



Lattice**CORE**

## PCI Express x1/x2/x4 Root Complex Lite IP Core User Guide

---

<b>Chapter 1. Introduction .....</b>	<b>4</b>
Quick Facts .....	4
Features .....	5
PHY Layer.....	5
Data Link Layer .....	5
Transaction Layer .....	5
Top Level IP Support .....	6
<b>Chapter 2. Functional Description .....</b>	<b>7</b>
Overview .....	7
Interface Description .....	17
Transmit TLP Interface.....	17
Transmit TLP Interface Waveforms for x1 .....	23
Receive TLP Interface.....	24
Message Decode Interface .....	26
Interrupt Signaling Messages.....	26
Error Signaling Messages .....	27
Using the Transmit and Receive Interfaces .....	28
As a Receiver.....	28
As a Transmitter.....	29
<b>Chapter 3. Parameter Settings .....</b>	<b>30</b>
General Tab .....	31
PCI Express Link Configuration .....	31
Include Master Loopback Data Path .....	31
Maximum Payload Size.....	31
Include ECRC Support.....	32
Flow Control Tab.....	32
Initial Receive Credits .....	32
Infinite PH Credits .....	32
Initial PH Credits Available.....	32
Infinite PD Credits .....	33
Initial PD Credits Available.....	33
Infinite NPH Credits.....	33
Initial NPH Credits Available .....	33
Infinite NPD Credits.....	33
Initial NPD Credits Available .....	33
Infinite CPLH Credits.....	33
Initial CPLH Credits Available .....	33
Infinite CPLD Credits.....	33
Initial CPLD Credits Available .....	33
Update Flow Control Generation Control .....	33
Number of P TLPs Between UpdateFC .....	33
Number of PD TLPs Between UpdateFC .....	34
Number of NP TLPs Between UpdateFC .....	34
Number of NPD TLPs Between UpdateFC .....	34
Number of CPL TLPs Between UpdateFC .....	34
Number of CPLD TLPs Between UpdateFC .....	34
Worst Case Number of 125 MHz Clock Cycles Between UpdateFC.....	34
<b>Chapter 4. IP Core Generation and Evaluation .....</b>	<b>35</b>
Licensing the IP Core.....	35

Licensing Requirements for ECP5/LatticeECP3 .....	35
IPexpress Flow for LatticeECP3 Devices .....	35
Getting Started .....	35
IPexpress-Created Files and Top Level Directory Structure .....	38
Running Functional Simulation .....	39
Synthesizing and Implementing the Core in a Top-Level Design .....	40
Hardware Evaluation .....	41
Updating/Regenerating the IP Core .....	41
Clarity Designer Flow for ECP5 Devices .....	42
Getting Started .....	42
Configuring and Placing the IP Core .....	44
Generating the IP Core .....	47
Clarity Designer-Created Files and Directory Structure .....	48
Instantiating the Core .....	49
Running Functional Simulation .....	49
Synthesizing and Implementing the Core in a Top-Level Design .....	50
Hardware Evaluation .....	50
Updating/Regenerating the IP Core .....	50
<b>Chapter 5. Using the IP Core .....</b>	<b>52</b>
Simulation and Verification .....	52
Simulation Strategies .....	52
Third Party Verification IP .....	53
FPGA Design Implementation .....	53
Setting Up the Core .....	53
Setting Up for x4 (No Flip) .....	53
Setting Up for x4 (Flipped) .....	54
Setting Design Constraints .....	54
Clocking Scheme .....	54
Locating the IP .....	55
Board-Level Implementation Information .....	56
PCI Express Power-Up .....	57
Board Layout Concerns .....	59
<b>Chapter 6. Core Verification .....</b>	<b>62</b>
<b>Chapter 7. Support Resources .....</b>	<b>63</b>
Lattice Technical Support .....	63
PCIe Solutions Web Site .....	63
PCI-SIG Website .....	63
References .....	63
ECP5 .....	63
LatticeECP3 .....	63
Revision History .....	63
<b>Appendix A. Resource Utilization .....</b>	<b>64</b>
Configuration .....	64
ECP5 Utilization (x4 RC-Lite) .....	64
Ordering Part Number .....	64
LatticeECP3 Utilization (x4 RC-Lite) .....	64
Ordering Part Number .....	64
ECP5 Utilization (x1 RC-Lite) .....	65
Ordering Part Number .....	65
LatticeECP3 Utilization (x1 RC-Lite) .....	65
Ordering Part Number .....	65

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 multi-drop bus in PCI. Each PCI Express device has the advantage of full duplex communication with its neighbor 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 Root Complex (RC) Lite core provides an x1 or x4 root complex solution from the electrical SERDES interface, physical layer, data link layer and a minimum transaction layer in PCI express protocol stack. This IP is a lighter version of the root complex intended to be used in simple local bus bridging applications. This solution supports the LatticeECP3™ and ECP5UM™ FPGA device families and is an extremely economical, high value FPGA platform.

This user guide covers following versions of the Lattice PCI Express RC-Lite core:

- The Native **PCI Express x4 RC-Lite Core** targets the LatticeECP3 and ECP5UM families of devices.
- The **Downgraded x2 Core** targets LatticeECP3 and ECP5UM family. The 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 **PCI Express x1 RC-Lite Core** targets the LatticeECP3 and LatticeECP5UM families. This is a reduced LUT count x1 core with a 16-bit datapath.

## Quick Facts

Table 1-1 gives quick facts about the PCI Express RC-Lite IP core.

**Table 1-1. PCI Express RC-Lite IP Core Quick Facts**

		PCI Express RC-Lite IP Configuration			
		Native x4 RC		Native x1 RC	
<b>Core Requirements</b>	FPGA Families Supported	LatticeECP3 and ECP5UM			
	Minimal Device Needed <sup>1</sup>	LFE3-17E-7FN484C	LFE5UM-45F-7BG756C	LFE3-17E-7FN484C	LFE5UM-45F-7BG756C
<b>Typical Resource Utilization</b>	Targeted Device	LFE3-70E-7FN672C	LFE5UM-85F-7BG756CES	LFE3-70E-7FN672C	LFE5UM-85F-7BG756CES
	Data Path Width	64	64	16	16
	LUTs	10650	11900	4700	4650
	sysMEM EBRs	9	9	3	3
	Registers	8500	8200	3100	3150
<b>Design Tool Support</b>	Lattice Implementation	Lattice Diamond® 3.4			
	Synthesis	Synopsys® Synplify® Pro for Lattice H-2013.03L			
	Simulation	Aldec Active-HDL® 9.2 (Windows only, Verilog and VHDL) Mentor Graphics ModelSim® SE 6.6g (Verilog Only)			

---

## Features

The Lattice PCI Express RC-Lite IP core supports the following features.

### PHY Layer

- 2.5 Gbps CML electrical interface
- PCI Express 1.1 electrical compliance
- Serialization and de-serialization
- 8b10b symbol encoding/decoding
- Link state machine for symbol alignment
- Clock tolerance compensation supports +/- 300 ppm
- Framing and application of symbols to lanes
- Data scrambling and de-scrambling
- 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

### Data Link Layer

- Data link control and management state machine
- Flow control initialization
- Ack/Nak DLLP generation/termination
- LCRC generation/checking
- Sequence number appending/checking/removing
- Retry buffer and management
- Receiver buffer

### Transaction Layer

- Transmit and Receive Flow control
- Malformed and poisoned TLP detection
- Optional ECRC generation/checking
- INTx message TLP decoding and interrupt signaling to user
- Error message TLP decoding and signaling to user.
- 128, 256, 512, 1k, 2k or 4k bytes maximum payload size

## Top Level IP Support

- 125 MHz user interface
  - Native x4 and Downgraded x2 support a 64-bit data path
  - x1 supports a 16-bit data path
- 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
- Higher layer control of LTSSM via ports

# Functional Description

This chapter provides a functional description of the Lattice PCI Express RC-Lite IP core.

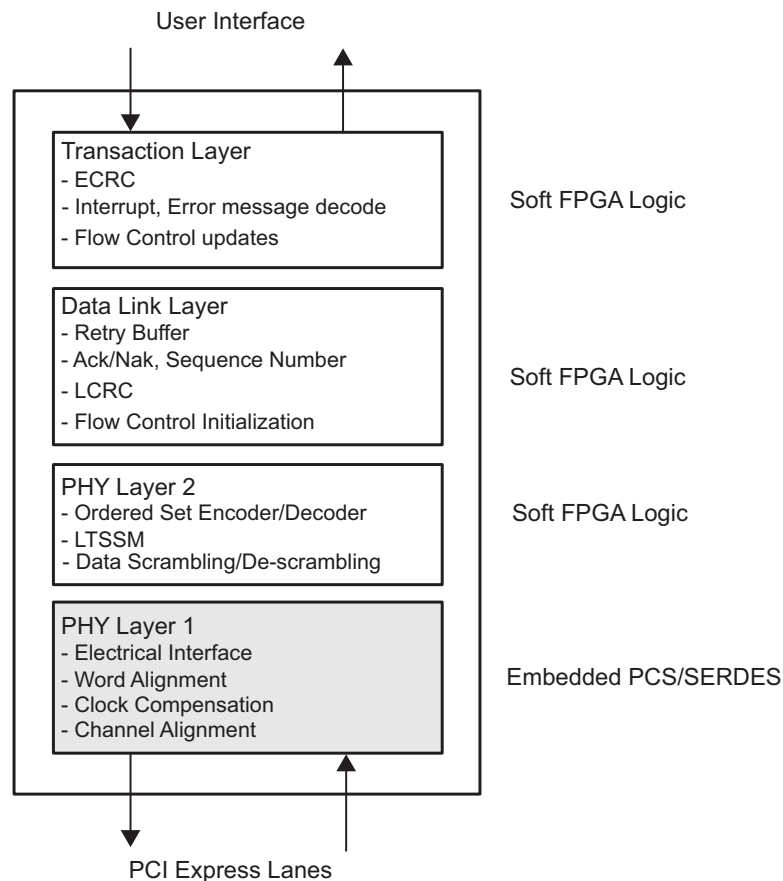
## Overview

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

The IPexpress™ tool for LatticeECP3 and Clarity Designer for ECP5UM devices is used to customize and create a complete IP module for the user to instantiate in a design. Inside the module created by the IPexpress/Clarity 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.

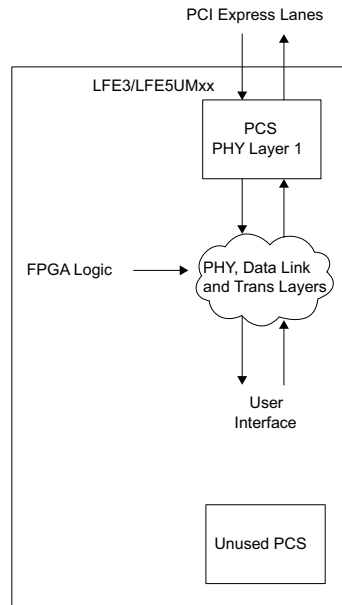
Figure 2-1 provides a high-level block diagram to illustrate the main functional blocks and the technology used to implement PCI Express RC-Lite IP core functions.

**Figure 2-1. .PCI Express RC-Lite IP Core Technology and Functions**



As the PCI Express RC-Lite IP 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 LFE3/LFE2M devices with a PCI Express RC-Lite IP core.

Figure 2-2. PCI Express RC-Lite IP Core Implementation in LatticeECP3 and ECP5 Devices



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 SERDES/PCS blocks as described in the [Overview](#) section. The FPGA logic placement and routing is the job of the Diamond design tools to select regions nearby the hard SERDES/PCS blocks to achieve the timing goals.



Figure 2-3 provides a high-level interface representation.

**Figure 2-3. PCI Express RC-Lite IP Core Interfaces**

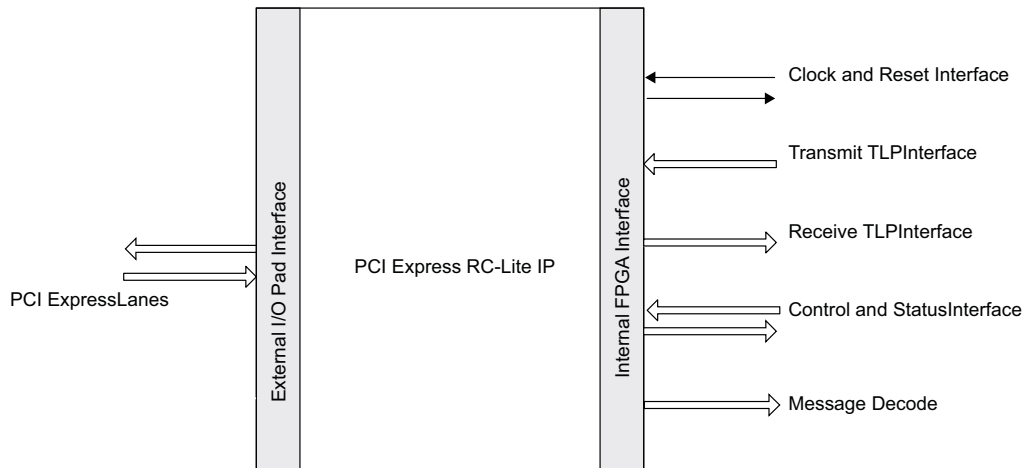


Table 2-1 provides the list of ports and descriptions for the PCI Express RC-Lite IP core.

**Table 2-1. PCI Express RC-Lite IP Core Port List**

Port Name	Direction	Clock	Description
<b>Clock and Reset Interface</b>			
refclk[p,n]	Input		100 MHz PCI Express differential reference clock used to generate the 2.5 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. This port will be connected to the GSR for the entire device. This reset is pulsed after bit stream download and will not need to be asserted by the user.
hdin[p,n]_[0,1,2,3]	Input		PCI Express 2.5 Gbps CML inputs for lanes 0,1,2, and 3. The port "flip_lanes" is used to define the connection of PCS/SERDES channel to PCI Express lane. flip_lanes=0 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  flip_lanes=1 hdin[p,n]_0 - PCI Express Lane 3 hdin[p,n]_1 - PCI Express Lane 2 hdin[p,n]_2 - PCI Express Lane 1 hdin[p,n]_3 - PCI Express Lane 0

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
hdout[p,n]_[0,1,2,3]	Output		PCI Express 2.5 Gbps CML outputs for lanes 0,1,2, and 3. The port “flip_lanes” is used to define the connection of PCS/SERDES channel to PCI Express lane.  flip_lanes=0 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  flip_lanes=1 hdout[p,n]_0 - PCI Express Lane 3 hdout[p,n]_1 - PCI Express Lane 2 hdout[p,n]_2 - PCI Express Lane 1 hdout[p,n]_3 - PCI Express Lane 0
<b>Transmit TLP Interface</b>			
tx_data_vc0[n:0]	Input	sys_clk_125	x4 Transmit data bus [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  x1 Transmit data bus [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 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 available only on the x4 core.
tx_val	Output	sys_clk_125	Active High transmit clock enable. When a x4 is downgraded to a x1, this signal is used as the clock enable to downshift the transmit bandwidth. This port is available only on the x4 core.

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
tx_ca_[ph,nph,cplh]_vc0[8:0]	Output	sys_clk_125	Transmit Interface 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 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.
<b>Receive TLP Interface</b>			
rx_data_vc0[n:0]	Output	sys_clk_125	x4 Receive data bus [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  x1 Receive data bus [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 available only on the x4 core.
rx_ecrc_err_vc0	Output	sys_clk_125	Active High ECRC error indicator. Indicates a ECRC error in the current TLP. This port is available only if ECRC feature is selected while generating the IP core.
rx_pois_tlp_vc0	Output	sys_clk_125	Active High poisoned TLP indicator. Asserted if “poisoned (EP) “ bits is set in any TLP with data.
rx_malf_tlp_vc0	Output	sys_clk_125	Active High malformed TLP indicator. Indicates a problem with the current TLPs length or format.
[ph,pd, nph,npd,cplh,cpld]_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,cplh]_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.

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
[pd, npd, cpld]_processed_vc0	Input	sys_clk_125	Active High enable for [pd, npd, cpld]_num_vc0 port. The user should place the number of data credits processed on the [pd, npd, cpld]_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 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 or CPLD credits processed. It is enabled by the [pd, npd, cpld]_processed_vc0 port.
<b>Control and Status</b>			
<b>PHYSICAL LAYER</b>			
flip_lanes	Input	Async	Reverses the lane connections to the SERDES. This function is used to provide flexibility for the PCB layout. The "Locating" section later in this document describes how this function can be used. 0-Lane 0 connects to SERDES Channel 0, etc. 1-Lane 0 connects to SERDES Channel 3, etc.
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

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
phy_ltssm_substate[2:0]	Output	sys_clk_125	<p>PHY Layer LTSSM current sub state. Each major LTSSM state has a series of sub states.</p> <p>When phy_ltssm_state=DETECT</p> <ul style="list-style-type: none"> <li>000 - DET_WAIT</li> <li>001 - DET_QUIET</li> <li>010 - DET_GODET1</li> <li>011 - DET_ACTIVE1</li> <li>100 - DET_WAIT12MS</li> <li>101 - DET_GODET2</li> <li>110 - DET_ACTIVE2</li> <li>111 - DET_EXIT</li> </ul> <p>When phy_ltssm_state=POLLING</p> <ul style="list-style-type: none"> <li>000 - POL_WAIT</li> <li>001 - POL_ACTIVE</li> <li>010 - POL_COMPLIANCE</li> <li>011 - POL_CONFIG</li> <li>100 - POL_EXIT</li> </ul> <p>When phy_ltssm_state=CONFIG</p> <ul style="list-style-type: none"> <li>000 - CFG_WAIT</li> <li>001 - CFG_LINK_WIDTH_ST</li> <li>010 - CFG_LINK_WIDTH_ACC</li> <li>011 - CFG_LANE_NUM_WAIT</li> <li>100 - CFG_LANE_NUM_ACC</li> <li>101 - CFG_COMPLETE</li> <li>110 - CFG_IDLE</li> <li>111 - CFG_EXIT</li> </ul> <p>When phy_ltssm_state=L0</p> <ul style="list-style-type: none"> <li>000 - L0_WAIT</li> <li>001 - L0_L0</li> <li>010 - L0_L0RX</li> <li>011 - L0_LOTX</li> <li>100 - L0_IDLE_0</li> <li>101 - L0_IDLE_1</li> <li>110 - L0_EXIT</li> </ul> <p>When phy_ltssm_state=L0s</p> <ul style="list-style-type: none"> <li>000 - L0s_RX_WAIT</li> <li>001 - L0s_RX_ENTRY</li> <li>010 - L0s_RX_IDLE</li> <li>011 - L0s_RX_FTS</li> <li>100 - L0s_RX_EXIT</li> </ul> <p>When phy_ltssm_state=L1</p> <ul style="list-style-type: none"> <li>000 - L1_WAIT</li> <li>001 - L1_ENTRY</li> <li>010 - L1_IDLE</li> <li>011 - L1_EXIT</li> </ul> <p>When phy_ltssm_state=L2</p> <ul style="list-style-type: none"> <li>000 - L2_WAIT</li> <li>001 - L2_IDLE</li> </ul>
phy_cfgln_sum[2:0]	Output	sys_clk_125	<p>Link Width</p> <ul style="list-style-type: none"> <li>000 - No link defined</li> <li>001 - Link width = 1</li> <li>100 - Link width = 4</li> </ul>

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
phy_pol_compliance	Output	sys_clk_125	Active High indicator that the LTSSM is in the Polling.Compliance state.
phy_force_cntl[3:0]	Input	sys_clk_125	These signals to be used only for debug purpose. [0] - force_lsm_active - Forces the Link State Machine for all of the channels to the linked state. [1] - force_rec_ei - Forces the detection of a received electrical idle. [2] - force_phy_status - Forces the detection of a receiver during the LTSSM Detect state on all of the channels. [3] - force_disable_scr - Disables scrambler and de-scrambler.
phy_ltssm_cntl[15:0]	Input	sys_clk_125	[0] – no_pcie_train - Active High signal disables LTSSM training and forces the LTSSM to L0 as a x4 configuration. This is intended to be used in simulation only to force the LTSSM into the L0 state. [1] - retrain - Active High request to re-train the link when LTSSM is in L0. [2] – hl_disable_scr - Active High to set the disable scrambling bit in the TS1/TS2 sequence [3] – hl_gto_dis – Active High request to go to Disable state when LTSSM is in Config or Recovery. [4] – hl_gto_det – Active High request to go to Detect state when LTSSM is in L2 or Disable. [5] – hl_gto_hrst - Active High request to go to Hot Reset when LTSSM is in Recovery. [6] – hl_gto_l0stx - Active High request to go to L0s when LTSSM is in L0. [7] – hl_gto_l0stxfts - Active High request to go to L0s and transmit FTS when LTSSM is in L0s. [8] – hl_gto_l1 - Active High request to go to L1 when LTSSM is in L0. [9] – hl_gto_l2 - Active High request to go to L2 when LTSSM is in L0. [13:10] – hl_gto_lbk - Active High request to go to Loopback when LTSSM is in Config or Recovery [14] – hl_gto_rcvry - Active High request to go to Recovery when LTSSM is in L0, L0s or L1. [15] – hl_gto_cfg - Active High request to go to Config when LTSSM is in Recovery.
tx_lbk_rdy	Output	sys_clk_125	This output port is used to enable the transmit master loopback data. This port is only available if the Master Loopback feature is enabled in the IPexpress tool.
tx_lbk_kcntl[3:0]	Input	sys_clk_125	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 IPexpress tool.  <b>x4 port</b> [7] - K control on tx_lbk_data[63:56] [6] - K control on tx_lbk_data[55:48] [5] - K control on tx_lbk_data[47:40] [4] - K control on tx_lbk_data[39:32] [3] - K control on tx_lbk_data[31:24] [2] - K control on tx_lbk_data[23:16] [1] - K control on tx_lbk_data[15:8] [0] - K control on tx_lbk_data[7:0]  <b>x1 port</b> [1] - K control on tx_lbk_data[15:8] [0] - K control on tx_lbk_data[7:0]

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
tx_lbk_data[31:0]	Input	sys_clk_125	<p>This input port is used to send 32-bit data for the master loopback. This port is only available if the Master Loopback feature is enabled in the IPexpress tool.</p> <p><b>x4 port</b>            [63:56] - Lane 7 data            [55:48] - Lane 6 data            [47:40] - Lane 5 data            [39:32] - Lane 4 data            [31:24] - Lane 3 data            [23:16] - Lane 2 data            [15:8] - Lane 1 data            [7:0] - Lane 0 data</p> <p><b>x1 port</b>            [15:8] - Lane 1 data            [7:0] - Lane 0 data</p>
rx_lbk_kcntl[3:0]	Output	sys_clk_125	<p>This output port is used to indicate a K control word is being received on rx_lbk_data port. This port is only available if the Master Loopback feature is enabled in the IPexpress tool.</p> <p><b>x4 port</b>            [7] - K control on rx_lbk_data[63:56]            [6] - K control on rx_lbk_data[55:48]            [5] - K control on rx_lbk_data[47:40]            [4] - K control on rx_lbk_data[39:32]            [3] - K control on rx_lbk_data[31:24]            [2] - K control on rx_lbk_data[23:16]            [1] - K control on rx_lbk_data[15:8]            [0] - K control on rx_lbk_data[7:0]</p> <p><b>x1 port</b>            [1] - K control on rx_lbk_data[15:8]            [0] - K control on rx_lbk_data[7:0]</p>
rx_lbk_data[31:0]	Output	sys_clk_125	<p>This output port is used to receive 32-bit data for the master loopback. This port is only available if the Master Loopback feature is enabled in the IPexpress tool.</p> <p><b>x4 port</b>            [63:56] - Lane 7 data            [55:48] - Lane 6 data            [47:40] - Lane 5 data            [39:32] - Lane 4 data            [31:24] - Lane 3 data            [23:16] - Lane 2 data            [15:8] - Lane 1 data            [7:0] - Lane 0 data</p> <p><b>x1 port</b>            [15:8] - Lane 1 data            [7:0] - Lane 0 data</p>

**Table 2-1. PCI Express RC-Lite IP Core Port List (Continued)**

Port Name	Direction	Clock	Description
<b>DATA LINK LAYER</b>			
dll_status[7:0]	Output	sys_clk_125	[0] – dl_up - Data Link Layer is in the up and ready to send/receive TLPs from/to Transaction Layer. [1] - dl_init - Data Link Layer is in the DL_Init state. [2] – dl_inactive - Data Link Layer is the DL_Inactive state. [3] – bad_dllp – Data link Layer received a bad DLLP. [4] – dlerr – Data Link Layer Protocol error. [5] – bad_tlp - Data link Layer received a bad TLP. [6] – rply_tout – Indicates a replay timeout [7] – rnum_rlor – Indicates replay number roll over which initiates Link re-training.
tx_dllp_val	Input	sys_clk_125	Active High power message send command. 00 - Nothing to send 01 - Send DLLP using tx_pmtyp DLLP 10 - Send DLLP using tx_vsd_data Vendor Defined DLLP 11 - Not used
tx_pmtyp[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	Received vendor-defined DLLP data.
rxdp_dllp_val	Output	sys_clk_125	Active High DLLP received
<b>TRANSACTION LAYER</b>			
ecrc_gen_enb	Input	Async	Active High enable for ECRC generation. When asserted, IP generates and inserts ECRC to transmitting TLPs. When asserted, the TD bit in the transmit TLP header must be set. This port is available only if ECRC feature is selected while generating the IP core.
ecrc_chk_enb	Input	Async	Active High enable for ECRC checking. When asserted, IP checks ECRC in received TLPs. This port is available only if ECRC feature is selected while generating the IP core.
<b>MESSAGE DECODES</b>			
int[a,b,c,d]_n	Output	sys_clk_125	Active Low interrupt wires. 0 – Received Assert INTx message TLP. 1 – Received De-Assert INTx message TLP.
nftl_err_msg	Output	sys_clk_125	Active High signal indicates receiving a NON-FATAL ERROR message TLP.
ftl_err_msg	Output	sys_clk_125	Active High signal indicates receiving a FATAL ERROR message TLP.
corr_err_msg	Output	sys_clk_125	Active High signal indicates receiving a CORRECTABLE ERROR message TLP.



## Interface Description

This section describes the datapath user interfaces of the IP core. Both the transmit and receive interfaces use the TLP as the data structure.

### 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,cplh,cpld]\_vc0 bus. There must be enough credits available for the entire TLP to be sent.

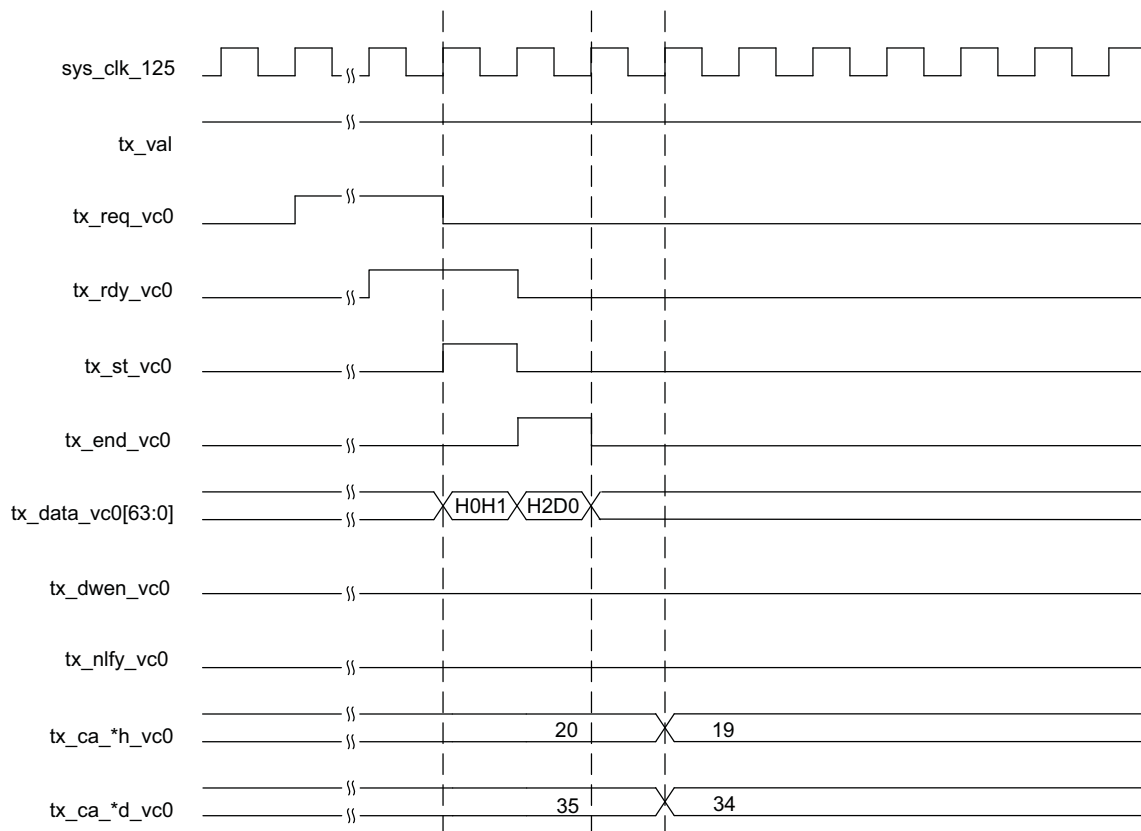
The user must then check that the core is ready to send the TLP. This is done by asserting the tx\_req\_vc0 port and waiting for the assertion of tx\_rdy\_vc0. If there is enough credit, the user can proceed with sending the data based on tx\_rdy\_vc0. If the credit becomes insufficient, tx\_req\_vc0 must be de-asserted 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 will assert tx\_st\_vc0.

The tx\_rdy\_vc0 signal 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.

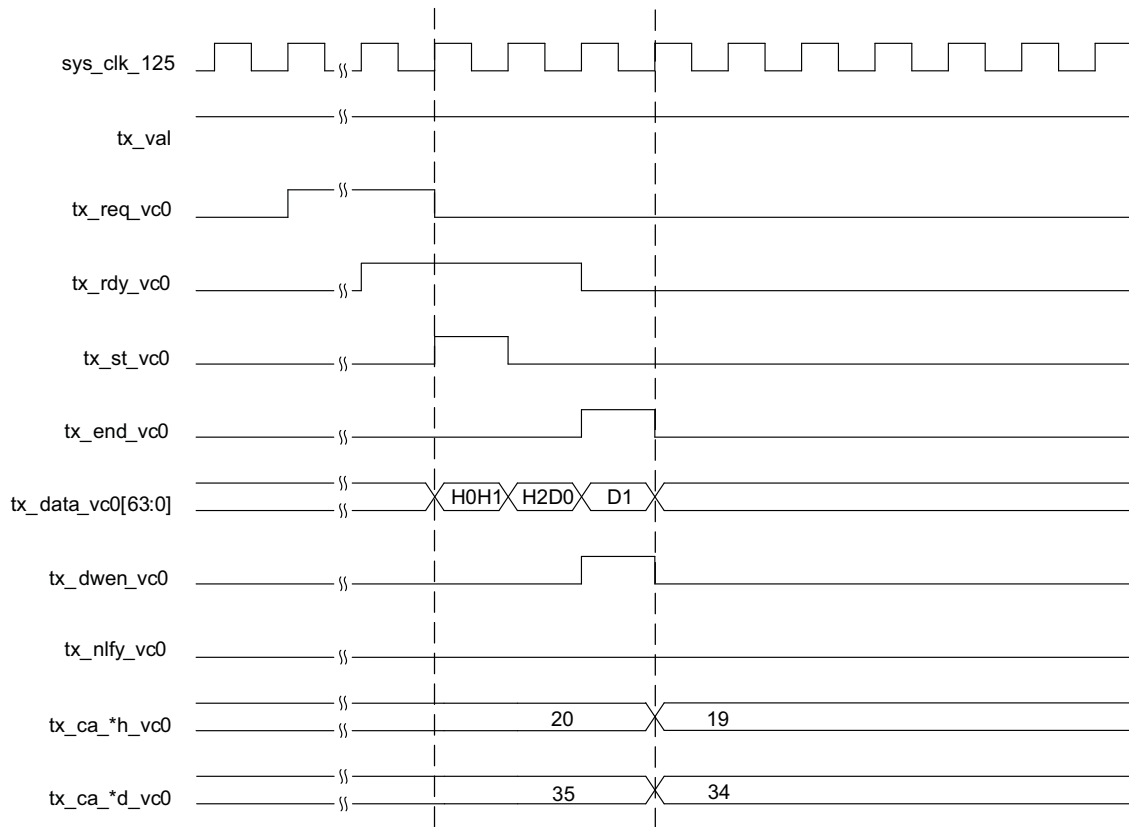
### Transmit TLP Interface Waveforms for x4 Core

Figure 2-4 through Figure 2-10 provide timing diagrams for the tx interface signals with a 64-bit datapath.

**Figure 2-4. Transmit Interface x4, 3DW Header, 1 DW Data**



**Figure 2-5. Transmit Interface x4, 3DW Header, 2 DW Data**



**Figure 2-6. Transmit Interface x4, 4DW Header, 0 DW**

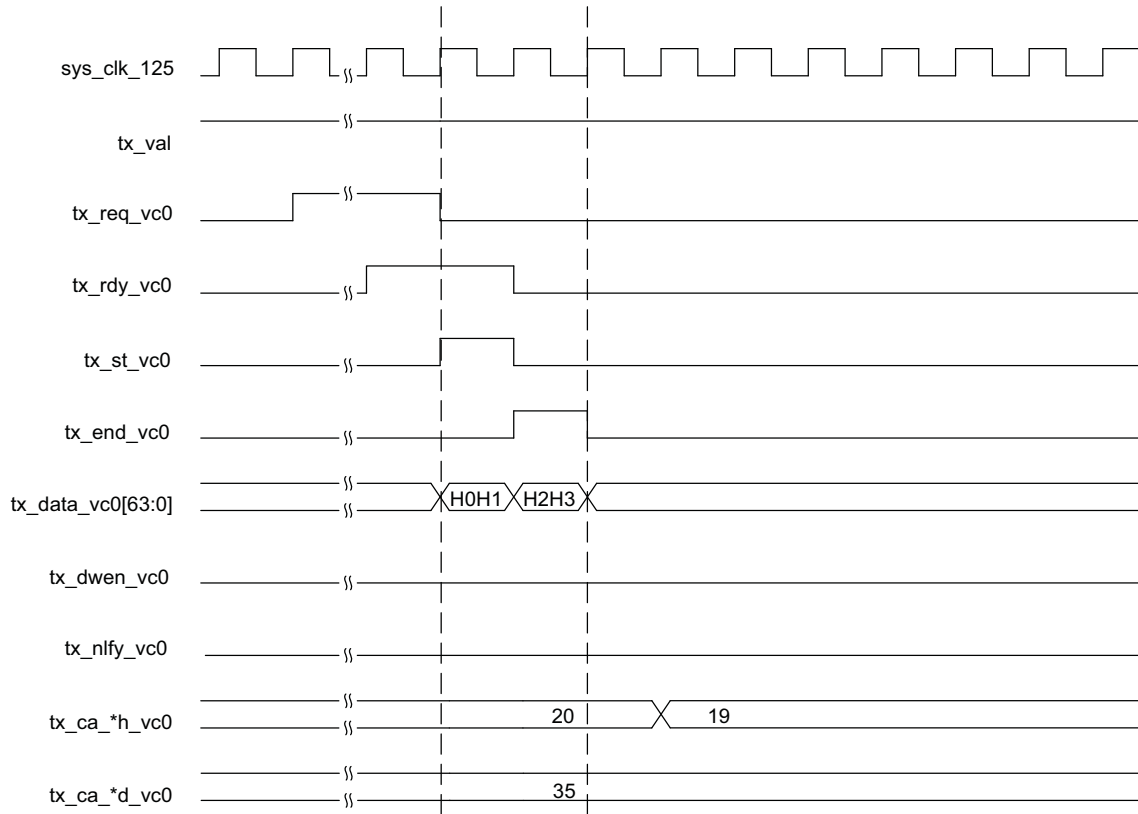
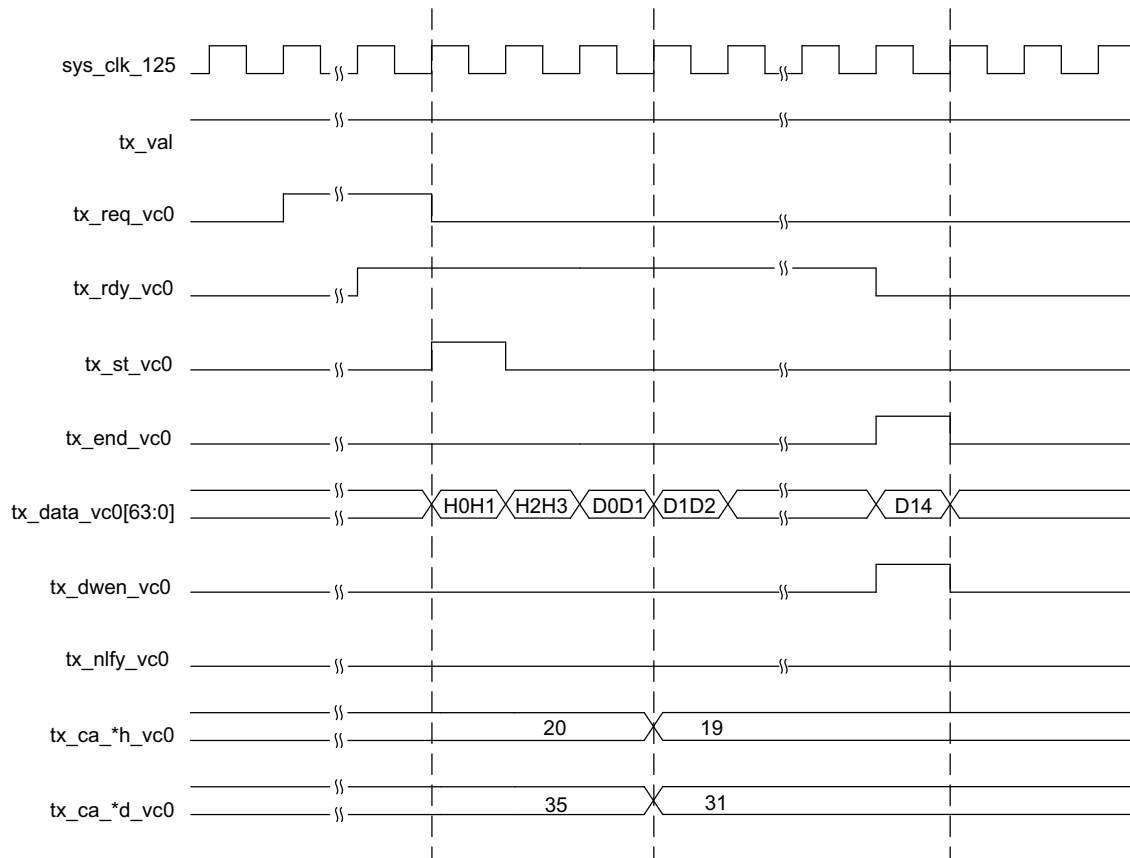
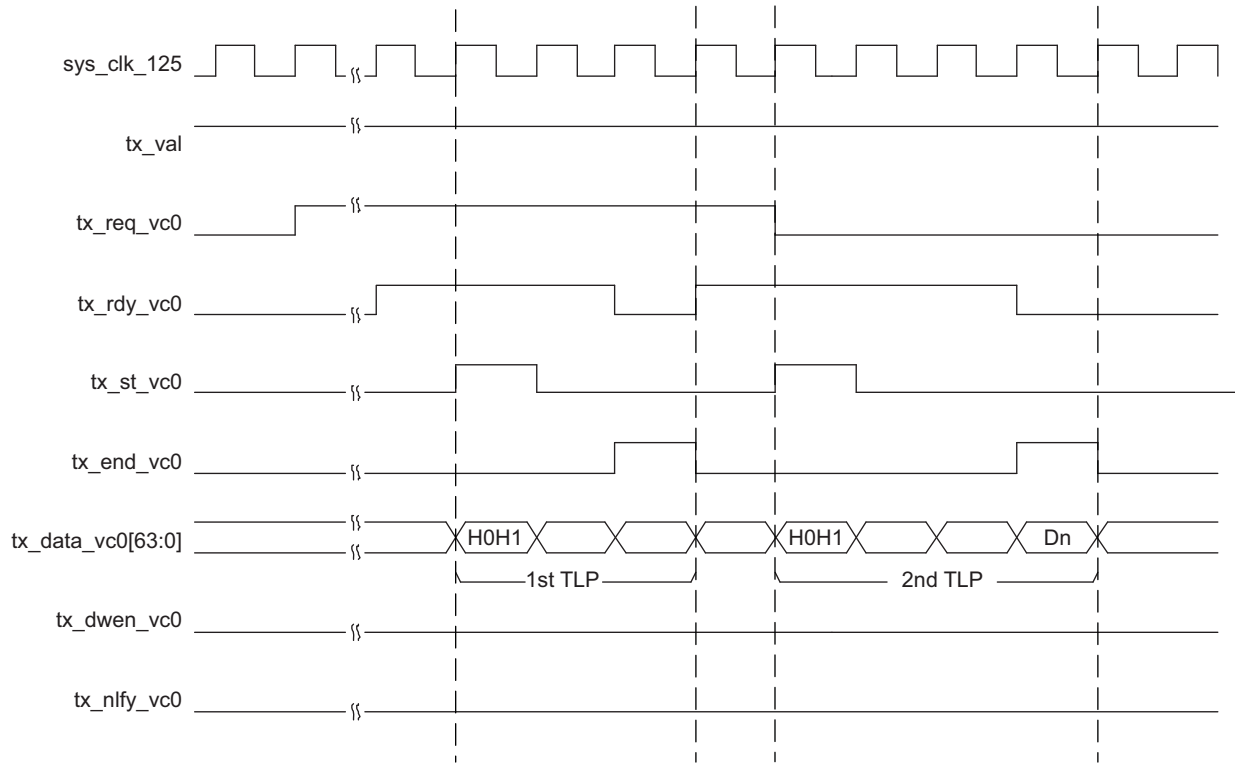


Figure 2-7. Transmit Interface x4, 4DW Header, Odd Number of DWs



**Figure 2-8. Transmit Interface x4, Burst of Two TLPs**



**Figure 2-9. Transmit Interface x4, Nullified TLP**

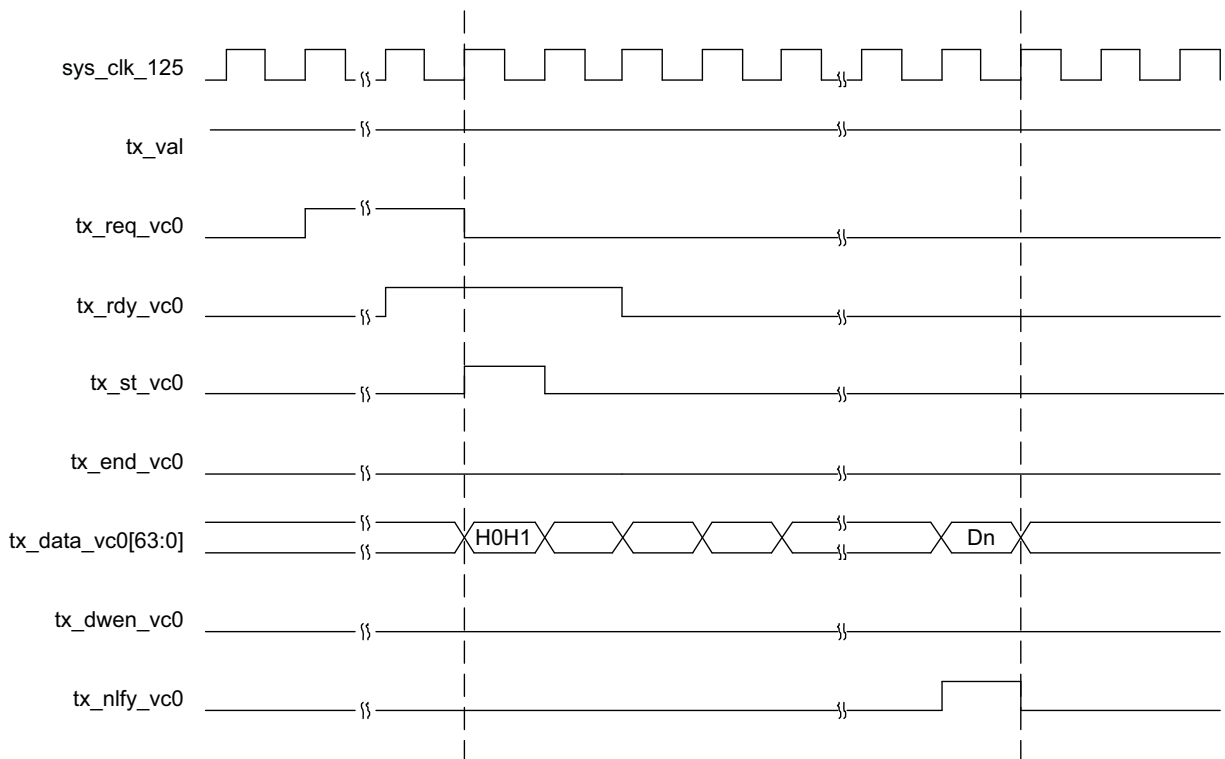
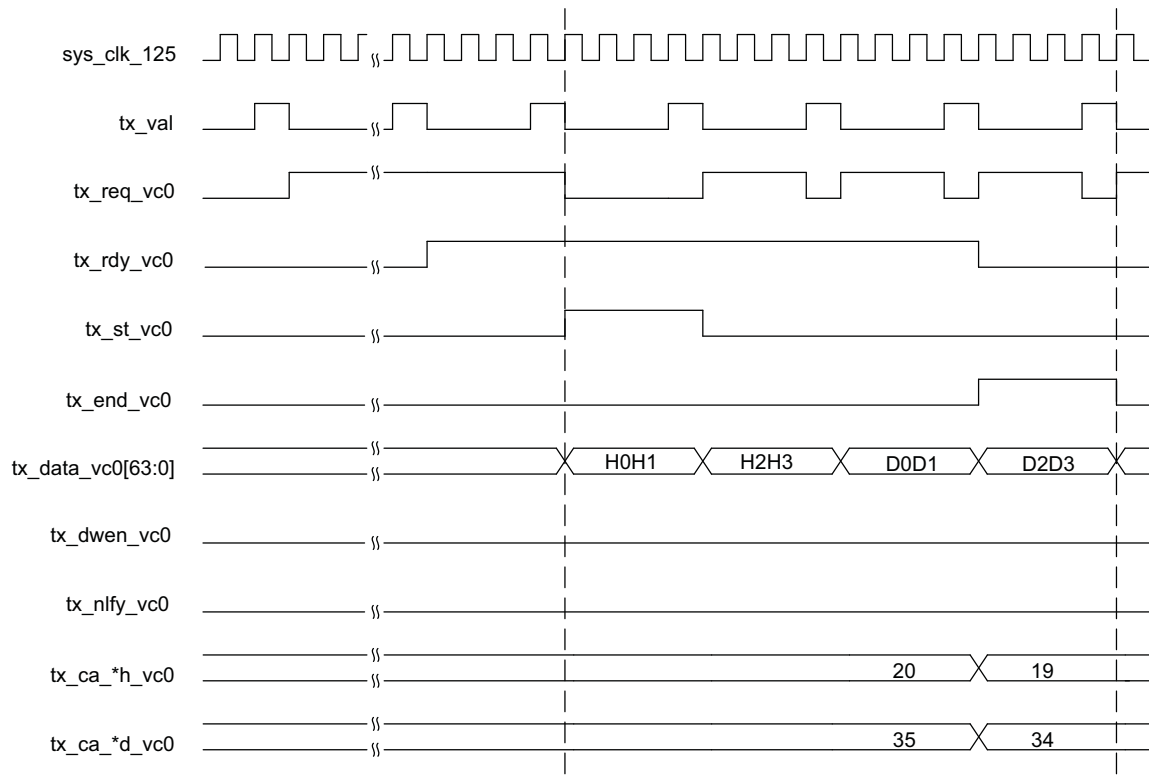
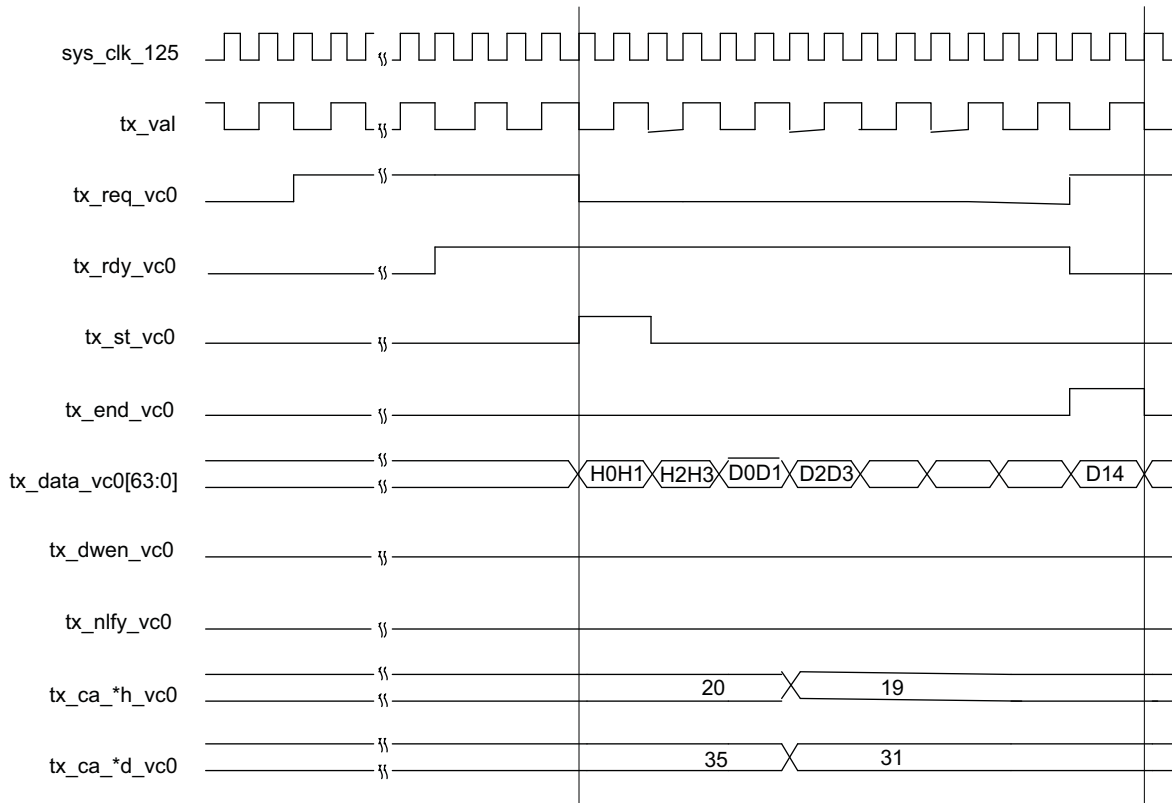


Figure 2-10. Transmit Interface x4 Downgraded to x1 Using tx\_val



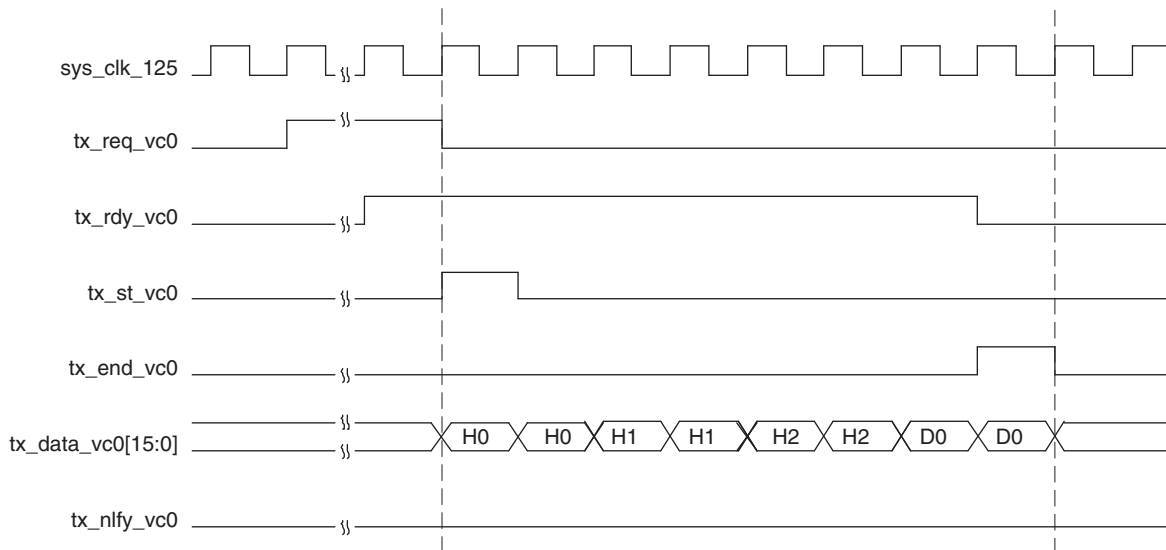
**Figure 2-11. Transmit Interface x2 Downgraded Using tx\_val**



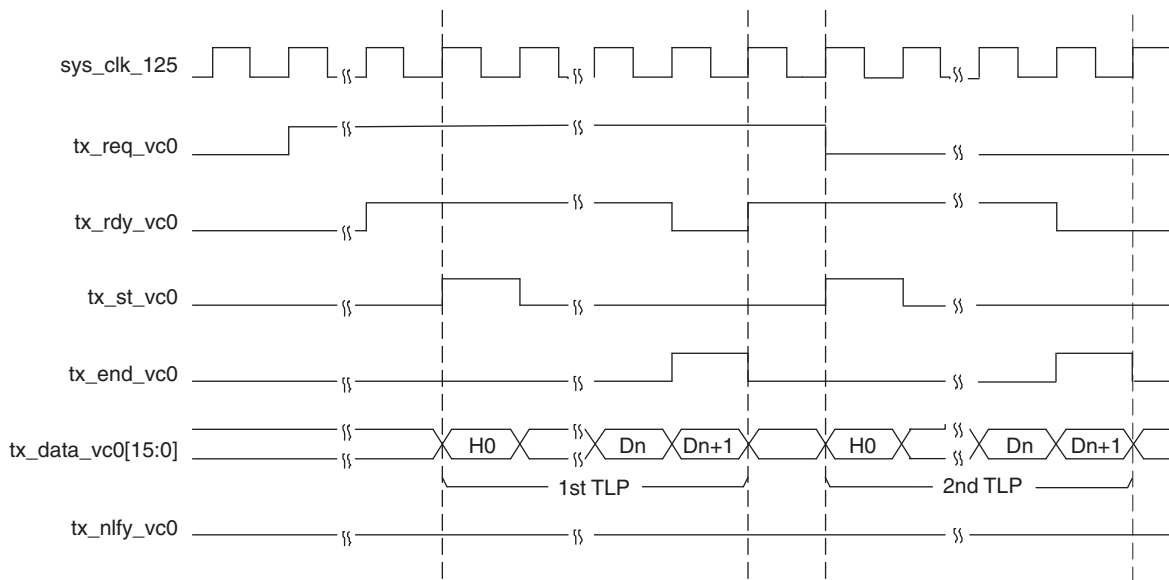
**Transmit TLP Interface Waveforms for x1**

Figure 2-12 through Figure 2-14 provide timing diagrams for the transmit interface signals with a 16-bit datapath.

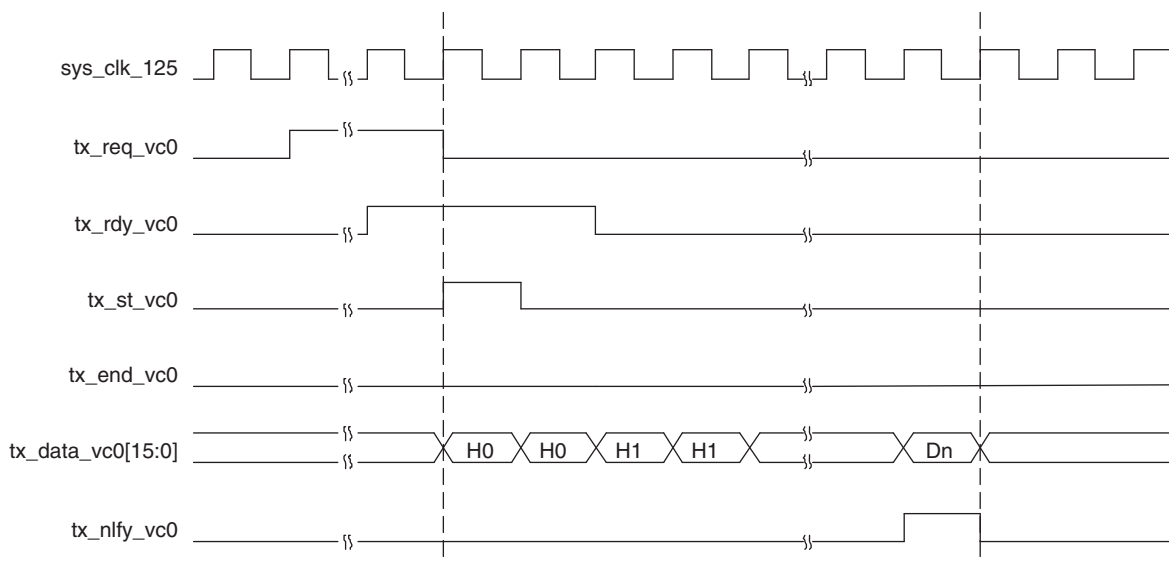
**Figure 2-12. Transmit Interface x1, 3DW Header, 1 DW Data**



**Figure 2-13. Transmit Interface x1, Burst of Two TLPs**



**Figure 2-14. Transmit Interface x1, Nullified TLP**



### Receive TLP Interface

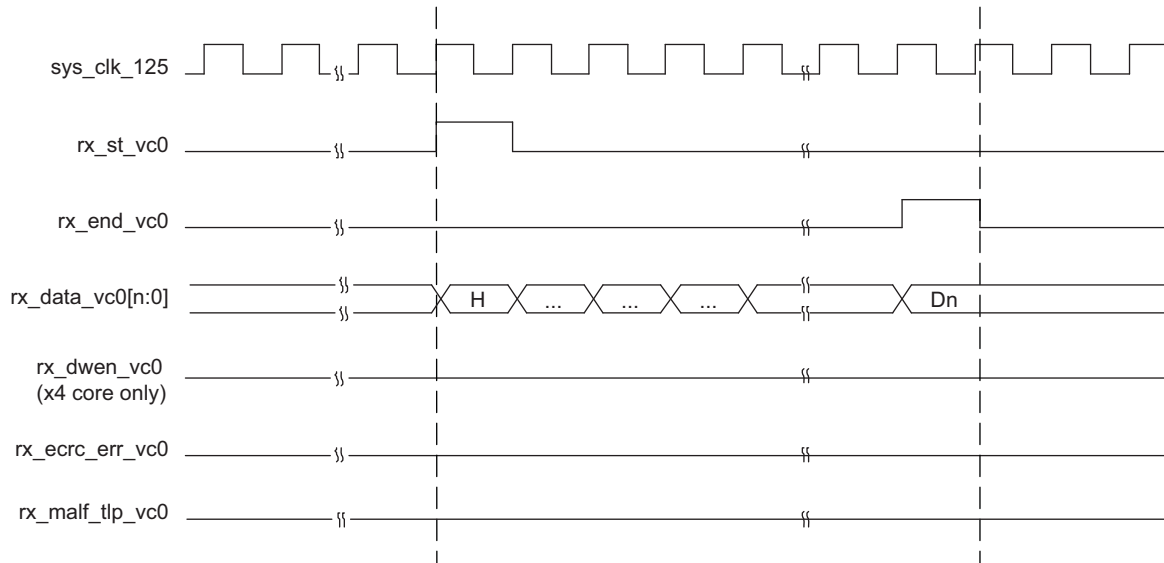
In the receive direction, TLPs will come from the core as they are received on the PCI Express lanes. Interrupt messages and Error Message TLPs are processed by the IP core and corresponding signals are provided through ports. Interrupt message TLPs are decoded to ports `inta_n`, `intb_n`, `intc_n` and `intd_n`. Error message TLPs are decoded to ports `ftl_err_msg`, `nftl_err_msg` and `cor_err_msg`. Figure 2-15 through Figure 2-17 provide timing diagrams of the these ports.

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 will be 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 2-15 through Figure 2-17 provide timing diagrams of the receive interface.

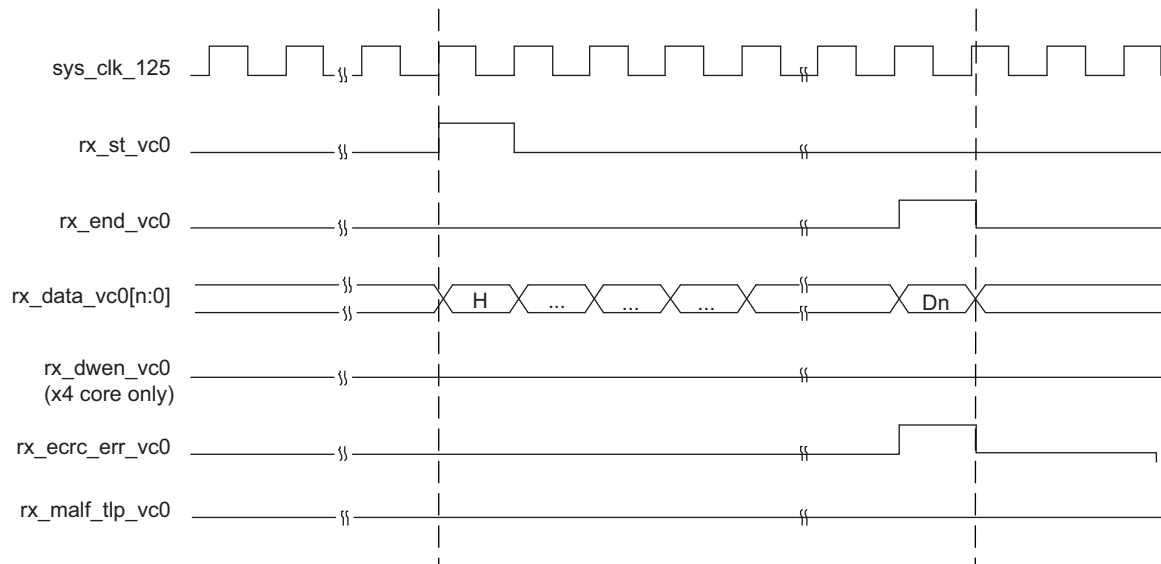


TLPs come from the receive interface only as fast as they come from the PCI Express lanes. There will always be at least one clock cycle between rx\_end\_vc0 and the next rx\_st\_vc0.

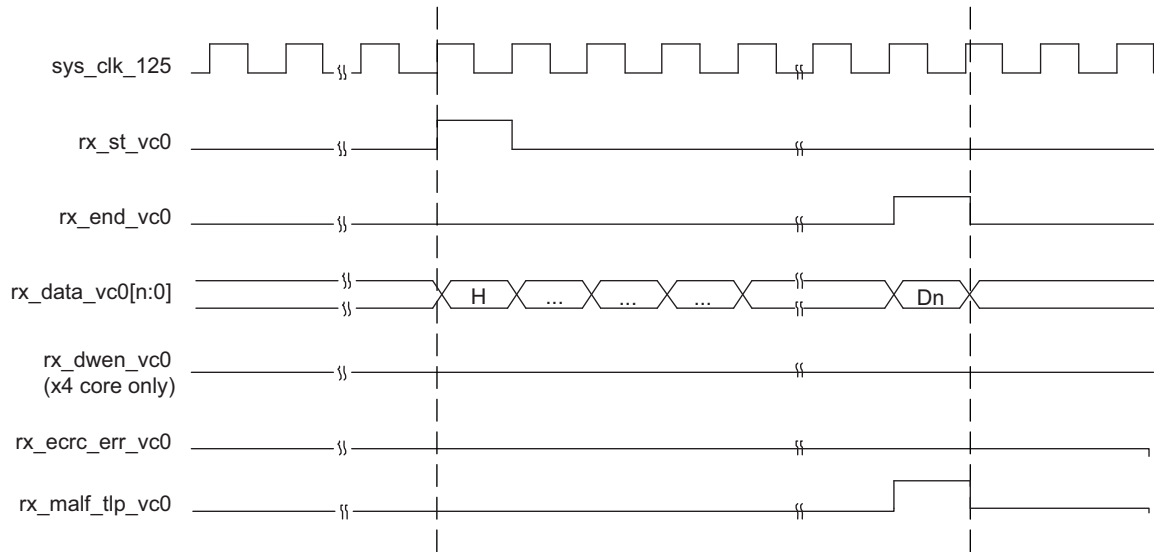
**Figure 2-15. Receive Interface, Clean TLP**



**Figure 2-16. Receive Interface, ECRC Errored TLP**



**Figure 2-17. Receive Interface, Malformed TLP**



## Message Decode Interface

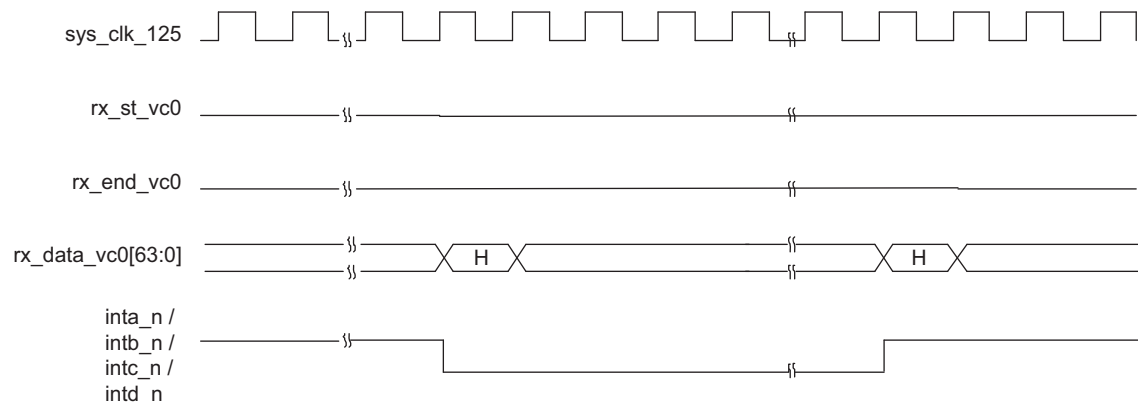
### Interrupt Signaling Messages

As a receiver, RC-Lite will decode all INTx Interrupt signaling messages and signal corresponding interrupts on ports to the user. Refer to signal descriptions of ports “inta\_n, intb\_n, intc\_n and intd\_n” in Table 2-1.

The signals “inta\_n, intb\_n, intc\_n and intd\_n” are active Low and are set to 1'b1 after the reset. When RC-Lite receives a valid “Assert\_INTA, Assert\_INTB, Assert\_INTC or Assert\_INTD” messages, corresponding port “inta\_n, intb\_n, intc\_n or intd\_n” is reset to 1'b0 and held until a corresponding “Deassert\_INTA, Deassert\_INTB, Deassert\_INTC or Deassert\_INTD” message is received.

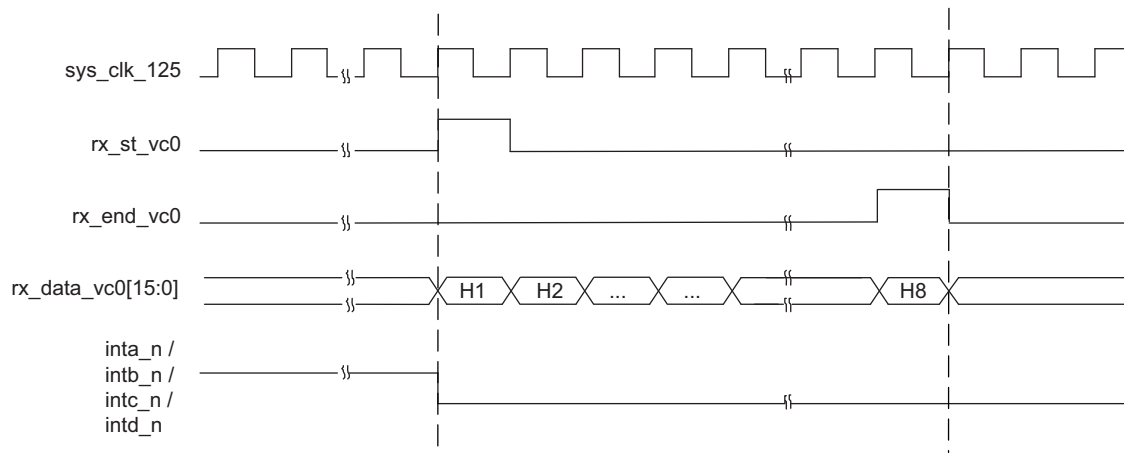
For designs using x4, whenever these ports change from 1'b1 to 1'b0 or 1'b0 to 1'b1, the data bus rx\_data\_vc0[63:0] contain the byte0 through byte7 of the message TLP header in the following clock cycle. Refer to Figure 2-18 for an interface diagram.

**Figure 2-18. Message Decode Interface x4, INTx signaling**



For designs using x1, the rx\_st\_vc0[15:0] signal will be asserted with the first word of the TLP. The remaining seven words of the interrupt message will be provided on consecutive clock cycles until the last word with rx\_end\_vc0 is asserted. Refer to Figure 2-19 for an interface diagram.

**Figure 2-19. Message Decode Interface x1, INTx signaling**

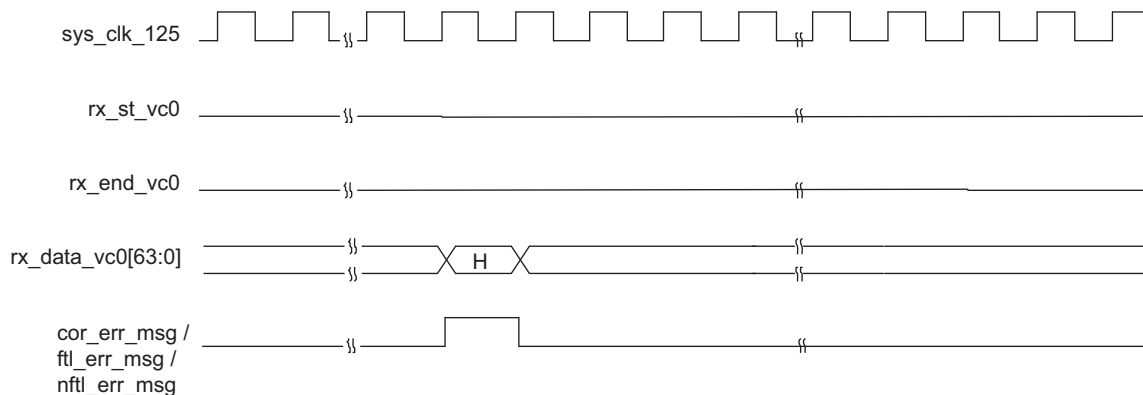


### Error Signaling Messages

As a receiver, RC-Lite will decode all Error signaling messages and signal corresponding errors on ports to the user. Refer to signal descriptions of ports “ftl\_err\_msg, nftl\_err\_msg and cor\_err\_msg” in Table 2-1.

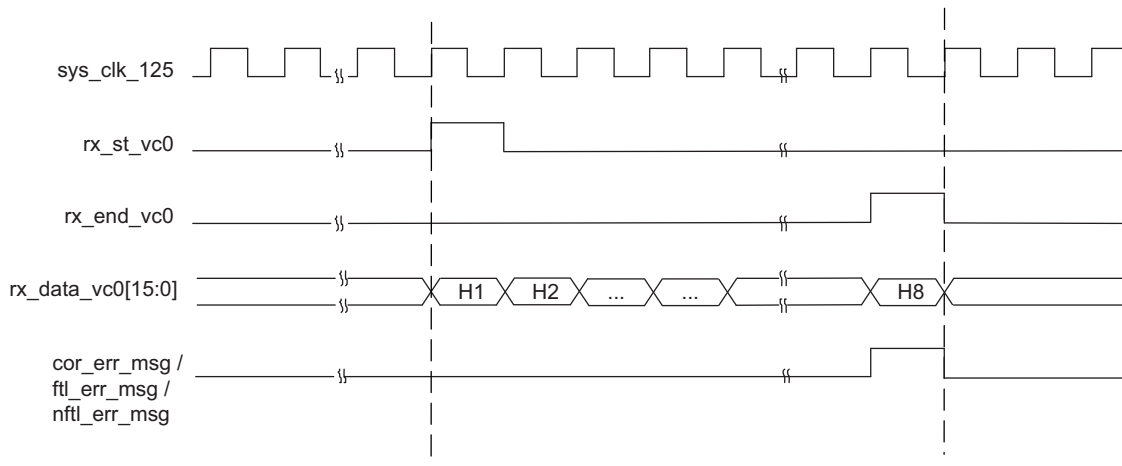
The signals “cor\_err\_msg, nftl\_err\_msg and ftl\_err\_msg” are active High and are set to 1'b0 after the reset. For designs using x4, when RC-Lite receives a valid “ERR\_COR or ERR\_NONFATAL or ERR\_FATAL” message, port “cor\_err\_msg or nftl\_err\_msg or ftl\_err\_msg” is set to 1'b1 for one clock cycle. Whenever these ports are pulsed, the data bus rx\_data\_vc0[63:0] contains the byte0 through byte7 of the message TLP header. Refer to Figure 2-20 for an interface diagram.

**Figure 2-20. Message Decode Interface x4, Error signaling**



For designs using x1, rx\_st\_vc0[15:0] signal will be asserted with the first word of the TLP. The remaining seven words of the error message will be provided on consecutive clock cycles until the last word with both rx\_end\_vc0 and err\_msg is asserted. Refer to Figure 2-21 for an interface diagram.

**Figure 2-21. Message Decode Interface x1, Error signaling**



## Using the Transmit and Receive Interfaces

There are two ways a PCI Express RC-Lite can interact with an endpoint. As a completer, the RC-Lite will respond to accesses made by the endpoint. As an initiator, the RC-Lite will perform accesses to the endpoint. The following sections will discuss how to use transmit and receive TLP interfaces for both of these types of interactions.

### As a Receiver

As a receiver, RC-Lite can receive any type of TLP from an endpoint. When a TLP other than Interrupt or error message TLP is received, the PCI Express RC-Lite core will forward the TLP to the user interface. The `rx_st_vc0` will be asserted with `rx_data_vc0` providing the first eight bytes of the TLP. The TLP 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 errored TLPs. 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.

If the TLP is a 32-bit MWr TLP or a 64-bit MWr TLP, 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 one header credit. There is one data credit used per four DWs of data. The length field provides the number of DWs used in the TLP. If the TLP length is on the 4DW boundary, the number of credits is the TLP length divided by four. If the TLP length is not on the 4DW boundary, the number of credits is the length field + 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 one clock cycle to latch in the `pd_num_vc0` port and release credits.

If the TLP is a 32-bit MRd TLP or a 64-bit MRd TLP, the address needs to be read creating a completion TLP with the data. A Completion with Data (CplD) TLP 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 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 `link_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 that the lower address of the current CplD allows the far end to piece the entire read data back together.

If the TLP is a CPL TLP, once the TLP is processed the completion credits for the CPL TLP must be released to the far end. This is done using the `cplh_processed_vc0`, `cpld_processed_vc0`, and `cpld_num_vc0` ports. Each CPL TLP takes one header credit. There is one data credit used per four DWs of data. The length field provides the

number of DWs used in the TLP. If the TLP length is on the 4DW boundary, the number of credits is the TLP length divided by four. If the TLP length is not on the 4DW boundary, the number of credits is the length field + 1 (round up by 1). The number of credits used should then be placed on `cpld_num_vc0[7:0]`. Assert `cplh_processed_vc0` and `cpld_processed_vc0` for one clock cycle to latch in the `cpld_num_vc0` port and release credits.

### **As a Transmitter**

As a transmitter, the RC-Lite can send any type of TLP to the far end endpoint. 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 four. 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. A MWr TLP takes one 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 a 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.

To send Cpl TLP, the user must assemble the Cpl TLP and then check to see if the credits are available to send the TLP. The Cpl TLP without data consumes only header credit and value should be compared against the `tx_ca_cplh` port value. If `tx_ca_cplh[8]` is High, this indicates the far end has infinite credits available. The Cpl with data TLP also consumes data credits and is the length field divided by four. This value should be compared against the `tx_ca_cpld` port value. If `tx_ca_cpld[12]` is High, this indicates the far end has infinite credits available.

## Parameter Settings

The IPexpress/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 RC-Lite IP core. The parameter settings are specified using the PCI Express RC-Lite IP core Configuration GUI in IPexpress.

**Table 3-1. IP Core Parameters**

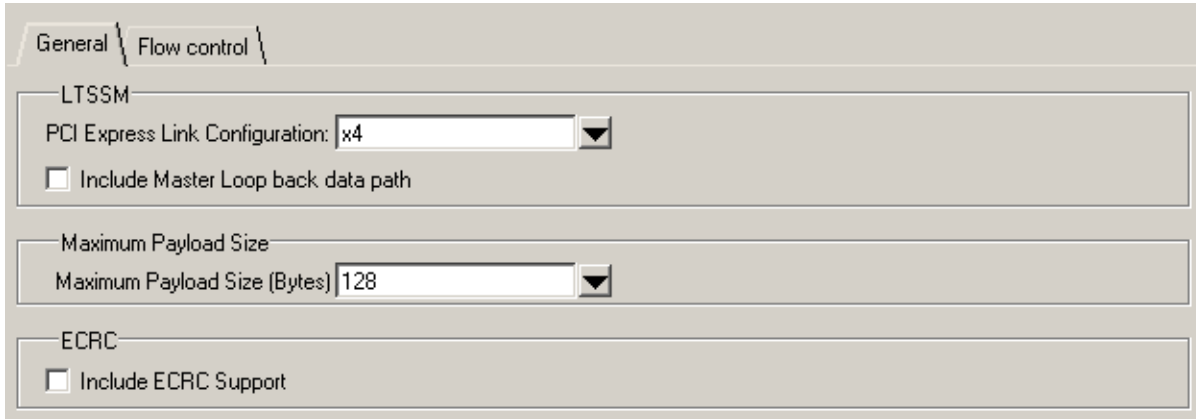
Parameters	Range/Options	Default
<b>General</b>		
PCI Express Link Configuration	x4, x1 (for LatticeECP3) x4, Downgraded x2, x1 (for ECP5)	x4
Include Master Loop back data path	Yes / No	No
Include ECRC support	Yes / No	No
Maximum Payload Size (Bytes)	128, 256, 512, 1k, 2k, 4k	128
<b>Flow Control</b>		
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 - 127	8
Number of NPD credits between UpdateFC NP	1 - 2047	255
Number of CPLH credits between UpdateFC CPL	1 - 127	8
Number of CPLD credits between UpdateFC CPL	1 - 2047	255
Worst case number of 125 MHz clock cycles between UpdateFC	3650 - 4095	4095
Infinite PH Credits	Yes / No	Yes
Initial PH credits available	1 - 127	0
Infinite PD Credits	Yes / No	Yes
Initial PD credits available	8 - 255	0
Infinite NPH Credits	Yes / No	Yes
Initial NPH credits available	1 - 127	0
Infinite NPD Credits	Yes / No	Yes
Initial NPD credits available	8 - 255	0
Infinite CPLH Credits	Yes / No	Yes
Initial CPLH credits available	1 - 127	0
Infinite CPLD Credits	Yes / No	Yes
Initial CPLD credits available	8 - 255	0

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

## General Tab

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

**Figure 3-1. PCI Express RC-Lite General Tab**



The General tab consists of the following parameters:

### PCI Express Link Configuration

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

- x4 – This is a x4 link width using a 64-bit datapath. This configuration can dynamically downgrade to a x1 link width.
- x1 – This is a x1 link width using a 16-bit datapath.

### 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.

### Maximum Payload Size

This option selects the maximum payload size to be supported in the IP core and will be used to check the length of the received packets and also to size the Retry Buffer contained in the Data Link Layer. 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 Maximum Payload Size**

Maximum payload size	LatticeECP3/ECP5 X4 EBR usage	LatticeECP3/ECP5 X1 EBR usage
128 B	9	3
256 B	9	3
512 B	9	3
1 kB	9	4
2 kB	11	8
4 kB	16	14

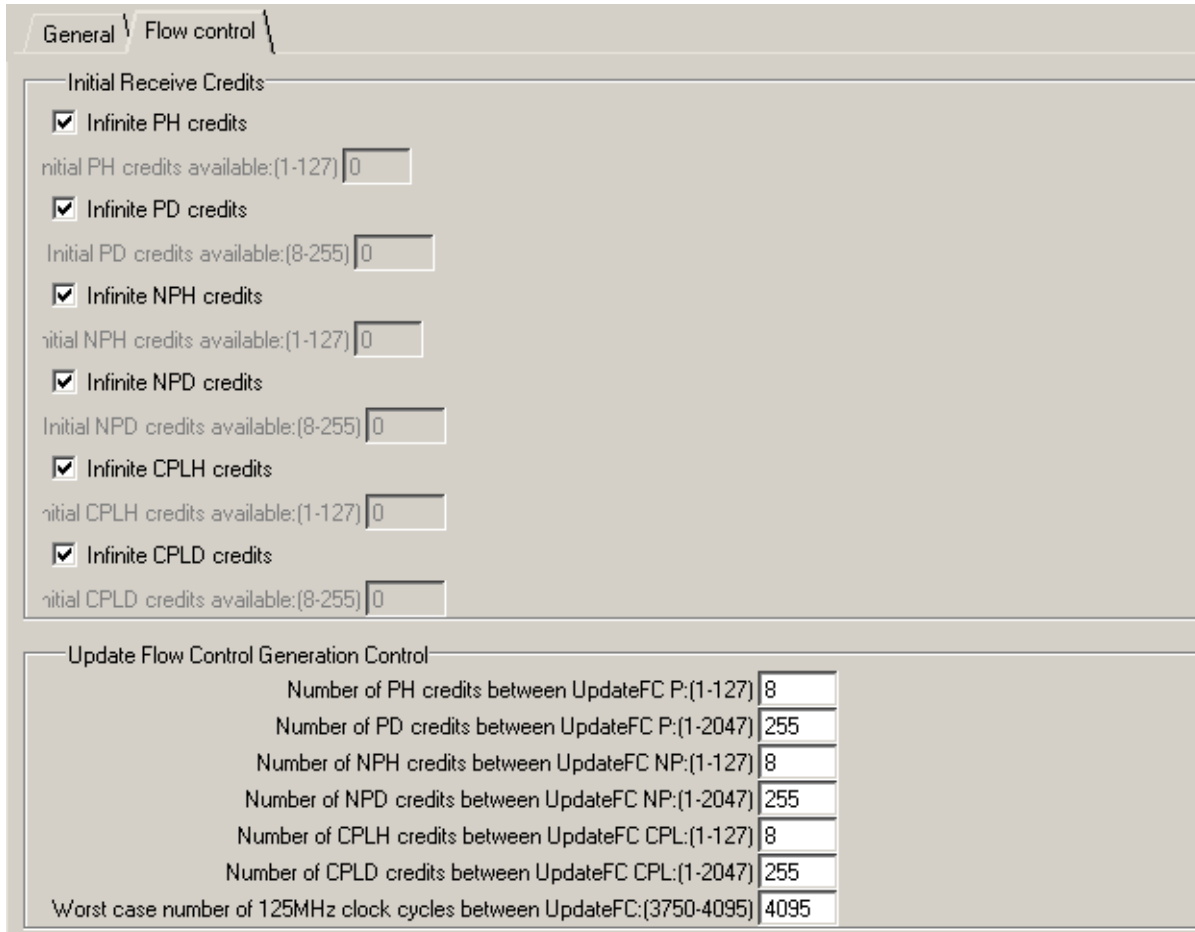
## Include ECRC Support

This option includes the ECRC generation and checking logic into the IP core. The ECRC logic is only utilized if the user enables this feature using the top level ports `ecrc_gen_enb` and `ecrc_chk_enb`. Not including this features saves nearly 1k LUTs from the core.

## Flow Control Tab

Figure 3-2 shows the contents of the Flow Control tab.

**Figure 3-2. PCI Express RC-Lite Flow Control Tab**



Initial Receive Credits	
<input checked="" type="checkbox"/> Infinite PH credits	initial PH credits available:(1-127) 0
<input checked="" type="checkbox"/> Infinite PD credits	Initial PD credits available:(8-255) 0
<input checked="" type="checkbox"/> Infinite NPH credits	initial NPH credits available:(1-127) 0
<input checked="" type="checkbox"/> Infinite NPD credits	Initial NPD credits available:(8-255) 0
<input checked="" type="checkbox"/> Infinite CPLH credits	initial CPLH credits available:(1-127) 0
<input checked="" type="checkbox"/> Infinite CPLD credits	initial CPLD credits available:(8-255) 0
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:(1-127)	8
Number of NPD credits between UpdateFC NP:(1-2047)	255
Number of CPLH credits between UpdateFC CPL:(1-127)	8
Number of CPLD credits between UpdateFC CPL:(1-2047)	255
Worst case number of 125MHz clock cycles between UpdateFC:(3750-4095)	4095

### 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.

### Infinite PH Credits

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

### Initial PH Credits Available

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



### **Infinite PD Credits**

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

### **Initial PD Credits Available**

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

### **Infinite NPH Credits**

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

### **Initial NPH Credits Available**

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

### **Infinite NPD Credits**

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

### **Initial NPD Credits Available**

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

### **Infinite CPLH Credits**

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

### **Initial CPLH Credits Available**

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

### **Infinite CPLD Credits**

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

### **Initial CPLD Credits Available**

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

## **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 device. A smaller number will increase the amount of UpdateFC DLLPs in the transmit path. But, the far end will see credits available more quickly.

## **Number of P TLPs Between UpdateFC**

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

**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.

**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.

**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.

**Number of CPL TLPs Between UpdateFC**

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

**Number of CPLD TLPs Between UpdateFC**

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

**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.

This chapter provides information on licensing the PCI Express RC-Lite IP core, generating the core using the Diamond software IPexpress tool, running functional simulation, and including the core in a top-level design.

The Lattice PCI Express RC-Lite IP core can be used in ECP5 and LatticeECP3 device families.

## Licensing the IP Core

An IP license is required to enable full, unrestricted use of the PCI Express RC-Lite IP core in a complete, top-level design. The specific PCI Express RC-Lite IP core licensing requirements are different for targeting the ECP5 and LatticeECP3 families.

### Licensing Requirements for ECP5/LatticeECP3

An IP license that specifies the IP core (PCI Express RC-Lite IP), device family (ECP5 or ECP3) and configuration (x1 or x4) is required to enable full use of the PCI Express RC-Lite IP core in ECP5 or LatticeECP3 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 RC-Lite IP core for ECP5 and LatticeECP3 and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The PCI Express RC-Lite IP core for ECP5 and Lattice ECP3 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 bitstreams that do not include the hardware evaluation timeout limitation.

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

## IPexpress Flow for LatticeECP3 Devices

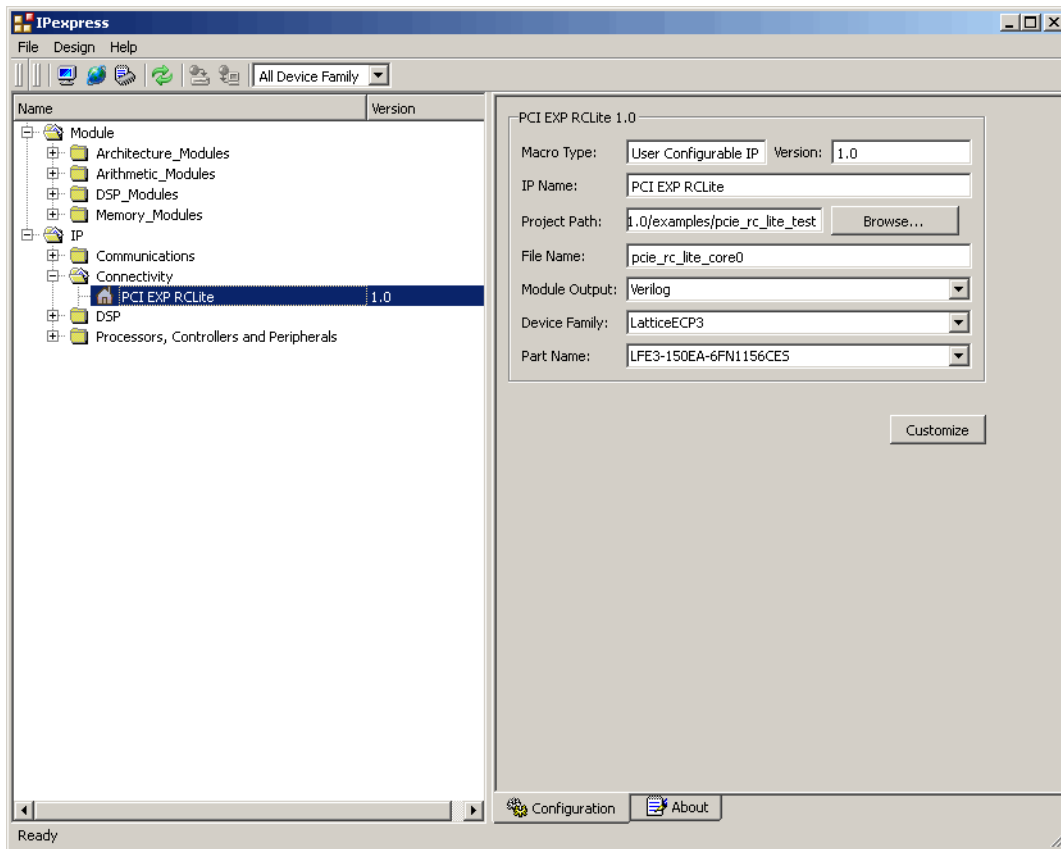
### Getting Started

The PCI Express RC-Lite 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.

The IPexpress tool GUI dialog box for the PCI Express RC-Lite IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **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. ECP5, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

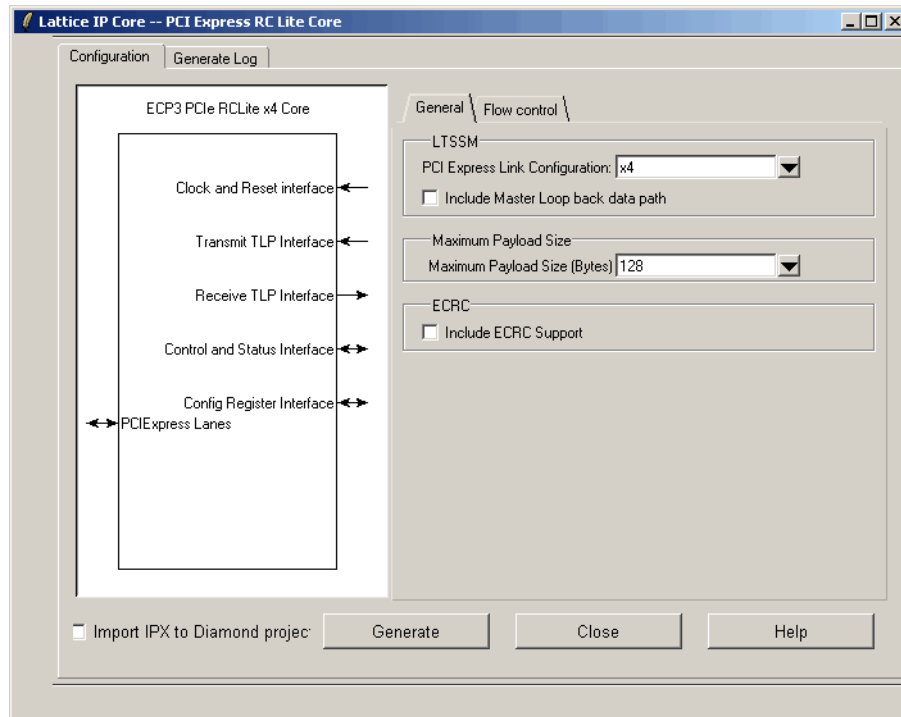
Figure 4-1. IPexpress Dialog Box (Diamond Version)



Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output (Design Entry in Diamond), 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, the user clicks the **Customize** button in the IPexpress tool dialog box to display the PCI Express RC-Lite 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 the [Parameter Settings](#) section for more information on the PCI Express RC-Lite IP core parameter settings. Additional information and known issues about the PCI Express RC-Lite IP core are provided in a ReadMe document that may be opened by clicking on the **Help** button in the Configuration GUI.

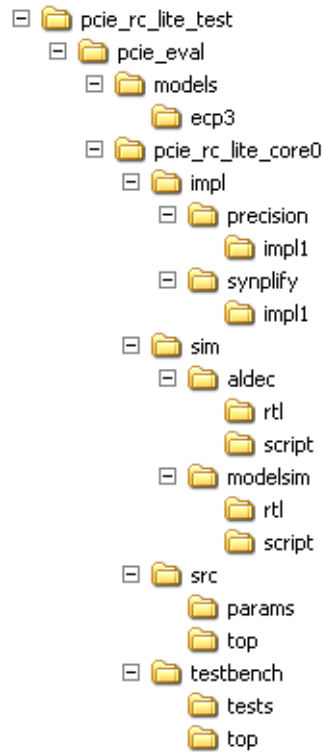
Figure 4-2. Configuration GUI (Diamond Version)



### 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.

**Figure 4-3. LatticeECP3 PCI Express RC-Lite IP 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 created files 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 pci_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 RC-Lite 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 RC-Lite IP core evaluation. The `\pcie_eval` directory contains files/folders with content that is constant for all configurations of the PCI Express RC-Lite IP core. The `\<username>` subfolder (`\pcie_rc_lite_core0` in this example) contains files/folders with content specific to the `<username>` configuration.

The PCI Express RC-Lite IP core ReadMe document is also provided in the `\pcie_eval` directory.

For example information and known issues on this core, see the Lattice PCI Express RC-Lite IP core 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 and ECP5).

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 eval design. The `\testbench` directory provides a top-level testbench and test case files.

## Running Functional Simulation

Simulation support for the PCI Express RC-Lite IP core is provided for Aldec and ModelSim simulators. The PCI Express RC-Lite IP core simulation model is generated from the IPexpress tool with the name `<username>.v`. This file calls `<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. VHDL users will use the same Verilog model for simulation.

When compiling the PCI Express RC-Lite 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`. Users 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`. Users 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.

## Synthesizing and Implementing the Core in a Top-Level Design

The PCI Express RC-Lite 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 and then synthesizing the entire design with Synplify RTL Synthesis.

The top-level file `pcie_core0_eval_top.v` provided in `\<project_dir>\pcie_eval\<username>\src\top` supports the ability to implement the PCI Express RC-Lite IP core in isolation. Push-button implementation of this top-level design with Synplify RTL Synthesis is supported via the Diamond software project files `<username>_eval.lfd` located in the `\<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>.lfd`. 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.



## Hardware Evaluation

The PCI Express RC-Lite 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.

### 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.

## 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.

### 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 you wish to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the **Target** box.
4. If you want 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 module's dialog box 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 stand-alone mode).
8. Click **Generate**.
9. Check the Generate Log tab to check for warnings and error messages.
10. Click **Close**.

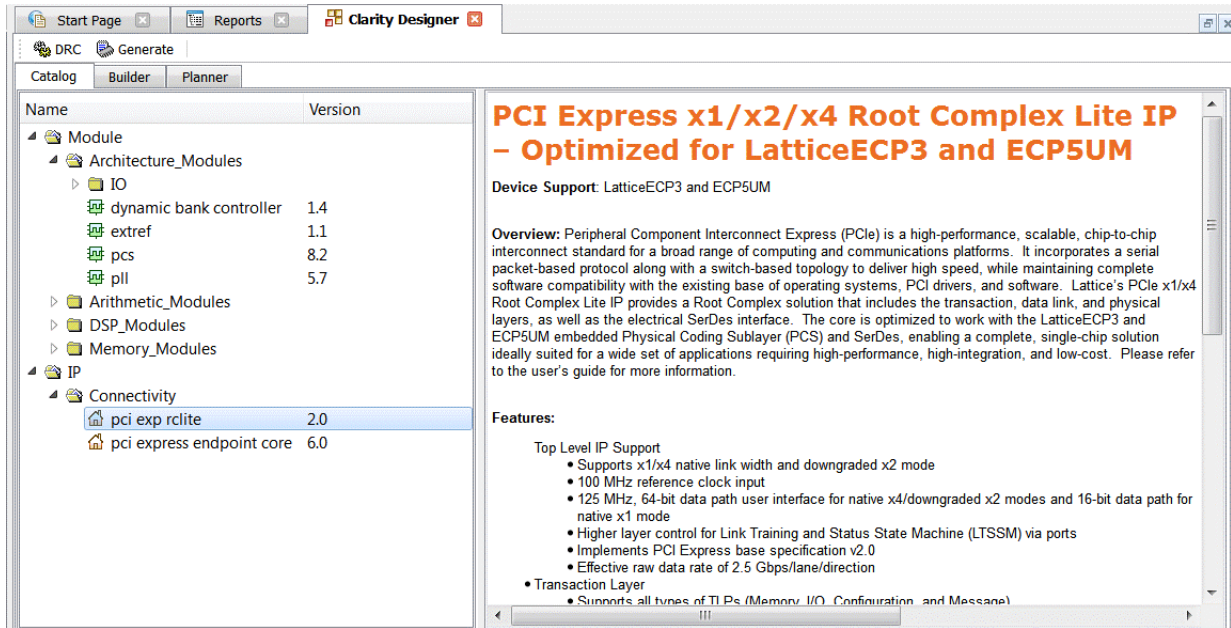
The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change 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.

## Clarity Designer Flow for ECP5 Devices

### Getting Started

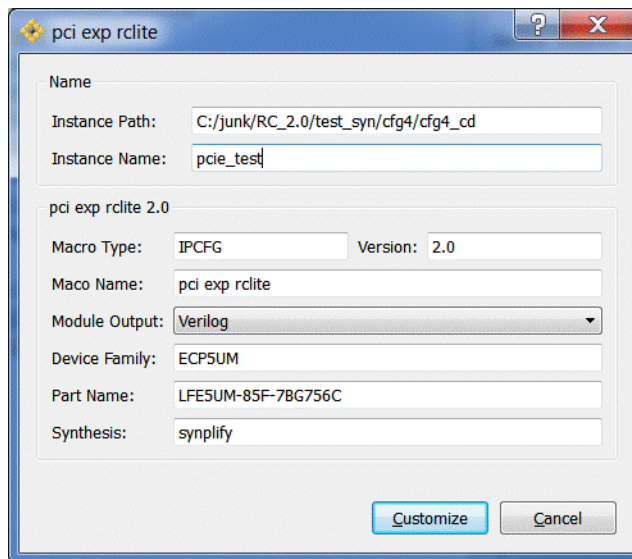
The PCI Express IP core is available for download from the Lattice IP Server using the Diamond Clarity Designer tool for ECP5 devices. The IP files are automatically installed using InstallShield® technology in any customer-specified directory. After the IP core has been installed, the IP core is listed in the Available Modules tab of the Clarity Designer GUI as shown in Figure 4-4.

Figure 4-4. Clarity Designer GUI (pci express RC-Lite core)



In the Catalog tab, double clicking the PCI Express endpoint entry opens the PCI Express GUI dialog box shown in Figure 4-5.

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.

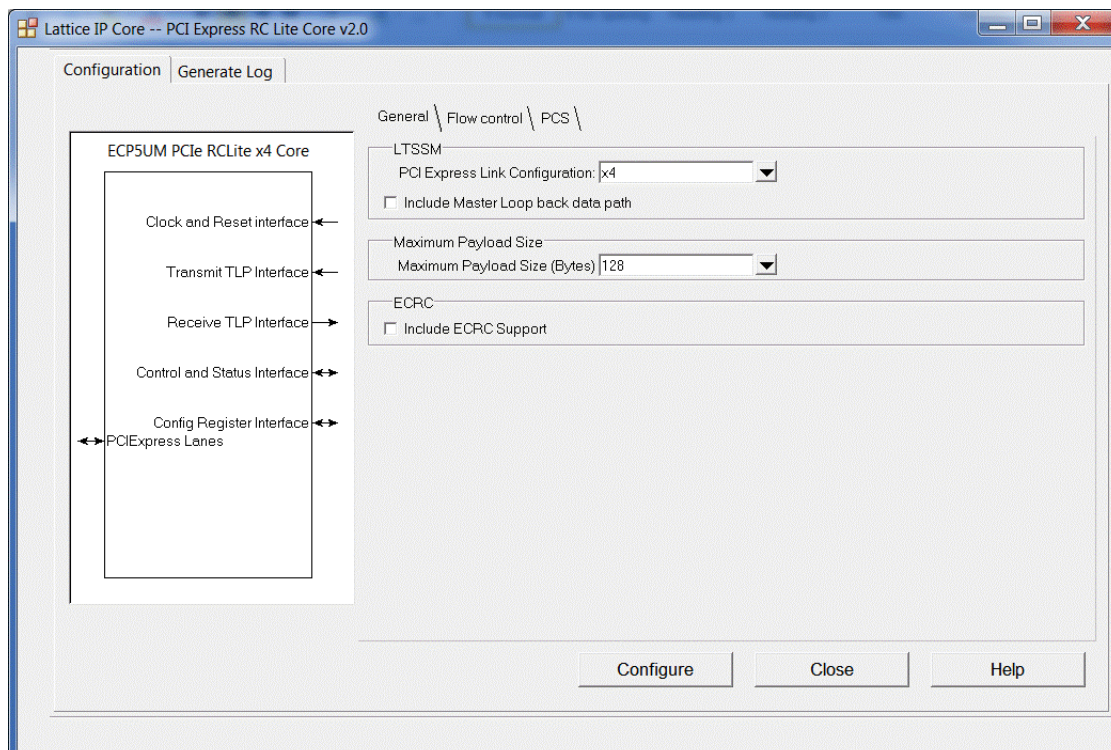
To generate a specific IP core configuration the user specifies:

- **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.

**Figure 4-6. PCI Express RC-Lite IP Core Configuration GUI**

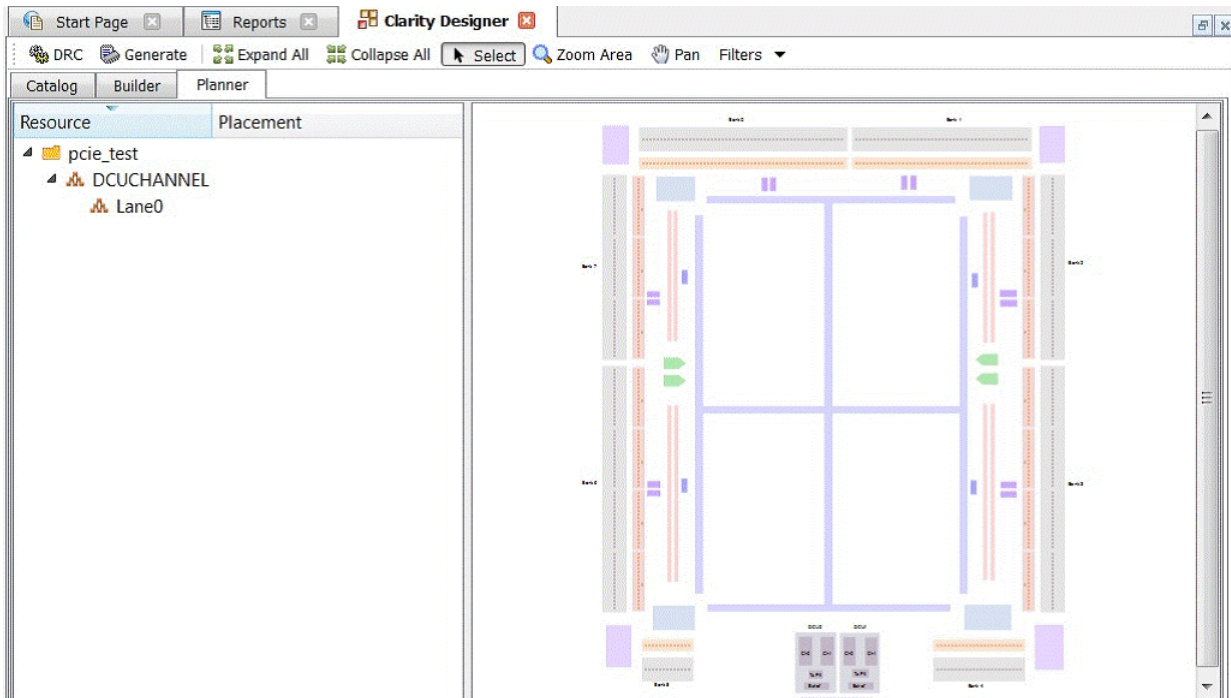


## Configuring and Placing the IP Core

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.

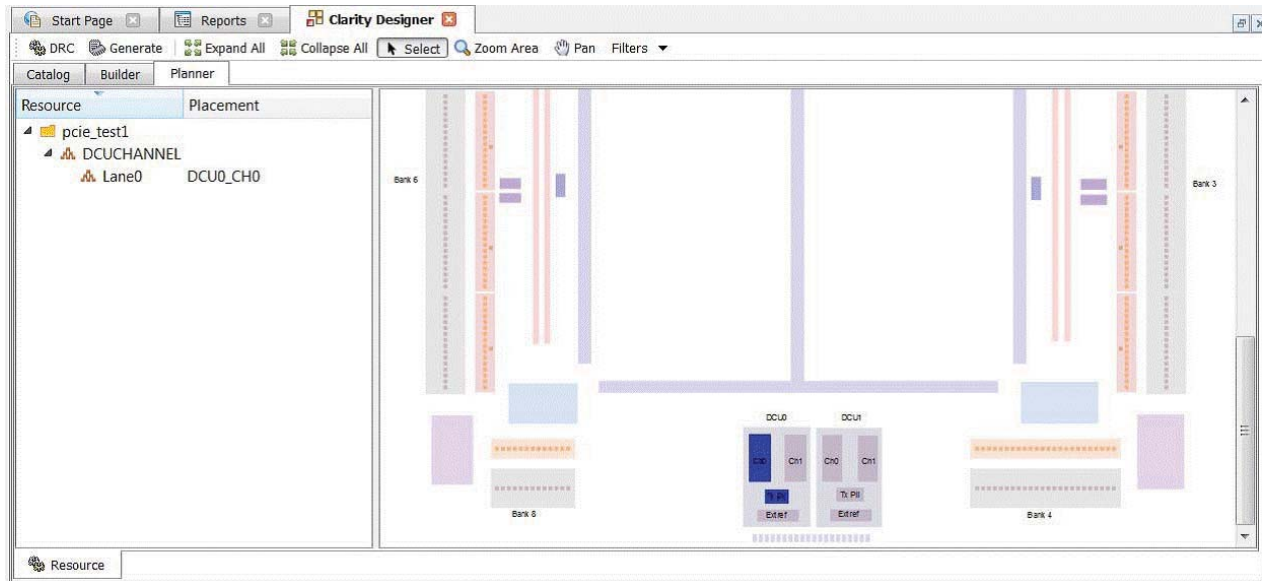
**Figure 4-7. PCI Express RC-Lite IP Core Clarity Designer GUI**



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.



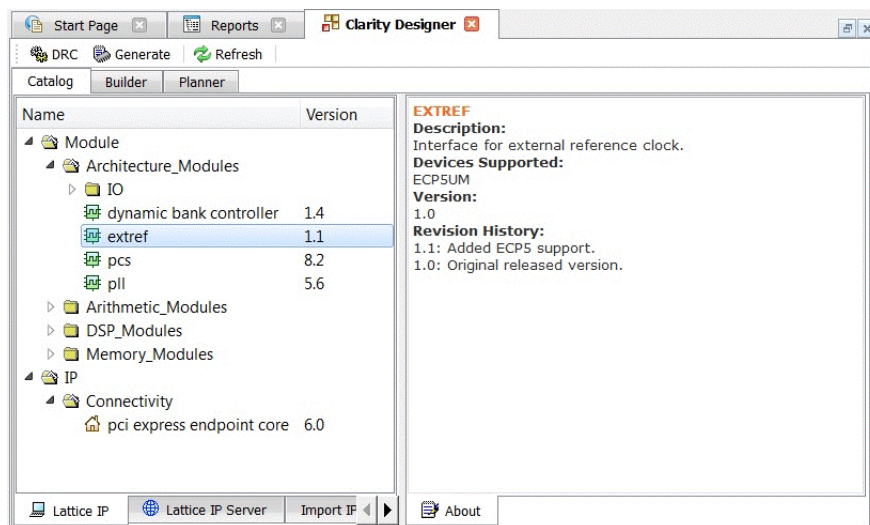
**Figure 4-8. Clarity Designer Placed Module**



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.

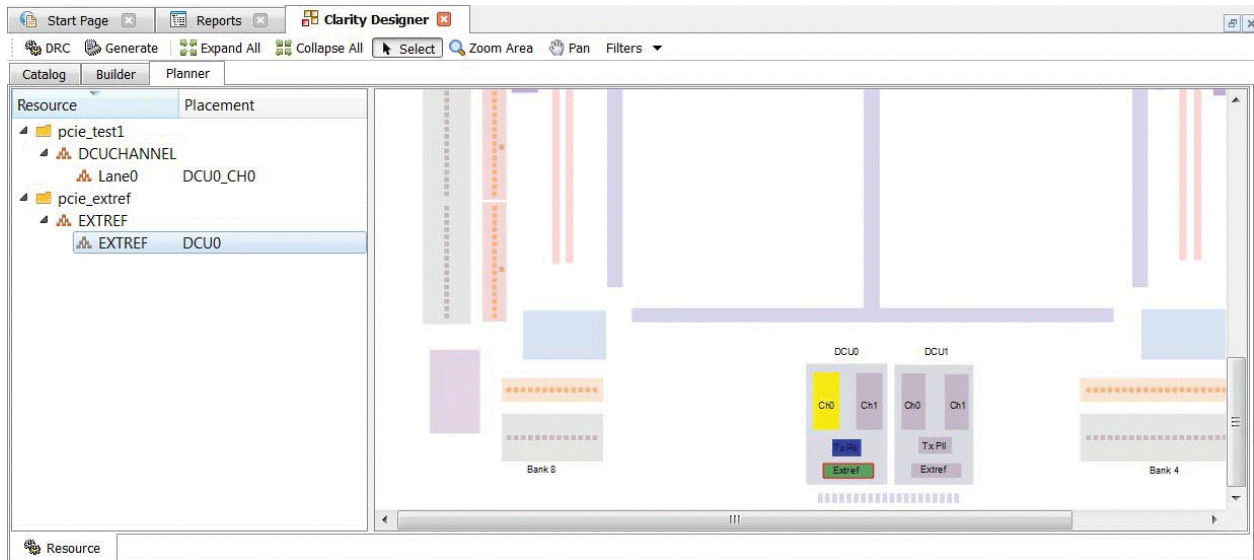
Note also that PCI express endpoint IP needs an EXTREF module to be connected for reference clocks which can be shared across multiple IPs. So it has to be generated outside the PCI express endpoint IP in the Clarity Designer tool and connected. Similar flow as the generation of PCI express endpoint core can be followed to generate extref module from the catalog tab as shown in the following Figure 4-9.

**Figure 4-9. Clarity Designer GUI (extref)**



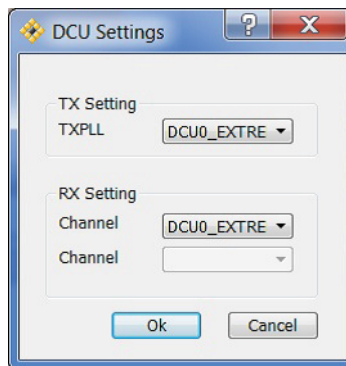
Both PCI RC-Lite endpoint core and extref instances can be dragged and dropped from the Planner tab to the Clarity Designer Placement tab. Once placed, the specific IP core placement is shown in the Configured Modules tab and highlighted in the Placement tab as shown in Figure 4-10.

**Figure 4-10. Clarity Designer Placed Modules (pci express RC-Lite and extref)**



3. After placing both PCI RC-Lite 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.

**Figure 4-11. Clarity Designer DCU Settings**

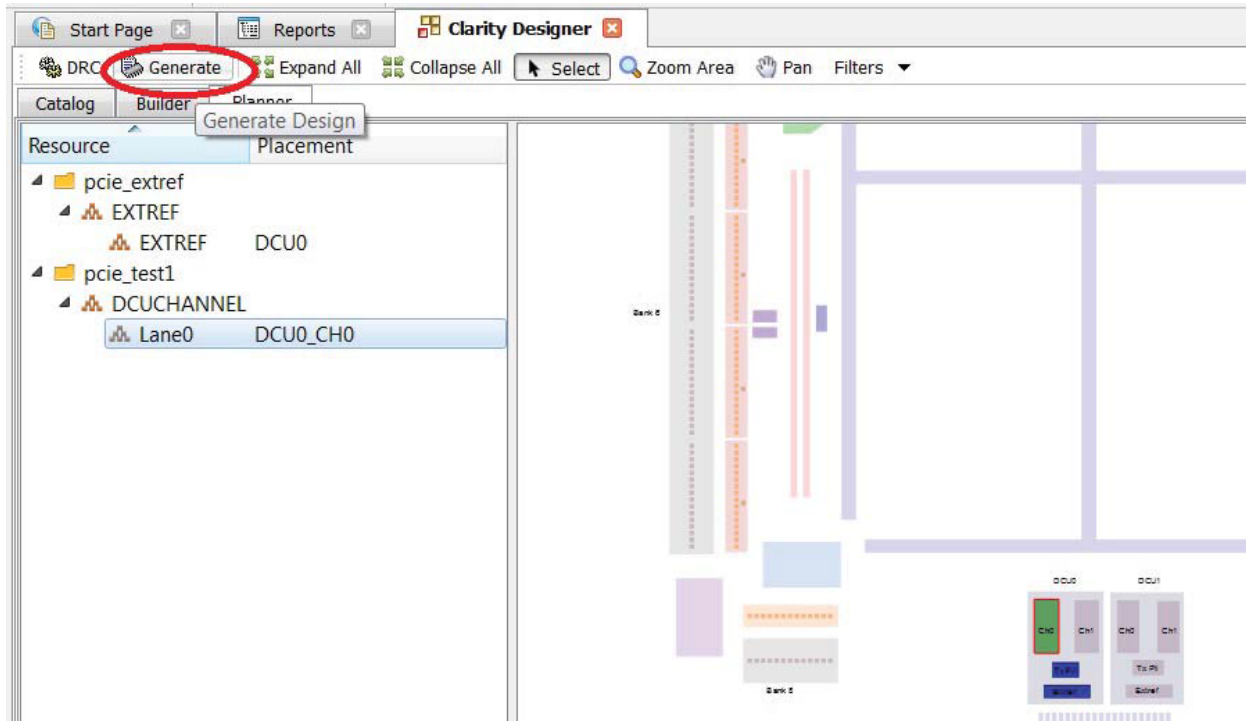


4. Select **DCU0\_EXTREF** (**DCU1\_EXTREF** if **DCU1**) as source for TXPLL and channel.
5. Click **OK**.

## 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.

**Figure 4-12. Generating the IP Core**



### Clarity Designer-Created Files and Directory Structure

The directory structure of the files created when the PCI express RC-Lite 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

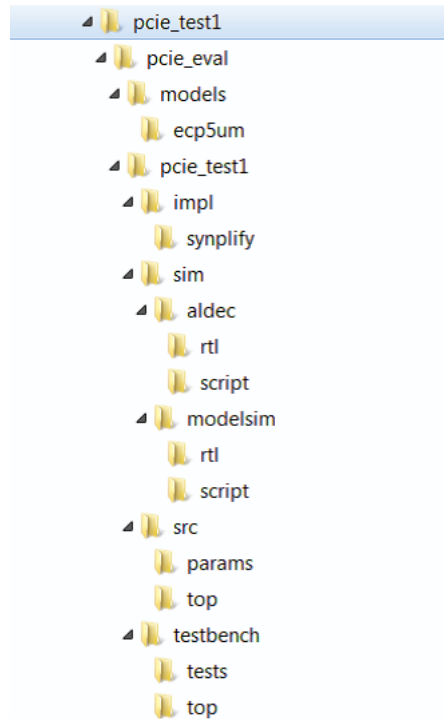


Table 4-2. File List

File	Sim	Synthesis	Description
<username>.v	Yes		This file provides the PCI Express core for simulation. This file provides 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 soft-ware. 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.

## Instantiating the Core

The generated PCI express RC-Lite core package includes black-box (`<username>_core_bb.v`, `<username>_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.

## 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."*

## Synthesizing and Implementing the Core in a Top-Level Design

The PCI Express RC-Lite 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 the [Instantiating the Core](#) section and then synthesizing the entire design with either Synplify. 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 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.

## 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.

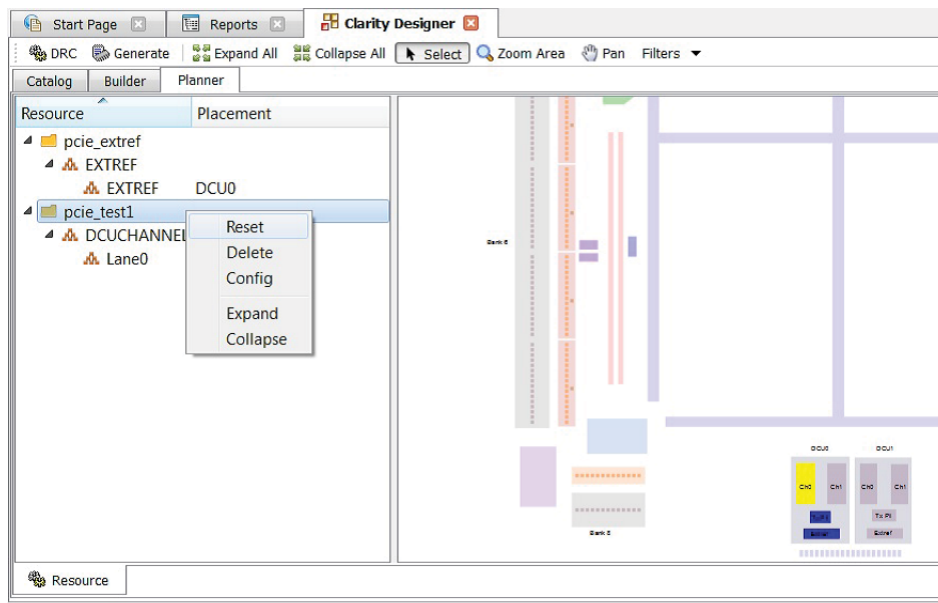
To enable hardware evaluation in Diamond, choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be Enabled/Disabled in the Strategy dialog box. It is enabled by default.

### 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.

**Figure 4-14. Reset, Delete, Config, Expand and Collapse Placement of the IP Core**



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

## Simulation and Verification

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

### Simulation Strategies

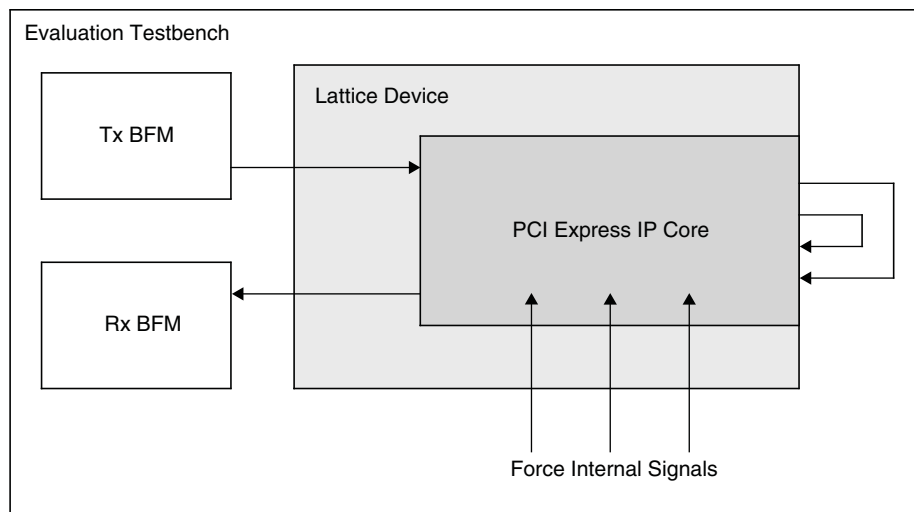
Included with the core from the IPexpress tool is the evaluation testbench located in the <username> directory. The intent of the evaluation testbench is to show the core performing in simulation, as well as to provide timing simulations post place and route. Many communication cores work in a loopback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

In a real system, however, PCI Express RC-Lite IP core requires that an upstream port connect to a downstream port. In the simple-to-use, Lattice-supplied eval testbench, 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 the transmit and receive interface. This is the extent of the evaluation testbench.

Figure 5-1 illustrates the evaluation testbench process.

**Figure 5-1. PCI Express RC-Lite IP 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 RC-Lite IP 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.

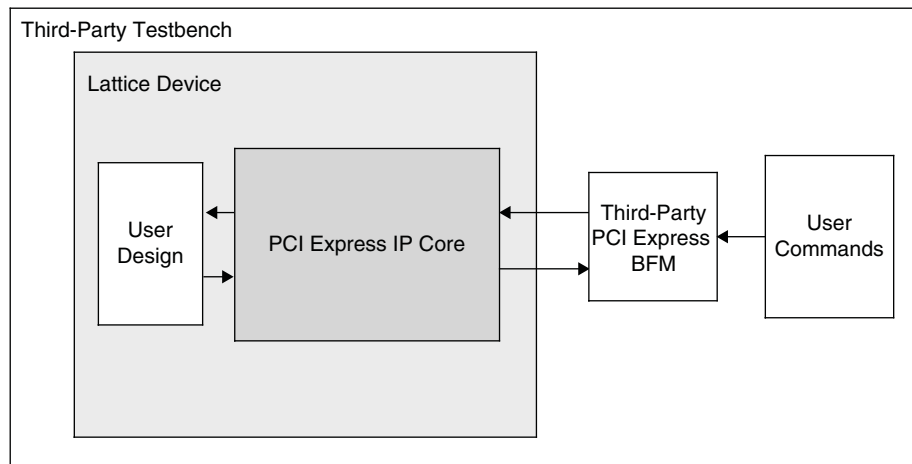
Users simulating a multi-lane core at the serial level should give consideration to lane ordering. Lane ordering is dependant on the layout of the chip on a board. Refer to the [Board Layout Concerns](#) section for further information.

### 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 BFM 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 RC-Lite IP core, as shown in Figure 5-2. There are other third party vendors for PCI Express RC-Lite IP core including Denali® and Cadence.

**Figure 5-2. PCI Express RC-Lite IP 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.

## FPGA Design Implementation

This section provides information on implementing the PCI Express RC-Lite 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.

### Setting Up the Core

This section describes how to set up the PCI Express RC-Lite IP core for various link width combinations. The user must provide 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.

Lane flipping is not applicable for x1. The user can select which channel of the quad to be the active channel in the IPexpress tool.

### Setting Up for 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.

### Setting Up for x4 (Flipped)

No changes required. Simply use the pcs\_pcie\_8b\_x4.txt file generated from the IPexpress tool.

### 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.

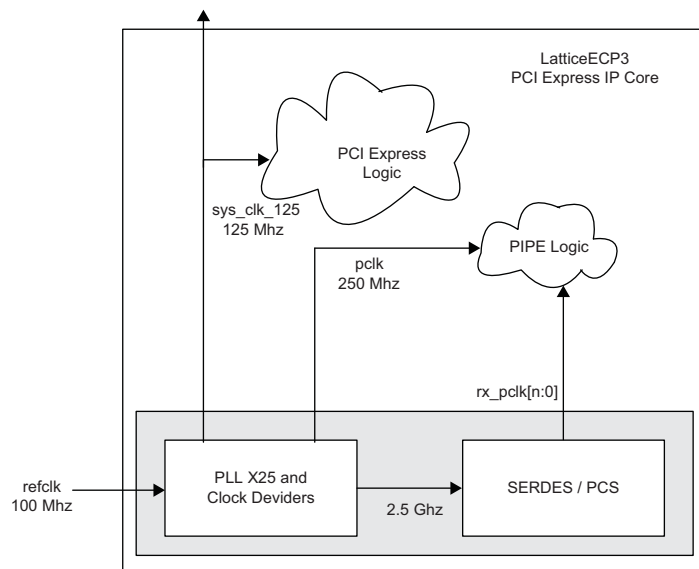
Refer to the .lpf file at the `\<project_dir>\pcie_eval\<username>\impl\synplify` directory for design constraints required by the IP.

### 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 RC-Lite IP core is clocked using a 125 MHz clock (sys\_clk\_125).

Figure 5-3 provides the internal clocking structures of the IP core in the LatticeECP3 and ECP5 families.

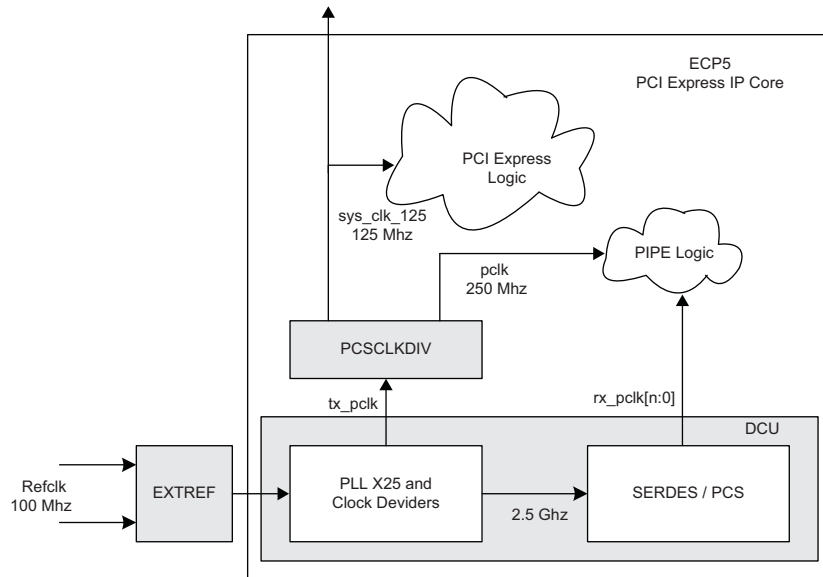
**Figure 5-3. LatticeECP3 PCI Express Clocking Scheme**



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 RC-Lite IP core then performs a clock domain change to the sys\_clk\_125 125 MHz clock for the user interface.

Figure 5-4. 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  $\Omega$  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.

### Locating the IP

The PCI Express RC-Lite IP 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 5-5 provides a block diagram with placement positions of the PCS/SERDES quads in the LatticeECP3 devices.

Figure 5-5. LatticeECP3 Device Arrays with PCS/SERDES

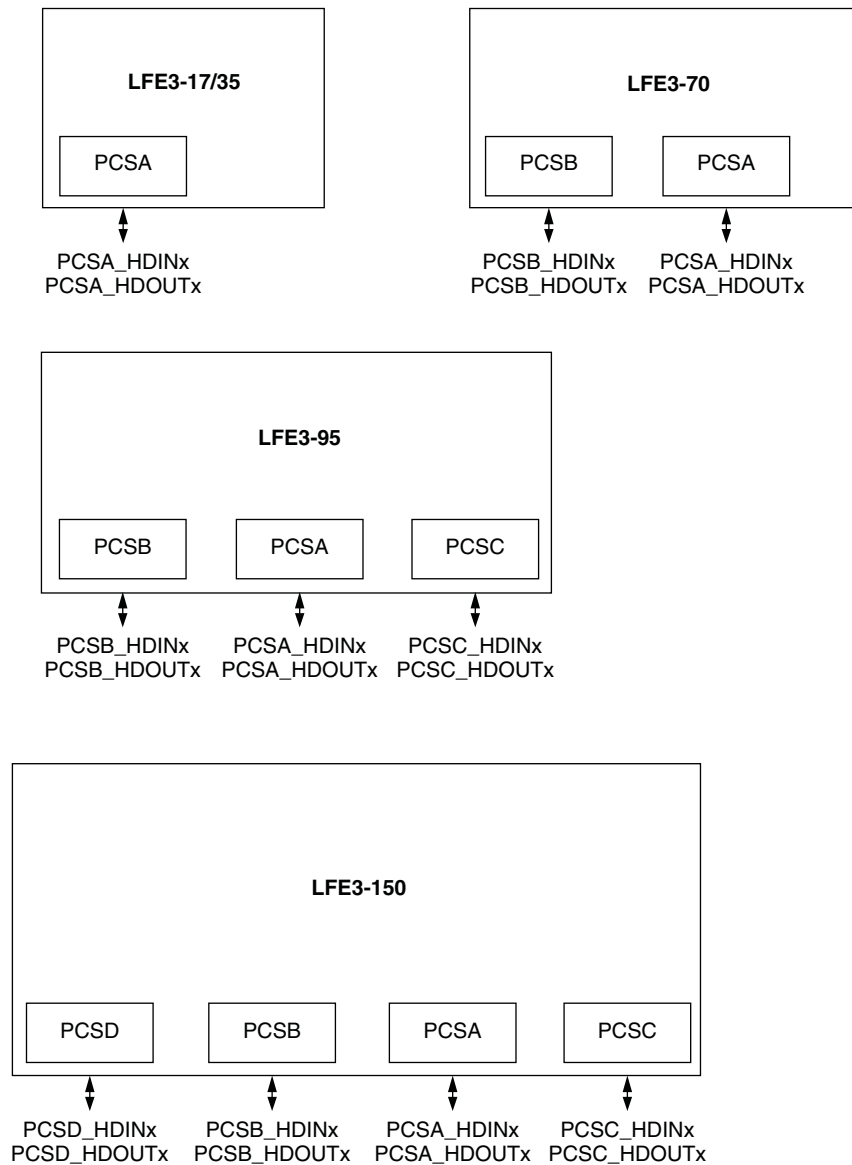
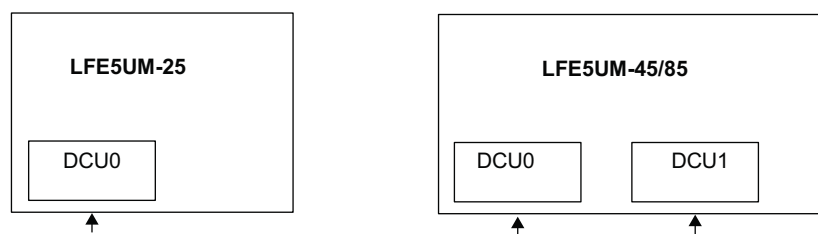


Figure 5-6 provides a block diagram with placement positions of the PCS/SERDES quads in the ECP5 devices.

Figure 5-6. ECP5 Device Arrays with PCS/SERDES



## Board-Level Implementation Information

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

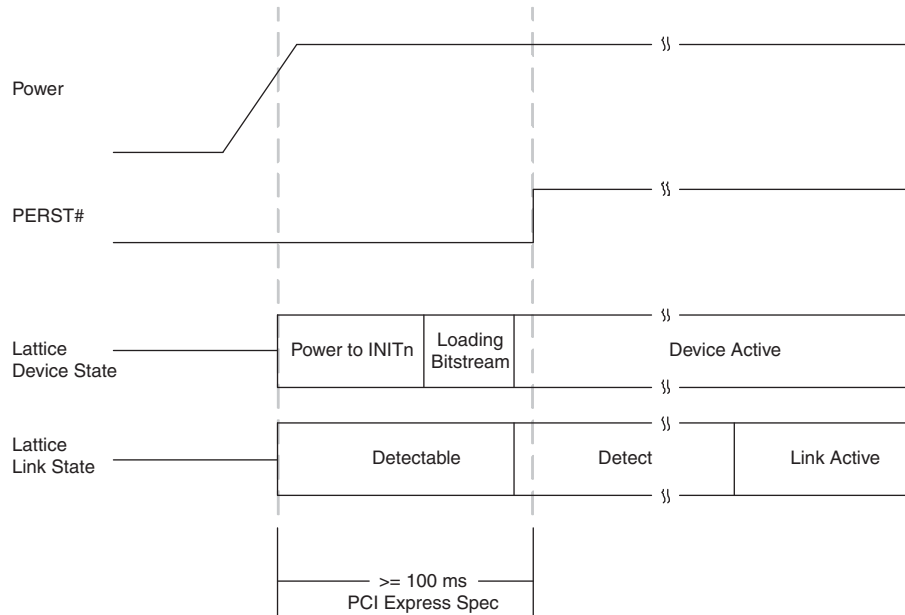


## PCI Express Power-Up

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 5-7 is a best case timing diagram of the Lattice device with respect to PERST#.

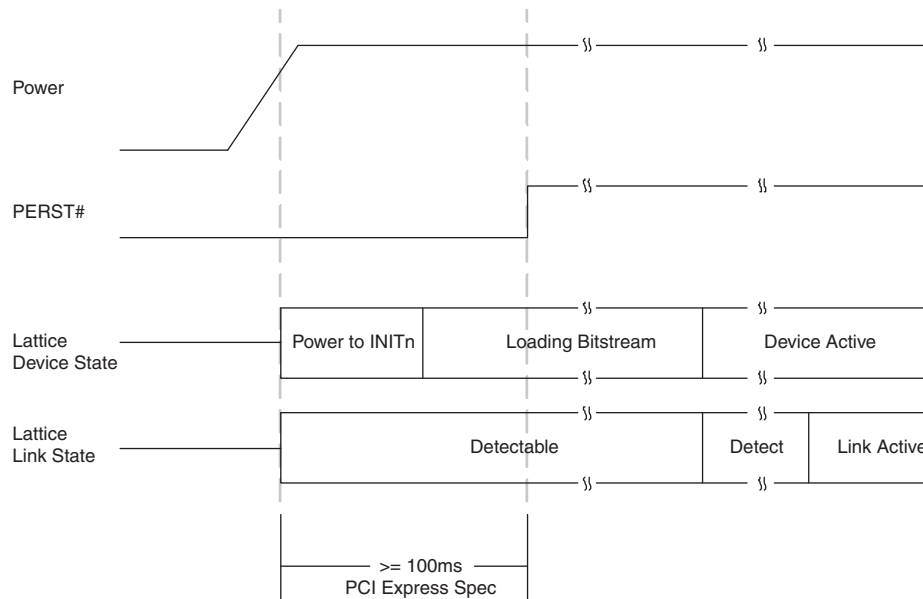
**Figure 5-7. Best Case Timing Diagram, Lattice Device with Respect to PERST#**



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 5-8 shows a worst case timing diagram of the Lattice device with respect to PERST#.

**Figure 5-8. Worst Case Timing Diagram, Lattice Device with Respect to PERST#**



If the Lattice device does not finish loading the bitstream until after the release of PERST#, then the link will still be established. The Lattice device turns on the 100  $\Omega$  differential resistor on the receiver data lines when power is applied. This 100  $\Omega$  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 contains the relative numbers for the LatticeECP3 and ECP5 families.

**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) <sup>1</sup>	17	31	85	85	135	ms

1. 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) <sup>1</sup>	20	36	69	ms

1. 8-bit wide Flash and external CPLD interfacing to ECP5 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.

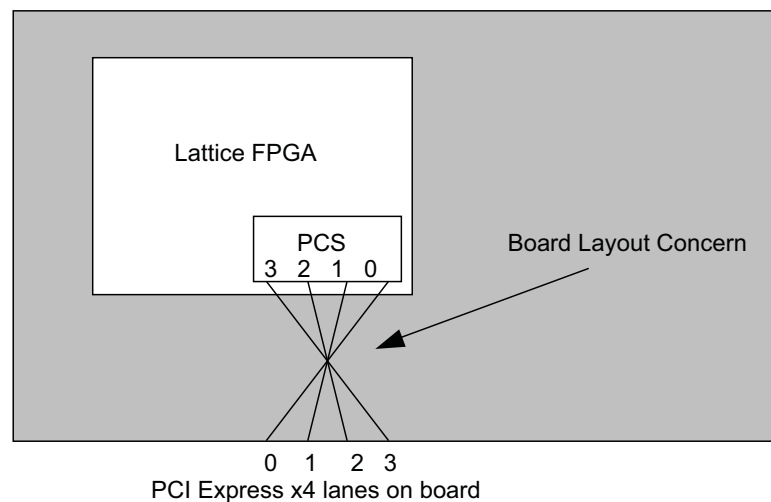
To reduce the bitstream loading time of the Lattice device a parallel Flash device and CPLD device can be used. The use of parallel Flash devices and Lattice devices is documented in AN8077, [Parallel Flash Programming and FPGA Configuration](#).

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

### Board Layout Concerns

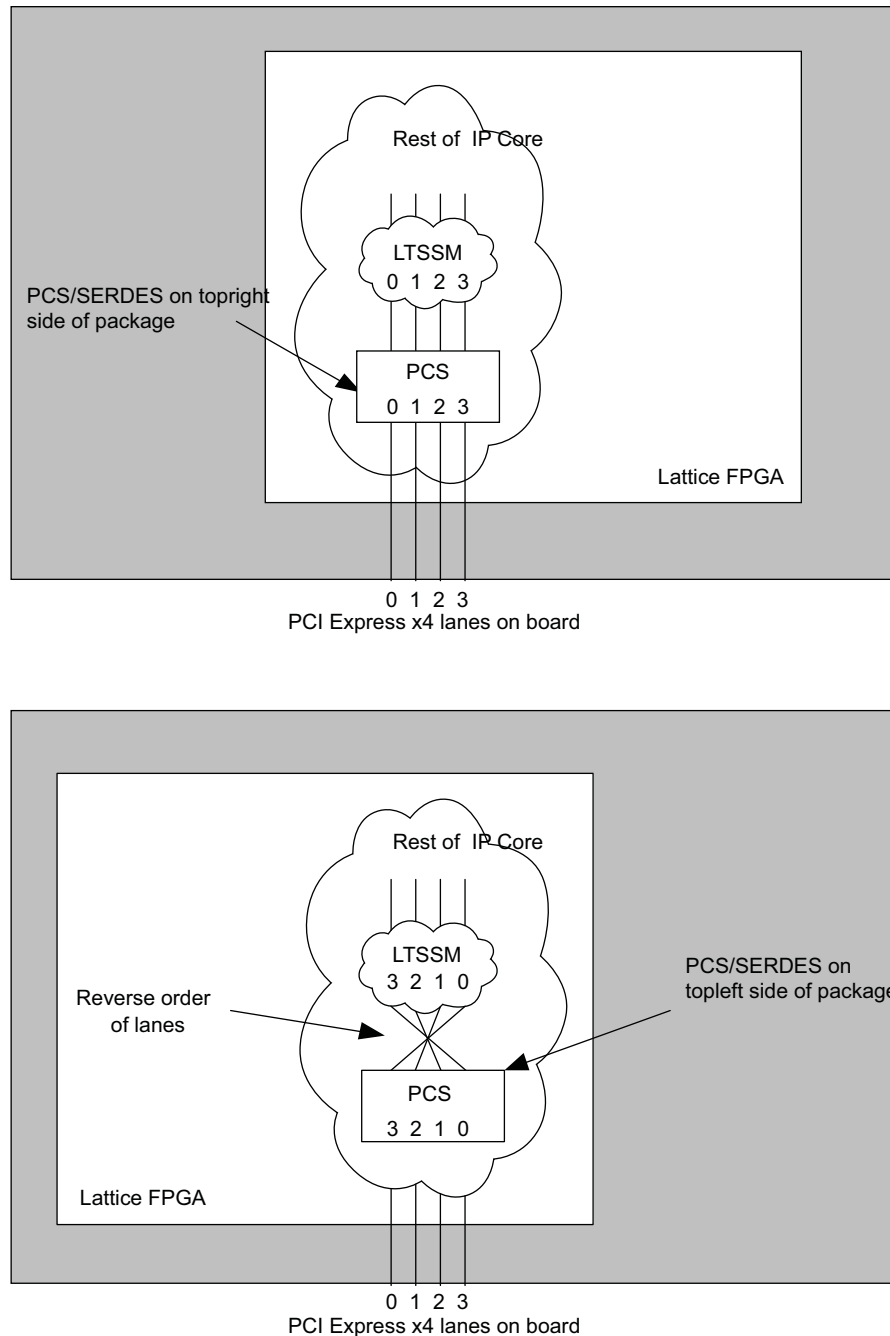
For multi-lane implementations there might be a layout concern in making the connection on board for a particular orientation of lanes. On some packages lane 0 of board will align with lane 0 of the SERDES and likewise for channels 1, 2 and 3. However, in other packages lane 0 of board will need to cross 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 5-9 provides an example of the board layout concern.

**Figure 5-9. 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 RC-Lite IP core. This allows the board layout to connect board lane 0 to SERDES lane 3, board lane 1 to SERDES lane 2, board lane 2 to SERDES lane 1, and board lane 3 to SERDES lane 0. The PCI Express RC-Lite IP core will then perform a reverse order connection so the PCI Express board 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 RC-Lite IP 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 5-10 provides a diagram of a normal and a reversed IP core implementation.

Figure 5-10. Implementation of x4 IP Core to Edge Fingers



As shown in Figure 5-10, 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.

### LatticeECP3 and ECP5 IP Simulation

The PCI Express IP simulation uses the PIPE module. This simulation model is found in the `pcie_eval/<user-name>/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>`.

### Simulation Behavior

When setting the SIMULATE variable for the simulation model of the PCI Express RC-Lite IP 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)

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

- Simulation environment verifying proper PCI Express Root complex functionality when testing with a Synopsys DesignWare behavioral model in Endpoint mode.
- 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.



## Core Verification

---

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

Simulation environment verifying proper PCI Express Root complex functionality when testing with a Synopsys DesignWare behavioral model in Endpoint mode.

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.



# Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

## Lattice Technical Support

Submit a technical support case via [www.latticesemi.com/techsupport](http://www.latticesemi.com/techsupport).

## PCIe Solutions Web Site

Lattice provides customers with low-cost and low-power programmable PCIe solutions that are ready to use right out of the box. A full suite of tested and interoperable PCIe solutions is available that includes development kits with evaluation boards and hardware and software reference designs. These solutions are valuable resources to jump start PCIe applications from a board design and FPGA design perspective. For more information on Lattice's PCIe solutions, visit:

<http://www.latticesemi.com/solutions/technologysolutions/pciexpresssolutions.cfm?source=topnav>

## PCI-SIG Website

The Peripheral Component Interconnect Special Interest Group (PCI-SIG) website contains specifications and documents referred to in this user's guide. The PCI-SIG URL is:

<http://www.pcisig.com>.

## References

### ECP5

- DS1044, [ECP5 Family Data Sheet](#)
- TN1261, [ECP5 SERDES/PCS Usage Guide](#)

### LatticeECP3

- DS1021, [LatticeECP3 Family Data Sheet](#)

## Revision History

Date	Document Version	IP Core Version	Change Summary
April 2015	1.0	1.0	Initial release.

# Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the PCI Express RC-Lite IP core.

## Configuration

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/software](http://www.latticesemi.com/software).

## ECP5 Utilization (x4 RC-Lite)

Table A-1 shows the resource utilization for the PCI Express x4 RC-Lite IP core implemented in an ECP5 FPGA.

**Table A-1. Resource Utilization<sup>1</sup>**

IPexpress Configuration <sup>1</sup>	Slices	LUTs	Registers	sysMEM EBRs
Config 1	8021	11907	8199	9

1. Performance and utilization data are generated targeting an LFE5UM-85E-7MG756C using Lattice Diamond 3.4 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 RC-Lite IP core targeting ECP5 devices is PCI-ERC4-E5-U1.

## LatticeECP3 Utilization (x4 RC-Lite)

Table A-2 shows the resource utilization for the PCI Express x4 RC-Lite IP core implemented in a LatticeECP3 FPGA.

**Table A-2. Resource Utilization<sup>1</sup>**

IPexpress Configuration <sup>1</sup>	Slices	LUTs	Registers	sysMEM EBRs
Config 1	7552	10537	8460	9

1. Performance and utilization data are generated targeting an LFE3-95E-7FN1156CES using Lattice Diamond 4.4 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 x4 RC-Lite IP core targeting LatticeECP3 devices is PCI-ERC4-E3-U1.



## ECP5 Utilization (x1 RC-Lite)

Table A-3 shows the resource utilization for the PCI Express x1 RC-Lite IP core implemented in an ECP5 FPGA.

**Table A-3. Resource Utilization<sup>1</sup>**

IPexpress Configuration <sup>1</sup>	Slices	LUTs	Registers	sysMEM EBRs
Config 1	3207	4649	3186	3

1. Performance and utilization data are generated targeting an LFE5UM-85E-7MG756C using Lattice Diamond 3.4 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 RC-Lite IP core targeting LatticeECP5 devices is PCI-ERC1-E5-U1

## LatticeECP3 Utilization (x1 RC-Lite)

Table A-2 shows the resource utilization for the PCI Express x1 RC-Lite IP core implemented in a LatticeECP3 FPGA.

**Table A-4. Resource Utilization<sup>1</sup>**

IPexpress Configuration <sup>1</sup>	Slices	LUTs	Registers	sysMEM EBRs
Config 1	3092	4521	3177	3

1. Performance and utilization data are generated targeting an LLFE3-95E-7FN1156CES using Lattice Diamond 3.4 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 RC-Lite IP core targeting LatticeECP3 devices is PCI-ERC1-E3-U1.