compliance

A high-level description of the PCI-SIG Compliance Workshop Program and summary of the compliance test results for our PCI Express Endpoint IP core for LatticeECP3 device is provided in TN1166, PCI Express SIG Compliance Overview for Lattice Semiconductor FPGAs (August 2007). As described in TN1166, the Lattice PCI Express IP core successfully passed PCI-SIG electrical, Configuration Verifier (CV) and link and transaction layer protocol testing. The PCI Express IP core also passed the 80% interoperability testing program specified by PCI- SIG. In accordance with successfully completing PCI-SIG compliance and interoperability testing, the Lattice PCI Express Endpoint Controller IP cores are currently included on the PCI-SIG Integrators List.

Support Resources

6.2. References

For more information, refer to the following documents:

6.2.1. LatticeECP3

- DS1021, LatticeECP3 EA Family Data Sheet
- TN1176, LatticeECP3 SERDES/PCS Usage Guide

6.2.2. ECP5 and ECP5-5G

- DS1044, ECP5 and ECP5-5G Family Data Sheet
- TN1261, ECP5 and ECP5-5G SERDES/PCS Usage Guide

6.3. Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Appendix A. Resource Utilization of 2.5G IP core

This appendix provides resource utilization information for Lattice FPGAs using the PCI Express IP core.

The IPexpress (for LatticeECP3 devices) and the Clarity Designer (for ECP5 devices) tool is the Lattice IP configuration utility, and is included as a standard feature of the Diamond design tools. Details regarding the usage of the IPexpress and Clarity tool can be found in Diamond help system. For more information on the Diamond design tools, visit the Lattice website at: www.latticesemi.com//Products/DesignSoftware.

LatticeECP3 Utilization (Native x1)

Table A.1 lists the resource utilization for the PCI Express x1 Endpoint core implemented in a LatticeECP3 FPGA.

Table A.1. Resource Utilization*

IPexpress Configuration1	Slices	LUTs	Registers	sysMEM EBRs
Native x1	4033	6040	4027	4

***Note**: Performance and utilization data are generated targeting an LFE3-95E-7FN1156CES using Lattice Diamond 3.3 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x1 Endpoint IP core targeting LatticeECP3 devices is PCI-EXP1-E3-U3.

LatticeECP3 Utilization (Native x4)

Table A.2 lists the resource utilization for the PCI Express x4 Endpoint core implemented in a LatticeECP3 FPGA.

Table A.2. Resource Utilization*

IPexpress Configuration	Slices	LUTs	Registers	sysMEM EBRs
Native x4	8799	12169	9796	11

***Note**: Performance and utilization data are generated targeting an LFE3-95E-7FN1156CES using Lattice Diamond 3.3 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family. When the x4 core downgrades to x1 mode, utilization and performance results for x1 are identical to x4 mode.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x4 Endpoint IP core targeting LatticeECP3 devices is PCI-EXP4-E3-U3.

ECP5 Utilization (Native x1)

Table A.3 shows the resource utilization for the PCI Express x1 Endpoint core implemented in a ECP5 FPGA.

Table A.3. Resource Utilization*

Clarity Configuration	Slices	LUTs	Registers	sysMEM EBRs
Native x1	4270	6207	4188	4

***Note:** Performance and utilization data are generated targeting LFE5UM-85E-7MG756C using Lattice Diamond 3.0 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the ECP5 family.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x1 Endpoint IP core targeting ECP5 devices is PCI-EXP1-E5-U or PCI-EXP1-E5-UT.

ECP5 Utilization (Native x4)

Table A.4 shows the resource utilization for the PCI Express x4 Endpoint core implemented in a ECP5 FPGA.

Table A.4. Resource Utilization*

Clarity Configuration	Slices	LUTs	Registers	sysMEM EBRs
Native x4	9384	13906	9763	11

***Note:** Performance and utilization data are generated targeting LFE5UM-85E-7MG756C using Lattice Diamond 3.0 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the ECP5 family.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x4 Endpoint IP core targeting ECP5 devices isPCI-EXP4-E5-U or PCI-EXP4-E5-UT.

ECP5 Utilization (Downgraded x2)

Table A.5 shows the resource utilization for the PCI Express x2 Endpoint core implemented in a ECP5 FPGA.

Table A.5. Resource Utilization*

Clarity Configuration	Slices	LUTs	Registers	sysMEM EBRs
x4 Downgraded x2	8645	12911	8999	11

***Note:** Performance and utilization data are generated targeting LFE5UM-85E-7MG756C using Lattice Diamond 3.0 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the ECP5 family.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x2 Endpoint IP core targeting ECP5 devices is PCI-EXP4-E5-U or PCI-EXP4-E5-UT.

Appendix B. Resource Utilization of PCI Express 5G IP Core

This appendix provides resource utilization information for Lattice FPGAs using the PCI Express 5G IP core.

The Clarity Designer (for ECP5-5G devices) tool is the Lattice IP configuration utility, and is included as a standard feature of the Diamond design tools. Details regarding the usage of the IPexpress and Clarity tool can be found in Diamond help system. For more information on the Diamond design tools, visit the Lattice website at: www.latticesemi.com//Products/DesignSoftware.

ECP5-5G Utilization (Downgraded x1)

Table B.1 shows the resource utilization for the PCI Express x1 5G Endpoint core implemented in a ECP5-5G FPGA.

Table B.1. Resource Utilization*

Clarity Configuration	Slices	LUTs	Registers	sysMEM EBRs
x2 Downgraded x1	9307	13378	9354	7

***Note:** Performance and utilization data are generated targeting LFE5UM5G-85F-8BG756C using Lattice Diamond 3.8 software. Performance might vary when using a different software version or targeting a different device density or speed grade within the ECP5-5G family.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x1 5G Endpoint IP core targeting ECP5-5G devices is PCI-EXP2-E5G-U or PCI-EXP2-E5G-UT.

ECP5-5G Utilization (Native x2)

Table B.2 shows the resource utilization for the PCI Express x2 5G Endpoint core implemented in a ECP5-5G FPGA.

Table B.2. Resource Utilization*

Clarity Configuration1	Slices	LUTs	Registers	sysMEM EBRs
Native x2	10486	14909	10928	7

***Note:** Performance and utilization data are generated targeting LFE5UM5G-85F-8BG756C using Lattice Diamond 3.8 software. Performance might vary when using a different software version or targeting a different device density within the ECP5-5G family.

Ordering Part Number

The Ordering Part Number (OPN) for the PCI Express x2 5G Endpoint IP core targeting ECP5-5G devices is PCI-EXP2-E5G-U or PCI-EXP2-E5G-UT.

Revision History

Date	Document Version	IP Core Version	Package Name	Change Summary
October 2016	1.7	Beta	PCI Express 5G Endpoint	Corrected IP Core Version in Revision History.
	1.6	Beta	PCI Express 5G Endpoint	Added support for ECP5-5G.
		6.4	PCI Express Endpoint	
May 2016	1.5	6.3		Updated Lattice Technical Support section
				Added additional fabric pipeline registers to EBR output paths.
		6.2		Added mask logic to ECP5 RxValid signal from pipe wrapper.
		6.1		Added SoftLoL logic to ECP5 PIPE wrapper.
April 2015	1.4	6.0		Added LSE support for ECP5 devices.
November 2014	1.3	6.0		Added support for both LatticeECP3 and ECP5 in same package (IP core version 6.0).
April 2014	01.2	6.0_asr		Updated device name to ECP5.
January 2014	01.1	6.0_sbp		Added support for Clarity Designer flow.
September 2013	01.0	6.0ea		Initial EAP release.