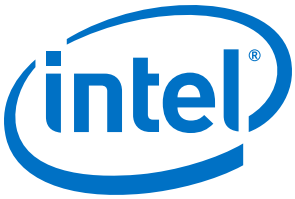


Intel® FPGA P-Tile Avalon® Streaming (Avalon-ST) IP for PCI Express* User Guide

Updated for Intel® Quartus® Prime Design Suite: **19.4**

IP Version: **1.1.0**

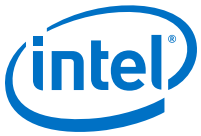
Subscribe

Send Feedback

UG-20225 | 2020.01.16

Latest document on the web: **PDF** | **HTML**

Contents



1. Introduction

1.1. Overview

P-Tile is an FPGA companion tile die that supports PCI Express* Gen4 in Endpoint, Root Port and TLP Bypass modes.

It serves as a companion tile for both Intel® Stratix® 10 DX and Intel Agilex™ devices.

P-Tile natively supports Gen3 and Gen4 configurations in accordance with the *PCI Express Base Specification 4.0 Rev 1.0*.

1.2. Features

The P-tile Avalon®-ST and Single Root I/O Virtualization (SR-IOV) IP for PCI Express supports the following features:

PCIe* Features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as a Hard IP.
- Natively supports Gen4 x16, Gen4 x8, Gen3 x16, Gen3 x8 for Endpoint mode. Gen4 x4/x2/x1, Gen3 x4/x2/x1 and all Gen2 and Gen1 configurations in Endpoint mode are not supported natively, but can be achieved through link negotiation.
- Natively supports Gen4 x16, Gen4 x4, Gen3 x16, Gen3 x4 for Root Port mode. Gen4 x8/x2/x1, Gen3 x8/x2/x1 and all Gen2 and Gen1 configurations in Root Port mode are not supported natively, but can be achieved through link negotiation.
- Static port bifurcation (four x4s Root Port, two x8s Endpoint).
- Supports TLP Bypass mode.
  - Supports one x16, two x8, or four x4 interfaces.
  - Supports upstream/downstream TLP bypass mode.
- Supports up to 512B maximum payload.
- Single Virtual Channel (VC).
- Latency Tolerance Reporting (LTR).
- Page Request Services (PRS).

  Note: In the Intel Quartus® Prime 19.4 release, only PF0 supports PRS. Furthermore, in this release, the PRS interface only has Compilation and Simulation support.
- Completion Timeout Ranges.
- Atomic Operations (FetchAdd/Swap/CAS).

- Extended Tag Support.
  - 10-bit Tag Support (Port 0 x16 Controller only. Maximum 512 Outstanding NPRs)
- Separate Refclk with Independent Spread Spectrum Clocking (SRIS).
- Separate Refclk with no Spread Spectrum Clocking (SRNS).
- Common Refclk architecture.
- PCI Express Advanced Error Reporting (PF only).

  Note: Advanced Error Reporting is always enabled in the P-Tile Avalon-ST IP for PCIe.
- ECRC generation and checking.
- Data bus parity protection.

  Note: Parity protection on the RX and TX Avalon Streaming interfaces is not available in the Intel Quartus Prime 19.4 release, but will be available in a future release.
- Supports D0 and D3 PCIe power states only.
- Lane Margining at Receiver.
- Retimers presence detection.

Multifunction and Virtualization Features:

- SR-IOV support (8 PFs, 2K VFs per each Endpoint).
- Access Control Service (ACS) capability.

  Note: For ACS, only ports 0 and 1 are supported.
- Alternative Routing-ID Interpretation (ARI).
- Function Level Reset (FLR).
- TLP Processing Hint (TPH).

  Note: TPH supports the "No Steering Tag (ST)" mode only.
- Address Translation Services (ATS).
- Process Address Space ID (PasID).
- Configuration Intercept Interface (for VirtIO).

IP Features:

- User packet interface with separate header, data and prefix.
- User packet interface can handle up to 2 TLPs in any given cycle (x16 mode only).
- Up to 512 outstanding Non-Posted requests (x16 core only).
- Up to 256 outstanding Non-Posted requests (x8 and x4 cores).
- Completion timeout interface.
  - The PCIe Hard IP can optionally track outgoing non-posted packets to report completion timeout information to the application.
- Supports Autonomous Hard IP mode.
  - This mode allows the PCIe Hard IP to communicate with the Host before the FPGA configuration and entry into User mode are complete.

- FPGA core configuration via PCIe link (CvP Init and CvP Update).

  Note: CvP Init and CvP Update are available for Intel Stratix 10 DX devices. For Intel Agilex devices, CvP Init is available, and CvP Update will be available in a future Intel Quartus Prime release.

  Note: For Gen3 and Gen4 x16 variants, Port 0 (corresponding to lanes 0 - 15) supports the CvP features. For Gen3 and Gen4 x8 variants, only Port 0 (corresponding to lanes 0 - 7) supports the CvP features. Port 1 (corresponding to lanes 8 - 15) does not support CvP.

- Device-dependent PLD clock (`coreclkout_hip`) frequency.
  - 350 MHz / 400 MHz for Intel Stratix 10 DX devices, 350 MHz / 400 MHz / 500 MHz for Intel Agilex devices.
- P-Tile Debug Toolkit including the following features:
  - Protocol and link status information.
  - Basic and advanced debugging capabilities including PMA register access and Eye viewing capability.

1.3. Release Information

Table 1. P-Tile Avalon Streaming (Avalon-ST) IP for PCI Express Release Information

Item	Description
IP Version	1.1.0
Intel Quartus Prime Version	19.4
Release Date	December 2019
Ordering Codes	No ordering code is required

IP versions are the same as the Intel Quartus Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IPs have a new IP versioning scheme.

The IP versioning scheme (X.Y.Z) number changes from one software version to another. A change in:

- X indicates a major revision of the IP. If you update your Intel Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

Intel verifies that the current version of the Intel Quartus Prime Pro Edition software compiles the previous version of each IP core, if this IP core was included in the previous release. Intel reports any exceptions to this verification in the *Intel IP Release Notes* or clarifies them in the Intel Quartus Prime Pro Edition IP Update tool. Intel does not verify compilation with IP core versions older than the previous release.

1.4. Device Family Support

The following terms define device support levels for Intel FPGA IP cores:

- **Advance support**—the IP core is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- **Preliminary support**—the IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
- **Final support**—the IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 2. Device Family Support

Device Family	Support Level
Intel Stratix 10 DX, Intel Agilex	Advance support.
Other device families	No support. Refer to the *Intel PCI Express Solutions* web page on the Intel website for support information on other device families.

1.5. Performance and Resource Utilization

The following table shows the typical resource utilization information for selected configurations.

The resource usage is based on the Avalon-ST IP core top-level entity (`intel_pcie_ptile_ast`) that includes IP core soft logic implemented in the FPGA fabric.

Table 3. Resource Utilization Information for the PIO Design Example

Design Example Used	Link Configuration	Device Family	ALMs	M20Ks	Logic Registers
Programmed I/O (PIO)	Gen4 x16, EP	Intel Stratix 10 DX	3,179	0	9,321
Programmed I/O (PIO)	Gen4 x16, EP	Intel Agilex	3,330	0	9,082

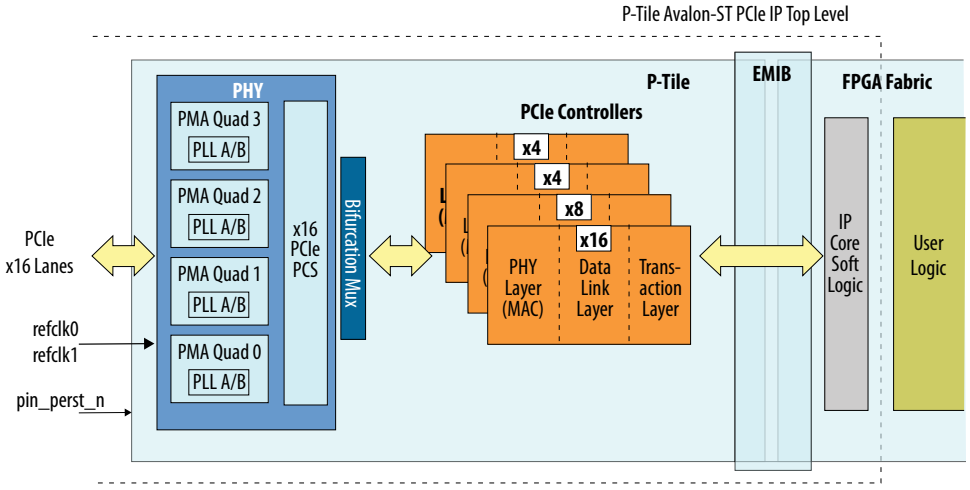
2. IP Architecture and Functional Description

2.1. Architecture

The P-tile Avalon-ST IP for PCI Express consists of the following major sub-blocks:

- PMA/PCS
- Four PCIe cores (one x16 core, one x8 core and two x4 cores)
- Embedded Multi-die Interconnect Bridge (EMIB)
- Soft logic blocks in the FPGA fabric to implement functions such as VirtIO, etc.

Figure 1. P-tile Avalon-ST IP for PCI Express top-level block diagram



Note: Each core in the PCIe Hard IP implements its own Data Link Layer and Transaction Layer.

The four cores in the PCIe Hard IP can be configured to support the following topologies:

Table 4. Configuration Modes Supported by the P-tile Avalon-ST IP for PCI Express

Configuration Mode	Native IP Mode	Endpoint (EP) / Root Port (RP) / TLP Bypass (BP)	Active Cores
Configuration Mode 0	Gen3x16 or Gen4x16	EP/RP/BP	x16
Configuration Mode 1	Gen3x8/Gen3x8 or Gen4x8/Gen4x8	EP/BP	x16, x8
Configuration Mode 2	Gen3x4/Gen3x4/Gen3x4/Gen3x4 or Gen4x4/Gen4x4/Gen4x4/Gen4x4	RP/BP	x16, x8, x4_0, x4_1

In Configuration Mode 0, only the x16 core is active, and it operates in x16 mode (in either Gen3 or Gen4).

In Configuration Mode 1, the x16 core and x8 core are active, and they operate as two Gen3 x8 cores or two Gen4 x8 cores.

Note: When you use only one of the x8 bifurcated ports, you must ensure that the other bifurcated port's lanes are not physically connected.

In Configuration Mode 2, all four cores (x16, x8, x4_0, x4_1) are active, and they operate as four Gen3 x4 cores or four Gen4 x4 cores.

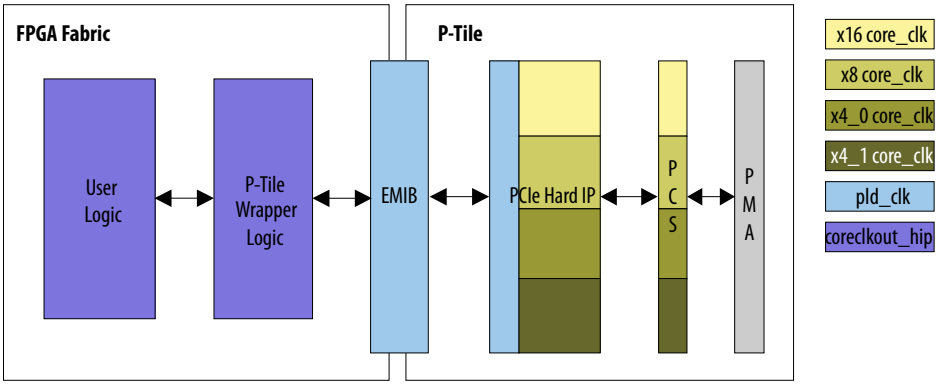
Each of the cores has its own Avalon-ST interface to the user logic. The number of IP-to-User Logic interfaces exposed to the FPGA fabric are different based on the configuration modes. For more details, refer to the *Overview* section of the *Interfaces* chapter.

2.1.1. Clock Domains

The P-Tile IP for PCI Express has three primary clock domains:

- PHY clock domain (i.e. `core_clk` domain): this clock is synchronous to the SerDes parallel clock.
- EMIB/FPGA fabric interface clock domain (i.e. `pld_clk` domain): this clock is derived from the same reference clock (`refclk0`) as the one used by the SerDes. However, this clock is generated from a stand-alone core PLL.
- Application clock domain (`coreclkout_hip`): this clock is an output from the P-Tile IP, and it has the same frequency as `pld_clk`.

Figure 2. Clock Domains



The PHY clock domain (i.e. `core_clk` domain) is a dynamic frequency domain. The PHY clock frequency is dependent on the current link speed.

Table 5. PHY Clock and Application Clock Frequencies

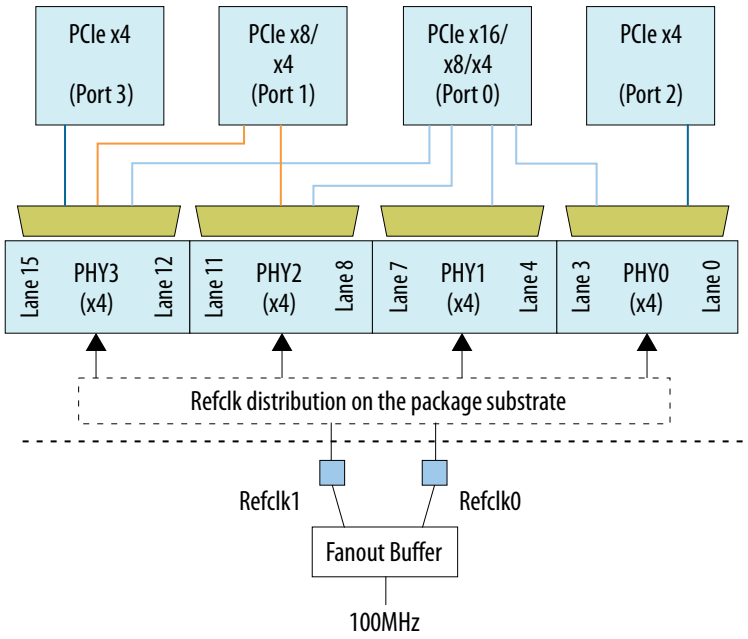
Link Speed	PHY Clock Frequency	Application Clock Frequency
Gen1	125 MHz	Gen1 is supported only via link down-training and not natively. Hence, the application clock frequency depends on the configuration you choose in the IP Parameter Editor. For example, if you choose a Gen3 configuration, the application clock frequency is 250 MHz.
Gen2	250 MHz	Gen2 is supported only via link down-training and not natively. Hence, the application clock frequency depends on the configuration you choose in the IP Parameter Editor. For example, if you choose a Gen3 configuration, the application clock frequency is 250 MHz.
Gen3	500 MHz	250 MHz
Gen4	1000 MHz	350 MHz / 400 MHz (Intel Stratix 10 DX) 350 MHz / 400 MHz / 500 MHz (Intel Agilex)

2.1.2. Refclk

P-Tile has two reference clock inputs at the package level, `refclk0` and `refclk1`. You must connect a 100 MHz reference clock source to these two inputs. Depending on the port mode, you can drive the two `refclk` inputs using either a single clock source or two independent clock sources.

In 1x16 and 4x4 modes, drive the `refclk` inputs with a single clock source (through a fanout buffer) as shown in the figure below.

Figure 3. Using a Single 100 MHz Clock Source in 1x16 and 4x4 Modes

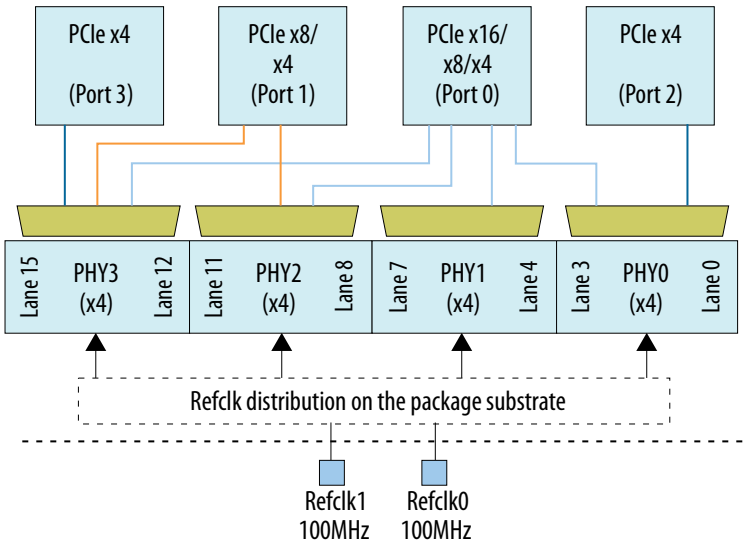


In 2x8 mode, you can drive the `refclk` inputs with either a single 100 MHz clock source as shown above, or two independent 100 MHz sources (see the figure below) depending on your system architecture. For example, if your system has each x8 port connected to a separate CPU/Root Complex, it may be required to drive these `refclk` inputs using independent clock sources. In that case, it is strongly recommended that the `refclk0` input for Port 0 (lanes 0 - 7) be always running because it feeds the reference clock for the P-Tile core PLL that controls the data transfers between the P-Tile and FPGA fabric via the EMIB. If this clock goes down, Port 0 link will go down and Port 1 will not be able to communicate with the FPGA fabric. Following are the guidelines for implementing two independent `refclks` in 2x8 mode:

- If the link can handle two separate `refclks`, the preferred implementation is to use two separate free-running clocks.
- If the link needs to use a shared `refclk`, then PERST# needs to indicate the stability of this `refclk`. If this `refclk` goes down, the entire P-Tile must be reset.
- If `refclk0` is affected, a warm reset for the entire P-Tile is required.

Intel recommends connecting `refclk0` to an on-board free-running oscillator.

Figure 4. Using Independent 100 MHz Clock Sources in 2x8 Mode



2.1.3. Reset (PERST#)

There is only one `pin_perst_n` pin on P-Tile. Therefore, toggling `pin_perst_n` will affect all four ports in P-Tile. Following are the guidelines for implementing the `pin_perst_n` reset signal:

- `pin_perst_n` is a "power good" indicator from the associated power domain (to which P-Tile is connected).
- `pin_perst_n` assertion is required for proper Autonomous P-Tile functionality. In Autonomous mode, P-Tile can successfully link up upon the release of `pin_perst_n` regardless of the FPGA fabric configuration and will send out CRS (Configuration Retry Status) until the FPGA fabric is configured and ready.
- If the P-Tile is part of an add-in card which gets power from the edge connector, then the P-Tile `pin_perst_n` should be connected to the power good signal generated from that power domain, which also qualifies that `refclk0` and `refclk1` are stable.
- If the P-Tile is part of an add-in card which gets power from an auxiliary power supply, then the P-Tile `pin_perst_n` should be connected to the power good signal generated from that auxiliary power supply, which also qualifies that `refclk0` and `refclk1` are stable.

If your system uses independent 100 MHz sources in 2x8 mode, ensure that you deassert `pin_perst_n` after both `refclk0` and `refclk1` are stable. To reset each port individually, you can use the in-band Hot Reset mechanism or the Function-Level Reset (FLR) mechanism.

2.2. Functional Description

2.2.1. PMA/PCS

The P-Tile Avalon-ST IP for PCI Express contains Physical Medium Attachment (PMA) and PCI Express Physical Coding Sublayer (PCIe PCS) blocks for handling the Physical layer (PHY) packets. The PMA receives and transmits high-speed serial data on the serial lanes. The PCS acts as an interface between the PMA and the PCIe controller, and performs functions like data encoding and decoding, scrambling and descrambling, block synchronization etc. The PCIe PCS in the P-Tile Avalon-ST IP for PCI Express is based on the *PHY Interface for PCI Express (PIPE) Base Specification 4.4.1*.

In this IP, the PMA consists of up to four quads. Each quad contains a pair of transmit PLLs and four SerDes lanes capable of running up to 16 GT/s to perform the various TX and RX functions.

PLLA generates the required transmit clocks for Gen1/Gen2 speeds, while PLLB generates the required clocks for Gen3/Gen4 speeds. For the x8 and x16 lane widths, one of the quads acts as the master PLL source to drive the clock inputs for each of the lanes in the other quads.

The PMA performs functions such as serialization/deserialization, clock data recovery, and analog front-end functions such as Continuous Time Linear Equalizer (CTLE), Decision Feedback Equalizer (DFE) and transmit equalization.

The transmitter consists of a 3-tap equalizer with one tap of pre-cursor, one tap of main cursor and one tap of post-cursor.

The receiver consists of attenuation (ATT), CTLE, Voltage gain amplifier (VGA) and a 5-tap DFE blocks that are adaptive for Gen3/Gen4 speeds. RX Lane Margining is supported by the PHY. The Lane Margining supports timing margining only. The optional voltage margining is not supported. Timing margining capabilities/parameters are as follows:

- Maximum Timing Offset: -0.2UI to +0.2UI.
- Number of timing steps: 9.
- Independent left and right timing margining is supported.
- Independent Error Sampler is not supported (lane margining may produce logical errors in the data stream and cause the LTSSM to go to the Recovery state).

The PHY layer uses a fixed 16-bit PCS-PMA interface width to output the PHY clock (`core_clk`). The frequency of this clock is dependent on the current link speed. Refer to Table 5 on page 12 for the frequencies at various link speeds.

Related Information

PHY Interface for PCI Express (PIPE) Base Specification

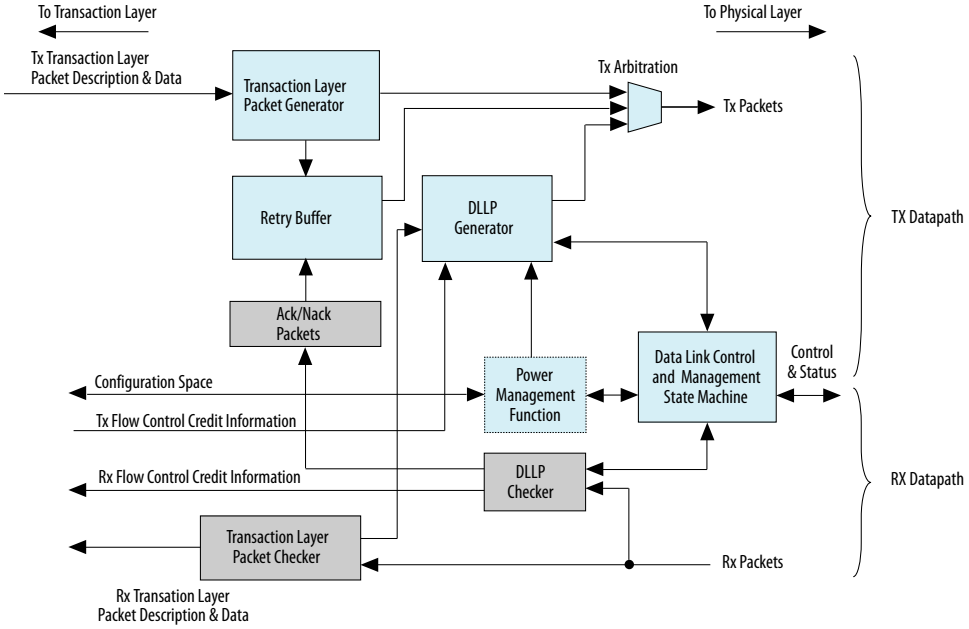
2.2.2. Data Link Layer Overview

The Data Link Layer (DLL) is located between the Transaction Layer and the Physical Layer. It maintains packet integrity and communicates (by DLL packet transmission) at the PCI Express link level.

The DLL implements the following functions:

- Link management through the reception and transmission of DLL Packets (DLLP), which are used for the following functions:
  - Power management of DLLP reception and transmission
  - To transmit and receive ACK/NAK packets
  - Data integrity through the generation and checking of CRCs for TLPs and DLLPs
  - TLP retransmission in case of NAK DLLP reception or replay timeout, using the retry (replay) buffer
  - Management of the retry buffer
  - Link retraining requests in case of error through the Link Training and Status State Machine (LTSSM) of the Physical Layer

Figure 5. Data Link Layer



The DLL has the following sub-blocks:

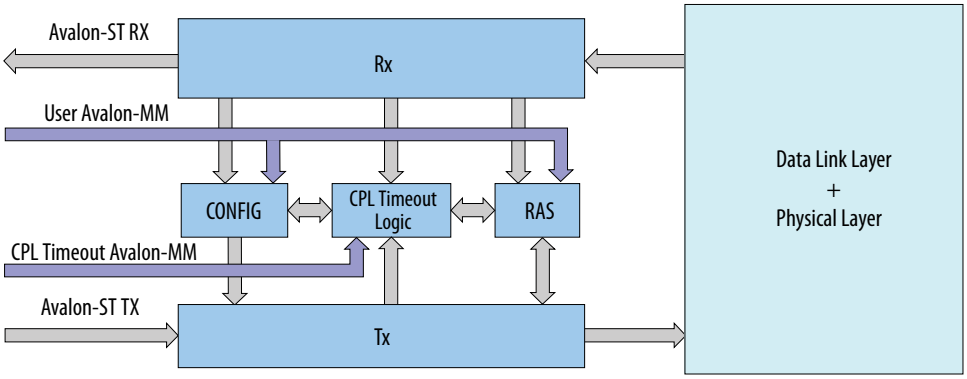
- Data Link Control and Management State Machine—This state machine connects to both the Physical Layer's LTSSM state machine and the Transaction Layer. It initializes the link and flow control credits and reports status to the Transaction Layer.
- Power Management—This function handles the handshake to enter low power mode. Such a transition is based on register values in the Configuration Space and received Power Management (PM) DLLPs. For more details on the power states supported by the P-Tile Avalon-ST IP for PCIe, refer to section Power Management Interface on page 79.
- Data Link Layer Packet Generator and Checker—This block is associated with the DLLP's 16-bit CRC and maintains the integrity of transmitted packets.

- Transaction Layer Packet Generator—This block generates transmit packets, including a sequence number and a 32-bit Link CRC (LCRC). The packets are also sent to the retry buffer for internal storage. In retry mode, the TLP generator receives the packets from the retry buffer and generates the CRC for the transmit packet.
- Retry Buffer—The retry buffer stores TLPs and retransmits all unacknowledged packets in the case of NAK DLLP reception. In case of ACK DLLP reception, the retry buffer discards all acknowledged packets.
- ACK/NAK Packets—The ACK/NAK block handles ACK/NAK DLLPs and generates the sequence number of transmitted packets.
- Transaction Layer Packet Checker—This block checks the integrity of the received TLP and generates a request for transmission of an ACK/NAK DLLP.
- TX Arbitration—This block arbitrates transactions, prioritizing in the following order:
  - Initialize FC Data Link Layer packet
  - ACK/NAK DLLP (high priority)
  - Update FC DLLP (high priority)
  - PM DLLP
  - Retry buffer TLP
  - TLP
  - Update FC DLLP (low priority)
  - ACK/NAK FC DLLP (low priority)

2.2.3. Transaction Layer Overview

The following figure shows the major blocks in the P-Tile Avalon-ST IP for PCI Express Transaction Layer:

Figure 6. P-Tile Avalon-ST IP for PCI Express Transaction Layer Block Diagram



The RAS (Reliability, Availability, and Serviceability) block includes a set of features to maintain the integrity of the link.

For example: Transaction Layer inserts an optional ECRC in the transmit logic and checks it in the receive logic to provide End-to-End data protection.

When the application logic sets the TLP Digest (TD) bit in the Header of the TLP, the P-Tile Avalon-ST IP for PCIe will append the ECRC automatically.

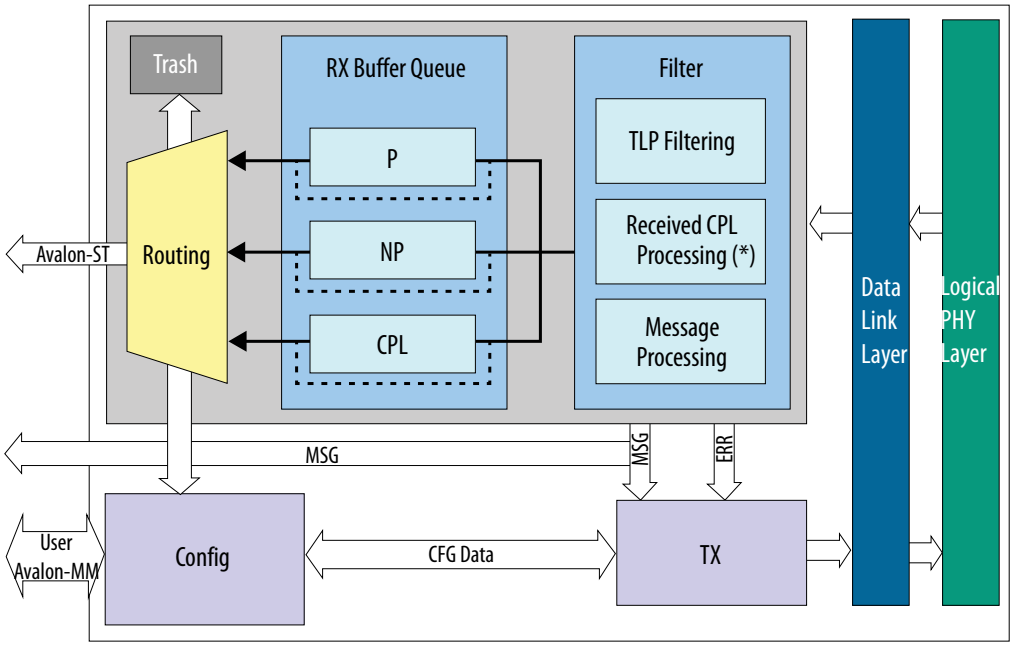
Note that in TLP Bypass mode, the PCIe Hard IP does not generate/check the ECRC and will not remove it if the received TLP has the ECRC.

The TX block sends out the TLPs that it receives as-is. It also sends the information about non-posted TLPs to the CPL Timeout Block for CPL timeout detection.

The P-Tile Avalon-ST IP for PCI Express RX block consists of two main blocks:

- Filtering block: This module checks if the TLP is good or bad and generates the associated error message and completion. It also tracks received completions and updates the completion timeout (CPL timeout) block.
- RX Buffer Queue: The P-Tile IP for PCIe has separate queues for posted/non-posted transactions and completions. This avoids head-of-queue blocking on the received TLPs and provides flexibility to extract TLPs according to the PCIe ordering rules.

Figure 7. P-Tile Avalon-ST IP for PCI Express RX Block Overview



Note: The Received CPL Processing block includes the CPL tracking mechanism.

3. Parameters

This chapter provides a reference for all the parameters that are configurable in the Intel Quartus Prime IP Parameter Editor for the P-Tile Avalon-ST IP for PCIe.

3.1. Top-Level Settings

Table 6. Top-Level Settings

Parameter	Value	Default Value	Description
Hard IP Mode	**Gen4x16, Interface - 512-bit** **Gen3x16, Interface - 512-bit** **Gen4x8, Interface - 512-bit** **Gen4x8, Interface - 256-bit** **Gen3x8, Interface - 256-bit** **Gen4x4, Interface - 256-bit** **Gen4x4, Interface - 128-bit** **Gen3x4, Interface - 128-bit**	**Gen4x16, Interface - 512-bit**	Select the following elements: The lane data rate: • Gen3, Gen4 are supported. The lane width: • x16 mode is for both Root Port and Endpoint. • x8 mode is for Endpoint only. • x4 mode is for Root Port only. *Note:* Gen4 x8 - 512-bit and Gen4 x4 - 256-bit configurations have compilation and simulation support only.
Port Mode	**Root Port** **Native Endpoint** *Note:* These are the available options when **Enable TLP Bypass** is set to **False**. If TLP Bypass mode is enabled, refer to the table *Port Mode Options in TLP Bypass* below for available port mode options.	**Native Endpoint**	Specifies the port type.
Enable PHY Reconfiguration	**True/False**	**False**	Enable the PHY Reconfiguration Interface.
Enable TLP Bypass	**True/False**	**False**	Enable the TLP Bypass feature.
Enable SRIS Mode	**True/False**	**False**	Enable the Separate Reference Clock with Independent Spread Spectrum Clocking (SRIS) feature.

continued...

Parameter	Value	Default Value	Description
			When this is disabled, the default mode of operation is Separate Reference Clock with no Spread Spectrum Clocking (SRNS).

Table 7. Port Mode Options in TLP Bypass

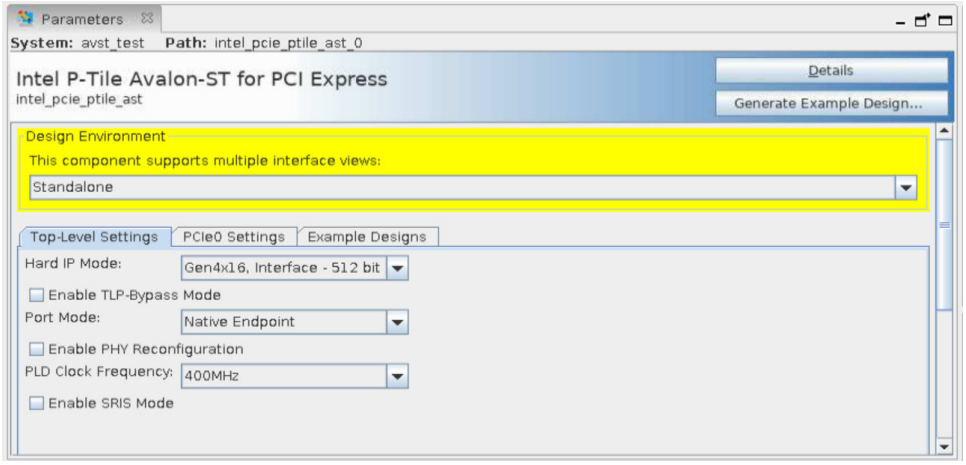
Configuration	Available Port Modes	Default Port Mode
1x16 (Gen4x16 or Gen3x16)	**Downstream** **Upstream**	**Downstream**
2x8 (Gen4x8/Gen4x8 or Gen3x8/ Gen3x8)	**Downstream / Downstream** **Upstream / Upstream** **Endpoint / Upstream** **Upstream / Downstream**	**Downstream / Downstream**
4x4 (Gen4x4/Gen4x4/Gen4x4/Gen4x4 or Gen3x4/Gen3x4/Gen3x4/Gen3x4)	**Downstream / Downstream / Downstream / Downstream** **Upstream / Upstream / Upstream / Upstream**	**Downstream / Downstream / Downstream / Downstream**

3.2. Core Parameters

Depending on which **Hard IP Mode** you choose in the **Top-Level Settings** tab, you will see different tabs for setting the core parameters.

Figure 8. Intel P-tile Avalon-ST Top-Level IP Parameter Editor for a x16 Hard IP Mode

If you choose a x16 mode (either Gen4 or Gen3), only the **PCIe0 Settings** tab will appear.



Note: You can enable the TLP Bypass mode in the **Top-Level Settings** tab of the IP Parameter Editor as shown in the figure below:

Figure 9. Enabling TLP Bypass Mode

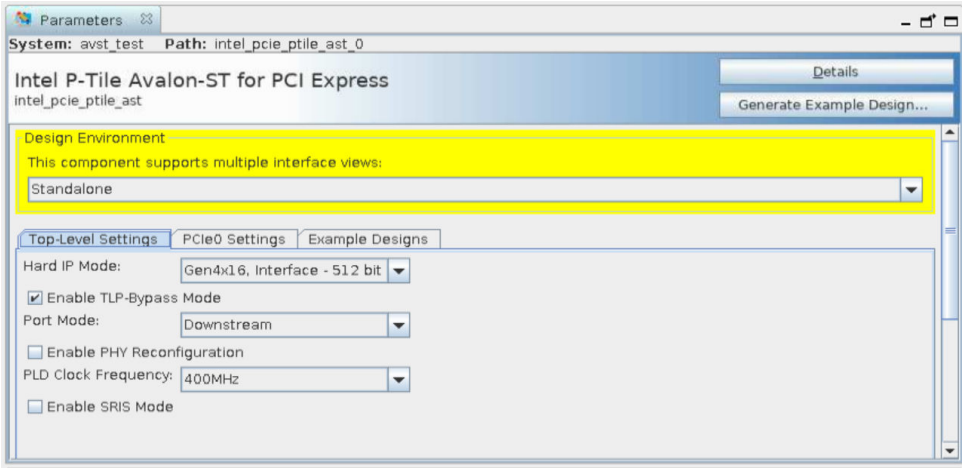


Figure 10. Intel P-tile Avalon-ST Top-Level IP Parameter Editor for a x8 Hard IP Mode

If you choose a x8 mode (either Gen4 or Gen3), the **PCIe0 Settings** and **PCIe1 Settings** tabs will appear.

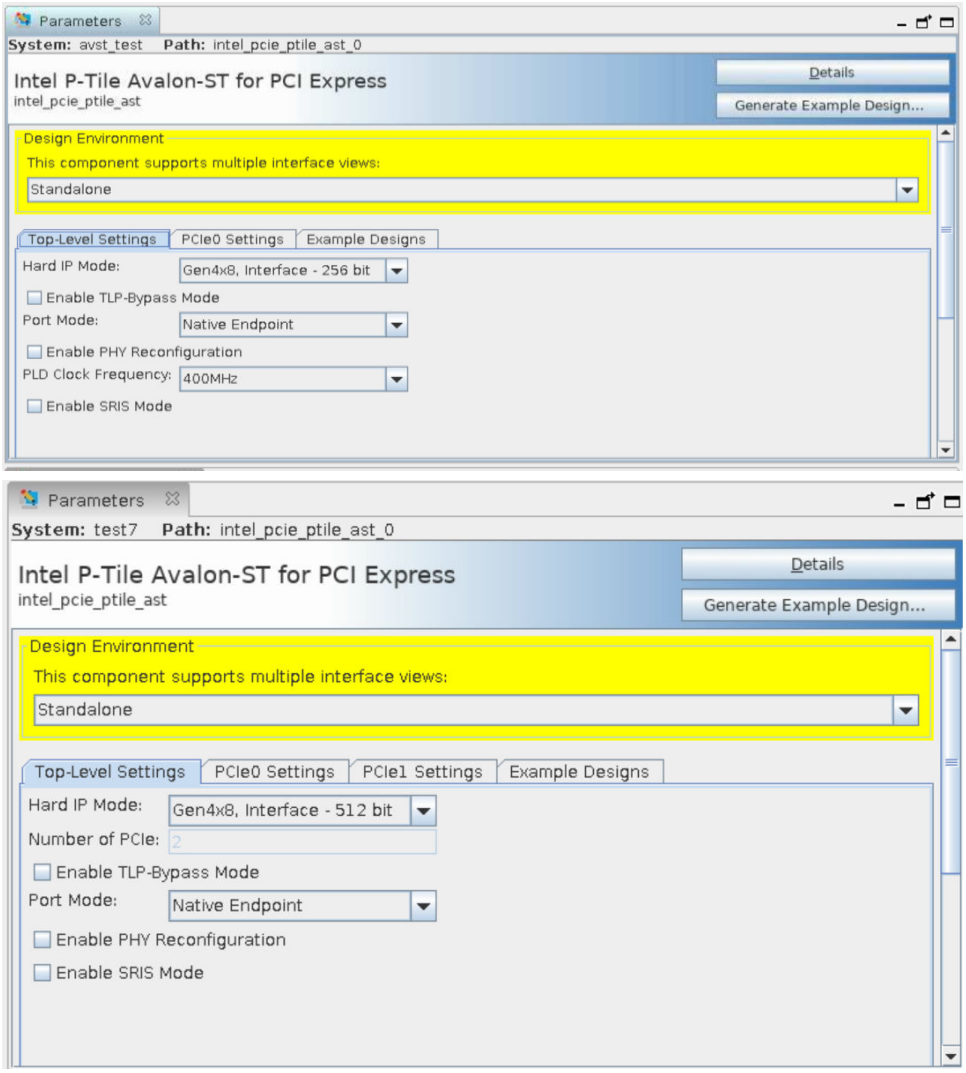
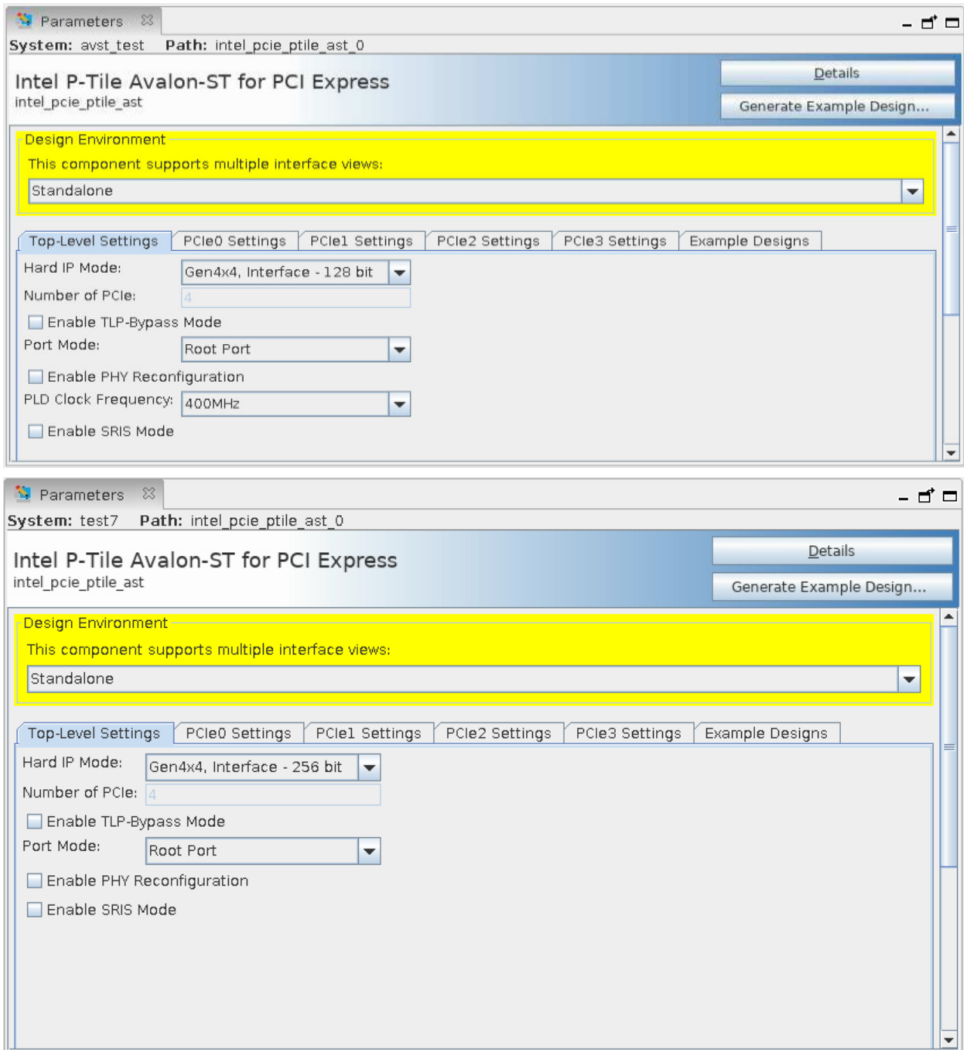


Figure 11. Intel P-tile Avalon-ST Top-Level IP Parameter Editor for a x4 Hard IP Mode

If you choose a x4 mode (either Gen4 or Gen3), the **PCIe0 Settings**, **PCIe1 Settings**, **PCIe2 Settings** and **PCIe3 Settings** tabs will appear.



3.2.1. System Parameters

Table 8. System Parameters

Parameter	Value	Default Value	Description
Enable Multiple Physical Functions	**True/False**	**False**	Enable support for multiple physical functions.

Note: If you set the **Enable Multiple Physical Functions** parameter to **True**, the **Multifunction and SR-IOV System Settings** tab will appear to allow you to set the number of physical functions and enable SR-IOV support.

3.2.2. Avalon Parameters

Table 9. Avalon Parameters

Parameter	Value	Default Value	Description
Enable Power Management Interface	**True/False**	**False**	Enable the Power Management Interface. For more details, refer to section Power Management Interface on page 79.
Enable Legacy Interrupt	**True/False**	**False**	Enable the support for legacy interrupts. For more details, refer to section Legacy Interrupts on page 66.
Enable Parity Error	**True/False**	**True**	Enable the support for parity error checking. Parity errors are indicated by outputs `rx_par_err_o` and `tx_par_err_o`. *Note:* Parity error checking is not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.
Enable Completion Timeout Interface	**True/False**	**False**	Enable the Completion Timeout Interface. For more details, refer to section Completion Timeout Interface on page 75.
Enable Configuration Intercept Interface	**True/False**	**False**	Enable the Configuration Intercept Interface. For more details, refer to section Configuration Intercept Interface (EP Only) on page 85. *Note:* This parameter is only available in EP mode.
Enable PRS Event	**True/False**	**False**	Enable the Page Request Service (PRS) Event Interface. For more details, refer to section Page Request Service (PRS) Interface (EP Only) on page 95. *Note:* This parameter is only available in EP mode.
Enable Error Interface	**True/False**	**False**	Enable the Error Interface. For more details, refer to section Error Interface on page 73.
Enable Byte Parity Ports on Avalon-ST Interface	**True/False**	**False**	When this parameter is enabled, the byte parity ports appear on the block symbol. These byte parity ports include: `rx_st_data_par_o`,

continued...

Parameter	Value	Default Value	Description
			`rx_st_hdr_par_o`, `rx_st_tlp_prfx_par_o`, `tx_st_data_par_o`, `tx_st_hdr_par_o`, and `tx_st_tlp_prfx_par_o` ports. *Note:* The byte parity signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.

3.2.3. Base Address Registers

Table 10. BAR Registers

Parameter	Value	Description
BAR0 Type	**Disabled** **64-bit prefetchable memory** **64-bit non-prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	If you select 64-bit prefetchable memory, 2 contiguous BARs are combined to form a 64-bit prefetchable BAR; you must set the higher numbered BAR to **Disabled**. Defining memory as prefetchable allows contiguous data to be fetched ahead. Prefetching memory is advantageous when the requestor may require more data from the same region than was originally requested. If you specify that a memory is prefetchable, it must have the following 2 attributes: • Reads do not have side effects such as changing the value of the data read. • Write merging is allowed.
BAR1 Type	**Disabled** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	
BAR2 Type	**Disabled** **64-bit prefetchable memory** **64-bit non-prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	
BAR3 Type	**Disabled** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	
BAR4 Type	**Disabled** **64-bit prefetchable memory** **64-bit non-prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	
BAR5 Type	**Disabled** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	
BARn Size	**128 Bytes - 16 EBytes**	Specifies the size of the address space accessible to BARn when BARn is enabled.

continued...

Parameter	Value	Description
		n = 0, 1, 2, 3, 4 or 5
Expansion ROM	**Disabled** **4 KBytes - 12 bits** **8 KBytes - 13 bits** **16 KBytes - 14 bits** **32 KBytes - 15 bits** **64 KBytes - 16 bits** **128 KBytes - 17 bits** **256 KBytes - 18 bits** **512 KBytes - 19 bits** **1 MByte - 20 bits** **2 MBytes - 21 bits** **4 MBytes - 22 bits** **8 MBytes - 23 bits** **16 MBytes - 24 bits**	Specifies the size of the expansion ROM from 4 KBytes to 16 MBytes when enabled.

3.2.4. Device Identification Registers

The following table lists the default values of the Device ID registers. You can use the parameter editor to change the values of these registers.

Table 11. Device ID Registers

Register Name	Range	Default Value	Description
Vendor ID	16 bits	0x00001172	Sets the read-only value of the Vendor ID register. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. *Note:* Set your own Vendor ID by changing this parameter. Address offset: 0x000.
Device ID	16 bits	0x00000000	Sets the read-only value of the Device ID register. This register is only valid in the Type 0 (Endpoint) Configuration Space. Address offset: 0x000.
Revision ID	8 bits	0x00000001	Sets the read-only value of the Revision ID register. Address offset: 0x008.
Class Code	24 bits	0x00000000	Sets the read-only value of the Class Code register. Address offset: 0x008. This parameter cannot be set to 0x0 per the *PCI Express Base Specification*.
Subsystem Vendor ID	16 bits	0x00000000	Sets the read-only value of the Subsystem Vendor ID register in the PCI Type 0 Configuration Space. This parameter cannot be set to 0xFFFF per the *PCI Express*

continued...

Register Name	Range	Default Value	Description
			Base Specification. This value is assigned by PCI-SIG to the device manufacturer.
Subsystem Device ID	16 bits	0x00000000	Sets the read-only value of the Subsystem Device ID register in the PCI Type 0 Configuration Space. Address offset: 0x02C.

3.2.5. Multi-function and SR-IOV

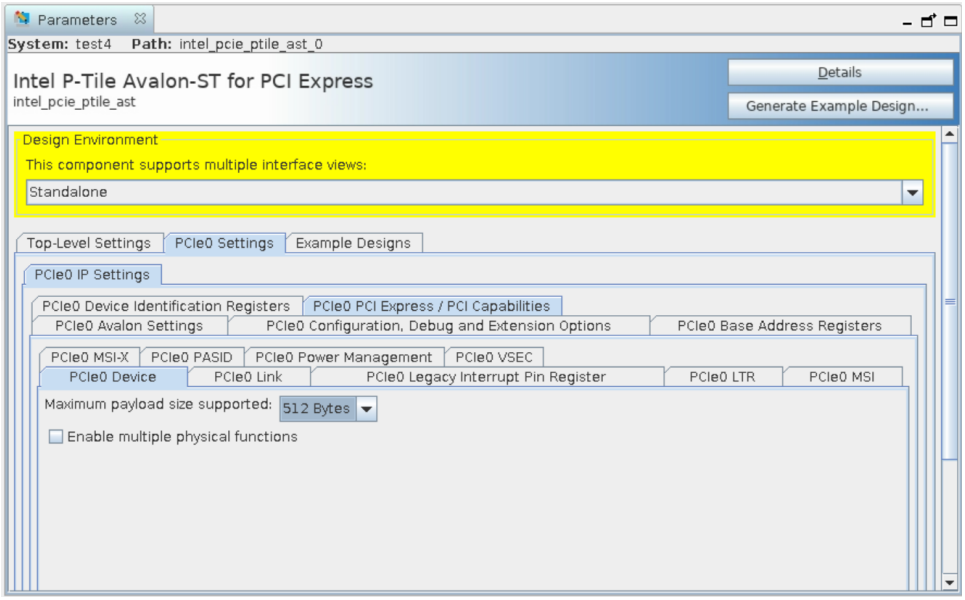
Table 12. Multi-function and SR-IOV

Parameter	Value	Default Value	Description
Total Physical Functions (PFs)	1 - 8	1	Set the number of physical functions. The IP core can support 1 - 8 PFs.

3.2.6. PCI Express and PCI Capabilities Parameters

For each core (PCIe0/PCIe1/PCIe2/PCIe3), the **PCI Express / PCI Capabilities** tab contains separate tabs for the device, link, MSI, MSI-X, power management, vendor specific extended capability (VSEC), Latency Tolerance Reporting (LTR), Process Address Space ID (PasID) and legacy interrupt pin register parameters.

Figure 12. PCI Express / PCI Capabilities Parameters



3.2.6.1. Device Capabilities

Table 13. Device Capabilities

Parameter	Value	Default Value	Description
Maximum payload sizes supported	**128 bytes** **256 bytes** **512 bytes**	512 bytes	Specifies the maximum payload size supported. This parameter sets the read-only value of the max payload size supported field of the Device Capabilities register.
Enable Function Level Reset	**True/False**	**False**	When this option is **True**, each function has its own individual reset. Required for all SR-IOV functions.

3.2.6.2. Link Capabilities

Table 14. Link Capabilities

Parameter	Value	Default Value	Description
Link port number (Root Port only)	0 - 255	1	Sets the read-only value of the port number field in the `Link Capabilities` register. This parameter is for Root Ports only. It should not be changed.
Slot clock configuration	**True/False**	**True**	When this parameter is **True**, it indicates that the Endpoint uses the same physical reference clock that the system provides on the connector. When it is **False**, the IP core uses an independent clock regardless of the presence of a reference clock on the connector. This parameter sets the Slot Clock Configuration bit (bit 12) in the `PCI Express Link Status` register.

3.2.6.3. MSI Capabilities

Table 15. MSI Capabilities

Parameter	Value	Default Value	Description
PF0 Enable MSI	**True/False**	**False**	Enables MSI functionality for PF0. If this parameter is **True**, the **Number of MSI messages requested**

continued...

Parameter	Value	Default Value	Description
			parameter will appear allowing you to set the number of MSI messages.
PF0 MSI 64-bit Addressing	**True/False**	**False**	Enables or disables MSI 64-bit addressing for PF0.
PF0 MSI Extended Data Capable	**True/False**	**False**	Enables or disables MSI extended data capability for PF0.
PF0 Number of MSI messages requested	**1** **2** **4** **8** **16** **32**	**1**	Sets the number of messages that the application can request in the multiple message capable field of the `Message Control` register.

3.2.6.4. MSI-X Capabilities

Table 16. MSI-X Capabilities

Parameter	Value	Default Value	Description
Enable MSI-X (Endpoint only)	**True/False**	**False**	Enables the MSI-X functionality. For SR-IOV: VFs and PFs are always MSI-X capable.
MSI-X Table Size	0x0 - 0x7FF (only values of powers of two minus 1 are valid)	0	System software reads this field to determine the MSI-X table size <**n**>, which is encoded as <**n-1**>. For example, a returned value of 2047 indicates a table size of 2048. This field is read-only. Address offset: 0x068[26:16]
MSI-X Table Offset	0x0 - 0xFFFFFFFF	0	Points to the base of the MSI-X table. The lower 3 bits of the table BAR indicator (BIR) are set to zero by software to form a 64-bit qword-aligned offset. This field is read-only after being programmed.
Table BAR indicator	0x0 - 0x5	0	Specifies which one of a function's BARs, located beginning at 0x10 in Configuration Space, is used to map the MSI-X table into memory space. This field is read-only after being programmed.
Pending bit array (PBA) offset	0x0 - 0xFFFFFFFF	0	Used as an offset from the address contained in one of the function's Base Address registers to point to the base of the MSI-X PBA. The lower 3 bits of the PBA BIR are set to zero by software to form

continued...

Parameter	Value	Default Value	Description
			a 32-bit qword-aligned offset. This field is read-only after being programmed.
PBA BAR indicator	0x0 - 0x5	0	Specifies the function's Base Address register, located beginning at 0x10 in Configuration Space, that maps the MSI-X PBA into memory space. This field is read-only after being programmed.
VF Table size	0x0 - 0x7FF (only values of powers of two minus 1 are valid)	0	Sets the number of entries in the MSI-X table for VFs. MSI-X cannot be disabled for VFs. Set to 1 to save resources.

3.2.6.5. Power Management

Table 17. Power Management

Parameter	Value	Default Value	Description
Enable L0s acceptable latency	Maximum of 64 ns Maximum of 128 ns Maximum of 256 ns Maximum of 512 ns Maximum of 1 us Maximum of 2 us Maximum of 4 us No limit	Maximum of 64 ns	This design parameter specifies the maximum acceptable latency that the application layer can tolerate for any link between the device and the root complex to exit the L0s state. It sets the read-only value of the Endpoint L0s acceptable latency field of the `Device Capabilities Register` (0x084). This Endpoint does not support the L0s or L1 states. However, in a switched system, there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latency for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 64 ns. This is the safest setting for most designs.
Endpoint L1 acceptable latency	Maximum of 1 us Maximum of 2 us Maximum of 4 us Maximum of 8 us Maximum of 16 us	Maximum of 1 us	This value indicates the acceptable latency that an Endpoint can withstand in the transition from the L1 state to L0 state. It is an indirect measure of the

continued...

Parameter	Value	Default Value	Description
	Maximum of 32 us Maximum of 64 us No limit		Endpoint's internal buffering. It sets the read-only value of the Endpoint L1 acceptable latency field of the `Device Capabilities Register`. This Endpoint does not support the L0s or L1 states. However, a switched system may include links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latency for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports.

3.2.6.6. Vendor Specific Extended Capability (VSEC) Registers

Table 18. VSEC Register

Parameter	Value	Default Value	Description
Vendor Specific Extended Capability	0/1	0	Enables the Vendor Specific Extended Capability (VSEC).
User ID register from the Vendor Specific Extended Capability	0 - 65534	0	Sets the read-only value of the 16-bit User ID register from the Vendor Specific Extended Capability. This parameter is only valid for Endpoints.
Drops Vendor Type0 Messages	0/1	0	When this parameter is set to **1**, the IP core drops vendor Type 0 messages while treating them as Unsupported Requests (UR). When it is set to **0**, the IP core passes these messages on to the user logic.
Drops Vendor Type1 Messages	0/1	0	When this parameter is set to **1**, the IP core silently drops vendor Type 1 messages. When it is set to **0**, the IP core passes these messages on to the user logic.

3.2.6.7. Latency Tolerance Reporting (LTR)

This capability allows the P-Tile Avalon Streaming IP, when operating in Endpoint mode, to report the delay that it can tolerate when requesting service from the Host. This information can help software optimize performance when the Endpoint needs a fast response, or optimize system power when a fast response is not necessary.

Table 19. Latency Tolerance Reporting (LTR) Parameters

Parameter	Value	Default Value	Description
PCIe0 Enable LTR	**True/False**	**False**	Enable or disable LTR capability for PCIe0.

3.2.6.8. Process Address Space ID (PASID)

Table 20. Process Address Space ID (PASID) Parameters

Parameter	Value	Default Value	Description
PCIe0 PF0 Enable PASID	**True/False**	**False**	Enable or disable PASID capability for PCIe0 PF0.
PCIe0 PF0 Enable Execute Permission Support	**True/False**	**False**	Enable or disable PASID Execute Permission Support for PCIe0 PF0.
PCIe0 PF0 Enable Privileged Mode Support	**True/False**	**False**	Enable or disable PASID Privileged Mode Support for PCIe0 PF0.
PCIe0 PF0 Max PASID Width	**0 - 20**	**0**	Set the Max PASID Width for PCIe0 PF0.

3.2.6.9. Legacy Interrupt Pin Register

Table 21. Legacy Interrupts Parameters

Parameter	Value	Default Value	Description
Enable Legacy Interrupts for PF0	**True/False**	**False**	Enable Legacy Interrupts (INTx) for PF0 of PCIe0.
Set Interrupt Pin for PF0	**NO INT** **INTA** **INTA/INTB/INTC/INTD**	**NO INT**	When Legacy Interrupts are not enabled, the only option available is **NO INT**. When Legacy Interrupts are enabled and multifunction is disabled, the only option available is **INTA**. When Legacy Interrupts are enabled and multifunction is enabled, the options available are **INTA**, **INTB**, **INTC** and **INTD**.

3.2.7. Slot Capabilities

Table 22. Slot Capabilities

Parameter	Value	Default Value	Description
Use Slot register	**True/False**	**False**	This parameter is only supported in Root Port mode. The slot capability is required for Root Ports if a slot is implemented on the port. Slot status is recorded in the `PCI Express Capabilities` register.

continued...

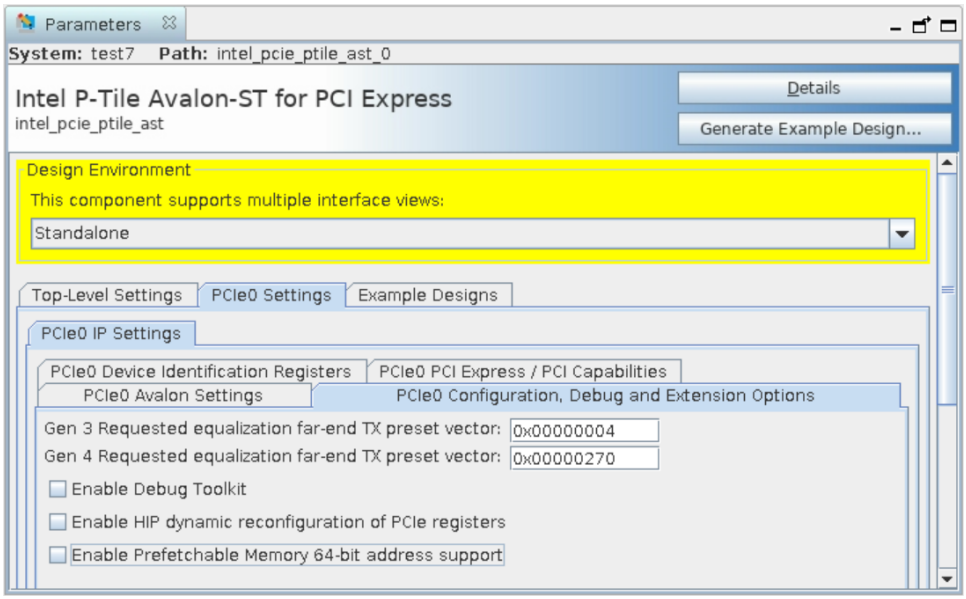
Parameter	Value	Default Value	Description
			Defines the characteristics of the slot. Refer to the figure below for bit definitions.
Slot power scale	0 - 3	0	Specifies the scale used for the slot power limit. The following coefficients are defined: • 0 = 1.0x • 1 = 0.1x • 2 = 0.01x • 3 = 0.001x The default value prior to hardware and firmware initialization is b'00. Writes to this register also cause the port to send the `Set_Slot_Power_Limit` message.
Slot power limit	0 - 255	0	In combination with the **Slot power scale** value, specifies the upper limit in watts for the power supplied by the slot.
Slot number	0 - 8191	0	Specifies the slot number.

3.2.8. Configuration, Debug and Extension Options

Table 23. Configuration, Debug and Extension Options

Parameter	Value	Default Value	Description
Gen 3 Requested equalization far-end TX preset vector	**0 - 65535**	**0x00000004**	Specifies the Gen 3 requested phase 2/3 far-end TX preset vector. Choosing a value different from the default is not recommended for most designs.
Gen 4 Requested equalization far-end TX preset vector	**0 - 65535**	**0x00000270**	Specifies the Gen 4 requested phase 2/3 far-end TX preset vector. Choosing a value different from the default is not recommended for most designs.
Enable Debug Toolkit	**True/False**	**False**	Enable the P-Tile Debug Toolkit for JTAG-based System Console debug access.
Enable HIP dynamic reconfiguration of PCIe registers	**True/False**	**False**	Enable the user Hard IP reconfiguration Avalon-MM interface.
Enable Prefetchable Memory 64-bit address support	**True/False**	**False**	When the P-Tile IP is in Root Port mode and this option is enabled, the prefetchable memory range supported is 64-bit.

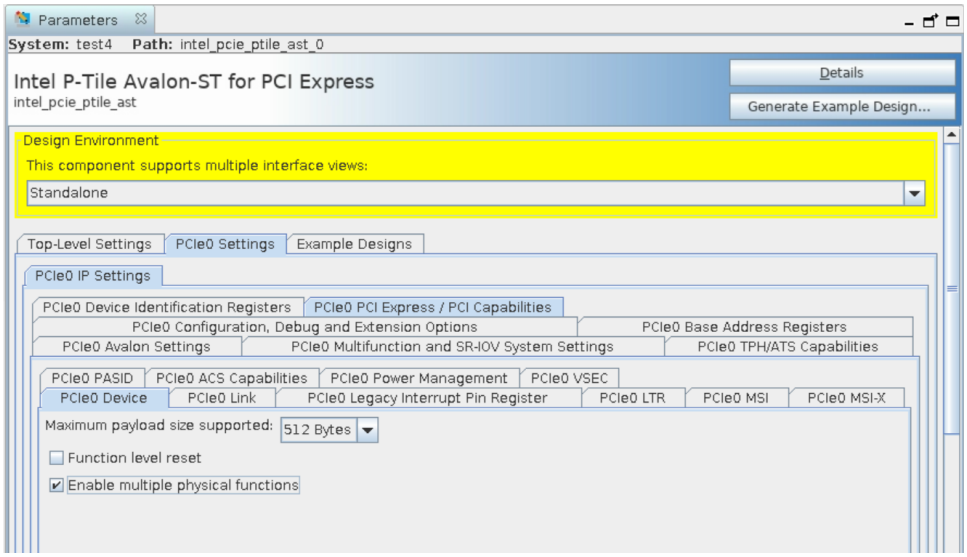
Figure 13. Configuration, Debug and Extension Parameters



3.2.9. VirtIO Parameters

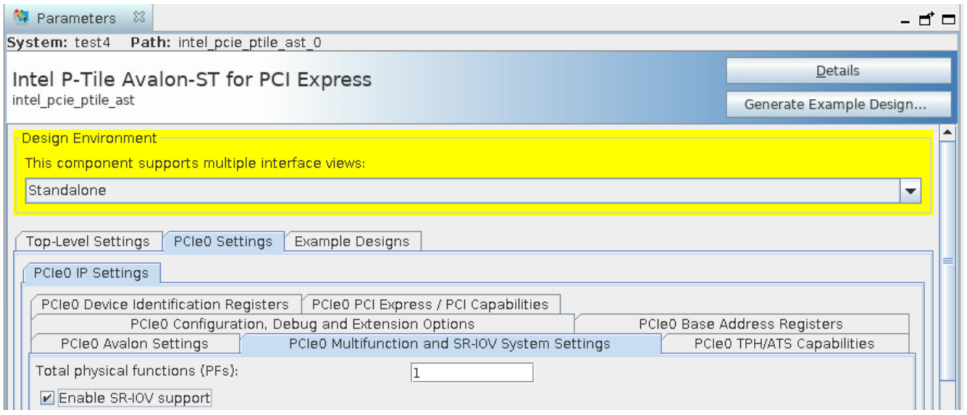
To enable VirtIO support, first enable the support for multiple physical functions in the IP Parameter Editor as shown in the following screenshot:

Figure 14. Enable Multifunction Support



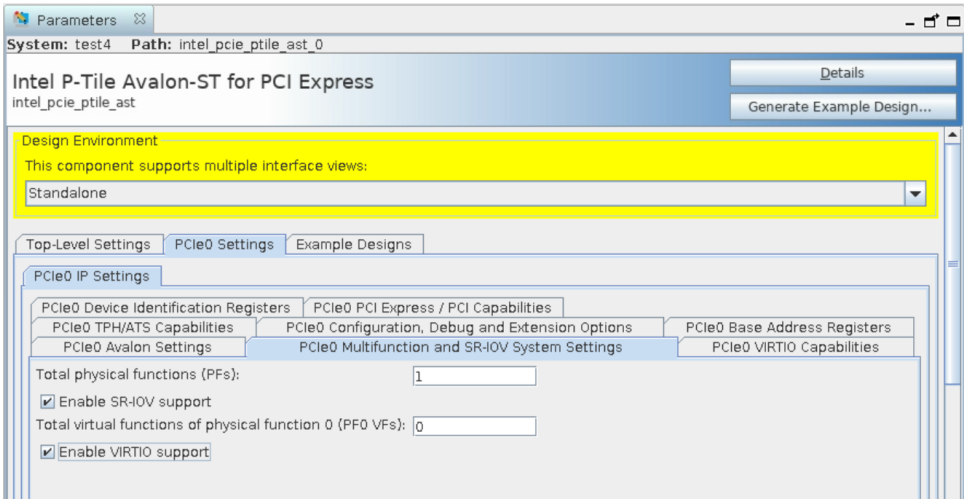
Make sure that SR-IOV support is also enabled:

Figure 15. Enable SR-IOV Support



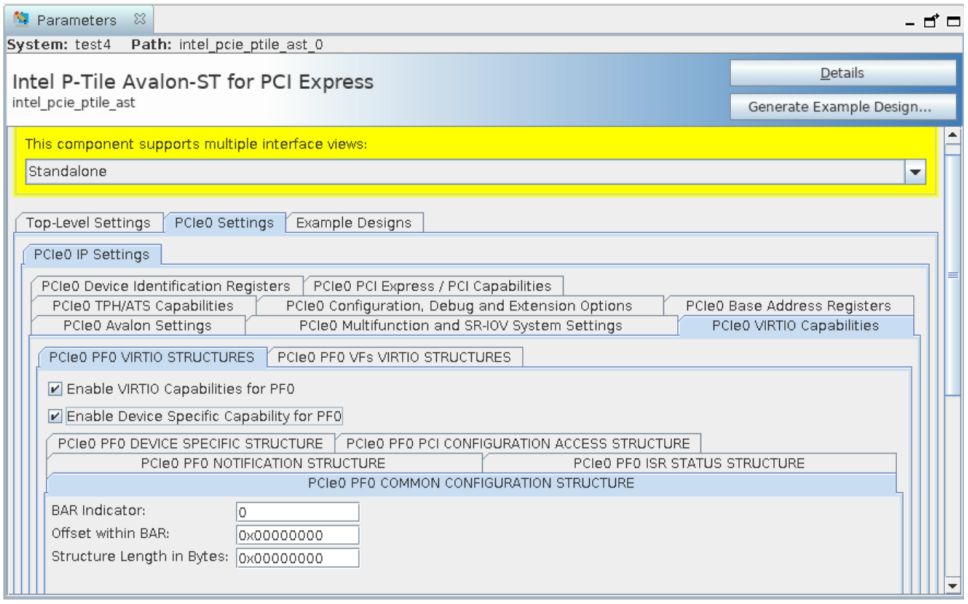
Enable VirtIO support as shown in the screenshot below:

Figure 16. Enable VirtIO Support



Finally, you can configure the appropriate VirtIO capability parameters in the tabs shown in the screenshot below:

Figure 17. Configure VirtIO Capability Parameters



The following table provides a reference for all the configurable high-level parameters of the VirtIO block for P-Tile. Parameters below are dedicated to each core.

Table 24. VirtIO High-Level Parameters

Parameter	Description	Allowed Range	Default Value
Total Physical Functions (PFs) Count Number	The number of supported Physical Functions.	1-8	1
Total Physical Functions (PFs) Count Number Width	Width of the supported Physical Functions number count.	3	3
Total Virtual Functions Count Number of PFs	Total number of VFs associated with PFs. Only present when SR-IOV is enabled.	0-2K	0
Total Virtual Functions Count Number Width of PFs	Width of the count of the total number of VFs associated with PFs. Only present when SR-IOV is enabled.	11	11
Virtual Functions Count Number associated with PF 0-7	Number of VFs associated with PFs 0-7. The sum of all the VF counts for PFs 0-7 cannot exceed the total number of VFs.	0-2K	0

continued...

Parameter	Description	Allowed Range	Default Value
Enable PF VirtIO	Enable Physical Function 0-7 VirtIO capability.	1'b1 / 1'b0	1'b0
Enable VF VirtIO	Enable VirtIO capability of VFs associated with PFs 0-7.	1'b1 / 1'b0	1'b0
Base Address of VirtIO Common Configuration Capability Structure	Start byte address of VirtIO common configuration capability structure for both PFs and VFs.	0x48	0x48

The next table summarizes the parameters associated with the five VirtIO device configuration structures:

Table 25. VirtIO Structure PCI Capabilities Parameters

Parameter	Description	Allowed Range	Default Value
PF/VF VirtIO Common Configuration Structure Capability Parameters			
PFs 0-7 Common Configuration Structure BAR Indicator	Indicates BAR holding the Common Configuration Structure of PFs 0-7.	0-5	0
PFs 0-7 VFs Common Configuration Structure BAR Indicator	Indicates BAR holding the Common Configuration Structure of VFs associated with PFs 0-7.	0-5	0
PFs 0-7 Common Configuration Structure Offset within BAR	Indicates starting position of Common Config Structure in a given BAR of PFs 0-7.	0-536870911	0
PFs 0-7 VFs Common Configuration Structure BAR Indicator	Indicates starting position of Common Config Structure in a given BAR of VFs associated with PFs 0-7.	0-536870911	0
PFs 0-7 Common Configuration Structure Length	Indicates length in bytes of Common Config Structure of PFs 0-7.	0-536870911	0
PFs 0-7 VFs Common Configuration Structure Length	Indicates length in bytes of Common Config Structure of VFs associated with PFs 0-7.	0-536870911	0
PF/VF VirtIO Notifications Structure Capability Parameters			
PFs 0-7 Notifications Structure BAR Indicator	Indicates BAR holding the Notifications Structure of PFs 0-7.	0-5	0
PFs 0-7 VFs Notifications Structure BAR Indicator	Indicates BAR holding the Notifications Structure of VFs associated with PFs 0-7.	0-5	0
PFs 0-7 Notifications Structure Offset within BAR	Indicates starting position of Notifications Structure in given BAR of PFs 0-7.	0-536870911	0
PFs 0-7 VFs Notifications Structure BAR Indicator	Indicates starting position of Notifications Structure in given BAR of VFs associated with PFs 0-7.	0-536870911	0
PFs 0-7 Notifications Structure Length	Indicates length in bytes of Notifications Structure of PFs 0-7.	0-536870911	0

continued...

Parameter	Description	Allowed Range	Default Value
PFs 0-7 VFs Notifications Structure Length	Indicates length in bytes of Notifications Structure of VFs associated with PFs 0-7.	0-536870911	0
PFs 0-7 Notifications Structure Notify Off Multiplier	Indicates multiplier for queue_notify_off in Notifications Structure of PFs 0-7.	0-536870911	0
PFs 0-7 VFs Notifications Structure Notify Off Multiplier	Indicates multiplier for queue_notify_off in Notifications Structure of VFs associated with PFs 0-7.	0-536870911	0
PF/VF VirtIO ISR Status Structure Capability Parameters			
PFs 0-7 ISR Status Structure BAR Indicator	Indicates BAR holding the ISR Status Structure of PFs 0-7.	0-5	0
PFs 0-7 VFs ISR Status Structure BAR Indicator	Indicates BAR holding the ISR Status Structure of VFs associated with PFs 0-7.	0-5	0
PFs 0-7 ISR Status Structure Offset within BAR	Indicates starting position of ISR Status Structure in given BAR of PFs 0-7.	0-536870911	0
PFs 0-7 VFs ISR Status Structure BAR Indicator	Indicates starting position of ISR Status Structure in given BAR of VFs associated with PFs 0-7.	0-536870911	0
PFs 0-7 ISR Status Structure Length	Indicates length in bytes of ISR Status Structure of PFs 0-7.	0-536870911	0
PFs 0-7 VFs ISR Status Structure Length	Indicates length in bytes of ISR Status Structure of VFs associated with PFs 0-7.	0-536870911	0
PF/VF VirtIO Device-Specific Configuration Structure Capability Parameters			
Enable PFs 0-7 VirtIO Device Specific Capability	Enable PFs 0-7 VirtIO Device-Specific Configuration Structure Capability.	Ture / False	False
Enable PFs 0-7 VFs VirtIO Device-Specific Capability	Enable VirtIO Device-Specific Configuration Structure Capability of VFs associated with PFs 0-7.	Ture / False	False
PFs 0-7 Device-Specific Configuration Structure BAR Indicator	Indicates BAR holding the Device-Specific Configuration Structure of PFs 0-7.	0-5	0
PFs 0-7 VFs Device-Specific Configuration Structure BAR Indicator	Indicates BAR holding the Device-Specific Configuration Structure of VFs associated with PFs 0-7.	0-5	0
PFs 0-7 Device-Specific Configuration Structure Offset within BAR	Indicates starting position of Device-Specific Configuration Structure in given BAR of PFs 0-7.	0-536870911	0

continued...

Parameter	Description	Allowed Range	Default Value
PFs 0-7 VFs Device-Specific Configuration Structure BAR Indicator	Indicates starting position of Device-Specific Configuration Structure in given BAR of VFs associated with PFs 0-7.	0-536870911	0
PFs 0-7 Device-Specific Configuration Structure Length	Indicates length in bytes of Device-Specific Configuration Structure of PFs 0-7.	0-536870911	0
PFs 0-7 VFs Device-Specific Configuration Structure Length	Indicates length in bytes of Device-Specific Configuration Structure of VFs associated with PFs 0-7.	0-536870911	0
PF/VF VirtIO PCI Configuration Access Structure Capability Parameters			
PFs 0-7 PCI Configuration Access Structure BAR Indicator	Indicates BAR holding the PCI Configuration Access Structure of PFs 0-7.	0-5	0
PFs 0-7 VFs PCI Configuration Access Structure BAR Indicator	Indicates BAR holding the PCI Configuration Access Structure of VFs associated with PFs 0-7.	0-5	0
PFs 0-7 PCI Configuration Access Structure Offset within BAR	Indicates Starting position of PCI Configuration Access Structure in given BAR of PFs 0-7.	0-536870911	0
PFs 0-7 VFs PCI Configuration Access Structure BAR Indicator	Indicates Starting position of PCI Configuration Access Structure in given BAR of VFs associated with PFs 0-7.	0-536870911	0
PFs 0-7 PCI Configuration Access Structure Length	Indicates length in bytes of PCI Configuration Access Structure of PFs 0-7.	0-536870911	0
PFs 0-7 VFs PCI Configuration Access Structure Length	Indicates length in bytes of PCI Configuration Access Structure of VFs associated with PFs 0-7.	0-536870911	0

4. Interfaces

This section focuses mainly on the signal interfaces that the P-Tile IP for PCIe uses to communicate with the Application Layer in the FPGA fabric core. However, it also briefly covers the Serial Data Interface, which allows the IP to communicate with the link partner across the PCIe link.

4.1. Overview

You can determine which core each interface in this section belongs to by looking at the prefixes in the signal names:

- p0 : x16 core
- p1 : x8 core
- p2 : x4_0 core
- p3 : x4_1 core

Figure 18 on page 42 shows the top-level signals of this IP. Note that the signal names in the figure will get the appropriate prefix pn (where n = 0, 1, 2 or 3) depending on which of the three supported configurations (1x16, 2x8, or 4x4) the P-Tile Avalon-ST IP for PCI Express is in.

As an example, the `rx_st_data_o` bus can take on the following names:

- In the 1x16 configuration, only the x16 core is active. In this case, this bus appears as `p0_rx_st_data_o[511:0]`.
- In the 2x8 configuration, both the x16 core and x8 core are active. In this case, this bus is split into `p0_rx_st_data_o[255:0]` and `p1_rx_st_data_o[255:0]`.
- In the 4x4 configuration, all four cores are active. In this case, this bus is split into `p0_rx_st_data_o[127:0]`, `p1_rx_st_data_o[127:0]`, `p2_rx_st_data_o[127:0]` and `p3_rx_st_data_o[127:0]`.

The only cases where the interface signal names do not get the pn prefixes are the interfaces that are common for all the cores, like the PHY reconfiguration interface, clocks and resets. For example, there is only one `xcvr_reconfig_clk` that is shared by all the cores.

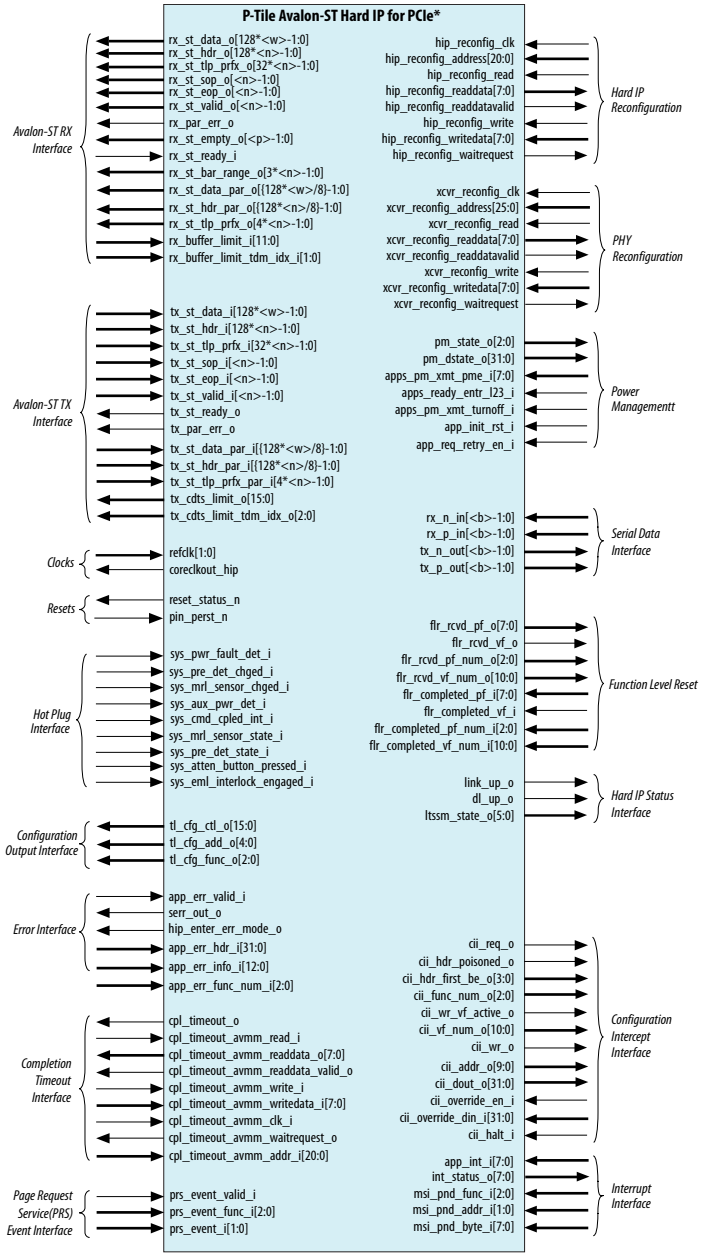
You can enable the PHY reconfiguration interface from the **Top Level Settings** in the GUI.

Each of the cores has its own Avalon-ST interface to the user logic. The number of IP-to-User Logic interfaces exposed to the FPGA fabric are different based on the configuration modes:

Table 26. IP to FPGA Fabric Interfaces Summary

Mode	Avalon-ST Interface Count	Data Width (each Interface)	Header Width (each Interface)	TLP Prefix Width (each Interface)	Application Clock Frequency
Gen4 x16 EP/RP mode	1	512-bit	256-bit	64-bit	400 MHz (Intel Stratix 10 DX) 500 MHz (Intel Agilex)
Gen3 x16 EP/RP mode	1	512-bit	256-bit	64-bit	250 MHz
Gen4 x8 x8 EP mode	2	256-bit	128-bit	32-bit	400 MHz (Intel Stratix 10 DX) 500 MHz (Intel Agilex)
Gen4 x8 x8 EP mode	2	512-bit	256-bit	64-bit	250 MHz
Gen3 x8 x8 EP mode	2	256-bit	128-bit	32-bit	250 MHz
Gen4 x4 x4 x4 x4 RP mode	4	128-bit	128-bit	32-bit	400 MHz (Intel Stratix 10 DX) 500 MHz (Intel Agilex)
Gen4 x4 x4 x4 x4 RP mode	4	256-bit	128-bit	32-bit	250 MHz
Gen3 x4 x4 x4 x4 RP mode	4	128-bit	128-bit	32-bit	250 MHz

Figure 18. P-Tile Avalon-ST IP for PCI Express Top-Level Signals



Note: The table below shows the variables that are used to define the bus indices for top-level signal busses shown in the top-level block diagram above. The values of these variables change depending on which configuration is active (1x16, 2x8 or 4x4). For example, for the 4x4 configuration, using `w=1` and `n=1` will give Avalon-ST RX bus widths of `p0_rx_st_data_o[127:0]`, `p0_rx_st_hdr_o[127:0]` and `p0_rx_st_tlp_prfx_o[31:0]`.

Table 27. Variables Used in the Bus Indices

Variable	1x16 Configuration	2x8 Configuration	4x4 Configuration
w	4	2	1
n	2	1	1
p	6	3	2
b	16	8	4

4.2. Clocks and Resets

4.2.1. Interface Clock Signals

Table 28. Interface Clock Signals

Name	I/O	Description	EP/RP/BP	Clock Frequency
`coreclkout_hip`	O	This clock drives the Application Layer. The frequency depends on the data rate and the number of lanes being used.	EP/RP/BP	Native Gen3: 250 MHz Native Gen4: 400 MHz (Intel Stratix 10 DX) / 500 MHz (Intel Agilex)
`refclk[1:0]`	I	These are the input reference clocks for the IP core. These clocks must be free-running. For more details on how to connect these clocks, refer to the section *Clock Sharing in Bifurcation Modes*.	EP/RP/BP	100 MHz ± 300 ppm
`p0_hip_reconfig_clk`	I	Clock for the `hip_reconfig` interface. This is an Avalon-MM interface. It is an optional interface that is enabled when the **Enable HIP dynamic reconfiguration of PCIe read-only registers** option in the **PCIe Configuration, Debug and Extension Options** tab is enabled.	EP/RP/BP	50 MHz - 125 MHz (range) 100 MHz (recommended)
`xcvr_reconfig_clk`	I	Clock for the PHY reconfiguration interface. This is an Avalon-MM interface. This optional interface is enabled when you turn on the **Enable PHY reconfiguration** option in the **Top-Level Settings** tab. This interface is shared among all the cores.	EP/RP/BP	50 MHz - 125 MHz (range) 100 MHz (recommended)
`p0_cpl_timeout_avmm_clk`	I	Avalon-MM clock for Completion timeout interface. This interface is optional, and is enabled when the **Enable Completion Timeout Interface** option in the **PCIe Avalon Settings** tab is enabled.	EP/RP/BP	50 MHz - 125 MHz (range) 100 MHz (recommended)

4.2.2. Resets

4.2.2.1. Interface Reset Signals

Table 29. Interface Reset Signals

Signal Name	Direction	Clock	EP/RP/BP	Description
pin_perst_n	Input	Asynchronous	EP/RP/BP	This is an active-low input to the PCIe Hard IP, and implements the PERST# function defined by the PCIe specification.
p0_reset_status_n	Output	Synchronous	EP/RP/BP	This active-low signal is held low until pin_perst_n has been deasserted and the PCIe Hard IP has come out of reset. This signal is synchronous to coreclkout_hip. When port bifurcation is used, there is one such signal for each Avalon-ST interface. The signals are differentiated by the prefixes pn.
ninit_done	Input	Asynchronous	EP/RP	A "1" on this active-low signal indicates that the FPGA device is not yet fully configured. A "0" indicates the device has been configured and is in normal operating mode.

4.2.2.2. Function-Level Reset (FLR) Interface (EP Only)

FLR allows specific physical/virtual functions to be reset without affecting other physical/virtual functions or the link they share. FLR can be enabled by checking the check-box **Enable Function Level Reset (FLR)** in the **PCIe Device** tab of the **PCIe PCI Express / PCI Capabilities** tab in the GUI.

This interface is only present in EP mode (for x16/x8 configurations).

Table 30. FLR Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
p0_flr_rcvd_pf_o[7:0]	O	One bit per PF. A one-cycle pulse on any bit indicates that an FLR was received from the host targeting a PF. When port bifurcation is used, there is one such bus for each Avalon-ST interface.	coreclkout_hip	EP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		These busses are differentiated by the prefixes pn.		
p0_flr_rcvd_vf_o	O	A one-cycle pulse indicates that an FLR was received from host targeting a VF. When port bifurcation is used, there is one such signal for each Avalon-ST interface. These signals are differentiated by the prefixes pn.	coreclkout_hip	EP
p0_flr_rcvd_pf_num_o[2:0]	O	Parent PF number of the VF undergoing FLR. When port bifurcation is used, there is one such bus for each Avalon-ST interface. These busses are differentiated by the prefixes pn.	coreclkout_hip	EP
p0_flr_rcvd_vf_num_o[10:0]	O	VF number offset of the VF undergoing FLR. When port bifurcation is used, there is one such bus for each Avalon-ST interface. These busses are differentiated by the prefixes pn.	coreclkout_hip	EP
p0_flr_completed_pf_i[7:0]	I	One bit per PF. A one-cycle pulse on any bit indicates that the application has completed the FLR sequence for the corresponding PF and is ready to be enabled. When port bifurcation is used, there is one such bus for each Avalon-ST interface. These busses are differentiated by the prefixes pn.	coreclkout_hip	EP
p0_flr_completed_vf_i	I	One-cycle pulse from the application re-enables a VF. When port bifurcation is used, there is one such signal for each Avalon-ST interface. These signals are differentiated by the prefixes pn.	coreclkout_hip	EP
p0_flr_completed_pf_num_i[2:0]	I	Parent PF number of the VF to re-enable. When port bifurcation is used, there is one such bus for each Avalon-ST interface. These busses are differentiated by the prefixes pn.	coreclkout_hip	EP
p0_flr_completed_vf_num_i[10:0]	I	VF number offset of the VF to re-enable. When port bifurcation is used,	coreclkout_hip	EP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		there is one such bus for each Avalon-ST interface. These busses are differentiated by the prefixes pn.		

4.3. Serial Data Interface

P-Tile natively supports 4, 8, or 16 PCIe lanes. Each lane includes a TX differential pair and an RX differential pair. Data is striped across all available lanes.

Table 31. Serial Data Interface

Signal Name	Direction	Description
`tx_p_out[<b>-1:0], tx_n_out[<b>-1:0]`	O	Transmit serial data outputs using the High Speed Differential I/O standard.
`rx_p_in[<b>-1:0], rx_n_in[<b>-1:0]`	I	Receive serial data inputs using the High Speed Differential I/O standard.

4.4. Avalon-ST Interface

The P-tile PCIe Hard IP provides an Avalon-ST-like interface with separate header and data to improve the bandwidth utilization.

The Avalon-ST interface has different data bus widths depending on the link width configuration of the PCIe IP.

In the 19.4 Intel Quartus Prime release, a high-bandwidth Adapter has been added. The purpose of this Adapter is to allow the user application logic to run Gen4 x8 and Gen4 x4 configurations with an application clock frequency of 250 MHz. The Adapter achieves this by doubling the data bus width, thus providing a Gen4 x8 512-bit interface or a Gen4 x4 256-bit interface. When either of these two configurations is selected in Platform Designer, Intel Quartus Prime will automatically instantiate the Adapter.

Table 32. Avalon-ST Interface Data and Header Bus Widths

PCIe Link Width	Data Width (bits)	Header Width (bits)	TLP Prefix Width (bits)
x16	512 (2 x 256)	256 (2 x 128)	64 (2 x 32)
x8 (1)	512 (2 x 256)	256 (2 x 128)	64 (2 x 32)
x8	256	128	32
x4 (2)	256	128	32
x4	128	128	32

(1) An Adapter is automatically instantiated by Intel Quartus Prime. This configuration only has compilation and simulation support in the 19.4 Intel Quartus Prime release.

(2) An Adapter is automatically instantiated by Intel Quartus Prime. This configuration only has compilation and simulation support in the 19.4 Intel Quartus Prime release.

Note:

- For the x16 configuration or x8 configuration, two segments of 256-bit data and two segments of 128-bit header are available.
- If a x4 configuration uses a x8 core, only the LSB 128 bits out of the 256-bit data bus are used. If a x4 configuration uses a x4 core, the data bus is 128-bit wide.
- x4 configuration is only present in Root Port mode.

4.4.1. TLP Header and Data Alignment for the Avalon-ST RX and TX Interfaces

The TLP prefix, header and data are sent and received on the TX and RX interfaces.

The ordering of bytes in the header and data portions of packets is different. The first byte of the header dword is located in the most significant byte of the dword. The first byte of the data dword is located in the least significant byte of the dword on the data bus.

Figure 19. Generic TLP Format

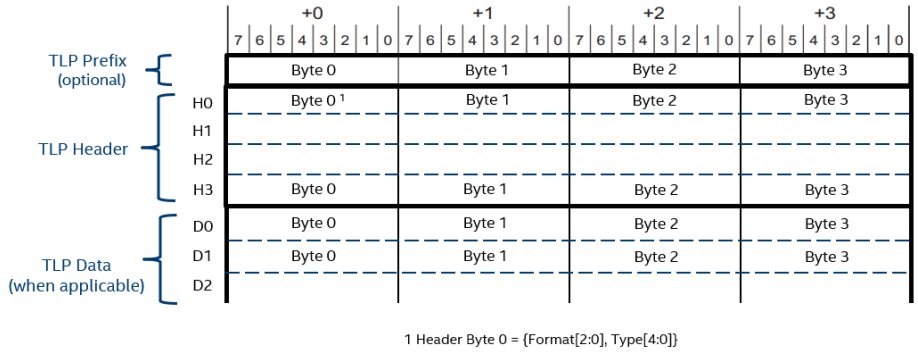
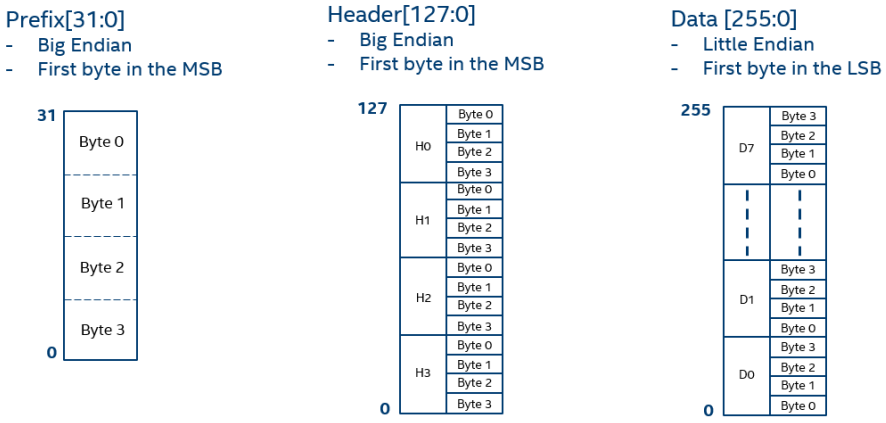


Figure 20. TLP Prefix, Header and Data on the RX and TX Interfaces of the P-Tile IP for PCIe



4.4.2. Avalon-ST RX Interface

The Application Layer receives data from the Transaction Layer of the PCI Express IP core over the Avalon-ST RX interface. The application must assert `rx_st_ready_i` before transfers can begin.

This interface supports two `rx_st_sop_o` signals and two `rx_st_eop_o` signals per cycle when the P-Tile IP is operating in a x16 configuration. It also does not follow a fixed latency between `rx_st_ready_i` and `rx_st_valid_o` as specified by the *Avalon Interface Specifications*.

The x16 core provides two segments with each one having 256 bits of data (`rx_st_data_o[511:256]` and `rx_st_data_o[255:0]`), 128 bits of header (`rx_st_hdr_o[255:128]` and `rx_st_hdr_o[127:0]`), and 32 bits of TLP prefix (`rx_st_tlp_prfx_o[63:32]` and `rx_st_tlp_prfx_o[31:0]`). If this core is configured in the 1x16 mode, both segments are used, so the data bus becomes a 512-bit bus `rx_st_data_o[511:0]`. The start of packet can appear in the upper segment or lower segment, as indicated by the `rx_st_sop_o[1:0]` signals.

If this core is configured in the 2x8 mode, only the lower segment is used. In this case, the data bus is a 256-bit bus `rx_st_data_o[255:0]`.

Finally, if this core is configured in the 4x4 mode, only the lower segment is used and only the MSB 128 bits of data are valid. In this case, the data bus is a 128-bit bus `rx_st_data_o[127:0]`.

The x8 core provides one segment with 256 bits of data, 128 bits of header and 32 bits of TLP prefix. If this core is configured in 4x4 mode, only the LSB 128 bits of data are used.

The x4 core provides one segment with 128 bits of data, 128 bits of header and 32 bits of TLP prefix.

Table 33. Avalon-ST RX Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
x16 PCIe configuration: `rx_st_data_o[511:0]` x8 PCIe configuration: `rx_st_data_o[255:0]` x4 configuration: `rx_st_data_o[127:0]`	O	This is the Receive data bus. The Application Layer receives data from the Transaction Layer on this bus. For TLPs with an end-of-packet cycle in the lower 256 bits, the 512-bit interface supports a start-of-packet cycle in the upper 256 bits.	`coreclkout_hip`	EP/RP/BP
x16: `rx_st_empty_o[5:0]` x8: `rx_st_empty_o[2:0]` x4: `rx_st_empty_o[1:0]`	O	Specify the number of dwords that are empty during cycles when the `rx_st_eop_o` signals are asserted. These signals are not valid when the `rx_st_eop_o` signals are not asserted.	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
rx_st_ready_i	I	Indicates the Application Layer is ready to accept data. The readyLatency is 27 cycles. If rx_st_ready_i is deasserted by the Application Layer on cycle <n>, the Transaction Layer in the PCIe Hard IP continues to send traffic up to <n>+ readyLatency cycles after the deassertion of rx_st_ready_i. Once rx_st_ready_i reasserts, rx_st_valid_o resumes data transfer within readyLatency cycles. To achieve the best performance, the Application Layer must include a receive buffer large enough to avoid the deassertion of rx_st_ready_i.	coreclkout_hip	EP/RP/BP
x16: rx_st_sop_o[1:0] x8/x4: rx_st_sop_o	O	Signals the first cycle of the TLP when asserted in conjunction with the corresponding bit of rx_st_valid_o[1:0]. rx_st_sop_o[1]: When asserted, signals the start of a TLP on rx_st_data_o[511:256]. rx_st_sop_o[0]: When asserted, signals the start of a TLP on rx_st_data_o[255:0].	coreclkout_hip	EP/RP/BP
x16: rx_st_eop_o[1:0] x8/x4: rx_st_eop_o	O	Signals the last cycle of the TLP when asserted in conjunction with the corresponding bit of rx_st_valid_o[1:0]. rx_st_eop_o[1]: When asserted, signals the end of a TLP on rx_st_data_o[511:256]. rx_st_eop_o[0]: When asserted, signals the end of a TLP on rx_st_data_o[255:0].	coreclkout_hip	EP/RP/BP
x16: rx_st_valid_o[1:0] x8/x4: rx_st_valid_o	O	These signals qualify the rx_st_data_o signals going into the Application Layer.	coreclkout_hip	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
x16: rx_st_hdr_o[255:0] x8/x4: rx_st_hdr_o[127:0]	O	This is the received header, which follows the TLP header format of the PCIe specifications.	coreclkout_hip	EP/RP/BP
x16: rx_st_tlp_prfx_o[63:0] x8/x4: rx_st_tlp_prfx_o[31:0]	O	This is the first TLP prefix received, which follows the TLP prefix format of the PCIe specifications. PASID is included. These signals are valid when the corresponding rx_st_sop_o is asserted. The TLP prefix uses a Big Endian implementation (i.e, the Fmt field is in bits [31:29] and the Type field is in bits [28:24]). If no prefix is present for a given TLP, that dword (including the Fmt field) is all zeros.	coreclkout_hip	EP/RP/BP
x16: rx_st_vf_active_o[1:0] x8: rx_st_vf_active_o x4: NA	O	When asserted, these signals indicate that the received TLP is targeting a virtual function. When these signals are deasserted, the received TLP is targeting a physical function and the rx_st_func_num signals indicate the function number. These signals are valid when the corresponding rx_st_sop_o is asserted. These signals are multiplexed with the rx_st_hdr_o signals in the x4 configuration. These signals are valid in Endpoint mode only.	coreclkout_hip	EP
x16: rx_st_func_num_o[5:0] x8: rx_st_func_num_o[2:0] x4: NA	O	Specify the target physical function number for the received TLP. These signals are valid when the corresponding rx_st_sop_o is asserted. These signals are multiplexed with the rx_st_hdr_o signals in the x4 configuration. These signals are valid in Endpoint mode only.	coreclkout_hip	EP
x16: rx_st_vf_num_o[19:0] x8: rx_st_vf_num_o[10:0] x4: NA	O	Specify the target VF number for the received TLP. The application uses this information for both request and completion	coreclkout_hip	EP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		TLPs. For a completion TLP, these bits specify the VF number of the requester for this completion TLP. These signals are valid when `rx_st_vf_active_o` and the corresponding `rx_st_sop_o` are asserted. These signals are multiplexed with the `rx_st_hdr_o` signals in the x4 configuration. These signals are valid in Endpoint mode only.		
x16: `rx_st_bar_range_o[5:0]` x8/x4: `rx_st_bar_range_o[2:0]`	O	Specify the BAR for the TLP being output. For each BAR range, the following encodings are defined: • 000: Memory BAR 0 • 001: Memory BAR 1 • 010: Memory BAR 2 • 011: Memory BAR 3 • 100: Memory BAR 4 • 101: Memory BAR 5 • 110: I/O BAR • 111: Expansion ROM BAR These outputs are valid when both `rx_st_sop_o` and `rx_st_valid_o` are asserted.	`coreclkout_hip`	EP/RP
x16: `rx_st_tlp_abort_o[1:0]` x8/x4: `rx_st_tlp_abort_o`	O	By default, the PCIe Hard IP drops an errored TLP (a malformed TLP, or a TLP with an ECRC error or tag/requester ID (RID) mismatches). The PCIe Hard IP asserts `rx_st_tlp_abort_o` to notify the application an errored TLP has been dropped.	`coreclkout_hip`	EP/RP
x16: `rx_st_data_par_o[63:0]` x8: `rx_st_data_par_o[31:0]` x4: `rx_st_data_par_o[15:0]`	O	Byte parity signals for `rx_st_data_o`. These parity signals are not available when ECC is enabled. *Note:* These signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
x16: rx_st_hdr_par_o[31:0] x8/x4: rx_st_hdr_par_o[15:0]	O	Byte parity signals for rx_st_hdr_o. These parity signals are not available when ECC is enabled. *Note:* These signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	coreclkout_hip	EP/RP/BP
x16: rx_st_tlp_prfx_par_o[7:0] x8/x4: rx_st_tlp_prfx_par_o[3:0]	O	Byte parity signals for rx_st_tlp_prfx_o. These parity signals are not available when ECC is enabled. *Note:* These signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	coreclkout_hip	EP/RP/BP
rx_par_err_o	O	Asserted for a single cycle to indicate that a parity error was detected in a TLP at the input of the RX buffer. This error is logged as an uncorrectable internal error in the VSEC registers. If this error occurs, you must reset the Hard IP because parity errors can leave the Hard IP in an unknown state. *Note:* This signal is not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	coreclkout_hip	EP/RP/BP

Figure 21. Avalon-ST RX Packet Interface in 1x16 Mode

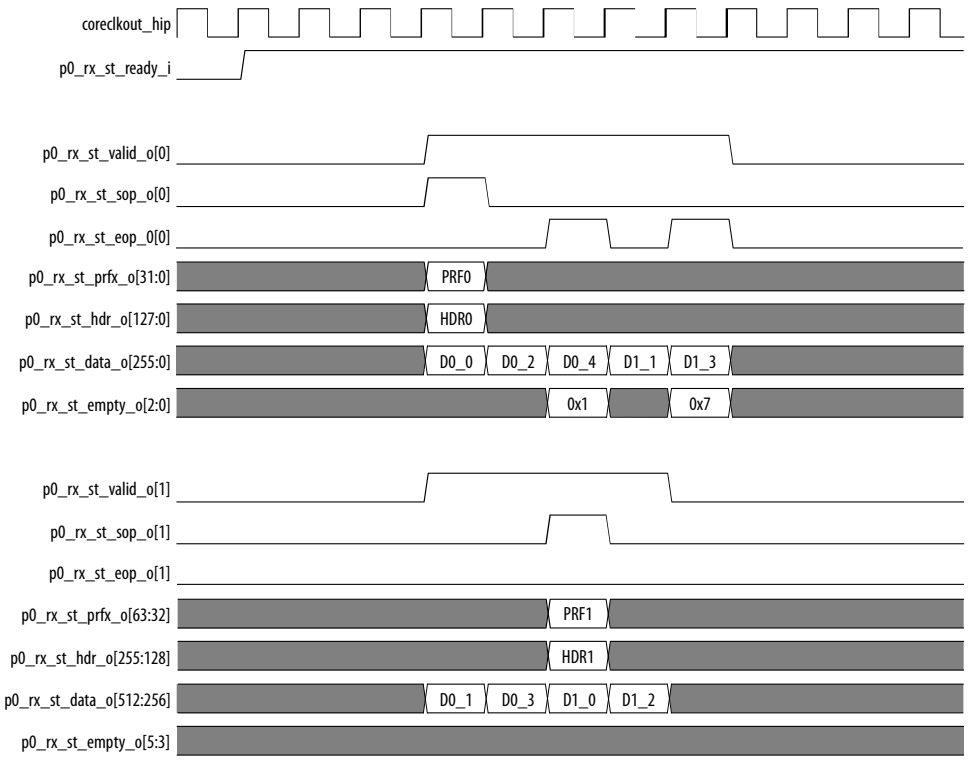
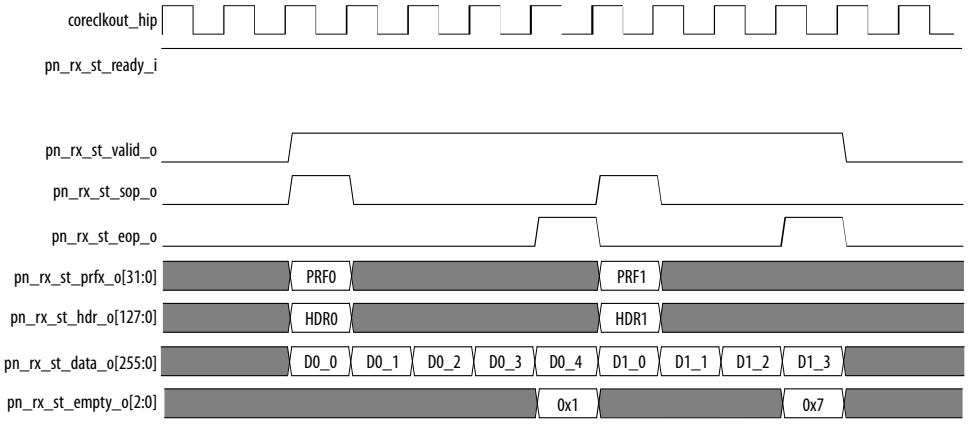
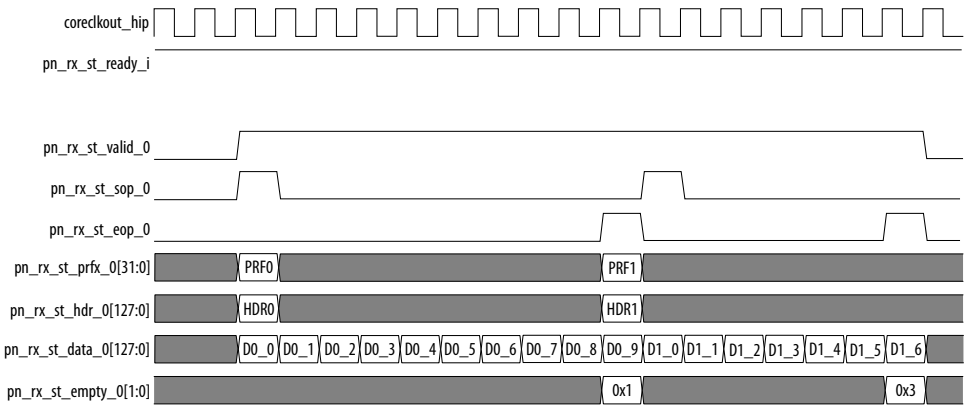


Figure 22. Avalon-ST RX Packet Interface in 2x8 Mode



Note: In 2x8 mode, the pn prefix in the signal names is p0 and p1 for the two x8 ports.

Figure 23. Avalon-ST RX Packet Interface in 4x4 Mode



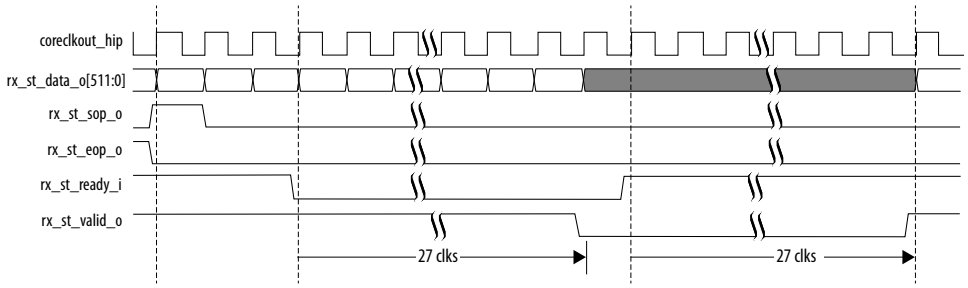
Note: In 4x4 mode, the pn prefix in the signal names is p0, p1, p2 and p3 for the four x4 ports.

Note: In the diagrams for the 1x16 or 2x8 modes, D0_0 represents a 256-bit block of data. However, in the diagram for the 4x4 mode, D0_0 represents a 128-bit block of data.

4.4.3. Avalon-ST RX Interface `rx_st_ready` Behavior

The following timing diagram illustrates the timing of the RX interface when the application throttles the P-Tile IP for PCIe by deasserting `rx_st_ready_i`. The Transaction Layer in the P-Tile IP deasserts `rx_st_valid_o` within 27 cycles of the `rx_st_ready_i` deassertion. It also reasserts `rx_st_valid_o` within 27 cycles after `rx_st_ready_i` reasserts if there is more data to send. `rx_st_data_o` is held until the application is able to accept it.

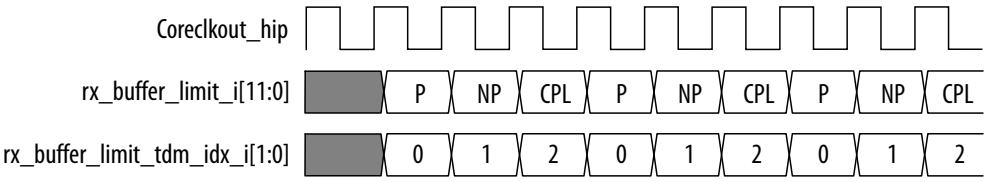
Figure 24. Avalon-ST RX Interface `rx_st_ready` Behavior



4.4.4. RX Flow Control Interface

The RX flow control interface provides information on the application's available RX buffer space to the PCIe Hard IP in a time-division multiplexing (TDM) manner. It reports the space available in units called flow control credits.

Figure 25. RX Flow Control Interface's TDM Reporting of Credit Limits



Flow control credits are available for the following TLP categories:

- Posted (P) transactions: TLPs that do not require a response.
- Non-posted (NP) transactions: TLPs that require a completion.
- Completions (CPL): TLPs that respond to non-posted transactions.

Table 34. Categorization of Transaction Types

TLP Type	Category
Memory Write	Posted
Memory Read	Non-posted
Memory Read Lock	
I/O Read	Non-posted
I/O Write	
Configuration Read	Non-posted
Configuration Write	
Message	Posted
Completion	Completion
Completion with Data	
Completion Lock	
Completion Lock with Data	
Fetch and Add AtomicOp	Non-posted

Table 35. RX Flow Control Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
`rx_buffer_limit_i[11:0]`	I	When the RX Flow Control Interface is enabled, the application can use these signals for TLP flow control. These signals indicate the application RX buffer space made available since reset/ initialization. Initially, the signals are set according to the buffer size (in terms of the number of TLPs the RX buffer can take). The value of these signals always increments and rolls over. For example, if the initial value is 0x0ff, the `rx_buffer_limit_i[11:0`	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		] value increments by 1 and rolls over to 0x000 when one received TLP exits the application RX buffer. If a TLP type is blocked due to a lack of the corresponding RX buffer space in the application layer, other TLP types may bypass it per the PCIe transaction ordering rules. Note that the initial value of rx_buffer_limit_i[11:0] cannot be larger than 2048 TLPs.		
rx_buffer_limit_tdm_idx_i[1:0]	I	These signals indicate the type of buffer for the corresponding rx_buffer_limit_i[11:0] signals. The Application Layer should provide the buffer limit information for all the enabled ports in a TDM manner. The following encodings are defined: • 00: buffer limit for P type of TLPs. • 01: buffer limit for NP type. • 10: buffer limit for CPL type. • 11: reserved.	coreclkout_hip	EP/RP/BP

Note: If you do not implement this interface, tie the rx_buffer_limit_i[11:0] and rx_buffer_limit_tdm_idx_i[1:0] signals to zero.

For more details on the usage of the scale factors, refer to Section 3.4.2 of the *PCI Express Base Specification, Rev. 4.0 Version 1.0*.

4.4.5. Avalon-ST TX Interface

The Application Layer transfers data to the Transaction Layer of the PCI Express IP core over the Avalon-ST TX interface. The Transaction Layer must assert tx_st_ready_o before transmission begins. Transmission of a packet must be uninterrupted when tx_st_ready_o is asserted.

This 512-bit interface supports two locations for the beginning of a TLP, bit[0] and bit[256]. The interface supports multiple TLPs per cycle only when an end-of-packet cycle occurs in the lower 256 bits.

Note: This interface supports two tx_st_sop_i signals and two tx_st_eop_i signals per cycle when the P-Tile IP is operating in a x16 configuration. It also does not follow a fixed latency between the tx_st_ready_o and tx_st_valid_i[1:0] signals as specified by the *Avalon Interface Specifications*. Data can be received any time within the defined readyLatency, which is three coreclkout_hip cycles.

The x16 core provides two segments with each one having 256 bits of data (`tx_st_data_i[511:256]` and `tx_st_data_i[255:0]`), 128 bits of header (`tx_st_hdr_i[255:128]` and `tx_st_hdr_i[127:0]`), and 32 bits of TLP prefix (`tx_st_tlp_prfx_i[63:32]` and `tx_st_tlp_prfx_i[31:0]`). If this core is configured in the 1x16 mode, both segments are used, so the data bus becomes a 512-bit bus `tx_st_data_i[511:0]`. The start of packet can appear in the upper segment or lower segment, as indicated by the `tx_st_sop_i[1:0]` signals.

If this core is configured in the 2x8 mode, only the lower segment is used. In this case, the data bus is a 256-bit bus `tx_st_data_i[255:0]`.

Finally, if this core is configured in the 4x4 mode, only the lower segment is used and only the LSB 128 bits of data are valid. In this case, the data bus is a 128-bit bus `tx_st_data_i[127:0]`.

The x8 core provides one segment with 256 bits of data, 128 bits of header and 32 bits of TLP prefix. If this core is configured in 4x4 mode, only the LSB 128 bits of data are used.

The x4 core provides one segment with 128 bits of data, 128 bits of header and 32 bits of TLP prefix.

Table 36. Avalon-ST TX Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
x16: `tx_st_data_i[511:0]` x8: `tx_st_data_i[255:0]` x16: `tx_st_data_i[127:0]`	I	Application Layer data for transmission. The Application Layer must provide a properly formatted TLP on the TX interface. Valid when the corresponding `tx_st_valid_i` signal is asserted. The mapping of message TLPs is the same as the mapping of Transaction Layer TLPs with 4-dword headers. The number of data cycles must be correct for the length and address fields in the header. Issuing a packet with an incorrect number of data cycles results in the TX interface hanging and becoming unable to accept further requests. *Note:* There must be no Idle cycle between the `tx_st_sop_i` and `tx_st_eop_i` cycles unless there is backpressure with the deassertion of `tx_st_ready_o`.	`coreclkout_hip`	EP/RP/BP
x16: `tx_st_sop_i[1:0]` x8/x4: `tx_st_sop_i`	I	Indicate the first cycle of a TLP when asserted in conjunction with the corresponding bit of `tx_st_valid_i`. For the x16 configuration: • `tx_st_sop_i[1]`: When asserted, indicates the start of a TLP in `tx_st_data_i[511:256]`. • `tx_st_sop_i[0]`: When asserted, indicates the start of a TLP in `tx_st_data_i[255:0]`.	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		These signals are asserted for one clock cycle per each TLP. They also qualify the corresponding `tx_st_hdr_i` and `tx_st_tlp_prfx_i` signals.		
x16: `tx_st_eop_i[1:0]` x8/x4: `tx_st_eop_i`	I	Indicate the last cycle of a TLP when asserted in conjunction with the corresponding bit of `tx_st_valid_i`. For the x16 configuration: • `tx_st_eop_i[1]`: When asserted, indicates the end of a TLP in `tx_st_data_i[511:256]`. • `tx_st_eop_i[0]`: When asserted, indicates the end of a TLP in `tx_st_data_i[255:0]`. These signals are asserted for one clock cycle per each TLP.	`coreclkout_hip`	EP/RP/BP
x16: `tx_st_valid_i[1:0]` x8/x4: `tx_st_valid_i`	I	Qualify the corresponding data segment of `tx_st_data_i` into the IP core on ready cycles. To facilitate timing closure, Intel recommends that you register both the `tx_st_ready_o` and `tx_st_valid_i` signals. *Note:* There must be no Idle cycle between the `tx_st_sop_i` and `tx_st_eop_i` cycles unless there is backpressure with the deassertion of `tx_st_ready_o`.	`coreclkout_hip`	EP/RP/BP
`tx_st_ready_o`	O	Indicates that the PCIe Hard IP is ready to accept data for transmission. The readyLatency is three cycles. If `tx_st_ready_o` is asserted by the Transaction Layer in the PCIe Hard IP on cycle <n>, then <n> + `readyLatency` is a ready cycle, during which the Application may assert `tx_st_valid_i` and transfer data. If `tx_st_ready_o` is deasserted by the Transaction Layer on cycle <n>, then the Application must deassert `tx_st_valid_i` within the `readyLatency` number of cycles after cycle <n>. `tx_st_ready_o` can be deasserted in the following conditions:	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		• The LTSSM is not ready. • A Retry is in progress. • There are not enough credits available to send the request. • The P-Tile Avalon-ST IP is busy sending internally generated TLPs. • The internal P-Tile TX FIFO is full.		
x16: `tx_st_err_i[1:0]` x8/x4: `tx_st_err_i`	I	When asserted, indicate an error in the transmitted TLP. These signals are asserted with `tx_st_eop_i` and nullify a packet. • `tx_st_err_i[1]`: When asserted, specifies an error in `tx_st_data_i[511:256]`. • `tx_st_err_i[0]`: When asserted, specifies an error in `tx_st_data_i[255:0]`.	`coreclkout_hip`	EP/RP/BP
x16: `tx_st_hdr_i[255:0]` x8/x4: `tx_st_hdr_i[127:0]`	I	This is the header to be transmitted, which follows the TLP header format of the PCIe specifications except for the requester ID/completer ID fields (`tx_st_hdr_i[95:80]`): • `tx_st_hdr_i[95:84]`: `tx_st_vf_num[11:0]` • `tx_st_hdr_i[83]`: `tx_st_vf_active` • `tx_st_hdr_i[82:80]`: `tx_st_func_num[2:0]` These signals are valid when the corresponding `tx_st_sop_i` signal is asserted. The header uses a Big Endian implementation.	`coreclkout_hip`	EP/RP/BP
x16: `tx_st_tlp_prfx_i[63:0]` x8/x4: `tx_st_tlp_prfx_i[31:0]`	I	This is the TLP prefix to be transmitted, which follows the TLP prefix format of the PCIe specifications. PASID is included. These signals are valid when the corresponding `tx_st_sop_i` signal is asserted. The TLP prefix uses a Big Endian implementation (i.e. the Fmt field is in bits [31:29] and the Type field is in bits [28:24]). If no prefix is present for a given TLP, that dword, including the Fmt field, is all zeros.	`coreclkout_hip`	EP/RP/BP
x16: `tx_st_data_par_i[63:0]` x8: `tx_st_data_par_i[31:0]` x4: `tx_st_data_par_i[15:0]`	I	Byte parity for `tx_st_data_i`. Bit [0] corresponds to `tx_st_data_i[7:0]`, bit [1] corresponds to `tx_st_data_i[15:8]`, and so on. By default, the PCIe Hard IP generates the parity for the TX data. However, when ECC is off, the parity	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		can be passed in from the FPGA core by setting the k_pcie_parity_bypass register. *Note:* These signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.		
x16: tx_st_hdr_par_i[31:0] x8/x4: tx_st_hdr_par_i[15:0]	I	Byte parity for tx_st_hdr_i. By default, the PCIe Hard IP generates the parity for the TX header. However, when ECC is off, the parity can be passed in from the FPGA core by setting the k_pcie_parity_bypass register. *Note:* These signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	coreclkout_hip	EP/RP/BP
x16: tx_st_tlp_prfx_par_i[7:0] x8/x4: tx_st_tlp_prfx_par_i[3:0]	I	Byte parity for tx_st_tlp_prfx_i. By default, the PCIe Hard IP generates the parity for the TX TLP prefix. However, when ECC is off, the parity can be passed in from the FPGA core by setting the k_pcie_parity_bypass register. *Note:* These signals are not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	coreclkout_hip	EP/RP/BP
tx_par_err_o	O	Asserted for a single cycle to indicate a parity error during TX TLP transmission. The IP core transmits TX TLP packets even when a parity error is detected. *Note:* This signal is not functional in the Intel Quartus Prime 19.4 release, but will be in a future release.	coreclkout_hip	EP/RP/BP

Figure 26. Avalon-ST TX Packet Interface in 1x16 Mode

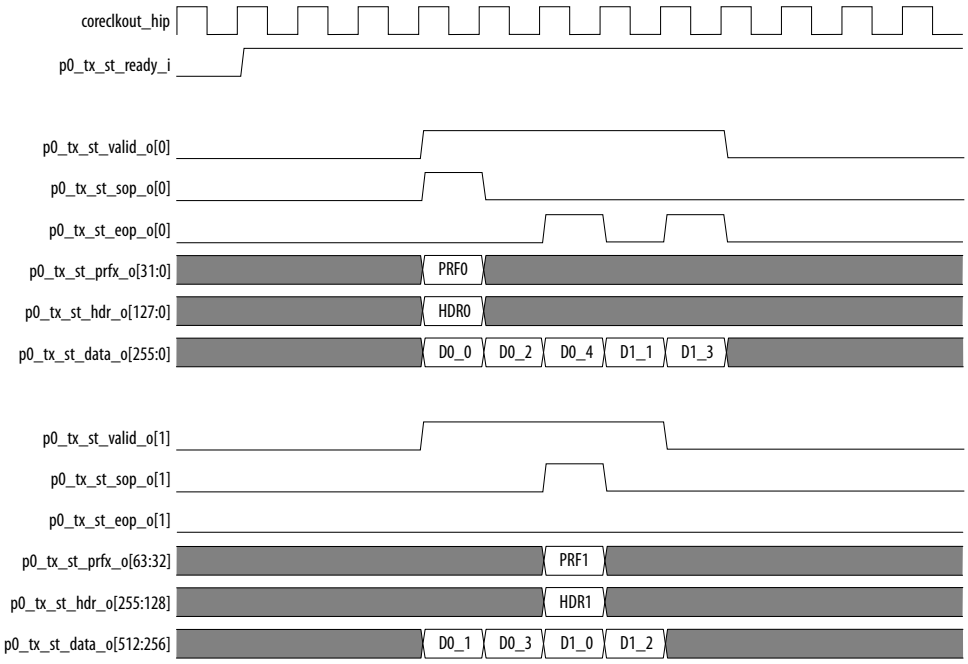
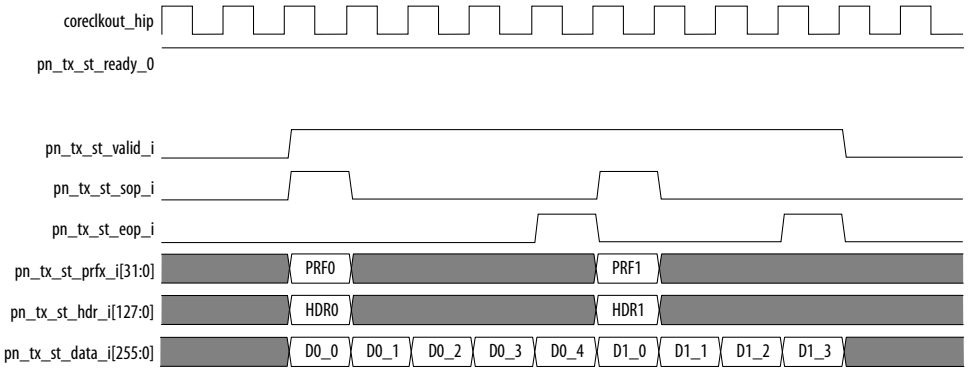
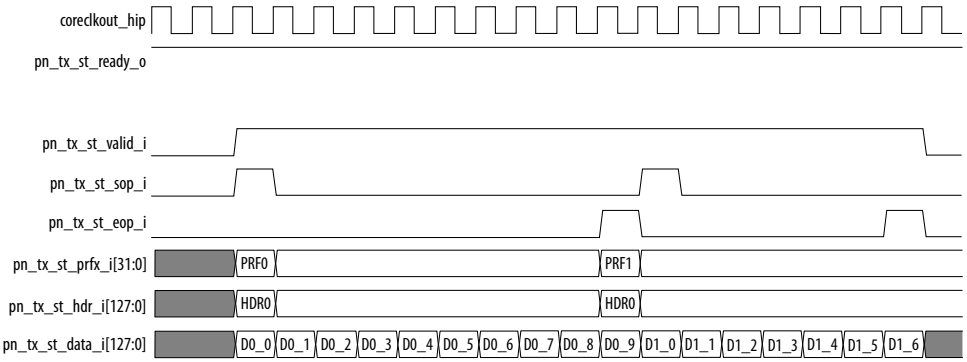


Figure 27. Avalon-ST TX Packet Interface in 2x8 Mode



Note: In 2x8 mode, the pn prefix in the signal names is p0 and p1 for the two x8 ports.

Figure 28. Avalon-ST TX Packet Interface in 4x4 Mode



Note: In 4x4 mode, the pn prefix in the signal names is p0, p1, p2 and p3 for the four x4 ports.

Note: In the diagrams for the 1x16 or 2x8 modes, D0_0 represents a 256-bit block of data. However, in the diagram for the 4x4 mode, D0_0 represents a 128-bit block of data.

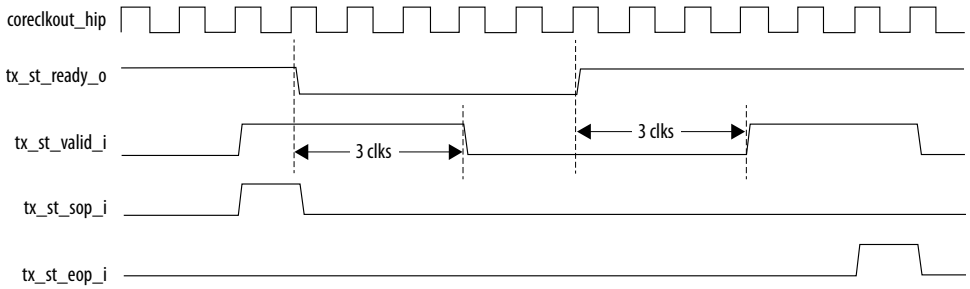
4.4.6. Avalon-ST TX Interface `tx_st_ready` Behavior

The following timing diagram illustrates the behavior of `tx_st_ready_o`, which is deasserted to pause the data transmission to the Transaction Layer of the P-Tile IP for PCIe, and then reasserted. The timing diagram shows a readyLatency of three cycles. The application deasserts `tx_st_valid_i` three cycles after `tx_st_ready_o` is deasserted.

The application must not deassert `tx_st_valid_i` between `tx_st_sop_i` and `tx_st_eop_i` on a ready cycle. For the definition of a ready cycle, refer to the *Avalon Interface Specifications*.

Note: This is an additional requirement for the P-Tile IP for PCIe that is not compliant to the Avalon-ST standard.

Figure 29. Avalon-ST TX Interface `tx_st_ready` Behavior



4.4.7. TX Flow Control Interface

Before a TLP can be transmitted, flow control logic verifies that the link partner's RX port has sufficient buffer space to accept it. The TX Flow Control interface reports the link partner's available RX buffer space to the Application. It reports the space available in units called Flow Control credits for posted, non-posted and completion TLPs (as defined in the *RX Flow Control Interface* section).

TX credit limit signals are provided in a TDM manner similar to how the RX credit limit signals are provided.

Figure 30. TX Flow Control Interface's TDM Reporting of Credit Limits

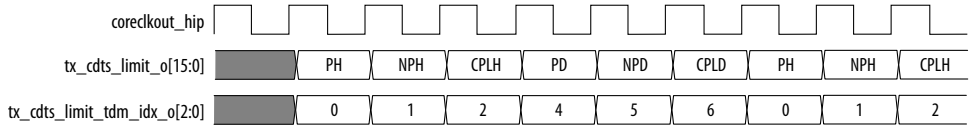


Table 37. TX Flow Control Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
`tx_cdts_limit_o[15:0]`	O	Indicate the Flow Control (FC) credit units advertised by the remote Receiver. These signals represent the total number of FC credits made available by the Receiver since Flow Control initialization. Initially, these signals indicate the number of FC credits available in the remote Receiver. The value of these signals always increments and rolls over. For example, if the remote Receiver advertises an initial Non-Posted Header (NPH) FC credit of 0xfff, after it receives a MRd request, the NPH FC credits value increments by 1 and rolls over to 0x000. The `tx_cdts_limit_tdm_idx_o[2:0]` signals determine the traffic type. When the traffic type is header credit, only the LSB 12 bits are valid. Note that, in addition to the TLPs transmitted by the user application,	`coreclkout_hip`	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		internally generated TLPs also consume FC credits.		
tx_cdts_limit_tdm_idx_o[2:0]	O	Indicate the traffic type for the tx_cdts_limit_o[15:0] signals. This interface provides credit limit information for all enabled ports in a TDM manner. The following encodings are defined: • 000: P header credit limit • 001: NP header credit limit • 010: CPL header credit limit • 011: reserved • 100: P data credit limit • 101: NP data credit limit • 110: CPL data credit limit • 111: reserved	coreclkout_hip	EP/RP/BP

4.4.8. Tag Allocation

The P-Tile PCIe Hard IP supports the 10-bit tag Requester capability in the x16 Controller (Port 0) only. It supports up to 512 outstanding Non-Posted Requests (NPRs) with valid tag values ranging from 256 to 767.

The x8 (Port 1) and x4 Controllers (Port 2/3) don't support the 10-bit tag Requester capability, although they support the 10-bit Completer capability.

Both x8 and x4 Controllers can allow up to 256 outstanding NPRs with valid tag values ranging from 0 to 255.

When enabling both 10-bit tags and 8-bit tags, the LSB 8 bits of the 8-bit tags cannot be shared with the LSB 8 bits of the 10-bit tags. For example, if you want to use 64 tags as 8-bit tags and the rest of the tags as 10-bit tags, you can partition the tag space as follows:

- 8-bit tags : 0 - 63
- 10-bit tags : 320 - 511, 576 - 767

Note that all PFs and their associated VFs share the same tag space. This means that different PFs and VFs cannot have outstanding tags having the same tag values.

4.5. Hard IP Status Interface

This interface includes the signals that are useful for debugging, such as the link status signal, LTSSM state outputs, etc. These signals are available when the optional Power Management interface is enabled.

Table 38. Hard IP Status Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
link_up_o	O	When asserted, this signal indicates the link is up.	coreclkout_hip	EP/RP/BP
dl_up_o	O	When asserted, this signal indicates the Data Link (DL) Layer is active.	coreclkout_hip	EP/RP/BP
ltssm_state_o[5:0]	O	Indicates the LTSSM state: • 6'h00: S_DETECT_QUIET • 6'h01: S_DETECT_ACT • 6'h02: S_POLL_ACTIVE • 6'h03: S_POLL_COMPLIANCE • 6'h04: S_POLL_CONFIG • 6'h05: S_PRE_DETECT_QUIET • 6'h06: S_DETECT_WAIT • 6'h07: S_CFG_LINKWD_START • 6'h08: S_CFG_LINKWD_ACCEPT • 6'h09: S_CFG_LANENUM_WAIT • 6'h0A: S_CFG_LANENUM_ACCEPT • 6'h0B: S_CFG_COMPLETE • 6'h0C: S_CFG_IDLE • 6'h0D: S_RCVRY_LOCK • 6'h0E: S_RCVRY_SPEED • 6'h0F: S_RCVRY_RCVRCFG • 6'h10: S_RCVRY_IDLE • 6'h11: S_L0 • 6'h12: S_L0S • 6'h13: S_L123_SEND_EIDLE • 6'h14: S_L1_IDLE • 6'h15: S_L2_IDLE • 6'h16: S_L2_WAKE • 6'h17: S_DISABLED_ENTRY • 6'h18: S_DISABLED_IDLE • 6'h19: S_DISABLED • 6'h1A: S_LPBK_ENTRY • 6'h1B: S_LPBK_ACTIVE • 6'h1C: S_LPBK_EXIT • 6'h1D: S_LPBK_EXIT_TIMEOUT • 6'h1E: S_HOT_RESET_ENTRY • 6'h1F: S_HOT_RESET • 6'h20: S_RCVRY_EQ0 • 6'h21: S_RCVRY_EQ1 • 6'h22: S_RCVRY_EQ2 • 6'h23: S_RCVRY_EQ3	coreclkout_hip	EP/RP/BP

4.6. Interrupt Interface

The P-Tile Avalon-ST IP for PCI Express supports Message Signaled Interrupts (MSI), MSI-X interrupts, and legacy interrupts. MSI and legacy interrupts are mutually exclusive.

The user application generates MSI which are single-Dword memory write TLPs to implement interrupts. This interrupt mechanism conserves pins because it does not use separate wires for interrupts. In addition, the single Dword provides flexibility for the data presented in the interrupt message. The MSI Capability structure is stored in the Configuration Space and is programmed using Configuration Space accesses.

The user application generates MSI-X messages which are single-Dword memory writes. The MSI-X Capability structure points to an MSI-X table structure and an MSI-X Pending Bit Array (PBA) structure which are stored in memory. This scheme is different than the MSI Capability structure, which contains all the control and status information for the interrupts.

Enable legacy interrupts by programming the `Interrupt Disable` bit (bit[10]) of the `Configuration Space Command` to 1'b0. When legacy interrupts are enabled, the IP core emulates INTx interrupts using virtual wires. The `app_int_i` ports control legacy interrupt generation.

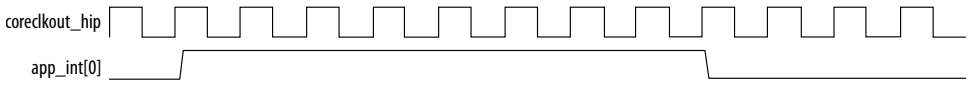
4.6.1. Legacy Interrupts

Legacy interrupts mimic the original PCI level-sensitive interrupts using *virtual wire* messages. The P-tile IP for PCIe signals legacy interrupts on the PCIe link using Message TLPs. The term INTx refers collectively to the four legacy interrupts, INTA#,INTB#, INTC# and INTD#. The P-tile IP for PCIe asserts `app_int_i` to cause an Assert_INTx Message TLP to be generated and sent upstream. A deassertion of `app_int_i`, i.e a transition of this signal from high to low, causes a Deassert_INTx Message TLP to be generated and sent upstream. To use legacy interrupts, you must clear the Interrupt Disable bit, which is bit 10 of the Command Register in the configuration header. Then, you must turn off the MSI Enable bit.

Table 39. Legacy Interrupt Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
x16/x8: `app_int_i[7:0]` x4: NA	I	When asserted, these signals indicate an assertion of an INTx message is requested. A transition from high to low indicates a deassertion of the INTx message is requested. This bus is for EP only. Each bit is associated with a corresponding physical function. These signals must be asserted for at least 8 cycles.	`coreclkout_hip`	EP
`int_status_o[7:0]`	O	These signals drive legacy interrupts to the Application Layer in Root Port mode. The source of the interrupt will be logged in the Root Port Interrupt Status registers in the Port Configuration and Status registers.	`coreclkout_hip`	RP

Figure 31. Generating an Assert_INTx Message TLP Using the `app_int_i` Signal



`app_int_i[0]` is asserted for at least eight clock cycles to cause an Assert_INTx Message TLP to be generated and sent upstream for physical function 0. For a multi-functions implementation, `app_int_i[0]` is for physical function 0, `app_int_i[1]` is for physical function 1 and so on. Deasserting an `app_int_i` signal by driving it from high to low causes a Deassert_INTx Message TLP to be generated and sent upstream.

4.6.2. MSI

MSI interrupts are signaled on the PCI Express link using a single dword Memory Write TLP. The user application issues an MSI request (MWr) through the Avalon-ST interface and updates the configuration space register using the MSI interface.

For more details on the MSI Capability Structure, refer to Figure 82 on page 184.

The Mask Bits register and Pending Bits register are 32 bits in length each, with each potential interrupt message having its own mask bit and pending bit. If bit[0] of the Mask Bits register is set, interrupt message 0 is masked. When an interrupt message is masked, the MSI for that vector cannot be sent. If software clears the mask bit and the corresponding pending bit is set, the function must send the MSI request at that time.

You should obtain the necessary MSI information (such as the message address and data) from the configuration output interface (`tl_cfg_*`) to create the MWr TLP in the format shown below to be sent via the Avalon-ST interface.

Figure 32. Creating a MWr TLP for an MSI Request

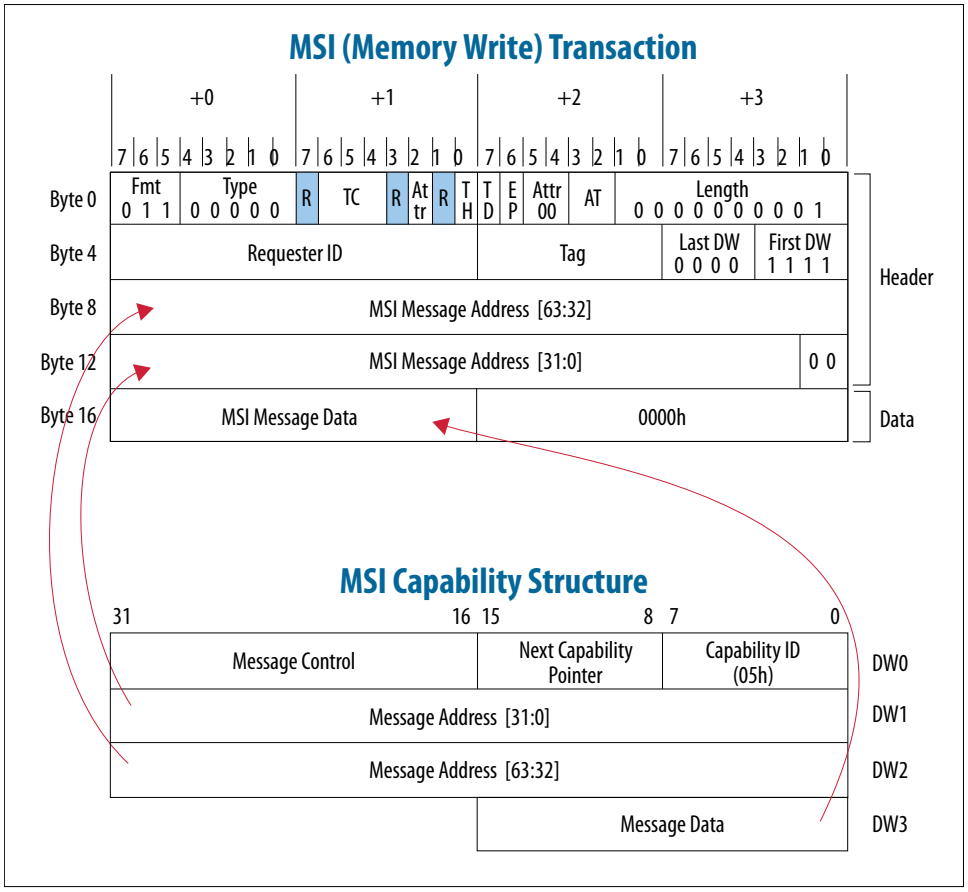
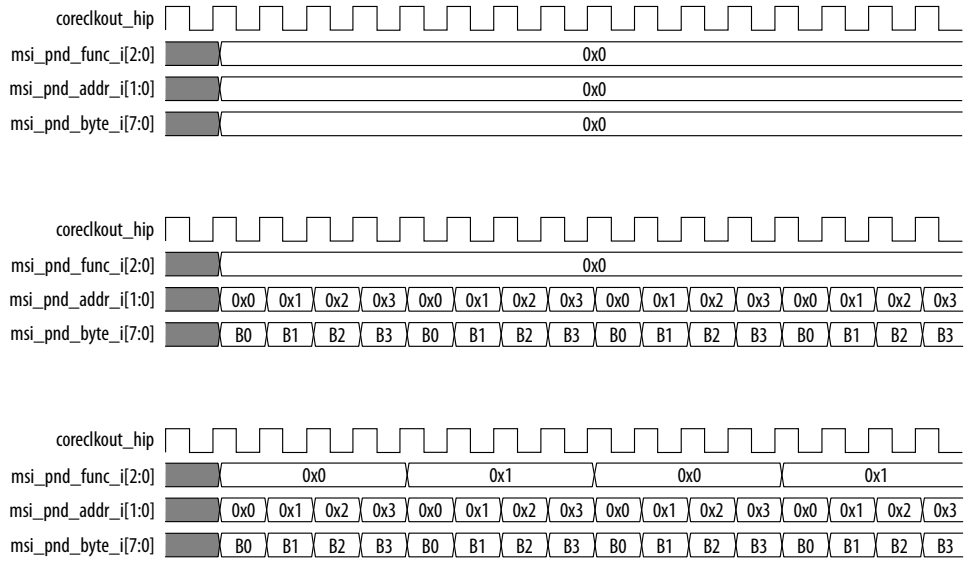


Table 40. MSI Pending Bits Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
msi_pnd_func_i[2:0]	I	Function number select for the Pending Bits register in the MSI capability structure.	coreclkout_hip	EP
msi_pnd_addr_i[1:0]	I	Byte select for Pending Bits Register in the MSI Capability Structure. For example if msi_pnd_addr_i[1:0] = 00, bits [7:0] of the Pending Bits register will be updated with msi_pnd_byte_i[7:0]. If msi_pnd_addr_i[1:0] = 01, bits [15:8] of the Pending Bits register will be updated with msi_pnd_byte_i[7:0].	coreclkout_hip	EP
msi_pnd_byte_i[7:0]	I	Indicate that function has a pending associated message.	coreclkout_hip	EP

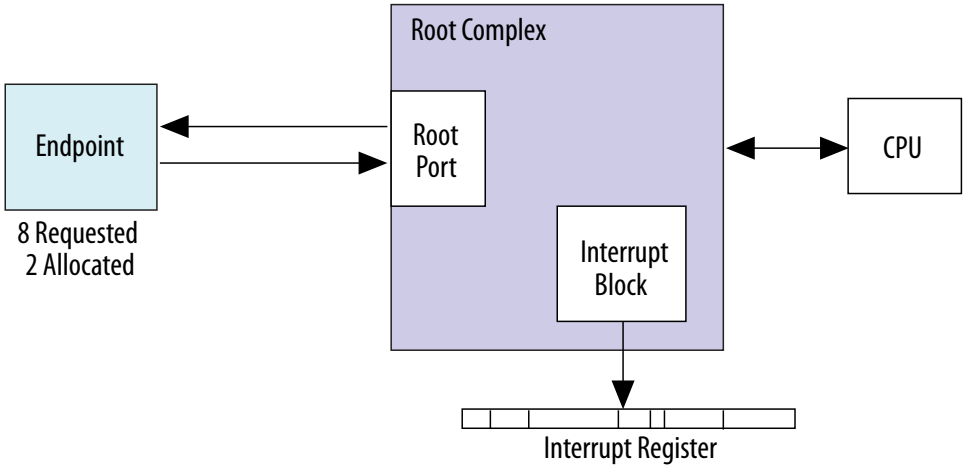
The following figure shows the timings of msi_pnd_* signals in three scenarios. The first scenario shows the case when the MSI pending bits register is not used. The second scenario shows the case when only physical function 0 is enabled and the MSI pending bits register is used. The last scenario shows the case when four physical functions are enabled and the MSI pending bits register is used.

Figure 33. Example Timing Diagrams for msi_pnd* Signals



There are 32 possible MSI messages. The number of messages requested by a particular component does not necessarily correspond to the number of messages allocated. For example, in the following figure, the Endpoint requests eight MSIs but is only allocated two. In this case, you must design the Application Layer to use only two allocated messages.

Figure 34. MSI Request Example



The following table describes three example implementations. The first example allocates all 32 MSI messages. The second and third examples only allocate 4 interrupts.

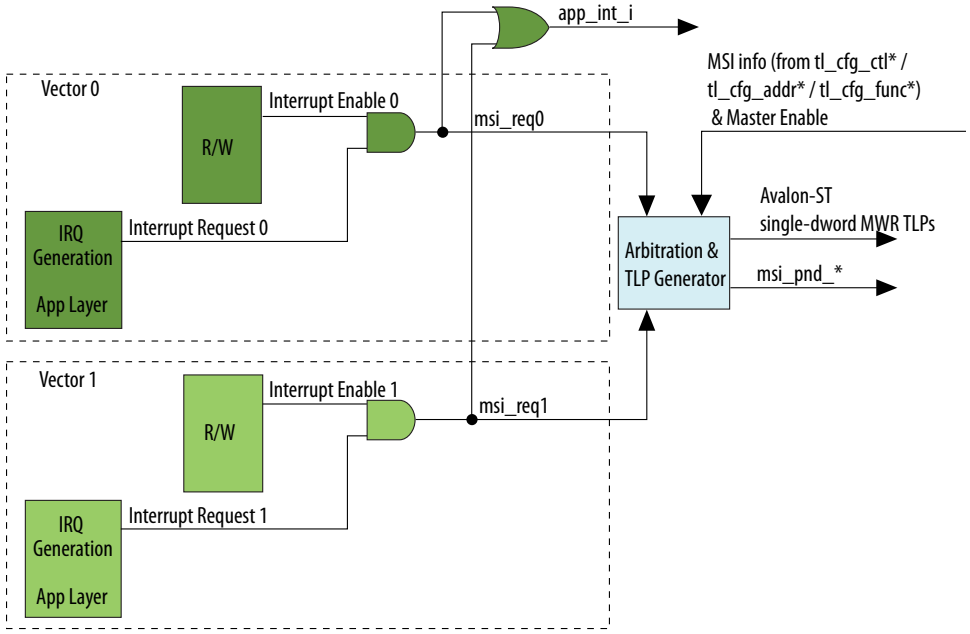
Table 41. MSI Messages Requested, Allocated, and Mapped

MSI	Allocated		
	32	4	4
System Error	31	3	3
Hot Plug and Power Management Event	30	2	3
Application Layer	29:0	1:0	2:0

MSI interrupts generated for Hot Plug, Power Management Events, and System Errors always use Traffic Class 0. MSI interrupts generated by the Application Layer can use any Traffic Class. For example, a DMA that generates an MSI at the end of a transmission can use the same traffic control as was used to transfer data.

The following figure illustrates a possible implementation of the Interrupt Handler Module with a per vector enable bit in the Application Layer. Alternatively, the Application Layer could implement a global interrupt enable instead of this per vector MSI.

Figure 35. Example Implementation of the Interrupt Handler Block



Related Information

Handling PCIe Legacy and MSI Interrupts

4.6.3. MSI-X

The P-Tile IP for PCIe provides a Configuration Intercept Interface. User soft logic can monitor this interface to get MSI-X Enable and MSI-X function mask related information. User application logic needs to implement the MSI-X tables for all PFs and VFs at the memory space pointed to by the BARs as a part of your Application Layer.

For more details on the MSI-X related information that you can obtain from the Configuration Intercept Interface, refer to the *MSI-X Registers* section in the *Registers* chapter.

MSI-X is an optional feature that allows the user application to support large amount of vectors with independent message data and address for each vector.

When MSI-X is supported, you need to specify the size and the location (BARs and offsets) of the MSI-X table and PBA. MSI-X can support up to 2048 vectors per function versus 32 vectors per function for MSI.

A function is allowed to send MSI-X messages when MSI-X is enabled and the function is not masked. The application uses the Configuration Output Interface (address 0x0C bit[5:4]) or Configuration Intercept Interface to access this information.

When the application needs to generate an MSI-X, it will use the contents of the MSI-X Table (Address and Data) and generate a Memory Write through the Avalon-ST interface.

You can enable MSI-X interrupts by turning on the **Enable MSI-X** option under the **PCI Express/PCI Capabilities** tab in the parameter editor. If you turn on the **Enable MSI-X** option, you should implement the MSI-X table structures at the memory space pointed to by the BARs as a part of your Application Layer.

The MSI-X Capability Structure contains information about the MSI-X Table and PBA Structure. For example, it contains pointers to the bases of the MSI-X Table and PBA Structure, expressed as offsets from the addresses in the function's BARs. The Message Control register within the MSI-X Capability Structure also contains the MSI-X Enable bit, the Function Mask bit, and the size of the MSI-X Table. For a picture of the MSI-X Capability Structure, refer to Figure 84 on page 185.

MSI-X interrupts are standard Memory Writes, therefore Memory Write ordering rules apply.

Example:

Table 42. MSI-X Configuration

MSI-X Vector	MSI-X Upper Address	MSI-X Lower Address	MSI-X Data
0	0x00000001	0xAAAA0000	0x00000001
1	0x00000001	0xBBBB0000	0x00000002
2	0x00000001	0xCCCC0000	0x00000003

Table 43. PBA Table

PBA Table	PBA Entries
Offset 0	0x0

If the application needs to generate an MSI-X interrupt (vector 1), it will read the MSI-X Table information, generate a MWR TLP through the Avalon-ST interface and assert the corresponding PBA bits (bit[1]) in a similar fashion as for MSI generation.

The generated TLP will be sent to address 0x00000001_BBBB0000 and the data will be 0x00000002. When the MSI-X has been sent, the application can clear the associated PBA bits.

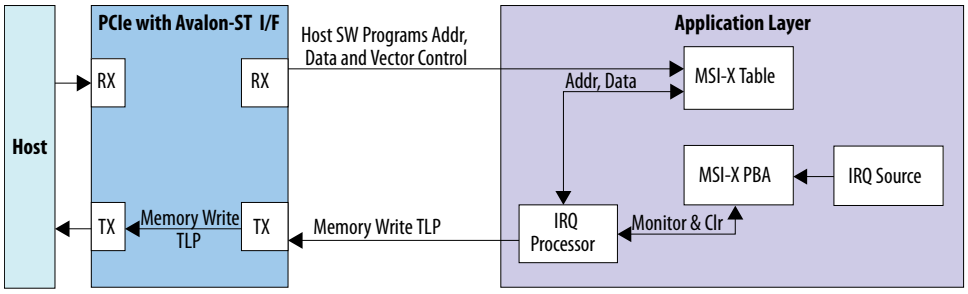
Related Information

Implementing MSI-X for PCI Express in Intel FPGA Devices

4.6.3.1. Implementing MSI-X Interrupts

Section 6.8.2 of the *PCI Local Bus Specification* describes the MSI-X capability and table structures. The MSI-X capability structure points to the MSI-X Table structure and MSI-X Pending Bit Array (PBA) registers. The BIOS sets up the starting address offsets and BAR associated with the pointer to the starting address of the MSI-X Table and PBA registers.

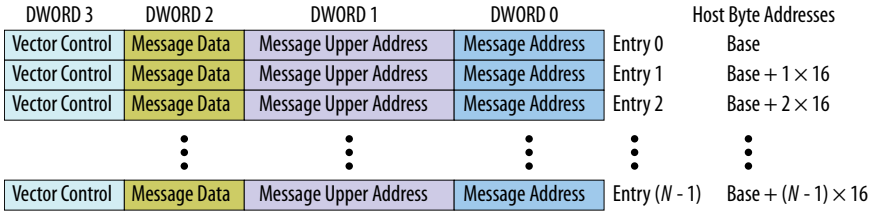
Figure 36. MSI-X Interrupt Components



1. Host software sets up the MSI-X interrupts in the Application Layer by completing the following steps:
   a. Host software reads the `Message Control` register at 0x050 register to determine the MSI-X Table size. The number of table entries is the *<value read> + 1*.

      The maximum table size is 2048 entries. Each 16-byte entry is divided in 4 fields as shown in the figure below. For multi-function variants, BAR4 accesses the MSI-X table. For all other variants, any BAR can access the MSI-X table. The base address of the MSI-X table must be aligned to a 4 KB boundary.
   b. The host sets up the MSI-X table. It programs MSI-X address, data, and masks bits for each entry as shown in the figure below.

Figure 37. Format of MSI-X Table



   c. The host calculates the address of the <n^{th}> entry using the following formula:

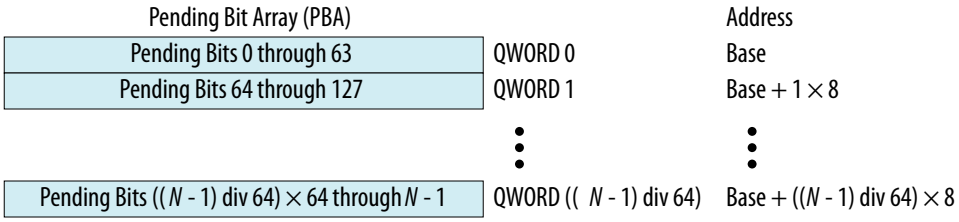
```
nth_address = base address[BAR] + 16<n>
```

2. When Application Layer has an interrupt, it drives an interrupt request to the IRQ Source module.
3. The IRQ Source sets appropriate bit in the MSI-X PBA table.

The PBA can use qword or dword accesses. For qword accesses, the IRQ Source calculates the address of the <m^{th}> bit using the following formulas:

```
qword address = <PBA base addr> + 8(floor(<m>/64))
qword bit = <m> mod 64
```

Figure 38. MSI-X PBA Table



4. The IRQ Processor reads the entry in the MSI-X table.
   a. If the interrupt is masked by the `Vector_Control` field of the MSI-X table, the interrupt remains in the pending state.
   b. If the interrupt is not masked, IRQ Processor sends Memory Write Request to the TX slave interface. It uses the address and data from the MSI-X table. If **`Message Upper Address`** = 0, the IRQ Processor creates a three-dword header. If the **`Message Upper Address`** > 0, it creates a 4-dword header.
5. The host interrupt service routine detects the TLP as an interrupt and services it.

Related Information

- Floor and ceiling functions
- PCI Local Bus Specification, Rev. 3.0

4.7. Error Interface

This is an optional interface in the Intel P-Tile Avalon-ST IP for PCI Express that allows the Application Layer to report errors to the IP core and vice versa. Specifically, the Application Layer can report the different types of errors defined by the `app_error_info_i` signal to the IP. For Advanced Error Reporting (AER), the Application Layer can provide the information to log the TLP header and the error log request via the `app_err_*` interface.

Note: The Intel P-Tile Avalon-ST IP for PCI Express enables the AER capability for Physical Functions (PFs) by default. There is no AER implementation for Virtual Functions (VFs). Use the VF Error Flag Interface instead of AER when using VFs.

Table 44. Error Interface Signals

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
`serr_out_o`	O	Indicates system error is detected. RP mode: A one-clock-cycle pulse on this signal indicates if any device in the hierarchy reports any of the following errors and the associated enable bit is set in	coreclkout_hip	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		the Root Control register: ERR_COR, ERR_FATAL, ERR_NONFATAL. Also asserted when an internal error is detected. The source of the error will be logged in the Root Port Error Status registers in the Port Configuration and Status registers. EP mode: Asserted when the P-Tile PCIe Hard IP sends a message of correctable/non-fatal/fatal error. BP mode: The transaction layer or data link layer errors detected by the Hard IP core trigger this signal. Detailed information are logged in the Bypass Mode Error Status registers in the Port Configuration and Status registers.		
hip_enter_err_mode_o	O	Asserted when the Hard IP enters the error mode. This usually happens when the Hard IP detects an uncorrectable RAM ECC error. Upon seeing the assertion of this signal, you should discard all the TLPs received.	coreclkout_hip	EP/RP/BP
app_err_valid_i	I	A one-cycle pulse on this signal indicates that the data on app_err_info, app_err_hdr, and app_err_func_num are valid in that cycle and app_err_hdr_i will be valid during the following four cycles.	coreclkout_hip	EP/RP
app_err_hdr_i[31:0]	I	This bus contains the header and TLP prefix information for the error TLP. The 128-bit header and 32-bit TLP prefix are sent to the Hard IP over five cycles (32 bits of information are sent in each clock cycle). Cycle 1 : header[31:0] Cycle 2 : header[63:32] Cycle 3 : header[95:64] Cycle 4 : header[127:96] Cycle 5 : TLP prefix	coreclkout_hip	EP/RP
app_err_info_i[12:0]	I	This error bus carries the following information: • [0]: Malformed TLP • [1]: Receiver overflow • [2]: Unexpected completion • [3]: Completer abort • [4]: Completion timeout • [5]: Unsupported request • [6]: Poisoned TLP received	coreclkout_hip	EP/RP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		• [7]: AtomicOp egress blocked • [8]: Uncorrectable internal error • [9]: Correctable internal error • [10]: Advisory error • [11]: TLP prefix blocked • [12]: ACS violation		
x16/x8: `app_err_func_num_i[2:0]` x4: NA	I	This bus contains the function number for the function that asserts the error valid signal.	coreclkout_hip	EP/RP

4.7.1. Completion Timeout Interface

The P-Tile IP for PCIe features a Completion timeout mechanism to keep track of Non-Posted requests sent by the user application and the corresponding Completions received. When the P-Tile IP detects a Completion timeout, it notifies the user application by asserting the `cpl_timeout_o` signal.

When a Completion timeout happens, the user application can use the Avalon-MM Completion Timeout Interface (for each port) to access the Completion timeout FIFO in the Hard IP to get more detailed information about the event and update the AER capability registers if required. After the completion timeout FIFO becomes empty, the IP core deasserts the `cpl_timeout_o` signal.

The `cpl_timeout_avmm` interface is synchronized to the `cpl_timeout_avmm_clk_i` clock.

Example:

When `cpl_timeout_o` is asserted, the user application can issue an Avalon-MM Read to retrieve information from the Completion FIFO. Then, it can issue an Avalon-MM Write to write 1 to bit[0] of the CONTROL register to get access to the next data.

Table 45. Completion Timeout Interface

Signal Name	Direction	Description	Clock domain	EP/RP/BP
`cpl_timeout_o`	O	Indicates the event that the completion TLP for a request has not been received within the expected time window. The IP core asserts this signal as long as the completion timeout FIFO in the Hard IP is not empty. You can obtain more details about the completion timeout event by looking at the signals on the	coreclkout_hip	EP/RP/BP

continued...

Signal Name	Direction	Description	Clock domain	EP/RP/BP
		completion timeout Avalon-MM interface (listed below).		
cpl_timeout_avmm_read_i	I	Avalon-MM read enable.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_readdata_o[7:0]	O	Avalon-MM read data outputs.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_readdata_valid_o	O	This signal qualifies the cpl_timeout_avmm_readdata_o signals into the Application Layer.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_write_i	I	Avalon-MM write enable.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_writedata_i[7:0]	I	Avalon-MM write data inputs.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_addr_i[20:0]	I	Avalon-MM address inputs. [20:3] : Reserved. Tie them to 0. [2:0] : Address for the FIFO register. Refer to the address map table below for more details.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_waitrequest_o	O	When asserted, this signal indicates the IP core is not ready to take any request.	cpl_timeout_avmm_clk_i	EP/RP/BP
cpl_timeout_avmm_clk_i	I	Avalon-MM clock. 50 MHz - 125 MHz (Range) 100 MHz (Recommended)		EP/RP/BP

Note: The Completion Timeout Interface has a separate address map that is isolated from other address maps.

Table 46. Address Map for the Completion Timeout Interface

Address	Name	Access Type	Description
0x0	STATUS	RO	[7:2] : Reserved [1] : Completion timeout FIFO full [0] : Completion timeout FIFO empty
0x1	CONTROL	WO	[7:1] : Reserved [0] : Read (popping data from the FIFO). You need to read all the information regarding the timed out request before writing 1 to bit 0 of the CONTROL

continued...

Address	Name	Access Type	Description
			register. Writing to bit 0 of the CONTROL register makes the next data appear.
0x2	VF	RO	[7:0] : vfunc_num[7:0] Virtual Function number for the VF that initiates the non-posted transaction for which the completion timeout is observed.
0x3	PF	RO	[7] : vfunc_active [6] : Reserved [5:3] : func_num[2:0] Physical function number (least significant 8 bits) for the PF that initiates the non-posted transaction for which the completion timeout is observed. [2:0] : vfunc_num[10:8] Virtual Function number (most significant 3 bits) for the VF that initiates the memory read request for which the completion timeout is observed.
0x4	LEN1	RO	[7:0] : cpl_lenn[7:0] Transfer length in bytes (least significant 8 bits), of the expected completion that timed out for the non-posted transaction. For a split completion, it indicates the number of bytes remaining to be delivered when the completion timed out (Max length is Max Read request size. Ex: 4K Bytes = 2^12 bytes)
0x5	LEN2	RO	[7:4] : Reserved [3:0] : cpl_lenn[11:8] Transfer length in bytes (most significant 4 bits), of the expected completion that timed out for the non-posted transaction. For a split completion, it indicates the number of bytes remaining to be delivered when the completion timed out (Max length is Max Read request size. Ex: 4K Bytes = 2^12 bytes)
0x6	TAG1	RO	[7:0] : cpl_tag[7:0] Tag ID (least significant 8 bits) of the expected completion that timed out for the non-posted transaction.
0x7	TAG2	RO	[7:5] : cpl_tc[2:0]

Address	Name	Access Type	Description
			Traffic class of the expected completion that timed out for the non-posted transaction. [4:3] : cpl_attr[1:0] Attribute of the expected completion that timed out for the non-posted transaction. ID based ordering is not supported. [4] -> Relaxed ordering, [3] -> No Snoop [2] : Reserved [1:0]: cpl_tag[9:8] Tag ID (most significant 2 bits) of the expected completion that timed out for the non-posted transaction.

4.8. Hot Plug Interface (RP Only)

Hot Plug support means that the device can be added to or removed from a system during runtime. The Hot Plug Interface in the P-Tile IP for PCIe allows an Intel FPGA with this IP to safely provide this capability.

This section describes the signals reported by the on-board hot plug components in the Downstream Port. This interface is available only if the `Slot Status Register` of the `PCI Express Capability Structure` is enabled.

Refer to the `Slot Status Register` of the `PCI Express Capability Structure` for additional information.

Table 47. Hot Plug Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
`sys_atten_button_pressed_i`	I	Attention Button Pressed. Indicates that the system attention button was pressed, and sets the Attention Button Pressed bit in the `Slot Status Register`.	`coreclkout_hip`	RP
`sys_pwr_fault_det_i`	I	Power Fault Detected. Indicates the power controller detected a power fault at this slot.	`coreclkout_hip`	RP
`sys_mrl_sensor_chged_i`	I	MRL Sensor Changed. Indicates that the state of the MRL sensor has changed.	`coreclkout_hip`	RP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
sys_pre_det_chged_i	I	Presence Detect Changed. Indicates that the state of the card presence detector has changed.	coreclkout_hip	RP
sys_cmd_cpled_int_i	I	Command Completed Interrupt. Indicates that the Hot Plug controller completed a command.	coreclkout_hip	RP
sys_pre_det_state_i	I	Indicates whether or not a card is present in the slot. 0 : slot is empty. 1 : card is present in the slot.	coreclkout_hip	RP
sys_mrl_sensor_state_i	I	MRL Sensor State. Indicates the state of the manually operated retention latch (MRL) sensor. 0 : MRL is closed. 1 : MRL is open.	coreclkout_hip	RP
sys_eml_interlock_engaged_i	I	Indicates whether the system electromechanical interlock is engaged, and controls the state of the electromechanical interlock status bit in the Slot Status Register.	coreclkout_hip	RP
sys_aux_pwr_det_i	I	Auxiliary Power Detected. Used to report to the host software that auxiliary power (Vaux) is present. Refer to the Device Status Register in the PCI Express Capability Structure.	coreclkout_hip	RP

4.9. Power Management Interface

Software programs the device into a D-state by writing to the Power Management Control and Status register in the PCI Power Management Capability Structure. The power management output signals indicate the current power state. The IP core supports the two mandatory power states: D0 (full power) and D3 (preparation for a loss of power). It does not support the optional D1 and D2 low-power states.

The correspondence between the device power states (D states) and link power states (L states) is as follows:

Table 48. Relationship Between Device and Link Power States

Device Power State	Link Power State
D0	L0
D1 (not supported)	L1
D2 (not supported)	L1
D3	L1, L2/L3 Ready

P-Tile does not support ASPM.

Table 49. Power Management Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
`pm_state_o[2:0]`	O	Indicates the current power state.	coreclkout_hip	EP/RP/BP
x16/x8: `pm_dstate_o[31:0]` x4: `pm_dstate_o[3:0]`	O	Power management D-state for each function. • 0001b : D0 • 0010b : D1 • 0100b : D2 • 1000b : D3 • 0000b : uninitialized or invalid	Async	EP/RP/BP
x16/x8: `apps_pm_xmt_pme_i[7:0]` x4: NA	I	The application logic asserts this signal for one cycle to wake up the Power Management Capability (PMC) state machine from a D1, D2, or D3 power state. Upon wake-up, the IP core sends a PM_PME message.	coreclkout_hip	EP/BP
`apps_pm_xmt_turnoff_i`	I	This signal is a request from the Application Layer to generate a PM_Turn_Off message. The Application Layer must assert this signal for one clock cycle. The IP core does not return an acknowledgement or grant signal. The Application Layer must not pulse the same signal again until the previous message has been transmitted.	coreclkout_hip	RP
`app_init_rst_i`	I	The Application Layer uses this signal to request a hot reset to downstream devices. The hot reset request will be sent when a single-cycle pulse is applied to this pin.	coreclkout_hip	RP

4.10. Configuration Output Interface

The Transaction Layer configuration output (`tl_cfg`) bus provides a subset of the information stored in the Configuration Space. Use this information in conjunction with the `app_err*` signals to understand TLP transmission problems.

Table 50. Configuration Output Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
`tl_cfg_ctl_o[15:0]`	O	Multiplexed data output from the register specified by `tl_cfg_add_o[4:0]`. The detailed information for each field in this bus is defined in the following table.	coreclkout_hip	EP/RP/BP
`tl_cfg_add_o[4:0]`	O	This address bus contains the index indicating which Configuration Space register information is being driven onto the `tl_cfg_ctl_o[15:0]` bits.	coreclkout_hip	EP/RP/BP
x16/x8: `tl_cfg_func_o[2:0]` x4: NA	O	Specifies the function whose Configuration Space register values are being driven out on `tl_cfg_ctl_o[15:0]`. • 3'b000: Physical Function 0 (PF0) • 3'b001: PF1 and so on	coreclkout_hip	EP/RP/BP

The table below provides the `tl_cfg_add_o[4:0]` to `tl_cfg_ctl_o[15:0]` mapping.

Table 51. Multiplexed Configuration Information Available on tl_cfg_ctl

`tl_cfg_add_o[4:0]`	`tl_cfg_ctl_o[15:8]`	`tl_cfg_ctl_o[7:0]`
5'h00	[15]: memory space enable [14]: IDO completion enable [13]: `perr_en` [12]: `serr_en` [11]: `fatal_err_rpt_en` [10]: `nonfatal_err_rpt_en` [9]: `corr_err_rpt_en` [8]: `unsupported_req_rpt_en`	Device control: [7]: bus master enable [6]: extended tag enable [5:3]: maximum read request size [2:0]: maximum payload size
5'h01	[15]: IDO request enable [14]: No Snoop enable [13]: Relaxed Ordering enable [12:8]: Device number	bus number
5'h02	[15]: `pm_no_soft_rst` [14]: RCB control [13]: Interrupt Request (IRQ) disable [12:8]: PCIe Capability IRQ message number	[7:5]: reserved [4]: system power control [3:2]: system attention indicator control [1:0]: system power indicator control
5'h03	Number of VFs [15:0]	
5'h04	[15]: reserved	[7]: ARI forward enable

continued...

tl_cfg_add_o[4:0]	tl_cfg_ctl_o[15:8]	tl_cfg_ctl_o[7:0]
	[14]: AtomicOP Egress Block field (cfg_atomic_egress_block) [13:9]: ATS Smallest Translation Unit (STU)[4:0] [8]: ATS cache enable	[6]: Atomic request enable [5:3]: TPH ST mode [2:1]: TPH enable [0]: VF enable
5'h05	[15:12]: auto negotiation link speed. Link speed encoding values are: • Gen1 : 0x1 • Gen2 : 0x2 • Gen3 : 0x4 • Gen4 : 0x8 [11:1]: Index of Start VF [10:0] [0]: reserved	
5'h06	MSI Address [15:0]	
5'h07	MSI Address [31:16]	
5'h08	MSI Address [47:32]	
5'h09	MSI Address [63:48]	
5'h0A	MSI Mask [15:0]	
5'h0B	MSI Mask [31:16]	
5'h0C	[15]: cfg_send_f_err [14]: cfg_send_nf_err [13]: cfg_send_cor_err [12:8]: AER IRQ message number	[7]: Enable extended message data for MSI (cfg_msi_ext_data_en) [6]: MSI-X func mask [5]: MSI-X enable [4:2]: Multiple MSI enable [1]: 64-bit MSI [0]: MSI enable
5'h0D	MSI Data [15:0]	
5'h0E	AER uncorrectable error mask [15:0]	
5'h0F	AER uncorrectable error mask [31:16]	
5'h10	AER correctable error mask [15:0]	
5'h11	AER correctable error mask [31:16]	
5'h12	AER uncorrectable error severity [15:0]	
5'h13	AER uncorrectable error severity [31:16]	
5'h14	[15:8]: ACS Egress Control Register (cfg_acs_egress_ctrl_vec)	[7]: ACS function group enable (cfg_acs_func_grp_en) [6]: ACS direct translated P2P enable (cfg_acs_p2p_direct_tranl_en) [5]: ACS P2P egress control enable (cfg_acs_egress_ctrl_en) [4]: ACS upstream forwarding enable (cfg_acs_up_forward_en) [3]: ACS P2P completion redirect enable (cfg_acs_p2p_compl_redirect_en) [2]: ACS P2P request redirect enable (cfg_acs_p2p_req_redirect_en)

continued...

tl_cfg_add_o[4:0]	tl_cfg_ctl_o[15:8]	tl_cfg_ctl_o[7:0]
		[1]: ACS translation blocking enable (cfg_acs_at_blocking_en) [0]: ACS source validation enable (RP) (cfg_acs_validation_en)
5'h15	[15]: reserved [14]: 10-bit tag requester enable (cfg_10b_tag_req_en) [13]: VF 10-bit tag requester enable (cfg_vf_10b_tag_req_en) [12]: PRS_RESP_FAILURE (cfg_prs_response_failure) [11]: PRS_UPRGI (cfg_prs_uprgi) [10]: PRS_STOPPED (cfg_prs_stopped) [9]: PRS_RESET (cfg_prs_reset) [8]: PRS_ENABLE (cfg_prs_enable)	[7:3]: reserved [2:0]: ARI function group (cfg_ari_func_grp)
5'h16	PRS_OUTSTANDING_ALLOCATION (cfg_prs_outstanding_allocation) [15:0]	
5'h17	PRS_OUTSTANDING_ALLOCATION (cfg_prs_outstanding_allocation) [31:16]	
5'h18	[15:10]: reserved [9]: Disable autonomous generation of LTR clear message (cfg_disable_ltr_clr_msg) [8]: LTR mechanism enable (cfg_ltr_m_en)	[7]: Infinite credits for Posted header [6]: Infinite credits for Posted data [5]: Infinite credits for Completion header [4]: Infinite credits for Completion data [3]: End-end TLP prefix blocking (cfg_end2end_tlp_pfx_blck) [2]: PASID enable (cfg_pf_pasid_en) [1]: Execute permission enable (cfg_pf_passid_execute_perm_en) [0]: Privileged mode enable (cfg_pf_passid_priv_mode_en)
5'h19	[15:9]: reserved [8]: Slot control attention button pressed enable (cfg_atten_button_pressed_en)	[7]: Slot control power fault detect enable (cfg_pwr_fault_det_en) [6]: Slot control MRL sensor changed enable (cfg_mrl_sensor_chged_en) [5]: Slot control presence detect changed enable (cfg_pre_det_chged_en) [4]: Slot control hot plug interrupt enable (cfg_hp_int_en) [3]: Slot control command completed interrupt enable (cfg_cmd_cpled_int_en) [2]: Slot control DLL state change enable (cfg_dll_state_change_en) [1]: Slot control accessed (cfg_hp_slot_ctrl_access) [0]: PF's SERR# enable (cfg_br_ctrl_serren)

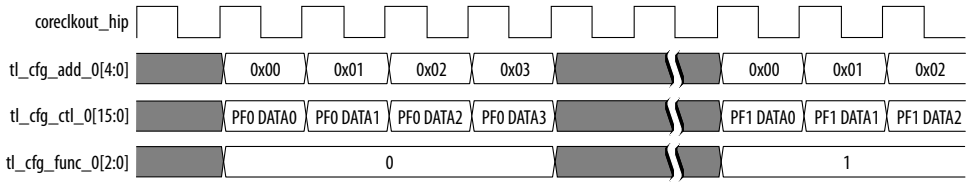
continued...

tl_cfg_add_o[4:0]	tl_cfg_ctl_o[15:8]	tl_cfg_ctl_o[7:0]
5'h1A	LTR maximum snoop latency register (cfg_ltr_max_latency[15:0])	
5'h1B	LTR maximum no-snoop latency register (cfg_ltr_max_latency[31:16])	
5'h1C	[15:8]: enabled Traffic Classes (TCs) (cfg_tc_enable[7:0])	[5:0]: auto negotiation link width 6'h01 = x1 6'h02 = x2 6'h04 = x4 6'h08 = x8 6'h10 = x16
5'h1D	MSI Data[31:16]	
5'h1E	N/A	
5'h1F	N/A	

Note: The information on the Configuration Output (tl_cfg) bus is time-division multiplexed (TDM).

- When tl_cfg_func[2:0] = 3'b000, tl_cfg_ctl[31:0] drive out the PF0 Configuration Space register values.
- Then, tl_cfg_func[2:0] are incremented to 3'b001.
- When tl_cfg_func[2:0] = 3'b001, tl_cfg_ctl[31:0] drive out the PF1 Configuration Space register values.
- This pattern repeats to cover all enabled PFs.

Figure 39. Configuration Output Interface Timing Diagram



Note: The P-Tile IP for PCIe provides a data link layer timer update output. Details on this signal are in the table below. When this signal asserts, you can sample the tl_cfg_ctl_o bus to see the new link speed, link width or max payload size and update the Replay/Ack-Nak timers accordingly.

Table 52. Data Link Layer Timer Update Signal

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
dl_timer_update_o	O	Active high pulse that asserts whenever the current link speed, link width, or max payload size changes. When any of these parameters changes, the IP's internal Replay/Ack-Nak timers	coreclkout_hip	EP/RP/BP

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		default back to their internally calculated PCIe tables. To override these default values, reprogram the Port Logic register when these events occur.		

4.11. Configuration Intercept Interface (EP Only)

The Configuration Intercept Interface (CII) allows the application logic to detect the occurrence of a Configuration (CFG) request on the link and to modify its behavior.

The application logic should detect the CFG request at the rising edge of `cii_req`. Due to the latency of the EMIB, the `cii_req` can be deasserted many cycles after the deassertion of `cii_halt`.

The application logic can use the CII to:

- Delay the processing of a CFG request by the controller. This allows the application to perform any housekeeping task first.
- Overwrite the data payload of a CfgWr request. The application logic can also overwrite the data payload of a CfgRd completion TLP.

This interface also allows you to implement the Intel Vendor Specific Extended Capability (VSEC) registers. All configuration accesses targeting the Intel VSEC registers (addresses 0xD00 to 0xFFF) are automatically mapped to this interface and can be monitored via this interface.

If you are not using this interface, tie `cii_halt_p0/1` to logic 0.

Figure 40. Configuration Intercept Interface Timing Diagram

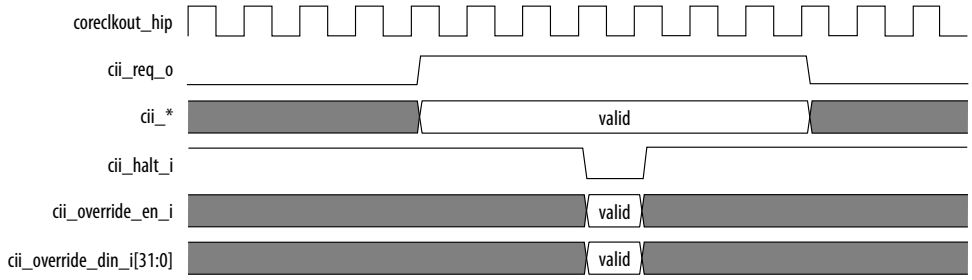


Table 53. Configuration Intercept Interface

Signal Name	Direction	Description	Clock domain	EP/RP/BP
`cii_req_o`	O	Indicates the CFG request is intercepted and all the other CII signals are valid.	coreclkout_hip	EP
`cii_hdr_poisoned_o`	O	The poisoned bit in the received TLP header on the CII.	coreclkout_hip	EP

continued...

Signal Name	Direction	Description	Clock domain	EP/RP/BP
cii_hdr_first_be_o[3:0]	O	The first dword byte enable field in the received TLP header on the CII.	coreclkout_hip	EP
cii_func_num_o[2:0]	O	The function number in the received TLP header on the CII.	coreclkout_hip	EP
cii_wr_o	O	Indicates that cii_dout_p0/1 is valid. This signal is asserted only for a configuration write request.	coreclkout_hip	EP
cii_addr_o[9:0]	O	The double word register address in the received TLP header on the CII.	coreclkout_hip	EP
cii_dout_o[31:0]	O	Received TLP payload data from the link partner to your application client. The data is in little endian format. The first received payload byte is in [7:0].	coreclkout_hip	EP
cii_override_en_i	I	Override enable. When the application logic asserts this input, the PCIe Hard IP overrides the CfgWr payload or CfgRd completion using the data supplied by the application logic on cii_override_din.	coreclkout_hip	EP
cii_override_din_i[31:0]	I	Override data. • CfgWr: override the write data to the PCIe Hard IP register with data supplied by the application logic on cii_override_din. • CfgRd: override the data payload of the completion TLP with data supplied by the application logic on cii_override_din.	coreclkout_hip	EP
cii_halt_i	I	Flow control input signal. When cii_halt_p0/1 is asserted, the PCIe Hard IP halts the processing of CFG requests for the PCIe configuration space registers.	coreclkout_hip	EP

4.12. Hard IP Reconfiguration Interface

The Hard IP reconfiguration interface is an Avalon-MM slave interface with a 21-bit address and an 8-bit data bus. It is also sometimes referred to as the User Avalon-MM Interface. You can use this interface to dynamically modify the value of configuration registers. Note that after a warm reset or cold reset, changes made to the configuration registers of the Hard IP via the Hard IP reconfiguration interface are lost as these registers revert back to their default values.

Note: This interface can be used in Endpoint, Root Port and TLP Bypass modes. However, it must be enabled if Root Port or TLP Bypass mode is selected.

In Root Port mode, the application logic uses the Hard IP reconfiguration interface to access its PCIe configuration space to perform link control functions (such as Hot Reset, link disable, or link retrain).

In TLP Bypass mode, the Hard IP forwards the received Type0/1 Configuration request TLPs to the application logic, which must respond with Completion TLPs with a status of Successful Completion (SC), Unsupported Request (UR), Configuration Request Retry Status (CRS), or Completer Abort (CA). If a received Configuration request TLP needs to update a PCIe configuration space register, the application logic needs to use the Hard IP reconfiguration interface to access that PCIe configuration space register.

Table 54. Hard IP Reconfiguration Interface

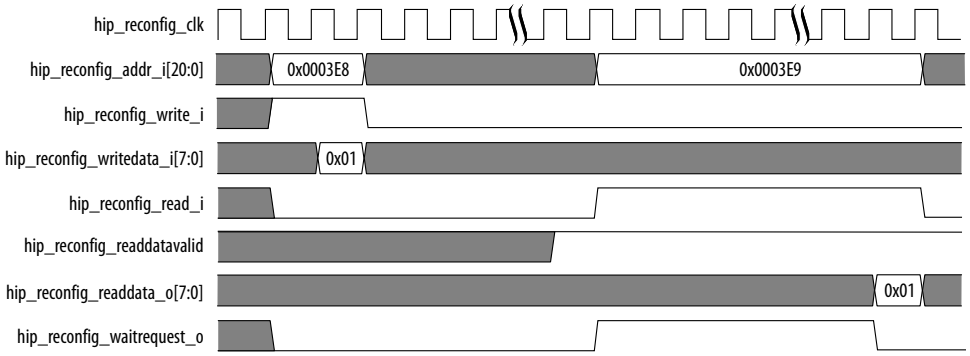
Signal Name	Direction	Description	Clock Domain	EP/RP/BP
hip_reconfig_clk	I	Reconfiguration clock 50 MHz - 125 MHz (Range) 100 MHz (Recommended)		EP/RP/BP
hip_reconfig_readdata_o[7:0]	O	Avalon-MM read data outputs	hip_reconfig_clk	EP/RP/BP
hip_reconfig_readdatavalid_o	O	Avalon-MM read data valid. When asserted, the data on hip_reconfig_readdata_o[7:0] is valid.	hip_reconfig_clk	EP/RP/BP
hip_reconfig_write_i	I	Avalon-MM write enable	hip_reconfig_clk	EP/RP/BP
hip_reconfig_read_i	I	Avalon-MM read enable	hip_reconfig_clk	EP/RP/BP
hip_reconfig_address_i[20:0]	I	Avalon-MM address	hip_reconfig_clk	EP/RP/BP
hip_reconfig_writedata_i[7:0]	I	Avalon-MM write data inputs	hip_reconfig_clk	EP/RP/BP
hip_reconfig_waitrequest_o	O	When asserted, this signal indicates that the IP core is not ready to respond to a request.	hip_reconfig_clk	EP/RP/BP
dummy_user_avmm_rst	I	Reset signal. You can tie it to ground or leave it floating when using the Hard IP Reconfiguration Interface.		EP/RP/BP

Reading and Writing to the Hard IP Reconfiguration Interface

Reading from the Hard IP reconfiguration interface of the P-Tile Avalon-ST IP for PCI Express retrieves the current value at a specific address. Writing to the reconfiguration interface changes the data value at a specific address. Intel recommends that you perform read-modify-writes when writing to a register, because two or more features may share the same reconfiguration address.

Modifying the PCIe configuration registers directly affects the behavior of the PCIe device.

Figure 41. Timing Diagram to Perform Read and Write Operations Using the Hard IP Reconfiguration Interface



4.12.1. Address Map for the User Avalon-MM Interface

The User Avalon-MM interface provides access to the configuration registers and the IP core registers. This interface includes an 8-bit data bus and a 21-bit address bus (which contains the byte addresses).

There are two methods to access the configuration registers:

- Using direct User Avalon-MM interface (byte access)
- Using the Debug (DBI) register access (dword access). This method is useful when you need to read/write the entire 32 bits at one time (Counter/ Lane Margining, etc.)

Figure 42. Address Map for the User Avalon-MM Interface

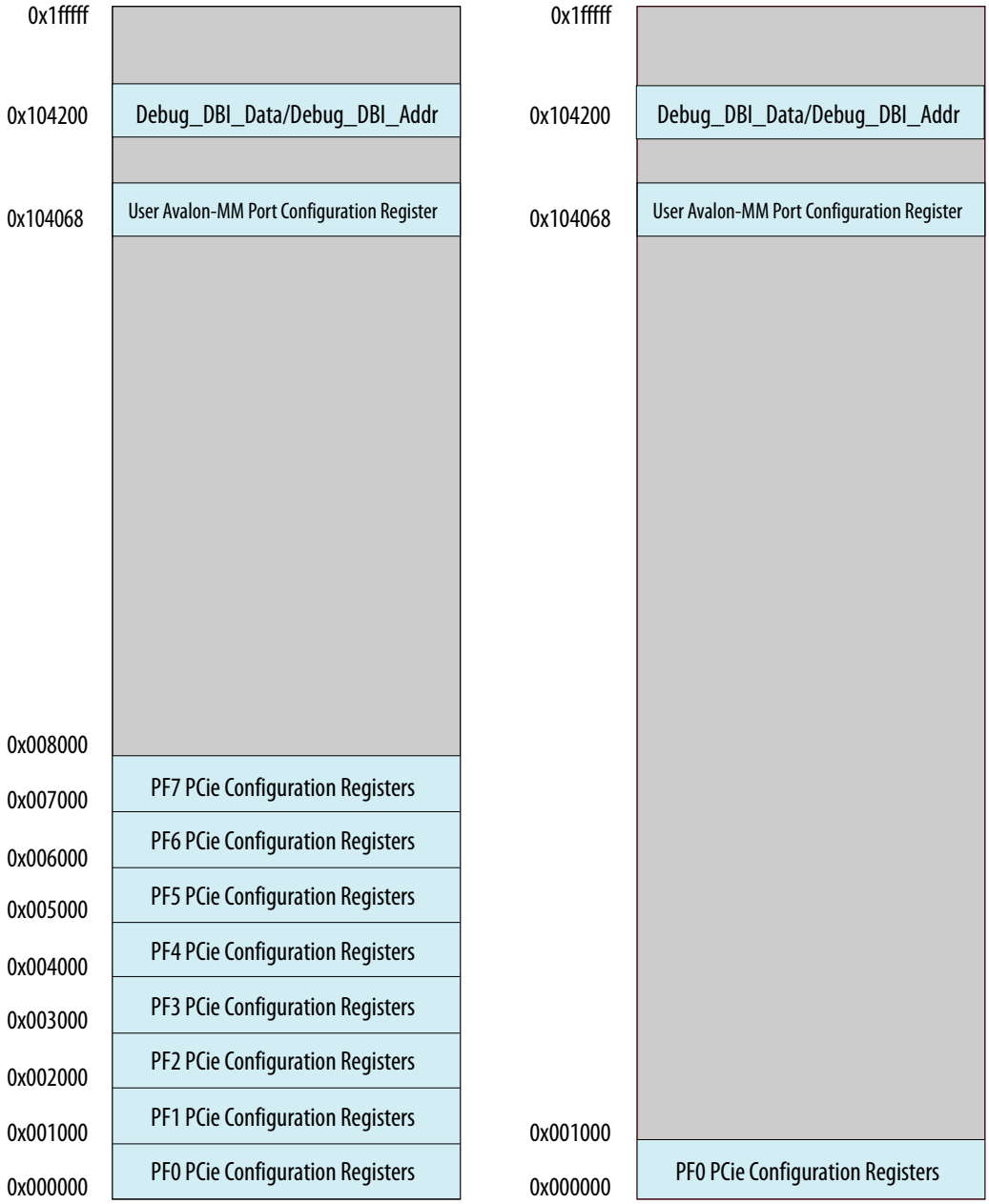


Table 55. Configuration Space Offsets

Registers	User Avalon-MM Offsets	Comments
Physical function 0	0x0000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16, x8 and x4 cores.
Physical function 1	0x1000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
Physical function 2	0x2000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
Physical function 3	0x3000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
Physical function 4	0x4000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
Physical function 5	0x5000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
Physical function 6	0x6000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
Physical function 7	0x7000	Refer to Appendix A for more details of the PF configuration space. This PF is available for x16 and x8 cores only.
User Avalon-MM Port Configuration Register	0x104068	Refer to User Avalon-MM Port Configuration Register (Offset 0x104068) on page 90 for more details.
Debug (DBI) Register	0x104200 to 0x104204	Refer to Using the Debug Register Interface Access (Dword Access) on page 92 for more details.

Note: The x4 configuration only supports the RP mode. Therefore, this configuration does not support the multi-function feature.

4.12.1.1. User Avalon-MM Port Configuration Register (Offset 0x104068)

Table 56. User Avalon-MM Port Configuration Register

Bits	Register Description	Default Value	Access
[31:29]	Reserved	0x0	RO
[28:18]	Select the virtual function number.	0x0	RW/RO
[17]	To access the virtual function registers, this bit should be set to one.	0x0	RW/RO

continued...

Bits	Register Description	Default Value	Access
[16:2]	Reserved	0x0	RO
[1]	Reserved. Clear this bit for access to standard PCIe configuration registers.	0x0	RW/RO
[0]	If set, it allows access to Intel VSEC registers.	0x0	RW/RO

4.12.2. Configuration Registers Access

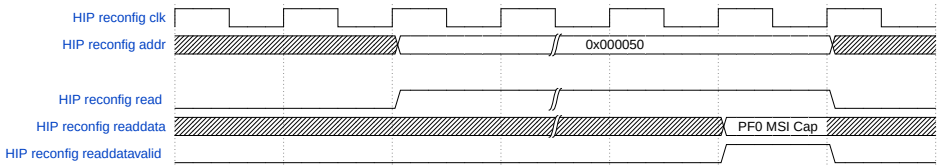
4.12.2.1. Using Direct User Avalon-MM Interface (Byte Access)

Targeting PF Configuration Space Registers

User application needs to specify the offsets of the targeted PF registers.

For example, if the application wants to read the MSI Capability Register of PF0, it will issue a Read with address 0x0050 to target the MSI Capability Structure of PF0.

Figure 43. PF Configuration Space Registers Access Timing Diagram



Targeting VF Configuration Space Registers

User application needs to first specify the VF number of the targeted configuration register.

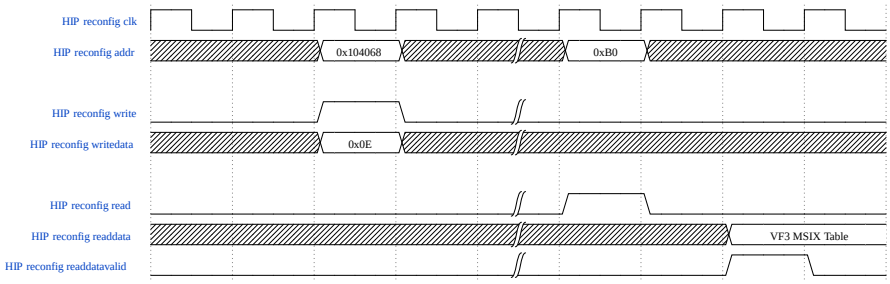
The application needs to program the User Avalon-MM Port Configuration Register at offset 0x104068 accordingly.

For example, to read VF3's MSI-X Capability registers, the user application needs to:

1. Issue a user Avalon-MM Write request with address 0x104068 and data 0xE (vf_num[28:18] = 3, vf _select[17] = 1, vsec[0]=0).
2. Issue a user Avalon-MM Read request with address 0xB0 to access VF3 registers.

Note: You need to reprogram the Port Configuration and Status Register to access PF registers.

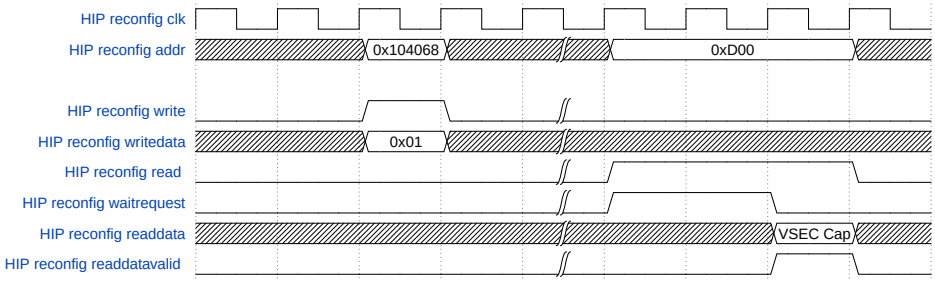
Figure 44. VF Configuration Space Registers Access Timing Diagram



Targeting VSEC Registers

User application needs to program the VSEC field (0x104068 bit[0]) first. Then all accesses from the user Avalon-MM interface starting at offset 0xD00 will be translated to VSEC configuration space registers.

Figure 45. VSEC Registers Access Timing Diagram



4.12.2.2. Using the Debug Register Interface Access (Dword Access)

DEBUG_DBI_ADDR register is located at user Avalon-MM offsets 0x104204 to 0x104207 (corresponding to byte 0 to byte 3).

Table 57. DEBUG_DBI_ADDR Register

Names	Bits	R/W	Descriptions
d_done	31	RO	1: indicates debug DBI read/ write access done
d_write	30	R/W	1: write access 0: read access
d_warm_reset	29	RO	1: normal operation 0: warm reset is on-going
d_vf	28:18	R/W	Specify the virtual function number.
d_vf_select	17	R/W	To access the virtual function registers, set this bit to one.
d_pf	16:14	R/W	Specify the physical function number.
reserved	13:12	R/W	Reserved

continued...

Names	Bits	R/W	Descriptions
d_addr	11:2	R/W	Specify the DW address for the P-Tile Hard IP DBI interface.
d_shadow_select	1	R/W	Reserved. Clear this bit for access to standard PCIe configuration registers.
d_vsec_select	0	R/W	If set, this bit allows access to Intel VSEC registers.

DEBUG_DBI_DATA register is located at user Avalon-MM offsets 0x104200 to 0x104203 (corresponding to byte 0 to byte 3).

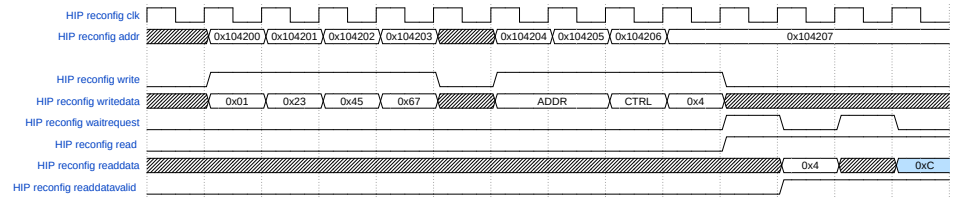
Table 58. DEBUG_DBI_DATA Register

Names	Bits	R/W	Descriptions
d_data	31:0	R/W	Read or write data for the P-Tile Hard IP register access.

To write all 32 bits in a Debug register at a time:

1. Use the user_avmm interface to access 0x104200 to 0x104203 to write the data first.
2. Use the user_avmm interface to access 0x104204 to 0x104206 to set the address and control bits.
3. Use the user_avmm interface to write to 0x104207 to enable the read/write bit (bit[30]).
4. Use the user_avmm interface to access 0x104207 bit[31] to poll if the write is complete.

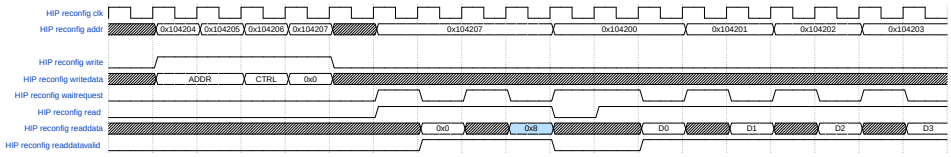
Figure 46. DBI Register Write Timing Diagram



To read all 32 bits in a Debug register at a time:

1. Use the user_avmm interface to access 0x104204 to 0x104206 to set the address and control bits.
2. Use the user_avmm interface to write to 0x104207 to enable the read bit (bit[30]).
3. Use the user_avmm interface to access 0x104207 bit[31] to poll if the read is complete.
4. Use the user_avmm interface to access 0x104200 to 0x104203 to read the data

Figure 47. DBI Register Read Timing Diagram



4.13. PHY Reconfiguration Interface

The PHY reconfiguration interface is an optional Avalon-MM slave interface with a 26-bit address and an 8-bit data bus. Use this bus to read the value of PHY registers. Refer to Table 100 on page 161 for details on addresses and bit mappings for the PHY registers that you can access using this interface.

These signals are present when you turn on **Enable PHY reconfiguration** on the **Top-Level Settings** tab using the parameter editor.

Please note that the PHY reconfiguration interface is shared among all the PMA quads.

Table 59. PHY Reconfiguration Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
xcvr_reconfig_clk	I	Reconfiguration clock 50 MHz - 125 MHz (Range) 100 MHz (Recommended)		EP/RP/BP
xcvr_reconfig_readdata[7:0]	O	Avalon-MM read data outputs	xcvr_reconfig_clk	EP/RP/BP
xcvr_reconfig_readdatavalid	O	Avalon-MM read data valid. When asserted, the data on xcvr_reconfig_readdata[7:0] is valid.	xcvr_reconfig_clk	EP/RP/BP
xcvr_reconfig_write	I	Avalon-MM write enable	xcvr_reconfig_clk	EP/RP/BP
xcvr_reconfig_read	I	Avalon-MM read enable. This interface is not pipelined. You must wait for the return of the xcvr_reconfig_readdata[7:0] from the current read before starting another read operation.	xcvr_reconfig_clk	EP/RP/BP
xcvr_reconfig_address[25:0]	I	Avalon-MM address [25:21] are used to indicate the Quad. 5'b00001 : Quad 0 5'b00010 : Quad 1 5'b00100 : Quad 2 5'b01000 : Quad 3	xcvr_reconfig_clk	EP/RP/BP

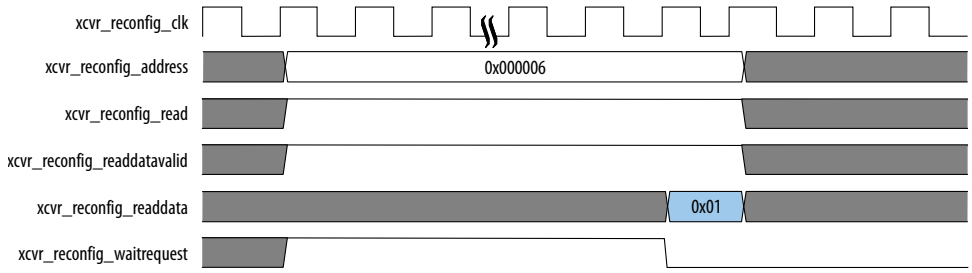
continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		[20:0] are used to indicate the offset address.		
xcvr_reconfig_writedata[7:0]	I	Avalon-MM write data inputs	xcvr_reconfig_clk	EP/RP/BP
xcvr_reconfig_waitrequest	O	When asserted, this signal indicates that the PHY is not ready to respond to a request.	xcvr_reconfig_clk	EP/RP/BP

Reading from the PHY Reconfiguration Interface

Reading from the PHY reconfiguration interface of the P-Tile Avalon-ST IP for PCI Express retrieves the current value at a specific address.

Figure 48. Timing Diagram to Perform Read Operations Using the PHY Reconfiguration Interface



4.14. Page Request Service (PRS) Interface (EP Only)

When an Endpoint determines that it requires access to a page for which the ATS translation is not available, it sends a Page Request message to request that the page be mapped into system memory.

The PRS interface allows the monitoring of when PRS events happen, what functions these PRS events belong to, and what types of events they are.

The PRS interface is only available in EP mode, and with TLP Bypass disabled.

Note: In the Intel Quartus Prime 19.4 release, only PF0 supports PRS. Furthermore, in this release, the PRS interface only has Compilation and Simulation support.

Table 60. PRS Interface Signals

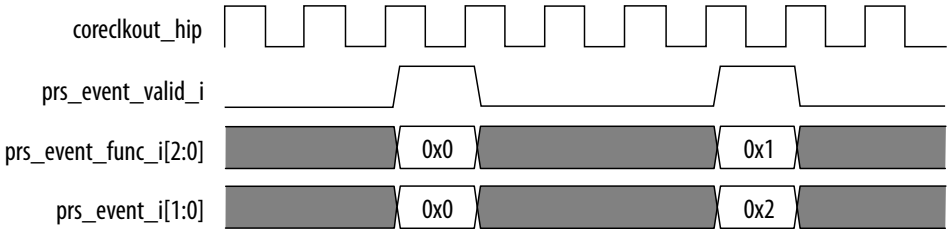
Signal Name	Direction	Description	Clock Domain	EP/RP/BP
prs_event_valid_i	I	This signal qualifies prs_event_func_i and prs_event_i.	coreclkout_hip	EP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
		There is a single-cycle pulse for each PRS event.		
prs_event_func_i[2:0]	I	The function number for the PRS event.	coreclkout_hip	EP
prs_event_i[1:0]	I	00 : Indicate that the function has received a PRG response failure. 01: Indicate that the function has received a response with Unexpected Page Request Group Index. 10: Indicate that the function has completed all previously issued page requests and that it has stopped requests for additional pages. Only valid when the PRS enable bit is clear. 11: reserved.	coreclkout_hip	EP

The figure below shows the timing diagram for the PRS event interface when the application layer of function 0 sends an event of PRG response reception, and the application layer of function 1 sends an event stopping requests for additional pages.

Figure 49. Example Timing Diagram for the PRS Event Interface



5. Advanced Features

5.1. PCIe Port Bifurcation and PHY Channel Mapping

The PCIe controller IP contains a set of port bifurcation muxes to remap the four controller PIPE lane interfaces to the shared 16 PCIe PHY lanes. The table below shows the relationship between PHY lanes and the port mapping.

Table 61. Port Bifurcation and PHY Channel Mapping

Bifurcation Mode	Port 0 (x16)	Port 1 (x8)	Port 2 (x4)	Port 3 (x4)
1 x16	0 - 15	NA	NA	NA
2 x8	0 - 7	8 - 15	NA	NA
4 x4	4 - 7	8 - 11	0 - 3	12 - 15

Note: For more details on the bifurcation modes, refer to the *Architecture* section in chapter 2.

5.2. Virtualization Support

The two components of the P-Tile IP for PCIe's virtualization support are:

- Single root I/O virtualization (SR-IOV)
- VirtIO

5.2.1. SR-IOV Support

The P-Tile IP for PCIe supports SR-IOV. The endpoint port controllers in the IP support up to eight physical functions (PF) and 2048 virtual functions (VF) per SR-IOV endpoint. The VF configuration space registers are hardened in the P-Tile. The specific VF-based work queues and interrupt tables must be implemented in the FPGA fabric by the user application.

For more details on the configuration space registers required for virtualization support, refer to Configuration Space Registers for Virtualization on page 187.

5.2.1.1. SR-IOV Supported Features List

Table 62. SR-IOV Supported Features

Feature	Support
SR-IOV	Supported in x16/x8 controller EP mode.

continued...

Feature	Support
	Not supported in RP mode (x4).
MSI	Supported in PFs only. Not supported in VFs. No Per Vector Masking (PVM). If you need PVM, you must use MSI-X. *Note:* When SR-IOV is enabled, either MSI or MSI-X must be enabled.
MSI-X	Supported by all PFs. For SR-IOV, PFs and VFs are always MSI-X capable. *Note:* VFs share a common Table Size. VF Table BIR/Offset and PBA BIR/Offset are fixed at compile time. *Note:* When SR-IOV is enabled, either MSI or MSI-X must be enabled.
Function Level Reset (FLR)	Supported by all PFs/VFs. Required for all SR-IOV functions.
Extended Tags	Supported by all PFs/VFs. The Extended Tags feature allows the TLP Tag field to be 8-bit, thus allowing the support of 256 tags. Note that the application is restricted to a max of 256 outstanding tags, at any given time, for all functions combined. The application logic is responsible for implementing the tag generation/tracking functions.
10-bit Tags	Supported by all PFs/VFs. Refer to Tag Allocation on page 64 for more details.
AER	PFs are always AER capable. No AER implemented for VFs.
Active-State Power Management (ASPM) Optionality Compliance	Supported by all PFs/VFs. Only used to indicate ASPM is not supported.
Atomic Ops	Requester capability is supported by all PFs/VFs. Completer capability is supported. Compare and Swap (CAS) AtomicOps are also supported. They can handle up to 128-bit operands.
Internal Error Reporting	Supported by all PFs (because all PFs are AER capable). No support for VFs (because VFs do not support AER).
TLP Processing Hints	2-bit Processing Hint and 8-bit Steering Tag are supported by all PFs/VFs. TPH Prefixes are not supported. You can optionally choose to enable the TPH Requestor capability. However, the IP is always TPH Completer capable.
ID-Based ordering	Supported by all PFs/VFs. However, the IP core does NOT perform the reordering. The Application Layer must do this. The IP core only provides the IDO Request & Completion Enable bits in the Device Control 2 register. This gives the application permission to set the Attr bits in Requests and Completions that it transmits. *Note:* Reordering capability on the RX side may be limited by your bypass queue. On the TX side, the IP core does not set the IDO bits on internally generated TLPs.
Relaxed Ordering	Implemented on the RX side. This feature is always active. On the TX side, reordering is done by the application.

continued...

Feature	Support
Alternative Routing ID Interpretation (ARI)	EP (PFs/VFs) is always ARI capable. This is a device-level option (all lanes or none will support ARI). In addition, RP will always be ARI capable (ARI Forwarding Supported bit is always 1).
Address Translation Service (ATS)	Supported by all EP PFs/VFs.
Page Request Service Interface (PRI)	Supported by all EP PFs/VFs.
User Extensions (Customer VSEC)	Supported by all PFs/VFs.
Gen3 Receiver Impedance (3.0 ECN)	Supported
Device Serial Number	Supported
Completion Timeout Ranges (Device Capabilities 2)	All ranges are supported.
Data Link Layer Active Reporting Capability (Link Capabilities)	This capability is always supported in RP mode, but not in EP mode.
Surprise Down Error Reporting Capability (Link Capabilities)	Supported
PM-PCI Power Management	Only D0/D3 states are supported.
ASPM (L0s/L1)	Not supported
Process Address Space ID (PASID)	Supported
TLP prefix	Supported, mainly for PASID
Latency Tolerance Reporting (LTR)	Supported (only for PASID)
Access Control Services	Supported

5.2.1.2. Implementation

The VF configuration space is implemented in P-Tile logic, and does not require FPGA fabric resources.

Accessing VF PCIe Information:

Due to the limited number of pins between P-Tile and the FPGA fabric, the PCIe configuration space for VFs is not directly available to the user application.

User application can use the following methods to retrieve necessary information (bus master enable, MSI-X etc…):

- Monitor specific VF registers using the Configuration Intercept Interface (for more details, refer to section Configuration Intercept Interface (EP Only) on page 85).
- Read/write specific VF registers using the Hard IP Reconfiguration Interface (for more details, refer to *Targeting VF Configuration Space Registers* in section Using Direct User Avalon-MM Interface (Byte Access) on page 91).

Accessing VF PCIe Information:

VF IDs are calculated within P-Tile. User application has sideband signals `rx_st_vf_num_o` and `rx_st_vf_active_o` with the TLP to identify the associated VFs within the PFs.

BDF Assignments:

When SR-IOV is enabled, the ARI capability is always enabled.

The P-Tile IP for PCIe automatically calculates the completer/requester ID on the Transmit side.

User application needs to provide the VF and PF information in the Header as shown below:

(For X16, sn is either s0 or s1. For X8, sn is s0).

- tx_st_hdr_sn[127]: must be set to 0
- tx_st_hdr_sn[83]: tx_st_vf_active
- tx_st_hdr_sn[82:80]: tx_st_func_num[2:0]
- tx_st_hdr_sn[95:84]: tx_st_vf_num[11:0]

In the following example, VF3 of PF1 is receiving and sending a request:

For the Receive TLP:

rx_st_func_num_o = 1h indicating that a VF associated with PF1 is making the request.

rx_st_vf_num_o = 3h, and rx_st_vf_active_o = 1 indicating that VF3 of PF1 is the active VF.

For the Transmit TLP of VF3 associated with PF1:

- tx_st_hdr_sn[83] = 1h
- tx_st_hdr_sn[82:80] = 1h
- tx_st_hdr_sn[95:84] = 3h

5.2.1.2.1. VF Error Flag Interface (for x16/x8 Cores Only)

The VFs, with no AER support, are required to generate Non-Fatal error messages. The IP does not generate any error message. It is up to the user application logic to generate appropriate messages when specific error conditions occur.

The P-Tile IP for PCIe makes necessary signals available to the user application logic to generate these messages. The Completion Timeout Interface (described in section Completion Timeout Interface on page 75) and the signals listed in the table below provide the necessary information to generate Non-Fatal error messages.

Table 63. VF Error Flag Interface

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
X16: vf_err_poisonedwrreq_s0/1/2/3_o X8: vf_err_poisonedwrreq_s0/1_o	O	Indicates a Poisoned Write Request is received.	coreclkout_hip	EP
X16: vf_err_poisonedcompl_s0/1/2/3_o	O	Indicates a Poisoned Completion is received.	coreclkout_hip	EP

continued...

Signal Name	Direction	Description	Clock Domain	EP/RP/BP
X8: vf_err_poisonedcompl_s0/1_o				
X16: vf_err_ur_posted_s0/1/2/3_o X8: vf_err_ur_posted_s0/1_o	O	Indicates the IP core received a Posted UR request.	coreclkout_hip	EP
X16: vf_err_ca_postedreq_s0/1/2/3_o X8: vf_err_ca_postedreq_s0/1_o	O	Indicates the IP core received a Posted CA request.	coreclkout_hip	EP
X16: vf_err_vf_num_s0/1/2/3_o[10:0] X8: vf_err_vf_num_s0/1_o[10:0]	O	Indicates the VF number for which the error is detected.	coreclkout_hip	EP
X16: vf_err_func_num_s0/1/2/3_o[2:0] X8: vf_err_func_num_s0/1_o[2:0]	O	Indicates the physical function number associated with the VF that has the error.	coreclkout_hip	EP
vf_err_overflow_o	O	Indicates a VF error FIFO overflow and a loss of an error report. The overflow can happen when coreclkout_hip is slower than the default value. If coreclkout_hip is running at the default frequency, the overflow will not happen.	coreclkout_hip	EP
user_sent_vfnonfatalmsg_s0_i	I	Indicates the user application sent a non-fatal error message in response to an error detected.	coreclkout_hip	EP
user_vfnonfatalmsg_vfnum_s0_i[10:0]	I	Indicates the VF number for which the error message was generated. This bus is valid when user_sent_vfnonfatalmsg_s0_i is high.	coreclkout_hip	EP
user_vfnonfatalmsg_func_num_s0_i[2:0]	I	Indicates the PF number associated with the VF with the error. This bus is valid when user_sent_vfnonfatalmsg_s0_i is high.	coreclkout_hip	EP

5.2.1.3. VF to PF Mapping

VF to PF mapping always starts from the lowest possible PF number. For instance, if the IP has 2 PFs, wherein PF0 has 64 VFs and PF1 has 16 VFs, VF1 to VF64 are mapped to PF0, and VF65 to VF80 are mapped to PF1.

Currently, the IP core only supports the following PF/VF combinations:

Table 64. Supported PF/VF Combinations

Number of PFs	Number of VFs per PF (PF0/PF1/PF2/PF3/PF4/PF5/PF6/PF7)	Total VFs
1	8	8
1	16	16
1	32	32
1	64	64
1	128	128
1	256	256
1	512	512
2	16/16	32
2	32/32	64
2	128/128	256
2	256/256	512
2	32/0	32
2	0/32	32
2	64/0	64
2	0/64	64
2	128/0	128
2	0/128	128
2	256/0	256
2	0/256	256
2	512/0	512
2	0/512	512
2	1024/0	1024
2	0/1024	1024
2	2048/0	2048
2	0/2048	2048
4	128/0/0/0	128
4	0/128/0/0	128
4	256/0/0/0	256
4	0/256/0/0	256
4	1024/0/0/0/0	1024
4	0/1024/0/0	1024
8	256/0/0/0/0/0/0/0	256
8	0/256/0/0/0/0/0/0	256

For example, the row that shows the combination of four PFs, 256 VFs, and the notation `256/0/0/0` in the `Number of VFs per PF` column indicates that all 256 VFs are mapped to PF0, while no VF is mapped to PF1, PF2 or PF3.

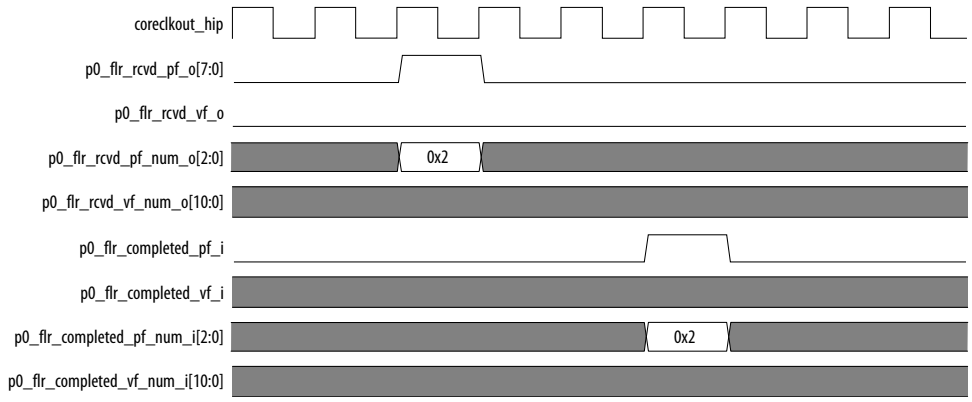
Note: SR-IOV permutations allow any PF to be assigned the initial VF allocation.

5.2.1.4. Function Level Reset (FLR)

Use the FLR interface to reset individual SR-IOV functions. The PCIe Hard IP supports FLR for both PFs and VFs. If the FLR is for a specific VF, the received packets for that VF are no longer valid. The flr_* interface signals are provided to the application interface for this purpose. When the flr_rcvd* signal is asserted, it indicates that an FLR is received for a particular PF/VF. Application logic needs to perform its FLR routine and send the completion status back on the flr_completed* interface. The Hard IP will wait for the flr_completed* status to re-enable the VF. Prior to that event, the Hard IP will respond to all transactions to the function that is reset by the FLR with completions with an Unsupported Request (UR) status.

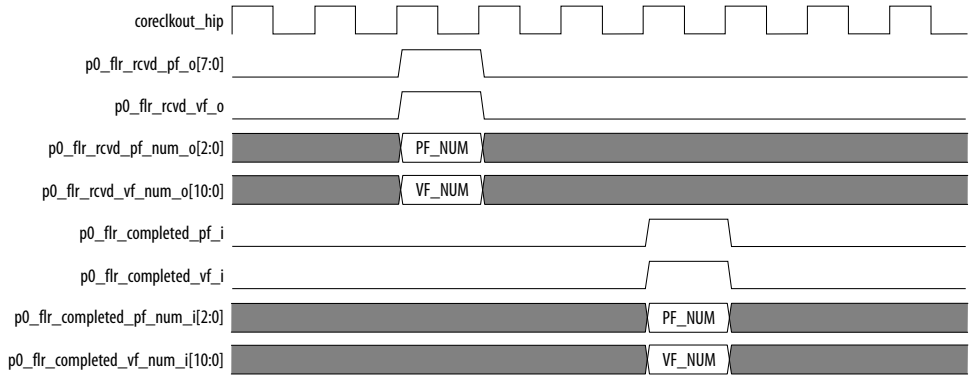
The following figure shows the timing diagram for an FLR event targeting a PF (PF2 in this example):

Figure 50. FLR for PF



Here is the timing diagram for an FLR event targeting a VF:

Figure 51. FLR for VF



5.2.2. VirtIO Support

Note: In the 19.4 release of Intel Quartus Prime, only compilation and simulation are supported for the VirtIO feature.

5.2.2.1. VirtIO Supported Features List

- VirtIO devices are implemented as PCI Express devices.
- Support 8 PFs and 2K VFs VirtIO capability structure for each EP.
- Configuration Intercept Interface in the P-Tile IP for PCIe (EP mode only) is provided for VirtIO transport.
- Five VirtIO device configuration structures are supported:
  - Common configuration
  - Notifications
  - ISR Status
  - Device-specific configuration (optional)
  - PCI configuration access
- Location of each structure is specified using a vendor-specific PCI capability located in the PCI configuration space of the device.
- VirtIO capability structure uses little-endian format.
- All fields of the VirtIO capability structure are read-only for the driver by default.
- Support PFs and VFs FLR
- Supports x16 and x8 cores.
- MSI is not supported with VirtIO.

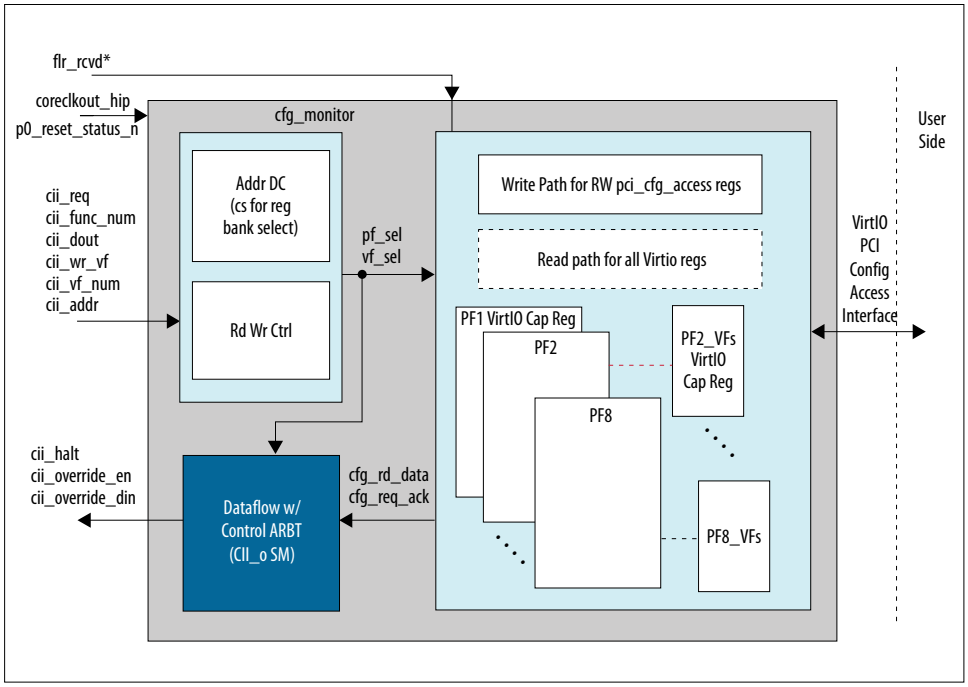
5.2.2.2. Overview

The VirtIO PCI configuration access capability creates an alternative access method to the common configuration, notifications, ISR, and device-specific configuration structure regions. This interface provides a means for the driver to access the VirtIO device region of Physical Functions (PFs) or Virtual Functions (VFs).

VirtIO is an industry standard for software-based virtualization that is supported natively by Linux. In VirtIO, software implements the virtualization stack, whereas in the case of SR-IOV, this stack is implemented mostly in hardware.

Below is the block diagram of the Soft IP which implements the VirtIO capability for PFs and VFs. This Soft IP block is automatically included when the VirtIO feature is enabled in the IP Parameter Editor.

Figure 52. VirtIO Soft IP Block Diagram



5.2.2.3. Parameters

For a detailed discussion of the VirtIO-related parameters, refer to the section VirtIO Parameters on page 34 in the *Parameters* chapter.

5.2.2.4. VirtIO PCI Configuration Access Interface

To access a VirtIO device region, pci_cfg_data will provide a window of size cap.length (1, 2 or 4 Bytes) into the given cap.bar (0x0 – 0x5) at offset cap.offset (multiple of cap.length). Detailed interfaces mapping for the user application logic are shown in the following table.

As for the VirtIO device, upon detecting a driver write access to pci_cfg_data, the user application side's VirtIO device must execute a write access at cap.offset at the BAR selected by cap.bar using the first cap.length bytes from pci_cfg_data. Moreover, upon detecting a driver read access to pci_cfg_data, the user application side's VirtIO device must execute a read access of length cap.length at cap.offset at the BAR selected by cap.bar and store the first cap.length bytes in pci_cfg_data.

Table 65. VirtIO PCI Configuration Access Interface

Name	Direction	Description	Clock Domain
`virtio_pcicfg_vfaccess_o`	O	Indicates the driver access is to a VF.	`coreclkout_hip`

continued...

Name	Direction	Description	Clock Domain
		The corresponding Virtual Function is identified from the value of virtio_pcicfg_vfnum_o.	
virtio_pcicfg_vfnum_o[VFNUM_WIDTH-1:0]	O	Indicates the corresponding Virtual Function number associated with the current Physical Function that the driver's write or read access is targeting. Validated by virtio_pcicfg_vfaccess_o and by driver write access to pci_cfg_data, or driver read access to pci_cfg_data.	coreclkout_hip
virtio_pcicfg_pfnum_o[PFNUM_WIDTH-1:0]	O	Indicates the corresponding Physical Function number that the driver's write or read access is targeting. Validated by driver write access to pci_cfg_data, or driver read access to pci_cfg_data.	coreclkout_hip
virtio_pcicfg_bar_o[7:0]	O	Indicates the BAR holding the PCI configuration access structure. The driver sets the BAR to access by writing to cap.bar. Values 0x0 to 0x5 specify a BAR belonging to the function located beginning at 10h in the PCI Configuration Space. The BAR can be either 32-bit or 64-bit. Validated by driver write access to pci_cfg_data, or driver read access to pci_cfg_data. The corresponding PF or VF is identified from the value of virtio_pcicfg_p/vfnum_o.	coreclkout_hip
virtio_pcicfg_length_o[31:0]	O	Indicates the length of the structure. The length may include padding, or fields unused by the driver, or future extensions. The driver sets the size of the access by writing 1, 2 or 4 to cap.length. Validated by driver write access to pci_cfg_data, or driver read access to pci_cfg_data. The corresponding PF or VF is identified from the value of virtio_pcicfg_p/vfnum_o.	coreclkout_hip
virtio_pcicfg_baroffset_o[31:0]	O	Indicates where the structure begins relative to the base address associated with the BAR. The driver sets the offset within the BAR by writing to cap.offset. Validated by driver write access to pci_cfg_data, or driver read access to pci_cfg_data. The corresponding PF or VF is identified from the value of virtio_pcicfg_p/vfnum_o.	coreclkout_hip
virtio_pcicfg_cfgdata_o[31:0]	O	Indicates the data for BAR access. The pci_cfg_data will provide a window of size cap.length into the given cap.bar at offset cap.offset. Validated by driver write access to pci_cfg_data, or driver read access to pci_cfg_data.	coreclkout_hip

continued...

Name	Direction	Description	Clock Domain
		The corresponding PF or VF is identified from the value of `virtio_pcicfg_p/vfnum_o`.	
`virtio_pcicfg_cfgwr_o`	O	Indicates driver write access to pci_cfg_data. The corresponding PF or VF is identified from the value of `virtio_pcicfg_p/vfnum_o`.	`coreclkout_hip`
`virtio_pcicfg_cfgrd_o`	O	Indicates driver read access to pci_cfg_data. The corresponding PF or VF is identified from the value of `virtio_pcicfg_p/vfnum_o`.	`coreclkout_hip`
`virtio_pcicfg_appvfnum_i[VFNUM_WIDTH-1:0]`	I	Indicates the corresponding Virtual Function number associated with the current Physical Function that the application config data storage is for. Validated by `virtio_pcicfg_rdack_i`.	`coreclkout_hip`
`virtio_pcicfg_apppfnum_i[PFNUM_WIDTH-1:0]`	I	Indicates the corresponding Physical Function number that the application config data storage is for. Validated by `virtio_pcicfg_rdack_i`.	`coreclkout_hip`
`virtio_pcicfg_rdack_i`	I	Indicates an application read access data ack to store the config data in pci_cfg_data. Usually the reasonable ack latency is no more than 10 cycles. The corresponding Virtual Function is identified from the value of `virtio_pcicfg_appvfnum_i`.	`coreclkout_hip`
`virtio_pcicfg_rdbe_i[3:0]`	I	Indicates application enabled bytes within `virtio_pcicfg_data_i`. Validated by `virtio_pcicfg_rdack_i`. The corresponding Virtual Function is identified from the value of `virtio_pcicfg_appvfnum_i`.	`coreclkout_hip`
`virtio_pcicfg_data_i[31:0]`	I	Indicates application data to be stored in PCI Configuration Access data registers. Validated by `virtio_pcicfg_rdack_i` and `virtio_pcicfg_rdbe_i`. The corresponding Virtual Function is identified from the value of `virtio_pcicfg_appvfnum_i`.	`coreclkout_hip`

5.2.2.5. Registers

The following VirtIO capability structure registers references apply to each PF and VF. Addresses shown are register addresses.

Table 66. PF/VF Capability Link List

Capability	Start Byte Address	Last Byte Address	DW Count
Type0	0x00	0x3F	16
PM (PF only)	0x40	0x47	2

continued...

Capability	Start Byte Address	Last Byte Address	DW Count
VirtIO Common Configuration	0x48	0x57	4
VirtIO Notifications	0x58	0x6B	5
Reserved	0x6C	0x6F	1
PCIe	0x70	0xA3	13
Reserved	0xA4	0xAF	3
MSIX	0xB0	0xBB	3
VirtIO ISR Status	0xBC	0xCB	4
VirtIO Device-Specific Configuration	0xCC	0xDB	4
VirtIO PCI Configuration Access	0xDC	0xEF	5
Reserved	0xF0	0xFF	4

Table 67. VirtIO Common Configuration Capability Structure

Address	Name	Description
012	Common Configuration Capability Register	Capability ID, next capability pointer, capability length
013	BAR Indicator Register	Lower 8 bits indicate which BAR holds the structure
014	BAR Offset Register	Indicates starting address of the structure within the BAR
015	Structure Length Register	Indicates length of structure
VirtIO Notifications Capability Structure		
016	Notifications Capability Register	Capability ID, next capability pointer, capability length
017	BAR Indicator Register	Lower 8 bits indicate which BAR holds the structure
018	BAR Offset Register	Indicates starting address of the structure within the BAR
019	Structure Length Register	Indicates length of structure
01A	Notify Off Multiplier	Multiplier for queue_notify_off
VirtIO ISR Status Capability Structure		
02F	ISR Status Capability Register	Capability ID, next capability pointer, capability length
030	BAR Indicator Register	Lower 8 bits indicate which BAR holds the structure
031	BAR Offset Register	Indicates starting address of the structure within the BAR
032	Structure Length Register	Indicates length of structure
VirtIO Device-Specific Capability Structure (Optional)		
033	Device Specific Capability Register	Capability ID, next capability pointer, capability length
034	BAR Indicator Register	Lower 8 bits indicate which BAR holds the structure
035	BAR Offset Register	Indicates starting address of the structure within the BAR
036	Structure Length Register	Indicates length of structure
VirtIO PCI Configuration Access Structure		

continued...

Address	Name	Description
037	PCI Configuration Access Capability Register	Capability ID, next capability pointer, capability length
038	BAR Indicator Register	Lower 8 bits indicate which BAR holds the structure
039	BAR Offset Register	Indicates starting address of the structure within the BAR
03A	Structure Length Register	Indicates length of structure
03B	PCI Configuration Data	Data for BAR access

5.2.2.5.1. VirtIO Common Configuration Capability Register (Address: 0x012)

The capability register identifies that this is a vendor-specific capability. It also identifies the structure type.

Table 68. VirtIO Common Configuration Capability Register

Bit Location	Description	Access Type	Default Value
31:24	Configuration Type	RO	0x01
23:16	Capability Length	RO	0x10
15:8	Next Capability Pointer	RO	0x58
7:0	Capability ID	RO	0x09

5.2.2.5.2. VirtIO Common Configuration BAR Indicator Register (Address: 0x013)

The Bar Indicator field holds the values 0x0 to 0x5 specifying a Base Address register (BAR) belonging to the function located beginning at 10h in PCI Configuration Space. The BAR is used to map the structure into the memory space. Any other value is reserved for future use.

Table 69. VirtIO Common Configuration BAR Indicator Register

Bit Location	Description	Access Type	Default Value
31:24	Padding	RO	0x00
23:16	Padding	RO	0x00
15:8	Padding	RO	0x00
7:0	BAR Indicator	RO	Settable through Platform Designer

5.2.2.5.3. VirtIO Common Configuration BAR Offset Register (Address: 0x014)

This register indicates where the structure begins relative to the base address associated with the BAR. The alignment requirements of the offset are indicated in each structure-specific section.

Table 70. VirtIO Common Configuration BAR Offset Register

Bit Location	Description	Access Type	Default Value
31:0	BAR Offset	RO	Settable through Platform Designer

5.2.2.5.4. VirtIO Common Configuration Structure Length Register (Address 0x015)

The length register indicates the length of the structure. The length may include padding, fields unused by the driver, or future extensions.

Table 71. VirtIO Common Configuration Structure Length Register

Bit Location	Description	Access Type	Default Value
31:0	Structure Length	RO	Settable through Platform Designer

5.2.2.5.5. VirtIO Notifications Capability Register (Address: 0x016)

The capability register identifies that this is a vendor-specific capability. It also identifies the structure type.

Table 72. VirtIO Notifications Capability Register

Bit Location	Description	Access Type	Default Value
31:24	Configuration Type	RO	0x02
23:16	Capability Length	RO	0x14
15:8	Next Capability Pointer	RO	0xBC
7:0	Capability ID	RO	0x09

5.2.2.5.6. VirtIO Notifications BAR Indicator Register (Address: 0x017)

The Bar Indicator field holds the values 0x0 to 0x5 specifying a Base Address register (BAR) belonging to the function located beginning at 10h in PCI Configuration Space. The BAR is used to map the structure into memory space. Any other value is reserved for future use.

Table 73. VirtIO Notifications BAR Indicator Register

Bit Location	Description	Access Type	Default Value
31:24	Padding	RO	0x00
23:16	Padding	RO	0x00
15:8	Padding	RO	0x00
7:0	BAR Indicator	RO	Settable through Platform Designer

5.2.2.5.7. VirtIO Notifications BAR Offset Register (Address: 0x018)

This register indicates where the structure begins relative to the base address associated with the BAR. The alignment requirements of the offset are indicated in each structure-specific section.

Table 74. VirtIO Notifications BAR Offset Register

Bit Location	Description	Access Type	Default Value
31:0	BAR Offset	RO	Settable through Platform Designer

5.2.2.5.8. VirtIO Notifications Structure Length Register (Address: 0x019)

The length register indicates the length of the structure. The length may include padding, fields unused by the driver, or future extensions.

Table 75. VirtIO Notifications Structure Length Register

Bit Location	Description	Access Type	Default Value
31:0	Structure Length	RO	Settable through Platform Designer

5.2.2.5.9. VirtIO Notifications Notify Off Multiplier Register (Address: 0x01A)

The notify off multiplier register indicates the multiplier for queue_notify_off in the structure.

Table 76. VirtIO Notifications Notify Off Multiplier Register

Bit Location	Description	Access Type	Default Value
31:0	Multiplier for queue_notify_off	RO	Settable through Platform Designer

5.2.2.5.10. VirtIO ISR Status Capability Register (Address: 0x02F)

The capability register identifies that this is a vendor-specific capability. It also identifies the structure type.

Table 77. VirtIO ISR Status Capability Register

Bit Location	Description	Access Type	Default Value
31:24	Configuration Type	RO	0x03
23:16	Capability Length	RO	0x10
15:8	Next Capability Pointer	RO	If Device-Specific Capability is present, then points to 0xCC, else points to 0xDC.
7:0	Capability ID	RO	0x09

5.2.2.5.11. VirtIO ISR Status BAR Indicator Register (Address: 0x030)

The Bar Indicator field holds the values 0x0 to 0x5 specifying a Base Address register (BAR) belonging to the function located beginning at 10h in PCI Configuration Space. The BAR is used to map the structure into memory space. Any other value is reserved for future use.

Table 78. VirtIO ISR Status BAR Indicator Register

Bit Location	Description	Access Type	Default Value
31:24	Padding	RO	0x00
23:16	Padding	RO	0x00
15:8	Padding	RO	0x00
7:0	BAR Indicator	RO	Settable through Platform Designer

5.2.2.5.12. VirtIO ISR Status BAR Offset Register (Address: 0x031)

This register indicates where the structure begins relative to the base address associated with the BAR. The alignment requirements of the offset are indicated in each structure-specific section.

Table 79. VirtIO ISR Status BAR Offset Register

Bit Location	Description	Access Type	Default Value
31:0	BAR Offset	RO	Settable through Platform Designer

5.2.2.5.13. VirtIO ISR Status Structure Length Register (Address: 0x032)

The length register indicates the length of the structure. The length may include padding, fields unused by the driver, or future extensions.

Table 80. VirtIO ISR Status Structure Length Register

Bit Location	Description	Access Type	Default Value
31:0	Structure Length	RO	Settable through Platform Designer

5.2.2.5.14. VirtIO Device Specific Capability Register (Address: 0x033)

The capability register identifies that this is vendor-specific capability. It also identifies the structure type.

Table 81. VirtIO Device Specific Capability Register

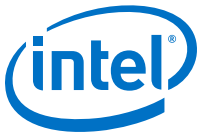
Bit Location	Description	Access Type	Default Value
31:24	Configuration Type	RO	0x04
23:16	Capability Length	RO	0x10
15:8	Next Capability Pointer	RO	If this capability is present, then points to 0xDC.
7:0	Capability ID	RO	0x09

5.2.2.5.15. VirtIO Device Specific BAR Indicator Register (Address: 0x034)

The BAR Indicator field holds the values 0x0 to 0x5 specifying a Base Address register (BAR) belonging to the function located beginning at 10h in PCI Configuration Space. The BAR is used to map the structure into memory space. Any other value is reserved for future use.

Table 82. VirtIO Device Specific BAR Indicator Register

Bit Location	Description	Access Type	Default Value
31:24	Padding	RO	0x00
23:16	Padding	RO	0x00
15:8	Padding	RO	0x00
7:0	BAR Indicator	RO	Settable through Platform Designer



5.2.2.5.16. VirtIO Device Specific BAR Offset Register (Address 0x035)

This register indicates where the structure begins relative to the base address associated with the BAR. The alignment requirements of the offset are indicated in each structure-specific section.

Table 83. VirtIO Device Specific BAR Offset Register

Bit Location	Description	Access Type	Default Value
31:0	BAR Offset	RO	Settable through Platform Designer

5.2.2.5.17. VirtIO Device Specific Structure Length Register (Address: 0x036)

The length register indicates the length of the structure. The length may include padding, fields unused by the driver, or future extensions.

Table 84. VirtIO Device Specific Structure Length Register

Bit Location	Description	Access Type	Default Value
31:0	Structure Length	RO	Settable through Platform Designer

5.2.2.5.18. VirtIO PCI Configuration Access Capability Register (Address: 0x037)

The capability register identifies that this is a vendor-specific capability. It also identifies the structure type.

Table 85. VirtIO PCI Configuration Access Capability Register

Bit Location	Description	Access Type	Default Value
31:24	Configuration Type	RO	0x05
23:16	Capability Length	RO	0x14
15:8	Next Capability Pointer	RO	0x00
7:0	Capability ID	RO	0x09

5.2.2.5.19. VirtIO PCI Configuration Access BAR Indicator Register (Address: 0x038)

The BAR Indicator field holds the values 0x0 to 0x5 specifying a Base Address register (BAR) belonging to the function located beginning at 10h in PCI Configuration Space. The BAR is used to map the structure into memory space. Any other value is reserved for future use.

Table 86. VirtIO PCI Configuration Access BAR Indicator Register

Bit Location	Description	Access Type	Default Value
31:24	Padding	RO	0x00
23:16	Padding	RO	0x00
15:8	Padding	RO	0x00
7:0	BAR Indicator	RW	Settable through Platform Designer

5.2.2.5.20. VirtIO PCI Configuration Access BAR Offset Register (Address: 0x039)

This register indicates where the structure begins relative to the base address associated with the BAR. The alignment requirements of the offset are indicated in each structure-specific section.

Table 87. VirtIO PCI Configuration Access BAR Offset Register

Bit Location	Description	Access Type	Default Value
31:0	BAR Offset	RW	Settable through Platform Designer

5.2.2.5.21. VirtIO PCI Configuration Access Structure Length Register (Address: 0x03A)

The length register indicates the length of the structure. The length may include padding, fields unused by the driver, or future extensions.

Table 88. VirtIO PCI Configuration Access Structure Length Register

Bit Location	Description	Access Type	Default Value
31:0	Structure Length	RW	Settable through Platform Designer

5.2.2.5.22. VirtIO PCI Configuration Access Data Register (Address: 0x03B)

The PCI configuration data register indicates the data for BAR access.

Table 89. VirtIO PCI Configuration Access Data Register

Bit Location	Description	Access Type	Default Value
31:0	PCI Configuration Data	RW	Settable through Platform Designer

5.3. TLP Bypass Mode

The P-Tile Avalon-ST IP for PCIe includes a TLP Bypass mode for both downstream and upstream ports to allow the implementation of advanced features such as:

- The upstream port or the downstream port of a switch.
- A custom implementation of a Transaction Layer to meet specific user requirements.

Table 90. Supported TLP Bypass Configurations

UP = upstream port; DN = downstream port

IP Mode	Port Mode
X16	UP DN
X8	UP/UP UP/DN DN/UP DN/DN
X4	UP/UP/UP/UP DN/DN/DN/DN

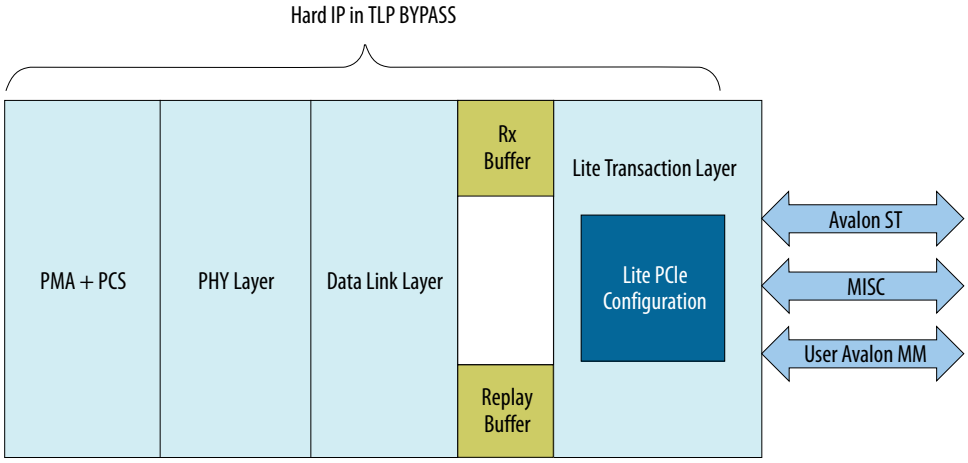
5.3.1. Overview

When the TLP Bypass feature is enabled, the P-Tile Avalon-ST IP does not process received TLPs internally but outputs them to the user application. This allows the application to implement a custom Transaction Layer.

The P-tile Avalon-ST IP in TLP Bypass mode still includes some of the PCIe configuration space registers related to link operation (refer to the Configuration Space Registers on page 180 chapter for the list of registers).

P-Tile interfaces with the application logic via the Avalon-ST interface (for all TLP traffic), the User Avalon-MM interface (for Lite TL's configuration registers access) and other miscellaneous signals.

Figure 53. P-Tile Avalon-ST IP in TLP Bypass Mode



In TLP bypass mode, P-Tile supports the autonomous Hard IP feature. It responds to configuration accesses before the FPGA fabric enters user mode with Completions with a CRS code.

However, in TLP bypass mode, CvP init and update are not supported.

5.3.2. Register Settings for the TLP Bypass Mode

When TLP Bypass mode is enabled, some error detection is still performed in the Physical and Link Layers inside the Hard IP. Per PCIe specification, the Hard IP must report these errors on the configuration space registers (in the AER Capability Structure). The P-tile IP for PCIe includes two registers called TLPBYPASS_ERR_EN and TLPBYPASS_ERR_STATUS to report errors detected while in TLP Bypass mode.

TLPBYPASS_ERR_EN and TLPBYPASS_ERR_STATUS are part of the configuration and status register.

5.3.2.1. TLPBYPASS_ERR_EN (Address 0x104194)

This register allows you to enable or disable error reporting. When this feature is disabled, the TLPBYPASS_ERR_ STATUS bits associated with an error are not set when the error is detected.

Table 91. TLPBYPASS_ERR_EN Register (Address 0x104194)

Name	Bits	Reset Value	Access Mode	Register Description
Reserved	[31:20]	12'b0	RO	Reserved
k_cfg_uncor_internal_err_sts_en	[19]	1'b1	RW	Enable error indication on serr_out_o for Uncorrectable Internal Error.
k_cfg_corrected_internal_err_sts_en	[18]	1'b1	RW	Enable error indication on serr_out_o for Corrected Internal Error.
k_cfg_rcvr_overflow_err_sts_en	[17]	1'b1	RW	Enable error indication on serr_out_o for Receiver Overflow Error.
k_cfg_fc_protocol_err_sts_en	[16]	1'b1	RW	Enable error indication on serr_out_o for Flow Control Protocol Error.
k_cfg_mlf_tlp_err_sts_en	[15]	1'b1	RW	Enable error indication on serr_out_o for Malformed TLP Error.
k_cfg_surprise_down_err_sts_en	[14]	1'b1	RW	Enable error indication on serr_out_o for Surprise Down Error.
k_cfg_dl_protocol_err_sts_en	[13]	1'b1	RW	Enable error indication on serr_out_o for Data Link Protocol Error.
k_cfg_replay_number_rollover_err_sts_en	[12]	1'b1	RW	Enable error indication on serr_out_o for REPLAY_NUM Rollover Error.
k_cfg_replay_timer_timeout_err_st_en	[11]	1'b1	RW	Enable error indication on serr_out_o for Replay Timer Timeout Error.
k_cfg_bad_dllp_err_sts_en	[10]	1'b1	RW	Enable error indication on serr_out_o for Bad DLLP Error.
k_cfg_bad_tlp_err_sts_en	[9]	1'b1	RW	Enable error indication on serr_out_o for Bad TLP Error.
k_cfg_rcvr_err_sts_en	[8]	1'b1	RW	Enable error indication on serr_out_o for Receiver Error.
Reserved	[7:1]	7'b0	RO	Reserved
k_cfg_ecrc_err_sts_en	[0]	1'b1	RW	Enable error indication on serr_out_o for ECRC Error.

5.3.2.2. TLPBYPASS_ERR_STATUS (Address 0x104190)

When an error is detected, Intel recommends that you read the PF0 AER register inside P-Tile to get detailed information about the error. To clear the previous error status, you need to clear TLPBYPASS_ERR_STATUS and the corresponding correctable and uncorrectable error status registers in the AER capability structure. After doing that, you can get the new error update from this register.

Table 92. TLPBYPASS_ERR_STATUS Register (Address 0x104190)

Name	Bits	Reset Value	Access Mode	Register Description
Reserved	[31:20]	12'b0	RO	Reserved
cfg_uncor_internal_err_sts	[19]	1'b0	W1C	Uncorrectable Internal Error
cfg_corrected_internal_err_sts	[18]	1'b0	W1C	Corrected Internal Error

continued...

Name	Bits	Reset Value	Access Mode	Register Description
cfg_rcvr_overflow_err_sts	[17]	1'b0	W1C	Receiver Overflow Error
cfg_fc_protocol_err_sts	[16]	1'b0	W1C	Flow Control Protocol Error
cfg_mlf_tlp_err_sts	[15]	1'b0	W1C	Malformed TLP Error
cfg_surprise_down_err_sts	[14]	1'b0	W1C	Surprise Down Error. Available in downstream mode only.
cfg_dl_protocol_err_sts	[13]	1'b0	W1C	Data Link Protocol Error
cfg_replay_number_rollover_err_sts	[12]	1'b0	W1C	REPLAY_NUM Rollover Error
cfg_replay_timer_timeout_err_sts	[11]	1'b0	W1C	Replay Timer Timeout Error
cfg_bad_dllp_err_sts	[10]	1'b0	W1C	Bad DLLP Error
cfg_bad_tlp_err_sts	[9]	1'b0	W1C	Bad TLP Error
cfg_rcvr_err_sts	[8]	1'b0	W1C	Receiver Error
Reserved	[7:1]	7'b0	RO	Reserved
cfg_ecrc_err_sts	[0]	1'b0	W1C	ECRC Error

5.3.3. User Avalon-MM Interface

For more details on the signals in this interface, refer to the section Hard IP Reconfiguration Interface on page 86.

The majority of the PCIe standard registers are implemented in the user's logic outside of the P-Tile Avalon-ST IP.

However, the following registers still remain inside the P-Tile:

- Power management capability
- PCI Express capability
- Secondary PCI Express capability
- Data link feature extended capability
- Physical layer 16.0GT/s extended capability
- Lane margining at the receiver extended capability
- Advanced error reporting capability

The application can only access PCIe controller registers through the User Avalon-MM interface.

Table 93. Capability Registers to be Updated by the Application Logic via the User Avalon-MM Interface

Capability	Comments
Power Management Capability	Need to write back since it is required to trigger a PCI-PM entry.
PCI Express Capability	All the PCIe capabilities, control and status registers are for configuring the device. Write-back is required.
Secondary PCI Express Capability	Secondary PCIe Capability is required for configuring the device.
Data Link Feature Extended Capability	Data Link Capability is device specific.
Physical Layer 16.0 GT/s Extended Capability	Physical Layer 16G Capability is device specific.
Lane Margining at the Receiver Extended Capability	Margining Extended Capability is device specific.
Advanced Error Reporting Capability	Write-back to error status registers is required for TLP Bypass.

Note: Refer to Table 109 on page 180 for the address offsets information for the capability registers listed in the table above.

5.3.4. Avalon Streaming (Avalon-ST) Interface

For more details on the signals in this interface, refer to the section Avalon-ST Interface on page 46.

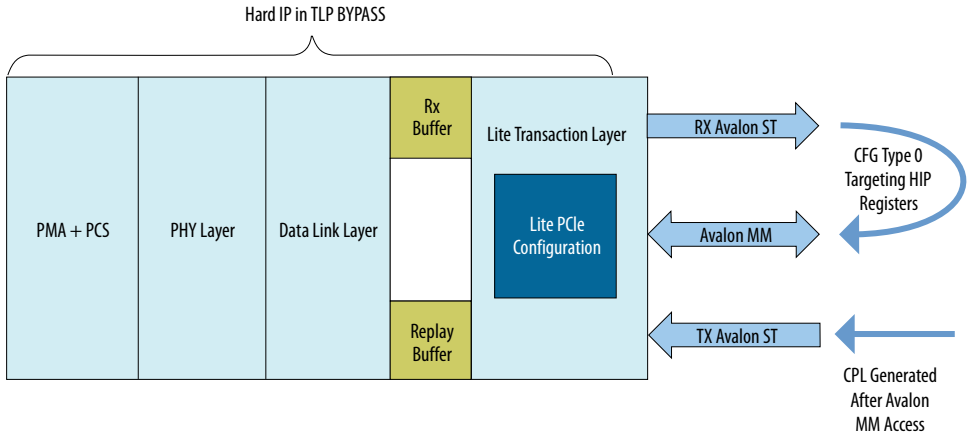
5.3.4.1. Configuration TLP

The P-Tile IP forwards any received Type0/1 Configuration TLP to the Avalon-ST RX streaming interface. User's logic has the responsibility to respond with a Completion TLP with a Completion code of Successful Completion (SC), Unsupported Request (UR), Configuration Request Retry Status (CRS), or Completer Abort (CA).

If a Configuration TLP needs to update a register in the PCIe configuration space in the P-Tile PCIe Hard IP, you need to use the User Avalon-MM interface.

The application needs to prevent link programming side effects such as writing into low-power states before sending the Completion associated with the request. The application logic can check the TX FIFO empty flag in the `tx_cdts_limit_o` after the Completion enters the TX streaming interface to confirm that the TLP has been sent. For more details on the User Avalon-MM interface, refer to the section *Hard IP Reconfiguration Interface (User Avalon-MM Interface)*.

Figure 54. Configuration TLP Received by P-Tile IP for PCIe Targeting the Hard IP Internal Registers



5.3.4.2. Transmit Interface

All TLPs transmitted by the application through the TX streaming interface are sent out as-is, without any tracking for completion. The P-Tile IP for PCIe does not perform any check on the TLPs. Your application logic is responsible for sending TLPs that comply with the PCIe specifications.

5.3.4.3. Receive Interface

ALL TLPs received by the IP are transmitted to the application through the RX streaming interface (except Malformed TLPs).

Please refer to the Packets Forwarded to the User Application in TLP Bypass Mode on page 200 Appendix for detailed information.

All PCIe protocol errors leading up to designating a TLP packet as a good packet or not will be detected by the Hard IP and communicated to user logic to take appropriate action in terms of error logging and escalation. The IP does not generate any error message internally, since this is the responsibility of the user logic.

5.3.4.4. Malformed TLP

In TLP Bypass mode, a malformed TLP is dropped in the P-Tile IP for PCIe and its event is logged in the AER capability registers. P-Tile also notifies you of this event by asserting the `serr_out_o` signal.

Refer to the *PCI Express Base Specification* for the definition of a malformed TLP.

5.3.4.5. ECRC

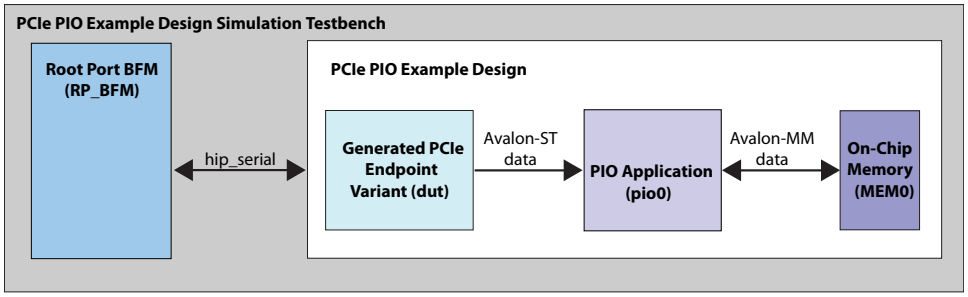
In TLP bypass mode, the ECRC is not generated or stripped by the P-Tile Avalon-ST IP for PCIe.

6. Quick Start Guide

Using the Intel Quartus Prime software, you can generate a programmed I/O (PIO) design example for the P-Tile Avalon-ST IP for PCI Express. The generated design example reflects the parameters that you specify. The PIO example transfers data from a host processor to a target device. This design example automatically creates the files necessary to simulate and compile in the Intel Quartus Prime software. You can download the compiled design to the Intel Development Board. To download to custom hardware, update the Intel Quartus Prime Settings File (`.qsf`) with the correct pin assignments.

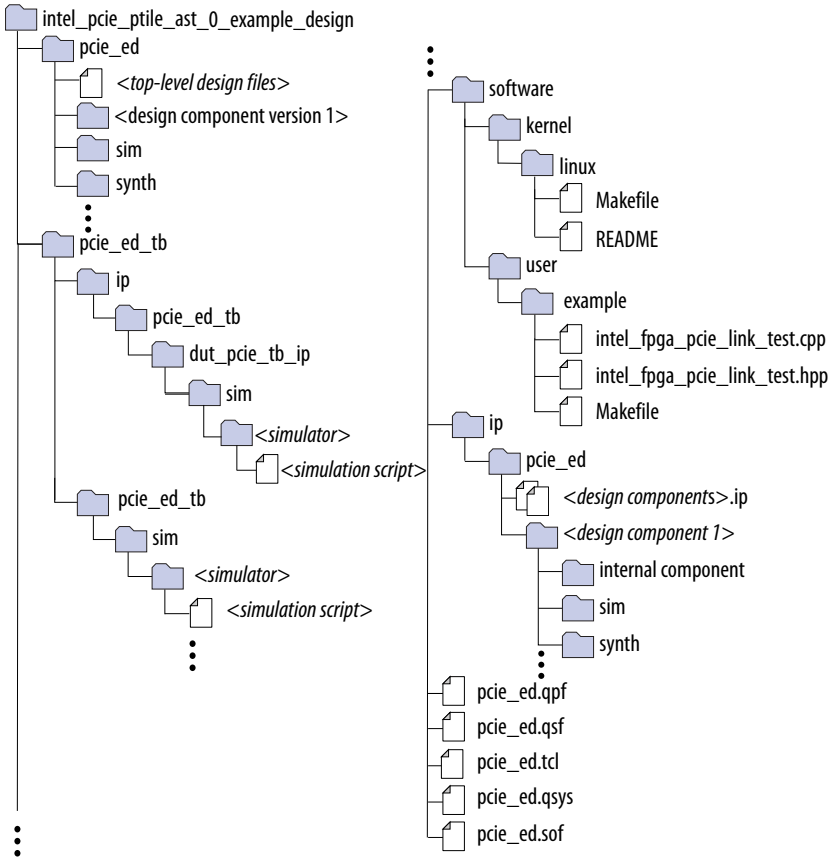
6.1. Design Components

Figure 55. Block Diagram for the Platform Designer Programmed Input/Output (PIO) Design Example Simulation Testbench



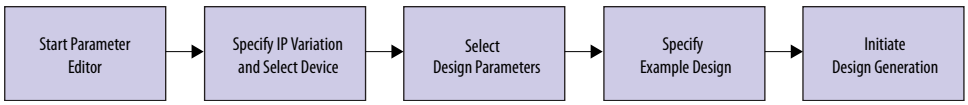
6.2. Directory Structure

Figure 56. Directory Structure for the Generated Design Example



6.3. Generating the Design Example

Figure 57. Design Example Generation Procedure

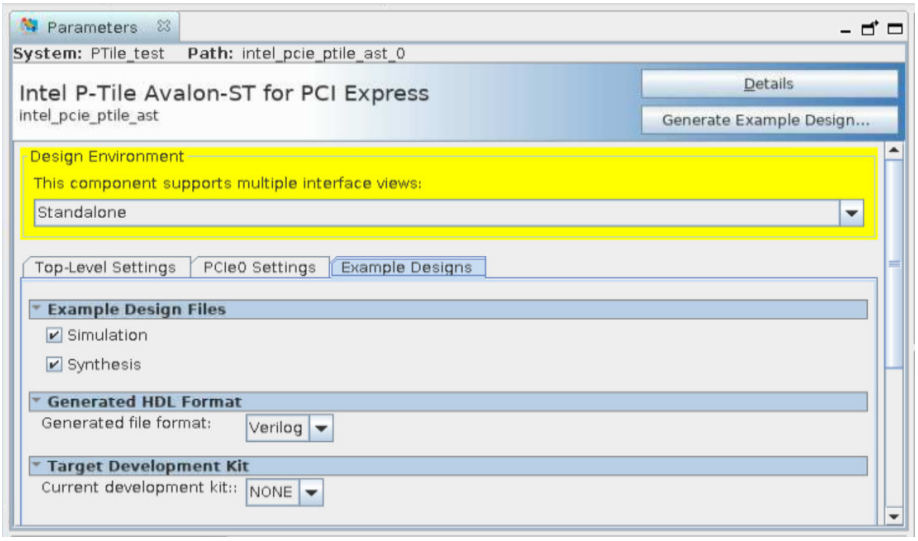


1. In the Intel Quartus Prime Pro Edition software, create a new project (**File → New Project Wizard**).
2. Specify the **Directory Name**, and **Top-Level Entity.**
3. For **Project Type**, accept the default value, **Empty project**. Click **Next**.
4. For **Add Files** click **Next**.

5. For **Family, Device & Board Settings** under **Family**, select **Intel Agilex** or **Intel Stratix 10**.
6. If you select **Intel Stratix 10** in the last step, select **Stratix 10 DX** in the **Device** pull-down menu.
7. Select the **Target Device** for your design.
8. Click **Finish**.
9. In the IP Catalog, go to **Interface Protocols**, then to **PCI Express**. Locate and add the **Intel P-Tile Avalon-ST IP for PCI Express**.
10. In the **New IP Variant** dialog box, specify a name for your IP. Click **Create**.
11. On the **Top-Level Settings** and **PCIe* Settings** tabs, specify the parameters for your IP variation.
12. On the **Example Designs** tab, make the following selections:
    a. For **Example Design Files**, turn on the **Simulation** and **Synthesis** options. If you do not need these simulation or synthesis files, leaving the corresponding option(s) turned off significantly reduces the example design generation time.
    b. For **Generated HDL Format**, only Verilog is available in the current release.
    c. For **Target Development Kit**, select the appropriate option.

    Note: For the current release of the IP core, hardware testing is not supported. Therefore, select **None** for the target development kit. The generated design example will target the device you specified in Step 5 above.
13. Select **Generate Example Design** to create a design example that you can simulate. When the prompt asks you to specify the directory for your example design, you can accept the default directory, <example_design>/intel_pcie_ptile_ast_0_example_design, or choose another directory.

Figure 58. Example Designs Tab

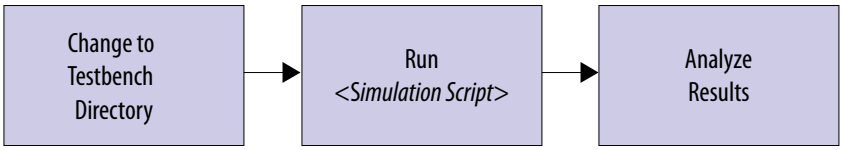


14. Click **Finish**. You may save your .ip file when prompted, but it is not required to be able to use the example design.

15. Open the example design project.
16. Compile the example design project to generate the .sof file for the complete example design. This file is what you download to a board to perform hardware verification.
17. Close your example design project.

6.4. Simulating the Design Example

Figure 59. Procedure



1. Change to the testbench simulation directory, `pcie_ed_tb`.
2. Run the simulation script for VCS. Refer to the table below.
3. Analyze the results.

Table 94. Steps to Run Simulation

Simulator	Working Directory	Instructions
VCS*	`<example_design>/pcie_ed_tb/pcie_ed_tb/sim/synopsys/vcs`	1. `sh vcs_setup.sh USER_DEFINED_COMPILE_OPTIONS="" USER_DEFINED_ELAB_OPTIONS="-xlrm\ uniq_prior_final" USER_DEFINED_SIM_OPTIONS=""` 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!" *Note:* To run a simulation in interactive mode, use this command: `sh vcs_setup.sh USER_DEFINED_COMPILE_OPTIONS="-debug_access+all -debug_pp" USER_DEFINED_ELAB_OPTIONS="-xlrm \ uniq_prior_final" USER_DEFINED_SIM_OPTIONS="-gui"`

This testbench simulates up to a Gen4 x16 variant.

The simulation reports, "Simulation stopped due to successful completion" if no errors occur.

6.5. Compiling the Design Example

1. Navigate to `<project_dir>/intel_pcie_ptile_ast_0_example_design/` and open `pcie_ed.qpf`.
2. If you are using the Intel Stratix 10 DX development kit, add the following assignments in the `.qsf` file of your project for the VID feature:
   - set_global_assignment -name USE_CONF_DONE SDM_IO16
   - set_global_assignment -name VID_OPERATION_MODE "PMBUS MASTER"
   - set_global_assignment -name USE_PWRMGT_SCL SDM_IO0

- set_global_assignment -name USE_PWRMGT_SDA SDM_IO12
- set_global_assignment -name PWRMGT_BUS_SPEED_MODE "100 KHZ"
- set_global_assignment -name PWRMGT_SLAVE_DEVICE_TYPE OTHER
- set_global_assignment -name PWRMGT_SLAVE_DEVICE0_ADDRESS 60
- set_global_assignment -name PWRMGT_SLAVE_DEVICE1_ADDRESS 00
- set_global_assignment -name PWRMGT_SLAVE_DEVICE2_ADDRESS 00
- set_global_assignment -name PWRMGT_SLAVE_DEVICE3_ADDRESS 00
- set_global_assignment -name PWRMGT_SLAVE_DEVICE4_ADDRESS 00
- set_global_assignment -name PWRMGT_SLAVE_DEVICE5_ADDRESS 00
- set_global_assignment -name PWRMGT_SLAVE_DEVICE6_ADDRESS 00
- set_global_assignment -name PWRMGT_SLAVE_DEVICE7_ADDRESS 00
- set_global_assignment -name PWRMGT_VOLTAGE_OUTPUT_FORMAT "DIRECT FORMAT"
- set_global_assignment -name PWRMGT_DIRECT_FORMAT_COEFFICIENT_M 1
- set_global_assignment -name PWRMGT_DIRECT_FORMAT_COEFFICIENT_R 0
- set_global_assignment -name PWRMGT_TRANSLATED_VOLTAGE_VALUE_UNIT MILLIVOLTS
- set_global_assignment -name PWRMGT_PAGE_COMMAND_ENABLE ON

3. If you are using the Intel Agilex development kit, add the following assignments in the `.qsf` file of your project for the VID feature:
   - set_global_assignment -name USE_CONF_DONE SDM_IO16
   - set_global_assignment -name USE_INIT_DONE SDM_IO0
   - set_global_assignment -name USE_CVP_CONFDONE SDM_IO15
   - set_global_assignment -name VID_OPERATION_MODE "PMBUS MASTER"
   - set_global_assignment -name USE_PWRMGT_SCL SDM_IO14
   - set_global_assignment -name USE_PWRMGT_SDA SDM_IO11
   - set_global_assignment -name PWRMGT_BUS_SPEED_MODE "100 KHZ"
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE_TYPE OTHER
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE0_ADDRESS 47
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE1_ADDRESS 00
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE2_ADDRESS 00
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE3_ADDRESS 00
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE4_ADDRESS 00
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE5_ADDRESS 00
   - set_global_assignment -name PWRMGT_SLAVE_DEVICE6_ADDRESS 00

- set_global_assignment -name PWRMGT_SLAVE_DEVICE7_ADDRESS 00
- set_global_assignment -name PWRMGT_TRANSLATED_VOLTAGE_VALUE_UNIT VOLTS
- set_global_assignment -name PWRMGT_PAGE_COMMAND_ENABLE ON

4. On the Processing menu, select **Start Compilation**.

6.6. Installing the Linux Kernel Driver

Before you can test the design example in hardware, you must install the Linux kernel driver. You can use this driver to perform the following tests:

- A PCIe link test that performs 100 writes and reads
- Memory space DWORD(3) reads and writes
- Configuration Space DWORD reads and writes

In addition, you can use the driver to change the value of the following parameters:

- The BAR being used
- The selected device by specifying the bus, device and function (BDF) numbers for that device

The driver also allows you to enable SR-IOV for P-Tile devices.

Complete the following steps to install the kernel driver:

1. Navigate to `./software/kernel/linux` under the example design generation directory.
2. Change the permissions on the `install`, `load`, and `unload` files:

   ```
   $ chmod 777 install load unload
   ```
3. Install the driver:

   ```
   $ sudo ./install
   ```
4. Verify the driver installation:

   ```
   $ lsmod | grep intel_fpga_pcie_drv
   ```

   Expected result:

   ```
   intel_fpga_pcie_drv 17792 0
   ```
5. Verify that Linux recognizes the PCIe design example:

   ```
   $ lspci -d 1172:000 -v | grep intel_fpga_pcie_drv
   ```

   Note: If you have changed the Vendor ID, substitute the new Vendor ID for Intel's Vendor ID in this command.

   Expected result:

   ```
   Kernel driver in use: intel_fpga_pcie_drv
   ```

(3) Throughout this user guide, the terms word, DWORD and QWORD have the same meaning that they have in the PCI Express Base Specification. A word is 16 bits, a DWORD is 32 bits, and a QWORD is 64 bits.

6.7. Running the Design Example Application

1. Navigate to `./software/user/example` under the design example directory.
2. Compile the design example application:

   ```
   $ make
   ```

3. Run the test:

   ```
   $ sudo ./intel_fpga_pcie_link_test
   ```

   You can run the Intel FPGA IP PCIe link test in manual or automatic mode.

   — In automatic mode, the application automatically selects the device. The test selects the Intel Stratix 10 DX or Intel Agilex PCIe device with the lowest BDF by matching the Vendor ID. The test also selects the lowest available BAR.

   — In manual mode, the test queries you for the bus, device, and function number and BAR.

   For the Intel Stratix 10 DX or Intel Agilex Development Kit, you can determine the BDF by typing the following command:

   ```
   $ lspci -d 1172
   ```

4. Here are sample transcripts for automatic and manual modes:

```
Intel FPGA PCIe Link Test - Automatic Mode
Version 2.0
0: Automatically select a device
1: Manually select a device
**************************************************
>0
Opened a handle to BAR 0 of a device with BDF 0x100
**************************************************
0: Link test - 100 writes and reads
1: Write memory space
2: Read memory space
3: Write configuration space
4: Read configuration space
5: Change BAR
6: Change device
7: Enable SR-IOV
8: Do a link test for every enabled virtual function
   belonging to the current device
9: Perform DMA
10: Quit program
**************************************************
> 0
Doing 100 writes and 100 reads . .
Number of write errors:     0
Number of read errors:      0
Number of DWORD mismatches: 0
```

```
Intel FPGA PCIe Link Test - Manual Mode
Version 1.0
0: Automatically select a device
1: Manually select a device
**************************************************
> 1
Enter bus number:
> 1
Enter device number:
> 0
Enter function number:
> 0
BDF is 0x100
```

```
Enter BAR number (-1 for none):
> 4
Opened a handle to BAR 4 of a device with BDF 0x100
```

7. Testbench and Design Example

This chapter introduces the Endpoint design example including a testbench and a test driver module. You can create this design example using design flows described in *Quick Start Guide*.

The testbench in this design example simulates up to a Gen4 x16 variant.

When configured as an Endpoint variation, the testbench instantiates a design example with a P-Tile Endpoint and a Root Port BFM containing a second P-Tile (configured as a Root Port) to interface with the Endpoint. The Root Port BFM provides the following functions:

- A configuration routine that sets up all the basic configuration registers in the Endpoint. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint.

This testbench simulates the scenario of a single Root Port talking to a single Endpoint.

The testbench uses a test driver module, `altpcietb_bfm_rp_gen3_x8.sv`, to initiate the configuration and memory transactions. At startup, the test driver module displays information from the Root Port and Endpoint Configuration Space registers, so that you can correlate to the parameters you specified using the parameter editor.

Note: The Intel testbench and Root Port BFM provide a simple method to do basic testing of the Application Layer logic that interfaces to the variation. This BFM allows you to create and run simple task stimuli with configurable parameters to exercise basic functionality of the Intel example design. The testbench and Root Port BFM are not intended to be a substitute for a full verification environment. Corner cases and certain traffic profile stimuli are not covered. Refer to the items listed below for further details. To ensure the best verification coverage possible, Intel suggests strongly that you obtain commercially available PCI Express verification IP and tools, in combination with performing extensive hardware testing.

Your Application Layer design may need to handle at least the following scenarios that are not possible to create with the Intel testbench and the Root Port BFM, or are due to the limitations of the example design:

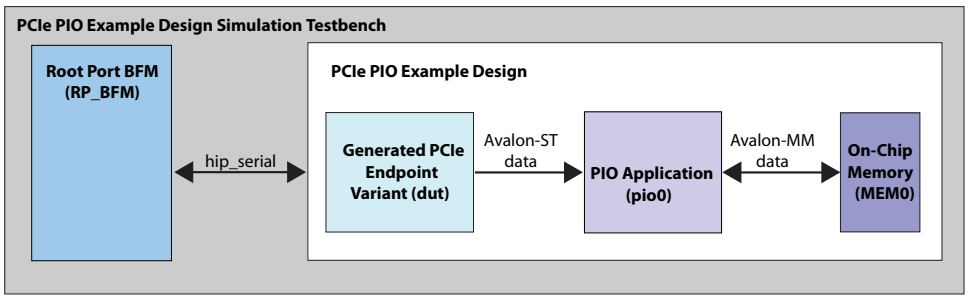
- It is unable to generate or receive Vendor Defined Messages. Some systems generate Vendor Defined Messages. The Hard IP block simply passes these messages on to the Application Layer. Consequently, you should make the decision, based on your application, whether to design the Application Layer to process them.
- It can only handle received read requests that are less than or equal to the currently set **Maximum payload size** option specified under the **Device** tab under the **PCI Express/PCI Capabilities** GUI using the parameter editor. Many systems are capable of handling larger read requests that are then returned in multiple completions.
- It always returns a single completion for every read request. Some systems split completions on every 64-byte address boundary.
- It always returns completions in the same order the read requests were issued. Some systems generate the completions out-of-order.
- It is unable to generate zero-length read requests that some systems generate as flush requests following some write transactions. The Application Layer must be capable of generating the completions to the zero-length read requests.
- It uses fixed credit allocation.
- It does not support parity.
- It does not support multi-function designs.
- It does not support Single Root I/O Virtualization (SR-IOV).
- It incorrectly responds to Type 1 vendor-defined messages with CplD packets.

7.1. Endpoint Testbench

The example design and testbench are dynamically generated based on the configuration that you choose for the P-Tile IP for PCIe. The testbench uses the parameters that you specify in the Parameter Editor in Intel Quartus Prime.

This testbench simulates up to a ×16 PCI Express link using the serial PCI Express interface. The testbench design does allow more than one PCI Express link to be simulated at a time. The following figure presents a high level view of the design example.

Figure 60. Design Example for Endpoint Designs



The top-level of the testbench instantiates the following main modules:

- `altpcietb_bfm_rp_gen3_x8.sv` —This is the Root Port PCIe BFM.

```
//Directory path
<project_dir>/intel_pcie_ptile_ast_0_example_design/
pcie_example_design_tb/ip/pcie_example_design_tb/dut_pcie_tb_ip/
intel_pcie_ptile_tbed_<ver>/sim
```

- `pcie_example_design_dut.ip`: This is the Endpoint design with the parameters that you specify.

```
//Directory path
<project_dir>/intel_pcie_ptile_ast_0_example_design/ip/
pcie_example_design
```

- `pcie_example_design_pio0.ip`: This module is a target and initiator of transactions.

```
//Directory path
<project_dir>/intel_pcie_ptile_ast_0_example_design/ip/
pcie_example_design/
```

In addition, the testbench has routines that perform the following tasks:

- Generates the reference clock for the Endpoint at the required frequency.
- Provides a PCI Express reset at start up.

Note: By default, the `serial_sim_hwtcl` parameter in *<project_dir>*`/intel_pcie_ptile_ast_0_example_design/pcie_example_design_tb/ip/pcie_example_design_tb/dut_pcie_tb_ip/intel_pcie_ptile_tbed_<ver>/sim/intel_pcie_ptile_tbed_hwtcl.v` is set to 1 for serial simulation. **Currently, P-Tile does not allow parallel PIPE simulations.**

7.2. Test Driver Module

The test driver module, `intel_pcie_ptile_tbed_hwtcl.v`, instantiates the top-level BFM,`altpcietb_bfm_top_rp.v`.

The top-level BFM completes the following tasks:

1. Instantiates the driver and monitor.
2. Instantiates the Root Port BFM.
3. Instantiates the serial interface.

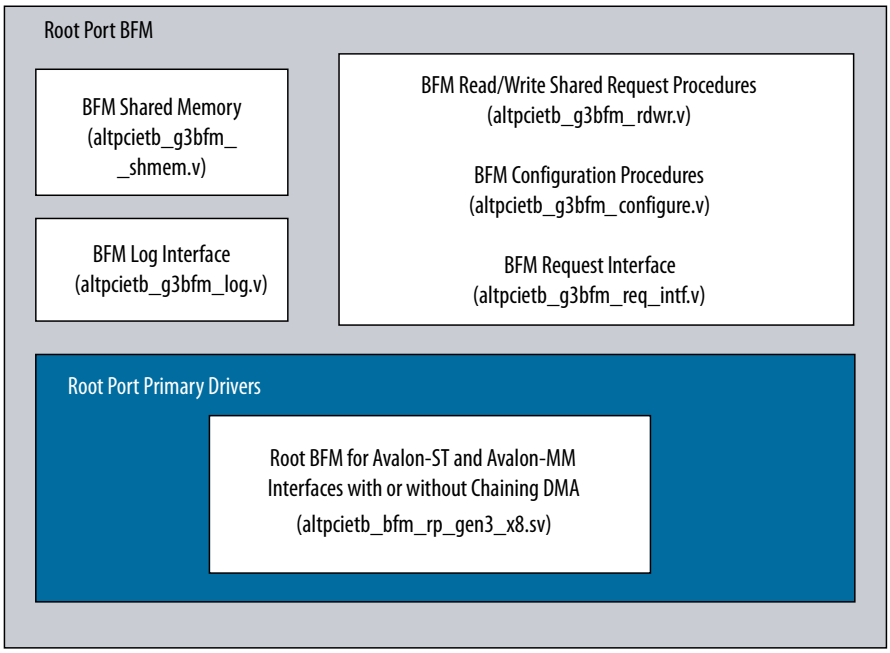
The configuration module, `altpcietb_g3bfm_configure.v`, performs the following tasks:

1. Configures and assigns the BARs.
2. Configures the Root Port and Endpoint.
3. Displays comprehensive Configuration Space, BAR, MSI, MSI-X, and AER settings.

7.3. Root Port BFM

The basic Root Port BFM provides a Verilog HDL task-based interface to request transactions to issue on the PCI Express link. The Root Port BFM also handles requests received from the PCI Express link. The following figure shows the major modules in the Root Port BFM.

Figure 61. Root Port BFM



These modules implement the following functionality:

- BFM Log Interface, `altpcietb_g3bfm_log.v` and `altpcietb_bfm_rp_gen3_x8.sv`: The BFM Log Interface provides routines for writing commonly formatted messages to the simulator standard output and optionally to a log file. It also provides controls that stop simulations on errors.
- BFM Read/Write Request Functions, `altpcietb_bfm_rp_gen3_x8.sv`: These functions provide the basic BFM calls for PCI Express read and write requests.
- BFM Configuration Functions, `altpcietb_g3bfm_configure.v` : These functions provide the BFM calls to request a configuration of the PCI Express link and the Endpoint Configuration Space registers.
- BFM shared memory, `altpcietb_g3bfm_shmem.v`: This module provides the Root Port BFM shared memory. It implements the following functionality:
  - Provides data for TX write operations
  - Provides data for RX read operations
  - Receives data for RX write operations
  - Receives data for received completions

- BFM Request Interface, `altpcietb_g3bfm_req_intf.v`: This interface provides the low-level interface between the `altpcietb_g3bfm_rdwr` and `altpcietb_g3bfm_configure` procedures or functions and the Root Port RTL Model. This interface stores a write-protected data structure containing the sizes and values programmed in the BAR registers of the Endpoint. It also stores other critical data used for internal BFM management.
- `altpcietb_g3bfm_rdwr.v`: This module contains the low-level read and write tasks.
- Avalon-ST Interfaces, `altpcietb_g3bfm_vc_intf_ast_common.v`: These interface modules handle the Root Port interface model. They take requests from the BFM request interface and generate the required PCI Express transactions. They handle completions received from the PCI Express link and notify the BFM request interface when requests are complete. Additionally, they handle any requests received from the PCI Express link, and store or fetch data from the shared memory before generating the required completions.

In the PIO design example, the `apps_type_hwtcl` parameter is set to 3. The tests run under this parameter value are defined in `ebfm_cfg_rp_ep_rootport`, `find_mem_bar` and `downstream_loop`.

The function `ebfm_cfg_rp_ep_rootport` is described in `altpcietb_g3bfm_configure.v`. This function performs the steps necessary to configure the root port and the endpoint on the link. It includes:

- Root port memory allocation
- Root port configuration space (base limit, bus number, etc.)
- Endpoint configuration (BAR, Bus Master enable, maxpayload size, etc.)

The functions `find_mem_bar` and `downstream_loop` in `altpcietb_bfm_rp_gen3_x8.sv` return the BAR implemented and perform the memory Write and Read accesses to the BAR, respectively.

7.3.1. BFM Memory Map

The BFM shared memory is 2 MBs. The BFM shared memory maps to the first 2 MBs of I/O space and also the first 2 MBs of memory space. When the Endpoint application generates an I/O or memory transaction in this range, the BFM reads or writes the shared memory.

7.3.2. Configuration Space Bus and Device Numbering

Enumeration assigns the Root Port interface device number 0 on internal bus number 0. Use the `ebfm_cfg_rp_ep` procedure to assign the Endpoint to any device number on any bus number (greater than 0). The specified bus number is the secondary bus in the Root Port Configuration Space.

7.3.3. Configuration of Root Port and Endpoint

Before you issue transactions to the Endpoint, you must configure the Root Port and Endpoint Configuration Space registers.

The `ebfm_cfg_rp_ep` procedure in `altpcietb_g3bfm_configure.v` executes the following steps to initialize the Configuration Space:

1. Sets the Root Port Configuration Space to enable the Root Port to send transactions on the PCI Express link.
2. Sets the Root Port and Endpoint PCI Express Capability Device Control registers as follows:
   a. Disables `Error Reporting` in both the Root Port and Endpoint. The BFM does not have error handling capability.
   b. Enables `Relaxed Ordering` in both Root Port and Endpoint.
   c. Enables `Extended Tags` for the Endpoint if the Endpoint has that capability.
   d. Disables `Phantom Functions`, `Aux Power PM`, and `No Snoop` in both the Root Port and Endpoint.
   e. Sets the `Max Payload Size` to the value that the Endpoint supports because the Root Port supports the maximum payload size.
   f. Sets the Root Port `Max Read Request Size` to 4 KB because the example Endpoint design supports breaking the read into as many completions as necessary.
   g. Sets the Endpoint `Max Read Request Size` equal to the `Max Payload Size` because the Root Port does not support breaking the read request into multiple completions.
3. Assigns values to all the Endpoint BAR registers. The BAR addresses are assigned by the algorithm outlined below.
   a. I/O BARs are assigned smallest to largest starting just above the ending address of the BFM shared memory in I/O space and continuing as needed throughout a full 32-bit I/O space.
   b. The 32-bit non-prefetchable memory BARs are assigned smallest to largest, starting just above the ending address of the BFM shared memory in memory space and continuing as needed throughout a full 32-bit memory space.
   c. The value of the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure controls the assignment of the 32-bit prefetchable and 64-bit prefetchable memory BARS. The default value of the `addr_map_4GB_limit` is `0`.

      If the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure is set to 0, then the `ebfm_cfg_rp_ep` procedure assigns the 32-bit prefetchable memory BARs largest to smallest, starting at the top of 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.

However, if the `addr_map_4GB_limit` input is set to 1, the address map is limited to 4 GB. The `ebfm_cfg_rp_ep` procedure assigns 32-bit and 64-bit prefetchable memory BARs largest to smallest, starting at the top of the 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.

d. If the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure is set to 0, then the `ebfm_cfg_rp_ep` procedure assigns the 64-bit prefetchable memory BARs smallest to largest starting at the 4 GB address assigning memory ascending above the 4 GB limit throughout the full 64-bit memory space.

   If the `addr_map_4 GB_limit` input to the `ebfm_cfg_rp_ep` procedure is set to 1, the `ebfm_cfg_rp_ep` procedure assigns the 32-bit and the 64-bit prefetchable memory BARs largest to smallest starting at the 4 GB address and assigning memory by descending below the 4 GB address to memory addresses as needed down to the ending address of the last 32-bit non-prefetchable BAR.

   The above algorithm cannot always assign values to all BARs when there are a few very large (1 GB or greater) 32-bit BARs. Although assigning addresses to all BARs may be possible, a more complex algorithm would be required to effectively assign these addresses. However, such a configuration is unlikely to be useful in real systems. If the procedure is unable to assign the BARs, it displays an error message and stops the simulation.

4. Based on the above BAR assignments, the `ebfm_cfg_rp_ep` procedure assigns the Root Port Configuration Space address windows to encompass the valid BAR address ranges.
5. The `ebfm_cfg_rp_ep` procedure enables master transactions, memory address decoding, and I/O address decoding in the Endpoint PCIe control register.

The `ebfm_cfg_rp_ep` procedure also sets up a `bar_table` data structure in BFM shared memory that lists the sizes and assigned addresses of all Endpoint BARs. This area of BFM shared memory is write-protected. Consequently, application logic write accesses to this area cause a fatal simulation error.

BFM procedure calls to generate full PCIe addresses for read and write requests to particular offsets from a BAR use this data structure. This procedure allows the testbench code that accesses the Endpoint application logic to use offsets from a BAR and avoid tracking specific addresses assigned to the BAR. The following table shows how to use those offsets.

Table 95. BAR Table Structure

Offset (Bytes)	Description
+0	PCI Express address in BAR0
+4	PCI Express address in BAR1
+8	PCI Express address in BAR2
+12	PCI Express address in BAR3
+16	PCI Express address in BAR4
+20	PCI Express address in BAR5
+24	PCI Express address in Expansion ROM BAR

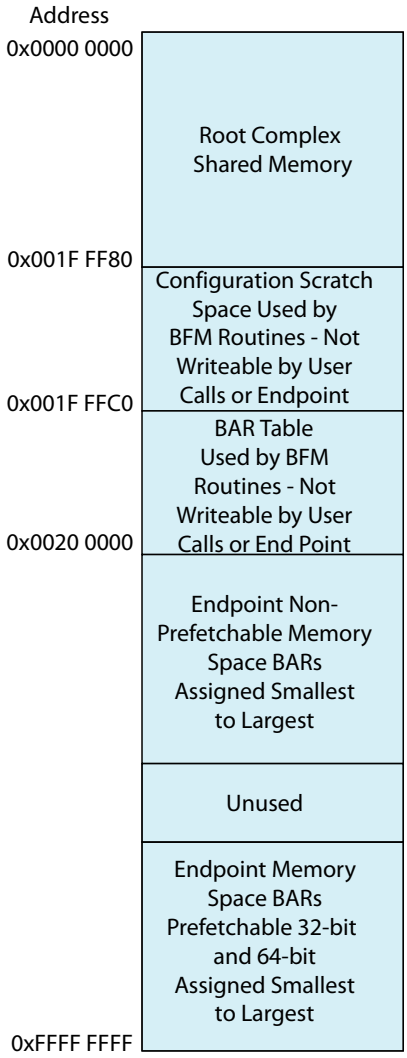
continued...

Offset (Bytes)	Description
+28	Reserved
+32	BAR0 read back value after being written with all 1's (used to compute size)
+36	BAR1 read back value after being written with all 1's
+40	BAR2 read back value after being written with all 1's
+44	BAR3 read back value after being written with all 1's
+48	BAR4 read back value after being written with all 1's
+52	BAR5 read back value after being written with all 1's
+56	Expansion ROM BAR read back value after being written with all 1's
+60	Reserved

The configuration routine does not configure any advanced PCI Express capabilities such as the AER capability.

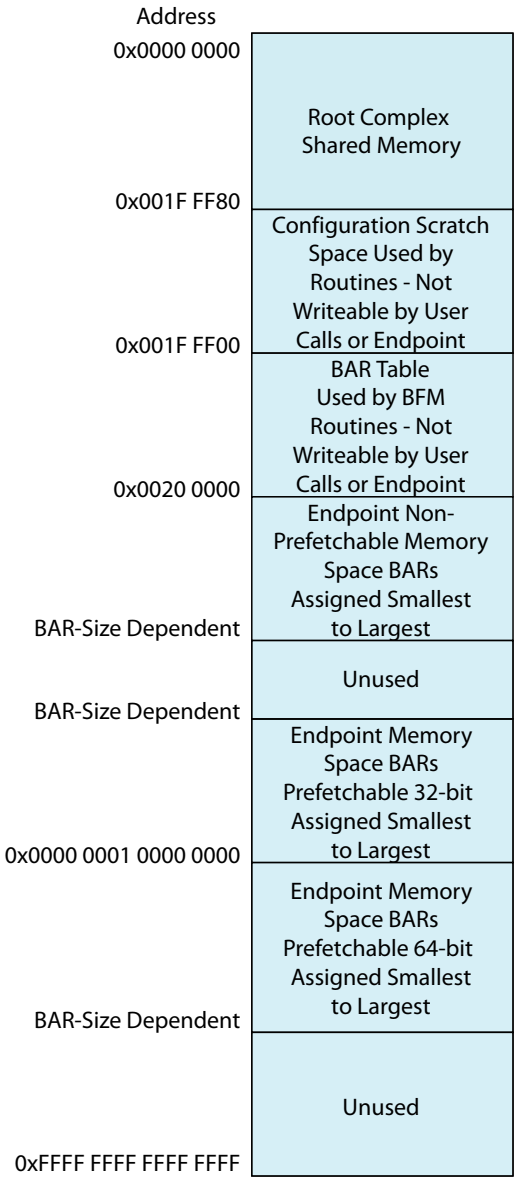
Besides the `ebfm_cfg_rp_ep` procedure in `altpcietb_bfm_rp_gen3_x8.sv`, routines to read and write Endpoint Configuration Space registers directly are available in the Verilog HDL include file. After the `ebfm_cfg_rp_ep` procedure runs, the PCI Express I/O and Memory Spaces have the layout shown in the following three figures. The memory space layout depends on the value of the **addr_map_4GB_limit** input parameter. The following figure shows the resulting memory space map when the **addr_map_4GB_limit** is 1.

Figure 62. Memory Space Layout—4 GB Limit



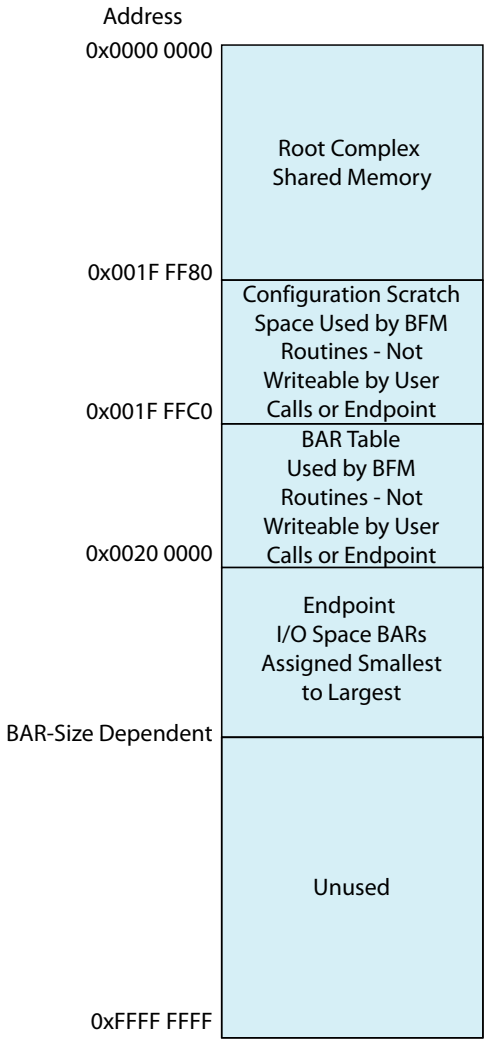
The following figure shows the resulting memory space map when the **addr_map_4GB_limit** is 0.

Figure 63. Memory Space Layout—No Limit



The following figure shows the I/O address space.

Figure 64. I/O Address Space



7.3.4. Issuing Read and Write Transactions to the Application Layer

The Root Port Application Layer issues read and write transactions by calling one of the `ebfm_bar` procedures in `altpcietb_g3bfm_rdwr.v`. The procedures and functions listed below are available in the Verilog HDL include file `altpcietb_g3bfm_rdwr.v`. The complete list of available procedures and functions is as follows:

- `ebfm_barwr`: writes data from BFM shared memory to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barwr_imm`: writes a maximum of four bytes of immediate data (passed in a procedure call) to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barrd_wait`: reads data from an offset of a specific Endpoint BAR and stores it in BFM shared memory. This procedure blocks waiting for the completion data to be returned before returning control to the caller.
- `ebfm_barrd_nowt`: reads data from an offset of a specific Endpoint BAR and stores it in the BFM shared memory. This procedure returns as soon as the request has been passed to the VC interface module for transmission, allowing subsequent reads to be issued in the interim.

These routines take as parameters a BAR number to access the memory space and the BFM shared memory address of the `bar_table` data structure that was set up by the `ebfm_cfg_rp_ep` procedure. (Refer to *Configuration of Root Port and Endpoint*.) Using these parameters simplifies the BFM test driver routines that access an offset from a specific BAR and eliminates calculating the addresses assigned to the specified BAR.

The Root Port BFM does not support accesses to Endpoint I/O space BARs.

7.4. BFM Procedures and Functions

The BFM includes procedures, functions, and tasks to drive Endpoint application testing. It also includes procedures to run the chaining DMA design example.

The BFM read and write procedures read and write data to BFM shared memory, Endpoint BARs, and specified configuration registers. The procedures and functions are available in the Verilog HDL. These procedures and functions support issuing memory and configuration transactions on the PCI Express link.

7.4.1. ebfm_barwr Procedure

The `ebfm_barwr` procedure writes a block of data from BFM shared memory to an offset from the specified Endpoint BAR. The length can be longer than the configured `MAXIMUM_PAYLOAD_SIZE`. The procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last transaction has been accepted by the VC interface module.

Location		
Syntax	`ebfm_barwr(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The `bar_table` structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address of the data to be written.
	`byte_len`	Length, in bytes, of the data written. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic class used for the PCI Express transaction.

7.4.2. ebfm_barwr_imm Procedure

The `ebfm_barwr_imm` procedure writes up to four bytes of data to an offset from the specified Endpoint BAR.

Location		
Syntax	`ebfm_barwr_imm(bar_table, bar_num, pcie_offset, imm_data, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The `bar_table` structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`imm_data`	Data to be written. In Verilog HDL, this argument is `reg [31:0]`.In both languages, the bits written depend on the length as follows: Length Bits Written • 4: 31 down to 0 • 3: 23 down to 0 • 2: 15 down to 0 • 1: 7 down to 0
	`byte_len`	Length of the data to be written in bytes. Maximum length is 4 bytes.
	`tclass`	Traffic class to be used for the PCI Express transaction.

7.4.3. ebfm_barrd_wait Procedure

The `ebfm_barrd_wait` procedure reads a block of data from the offset of the specified Endpoint BAR and stores it in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This procedure waits until all of the completion data is returned and places it in shared memory.

Location		
Syntax	`ebfm_barrd_wait(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address where the read data is stored.
	`byte_len`	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic class used for the PCI Express transaction.

7.4.4. ebfm_barrd_nowt Procedure

The `ebfm_barrd_nowt` procedure reads a block of data from the offset of the specified Endpoint BAR and stores the data in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last read transaction has been accepted by the VC interface module, allowing subsequent reads to be issued immediately.

Location		
Syntax	`ebfm_barrd_nowt(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address where the read data is stored.
	`byte_len`	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic Class to be used for the PCI Express transaction.

7.4.5. ebfm_cfgwr_imm_wait Procedure

The `ebfm_cfgwr_imm_wait` procedure writes up to four bytes of data to the specified configuration register. This procedure waits until the write completion has been returned.

Location		
Syntax	`ebfm_cfgwr_imm_wait(bus_num, dev_num, fnc_num, imm_regb_ad, regb_ln, imm_data, compl_status`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.

continued...

Location		
	regb_ad	Byte-specific address of the register to be written.
	regb_ln	Length, in bytes, of the data written. Maximum length is four bytes. The regb_ln and the regb_ad arguments cannot cross a DWORD boundary.
	imm_data	Data to be written. This argument is reg [31:0]. The bits written depend on the length: • 4: 31 down to 0 • 3: 23 down to 0 • 2: 15 down to 0 • 1: 7 down to 0
	compl_status	This argument is reg [2:0]. This argument is the completion status as specified in the PCI Express specification. The following encodings are defined: • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

7.4.6. ebfm_cfgwr_imm_nowt Procedure

The ebfm_cfgwr_imm_nowt procedure writes up to four bytes of data to the specified configuration register. This procedure returns as soon as the VC interface module accepts the transaction, allowing other writes to be issued in the interim. Use this procedure only when successful completion status is expected.

Location		
Syntax	ebfm_cfgwr_imm_nowt(bus_num, dev_num, fnc_num, imm_regb_adr, regb_len, imm_data)	
Arguments	bus_num	PCI Express bus number of the target device.
	dev_num	PCI Express device number of the target device.
	fnc_num	Function number in the target device to be accessed.
	regb_ad	Byte-specific address of the register to be written.
	regb_ln	Length, in bytes, of the data written. Maximum length is four bytes, The regb_ln the regb_ad arguments cannot cross a DWORD boundary.
	imm_data	Data to be written This argument is reg [31:0]. In both languages, the bits written depend on the length. The following encodes are defined. • 4: [31:0] • 3: [23:0] • 2: [15:0] • 1: [7:0]

7.4.7. ebfm_cfgrd_wait Procedure

The ebfm_cfgrd_wait procedure reads up to four bytes of data from the specified configuration register and stores the data in BFM shared memory. This procedure waits until the read completion has been returned.

Location		
Syntax	`ebfm_cfgrd_wait(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr, compl_status)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data read. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`lcladdr`	BFM shared memory address of where the read data should be placed.
	`compl_status`	Completion status for the configuration transaction. This argument is reg [2:0]. In both languages, this is the completion status as specified in the PCI Express specification. The following encodings are defined. • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

7.4.8. ebfm_cfgrd_nowt Procedure

The `ebfm_cfgrd_nowt` procedure reads up to four bytes of data from the specified configuration register and stores the data in the BFM shared memory. This procedure returns as soon as the VC interface module has accepted the transaction, allowing other reads to be issued in the interim. Use this procedure only when successful completion status is expected and a subsequent read or write with a wait can be used to guarantee the completion of this operation.

Location		
Syntax	`ebfm_cfgrd_nowt(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and `regb_ad` arguments cannot cross a DWORD boundary.
	`lcladdr`	BFM shared memory address where the read data should be placed.

7.4.9. BFM Configuration Procedures

All Verilog HDL arguments are type `integer` and are input-only unless specified otherwise.

7.4.9.1. ebfm_cfg_rp_ep Procedure

The `ebfm_cfg_rp_ep` procedure configures the Root Port and Endpoint Configuration Space registers for operation.

Location		
Syntax	`ebfm_cfg_rp_ep(bar_table, ep_bus_num, ep_dev_num, rp_max_rd_req_size, display_ep_config, addr_map_4GB_limit)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. This routine populates the `bar_table` structure. The `bar_table` structure stores the size of each BAR and the address values assigned to each BAR. The address of the `bar_table` structure is passed to all subsequent read and write procedure calls that access an offset from a particular BAR.
	`ep_bus_num`	PCI Express bus number of the target device. This number can be any value greater than 0. The Root Port uses this as the secondary bus number.
	`ep_dev_num`	PCI Express device number of the target device. This number can be any value. The Endpoint is automatically assigned this value when it receives the first configuration transaction.
	`rp_max_rd_req_size`	Maximum read request size in bytes for reads issued by the Root Port. This parameter must be set to the maximum value supported by the Endpoint Application Layer. If the Application Layer only supports reads of the `MAXIMUM_PAYLOAD_SIZE`, then this can be set to 0 and the read request size is set to the maximum payload size. Valid values for this argument are 0, 128, 256, 512, 1,024, 2,048 and 4,096.
	`display_ep_config`	When set to 1 many of the Endpoint Configuration Space registers are displayed after they have been initialized, causing some additional reads of registers that are not normally accessed during the configuration process such as the Device ID and Vendor ID.
	`addr_map_4GB_limit`	When set to 1 the address map of the simulation system is limited to 4 GB. Any 64-bit BARs are assigned below the 4 GB limit.

7.4.9.2. ebfm_cfg_decode_bar Procedure

The `ebfm_cfg_decode_bar` procedure analyzes the information in the BAR table for the specified BAR and returns details about the BAR attributes.

Location		
Syntax	`ebfm_cfg_decode_bar(bar_table, bar_num, log2_size, is_mem, is_pref, is_64b)`	
Arguments	`bar_table`	Address of the Endpoint bar_table structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`log2_size`	This argument is set by the procedure to the log base 2 of the size of the BAR. If the BAR is not enabled, this argument is set to 0.
	`is_mem`	The procedure sets this argument to indicate if the BAR is a memory space BAR (1) or I/O Space BAR (0).
	`is_pref`	The procedure sets this argument to indicate if the BAR is a prefetchable BAR (1) or non-prefetchable BAR (0).
	`is_64b`	The procedure sets this argument to indicate if the BAR is a 64-bit BAR (1) or 32-bit BAR (0). This is set to 1 only for the lower numbered BAR of the pair.

7.4.10. BFM Shared Memory Access Procedures

These procedures and functions support accessing the BFM shared memory.

7.4.10.1. Shared Memory Constants

Table 96. Constants: Verilog HDL Type INTEGER

Constant	Description
SHMEM_FILL_ZEROS	Specifies a data pattern of all zeros
SHMEM_FILL_BYTE_INC	Specifies a data pattern of incrementing 8-bit bytes (0x00, 0x01, 0x02, etc.)
SHMEM_FILL_WORD_INC	Specifies a data pattern of incrementing 16-bit words (0x0000, 0x0001, 0x0002, etc.)
SHMEM_FILL_DWORD_INC	Specifies a data pattern of incrementing 32-bit DWORDs (0x00000000, 0x00000001, 0x00000002, etc.)
SHMEM_FILL_QWORD_INC	Specifies a data pattern of incrementing 64-bit qwords (0x0000000000000000, 0x0000000000000001, 0x0000000000000002, etc.)
SHMEM_FILL_ONE	Specifies a data pattern of all ones

7.4.10.2. shmem_write Task

The `shmem_write` procedure writes data to the BFM shared memory.

Location		
Syntax	`shmem_write(addr, data, leng)`	
Arguments	`addr`	BFM shared memory starting address for writing data
	`data`	Data to write to BFM shared memory. This parameter is implemented as a 64-bit vector. `leng` is 1–8 bytes. Bits 7 down to 0 are written to the location specified by `addr`; bits 15 down to 8 are written to the `addr+1` location, etc.
	`length`	Length, in bytes, of data written

7.4.10.3. shmem_read Function

The `shmem_read` function reads data to the BFM shared memory.

Location		
Syntax	`data:= shmem_read(addr, leng)`	
Arguments	`addr`	BFM shared memory starting address for reading data
	`leng`	Length, in bytes, of data read
Return	`data`	Data read from BFM shared memory. This parameter is implemented as a 64-bit vector. `leng` is 1- 8 bytes. If `leng` is less than 8 bytes, only the corresponding least significant bits of the returned data are valid. Bits 7 down to 0 are read from the location specified by `addr`; bits 15 down to 8 are read from the addr+1 location, etc.

7.4.10.4. shmem_display Verilog HDL Function

The `shmem_display` Verilog HDL function displays a block of data from the BFM shared memory.

Location		
Syntax	Verilog HDL: `dummy_return:=shmem_display(addr, leng, word_size, flag_addr, msg_type);`	
Arguments	`addr`	BFM shared memory starting address for displaying data.
	`leng`	Length, in bytes, of data to display.
	`word_size`	Size of the words to display. Groups individual bytes into words. Valid values are 1, 2, 4, and 8.
	`flag_addr`	Adds a <== flag to the end of the display line containing this address. Useful for marking specific data. Set to a value greater than 2**21 (size of BFM shared memory) to suppress the flag.
	`msg_type`	Specifies the message type to be displayed at the beginning of each line. See "BFM Log and Message Procedures" on page 18–37 for more information about message types. Set to one of the constants defined in Table 18–36 on page 18–41.

7.4.10.5. shmem_fill Procedure

The `shmem_fill` procedure fills a block of BFM shared memory with a specified data pattern.

Location		
Syntax	`shmem_fill(addr, mode, leng, init)`	
Arguments	`addr`	BFM shared memory starting address for filling data.
	`mode`	Data pattern used for filling the data. Should be one of the constants defined in section *Shared Memory Constants*.
	`leng`	Length, in bytes, of data to fill. If the length is not a multiple of the incrementing data pattern width, then the last data pattern is truncated to fit.
	`init`	Initial data value used for incrementing data pattern modes. This argument is `reg [63:0]`. The necessary least significant bits are used for the data patterns that are smaller than 64 bits.

7.4.10.6. shmem_chk_ok Function

The `shmem_chk_ok` function checks a block of BFM shared memory against a specified data pattern.

Location		
Syntax	`result:= shmem_chk_ok(addr, mode, leng, init, display_error)`	
Arguments	`addr`	BFM shared memory starting address for checking data.
	`mode`	Data pattern used for checking the data. Should be one of the constants defined in section "Shared Memory Constants" on page 18–35.
	`leng`	Length, in bytes, of data to check.

continued...

Location		
	`init`	This argument is `reg [63:0]`.The necessary least significant bits are used for the data patterns that are smaller than 64-bits.
	`display_error`	When set to 1, this argument displays the data failing comparison on the simulator standard output.
Return	`Result`	Result is 1-bit. • 1'b1 — Data patterns compared successfully • 1'b0 — Data patterns did not compare successfully

7.4.11. BFM Log and Message Procedures

These procedures provide support for displaying messages in a common format, suppressing informational messages, and stopping simulation on specific message types.

The following constants define the type of message and their values determine whether a message is displayed or simulation is stopped after a specific message. Each displayed message has a specific prefix, based on the message type in the following table.

You can suppress the display of certain message types. The default values determining whether a message type is displayed are defined in the following table. To change the default message display, modify the display default value with a procedure call to `ebfm_log_set_suppressed_msg_mask`.

Certain message types also stop simulation after the message is displayed. The following table shows the default value determining whether a message type stops simulation. You can specify whether simulation stops for particular messages with the procedure `ebfm_log_set_stop_on_msg_mask`.

All of these log message constants type `integer`.

Table 97. Log Messages

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
`EBFM_MSG_DEBUG`	Specifies debug messages.	0	No	No	`DEBUG:`
`EBFM_MSG_INFO`	Specifies informational messages, such as configuration register values, starting and ending of tests.	1	Yes	No	`INFO:`
`EBFM_MSG_WARNING`	Specifies warning messages, such as tests being skipped due to the specific configuration.	2	Yes	No	`WARNING:`
`EBFM_MSG_ERROR_INFO`	Specifies additional information for an error. Use this message to display preliminary information before an error message that stops simulation.	3	Yes	No	`ERROR:`

continued...

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_ERROR_CONTINUE	Specifies a recoverable error that allows simulation to continue. Use this error for data comparison failures.	4	Yes	No	ERROR:
EBFM_MSG_ERROR_FATAL	Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible.	N/A	Yes Cannot suppress	Yes Cannot suppress	FATAL:
EBFM_MSG_ERROR_FATAL_TB_ERR	Used for BFM test driver or Root Port BFM fatal errors. Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible. Use this error message for errors that occur due to a problem in the BFM test driver module or the Root Port BFM, that are not caused by the Endpoint Application Layer being tested.	N/A	Y Cannot suppress	Y Cannot suppress	FATAL:

7.4.11.1. ebfm_display Verilog HDL Function

The `ebfm_display` procedure or function displays a message of the specified type to the simulation standard output and also the log file if `ebfm_log_open` is called.

A message can be suppressed, simulation can be stopped or both based on the default settings of the message type and the value of the bit mask when each of the procedures listed below is called. You can call one or both of these procedures based on what messages you want displayed and whether or not you want simulation to stop for specific messages.

- When `ebfm_log_set_suppressed_msg_mask` is called, the display of the message might be suppressed based on the value of the bit mask.
- When `ebfm_log_set_stop_on_msg_mask` is called, the simulation can be stopped after the message is displayed, based on the value of the bit mask.

Location		
Syntax	`Verilog HDL: dummy_return:=ebfm_display(msg_type, message);`	
Argument	`msg_type`	Message type for the message. Should be one of the constants defined in Table 96 on page 145.
	`message`	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.
Return	`always 0`	Applies only to the Verilog HDL routine.

7.4.11.2. ebfm_log_stop_sim Verilog HDL Function

The `ebfm_log_stop_sim` procedure stops the simulation.

Location		
Syntax	Verilog HDL: `return:=ebfm_log_stop_sim(success);`	
Argument	`success`	When set to a 1, this process stops the simulation with a message indicating successful completion. The message is prefixed with SUCCESS. Otherwise, this process stops the simulation with a message indicating unsuccessful completion. The message is prefixed with FAILURE.
Return	Always 0	This value applies only to the Verilog HDL function.

7.4.11.3. ebfm_log_set_suppressed_msg_mask Task

The `ebfm_log_set_suppressed_msg_mask` procedure controls which message types are suppressed.

Location		
Syntax	`ebfm_log_set_suppressed_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE: EBFM_MSG_DEBUG]`. A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to be suppressed.

7.4.11.4. ebfm_log_set_stop_on_msg_mask Verilog HDL Task

The `ebfm_log_set_stop_on_msg_mask` procedure controls which message types stop simulation. This procedure alters the default behavior of the simulation when errors occur as described in the *BFM Log and Message Procedures*.

Location		
Syntax	`ebfm_log_set_stop_on_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE:EBFM_MSG_DEBUG]`. A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to stop the simulation after the message is displayed.

7.4.11.5. ebfm_log_open Verilog HDL Function

The `ebfm_log_open` procedure opens a log file of the specified name. All displayed messages are called by `ebfm_display` and are written to this log file as simulator standard output.

Location		
Syntax	`ebfm_log_open (fn)`	
Argument	`fn`	This argument is type `string` and provides the file name of log file to be opened.

7.4.11.6. ebfm_log_close Verilog HDL Function

The `ebfm_log_close` procedure closes the log file opened by a previous call to `ebfm_log_open`.

Location	
Syntax	`ebfm_log_close`
Argument	NONE

7.4.12. Verilog HDL Formatting Functions

7.4.12.1. himage1

This function creates a one-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= himage(vec)`	
Argument	`vec`	Input data type `reg` with a `range` of 3:0.
Return range	`string`	Returns a 1-digit hexadecimal representation of the input argument. Return data is type `reg` with a `range` of 8:1

7.4.12.2. himage2

This function creates a two-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 7:0.
Return range	`string`	Returns a 2-digit hexadecimal presentation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 16:1

7.4.12.3. himage4

This function creates a four-digit hexadecimal string representation of the input argument can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 15:0.
Return range	Returns a four-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 32:1.	

7.4.12.4. himage8

This function creates an 8-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 31:0.
Return range	`string`	Returns an 8-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 64:1.

7.4.12.5. himage16

This function creates a 16-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 63:0.
Return range	`string`	Returns a 16-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 128:1.

7.4.12.6. dimage1

This function creates a one-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 1-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 8:1. Returns the letter *U* if the value cannot be represented.

7.4.12.7. dimage2

This function creates a two-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 2-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 16:1. Returns the letter *U* if the value cannot be represented.

7.4.12.8. dimage3

This function creates a three-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	string	Returns a 3-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 24:1. Returns the letter *U* if the value cannot be represented.

7.4.12.9. dimage4

This function creates a four-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 4-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 32:1. Returns the letter *U* if the value cannot be represented.

7.4.12.10. dimage5

This function creates a five-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 5-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 40:1. Returns the letter *U* if the value cannot be represented.

7.4.12.11. dimage6

This function creates a six-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 6-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 48:1. Returns the letter *U* if the value cannot be represented.

7.4.12.12. dimage7

This function creates a seven-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location		
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of `31:0`.
Return range	`string`	Returns a 7-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of `56:1`. Returns the letter <*U*> if the value cannot be represented.

8. Troubleshooting/Debugging

As you bring up your PCI Express system, you may face issues related to FPGA configuration, link training, BIOS enumeration, data transfer, and so on. This chapter suggests some strategies to resolve the common issues that occur during bring-up.

You can additionally use the P-Tile Debug Toolkit to identify the issues.

8.1. Hardware

Typically, PCI Express link-up involves the following steps:

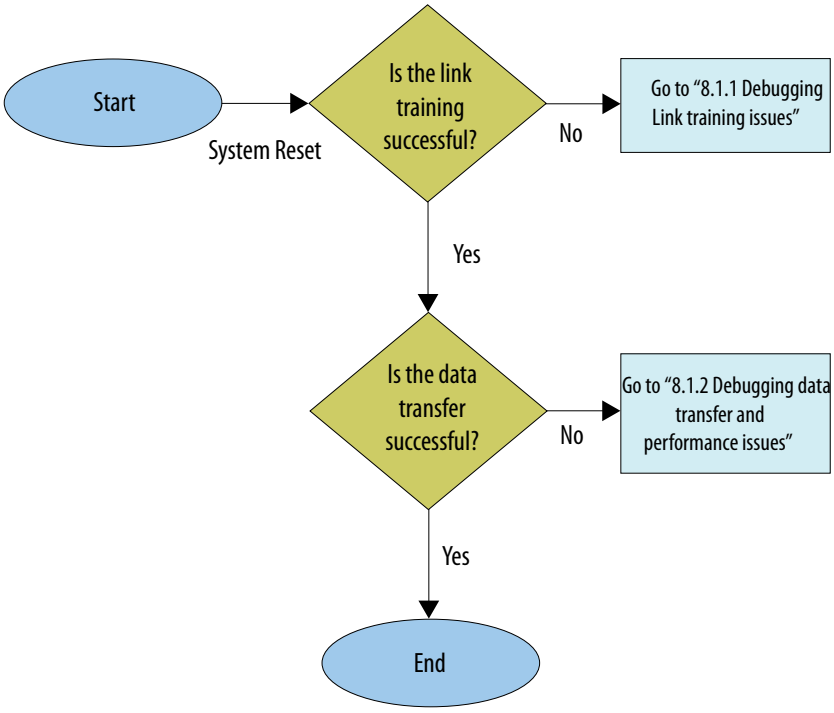
1. Link training
2. BIOS enumeration and data transfer

The following sections describe the flow to debug link issues during the hardware bring-up. Intel recommends a systematic approach to diagnosing issues as illustrated in the following figure.

Additionally, you can use the P-Tile Debug Toolkit for debugging the PCIe links when using the P-Tile Avalon-ST IP for PCI Express. The P-Tile Debug Toolkit includes the following features:

- Protocol and link status information.
- Basic and advanced debugging capabilities including PMA register access and Eye viewing capability.

Figure 65. PCI Express Debug Flow Chart



8.1.1. Debugging Link Training Issues

The Physical Layer automatically performs link training and initialization without software intervention. This is a well-defined process to configure and initialize the device's Physical Layer and link so that PCIe packets can be transmitted.

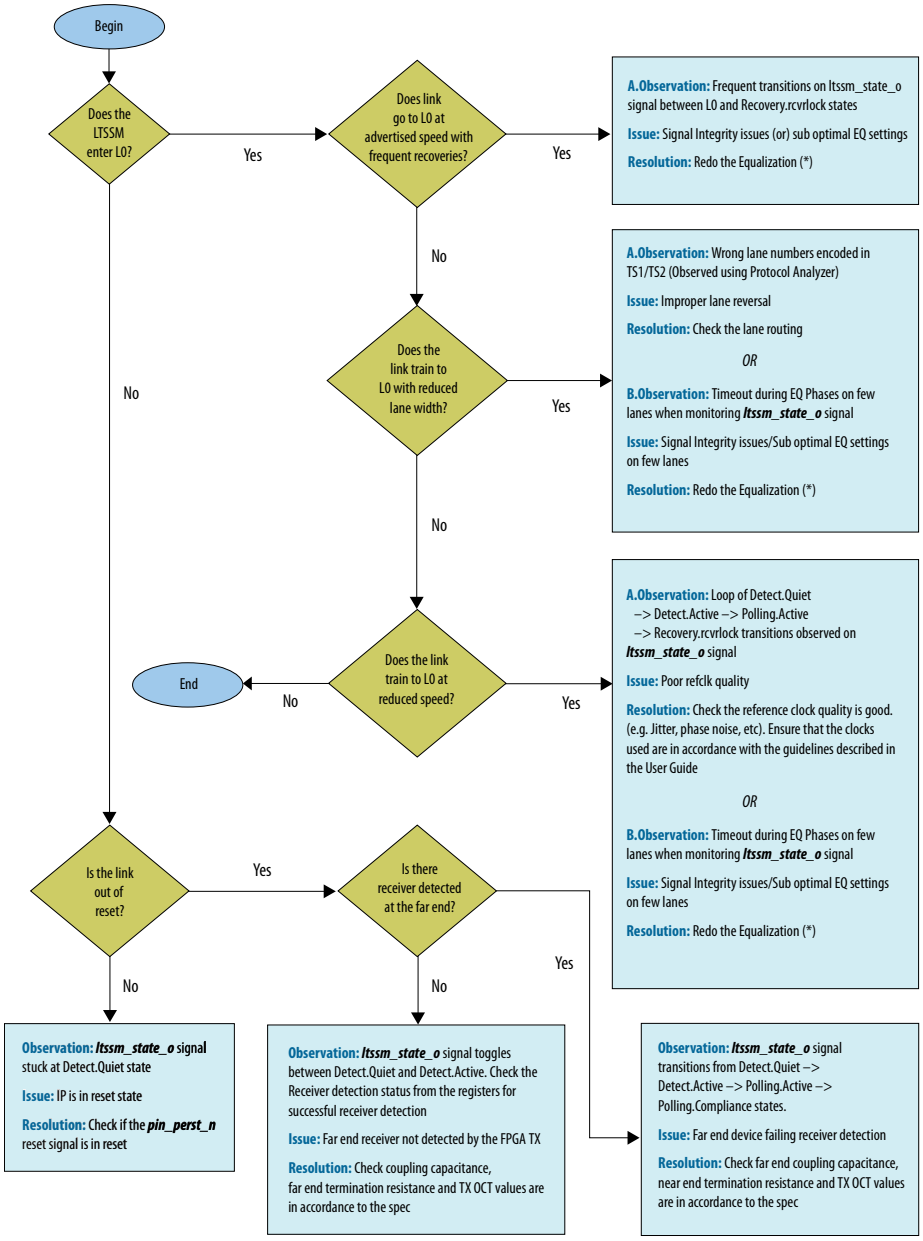
Some examples of link training issues include:

- Link fails to negotiate to expected link speed.
- Link fails to negotiate to the expected link width.
- LTSSM fails to reach/stay stable at L0.

Flow Chart for Debugging Link Training Issues

Use the flow chart below to identify the potential cause of the issue seen during link training when using the P-Tile Avalon-ST IP for PCI Express.

Figure 66. Link Training Debugging Flow



Note: (*) Redo the equalization using the **Link Equalization Request 8.0 GT/s** bit of the Link Status 2 register for 8.0 GT/s or **Link Equalization Request 16.0 GT/s** bit of the 16.0 GT/s Status Register.

Use the following debug tools for debugging link training issues observed on the PCI Express link when using the P-tile Avalon-ST IP for PCI Express.

8.1.1.1. Generic Tools and Utilities

You can use utilities like lspci, setpci to obtain general information of the device like link speed, link width etc.

Example: To read the negotiated link speed for the P-Tile device in a system, you can use the following commands:

sudo lspci –s $bdf -vvv

-s refers to "slot" and is used with the bus/device/function number (bdf) information. Use this command if you know the bdf of the device in the system topology.

sudo lspci –d <1172>:$did -vvv

-d refers to device and is used with the device ID (vid:did). Use this command to search using the device ID.

Figure 67. `lspci` Output

```
lspci -vvv -s 0a:00.0
0a:00.0 Non-VGA unclassified device: Altera Corporation Device 0000 (rev 01)
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx-
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0, Cache Line Size: 64 bytes
        Interrupt: pin A routed to IRQ 72
        Region 0: Memory at f0000000 (64-bit, prefetchable) [size=64K]
        Capabilities: [40] Power Management version 3
                Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
                Status: D0 NoSoftRst- PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [70] Express (v2) Endpoint, MSI 00
                DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s <64ns, L1 <1us
                        ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset- SlotPowerLimit 0.000W
                DevCtl: Report errors: Correctable+ Non-Fatal+ Fatal+ Unsupported-
                        RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+
                        MaxPayload 512 bytes, MaxReadReq 512 bytes
                DevSta: CorrErr+ UncorrErr- FatalErr- UnsuppReq+ AuxPwr- TransPend-
                LnkCap: Port #1, Speed 16GT/s, Width x16, ASPM not supported, Exit Latency L0s <64ns, L1 <1us
                        ClockPM- Surprise- LLActRep- BwNot- ASPMOptComp+
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk+
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 16GT/s, Width x16, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
                DevCap2: Completion Timeout: Range ABCD, TimeoutDis+, LTR+, OBFF Not Supported
                DevCtl2: Completion Timeout: 50us to 50ms, TimeoutDis-, LTR-, OBFF Disabled
                LnkCtl2: Target Link Speed: 16GT/s, EnterCompliance- SpeedDis-
                         Transmit Margin: Normal Operating Range, EnterModifiedCompliance- ComplianceSOS-
                         Compliance De-emphasis: -6dB
                LnkSta2: Current De-emphasis Level: -3.5dB, EqualizationComplete+, EqualizationPhase1+
                         EqualizationPhase2+, EqualizationPhase3+, LinkEqualizationRequest-
        Capabilities: [100 v2] Advanced Error Reporting
                UESta:  DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTLP- ECRC- UnsupReq- ACSViol-
                UEMsk:  DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTLP- ECRC- UnsupReq- ACSViol-
                UESvrt: DLP+ SDES+ TLP- FCP+ CmpltTO- CmpltAbrt- UnxCmplt- RxOF+ MalfTLP+ ECRC- UnsupReq- ACSViol-
                CESta:  RxErr- BadTLP- BadDLLP- Rollover- Timeout- NonFatalErr+
                CEMsk:  RxErr- BadTLP- BadDLLP- Rollover- Timeout- NonFatalErr+
                AERCap: First Error Pointer: 00, GenCap+ CGenEn- ChkCap+ ChkEn-
        Capabilities: [148 v1] Virtual Channel
                Caps:   LPEVC=0 RefClk=100ns PATEntryBits=1
                Arb:    Fixed- WRR32- WRR64- WRR128-
                Ctrl:   ArbSelect=Fixed
                Status: InProgress-
                VC0:    Caps:   PATOffset=00 MaxTimeSlots=1 RejSnoopTrans-
                        Arb:    Fixed- WRR32- WRR64- WRR128- TWRR128- WRR256-
```

The **LnkCap** under **Capabilities** indicates the advertised link speed and width capabilities of the device. The **LnkSta** under **Capabilities** indicates the negotiated link speed and width of the device.

8.1.1.2. SignalTapII Logic Analyzer

Using the SignalTapII Logic Analyzer, you can monitor the following top-level signals from the P-Tile Avalon-ST IP for PCI Express to confirm the failure symptom for any port type (Root Port, Endpoint or TLP Bypass) and configuration (Gen4/Gen3).

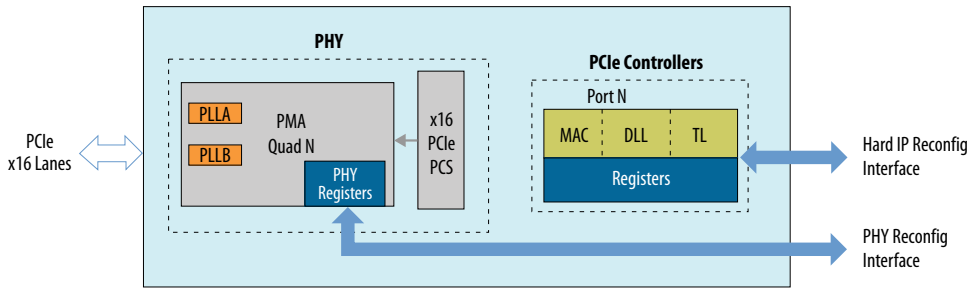
Table 98. Top-Level Signals to be Monitored for Debugging

Signals	Description	Expected Value for Successful Link-up
pin_perst_n	Active-low asynchronous input signal to the PCIe Hard IP. Implements the PERST# function defined by the PCIe specification.	1'b1
p0_reset_status_n	Active-low output signal from the PCIe Hard IP, synchronous to coreclkout_hip. Held low until pin_perst_n is deasserted and the PCIe Hard IP comes out of reset, synchronous to coreclkout_hip. When port bifurcation is used, there is one such signal for each Avalon-ST interface.	1'b1
ninit_done	Active-low output signal from the Reset Release Intel FPGA IP. High indicates that the FPGA device is not yet fully configured, and low indicates the device has been configured and is in normal operating mode. For more details on the Reset Release Intel FPGA IP, refer to https://www.intel.com/content/www/us/en/programmable/documentation/prh1555609801770.html	1'b0
link_up_o	Active-high output signal from the PCIe Hard IP, synchronous to coreclkout_hip. Indicates that the Physical Layer link is up.	1'b1
dl_up_o	Active-high output signal from the PCIe Hard IP, synchronous to coreclkout_hip. Indicates that the Data Link Layer is active.	1'b1
ltssm_state_o[5:0]	Indicates the LTSSM state, synchronous to coreclkout_hip.	6'h11 (L0)
Negotiated link speed using the Transaction Layer Configuration Output interface (tl_cfg): tl_cfg_add_o[4:0] tl_cfg_ctl_o[15:12] tl_cfg_func_o[2:0]	Use the Transaction Layer Configuration Output interface (tl_cfg) to monitor the auto-negotiated link speed.	tl_cfg_add_o[4:0] = 5'h05 tl_cfg_ctl_o[15:12] = • 4'h01 (Gen1) • 4'h02 (Gen2) • 4'h04 (Gen3) • 4'h08 (Gen4) tl_cfg_func_o[2:0] (NA for x4) = • 3'b000: PF0 • 3'b001: PF1, etc.
Negotiated link width using the Transaction Layer Configuration Output interface (tl_cfg): tl_cfg_add_o[4:0] tl_cfg_ctl_o[15:12] tl_cfg_func_o[2:0]	Use the Transaction Layer Configuration Output interface (tl_cfg) to monitor the auto-negotiated link width.	tl_cfg_add_o[4:0] = 5'h1C tl_cfg_ctl_o[5:0] = • 6'h01 (x1) • 6'h02 (x2) • 6'h04 (x4) • 6'h08 (x8) • 6'h10 (x16)

8.1.1.3. Additional Debug Tools

Use the Hard IP reconfiguration interface and PHY reconfiguration interface on the P-Tile Avalon-ST IP for PCI Express to access additional registers (for example, receiver detection, lane reversal etc.).

Figure 68. Register Access for Debug



Using the Hard IP Reconfiguration Interface

Refer to the section *Hard IP Reconfiguration Interface* for details on this interface and the associated address map.

The following table lists the address offsets and bit settings for the PHY status registers. Use the Hard IP Reconfiguration Interface to access these read-only registers.

Table 99. Hard IP Reconfiguration Interface Register Map for PHY Status

Offset	Bit Position	Register
0x0003E9	[0]	RX polarity
	[1]	RX detection
	[2]	RX Valid
	[3]	RX Electrical Idle
	[4]	TX Electrical Idle
0x0003EC	[7]	Framing error
0x0003ED	[7]	Lane reversal

Follow the steps below to access registers in Table 99 on page 159 using the Hard IP reconfiguration interface:

1. Enable the Hard IP reconfiguration interface (User Avalon-MM interface) using the IP Parameter Editor.
2. Set the lane number for which you want to read the status by performing a read-modify-write to the address `hip_reconfig_addr_i[20:0]` with write data of lane number on `hip_reconfig_writedata_i[7:0]` using the Hard IP reconfiguration interface signals.

- `hip_reconfig_write_i` = 1'b1
- `hip_reconfig_addr_i[20:0]` = 0x0003E8
- `hip_reconfig_writedata_i[3:0]` = <Lane number>, where Lane number = 4'h0 for lane 0, 4'h1 for lane 1, 4'h2 for lane 2, …

3. Read the status of the register you want by performing a read operation from the address `hip_reconfig_addr_i[20:0]` using the Hard IP reconfiguration interface signals.
   - `hip_reconfig_read_i` = 1'b1
   - `hip_reconfig_addr_i[20:0]` = <offset>

     Offset = Refer to Table 99 on page 159 for the offset mapping.
   - `hip_reconfig_readdata_o[7:0]` = Refer to Table 99 on page 159 for the bit position mapping.

Example 1: To read the RX detection status of Lane0 using the registers

1. Enable the Hard IP reconfiguration interface using the IP Parameter Editor.
2. Perform read-modify-write to address 0x0003E8 to set the lane number to 0 using the Hard IP reconfiguration interface signals.
   - `hip_reconfig_write_i` = 1'b1
   - `hip_reconfig_addr_i[20:0]` = 0x0003E8
   - `hip_reconfig_writedata_i[3:0]` = 4'h0
3. Read the status of the RX detection register by performing a read operation from the address 0x0003E9[1] using the Hard IP reconfiguration interface signals.
   - `hip_reconfig_read_i` = 1'b1
   - `hip_reconfig_addr_i[20:0]` = 0x0003E9
   - `hip_reconfig_readdata_o[1]` = 1'b1 (Far end receiver detected)

Using the PHY Reconfiguration Interface

Refer to the section *PHY Reconfiguration Interface* for details on how to use this interface.

Follow the steps below to access registers in Table 100 on page 161 using the PHY reconfiguration interface.

1. Enable the PHY reconfiguration interface using the IP Parameter Editor.
2. Set the Quad and address offset from which you want to read the status by performing a read operation from the address `xcvr_reconfig_addr_i[25:0]` using the PHY reconfiguration interface signals.
   - `xcvr_reconfig_read_i` = 1'b1
   - `xcvr_reconfig_addr_i[25:0]` = {5-bit Quad mapping, 21-bit address offset}. Refer to Table 100 on page 161 for the address offset and bit mapping.
   - `xcvr_reconfig_readdata_o[7:0]` = Refer to Table 100 on page 161 for the address offset and bit mapping.

Table 100. PHY Reconfiguration Interface Register Map for PHY Status

PHY Offset	Bit Position	Register
0x000006	[7]	PLLA state output status signal. 1'b1 indicates that PLLA is locked.
0x00000a	[7]	PLLB state output status signal. 1'b1 indicates that PLLB is locked.

Example 2: To read the PLLA status using the registers

1. Enable the PHY reconfiguration interface using the IP Parameter Editor.
2. Perform read from address 0x000006 to read the PLLA status output of Quad0 using the PHY reconfiguration interface signals.
   - `xcvr_reconfig_read_i` = 1'b1
   - `xcvr_reconfig_addr_i[25:0]` = 0x000006
   - `xcvr_reconfig_readdata_o[7:0]` = 8'h80
   - `xcvr_reconfig_readdata_i[1]` = 1'b1 (PLLA state output high indicating PLL lock)

8.1.2. Debugging Data Transfer and Performance Issues

There are many possible reasons causing the PCIe link to stop transmitting data. The PCI Express base specification defines three types of errors, outlined in the table below:

Table 101. Error Types Defined by the PCI Express Base Specification

Type	Responsible Agent	Description
Correctable	Hardware	While correctable errors may affect system performance, data integrity is maintained.
Uncorrectable, non-fatal	Device software	Uncorrectable, non-fatal errors are defined as errors in which data is lost, but system integrity is maintained. For example, the fabric may lose a particular TLP, but it still works without problems.
Uncorrectable, fatal	System software	Errors generated by a loss of data and system failure are considered uncorrectable and fatal. Software must determine how to handle such errors: whether to reset the link or implement other means to minimize the problem.

Table 102. Correctable Error Status Register (AER)

Observation	Issue	Resolution
Receiver error bit set	Physical layer error which may be due to a PCS error when a lane is in L0, or a Control symbol being received in the wrong lane, or signal Integrity issues where the link may transition from L0 to the Recovery state.	Use the configuration output interface, or the Hard IP reconfiguration interface and the flow chart in Figure 66 on page 156 to obtain more information about the error.

continued...

Observation	Issue	Resolution
Bad DLLP bit set	Data link layer error which may occur when a CRC verification fails.	Use the configuration output interface or the Hard IP reconfiguration interface to obtain more information about the error.
Bad TLP bit set	Data link layer error which may occur when an LCRC verification fails or when a sequence number error occurs.	Use the configuration output interface or the Hard IP reconfiguration interface to obtain more information about the error.
Replay_num_rollover bit set	Data link layer error which may be due to TLPs sent without success (no ACK) four times in a row.	Use the configuration output interface or the Hard IP reconfiguration interface to obtain more information about the error.
replay timer timeout status bit set	Data link layer error which may occur when no ACK or NAK was received within the timeout period for the TLPs transmitted.	Use the configuration output interface or the Hard IP reconfiguration interface to obtain more information about the error.
Advisory non-fatal	Transaction layer error which may be due to higher priority uncorrectable error detected.	
Corrected internal error bits set	Transaction layer error which may be due to an ECC error in the internal Hard IP RAM.	Use the error interface, configuration output interface, or the Hard IP reconfiguration interface and DBI registers to obtain more information about the error.

Table 103. Uncorrectable Error Status Register (AER)

Observation	Issue	Resolution
Data link protocol error	Data link layer error which may be due to transmitter receiving an ACK/NAK whose Seq ID does not correspond to an unacknowledged TLP or ACK sequence number.	Use the configuration output interface, Hard IP reconfiguration interface to obtain more information about the error.
Surprise down error	Data link layer error which may be due to `link_up_o` getting deasserted during L0, indicating the physical layer link is going down unexpectedly.	Use the error interface, configuration output interface, Hard IP reconfiguration interface and DBI registers to obtain more information about the error.
Flow control protocol error	Transaction layer error which can be due to the receiver reporting more than the allowed credit limit. This error occurs when a component does not receive updated flow control credits with the 200 µs limit.	Use the TX/RX flow control interface, configuration output interface, Hard IP reconfiguration interface to obtain more information about the error.
Poisoned TLP received	Transaction layer error which can be due to a received TLP with the EP bit set.	Use the error interface, configuration output interface, configuration intercept interface, Hard IP reconfiguration interface to obtain more information on the error and determine the appropriate action.
Completion timeout	Transaction layer error which can be due to a completion not received within the required amount of time after a non-posted request was sent.	Use the error interface, completion timeout interface, configuration output interface, Hard IP reconfiguration interface to obtain more information on the error.

continued...

Observation	Issue	Resolution
Completer abort	Transaction layer error which can be due to a completer being unable to fulfill a request due to a problem with the requester or a failure of the completer.	Use the configuration output interface, error interface, Hard IP reconfiguration interface to obtain more information on the error.
Unexpected completion	Transaction layer error which can be due to a requester receiving a completion that doesn't match any request awaiting a completion. The TLP is deleted by the Hard IP and not presented to the Application Layer.	Use the configuration output interface, error interface, Hard IP reconfiguration interface to obtain more information on the error.
Receiver overflow	Transaction layer error which can be due to a receiver receiving more TLPs than the available receive buffer space. The TLP is deleted by the Hard IP and not presented to the Application Layer.	Use the TX/RX flow control interface, error interface, configuration output interface, Hard IP reconfiguration interface to obtain more information on the error.
Malformed TLP	Transaction layer error which can be due to errors in the received TLP header. The TLP is deleted by the Hard IP and not presented to the Application Layer.	Use the error interface, configuration output interface, Hard IP reconfiguration interface to obtain more information on the error.
ECRC error	Transaction layer error which can be due to an ECRC check failure at the receiver despite the fact that the TLP is not malformed and the LCRC check is valid. The Hard IP block handles this TLP automatically. If the TLP is a non-posted request, the Hard IP block generates a completion with a completer abort status. The TLP is deleted by the Hard IP and not presented to the Application Layer.	Use the configuration output interface, Hard IP reconfiguration interface to obtain more information on the error.
Unsupported request	Transaction layer error which can be due to the completer being unable to fulfill the request. The TLP is deleted in the Hard IP block and not presented to the Application Layer. If the TLP is a non-posted request, the Hard IP block generates a completion with Unsupported Request status.	Use the configuration output interface, error interface, Hard IP reconfiguration interface to obtain more information on the error.
ACS violation	Transaction layer error which can be due to access control error in the received posted or non-posted request.	Use the configuration output interface, error interface, Hard IP reconfiguration interface to obtain more information on the error.
Uncorrectable internal error	Transaction layer error which can be due to an internal error that cannot be corrected by the hardware.	Use the error interface, configuration output interface, Hard IP reconfiguration interface and DBI registers to obtain more information on the error.

continued...

Observation	Issue	Resolution
Atomic egress blocked		Use the error interface, configuration output interface, Hard IP reconfiguration interface to obtain more information on the error.
TLP prefix blocked	EP or RP only	Use the error interface, configuration output interface, Hard IP reconfiguration interface to obtain more information on the error.
Poisoned TLP egress blocked	EP or RP only	Use the error interface, configuration output interface, configuration intercept interface, Hard IP reconfiguration interface to obtain more information on the error.

Use the debug tools mentioned in the next two sections for debugging link training issues observed on the PCI Express link when using the P-Tile Avalon-ST IP for PCI Express.

Related Information

PCI Express Base Specification Revision 4.0 version 1.0

8.1.2.1. Advanced Error Reporting (AER)

Each PCI Express compliant device must implement a basic level of error management and can optionally implement advanced error management. The PCI Express Advanced Error Reporting Capability is an optional Extended Capability that may be implemented by PCI Express device functions supporting advanced error control and reporting.

The P-Tile Avalon-ST IP for PCI Express implements both basic and advanced error reporting. Error handling for a Root Port is more complex than that of an Endpoint. In this IP, the Physical Functions (PFs) are always capable of AER (enabled by default). There is no AER implementation for Virtual Functions (VFs).

Use the AER capability of the IP to identify the type of error and the protocol stack layer in which the error may have occurred. Refer to the *PCI Express Capability Structures* section of the *Configuration Space Registers* appendix for the AER Extended Capability Structure and the associated registers.

8.1.2.2. Second-Level Debug Tools

Use the following debug tools for second-level debug of any issue observed on the PCI Express link when using P-Tile:

Using the Configuration Output Interface

- Refer to the section Configuration Output Interface on page 81 for details on this interface and the address map.

Using the Error Interface

- Refer to the section Error Interface on page 73 for details on this interface and the address map.

Using the Configuration Intercept Interface

- Refer to the section Configuration Intercept Interface (EP Only) on page 85 for details on this interface and the address map.

Using the TX/RX Flow Control Interfaces

- Refer to the sections TX Flow Control Interface on page 63 and RX Flow Control Interface on page 54 for details on these interfaces and their address maps.

Using the Hard IP Reconfiguration Interface

- Refer to the section Hard IP Reconfiguration Interface on page 86 for details on this interface and the address map.

Using the PHY Reconfiguration Interface

- Refer to the section PHY Reconfiguration Interface on page 94 for details on this interface and the address map.

8.2. Debug Toolkit

8.2.1. Overview

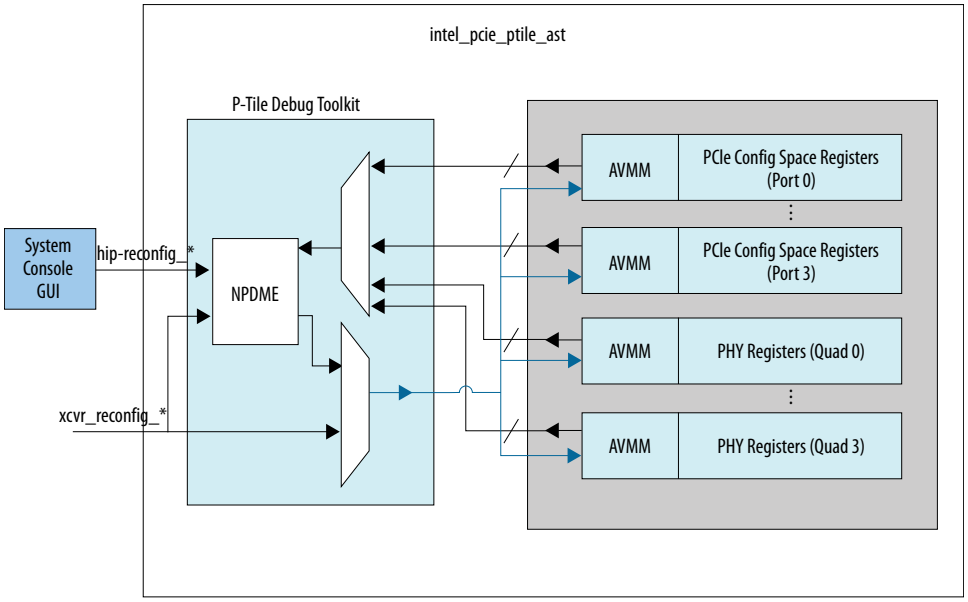
The P-Tile Debug Toolkit is a System Console-based tool for P-Tile that provides real-time control, monitoring and debugging of the PCIe links at the Physical Layer.

The P-Tile Debug Toolkit allows you to:

- View protocol and link status of the PCIe links.
- View PLL and per-channel status of the PCIe links.
- Control the channel analog settings.
- View the receiver eye and measure the eye height and width.
- Indicate the presence of a re-timer connected between the link partners.

The following figure provides an overview of the P-Tile Debug Toolkit in the P-Tile Avalon-ST IP for PCI Express.

Figure 69. Overview of the P-Tile Debug Toolkit



When you enable the P-Tile Debug Toolkit, the `intel_pcie_ptile_ast` module of the generated IP includes the Debug Toolkit modules and related logic as shown in the figure above.

Drive the Debug Toolkit from a System Console. The System Console connects to the Debug Toolkit via an Native PHY Debug Master Endpoint (NPDME). Make this connection via an Intel FPGA Download Cable.

When you enable the Debug Toolkit, the PCIe IP has a Hard IP reconfiguration interface (hip_reconfig_*) brought out at the top level. This interface provides the clock to the following interfaces:

- The NPDME module
- PHY reconfiguration interface (xcvr_reconfig)
- Hard IP reconfiguration interface (hip_reconfig)

Provide a clock source (50 MHz - 125 MHz, 100 MHz recommended clock frequency) to drive the `xcvr_reconfig_clk` clock. Use the output of the Reset Release Intel FPGA IP to drive the `ninit_done`, which provides the reset signal to the NPDME module.

Note: Enable the PHY Reconfiguration Interface to use the P-Tile Debug Toolkit.

When you run a dynamically-generated design example on the Intel Development Kit, make sure that clock and reset signals are connected to their respective sources and appropriate pin assignments are made. Here are some sample `.qsf` assignments for the Debug Toolkit:

- set_location_assignment PIN_A31 -to p0_hip_reconfig_clk_clk
- set_location_assignment PIN_C23 -to xcvr_reconfig_clk_clk
- #set_instance_assignment -name VIRTUAL_PIN ON -to *p0_hip_reconfig*
- #set_instance_assignment -name VIRTUAL_PIN ON -to *xcvr_reconfig*

8.2.2. Enabling the P-Tile Debug Toolkit

To enable the P-Tile Debug Toolkit in your design, enable the option **Enable Debug Toolkit** in the **PCIe Configuration, Debug and Extension** options tab of the Intel FPGA P-Tile Avalon-ST IP for PCI Express. The P-Tile Debug Toolkit also requires the options **Enable HIP dynamic reconfiguration of PCIe registers** and **Enable PHY reconfiguration** to be enabled in the IP Parameter Editor.

Note: When you enable the P-Tile Debug Toolkit in the IP, the Hard IP reconfiguration interface and the PHY reconfiguration interface will be used by the Debug Toolkit. Hence, you will not be able to drive logic on these interfaces from the FPGA fabric.

8.2.3. Launching the P-Tile Debug Toolkit

Use the design example you compiled by following the *Quick Start Guide* to familiarize yourself with the P-Tile Debug Toolkit. Follow the steps in the *Generating the Design Example* and *Compiling the Design Example* to generate the SRAM Object File, (.sof) for this design example.

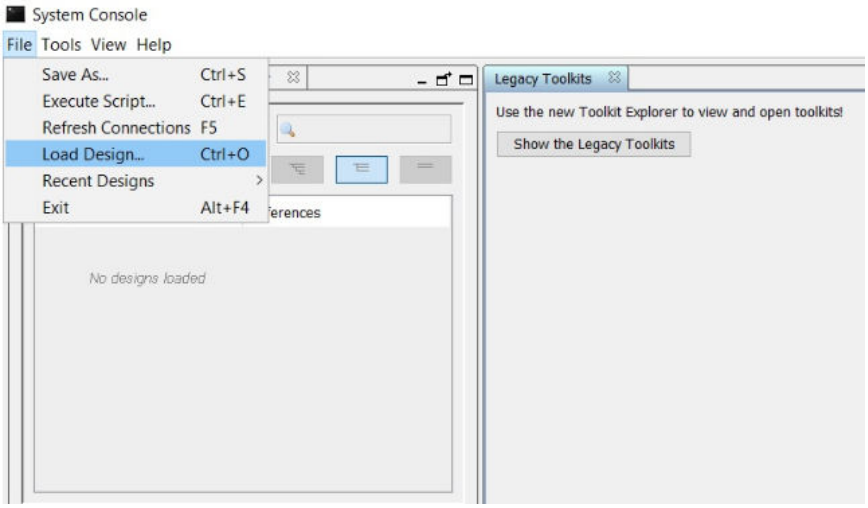
To use the P-Tile Debug Toolkit, download the .sof to the Intel Development Kit. Then, open the System Console and load the design to the System Console as well. Loading the .sof to the System Console allows the System Console to communicate with the design using NPDME. NPDME is a JTAG-based Avalon-MM master. It drives Avalon-MM slave interfaces in the PCIe design. When using NPDME, the Intel Quartus Prime software inserts the debug interconnect fabric to connect with JTAG.

Here are the steps to complete these tasks:

1. Use the Intel Quartus Prime Programmer to download the .sof to the Intel FPGA Development Kit.

   Note: To ensure correct operation, use the same version of the Intel Quartus Prime Programmer and Intel Quartus Prime Pro Edition software that you used to generate the .sof.

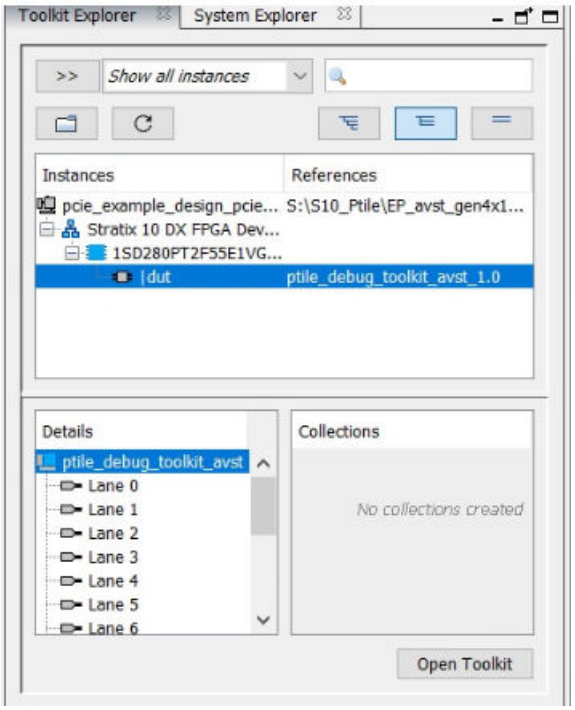
2. To load the design into System Console:
   a. Launch the Intel Quartus Prime Pro Edition software.
   b. Start System Console by choosing **Tools**, then **System Debugging Tools**, then **System Console**.
   c. On the System Console File menu, select **Load design** and browse to the .sof file.



d. Select the .sof and click **OK**. The .sof loads to the System Console.

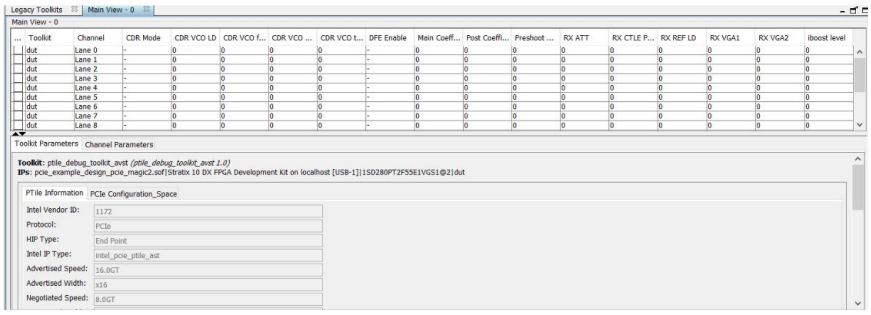
3. The System Console Toolkit Explorer window will list all the DUTs in the design that have the P-Tile Debug Toolkit enabled.

   a. Select the DUT with the P-Tile Debug Toolkit you want to view. This will open the Debug Toolkit instance of that DUT in the **Details** window.



   b. Click on the **ptile_debug_toolkit_avst** to open that instance of the Toolkit. Once the Debug Toolkit is initialized and loaded, you will see the following message in the **Messages** window: **"Initializing P-Tile debug toolkit – done"**.

c. A new window **Main view** will open with a view of all the channels in that instance.

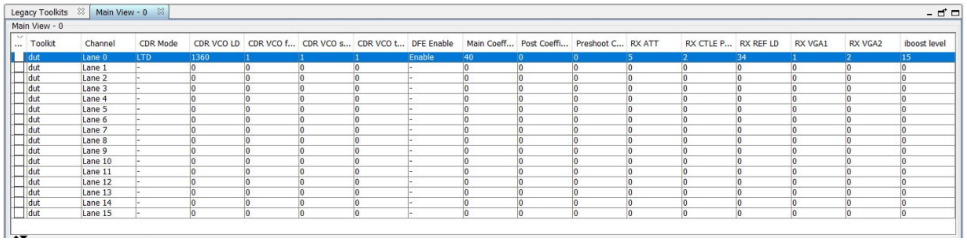


8.2.4. Using the P-Tile Debug Toolkit

The following sections describe the different tabs and features available in the Debug Toolkit.

A. Main View

The main view tab lists a summary of the transmitter and receiver settings per channel for the given instance of the PCIe IP.



The following table shows the channel mapping when using bifurcated ports.

Table 104. Channel Mapping for Bifurcated Ports

Toolkit Channel	X16 Mode	2X8 Mode	4x4 Mode
Lane 0	Lane 0	Lane 0	Lane 0
Lane 1	Lane 1	Lane 1	Lane 1
Lane 2	Lane 2	Lane 2	Lane 2
Lane 3	Lane 3	Lane 3	Lane 3
Lane 4	Lane 4	Lane 4	Lane 0
Lane 5	Lane 5	Lane 5	Lane 1

continued...

Toolkit Channel	X16 Mode	2X8 Mode	4x4 Mode
Lane 6	Lane 6	Lane 6	Lane 2
Lane 7	Lane 7	Lane 7	Lane 3
Lane 8	Lane 8	Lane 0	Lane 0
Lane 9	Lane 9	Lane 1	Lane 1
Lane 10	Lane 10	Lane 2	Lane 2
Lane 11	Lane 11	Lane 3	Lane 3
Lane 12	Lane 12	Lane 4	Lane 0
Lane 13	Lane 13	Lane 5	Lane 1
Lane 14	Lane 14	Lane 6	Lane 2
Lane 15	Lane 15	Lane 7	Lane 3

B. Toolkit Parameters

The Toolkit parameters window has 2 sub-tabs.

B.1. P-Tile Information

This lists a summary of the P-Tile PCIe IP parameter settings in the PCIe IP Parameter Editor when the IP was generated, as read by the P-Tile Debug Toolkit when initialized.

All the information is read-only.

Use the **Get P-tile Info** button to read the settings.

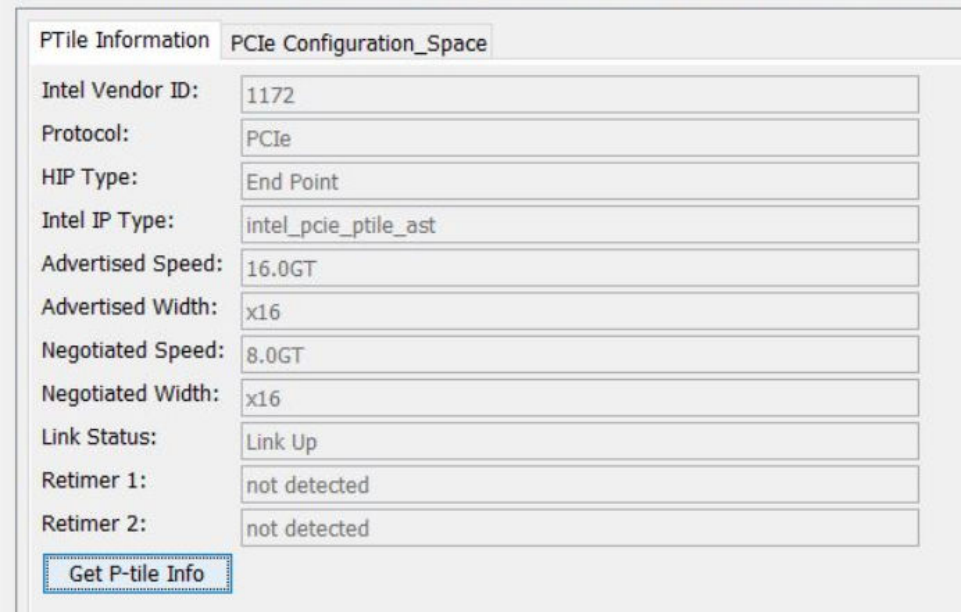
Table 105. P-Tile Available Parameter Settings

Parameter	Values	Descriptions
Intel Vendor ID	1172	Indicates the Vendor ID as set in the IP Parameter Editor.
Protocol	PCIe	Indicates the Protocol.
HIP Type	Root Port, End Point	Indicates the Hard IP Port type.
Intel IP Type	intel_pcie_ptile_ast, intel_pcie_ptile_avmm	Indicates the IP type used.
Advertised speed	Gen3, Gen4	Indicates the advertised speed as configured in the IP Parameter Editor.
Advertised width	x16, x8, x4	Indicates the advertised width as configured in the IP Parameter Editor.
Negotiated speed	Gen3, Gen4	Indicates the negotiated speed during link training.
Negotiated width	x16, x8, x4	Indicates the negotiated link width during link training.

continued...

Parameter	Values	Descriptions
Link status	Link up, link down	Indicates if the link (DL) is up or not.
Retimer 1	Detected, not detected	Indicates if a retimer was detected between the Root Port and the Endpoint.
Retimer 2	Detected, not detected	Indicates if a retimer was detected between the Root Port and the Endpoint.

Figure 70. Example of P-Tile Parameter Settings



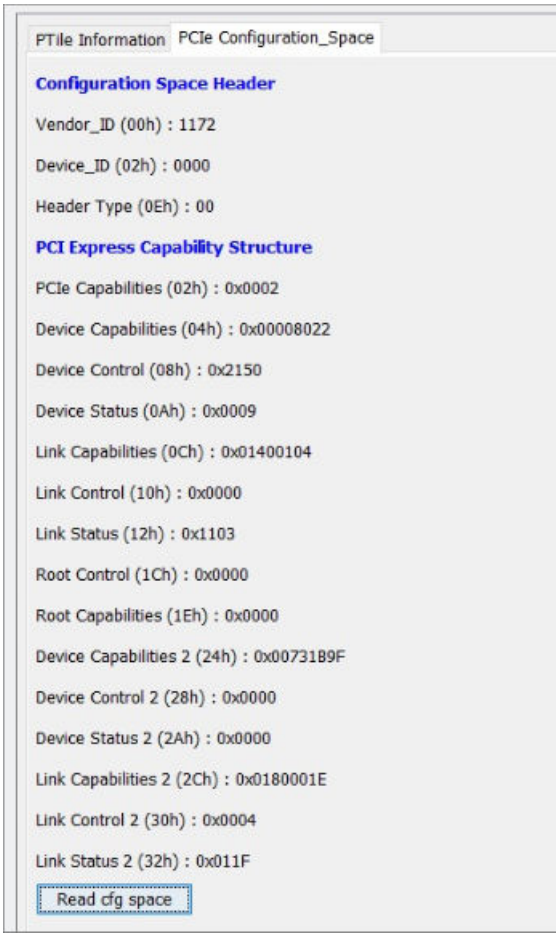
B.2. PCIe Configuration Space

This lists a summary of the P-Tile PCIe configuration settings of the PCIe configuration space registers, as read by the P-Tile Debug Toolkit when initialized.

All the information is read-only.

Use the **Read cfg space** button to read the settings.

Figure 71. Example of P-Tile PCIe Configuration Settings



C. Channel Parameters

The channel parameters window allows you to monitor and control the transmitter and receiver settings for a given channel. It has the following 2 sub-windows.

C.1. TX Path

This tab allows you to monitor and control the transmitter settings for the channel selected. Use the **TX Refresh** button to read the settings, **TX Apply Ch** to apply the settings to the selected channel, and **TX apply all** to apply the settings to all channels.

Table 106. Transmitter Settings

	Parameters	Values	Descriptions
PHY Status	Refclk enable	Enable, Disable	Indicates reference clock is enabled for the PHY. Enable: Reference clock is enabled for the PHY.

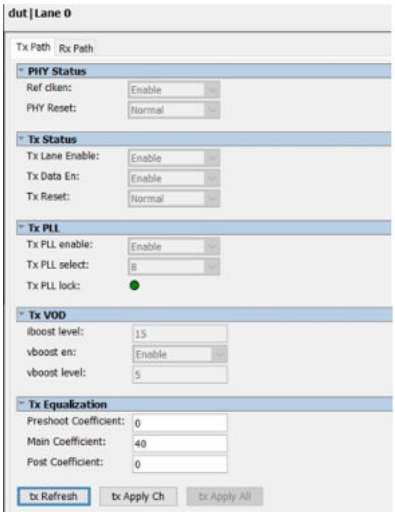
continued...

	Parameters	Values	Descriptions
			Disable: Reference clock is disabled for the PHY.
	PHY reset	Normal, Reset	Indicates the PHY is in reset mode. Normal: PHY is out of reset. Reset: PHY is in reset.
TX Status	TX Lane enable	Enable, Disable	Indicates if TX lane is enabled in the PHY. Enable: TX lane is enabled in the PHY. Disable: TX lane is disabled in the PHY.
	TX Data enable	Enable, Disable	Indicates if TX driver is enabled and serial data is transmitted. Enable: TX driver for the corresponding lane is enabled. Disable: TX driver for the corresponding lane is disabled.
	TX Reset	Normal, Reset	Indicates if TX (TX datapath, TX settings) is in reset or normal operating mode. Normal: TX is in normal operating mode. Reset: TX is in reset.
TX PLL	TX PLL enable	Enable, Disable	Indicates if the TX PLL is powered on or powered down. This is dependent on the PLL selected as indicated by TX PLL select. There is one set of PLLs per Quad. The TX path of each channel reads out the PLL status corresponding to that Quad. • TX path for Ch0 to 3: Status of PLLs in Quad0 • TX path for Ch4 to 7: Status of PLLs in Quad1 • TX path for Ch8 to 11: Status of PLLs in Quad2 • TX path for Ch12 to 15: Status of PLLs in Quad3 Enable: TX PLL is powered on. Disable: TX PLL is powered down.
	TX PLL select	PLLA: Gen1/Gen2 PLLB: Gen3/Gen4	Indicates which PLL is selected. There is one set of PLLs per Quad. The TX path of each channel reads out the PLL status corresponding to that Quad.

continued...

	Parameters	Values	Descriptions
			• TX path for Ch0 to 3: Status of PLLs in Quad0 • TX path for Ch4 to 7: Status of PLLs in Quad1 • TX path for Ch8 to 11: Status of PLLs in Quad2 • TX path for Ch12 to 15: Status of PLLs in Quad3
	TX PLL lock	Green, Red	Indicates if TX PLL is locked. This is dependent on the PLL selected as indicated by TX PLL select. There is one set of PLLs per Quad. The TX path of each channel reads out the PLL status corresponding to that Quad. • TX path for Ch0 to 3: Status of PLLs in Quad0 • TX path for Ch4 to 7: Status of PLLs in Quad1 • TX path for Ch8 to 11: Status of PLLs in Quad2 • TX path for Ch12 to 15: Status of PLLs in Quad3 Green: TX PLL is locked. Red: TX PLL is not locked.
TX VOD	Iboost level	Gen3: 15 Gen4: 15	Indicates the transmitter current boost level when the TX amplitude boost mode is enabled.
	Vboost en	Gen3: Enable Gen4: Enable	Indicates if the TX swing boost level is enabled. Enable: TX swing boost is enabled. Disable: TX swing boost is disabled.
	Vboost level	Gen3: 5 Gen4: 5	Indicates the TX Vboost level.
TX Equalization	Pre-shoot coefficient	Gen3: 20 (Preset 8) Gen4: 0 (Preset 0)	Indicates transmitter driver output pre-emphasis (pre-shoot coefficient).
	Main coefficient	Gen3: 30 (Preset 8) Gen4: 30 (Preset 0)	Indicates transmitter driver output pre-emphasis (main coefficient).
	Post coefficient	Gen3: 20 (Preset 8) Gen4: 40 (Preset 0)	Indicates transmitter driver output pre-emphasis (post coefficient).

Figure 72. Example of Transmitter Settings



C.2. RX Path

This tab allows you to monitor and control the receiver settings for the channel selected. Use the **RX Refresh** button to read the settings, **RX Apply Ch** to apply the settings to the selected channel, and **RX apply all** to apply the settings to all channels.

Table 107. Receiver Settings

	Parameters	Values	Descriptions
RX Status	RX Lane enable	Enable, Disable	Indicates if RX lane is enabled in the PHY. Enable: RX lane is enabled in the PHY. Disable: RX lane is disabled in the PHY.
	RX Data enable	Enable, Disable	Indicates if RX driver is enabled and serial data is transmitted. Enable: RX driver for the corresponding lane is enabled. Disable: RX driver for the corresponding lane is disabled.
	RX Reset	Normal, Reset	Indicates if RX (RX datapath, RX settings) is in reset or normal operating mode. Normal: RX is in normal operating mode. Reset: RX is in reset.
	RX LOS	<1,0>	Indicates if the receiver has lost the signal. 1: Receiver loss of signal. 0: Receiver has a data signal.

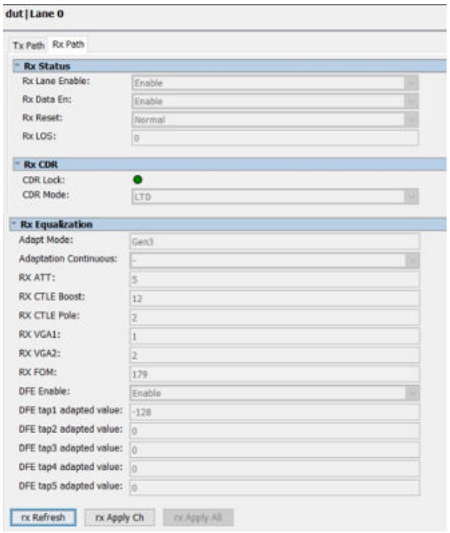
continued...

	Parameters	Values	Descriptions
RX CDR	CDR Lock	Green, Red	Indicates the CDR lock state. Green: CDR is locked. Red: CDR is not locked.
	CDR Mode	Locked to Reference (LTR), Locked to Data (LTD)	Indicates the CDR lock mode. LTR: CDR is locked to reference clock. LTD: CDR is locked to data.
RX Equalization	Adapt Mode	Gen3: Gen3 adaptation mode. Gen4: Gen4 adaptation mode.	Indicates the RX adaptation mode.
	Adapt Continuous	Gen3: 1 Gen4: 1	Indicates if the receiver is in continuous adaptation. • 0 - continuous adaptation off. • 1 - continuous adaptation on.
	RX ATT	Gen3: 0 Gen4: 0	Indicates the RX equalization attenuation level.
	RX CTLE Boost	Gen3: 12 Gen4: 16	Indicates the RX CTLE boost value.
	RX CTLE Pole	Gen3: 2 Gen4: 2	Indicates the RX CTLE pole value.
	RX VGA1	Gen3: 5 Gen4: 5	Indicates the RX AFE first stage VGA gain value.
	RX VGA2	Gen3: 5 Gen4: 5	Indicates the RX AFE second stage VGA gain value.
	RX FOM	<0 - 255>	Indicates the Receiver Figure of Merit (FOM) / quality of the received data eye. A higher value indicates better link equalization, with 8'd0 indicating the worst equalization setting and 8'd255 indicating the best equalization setting.
	DFE Enable	Enable, Disable	Indicates DFE adaptation is enabled for taps 1 - 5. Enable: DFE adaptation is enabled for taps 1 - 5. Disable: DFE adaptation is disabled for taps 1 - 5.
	DFE Tap1 adapted value	<-128 to 127>	Indicates the adapted value of DFE tap 1. This is a signed input (two's complement encoded).
	DFE Tap2 adapted value	<-32 to 31>	Indicates the adapted value of DFE tap 2. This is a signed input (two's complement encoded).

continued...

	Parameters	Values	Descriptions
	DFE Tap3 adapted value	<-32 to 31>	Indicates the adapted value of DFE tap 3. This is a signed input (two's complement encoded).
	DFE Tap4 adapted value	<-32 to 31>	Indicates the adapted value of DFE tap 4. This is a signed input (two's complement encoded).
	DFE Tap5 adapted value	<-32 to 31>	Indicates the adapted value of DFE tap 5. This is a signed input (two's complement encoded).

Figure 73. Example of Receiver Settings

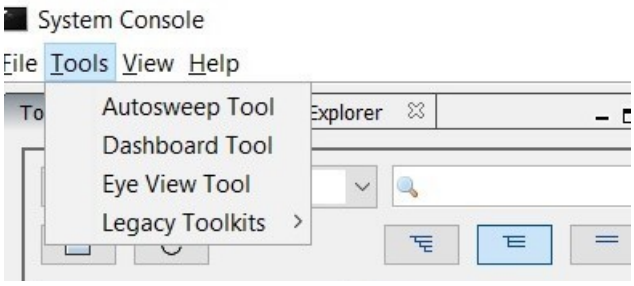


D. Eye Viewer

The P-Tile Debug Toolkit supports running eye tests for Intel devices with P-Tile. The Eye Viewer tool allows you to set up and run eye tests, monitoring bit errors.

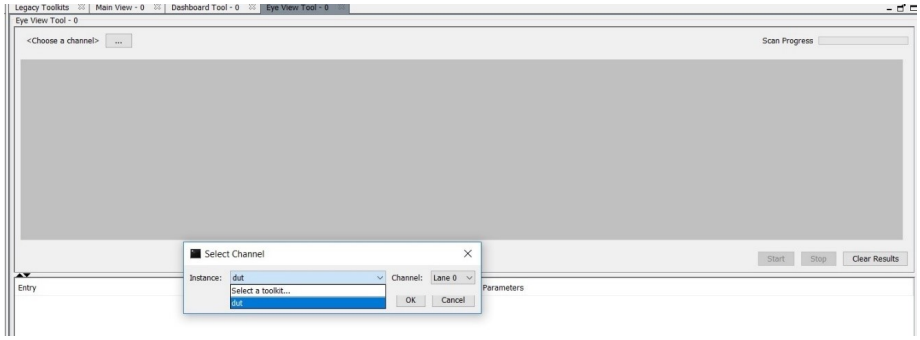
1. In the System Console **Tools** menu option, click on **Eye View Tool**.

Figure 74. Opening the Eye Viewer



2. This will open a new tab **Eye View Tool** next to the **Main View** tab. Choose the instance and channel for which you want to run the eye view tests.

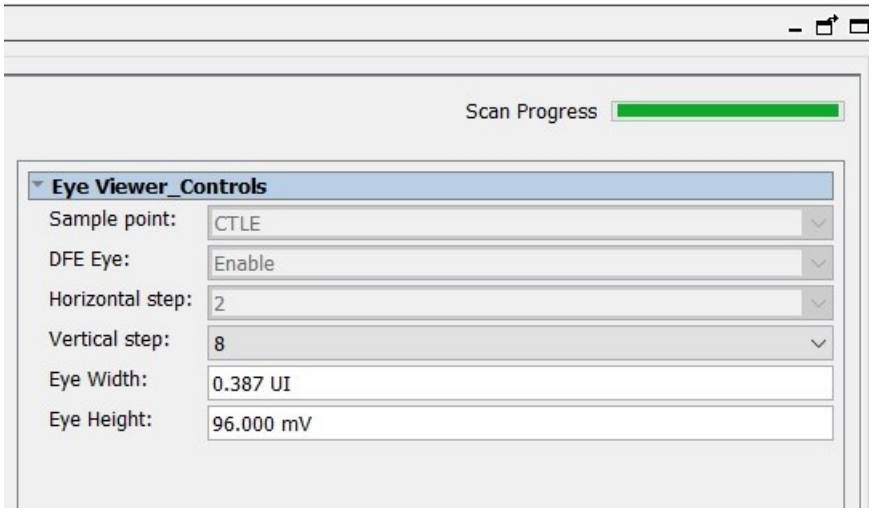
Figure 75. Opening the Instance and Channel



3. Choose the eye vertical step setting from the drop-down menu. The eye view tool allows you to choose between vertical step sizes of 1, 2, 4, 8.

   Note: The time taken for the eye view tool to draw the eye varies with different vertical step sizes (1 results in a faster eye plot when compared to 8).

Figure 76. Choosing the Step Size

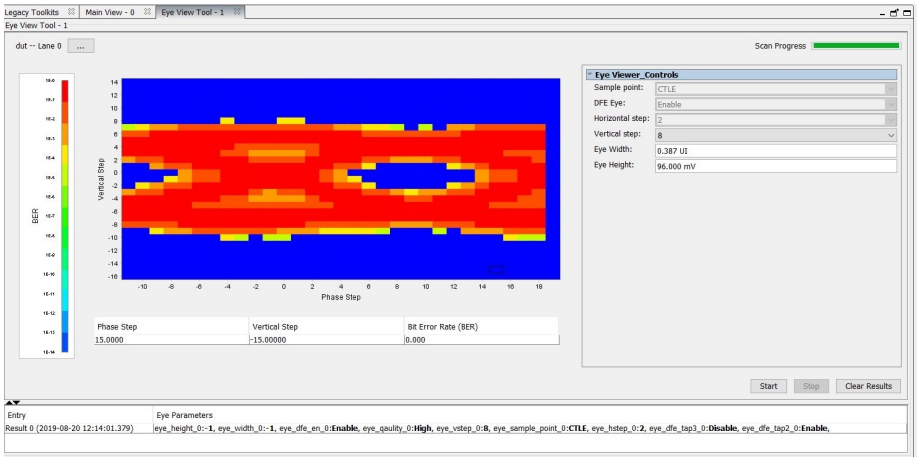


4. The messages window displays information messages to indicate the eye view tool's progress.

Figure 77. Eye View Tool Messages

5. Once the eye plot is complete, the eye height, eye width and eye diagram are displayed.

Figure 78. Sample Eye Plot



A. Configuration Space Registers

A.1. Configuration Space Registers

In addition to accessing the Endpoint's configuration space registers by sending Configuration Read/Write TLPs via the Avalon-ST interface, the application logic can also gain read access to these registers via the Configuration Output Interface (`tl_cfg*`). Furthermore, the Hard IP Reconfiguration Interface (a User Avalon-MM interface) also provides read/write access to these registers.

For signal timings on the User Avalon-MM interface, refer to the *Avalon Interface Specifications* document.

The table *PCIe Configuration Space Registers* describes the registers for each PF. To calculate the address for a particular register in a particular PF, add the offset for that PF from the table *Configuration Space Offsets* to the byte address for that register as given in the table *PCIe Configuration Space Registers*.

Table 108. Configuration Space Offsets

Registers	User Avalon-MM Offsets
Physical function 0	0x00000
Physical function 1	0x10000
Physical function 2	0x20000
Physical function 3	0x30000
Physical function 4	0x40000
Physical function 5	0x50000
Physical function 6	0x60000
Physical function 7	0x70000
Port Configuration and Status Register	0x104000
Debug (DBI) Register	0x104200, 0x104204

Table 109. PCIe Configuration Space Registers for x16/x8/x4 Controllers

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
x16 (Port 0) = 0x000 : 0x03C x8 (Port 1) = 0x000 : 0x03C x4 (Ports 2,3) = 0x000 : 0x03C	PCI Header Type 0/1 Configuration Registers	Type 0/1 Configuration Space Header
x16 (Port 0) = 0x040 : 0x044 x8 (Port 1) = 0x040 : 0x044 x4 (Ports 2,3) = 0x040 : 0x044	Power Management	PCI Power Management Capability Structure

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
x16 (Port 0) = 0x050 : 0x064 x8 (Port 1) = 0x050 : 0x064 x4 (Ports 2,3) = 0x050 : 0x064	MSI Capability	MSI Capability Structure, see also PCI Local Bus Specification
x16 (Port 0) = 0x070 : 0x0A8 x8 (Port 1) = 0x070 : 0x0A8 x4 (Ports 2,3) = 0x070 : 0x0A8	PCI Express Capability	PCI Express Capability Structure
x16 (Port 0) = 0x0B0 : 0x0B9 x8 (Port 1) = 0x0B0 : 0x0B9 x4 (Ports 2,3) = 0x0B0 : 0x0B9	MSI-X Capability	MSI-X Capability Structure, see also PCI Local Bus Specification
x16 (Port 0) = 0x0BC : 0x0FC x8 (Port 1) = 0x0BC : 0x0FC x4 (Ports 2,3) = 0x0BC : 0x0FC	Reserved	N/A
x16 (Port 0) = 0x100 : 0x144 x8 (Port 1) = 0x100 : 0x144 x4 (Ports 2,3) = 0x100 : 0x144	Advanced Error Reporting (AER)	Advanced Error Reporting Capability Structure
x16 (Port 0) = 0x148 : 0x164 x8 (Port 1) = 0x148 : 0x164 x4 (Ports 2,3) = 0x148 : 0x164	Virtual Channel Capability	Virtual Channel Capability Structure
x16 (Port 0) = 0x178 : 0x17C x8 (Port 1) = 0x178 : 0x17C x4 (Ports 2,3) = N/A	Alternative Routing-ID Implementation (ARI)	ARI Capability Structure
x16 (Port 0) = 0x188 : 0x1B4 x8 (Port 1) = 0x188 : 0x1A4 x4 (Ports 2,3) = 0x188 : 0x1A4	Secondary PCI Express Extended Capability Header	PCI Express Extended Capability
x16 (Port 0) = 0x1B8 : 0x1E4 x8 (Port 1) = 0x1A8 : 0x1CC x4 (Ports 2,3) = 0x1A8 : 0x1C8	Physical Layer 16.0 GT/s Extended Capability	Physical Layer 16.0 GT/s Extended Capability Structure
x16 (Port 0) = 0x1E8 : 0x22C x8 (Port 1) = 0x1D0 : 0x1F4 x4 (Ports 2,3) = 0x1CC : 0x1E0	Margining Extended Capability	Margining Extended Capability Structure
x16 (Port 0) = 0x230 : 0x26C x8 (Port 1) = 0x1F8 : 0x234 x4 (Ports 2,3) = N/A	SR-IOV Capability	SR-IOV Capability Structure
x16 (Port 0) = 0x270 : 0x2F8 x8 (Port 1) = 0x238 : 0x2C0 x4 (Ports 2,3) = 0x1E4 : 0x26C	TLP Processing Hints (TPH) Capability	TLP Processing Hints (TPH) Capability Structure
x16 (Port 0) = 0x2FC : 0x300 x8 (Port 1) = 0x2C4 : 0x2C8 x4 (Ports 2,3) = N/A	Address Translation Services (ATS) Capability	Address Translation Services Extended Capability (ATS) in Single Root I/O Virtualization and Sharing Specification
x16 (Port 0) = 0x30C : 0x314 x8 (Port 1) = 0x2D4 : 0x2DC x4 (Ports 2,3) = 0x280 : 0x288	Access Control Services (ACS) Capability	Access Control Services (ACS) Capability
x16 (Port 0) = 0x318 : 0x324 x8 (Port 1) = 0x2E0 : 0x2EC x4 (Ports 2,3) = N/A	Page Request Services (PRS) Capability	Page Request Services (PRS) Capability

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
x16 (Port 0) = 0x328 : 0x32C x8 (Port 1) = 0x2F0 : 0x2F4 x4 (Ports 2,3) = N/A	Latency Tolerance Reporting (LTR) Capability	Latency Tolerance Reporting (LTR) Capability
x16 (Port 0) = 0x330 : 0x334 x8 (Port 1) = 0x2F8 : 0x2FC x4 (Ports 2,3) = N/A	Process Address Space (PASID) Capability	Process Address Space (PASID) Capability Structure
x16 (Port 0) = 0x338 : 0x434 x8 (Port 1) = 0x300 : 0x3FC x4 (Ports 2,3) = 0x2AC : 0x3A8	RAS D.E.S. Capability (VSEC)	
x16 (Port 0) = 0x470 : 0x478 x8 (Port 1) = 0x438 : 0x440 x4 (Ports 2,3) = 0x3E4 : 0x3EC	Data Link Feature Extended Capability	
x16 (Port 0) = 0xD00 : 0xD58 x8 (Port 1) = 0xD00 : 0xD58 x4 (Ports 2,3) = 0xD00 : 0xD58	Intel-defined VSEC	

A.1.1. Register Access Definitions

This document uses the following abbreviations when describing register accesses.

Table 110. Register Access Abbreviations

Abbreviation	Meaning
RW	Read and write access
RO	Read only
WO	Write only
RW1C	Read write 1 to clear
RW1CS	Read write 1 to clear sticky
RWS	Read write sticky

Note: Sticky bits are not initialized or modified by hot reset or function-level reset.

A.1.2. PCIe Configuration Header Registers

The *Corresponding Section in PCIe Specification* column in the tables in the *Configuration Space Registers* section lists the appropriate sections of the *PCI Express Base Specification* that describe these registers.

Figure 79. PCIe Type 0 Configuration Space Registers - Byte Address Offsets and Layout

	31 24	23 16	15 8	7 0
0x000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class Code			Revision ID
0x00C	0x00	Header Type	0x00	Cache Line Size
0x010	BAR Registers			
0x014	BAR Registers			
0x018	BAR Registers			
0x01C	BAR Registers			
0x020	BAR Registers			
0x024	BAR Registers			
0x028	Reserved			
0x02C	Subsystem Device ID		Subsystem Vendor ID	
0x030	Reserved			
0x034	Reserved			Capabilities Pointer
0x038	Reserved			
0x03C	0x00		Interrupt Pin	Interrupt Line

Figure 80. PCIe Type 1 Configuration Space Registers - Byte Address Offsets and Layout

	31 24	23 16	15 8	7 0
0x0000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class Code			Revision ID
0x00C	BIST	Header Type	Primary Latency Timer	Cache Line Size
0x010	BAR Registers			
0x014	BAR Registers			
0x018	Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number
0x01C	Secondary Status		I/O Limit	I/O Base
0x020	Memory Limit		Memory Base	
0x024	Prefetchable Memory Limit		Prefetchable Memory Base	
0x028	Prefetchable Base Upper 32 Bits			
0x02C	Prefetchable Limit Upper 32 Bits			
0x030	I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits	
0x034	Reserved			Capabilities Pointer
0x038	Expansion ROM Base Address			
0x03C	Bridge Control		Interrupt Pin	Interrupt Line

Related Information

PCI Express Base Specification 4.0

A.1.3. PCI Express Capability Structures

The layouts of the most basic Capability Structures are provided below. Refer to the *PCI Express Base Specification* for more information about these registers.

Figure 81. Power Management Capability Structure - Byte Address Offsets and Layout

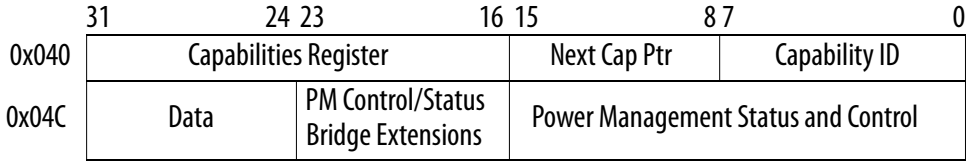


Figure 82. MSI Capability Structure

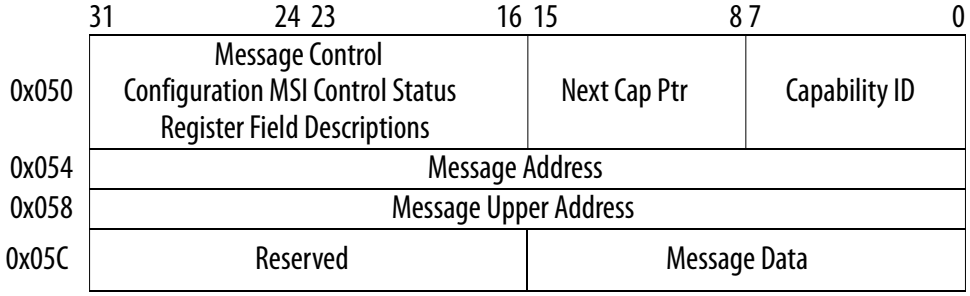


Figure 83. PCI Express Capability Structure - Byte Address Offsets and Layout

In the following table showing the PCI Express Capability Structure, registers that are not applicable to a device are reserved.

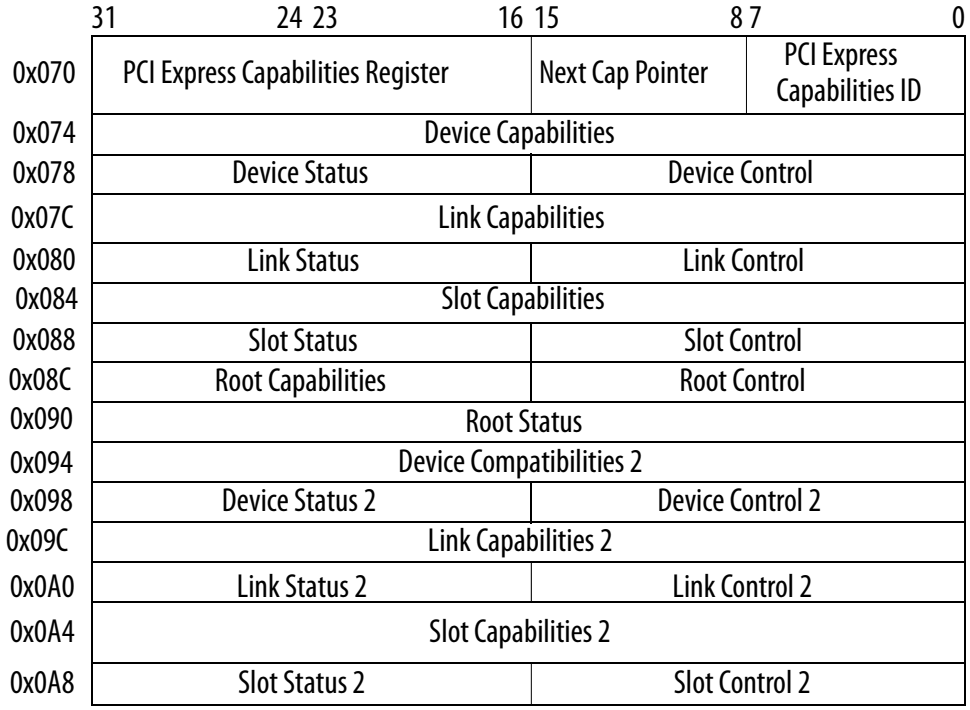


Figure 84. MSI-X Capability Structure

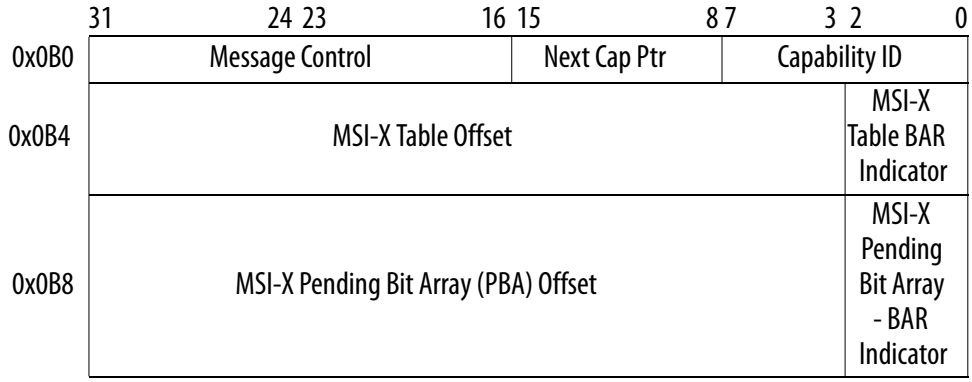
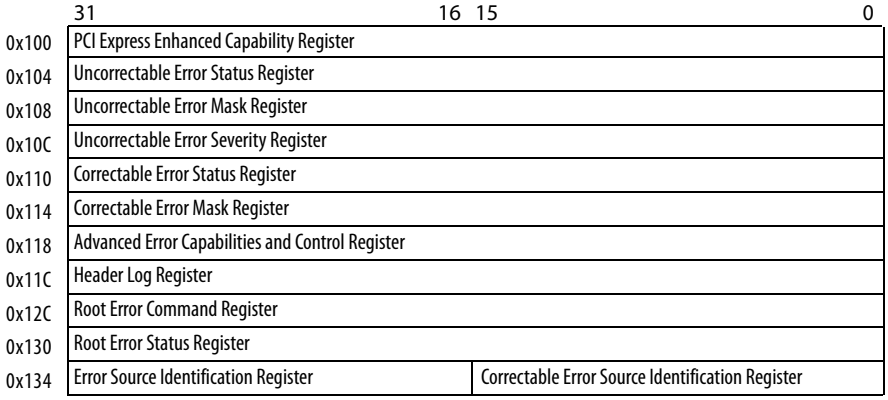


Figure 85. PCI Express AER Extended Capability Structure



Related Information

PCI Express Base Specification 4.0

A.1.4. MSI-X Registers

This section describes the registers previously shown in the MSI-X capability structure.

Table 111. MSI-X Control Register

Bit Location	Description	Access	Default Value
31	MSI-X Enable: This bit must be set to enable the MSI-X interrupt generation. You need to obtain this information from the Configuration Intercept Interface.	RW	0
30	MSI-X Function Mask: This bit can be set to mask all MSI-X interrupts from this function.	RW	0

continued...

Bit Location	Description	Access	Default Value
	You need to obtain this information from the Configuration Intercept Interface.		
29:27	Reserved	RO	0
26:16	Size of the MSI-X table (number of MSI-X interrupt vectors). The value in this field is one less than the size of the table set up for this function. Maximum value is 0x7FF (2048 interrupt vectors). This field is shared among all VFs attached to one PF.	RO	Programmed via the programming interface.
15:8	Next Capability Pointer Points to the PCI Express Capability.	RO	Programmed via the programming interface.
7:0	Capability ID assigned by PCI-SIG.	RO	0x11

Table 112. MSI-X Table Offset Register

Bit Location	Description	Access	Default Value
2:0	BAR Indicator Register: Specifies the BAR corresponding to the memory address range where the MSI-X table of this function is located (000 = VF BAR0, 001 = VF BAR1, …, 101 = VF BAR5). This field is shared among all VFs attached to one PF.	RO	Programmed via the programming interface.
31:3	Offset of the memory address where the MSI-X table is located, relative to the specified BAR. The address is extended by appending three zeroes to make it Qword aligned. This field is shared among all VFs attached to one PF.	RO	Programmed via the programming interface.

Table 113. MSI-X Pending Bit Array Register

Bit Location	Description	Access	Default Value
2:0	BAR Indicator Register: Specifies the BAR corresponding to the memory address range where the Pending Bit Array of this function is located (000 = VF BAR0, 001 = VF BAR1, …, 101 = VF BAR5).	RO	Programmed via the programming interface.

continued...

Bit Location	Description	Access	Default Value
	This field is shared among all VFs attached to one PF.		
31:3	Offset of the memory address where the Pending Bit Array is located, relative to the specified BAR. The address is extended by appending three zeroes to make it Qword aligned. This field is shared among all VFs attached to one PF.	RO	Programmed via the programming interface.

A.2. Configuration Space Registers for Virtualization

A.2.1. SR-IOV Virtualization Extended Capabilities Registers Address Map

Figure 86. SR-IOV Virtualization Extended Capabilities Registers (0x230 : 0x26C)

	31 24	23 20	19 16	15 0
0x230	SR-IOV Extended Capability Header Register			
0x234	SR-IOV Capabilities			
0x238	SR-IOV Status			SR-IOV Control
0x23C	TotalVFs (RO)			InitialVFs (RO)
0x240	RsvdP	Function Dependency Link (RO)		NumVFs (RW)
0x244	VF Stride (RO)			First VF Offset (RO)
0x248	VF Device ID (RO)			RsvdP
0x24C	Supported Pages Sizes (RO)			
0x250	System Page Size (RW)			
0x254	VF BAR0 (RW)			
0x258	VF BAR1 (RW)			
0x25C	VF BAR2 (RW)			
0x260	VF BAR3 (RW)			
0x264	VF BAR4 (RW)			
0x268	VF BAR5 (RW)			
0x26C	VF Migration State Array (RO)			

A.2.2. PCIe Configuration Registers for Each Virtual Function

This section provides a description of the individual registers in the configuration space of each virtual function (VF). To access the configuration space of VFs, refer to the section *Address Map for the User Avalon-MM Interface*.

Table 114. PCIe Configuration Registers for Each Virtual Function

Address Range	Register Description
0x0 : 0x3C	VF PCI-Compatible Configuration Space Header Type0
0x70 : 0xA0	VF PCI Express Capability Structure
0xB0 : 0xB8	VF MSI-X Capability Structure
0x178 : 0x17C	VF Alternative Routing ID (ARI) Capability Structure
0x270 : 0x27C	VF TLP Processing Hints Capability Structure
0x2FC : 0x300	VF Address Translation Services Capability Structure
0x30C : 0x314	VF Access Control Services (ACS) Capability Structure

A.2.2.1. Alternative Routing ID (ARI) Capability Structure

A.2.2.1.1. ARI Enhanced Capability Header Register (Offset 0x0)

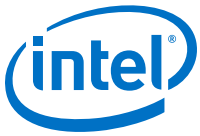
Table 115. ARI Enhanced Capability Header Register

Bits	Register Description	Default Value	Access
[15:0]	PCI Express Extended Capability ID for ARI	0x000E	RO
[19:16]	Capability Version	0x1	RO
[31:20]	Next Capability Pointer: When TPH Requester Capability is present, points to TPH Requester Capability. When ATS Capability is present, but TPH Requester Capability is not, points to ATS Capability. When neither TPH Requester Capability nor ATS Capability is present, its value is 0.	See description. Programmed via Programming Interface.	RO

A.2.2.1.2. ARI Capability and Control Register (Offset 0x4)

The lower 16 bits of this location contain the ARI Capability Register and the upper 16 bits contain the ARI Control Register. All the fields in these registers are hardwired to 0 for all VFs.

ARI is enabled only if SR-IOV is enabled. Therefore, ARI support may be available in a future Intel Quartus Prime release.



A.2.2.2. TLP Processing Hint (TPH) Capability Structure

A.2.2.2.1. TPH Requester Enhanced Capability Header (Offset 0x0)

Table 116. TPH Requester Enhanced Capability Header

Bits	Register Description	Default Value	Access
[15:0]	PCI Express Extended Capability ID	0x0017	RO
[19:16]	Capability Version	0x1	RO
[31:20]	Next Capability Pointer: Points to ATS Capability when present, NULL otherwise.	Programmed via Programming Interface.	RO

A.2.2.2.2. TPH Requester Capability Register (Offset 0x4)

This is a read-only register that specifies the capabilities associated with the implementation of TPH in the device.

Note: Steering Tag (ST) table must be implemented in the user logic if present. The capability will not hold the ST table.

Table 117. TPH Requester Capability Register

Bits	Register Description	Default Value	Access
[0]	`No ST Mode` Supported: When set to 1, indicates that this Function supports the `No ST Mode` for the generation of TPH Steering Tags. In the `No ST Mode`, the device must use a Steering Tag value of 0 for all requests. This bit is hardwired to 1, as all TPH Requesters are required to support the `No ST Mode` of operation.	0x1	RO
[1]	`Interrupt Vector Mode` Supported: A setting of 1 indicates that the Function supports the `Interrupt Vector Mode` for TPH Steering Tag generation. In the `Interrupt Vector Mode`, Steering Tags are attached to MSI/MSI-X interrupt requests. The Steering Tag for each interrupt request is selected by the MSI/MSI-X interrupt vector number.	Programmed via Programming Interface.	RO
[2]	`Device Specific Mode` Supported: A setting of 1 indicates that the Function supports the `Device Specific Mode` for TPH Steering Tag generation. The	Programmed via Programming Interface.	RO

continued...

Bits	Register Description	Default Value	Access
	client typically choses the Steering Tag values from the ST Table, but is not required to do so.		
[7:3]	Reserved	0x0	RO
[8]	Extended TPH Requester Supported: When set to 1, indicates that the Function is capable of generating requests with 16-bit Steering Tags, using TLP Prefix.	Programmed via Programming Interface.	RO
[10:9]	ST Table Location: The setting of this field indicates if a Steering Tag table is implemented for this Function, and its location if present. • 2'b00 = ST Table not present • 2'b01 = ST Table stored in the TPH Requester Capability Structure • 2'b10 = ST values stored in the MSI-X Table in client RAM • 2'b11 = reserved Valid settings are 2'b00 or 2'b10.	Programmed via Programming Interface.	RO
[15:11]	Reserved	0x0	RO
[26:16]	ST Table Size: Specifies the number of entries in the Steering Tag table (0 = 1 entry, 1 = 2 entries, and so on). The maximum table size is 2048 entries when located in the MSI-X table. Each entry is 8 bits.	Programmed via Programming Interface.	RO
[31:27]	Reserved	0x0	RO

A.2.2.2.3. TPH Requester Control Register (Offset 0x8)

Table 118. TPH Requester Control Register

Bits	Register Description	Default Value	Access
[2:0]	Steering Tag (ST) Mode: This field selects the ST mode: • 3'b000 = No Steering Tag Mode • 3'b001 = Interrupt Vector Mode • 3'b010 = Device-Specific Mode • 3'b011 - 3'b111 = Reserved	0x0	RW

continued...

Bits	Register Description	Default Value	Access
	You need to obtain this information from the configuration intercept interface.		
[7:3]	Reserved	0x0	RO
[8]	TPH Requester Enable: When set to 1, the Function is allowed to generate requests with TLP Processing Hints. You need to obtain this information from the configuration intercept interface.	0x0	RW
[31:9]	Reserved	0x0	RO

A.2.2.3. Address Translation Services (ATS) Capability Structure

A.2.2.3.1. ATS Enhanced Capability Header (Offset 0x0)

Table 119. ATS Enhanced Capability Header

Bits	Register Description	Default Value	Access
[15:0]	PCI Express Extended Capability ID	0x000F	RO
[19:16]	Capability Version	0x1	RO
[31:20]	Next Capability Pointer: Points to Null	See description. Programmed via Programming Interface.	RO

A.2.2.3.2. ATS Capability Register and ATS Control Register (Offset 0x4)

The lower 16 bits of this location make up the ATS Capability Register, and the upper 16 bits make up the ATS Control Register.

Table 120. ATS Capability Register and ATS Control Register

Bits	Register Description	Default Value	Access
[4:0]	Invalidate Queue Depth: The number of Invalidate Requests that the Function can accept before throttling the upstream connection. If 0, the Function can accept 32 Invalidate Requests. This field is hardwired to 0 for VFs. VFs use the setting from the parent PF's ATS Capability Register.	0x0	RO
[5]	Page Aligned Request: If set, indicates the untranslated address is always aligned to a 4096-byte boundary. This bit is hardwired to 1.	0x1	RO

continued...

Bits	Register Description	Default Value	Access
[15:6]	Reserved	0x0	RO
[20:16]	Smallest Translation Unit (STU): This value indicates to the Function the minimum number of 4096-byte blocks specified in a Translation Completion or Invalidate Request. This is a power of 2 multiplier. The number of blocks is 2^{STU}. A value of 0 indicates one block and a value of 0x1F indicates 2^{31} blocks, or 8 terabytes (TB) total. This field is hardwired to 0 for VFs. VFs use the setting from the parent PF's ATS Control Register.	0x0	RO
[30:21]	Reserved	0x0	RO
[31]	Enable (E) bit. When Set, the Function can cache translations. You need to obtain this information from configuration intercept interface.	0x0	RW

A.2.2.4. Access Control Services (ACS) Capability Structure

The VF implements the ACS capability structure with bits hardwired to 0. This allows the OS/Virtual Machine Manager (VMM) to properly assign a device to the guest Virtual Machine (VM).

A.2.2.4.1. ACS Extended Capability Header (Offset 0x0)

Table 121. ACS Extended Capability Header

Bits	Register Description	Default Value	Access
[15:0]	PCI Express Extended Capability ID	0x000D	RO
[19:16]	Capability Version	0x1	RO
[31:20]	Next Capability Pointer: Points to Null	See description. Programmed via Programming Interface.	RO

A.2.2.4.2. ACS Capability and Control Register (Offset 0x4)

Table 122. ACS Capability and Control Register

Bits	Register Description	Default Value	Access
[31:0]	Capability and Control Fields	0x0	RO

A.2.2.4.3. Egress Control Vector (Offset 0x8)

Table 123. Egress Control Vector

Bits	Register Description	Default Value	Access
[31:0]	Egress Control Vector	0x0	RO

A.3. Intel-Defined VSEC Capability Registers

Table 124. Intel-Defined VSEC Capability Registers (0xD00 : 0xD58)

31 : 24	23 : 16	15 : 8	7 : 0	PCIe Byte Offset
Next Cap Offset	Version	PCI Express Extended Capability ID		00h
VSEC Length	VSEC Rev	VSEC ID		04h
Intel Marker				08h
JTAG Silicon ID DW0				0Ch
JTAG Silicon ID DW1				10h
JTAG Silicon ID DW2				14h
JTAG Silicon ID DW3				18h
CvP Status		User Configurable Device/Board ID		1Ch
CvP Mode Control				20h
CvP Data 2				24h
CvP Data				28h
CvP Programming Control				2Ch
General Purpose Control and Status				30h
Uncorrectable Internal Error Status Register				34h
Uncorrectable Internal Error Mask Register				38h
Correctable Error Status Register				3Ch
Correctable Error Mask Register				40h
SSM IRQ Request & Status				44h
SSM IRQ Result Code 1 Shadow				48h
SSM IRQ Result Code 2 Shadow				4Ch
SSM Mailbox				50h
SSM Credit 0 Shadow				54h
SSM Credit 1 Shadow				58h

A.3.1. Intel-Defined VSEC Capability Header (Offset 00h)

Table 125. Intel-Defined VSEC Capability Header

Bits	Register Description	Default Value	Access
[31:20]	Next Capability Pointer. Value is the starting address of the next Capability Structure implemented, if any. Otherwise, NULL. Refer to the Configuration Address Map.	Variable	RO
[19:16]	Capability Version. PCIe specification-defined value for VSEC Capability Version.	0x1	RO
[15:0]	Extended Capability ID. PCIe specification-defined value for VSEC Extended Capability ID.	0x000B	RO

A.3.2. Intel-Defined Vendor Specific Header (Offset 04h)

Table 126. Intel-Defined Vendor Specific Header

Bits	Register Description	Default Value	Access
[31:20]	VSEC Length. Total length of this structure in bytes.	0x5C	RO
[19:16]	VSEC Rev. User configurable VSEC revision.	k_vsec_rev_i	RO
[15:0]	VSEC ID. User configurable VSEC ID. The default value is 0x1172 (the Intel Vendor ID), but you can change this ID to your own Vendor ID.	0x1172	RO

A.3.3. Intel Marker (Offset 08h)

Table 127. Intel Marker

Bits	Register Description	Default Value	Access
[31:0]	Intel Marker - An additional marker for standard Intel programming software to be able to verify that this is the right structure.	0x41721172	RO

A.3.4. JTAG Silicon ID (Offset 0x0C - 0x18)

This read-only register returns the JTAG Silicon ID. Intel programming software uses this JTAG ID to ensure that is is using the correct SRM Object File (*.sof).

These registers are only good for Port 0 (PCIe Gen4 x16). They are blocked for the other Ports.

Table 128. JTAG Silicon ID Registers

Bits	Register Description	Default Value[4]	Access
[127:96]	JTAG Silicon ID DW3	Unique ID	RO
[95:64]	JTAG Silicon ID DW2	Unique ID	RO
[63:32]	JTAG Silicon ID DW1	Unique ID	RO
[31:0]	JTAG Silicon ID DW0	Unique ID	RO

A.3.5. User Configurable Device and Board ID (Offset 0x1C - 0x1D)

This register provides a user configurable device or board ID so that the user software can determine which .sof file to load into the device.

This register is only available for Port 0 (PCIe Gen4 x16). It is blocked for the other Ports.

Table 129. User Configurable Device and Board ID Register

Bits	Register Description	Default Value	Access
[15:0]	This register allows you to specify the ID of the `.sof` file to be loaded.	From configuration bits	RO

A.3.6. General Purpose Control and Status Register (Offset 0x30)

This register provides up to eight I/O pins each for Application Layer Control and Status requirements. This feature supports Partial Reconfiguration of the FPGA fabric. Partial Reconfiguration only requires one input pin and one output pin. The other seven I/Os make this interface extensible.

Table 130. General Purpose Control and Status Register

Bits	Register Description	Default Value	Access
[31:16]	Reserved.	N/A	RO
[15:8]	General Purpose Status. The Application Layer can read these status bits. These bits are only available for Port 0 (PCIe Gen4 x16). They are blocked for the other Ports.	0x00	RO
[7:0]	General Purpose Control. The Application Layer can write these control bits. These bits are only available for Port 0 (PCIe Gen4 x16). They are blocked for the other Ports.	0x00	RW

(4) Because the Silicon ID is a unique value, it does not have a global default value.

A.3.7. Uncorrectable Internal Error Status Register (Offset 0x34)

This register reports the status of the internally checked errors that are uncorrectable. When these specific errors are enabled by the `Uncorrectable Internal Error Mask` register, they are forwarded as Uncorrectable Internal Errors.

Note: This register is for debug only. Only use this register to observe behavior, not to drive logic custom logic.

Table 131. Uncorrectable Internal Error Status Register

Bits	Register Description	Default Value	Access
[31:13]	Reserved	0x0	RO
[12]	Debug Bus Interface (DBI) access error status from Config RAM block.	0x0	RW1CS
[11]	Uncorrectable ECC error from Config RAM block.	0x0	RW1C
[10:9]	Reserved	0x0	RO
[8]	RX Transaction Layer parity error reported by the IP core.	0x0	RW1CS
[7]	TX Transaction Layer parity error reported by the IP core.	0x0	RW1CS
[6]	Uncorrectable Internal Error reported by the FPGA.	0x0	RW1CS
[5]	`cvp_config_error_latched`: Configuration error detected in CvP mode is reported as an uncorrectable error. Set whenever `ssm_cvp_config_error` of the SSM Scratch CvP Status register bit[1] rises in CvP mode. This bit is only available for Port 0 (PCIe Gen4 x16), but not for the other Ports.	0x0	RW1CS
[4:0]	Reserved	0x0	RO

Note: The access code RW1CS represents Read Write 1 to Clear Sticky.

A.3.8. Uncorrectable Internal Error Mask Register (Offset 0x38)

This register controls which errors are forwarded as internal uncorrectable errors.

Table 132. Uncorrectable Internal Error Mask Register

Bits	Register Description	Default Value	Access
[31:13]	Reserved	0x0	RO
[12]	Mask for Debug Bus Interface (DBI) access error.	0x1	RWS

continued...

Bits	Register Description	Default Value	Access
[11]	Mask for Uncorrectable ECC error from Config RAM block.	0x1	RWS
[10:9]	Reserved	0x0	RO
[8]	Mask for RX Transaction Layer parity error reported by the IP core.	0x1	RWS
[7]	Mask for TX Transaction Layer parity error reported by the IP core.	0x1	RWS
[6]	Mask for Uncorrectable Internal error reported by the FPGA.	0x1	RWS
[5]	Mask for Configuration Error detected in CvP mode. This bit is only available for Port 0 (PCIe Gen4 x16), but not for the other Ports.	0x0	RWS
[4:0]	Reserved	0x0	RO

Note: The access code RWS stands for Read Write Sticky, meaning that the value is retained after a soft reset of the IP core.

A.3.9. Correctable Internal Error Status Register (Offset 0x3C)

The `Correctable Internal Error Status` register reports the status of the internally checked errors that are correctable. When these specific errors are enabled by the `Correctable Internal Error Mask` register, they are forwarded as correctable internal errors. This register is for debug only. Only use this register to observe behavior, not to drive custom logic

Table 133. Correctable Internal Error Status Register

Bits	Register Description	Default Value	Access
[31:12]	Reserved	0x0	RO
[11]	Correctable ECC error status from Config RAM.	0x0	RW1CS
[10:7]	Reserved	0x0	RO
[6]	Correctable Internal Error reported by the FPGA.	0x0	RW1CS
[5]	`cvp_config_error_latched`: Configuration error detected in CvP mode (to be reported as correctable) - Set whenever `cvp_config_error` rises while in CvP mode. This bit is only available for Port 0 (PCIe Gen4 x16), but not for the other Ports.	0x0	RW1CS
[4:0]	Reserved	0x0	RO

A.3.10. Correctable Internal Error Mask Register (Offset 0x40)

This register controls which errors are forwarded as internal correctable errors.

Table 134. Correctable Internal Error Mask Register

Bits	Register Description	Default Value	Access
[31:12]	Reserved	0x0	RO
[11]	Mask for Correctable ECC error status for Config RAM.	0x1	RWS
[10:7]	Reserved	0x0	RWS
[6]	Mask for Correctable Internal Error reported by the FPGA.	0x1	RWS
[5]	Mask for Configuration Error detected in CvP mode. This bit is only available for Port 0 (PCIe Gen4 x16), but not for the other Ports.	0x1	RWS
[4]	Reserved	0x1	RWS
[3:0]	Reserved	0x0	RWS

B. Implementation of ATS in Endpoint Mode

With the P-Tile Avalon-ST IP for PCIe:

- ATS messages/completions are sent and received through the Avalon Streaming interface.
- Address translation caches (ATC) need to be implemented in the user logic.
- User logic needs to have a separate ATC for each VF/PF that supports ATS.

Refer to the *Address Translation Services Revision 1.1* specification, section 4.1 *Page Request Message* for more details.

B.1. Sending Translated/Untranslated Requests

The function with an ATC can send Memory Read requests that contain either translated or untranslated addresses. If the Endpoint wants to receive the associated translated address to update the ATC, it will generate a Memory Read with the "Translation Request" field set.

The Endpoint will receive the translated address in the associated Completion with the ATS field's S/N/G/P/E/U/W/R values.

B.2. Sending a Page Request Message from the Endpoint (EP) to the Root Complex (RC)

The user application issues a Page Request Message while using the Avalon-ST interface to send the contents of the ATS message.

The RC will respond with PRG Response message(s).

The EP will update its ATC accordingly.

B.3. Invalidating Requests/Completions

Invalidation is done via the Message mechanism:

- When the EP receives an Invalidate request from the RP, it needs to clear the associated ATC.
- When the ATC is cleared, the user application generates a Completion message.

C. Packets Forwarded to the User Application in TLP Bypass Mode

In TLP Bypass mode, the P-Tile IP for PCIe forwards TLPs to the Avalon-ST RX interface except for malformed TLPs. The following tables describe how the IP handles each TLP type for upstream and downstream.

C.1. EP TLP Bypass Mode (Upstream)

Table 135. Packets Forwarded in EP TLP Bypass Mode

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
ASSERT/DEASSERT INTx	Local	Upstream	None	No
			Ecrc_err	No
			Malformed	No
VENDOR_MESSAGE_0 /1	Route_to_RC	Upstream	None	No (VENDOR0) Yes (VENDOR1)
			Poisoned	No (VENDOR0) Yes (VENDOR1)
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0 /1	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0 /1	Broadcast	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0 /1	Local	Both	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
PM_ACTIVE_STATE_NAK	Local	Downstream	None	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Ecrc_err	Yes
			Malformed	No
PM_PME	Route_to_RC	Upstream	None	No
			Ecrc_err	Yes
			Malformed	No
PME_TURN_OFF	Broadcast	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PME_TO_ACK	Gather	Upstream	None	No
			Ecrc_err	Yes
			Malformed	No
ERR_COR	Route_to_RC	Upstream	None	No
			Ecrc_err	Yes
			Malformed	No
ERR_NONFATAL	Route_to_RC	Upstream	None	No
			Ecrc_err	Yes
			Malformed	No
ERR_FATAL	Route_to_RC	Upstream	None	No
			Ecrc_err	Yes
			Malformed	No
UNLOCK	Broadcast	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
SET_SLOT_POWER_LIMIT	Local	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
LN_MESSAGE	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
LN_MESSAGE	Broadcast	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
DRS_MESSAGE	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
FRS_MESSAGE	Route_to_RC	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
HIERARCHY_ID_MSG	Broadcast	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_ON	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_BLINK	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_OFF	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_IND_ON	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_IND_BLINK	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_IND_OFF	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_BT_PRESS	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
LTR_MESSAGE	Local	Upstream	None	No
			Poisoned	No
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
OBFF_MESSAGE	Local	Downstream	None	No
			Poisoned	No
			Ecrc_err	Yes
			Malformed	No
PTM_REQUEST	Local	Upstream	None	No
			Ecrc_err	Yes
			Malformed	No
PTM_RESPONSE	Local	Downstream	None	No
			Poisoned	No
			Ecrc_err	Yes
			Malformed	No
PTM_RESPONSE_D	Local	Downstream	None	No
			Poisoned	No
			Ecrc_err	Yes
			Malformed	No
INVALIDATE_REQUEST	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
INVALIDATE_COMPLETION	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_WR_0	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_WR_1	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
CFG_RD_0	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_RD_1	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
IO_WR	Address	Downstream	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
IO_RD	Address	Downstream	None	Yes
			Addr_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
MEM_WR_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
MEM_RD_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
MEM_RD_LK_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
ATOMIC_FETCH_ADD_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
ATOMIC_SWAP_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
ATOMIC_CAS_32/64/128	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	32/64: No 128: No stimulus
CPL	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			LUT_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
			CA_status	Yes
			UR_status	Yes
			CRS_status	Yes
CPLD	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			LUT_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No

C.2. RC TLP Bypass Mode (Downstream)

Table 136. Packets Forwarded in RC TLP Bypass Mode

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
ASSERT/DEASSERT INTx	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0/1	Route_to_RC	Upstream	None	Yes
			Poisoned	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0/1	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0/1	Broadcast	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
VENDOR_MESSAGE_0/1	Local	Both	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
PM_ACTIVE_STATE_NAK	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PM_PME	Route_to_RC	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PME_TURN_OFF	Broadcast	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PME_TO_ACK	Gather	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
ERR_COR	Route_to_RC	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
ERR_NONFATAL	Route_to_RC	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
ERR_FATAL	Route_to_RC	Upstream	None	Yes
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
UNLOCK	Broadcast	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
SET_SLOT_POWER_LIMIT	Local	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
LN_MESSAGE	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
LN_MESSAGE	Broadcast	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
DRS_MESSAGE	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
FRS_MESSAGE	Route_to_RC	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
HIERARCHY_ID_MSG	Broadcast	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_ON	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_BLINK	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_OFF	Local	Downstream	None	Yes
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
IGNORED_MSG_IND_ON	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_IND_BLINK	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_IND_OFF	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
IGNORED_MSG_ATT_BT_PRESS	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
LTR_MESSAGE	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
OBFF_MESSAGE	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PTM_REQUEST	Local	Upstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PTM_RESPONSE	Local	Downstream	None	Yes
			Ecrc_err	Yes
			Malformed	No
PTM_RESPONSE_D	Local	Downstream	None	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
INVALIDATE_REQUEST	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
INVALIDATE_COMPLETION	Route_by_ID	Both	None	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_WR_0	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_WR_1	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_RD_0	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
CFG_RD_1	Route_by_ID	Downstream	None	Yes
			ID_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
IO_WR	Address	Downstream	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
IO_RD	Address	Downstream	None	Yes
			Addr_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
MEM_WR_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes

continued...

TLP Type	Routing	Direction	TLP Corruption	Forwarded to Avalon-ST Interface
			Malformed	No
MEM_RD_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
ATOMIC_FETCH_ADD_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
ATOMIC_SWAP_32/64	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
ATOMIC_CAS_32/64/128	Address	Both	None	Yes
			Addr_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No
CPL	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			LUT_mismatch	Yes
			Ecrc_err	Yes
			Malformed	No
			CA_status	Yes
			UR_status	Yes
			CRS_status	Yes
CPLD	Route_by_ID	Both	None	Yes
			ID_mismatch	Yes
			LUT_mismatch	Yes
			Poisoned	Yes
			Ecrc_err	Yes
			Malformed	No

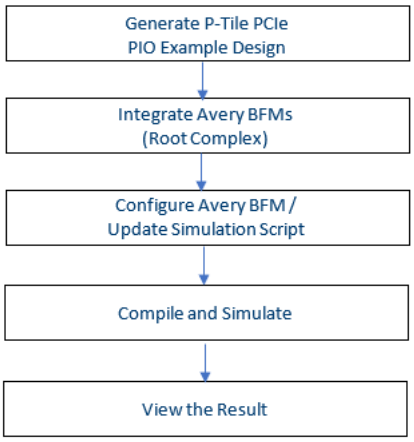
D. Using the Avery BFM for Intel P-Tile PCI Express Gen4 x16 Simulations

D.1. Overview

This appendix describes how to set up an Intel P-Tile PCIe Gen4 x16 Endpoint simulation using Avery BFMs for the Synopsys VCS simulator.

The Avery BFM simulation example described here is based on the PCIe Programmed I/O (PIO) example design generated from the Intel Quartus Prime PCIe IP GUI. Although the simulation flow and testbench setup leverage the Intel Quartus Prime example design testbench files, a similar flow and setup can be used for other PCIe system simulations with the P-Tile PCIe IP core.

Figure 87. Simulation Flow



Software Requirements

- Intel Quartus Prime version 19.4
- Intel P-Tile Avalon-MM/Avalon-ST PCIe IP version 1.1.0
- Avery BFM version 2.2b
- Synopsys VCS Simulator version O-2018.09-SP2-2

Simulation Script Files

The following table describes the files required for running simulations and the locations where they need to be. Contact your local Field Applications Engineer (FAE) to get a sample copy of these files.

You can use these files as-is for Gen4 x16 PIO simulations based on the P-Tile PCIe PIO example design.

Table 137. Simulation Script Files

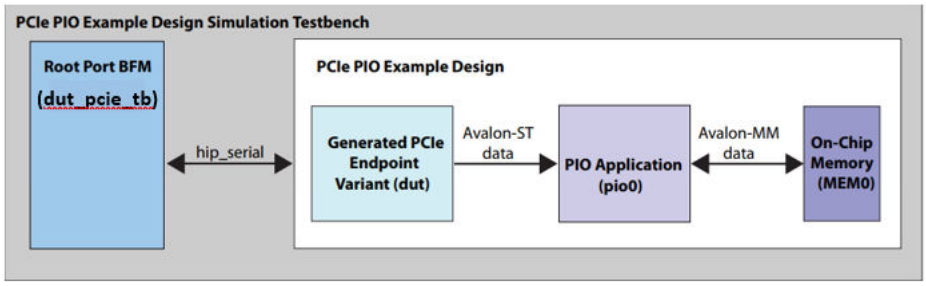
File Name	Description	Destination Folder
apci_top_rc.sv	Avery BFMs (Gen4 x16) and memory write & read traffic generation	<example design folder>/pcie_ed_tb/pcie_ed_tb/sim
pci_ed_tb.sv	Top-level testbench including Avery BFMs and PCIe PIO example design	<example design folder>/pcie_ed_tb/pcie_ed_tb/sim
vcs_files.tcl	List of simulation files	<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/common
avery_files_vcs.f	Synopsys VCS command file	<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/synopsys/vcs
vcs_setup.sh	VCS simulation script	<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/synopsys/vcs

D.2. Generate the PCIe PIO Example Design

D.2.1. Avalon-ST PCIe PIO Example Design

In the Avalon-ST PCIe IP GUI, configure the Avalon-ST PCIe IP as a Gen4 x16 Endpoint and generate the example design. This example design automatically creates the files for simulation. Following is the top-level testbench block diagram showing the default Root Port BFM and PIO example design.

Figure 88. PCIe PIO Example Design Simulation Testbench

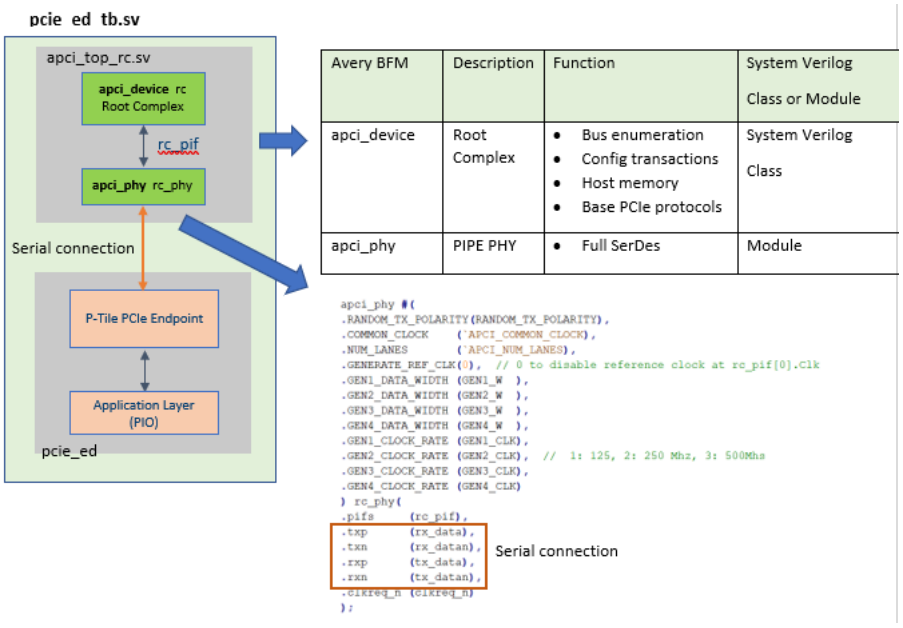


For details on the example design generation, refer to the *Quick Start Guide* chapter of the Intel FPGA P-Tile Avalon-ST IP for PCI Express User Guide.

D.3. Integrate Avery BFMs (Root Complex)

P-Tile supports serial mode connection only. The Avery Root Complex must instantiate a standalone BFM PHY to interface with the P-Tile PCIe Hard IP. The Avery BFM replaces the default Root Port BFM (dut_pcie_tb) in the P-Tile PCIe PIO example design testbench. The following diagram is a new top-level testbench, `pcie_ed_tb.sv`, that includes the Avery BFM.

Figure 89. Integrating the Avery BFM (Root Complex) and the PCIe Example Design



Avery BFM	Description	Function	System Verilog Class or Module
apci_device	Root Complex	• Bus enumeration • Config transactions • Host memory • Base PCIe protocols	System Verilog Class
apci_phy	PIPE PHY	• Full SerDes	Module

```
apci_phy #(
.RANDOM_TX_POLARITY(RANDOM_TX_POLARITY),
.COMMON_CLOCK     (`APCI_COMMON_CLOCK),
.NUM_LANES        (`APCI_NUM_LANES),
.GENERATE_REF_CLK(0),  // 0 to disable reference clock at rc_pif[0].Clk
.GEN1_DATA_WIDTH (GEN1_W  ),
.GEN2_DATA_WIDTH (GEN2_W  ),
.GEN3_DATA_WIDTH (GEN3_W  ),
.GEN4_DATA_WIDTH (GEN4_W  ),
.GEN1_CLOCK_RATE (GEN1_CLK),
.GEN2_CLOCK_RATE (GEN2_CLK),  //  1: 125, 2: 250 Mhz, 3: 500Mhz
.GEN3_CLOCK_RATE (GEN3_CLK),
.GEN4_CLOCK_RATE (GEN4_CLK)
) rc_phy(
.pifs      (rc_pif),
.txp       (rx_data),
.txn       (rx_datan),
.rxp       (tx_data),
.rxn       (tx_datan),
.clkreq_n (clkreq_n)
);
```

To include the Avery BFM, use the `pcie_ed_tb.sv` provided in the .zip file that you download or modify the example design top-level testbench as follows:

1. Open the example design top-level testbench file `<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/pcie_ed_tb.v`.
2. Import Avery BFM packages by adding the following lines below the `timescale statement:
   - import avery_pkg::*;
   - import apci_pkg::*;
   - import apci_pkg_test::*;
   - `include "apci_defines.svh"
3. Include the Avery BFM by adding the following line above the example design Root Port BFM (dut_pcie_tb).
   - `include "./../../apci_top_rc.sv"
4. Comment out the example design Root Port BFM (dut_pcie_tb).
5. Save the file and rename it to `pcie_ed_tb.sv`.
6. Open `<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/common/vcs_files.tcl`. Find and replace `pcie_ed_tb.v` with `pcie_ed_tb.sv`.

For details, refer to `apci_top_rc.sv`, `pcie_ed_tb.sv`, and `vcs_files.tcl` files available in the .zip file that you download.

D.4. Configure the Avery BFM and Update the Simulation Script

Configure the Avery BFM

In this example, the Avery BFM in the `apci_top_rc.v` file is configured to support Gen4 x16 simulations as shown below:

`define APCI_NUM_LANES 16 // default: 16 lanes

rc.cfg_info.speed_sup = 4 // default: Gen4 speed

Dumping waveforms into the VPD file is also enabled in `apci_top_rc.v` (see $vcdpluson() task). If you want to disable it, comment out +define+APCI_DUMP_VPD in the `avery_files_vcs.f` file.

Update the Simulation Script

Before you compile/simulate the design, update the `<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/synopsys/vcs/vcs_setup.sh` script file to include the Avery file and debug options (shown in bold) for the VCS command:

- vcs -lca **-f avery_files_vcs.f -debug_pp** -timescale=1ps/1ps …

Note: The QUARTUS_INSTALL_DIR in `vcs_setup.sh` points to the Quartus installation directory. If you are using the `vcs_setup.sh` provided in the downloaded .zip file, you need to update the path per your local Quartus installation.

For details, refer to `apci_top_rc.v`, `avery_vcs_files.f`, and `vcs_setup.sh` provided in the downloaded .zip file.

D.5. Compile and Simulate

You can run simulations in either batch mode or interactive mode.

Batch Mode

In batch mode, the VCS script compiles the design and runs the simulation until $finish(). Run the following command under the `<example design folder>/pcie_ed_tb/pcie_ed_tb/sim/synopsys/vcs` folder.

% sh ./vcs_setup.sh USER_DEFINED_SIM_OPTIONS=””

Setting USER_DEFINED_SIM_OPTIONS to an empty string overrides the default USER_DEFINED_SIM_OPTIONS and allows the simulation to run until $finish().

Interactive Mode

The VCS script compiles the design and generates the simv executable without starting the simulation. To run the compilation only, set SKIP_SIM to 1 as follows:

% sh ./vcs_setup.sh USER_DEFINED_SIM_OPTIONS=”” SKIP_SIM=1

After the simv executable is generated, run simv in DVE GUI mode:

% ./simv -gui

In the DVE GUI, add signals to view in the GUI, then start the simulation.

D.6. View the Results

The Avery BFM includes a memory transaction test that runs a sequence of 10 memory write/read combinations, where the test writes to a memory location and immediately reads back from the same location. For details on the test, refer to mem_tr_test class in apci_top_rc.sv. When the test passes, the following is displayed:

Figure 90. Simulation Results

```
AVY_INF: apci_test_log@218104.673ns : Pick one of the bars bar0(64bit, MEM, Prefetch, base 5b704490cc400000, len 10000)
AVY_INF: apci_test_log@218264.671ns : APCI_TRANS_mem#172a78 addr 5b704490cc40033c, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@218582.672ns : APCI_TRANS_mem#173480 addr 5b704490cc40033c, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@218740.671ns : APCI_TRANS_mem#174868 addr 5b704490cc4001fc, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@219066.672ns : APCI_TRANS_mem#17524f addr 5b704490cc4001fc, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@219226.671ns : APCI_TRANS_mem#1766b9 addr 5b704490cc4000e8, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@219550.672ns : APCI_TRANS_mem#1770c0 addr 5b704490cc4000e8, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@219710.671ns : APCI_TRANS_mem#17851b addr 5b704490cc400074, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@220030.672ns : APCI_TRANS_mem#178f34 addr 5b704490cc400074, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@220190.671ns : APCI_TRANS_mem#17a33f addr 5b704490cc4000f4, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@220516.672ns : APCI_TRANS_mem#17ad46 addr 5b704490cc4000f4, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@220676.671ns : APCI_TRANS_mem#17c1b0 addr 5b704490cc400054, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@220990.672ns : APCI_TRANS_mem#17cbb7 addr 5b704490cc400054, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@221148.671ns : APCI_TRANS_mem#17df5f addr 5b704490cc40023c, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@221466.672ns : APCI_TRANS_mem#17e946 addr 5b704490cc40023c, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@221626.671ns : APCI_TRANS_mem#17fd2e addr 5b704490cc40030c, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@221942.672ns : APCI_TRANS_mem#180735 addr 5b704490cc40030c, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@222102.671ns : APCI_TRANS_mem#181afd addr 5b704490cc400198, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@222416.672ns : APCI_TRANS_mem#182504 addr 5b704490cc400198, length 8, is_write 0, first_be f, last_be f:
AVY_INF: apci_test_log@222576.671ns : APCI_TRANS_mem#1838bd addr 5b704490cc4000c4, length 8, is_write 1, first_be f, last_be f:
AVY_INF: apci_test_log@222898.672ns : APCI_TRANS_mem#1842d6 addr 5b704490cc4000c4, length 8, is_write 0, first_be f, last_be f:

AVY_INF: apci_test_log@222898.672ns : ----   End testing of test_body (rc): Passed ----
AVY_INF: apci_test_log@222898.673ns : Test passed
AVY_INF: apci_test_log@222898.673ns : _________________ End of mem_tr_test _________________
```

In the simulation results above, is_write = 1 denotes a memory write, and is_write = 0 denotes a memory read.

When you want to view the VPD waveforms, invoke the DVE GUI:

% dve

In the DVE GUI, click File -> Open Database, and select `apci_top.vpd`.

To add waveforms, select a component (e.g. dut) in the Hierarchy pane, select signals in the Variable pane, and then add them to the Wave pane.

Figure 91. Sample DVE GUI Hierarchy and Variables Screen

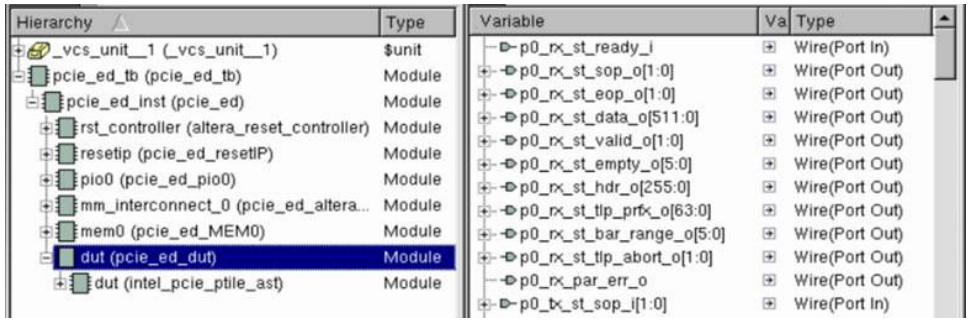
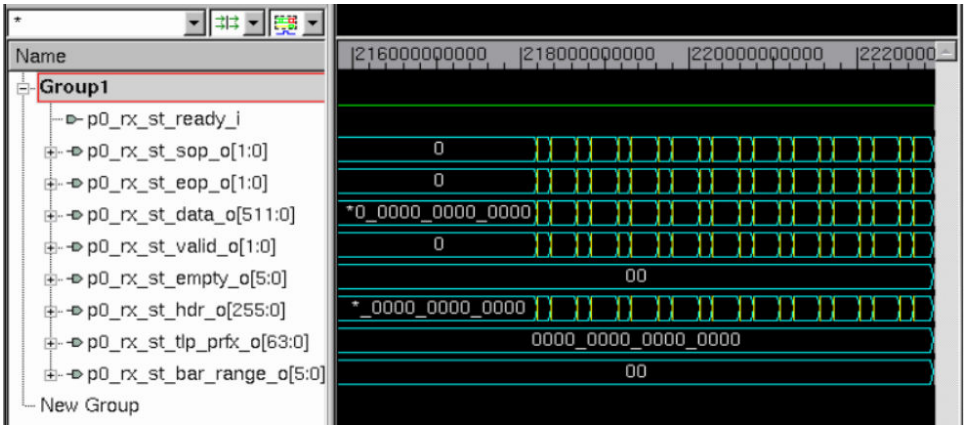


Figure 92. Sample Waveforms Screen



In addition, the Avery BFM enables dumping traffic into three text files to facilitate the debugging of the transaction layer, data link layer, and physical layer functions:

In `apci_top_rc.sv`:

initial begin

rc = new("rc", null, APCI_DEVICE_rc, 1);

…

rc.port_set_tracker(-1, "tl" , 1);

rc.port_set_tracker(-1, "dll", 1);

rc.port_set_tracker(-1, "phy", 1);

…

end

The three tracker files that the Avery BFM generates are:

- `tracker_tl_rc.txt` (transaction layer TLP dump)
- `tracker_dll_rc.txt` (data link layer DLLP dump)
- `tracker_phy_rc.txt` (physical layer Ordered Set dump)

Here is an example of TLP transactions captured in `tracker_tl_rc.txt`:

Figure 93. Excerpt from `tracker_tl_rc.txt`

```
==> @183437.673ns  MSGD#ea64c (set_slot_power_limit)
    | fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length      |
    | 011b|  10100b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001 _______|
    |________ req_id: 0000 _________|___ tag: 7f ___| msg_code: 50 _|
    74000001 00007f50 00000000 00000000
     00000000

==> @183437.674ns  CFGRD0#ea717 (bdf_1_0_0, offset 0)(req_id 0000, tag 000) (APCI_TRANS_cfg#ea716)
    | fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length      |
    | 000b|  00100b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001 _______|
    |________ req_id: 0000 _________|___ tag: 00 ___|lbe: 0 |fbe: f |
    |    bdf.bus  |bdf.dev|bdf.func|rsvd20|reg_no.ext|reg_no.low|rsv|
    |______ 01 ___|__ 00 _|___ 0 __|_ 0 __|___ 0 ____|___ 00 ___| 0 |
    04000001 0000000f 01000000
    ((Probe func_1_0_0))

       <== @183564.673ns  CPLD#eaf61 (CFGRD0#ea717, SUCCESS)(req_id 0000, tag 000)
           | fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length      |
           | 010b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001 _______|
           |_____ cpl_id: 0000 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 004 __|
           |_____ req_id: 0000 ___|___ tag: 00 ___|rsvd20: 0| low_addr: 00 |
           4a000001 00000004 00000000
            00001172
```

E. Document Revision History

E.1. Document Revision History for the Intel FPGA P-Tile Avalon Streaming (Avalon-ST) IP for PCI Express User Guide

Document Version	Intel Quartus Prime Version	IP Version	Changes
2020.01.16	19.4	1.1.0	Added information about the availability of the CvP Init and CvP Update features in Intel Stratix 10 DX and Intel Agilex devices to the *Features* section. Added the `rx_st_tlp_abort_o[1:0]` signals to the *Avalon-ST RX Interface* section. Removed the `app_ready_entr_l23_i` signal from the *Power Management Interface* section.
2019.12.16	19.4	1.1.0	Added parameters in Intel Quartus Prime to control PASID and LTR. Added MSI extended data support.
2019.11.04	19.3	1.0.0	Added resource utilization numbers for the PIO design example in Intel Stratix 10 DX devices. Added a step to select Intel Stratix 10 DX devices in the *Generating the Design Example* section.
2019.10.23	19.3	1.0.0	Added the description and usage instructions for the P-Tile Debug Toolkit. Added an Appendix chapter on how to use the Avery BFM to run Gen4 x16 simulations.
2019.07.19	19.2	1.0.0	Added features such as SR-IOV support and VirtIO support.
2019.05.03	19.1.1		Initial release.