

Versal ACAP CPM Mode for PCI Express v3.0

Product Guide

Vivado Design Suite

PG346 (v3.0) November 9, 2021

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this [link](#) for more information.



Table of Contents

Chapter 1: Overview	4
Navigating Content by Design Process.....	4
Introduction to the CPM4.....	5
Use Modes.....	10
Limitations.....	16
Licensing and Ordering.....	17
Chapter 2: Product Specification	18
Minimum Device Requirements.....	18
Port Descriptions.....	18
Register Space.....	80
Chapter 3: Designing with the Core	82
Clocking.....	82
Resets.....	84
Tandem Configuration.....	84
Chapter 4: Design Flow Steps	92
Customizing and Generating the CIPS IP Core.....	92
Appendix A: GT Selection and Pin Planning for CPM4	120
CPM4 GT Selection.....	121
CPM4 Additional Considerations.....	123
GT Locations.....	124
Appendix B: Debugging	128
Finding Help on Xilinx.com.....	128
PCIe Link Debug Enablement.....	129
Appendix C: Migrating	136
Ports.....	136
GT Locations.....	138

Clocking.....	138
Reset.....	139
Features.....	139
Attributes.....	141
Appendix D: Additional Resources and Legal Notices.....	142
Xilinx Resources.....	142
Documentation Navigator and Design Hubs.....	142
References.....	143
Revision History.....	143
Please Read: Important Legal Notices.....	144

Overview

Navigating Content by Design Process

Xilinx[®] documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal[®] ACAP design process [Design Hubs](#) and the [Design Flow Assistant](#) materials can be found on the [Xilinx.com](#) website. This document covers the following design processes:

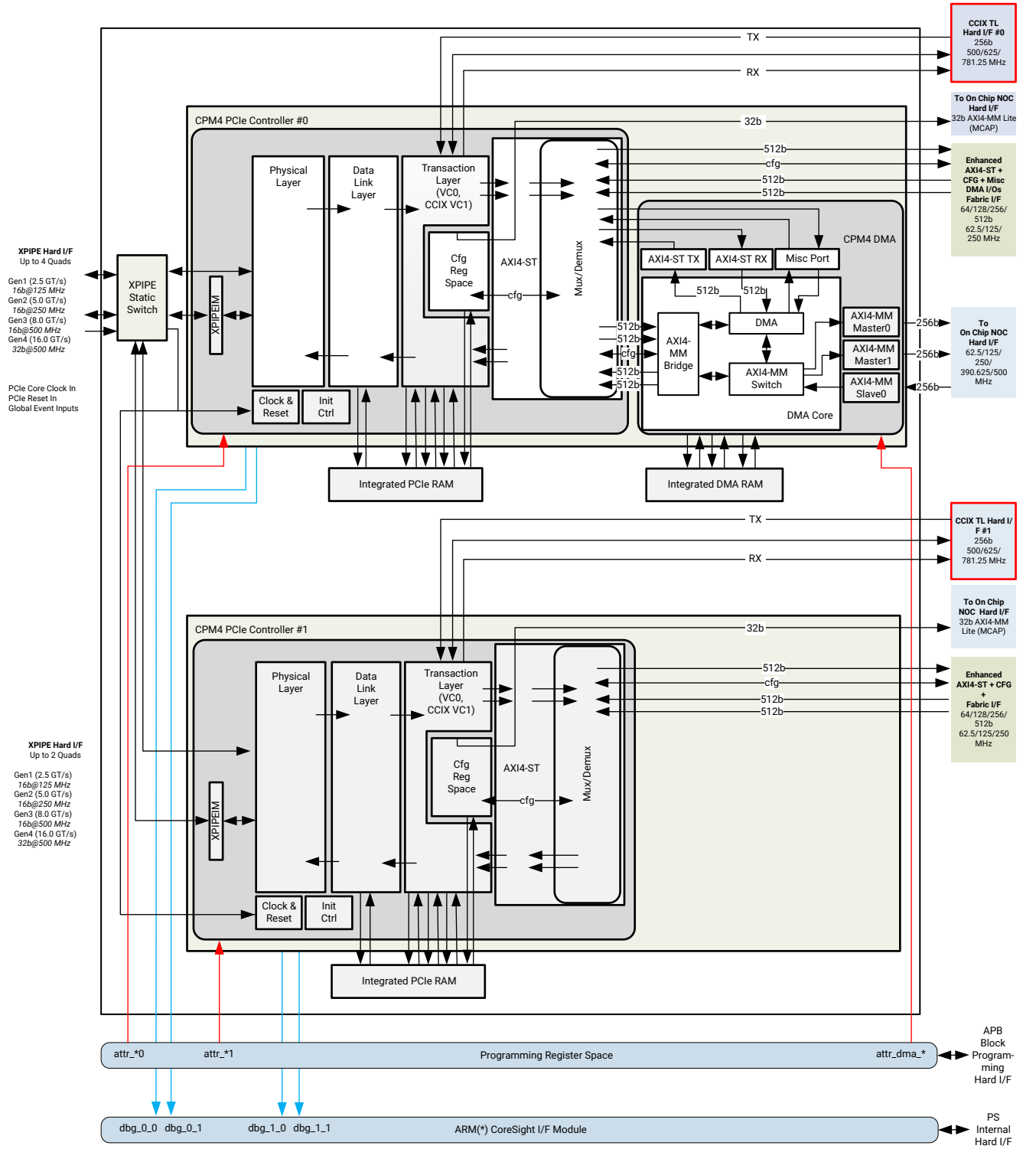
- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include:
 - [Introduction to the CPM4](#)
 - [Use Modes](#)
- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. The topic in this document that applies to this design process include:
 - [Register Space](#)
- **Host Software Development:** Developing the application code, accelerator development, including library, XRT, and Graph API use. The topic in this document that applies to this design process include:
 - [Register Space](#)
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado[®] timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
 - [Chapter 4: Design Flow Steps](#)
 - [Appendix A: GT Selection and Pin Planning for CPM4](#)
 - [PCIe Link Debug Enablement](#)

Introduction to the CPM4

The integrated block for PCIe® Rev. 4.0 with DMA and CCIX Rev. 1.0 (CPM4) consists of two PCIe® controllers, DMA features, CCIX features, and network on chip (NoC) integration. The Versal® ACAP CPM Mode for PCI Express enables direct access to the two high-performance, independently customizable PCIe controllers. The CPM4 uses up to 16 Versal device GTY channels over the XPIPE. Application designs can also interface to the CPM4 with soft logic and clocking resources in the programmable logic. All feature references are applicable to both instances of CPM4 PCIe controllers, with the following exceptions:

- CPM4 PCIe Controller 0 supports up to x16 operation, and CPM4 PCIe Controller 1 supports up to x8 operation.
- CPM4 PCIe Controller 1 with up to x8 support is available only when CPM4 PCIe Controller 0 is configured with 8 lanes or fewer.
- The CPM4 DMA features are supported only with CPM4 PCIe Controller 0. For more information about CPM4 DMA features, see the *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide* ([PG347](#)).

Figure 1: CPM4 Sub-Block for PCIe Function (CPM4 PCIe)



The CPM4 PCIe controllers are designed to the PCI Express Base Specification Revision 4.0 and support the Gen4 data rate (16 GT/s per lane). They also support the Gen1 (2.5 GT/s per lane), Gen2 (5 GT/s per lane) and Gen3 (8 GT/s per lane) data rates, and can interoperate with components that are compliant with all versions of the PCI Express Base Specification.

The CPM4 PCIe controllers are available through the Vivado IP catalog in the Vivado Integrated Design Environment (IDE). The combination of the CPM4 PCIe controllers, the GTY, and clocking implement all layers of the PCI Express protocol, and the configuration space and controller.

Protocol Layers

The layers of the protocol are the AXI4-Stream layer, the transaction layer, the data link layer and the physical layer, and they are described below.

AXI4-Stream Layer

The AXI4-Stream layer implements Xilinx-specific requirements. In the transmit or outbound direction, the AXI4 layer interfaces the transaction layer with two AXI4-Stream interfaces. In the receive or inbound direction, the transaction layer output is forwarded to two AXI4-Stream interfaces. Application designs can attach to the AXI4-Stream interfaces, exchange information with the Versal® ACAP CPM Mode for PCI Express encoded as a Xilinx-specific streaming protocol implementation, and run on top of the industry standard AXI4-Stream interface. The CPM4 PCIe controllers support management of up to 256 (extended tag) or 768 (10 bit Tag) outstanding customer initiated read requests, as part of the streaming protocol. The AXI4-Stream layer supports:

- Reception and transmission of address translation services (ATS) invalid requests, ATS invalid completions, ATS page requests and ATS PRG response message TLPs, which enable ATS to be implemented in the fabric logic.
- AXI4-Stream interface widths of 64 bits, 128 bits, 256 bits and 512 bits.

Transaction Layer

The transaction layer is the upper layer of the PCI Express architecture, and its primary function is to accept, buffer, and forward transaction layer packets (TLPs). TLPs communicate information with the use of memory, I/O, configuration, and message transactions. To maximize the efficiency of communication between devices, the transaction layer enforces PCI-compliant transaction ordering rules and supports relaxed ordering (RO) of received transactions. The transaction layer also manages TLP buffer space through credit-based flow control. The transaction layer implements built-in tag management for transmitted non-posted transactions. It also implements cut-through forwarding of transactions in the transmit (or outbound) direction.

CCIX Transaction Layer

The Cache Coherent Interconnect for Accelerators (CCIX) transaction layer requirements are implemented by the optional virtual channel 1 (VC1) in the design. Note that VC1 storage is in addition to the PCI Express-compliant virtual channel 0 (VC0) storage. The CCIX transaction layer interfaces with the CCIX protocol layer is implemented externally to the PCIe ports over the CCIX transaction layer (ARM CXS) hard interface. For more information, see the *Versal ACAP CPM CCIX Architecture Manual (AM016)*.

Data Link Layer

The data link layer acts as an intermediate stage between the transaction layer and the physical layer. Its primary responsibility is to provide a reliable mechanism for the exchange of information between two components on a link. This includes data exchange (TLPs), error detection and recovery, initialization services and the generation and consumption of data link layer packets (DLLPs). DLLPs are used to transfer information between data link layers of two directly connected components on the link. DLLPs convey information, such as power management, flow control, and TLP acknowledgments. The data link layer supports 32 kilobyte replay buffers and the feature DLLP.

Physical Layer

The physical layer interfaces the data link layer with signaling technology for link data interchange, and is subdivided into the logical sub-block and the electrical sub-block.

- The logical sub-block frames and de-frames TLPs and DLLPs. It also implements the link training and status state machine (LTSSM), which handles link initialization, training, and maintenance. Scrambling and descrambling of data (for Gen1/Gen2/Gen3/Gen4 operation) is also performed in this sub-block.
- The electrical sub-block defines the input and output buffer characteristics that interface the device to the PCIe link. The physical layer also supports lane reversal (for multi-lane designs) and lane polarity inversion, as required by the *PCI Express Base Specification 4.0* (<https://www.pcisig.com/specifications>).

Data exchange with the other components on the link occurs over the serial lines of one or more gigabit transceivers (GTs), which expose parallel interfaces at lower clock frequencies to the PCIe controller. For Gen1, Gen2, Gen3 and Gen4 operation, the physical layer is up-configuration capable in the downstream port mode only.

Standards

The CPM4 block adheres to the following standards:

- PCI Express Base Specification 4.0 Version 1.0, and Errata updates (available at <http://pcisig.com/specifications>).

- Cache Coherent Interconnect for Accelerators (CCIX) Transport Specification 1.0 (available at <http://www.ccixconsortium.com>).

Features

- Support for the following PCI Express architecture components:
 - PCI Express Endpoint, Legacy Endpoint
 - Root Port
 - Switch Upstream and Downstream Ports
- 2.5 GT/s, 5.0 GT/s and 8.0 GT/s line rates with x1, x2, x4, x8, and x16 lane operation.
- 16.0 GT/s line rate with x1, x2, x4, x8 lane operation.
- CCIX support in PCI Express and EDR PHY Modes
 - PCI Express support for Gen4x4, and Gen4x8
- Advanced Error Reporting (AER) and End-to-End CRC (ECRC)
- Two PCI Express virtual channels
 - One PCI Express compliant virtual channel, eight traffic classes
 - One CCIX compliant virtual channel
- Support for multiple functions and Single-Root IO Virtualization (SR-IOV)
 - Up to 4 physical functions
 - Up to 252 virtual functions
- Built-in lane reversal and receiver lane-lane de-skew
- 3 x 64-bit or 6 x 32-bit Base Address Registers (BARs) that are fully configurable
 - Expansion ROM BAR supported
- All Interrupt types are supported:
 - INTx
 - 32 multi-vector MSI capability
 - MSI-X capability with up to 2048 vectors with optional built-in vector tables
- Features that enable high-performance applications include:
 - AXI4-Stream TLP Straddle on Requester Completion Interface
 - Address Translation Services (ATS) and Page Request Interface (PRI) Messaging
 - Atomic Operation Transactions Support

- Transaction Tag Scaling as Completer
- Flow Control Scaling

Use Modes

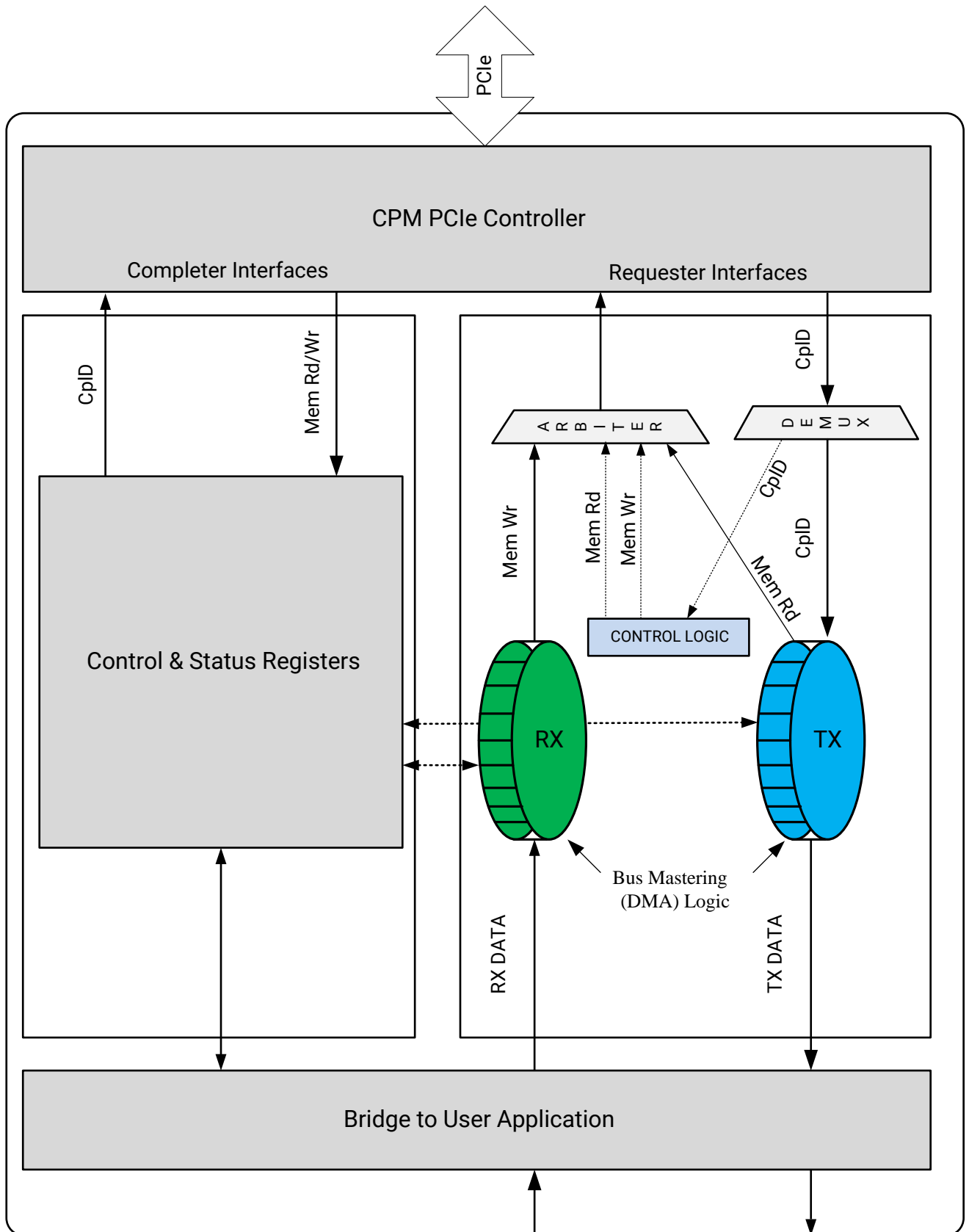
All design use modes support Endpoint, Legacy Endpoint, and Root Port configurations.

PCI Express Endpoint Use Modes

Illustrative Example of Basic Bus Mastering Endpoint

By far the most common use of the Versal® ACAP CPM Mode for PCI Express is to construct a bus mastering Endpoint using a CPM PCIe controller. This use model is applicable to most applications that interface the Endpoint port on the ACAP (on an add-in card) to a root complex or that switch downstream port through a PCI Express connector. The following figure shows a block diagram of the bus mastering Endpoint use case.

Figure 2: Basic PCI Express Bus Mastering Endpoint Use Case

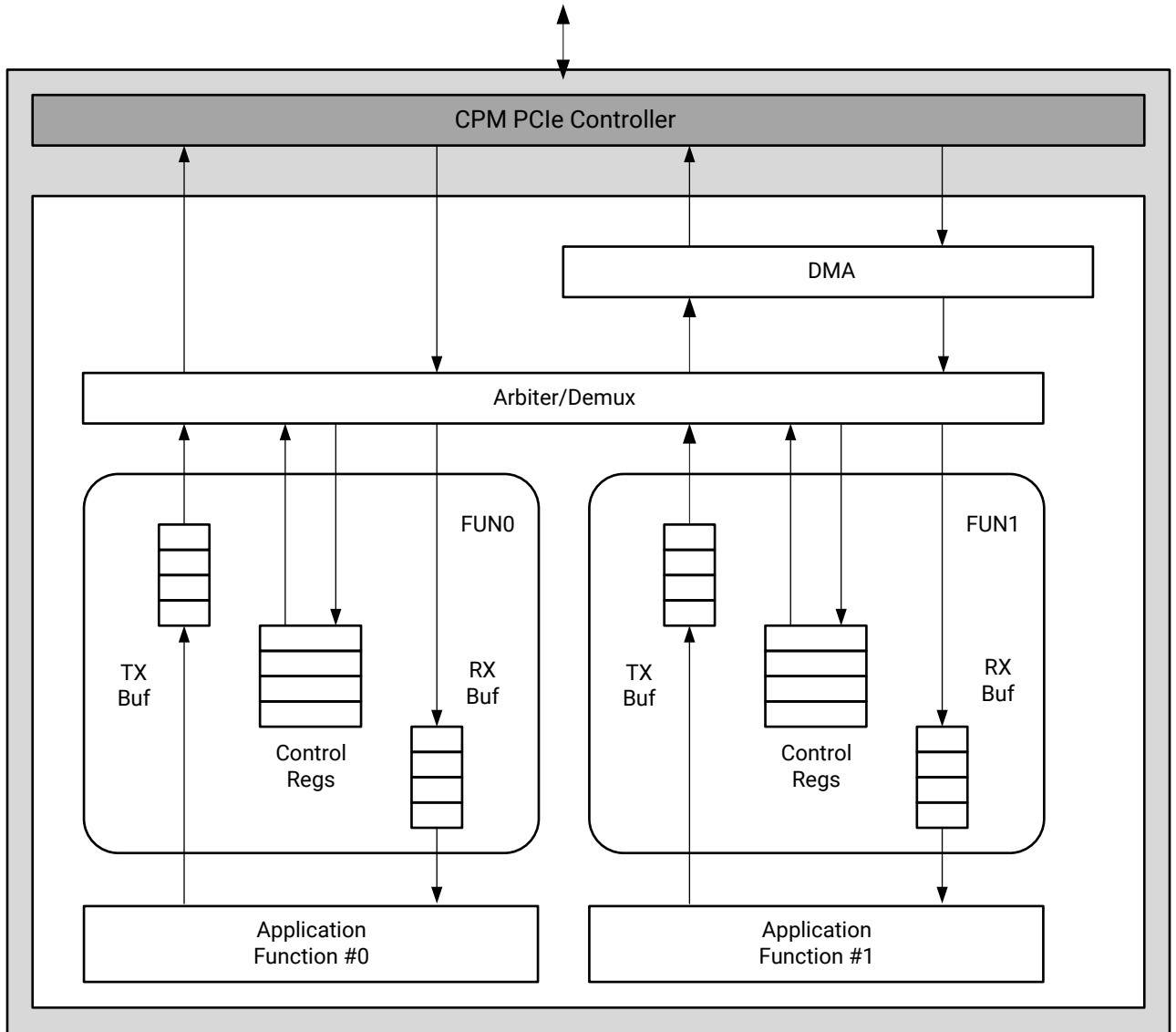


X22666-071620

PCI Express Two Function Endpoint

The following figure shows the architecture of a two-function Endpoint design. The CPM PCIe Controller is configured to enable two built-in function configuration spaces. This use case enables the application device driver to access and control two distinct applications independently. The user logic implements the DMA, control registers and applications.

Figure 3: Illustrative Example of Two Function Endpoint Use Case

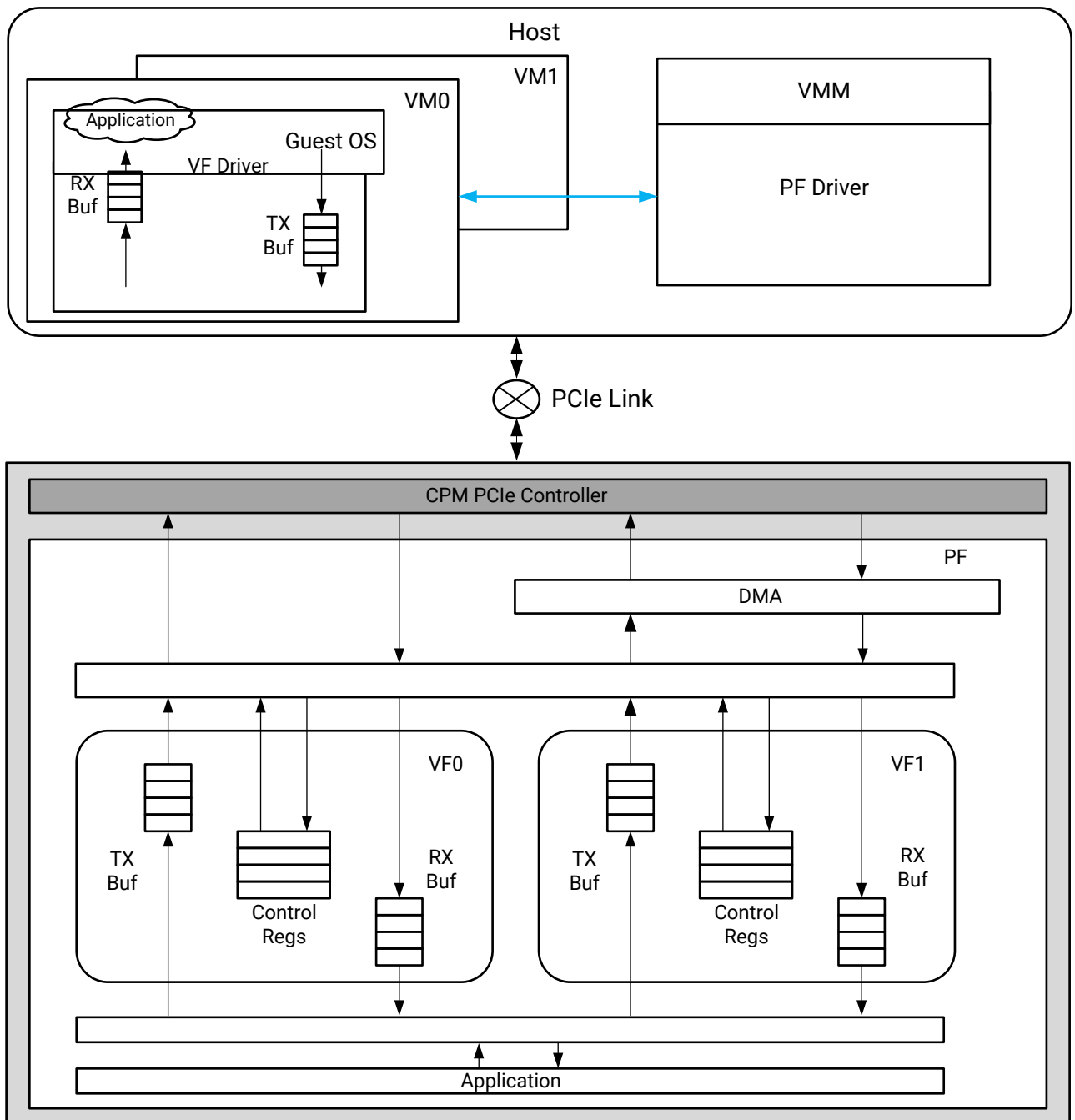


X22667-071620

PCI Express Endpoint with SR-IOV

The following figure shows the CPM PCIe Controller configured as a SR-IOV capable Endpoint, interfacing with the user design. This use case addresses requirements for up to four physical and 252 virtual functions, and minimizes the soft logic requirement to implement an SR-IOV Endpoint.

Figure 4: Illustrative Example of Endpoint with SR-IOV Use Case



X22668-071620

PCI Express Endpoint with AXI4 Memory Mapped Interface

This use case describes a PCI Express Endpoint functional unit that implements AXI4 Memory Mapped (AXI-MM) interfaces. This functional block implements a soft logic bridge between the native AXI4-Stream interface on the CPM PCIe controller and AXI4 Memory Mapped interconnect.

PCI Express Endpoint Using Tandem PROM

This use case addresses the ability to configure the ACAP in two stages and bring up the PCI Express protocol in less than 100 ms after power to the ACAP is stable. This is accomplished through a staged configuration flow.

PCI Express Endpoint Using Tandem PCIe

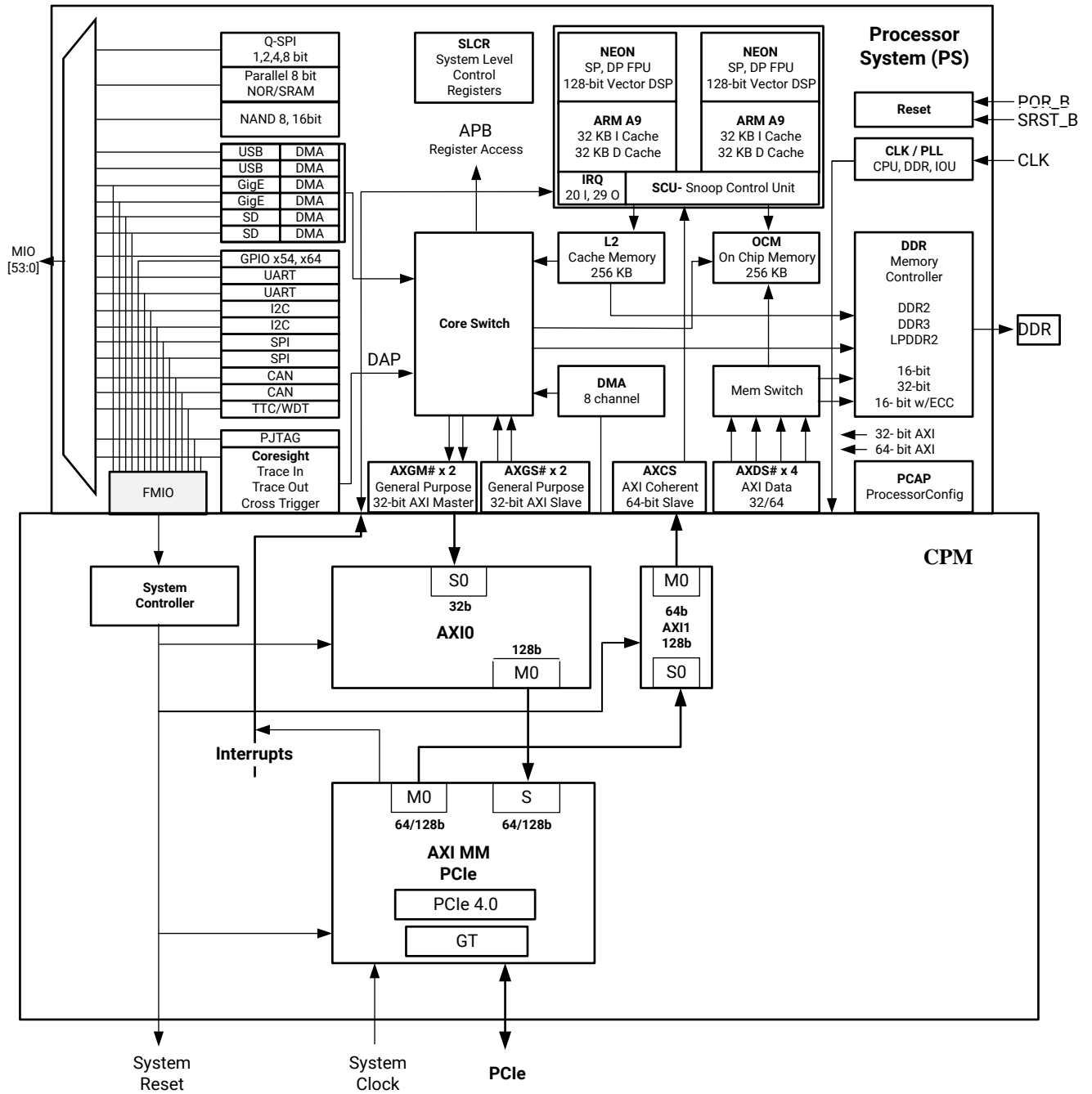
This use case addresses the ability to initially load fully configurable PCI Express protocol solution from a small external ROM, so as to meet the 100 ms configuration requirement. A PCIe link is formed with a Root Complex or Switch component, which is subsequently used to download the design that configures the rest of the ACAP. In this case the PCIe link is used by the user application. This is accomplished through a staged configuration flow.

PCI Express Root Port Use Mode

Basic PCI Express Root Complex Use Case

The following figure shows a PCI Express Root Complex in the simplest form consisting of a PCI Express Root Port to an AXI4 memory mapped bridge interfaced with the interconnect. The interconnect consists of an Arm[®]-based processor system (PS) containing most of the critical blocks such as CPU, memory controller and other important peripherals. One of the goals of this use case is to minimize ACAP soft logic requirements.

Figure 5: Basic PCI Express Root Complex Use Case



X22670-121420

Limitations

Speed Change Related Issue

- **Description:** Repeated speed changes can result in the link not coming up to the intended targeted speed.
- **Workaround:** A follow-on attempt should bring the link back. In extremely rare scenarios a full reboot might be required.

Link Autonomous Bandwidth Status (LABS) Bit

- **Description:** As a Root Complex when performing the link width/rate changes, the link width change works as expected. However, the PCIe protocol requires a LABS bit which is not getting set after the link width/rate change.

Note: This is an informational bit and does not impact actual functionality.

- **Workaround:** Software / Application to ignore LABS bit as this is an informational bit and does not impact functionality.

Power Management - ASPM L1/L0s/PM D3

- **Description:**
 1. Enabling ASPM L0s / ASPM L1 could show correctable errors being reported on link by both Link partners (i.e., replay timer timeout, replay timer rollover, receiver error).
 2. PCIe Endpoint device might also log errors when Configuration PM D3 transition request comes in during non-quiesced traffic mode.
- **Workaround:**
 1. It is recommended that the application disables correctable error reporting or ignores correctable errors reported in event of link transitioned to ASPM L0s / ASPM L1.
 2. For transition to D3Hot, software needs to make sure that the link is quiesced. To ensure Memory Write packets are finished, issue a Memory Read request to the same location. When the completion packet is received, it indicates that the link is quiesced and PM D3 request can be issued.

ECAM Access

- **Description:** Repeated Link Rate changes might also result in ECAM access becoming unresponsive, applicable to both Root Port and End Point.
 - In EP mode, the host may report Machine Check 0x12 Vector Trap if attempt is made to access multiple EP ECAM / Bridge Registers in this scenario.

- When PM D3 is also enabled and when the Pre-Read is done before the PM D3 sequence is targeted to the EP ECAM space, the host might receive completion timeout for this read due to this scenario.
- **Workaround:** Waiting for some time (in order of tens of milliseconds) after link rate before attempting any EP ECAM / Bridge access may help, recommendation for Pre-Read before PM D3 sequence is to target any valid EP address except EP ECAM space. In scenarios where the transaction does not complete, a full reboot would be required.

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the [Xilinx End User License](#).

For more information about this Versal® ACAP CPM Mode for PCIe, visit the [Versal® ACAP CPM Mode for PCIe product web page](#).

Information about other Xilinx® LogiCORE™ IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your [local Xilinx sales representative](#).

Product Specification

Minimum Device Requirements

CPM4

CPM4 Gen4x16 configurations can only be enabled when using DMA mode, as described in *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide (PG347)*. PCIe® controller-only mode does not support Gen4x16. The support is limited to Gen4x8.

Table 1: CPM4 Controller-Only Maximum Configurations (Versal Prime, Versal AI Core, Versal AI Edge)

Speed Grade	-1	-1	-2	-2	-2	-3
Voltage Grade	L (0.70V)	M (0.80V)	L (0.70V)	M (0.80V)	H (0.88V)	H (0.88V)
Gen1 (2.5 GT/s per lane)	x16	x16	x16	x16	x16	x16
Gen2 (5 GT/s per lane)	x16	x16	x16	x16	x16	x16
Gen3 (8 GT/s per lane)	x16	x16	x16	x16	x16	x16
Gen4 (16 GT/s per lane)	x8	x8	x8	x8	x8 ¹	x8 ¹

Notes:

1. x16 configuration is available using CPM4 in DMA mode. For details, see *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide (PG347)*.

Port Descriptions

Note: All the ports in the description field of the table excludes pcie0/pcie1 appended to the port names as the description is same for both. Please note that the pcie0* port name maps to pcie0* port names in the description field and pcie1* port name maps to pcie1* port names in the description field.

For example, the description for pcie0_m_axis_cq_tuser and pcie1_m_axis_cq_tuser states: *These signals are valid when m_axis_cq_tvalid is High.* Here, pcie0_m_axis_cq_tuser is valid when pcie0_m_axis_cq_tvalid is High, and pcie1_m_axis_cq_tuser is valid when pcie1_m_axis_cq_tvalid is High.

AXI4-Stream Core Interfaces

64/128/256-Bit Interfaces

In addition to status and control interfaces, the core has four required AXI4-Stream interfaces used to transfer and receive transactions, which are described in this section.

Completer Request Interface

The Completer Request (CQ) interface are the ports through which all received requests from the link are delivered to the user application. The following table defines the ports in the CQ interface of the core. In the Width column, DW denotes the configured data bus width (64, 128, or 256 bits).

Table 2: Completer Request Interface Port Descriptions

Port	I/O	Width	Description
pcie0_m_axis_cq_tdata pcie1_m_axis_cq_tdata	O	DW	Transmit Data from the CQ Interface. Only the lower 128 bits are used when the interface width is 128 bits, and only the lower 64 bits are used when the interface width is 64 bits. Bits [255:128] are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [255:64] are set permanently to 0 when the interface width is configured as 64 bits.
pcie0_m_axis_cq_tuser pcie1_m_axis_cq_tuser	O	108	CQ User Data. This set of signals contains sideband information for the transaction layer packets (TLP) being transferred. These signals are valid when pcie(n)_m_axis_cq_tvalid is High. Table 3: Sideband Signal Descriptions in pcie(n)_m_axis_cq_tuser describes the individual signals in this set.
pcie0_m_axis_cq_tlast pcie1_m_axis_cq_tlast	O	1	TLAST indication for CQ Data. The core asserts this signal in the last beat of a packet to indicate the end of the packet. When a TLP is transferred in a single beat, the core sets this signal in the first beat of the transfer.
pcie0_m_axis_cq_tkeep pcie1_m_axis_cq_tkeep	O	DW/32	TKEEP indication for CQ Data. The assertion of bit <i>i</i> of this bus during a transfer indicates to the user application that Dword <i>i</i> of the pcie(n)_m_axis_cq_tdata bus contains valid data. The core sets this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_m_axis_cq_tdata is set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (in both Dwords). This is true for both Dword-aligned and address-aligned modes of payload transfer. Bits [7:4] of this bus are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [7:2] are set permanently to 0 when the interface width is configured as 64 bits.

Table 2: Completer Request Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_m_axis_cq_tvalid pcie1_m_axis_cq_tvalid	O	1	CQ Data Valid. The core asserts this output whenever it is driving valid data on the pcie(n)_m_axis_cq_tdata bus. The core keeps the valid signal asserted during the transfer of a packet. The user application can pace the data transfer using the pcie(n)_m_axis_cq_tready signal.
pcie0_m_axis_cq_tready pcie1_m_axis_cq_tready	I	1	CQ Data Ready. Activation of this signal by the user logic indicates to the core that the user application is ready to accept data. Data is transferred across the interface when both pcie(n)_m_axis_cq_tvalid and pcie(n)_m_axis_cq_tready are asserted in the same cycle. If the user application deasserts the ready signal when pcie(n)_m_axis_cq_tvalid is High, the core maintains the data on the bus and keeps the valid signal asserted until the user application has asserted the ready signal.
pcie0_cq_np_req pcie1_cq_np_req	I	2	This input is used by the user application to request the delivery of a Non-Posted request. The core implements a credit-based flow control mechanism to control the delivery of Non-Posted requests across the interface, without blocking Posted TLPs. This input to the core controls an internal credit count. The credit count is updated in each clock cycle based on the setting of pcie(n)_cq_np_req[1:0] as follows: <ul style="list-style-type: none"> • 00: No change • 01: Increment by 1 • 10 or 11: Reserved (bit [1] only applicable in 512-bit interface) The credit count is decremented on the delivery of each Non-Posted request across the interface. The core temporarily stops delivering Non-Posted requests to the user logic when the credit count is zero. It continues to deliver any Posted TLPs received from the link even when the delivery of Non-Posted requests has been paused. The user application can either set pcie_cq_np_req[1:0] in each cycle based on the status of its Non-Posted request receive buffer, or can set it to 11 permanently if it does not need to exercise selective backpressure on Non-Posted requests. The setting of pcie(n)_cq_np_req[1:0] does not need to be aligned with the packet transfers on the completer request interface.
pcie0_cq_np_req_count pcie1_cq_np_req_count	O	6	This output provides the current value of the credit count maintained by the core for delivery of Non-Posted requests to the user logic. The core delivers a Non-Posted request across the completer request interface only when this credit count is non-zero. This counter saturates at a maximum limit of 32. Because of internal pipeline delays, there can be several cycles of delay between the user application providing credit on the pcie(n)_cq_np_req[1:0] inputs and the PCIe core updating the pcie_cq_np_req_count output in response. This count resets on user_reset and de-assertion of user_lnk_up.

When PASID_CAP_ON is enabled then `pcie(n)_m_axis_cq_tuser [107:85]` pins are shared with `cfg*` ports. The following table provides more information.

Table 3: Sideband Signal Descriptions in `pcie(n)_m_axis_cq_tuser`

Bit Index	Name	Width	Description
3:0	<code>first_be[3:0]</code>	4	<p>Byte enables for the first Dword of the payload. This field reflects the setting of the First_BE bits in the Transaction-Layer header of the TLP. For Memory Reads and I/O Reads, these four bits indicate the valid bytes to be read in the first Dword. For Memory Writes and I/O Writes, these bits indicate the valid bytes in the first Dword of the payload. For Atomic Operations and Messages with a payload, these bits are set to all 1s. This field is valid in the first beat of a packet, that is, when <code>sop</code> and <code>pcie(n)_m_axis_cq_tvalid</code> are both High.</p>
7:4	<code>last_be[3:0]</code>	4	<p>Byte enables for the last Dword. This field reflects the setting of the Last_BE bits in the Transaction-Layer header of the TLP. For Memory Reads, these four bits indicate the valid bytes to be read in the last Dword of the block of data. For Memory Writes, these bits indicate the valid bytes in the ending Dword of the payload. For Atomic Operations and Messages with a payload, these bits are set to all 1s. For Memory Reads and Writes of one DW transfers and zero length transfers, these bits should be 0s. This field is valid in the first beat of a packet, that is, when <code>sop</code> and <code>pcie(n)_m_axis_cq_tvalid</code> are both High.</p>
39:8	<code>byte_en[31:0]</code>	32	<p>The user logic can optionally use these byte enable bits to determine the valid bytes in the payload of a packet being transferred. The assertion of bit <i>i</i> of this bus during a transfer indicates that byte <i>i</i> of the <code>pcie(n)_m_axis_cq_tdata</code> bus contains a valid payload byte. This bit is not asserted for descriptor bytes. Although the byte enables can be generated by user logic from information in the request descriptor (address and length) as well as the settings of the <code>first_be</code> and <code>last_be</code> signals, you can use these signals directly instead of generating them from other interface signals. When the payload size is more than two Dwords (eight bytes), the one bit on this bus for the payload is always contiguous. When the payload size is two Dwords or less, the one bit can be non-contiguous. For the special case of a zero-length memory write transaction defined by the PCI Express specifications, the <code>byte_en</code> bits are all 0s when the associated one-DW payload is being transferred. Bits [31:16] of this bus are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [31:8] are set permanently to 0 when the interface width is configured as 64 bits.</p>
40	<code>sop</code>	1	<p>Start of packet. This signal is asserted by the core in the first beat of a packet to indicate the start of the packet. Using this signal is optional.</p>

Table 3: Sideband Signal Descriptions in `pcie(n)_m_axis_cq_tuser` (cont'd)

Bit Index	Name	Width	Description
41	discontinue	1	<p>This signal is asserted by the core in the last beat of a TLP, if it has detected an uncorrectable error while reading the TLP payload from its internal FIFO memory. The user application must discard the entire TLP when such an error is signaled by the core.</p> <p>This signal is never asserted when the TLP has no payload. It is asserted only in a cycle when <code>pcie(n)_m_axis_cq_tlast</code> is High.</p> <p>When the core is configured as an Endpoint, the error is also reported by the core to the Root Complex to which it is attached, using Advanced Error Reporting (AER).</p>
84:53	parity	32	<p>Bit <i>i</i> provides the odd parity computed for byte <i>i</i> of <code>pcie(n)_m_axis_cq_tdata</code>. Only the lower 16 bits are used when the interface width is 128 bits, and only the lower 8 bits are used when the interface width is 64 bits. Bits [31:16] are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [31:8] are set permanently to 0 when the interface width is configured as 64 bits.</p>
85	PASID TLP Valid	1	Indicates PASID TLP is valid.
105:86	PASID	20	Indicates PASID TLP prefix.
106	Execute Requested	1	Indicates Execute Requested to the user design.
107	Privileged Mode Requested	1	Indicates Privileged Mode Requested to the user design.

Completer Completion Interface

The Completer Completion (CC) interface are the ports through which completions generated by the user application responses to the completer requests are transmitted. You can process all Non-Posted transactions as split transactions. That is, the CC interface can continue to accept new requests on the requester completion interface while sending a completion for a request. The following table defines the ports in the CC interface of the core. In the Width column, DW denotes the configured data bus width (64, 128, or 256 bits).

Table 4: Completer Completion Interface Port Descriptions

Port	I/O	Width	Description
<code>pcie0_s_axis_cc_tdata</code> <code>pcie1_s_axis_cc_tdata</code>	I	DW	<p>Completer Completion Data bus.</p> <p>Completion data from the user application to the core. Only the lower 128 bits are used when the interface width is 128 bits, and only the lower 64 bits are used when the interface width is 64 bits.</p>
<code>pcie0_s_axis_cc_tuser</code> <code>pcie1_s_axis_cc_tuser</code>	I	33	<p>Completer Completion User Data.</p> <p>This set of signals contain sideband information for the TLP being transferred. These signals are valid when <code>pcie(n)_s_axis_cc_tvalid</code> is High.</p> <p>The following tables describe the individual signals in this set.</p>

Table 4: Completer Completion Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_s_axis_cc_tlast pcie1_s_axis_cc_tlast	I	1	TLAST indication for Completer Completion Data. The user application must assert this signal in the last cycle of a packet to indicate the end of the packet. When the TLP is transferred in a single beat, the user application must set this bit in the first cycle of the transfer.
pcie0_s_axis_cc_tkeep pcie1_s_axis_cc_tkeep	I	DW/32	TKEEP indication for Completer Completion Data. The assertion of bit <i>i</i> of this bus during a transfer indicates to the core that Dword <i>i</i> of the pcie(n)_s_axis_cc_tdata bus contains valid data. Set this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_s_axis_cc_tdata must be set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (both in Dwords). This is true for both Dword-aligned and address-aligned modes of payload transfer. Bits [7:4] of this bus are not used by the core when the interface width is configured as 128 bits, and bits [7:2] are not used when the interface width is configured as 64 bits.
pcie0_s_axis_cc_tvalid pcie1_s_axis_cc_tvalid	I	1	Completer Completion Data Valid. The user application must assert this output whenever it is driving valid data on the pcie(n)_s_axis_cc_tdata bus. The user application must keep the valid signal asserted during the transfer of a packet. The core paces the data transfer using the pcie(n)_s_axis_cc_tready signal.
pcie0_s_axis_cc_tready pcie1_s_axis_cc_tready	O	4	Completer Completion Data Ready. Activation of this signal by the core indicates that it is ready to accept data. Data is transferred across the interface when both pcie(n)_s_axis_cc_tvalid and pcie(n)_s_axis_cc_tready are asserted in the same cycle. If the core deasserts the ready signal when the valid signal is High, the user application must maintain the data on the bus and keep the valid signal asserted until the core has asserted the ready signal.

Table 5: Sideband Signal Descriptions in pcie(n)_s_axis_cc_tuser

Bit Index	Name	Width	Description
0	discontinue	1	<p>This signal can be asserted by the user application during a transfer if it has detected an error (such as an uncorrectable ECC error while reading the payload from memory) in the data being transferred and needs to abort the packet. The core nullifies the corresponding TLP on the link to avoid data corruption.</p> <p>The user application can assert this signal during any cycle during the transfer. It can either choose to terminate the packet prematurely in the cycle where the error was signaled, or can continue until all bytes of the payload are delivered to the core. In the latter case, the core treats the error as sticky for the following beats of the packet, even if the user application deasserts the discontinue signal before the end of the packet.</p> <p>The discontinue signal can be asserted only when pcie(n)_s_axis_cc_tvalid is High. The core samples this signal only when pcie(n)_s_axis_cc_tready is High. Thus, when asserted, it should not be deasserted until pcie(n)_s_axis_cc_tready is High.</p> <p>When the core is configured as an Endpoint, this error is also reported by the core to the Root Complex to which it is attached, using AER.</p>
32:1	parity	32	<p>Odd parity for the 256-bit data.</p> <p>When parity checking is enabled in the core, user logic must set bit i of this bus to the odd parity computed for byte i of pcie(n)_s_axis_cc_tdata. Only the lower 16 bits are used when the interface width is 128 bits, and only the lower 8 bits are used when the interface width is 64 bits.</p> <p>When an interface parity error is detected, it is recorded as an uncorrectable internal error and the packet is discarded. According to the Base Spec 6.2.9, an uncorrectable internal error is an error that occurs within a component that results in improper operation of the component. The only method of recovering from an uncorrectable internal error is a reset or hardware replacement.</p> <p>The parity bits can be permanently tied to 0 if parity check is not enabled in the core.</p>

Requester Request Interface

The Requester Request (RQ) interface consists of the ports through which the user application generates requests to remote PCIe® devices. The following table defines the ports in the RQ interface of the core. In the Width column, DW denotes the configured data bus width (64, 128, or 256 bits).

Table 6: Requester Request Interface Port Descriptions

Port	I/O	Width	Description
pcie0_s_axis_rq_tdata pcie1_s_axis_rq_tdata	I	DW	<p>Requester reQuest Data bus.</p> <p>This input contains the requester-side request data from the user application to the core. Only the lower 128 bits are used when the interface width is 128 bits, and only the lower 64 bits are used when the interface width is 64 bits.</p>

Table 6: Requester Request Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_s_axis_rq_tuser pcie1_s_axis_rq_tuser	I	85	Requester reQuest User Data. This set of signals contains sideband information for the TLP being transferred. These signals are valid when pcie(n)_s_axis_rq_tvalid is High. The following tables describes the individual signals in this set.
pcie0_s_axis_rq_tlast pcie1_s_axis_rq_tlast	I	1	TLAST Indication for Requester reQuest Data. The user application must assert this signal in the last cycle of a TLP to indicate the end of the packet. When the TLP is transferred in a single beat, the user application must set this bit in the first cycle of the transfer.
pcie0_s_axis_rq_tkeep pcie1_s_axis_rq_tkeep	I	DW/32	TKEEP Indication for Requester reQuest Data. The assertion of bit i of this bus during a transfer indicates to the core that Dword i of the pcie(n)_s_axis_rq_tdata bus contains valid data. The user application must set this bit to 1 contiguously for all Dwords, starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_s_axis_rq_tkeep must be set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (in both Dwords). This is true for both Dword-aligned and address-aligned modes of payload transfer. Bits [7:4] of this bus are not used by the core when the interface width is configured as 128 bits, and bits [7:2] are not used when the interface width is configured as 64 bits.
pcie0_s_axis_rq_tvalid pcie1_s_axis_rq_tvalid	I	1	Requester reQuest Data Valid. The user application must assert this output whenever it is driving valid data on the pcie(n)_s_axis_rq_tdata bus. The user application must keep the valid signal asserted during the transfer of a packet. The core paces the data transfer using the pcie(n)_s_axis_rq_tready signal.
pcie0_s_axis_rq_tready pcie1_s_axis_rq_tready	O	4	Requester reQuest Data Ready. Activation of this signal by the core indicates that it is ready to accept data. Data is transferred across the interface when both pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are asserted in the same cycle. If the core deasserts the ready signal when the valid signal is High, the user application must maintain the data on the bus and keep the valid signal asserted until the core has asserted the ready signal. You can assign all 4 bits to 1 or 0.
pcie0_rq_seq_num0 pcie1_rq_seq_num0	O	6	Requester reQuest TLP transmit sequence number. You can optionally use this output to track the progress of the request in the core transmit pipeline. To use this feature, provide a sequence number for each request on the seq_num[3:0] bus. The core outputs this sequence number on the pcie(n)_rq_seq_num0[3:0] output when the request TLP has reached a point in the pipeline where a Completion TLP from the user application cannot pass it. This mechanism enables you to maintain ordering between Completions sent to the CC interface of the core and Posted requests sent to the requester request interface. Data on the pcie(n)_rq_seq_num0[3:0] output is valid when pcie(n)_rq_seq_num_vld0 is High.

Table 6: Requester Request Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_rq_seq_num_vld0 pcie1_rq_seq_num_vld0	O	1	Requester reQuest TLP transmit sequence number valid. This output is asserted by the core for one cycle when it has placed valid data on pcie(n)_rq_seq_num0[3:0].
pcie0_rq_tag0 pcie0_rq_tag1 pcie1_rq_tag0 pcie1_rq_tag1	O	8	Requester reQuest Non-Posted tag. When tag management for Non-Posted requests is performed by the core (AXISTEN_IF_ENABLE_CLIENT_TAG is 0), this output is used by the core to communicate the allocated tag for each Non-Posted request received. The tag value on this bus is valid for one cycle when pcie(n)_rq_tag_vld0 is High. You must copy this tag and use it to associate the completion data with the pending request. There can be a delay of several cycles between the transfer of the request on the pcie(n)_s_axis_rq_tdata bus and the assertion of pcie(n)_rq_tag_vld0 by the core to provide the allocated tag for the request. Meanwhile, the user application can continue to send new requests. The tags for requests are communicated on this bus in FIFO order, so the user application can easily associate the tag value with the request it transferred.
pcie0_rq_tag_vld0 pcie0_rq_tag_vld1 pcie1_rq_tag_vld0 pcie1_rq_tag_vld1	O	1	Requester reQuest Non-Posted tag valid. The core asserts this output for one cycle when it has allocated a tag to an incoming Non-Posted request from the requester request interface and placed it on the pcie(n)_rq_tag0 output.

When PASID_CAP_ON is enabled then `pcie(n)_s_axis_rq_tuser [84:62]` pins are shared with `cfg*` ports. The following table provides more information.

 Table 7: Sideband Signal Descriptions in `pcie(n)_s_axis_rq_tuser`

Bit Index	Name	Width	Description
3:0	first_be[3:0]	4	Byte enables for the first Dword. This field must be set based on the desired value of the First_BE bits in the Transaction-Layer header of the request TLP. For Memory Reads, I/O Reads, and Configuration Reads, these four bits indicate the valid bytes to be read in the first Dword. For Memory Writes, I/O Writes, and Configuration Writes, these bits indicate the valid bytes in the first Dword of the payload. The core samples this field in the first beat of a packet, when <code>pcie(n)_s_axis_rq_tvalid</code> and <code>pcie(n)_s_axis_rq_tready</code> are both High.
7:4	last_be[3:0]	4	Byte enables for the last Dword. This field must be set based on the desired value of the Last_BE bits in the Transaction-Layer header of the TLP. For Memory Reads of two Dwords or more, these four bits indicate the valid bytes to be read in the last Dword of the block of data. For Memory Reads and Writes of one DW transfers and zero length transfers, these bits should be 0s. For Memory Writes of two Dwords or more, these bits indicate the valid bytes in the last Dword of the payload. The core samples this field in the first beat of a packet, when <code>pcie(n)_s_axis_rq_tvalid</code> and <code>pcie(n)_s_axis_rq_tready</code> are both High.

Table 7: Sideband Signal Descriptions in `pcie(n)_s_axis_rq_tuser` (cont'd)

Bit Index	Name	Width	Description
10:8	<code>addr_offset[2:0]</code>	3	<p>When the address-aligned mode is in use on this interface, the user application must provide the byte lane number where the payload data begins on the data bus, modulo 4, on this sideband bus. This enables the core to determine the alignment of the data block being transferred.</p> <p>The core samples this field in the first beat of a packet, when <code>pcie(n)_s_axis_rq_tvalid</code> and <code>pcie(n)_s_axis_rq_tready</code> are both High.</p> <p>When the requester request interface is configured in the Dword-alignment mode, this field must always be set to 0.</p> <p>In Root Port configuration, Configuration Packets must always be aligned to DW0, and therefore for this type of packets, this field must be set to 0 in both alignment modes.</p>
11	<code>discontinue</code>	1	<p>This signal can be asserted by the user application during a transfer if it has detected an error in the data being transferred and needs to abort the packet. The core nullifies the corresponding TLP on the link to avoid data corruption. You can assert this signal in any cycle during the transfer. You can either choose to terminate the packet prematurely in the cycle where the error was signaled, or continue until all bytes of the payload are delivered to the core. In the latter case, the core treats the error as sticky for the following beats of the packet, even if the user application deasserts the <code>discontinue</code> signal before the end of the packet.</p> <p>The <code>discontinue</code> signal can be asserted only when <code>pcie(n)_s_axis_rq_tvalid</code> is High. The core samples this signal only when <code>pcie(n)_s_axis_rq_tready</code> is High. Thus, when asserted, it should not be deasserted until <code>pcie(n)_s_axis_rq_tready</code> is High. <code>Discontinue</code> is not supported for Non-Posted TLPs. The user logic can assert this signal in any cycle except the first cycle during the transfer.</p> <p>When the core is configured as an Endpoint, this error is also reported by the core to the Root Complex to which it is attached, using Advanced Error Reporting (AER).</p>
27:24	<code>seq_num[3:0]</code>	4	<p>You can optionally supply a 4-bit sequence number in this field to keep track of the progress of the request in the core transmit pipeline. The core outputs this sequence number on its <code>pcie_rq_seq_num[3:0]</code> output when the request TLP has progressed to a point in the pipeline where a Completion TLP is not able to pass it.</p> <p>The core samples this field in the first beat of a packet, when <code>pcie(n)_s_axis_rq_tvalid</code> and <code>pcie(n)_s_axis_rq_tready</code> are both High.</p> <p>This input can be hardwired to 0 when the user application is not monitoring the <code>pcie_rq_seq_num[3:0]</code> output of the core.</p>

Table 7: Sideband Signal Descriptions in `pcie(n)_s_axis_rq_tuser` (cont'd)

Bit Index	Name	Width	Description
59:28	parity	32	<p>Odd parity for the 256-bit data.</p> <p>When parity checking is enabled in the core, the user logic must set bit <i>i</i> of this bus to the odd parity computed for byte <i>i</i> of <code>pcie(n)_s_axis_rq_tdata</code>. Only the lower 16 bits are used when the interface width is 128 bits, and only the lower 8 bits are used when the interface width is 64 bits.</p> <p>When an interface parity error is detected, it is recorded as an uncorrectable internal error and the packet is discarded. According to the Base Spec 6.2.9 (<i>PCI-SIG Specifications</i> (https://www.pcisig.com/specifications)), an uncorrectable internal error is an error that occurs within a component that results in improper operation of the component. The only method of recovering from an uncorrectable internal error is a reset or hardware replacement.</p> <p>The parity bits can be permanently tied to 0 if parity check is not enabled in the core.</p>
61:60	<code>seq_num[5:4]</code>	2	Extension of <code>seq_num</code> as in [27:24].
62	PASID TLP Valid	1	Indicates PASID TLP is valid.
82:63	PASID	20	Indicates PASID TLP prefix.
83	Execute Requested	1	Indicates Execute Requested.
84	Privileged Mode Requested	1	Indicates Privileged Mode Requested.

Requester Completion Interface

The Requester Completion (RC) interface are the ports through which the completions received from the link in response to your requests are presented to the user application. The following table defines the ports in the RC interface of the core. In the Width column, DW denotes the configured data bus width (64, 128, or 256 bits).

Table 8: Requester Completion Interface Port Descriptions

Port	I/O	Width	Description
<code>pcie0_m_axis_rc_tdata</code> <code>pcie1_m_axis_rc_tdata</code>	O	DW	<p>Requester Completion Data bus.</p> <p>Transmit data from the core requester completion interface to the user application. Only the lower 128 bits are used when the interface width is 128 bits, and only the lower 64 bits are used when the interface width is 64 bits.</p> <p>Bits [255:128] are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [255:64] are set permanently to 0 when the interface width is configured as 64 bits.</p>
<code>pcie0_m_axis_rc_tuser</code> <code>pcie1_m_axis_rc_tuser</code>	O	75	<p>Requester Completion User Data.</p> <p>This set of signals contains sideband information for the TLP being transferred. These signals are valid when <code>pcie(n)_m_axis_rc_tvalid</code> is High.</p> <p>The following table describes the individual signals in this set.</p>

Table 8: Requester Completion Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_m_axis_rc_tlast pcie1_m_axis_rc_tlast	O	1	<p>TLAST indication for Requester Completion Data.</p> <p>The core asserts this signal in the last beat of a packet to indicate the end of the packet. When a TLP is transferred in a single beat, the core sets this bit in the first beat of the transfer. This output is used only when the straddle option is disabled. When the straddle option is enabled (for the 256-bit interface), the core sets this output permanently to 0.</p>
pcie0_m_axis_rc_tkeep pcie1_m_axis_rc_tkeep	O	DW/32	<p>TKEEP indication for Requester Completion Data.</p> <p>The assertion of bit <i>i</i> of this bus during a transfer indicates that Dword <i>i</i> of the pcie(n)_m_axis_rc_tdata bus contains valid data. The core sets this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_m_axis_rc_tkeep sets to 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (both in Dwords). This is true for both Dword-aligned and address-aligned modes of payload transfer.</p> <p>Bits [7:4] of this bus are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [7:2] are set permanently to 0 when the interface width is configured as 64 bits.</p> <p>These outputs are permanently set to all 1s when the interface width is 256 bits and the straddle option is enabled. The user logic must use the signals in pcie(n)_m_axis_rc_tuser in that case to determine the start and end of Completion TLPs transferred over the interface.</p>
pcie0_m_axis_rc_tvalid pcie1_m_axis_rc_tvalid	O	1	<p>Requester Completion Data Valid.</p> <p>The core asserts this output whenever it is driving valid data on the pcie(n)_m_axis_rc_tdata bus. The core keeps the valid signal asserted during the transfer of a packet. The user application can pace the data transfer using the pcie(n)_m_axis_rc_tready signal.</p>
pcie0_m_axis_rc_tready pcie1_m_axis_rc_tready	I	1	<p>Requester Completion Data Ready.</p> <p>Activation of this signal by the user logic indicates to the core that the user application is ready to accept data. Data is transferred across the interface when both pcie(n)_m_axis_rc_tvalid and pcie(n)_m_axis_rc_tready are asserted in the same cycle.</p> <p>If the user application deasserts the ready signal when the valid signal is High, the core maintains the data on the bus and keeps the valid signal asserted until the user application has asserted the ready signal.</p>

Table 9: Sideband Signal Descriptions in `pcie(n)_m_axis_rc_tuser`

Bit Index	Name	Width	Description
31:0	<code>byte_en</code>	32	<p>The user logic can optionally use these byte enable bits to determine the valid bytes in the payload of a packet being transferred. The assertion of bit <i>i</i> of this bus during a transfer indicates that byte <i>i</i> of the <code>pcie(n)_m_axis_rc_tdata</code> bus contains a valid payload byte. This bit is not asserted for descriptor bytes.</p> <p>Although the byte enables can be generated by user logic from information in the request descriptor (address and length), the logic has the option to use these signals directly instead of generating them from other interface signals. The 1 bit in this bus for the payload of a TLP is always contiguous.</p> <p>Bits [31:16] of this bus are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [31:8] are set permanently to 0 when the interface width is configured as 64 bits. The byte enable bit is also set on completions received in response to zero length memory read requests.</p>
32	<code>is_sof_0</code>	1	<p>Start of a first Completion TLP.</p> <p>For 64-bit and 128-bit interfaces, and for the 256-bit interface with no straddling, <code>is_sof_0</code> is asserted by the core in the first beat of a packet to indicate the start of the TLP. On these interfaces, only a single TLP can be started in a data beat, and <code>is_sof_1</code> is permanently set to 0. Use of this signal is optional when the straddle option is not enabled.</p> <p>When the interface width is 256 bits and the straddle option is enabled, the core can straddle two Completion TLPs in the same beat. In this case, the Completion TLPs are not formatted as AXI4-Stream packets. The assertion of <code>is_sof_0</code> indicates a Completion TLP starting in the beat. The first byte of this Completion TLP is in byte lane 0 if the previous TLP ended before this beat, or in byte lane 16 if the previous TLP continues in this beat.</p>
33	<code>is_sof_1</code>	1	<p>This signal is used when the interface width is 256 bits and the straddle option is enabled, when the core can straddle two Completion TLPs in the same beat. The output is permanently set to 0 in all other cases.</p> <p>The assertion of <code>is_sof_1</code> indicates a second Completion TLP starting in the beat, with its first byte in byte lane 16. The core starts a second TLP at byte position 16 only if the previous TLP ended in one of the byte positions 0-15 in the same beat; that is, only if <code>is_eof_0[0]</code> is also set in the same beat.</p>
37:34	<code>is_eof_0[3:0]</code>	4	<p>End of a first Completion TLP and the offset of its last Dword.</p> <p>These outputs are used only when the interface width is 256 bits and the straddle option is enabled.</p> <p>The assertion of the bit <code>is_eof_0[0]</code> indicates the end of a first Completion TLP in the current beat. When this bit is set, the bits <code>is_eof_0[3:1]</code> provide the offset of the last Dword of this TLP.</p>

Table 9: Sideband Signal Descriptions in `pcie(n)_m_axis_rc_tuser` (cont'd)

Bit Index	Name	Width	Description
41:38	<code>is_eof_1[3:0]</code>	4	<p>End of a second Completion TLP and the offset of its last Dword.</p> <p>These outputs are used only when the interface width is 256 bits and the straddle option is enabled. The core can then straddle two Completion TLPs in the same beat. These outputs are reserved in all other cases.</p> <p>The assertion of <code>is_eof_1[0]</code> indicates a second TLP ending in the same beat. When bit 0 of <code>is_eof_1</code> is set, bits [3:1] provide the offset of the last Dword of the TLP ending in this beat. Because the second TLP can only end at a byte position in the range 27-31, <code>is_eof_1[3:1]</code> can only take one of two values (6 or 7).</p> <p>The offset for the last byte of the second TLP can be determined from the starting address and length of the TLP, or from the byte enable signals <code>byte_en[31:0]</code>.</p> <p>If <code>is_eof_1[0]</code> is High, the signals <code>is_eof_0[0]</code> and <code>is_sof_1</code> are also High in the same beat.</p>
42	<code>discontinue</code>	1	<p>This signal is asserted by the core in the last beat of a TLP, if it has detected an uncorrectable error while reading the TLP payload from its internal FIFO memory. The user application must discard the entire TLP when such an error is signaled by the core.</p> <p>This signal is never asserted when the TLP has no payload. It is asserted only in the last beat of the payload transfer; that is, when <code>is_eof_0[0]</code> is High.</p> <p>When the straddle option is enabled, the core does not start a second TLP if it has asserted <code>discontinue</code> in a beat.</p> <p>When the core is configured as an Endpoint, the error is also reported by the core to the Root Complex to which it is attached, using Advanced Error Reporting (AER).</p>
74:43	<code>parity</code>	32	<p>Odd parity for the 256-bit transmit data.</p> <p>Bit <i>i</i> provides the odd parity computed for byte <i>i</i> of <code>pcie(n)_m_axis_rc_tdata</code>. Only the lower 16 bits are used when the interface width is 128 bits, and only the lower 8 bits are used when the interface width is 64 bits. Bits [31:16] are set permanently to 0 by the core when the interface width is configured as 128 bits, and bits [31:8] are set permanently to 0 when the interface width is configured as 64 bits.</p>

512-bit Interfaces

This section provides the description for ports associated with the user interfaces of the core.

Completer Request Interface

Table 10: Completer Request Interface Port Descriptions (512-bit Interface)

Name	I/O	Width	Description
<code>pcie0_m_axis_cq_tdata</code> <code>pcie1_m_axis_cq_tdata</code>	O	512	Transmit data from the PCIe completer request interface to the user application.

Table 10: Completer Request Interface Port Descriptions (512-bit Interface) (cont'd)

Name	I/O	Width	Description
pcie0_m_axis_cq_tuser pcie1_m_axis_cq_tuser	O	229	This is a set of signals containing sideband information for the TLP being transferred. These signals are valid when pcie(n)_m_axis_cq_tvalid is High. The individual signals in this set are described in the following table.
pcie0_m_axis_cq_tlast pcie1_m_axis_cq_tlast	O	1	The core asserts this signal in the last beat of a packet to indicate the end of the packet. When a TLP is transferred in a single beat, the core sets this bit in the first beat of the transfer. This output is used only when the straddle option is disabled. When the straddle option is enabled, the core sets this output permanently to 0.
pcie0_m_axis_cq_tkeep pcie1_m_axis_cq_tkeep	O	16	The assertion of bit <i>i</i> of this bus during a transfer indicates to the user logic that Dword <i>i</i> of the pcie(n)_m_axis_cq_tdata bus contains valid data. The core sets this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_m_axis_cq_tdata is set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (both in Dwords). This is true for both Dword-aligned and 128b address-aligned modes of payload transfer. The tkeep bits are valid only when straddle is not enabled on the CQ interface. When straddle is enabled, the tkeep bits are permanently set to all 1s in all beats. The user logic must use the is_sop/is_eop signals in the pcie(n)_m_axis_cq_tuser bus in that case to determine the start and end of TLPs transferred over the interface.
pcie0_m_axis_cq_tvalid pcie1_m_axis_cq_tvalid	O	1	The core asserts this output whenever it is driving valid data on the pcie(n)_m_axis_cq_tdata bus. The core keeps the valid signal asserted during the transfer of a packet. The user application can pace the data transfer using the pcie(n)_m_axis_cq_tready signal.
pcie0_m_axis_cq_tready pcie1_m_axis_cq_tready	I	1	Activation of this signal by the user logic indicates to the PCIe core that the user logic is ready to accept data. Data is transferred across the interface when both pcie(n)_m_axis_cq_tvalid and pcie(n)_m_axis_cq_tready are asserted in the same cycle. If the user logic deasserts the ready signal when pcie(n)_m_axis_cq_tvalid is High, the core maintains the data on the bus and keeps the valid signal asserted until the user logic has asserted the ready signal.

Table 10: Completer Request Interface Port Descriptions (512-bit Interface) (cont'd)

Name	I/O	Width	Description
pcie0_cq_np_req pcie1_cq_np_req	I	2	<p>This input is used by the user application to request the delivery of a Non-Posted request. The core implements a credit-based flow control mechanism to control the delivery of Non-Posted requests across the interface, without blocking Posted TLPs.</p> <p>This input to the core controls an internal credit count. The credit count is updated in each clock cycle based on the setting of <code>pcie_cq_np_req[1:0]</code> as follows:</p> <ul style="list-style-type: none"> 00: No change 01: Increment by 1 10 or 11: Increment by 2 <p>The credit count is decremented on the delivery of each Non-Posted request across the interface. The core temporarily stops delivering Non-Posted requests to the user logic when the credit count is zero. It continues to deliver any Posted TLPs received from the link even when the delivery of Non-Posted requests has been paused.</p> <p>The user application can either set <code>pcie_cq_np_req[1:0]</code> in each cycle based on the status of its Non-Posted request receive buffer, or can set it to 11 permanently if it does not need to exercise selective backpressure on Non-Posted requests.</p> <p>The setting of <code>pcie_cq_np_req[1:0]</code> does not need to be aligned with the packet transfers on the completer request interface.</p>
pcie0_cq_np_req_count pcie1_cq_np_req_count	O	6	<p>This output provides the current value of the credit count maintained by the core for delivery of Non-Posted requests to the user logic. The core delivers a Non-Posted request across the completer request interface only when this credit count is non-zero. This counter saturates at a maximum limit of 32.</p> <p>Because of internal pipeline delays, there can be several cycles of delay between the user application providing credit on the <code>pcie_cq_np_req[1:0]</code> inputs and the PCIe core updating the <code>pcie_cq_np_req_count</code> output in response.</p> <p>This count resets on <code>user_reset</code> and de-assertion of <code>user_lnk_up</code>.</p>

When `PASID_CAP_ON` is enabled then `pcie(n)_m_axis_cq_tuser[228:183]` pins are shared with `cfg*` ports. The following table provides more information.

Table 11: Sideband Signals in pcie(n)_m_axis_cq_tuser (512-bit Interface)

Bit Index	Name	Width	Description
7:0	first_be[7:0]	8	<p>Byte enables for the first Dword of the payload. first_be[3:0] reflects the setting of the First Byte Enable bits in the Transaction-Layer header of the first TLP in this beat; and first_be[7:4] reflects the setting of the First Byte Enable bits in the Transaction-Layer header of the second TLP in this beat. For Memory Reads and I/O Reads, the 4 bits indicate the valid bytes to be read in the first Dword. For Memory Writes and I/O Writes, these bits indicate the valid bytes in the first Dword of the payload. For Atomic Operations and Messages with a payload, these bits are set to all 1s.</p> <p>Bits [7:4] of first_be are valid only when straddle is enabled on the CQ interface. When straddle is disabled, these bits are permanently set to 0s.</p> <p>This field is valid in the first beat of a packet. first_be[3:0] is valid when pcie(n)_m_axis_cq_tvalid and is_sop[0] are both asserted High. first_be[7:4] is valid when pcie(n)_m_axis_cq_tvalid and is_sop[1] are both asserted High.</p>
15:8	last_be[7:0]	8	<p>Byte enables for the last Dword of the payload. last_be[3:0] reflects the setting of the Last Byte Enable bits in the Transaction-Layer header of the first TLP in this beat; and last_be[7:4] reflects the setting of the Last Byte Enable bits in the Transaction-Layer header of the second TLP in this beat. For Memory Reads, the 4 bits indicate the valid bytes to be read in the last Dword of the block of data. For Memory Writes, these bits indicate the valid bytes in the ending Dword of the payload. For Memory Reads and Writes of one DW transfers and zero length transfers, these bits should be 0s. For Atomic Operations and Messages with a payload, these bits are set to all 1s.</p> <p>Bits [7:4] of last_be are valid only when straddle is enabled on the CQ interface. When straddle is disabled, these bits are permanently set to 0s.</p> <p>This field is valid in the first beat of a packet. last_be[3:0] is valid when pcie(n)_m_axis_cq_tvalid and is_eop[0] are both asserted High. last_be[7:4] is valid when pcie(n)_m_axis_cq_tvalid and is_eop[1] are both asserted High.</p>
79:16	byte_en[63:0]	64	<p>The user logic can optionally use these byte enable bits to determine the valid bytes in the payload of a packet being transferred. The assertion of bit <i>i</i> of this bus during a transfer indicates to the user logic that byte <i>i</i> of the pcie(n)_m_axis_cq_tdata bus contains a valid payload byte. This bit is not asserted for descriptor bytes.</p> <p>Although the byte enables can be generated by user logic from information in the request descriptor (address and length), as well as the settings of the first_be and last_be signals, the user logic has the option of using these signals directly instead of generating them from other interface signals.</p> <p>When the payload size is more than 2 Dwords (8 bytes), the first bits on this bus for the payload are always contiguous. When the payload size is 2 Dwords or less, the first bits might be non-contiguous.</p> <p>For the special case of a zero-length memory write transaction defined by the PCI Express Specifications, the byte_en bits are all 0 when the associated 1 Dword payload is being transferred.</p>

Table 11: Sideband Signals in pcie(n)_m_axis_cq_tuser (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
81:80	is_sop[1:0]	2	<p>Signals the start of a new TLP in this beat. These outputs are set in the first beat of a TLP. When straddle is disabled, only is_sop[0] is valid and is_sop[1] is permanently set to 0. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> 00: No new TLP starting in this beat. 01: A single new TLP starts in this beat. Its start position is indicated by is_sop0_ptr[1:0]. 11: Two new TLPs are starting in this beat. is_sop0_ptr[1:0] provides the start position of the first TLP and is_sop1_ptr[1:0] provides the start position of the second TLP. 10: Reserved. <p>Use of this signal is optional for the user logic when the straddle option is disabled, because a new TLP always starts in the beat following tlast assertion.</p>
83:82	is_sop0_ptr[1:0]	2	<p>Indicates the position of the first byte of the first TLP starting in this beat:</p> <ul style="list-style-type: none"> 00: Byte lane 0 10: Byte lane 32 01, 11: Reserved <p>This field is valid only when the straddle option is enabled on the CQ interface. Otherwise, it is set to 0 permanently, as a TLP can only start in byte lane 0.</p>
85:84	is_sop1_ptr[1:0]	2	<p>Indicates the position of the first byte of the second TLP starting in this beat:</p> <ul style="list-style-type: none"> 10: Byte lane 32 00, 01, 11: Reserved. <p>This output is used only when the straddle option is enabled on the CQ interface. The core can then straddle two TLPs in the same beat. The output is permanently set to 0 when straddle is disabled.</p>
87:86	is_eop[1:0]	2	<p>Indicates that a TLP is ending in this beat. These outputs are set in the final beat of a TLP. When straddle is disabled, only is_eop[0] is valid and is_eop[1] is permanently set to 0. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> 00: No TLPs ending in this beat. 01: A single TLP is ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of this TLP. 11: Two TLPs are ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP and is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP. 10: Reserved. <p>The use of this signal is optional for the user logic when the straddle option is not enabled, because tlast is asserted in the final beat of a TLP.</p>
91:88	is_eop0_ptr[3:0]	4	<p>Offset of the last Dword of the first TLP ending in this beat. This output is valid when is_eop[0] is asserted.</p>
95:92	is_eop1_ptr[3:0]	4	<p>Offset of the last Dword of the second TLP ending in this beat. This output is valid when is_eop[1] is asserted. The output is permanently set to 0 when straddle is disabled.</p>

Table 11: Sideband Signals in pcie(n)_m_axis_cc_tuser (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
96	discontinue	1	<p>This signal is asserted by the core in the last beat of a TLP, if it has detected an uncorrectable error while reading the TLP payload from its internal FIFO memory. The user application must discard the entire TLP when such an error is signaled by the core.</p> <p>This signal is never asserted when the TLP has no payload. It is asserted only in the last beat of the payload transfer, that is when is_eop[0] is High.</p> <p>When the straddle option is enabled, the core does not start a second TLP if it has asserted discontinue in a beat.</p> <p>When the core is configured as an Endpoint, the error is also reported by the core to the Root Complex it is attached to, using Advanced Error Reporting (AER).</p>
182:119	parity	64	Odd parity for the 512-bit transmit data. Bit <i>i</i> provides the odd parity computed for byte <i>i</i> of pcie(n)_m_axis_cc_tdata.
183	PASID TLP Valid 0	1	Indicates PASID TLP 0 is valid.
184	PASID TLP Valid 1	1	Indicates PASID TLP 1 is valid.
204:185	PASID 0	20	Indicates PASID TLP Prefix for packet0 to the user design.
224:205	PASID 1	20	Indicates PASID TLP Prefix for packet1 to the user design.
225	Execute Requested 0	1	Indicates Execute Requested for packet0.
226	Execute Requested 1	1	Indicates Execute Requested for packet1.
227	Privileged Mode Requested 0	1	Indicates Privileged Mode Requested for packet0 to the user design.
228	Privileged Mode Requested 1	1	Indicates Privileged Mode Requested for packet1 to the user design.

Completer Completion Interface

Table 12: Completer Completion Interface Port Descriptions (512-bit Interface)

Name	I/O	Width	Description
pcie0_s_axis_cc_tdata pcie1_s_axis_cc_tdata	I	512	Completion data from the user application to the PCIe core.
pcie0_s_axis_cc_tuser pcie1_s_axis_cc_tuser	I	81	<p>This is a set of signals containing sideband information for the TLP being transferred. These signals are valid when pcie(n)_s_axis_cc_tvalid is High.</p> <p>The individual signals in this set are described in the following table.</p>
pcie0_s_axis_cc_tlast pcie1_s_axis_cc_tlast	I	1	<p>The user application must assert this signal in the last cycle of a packet to indicate the end of the packet. When the TLP is transferred in a single beat, the user application must set this bit in the first cycle of the transfer.</p> <p>This input is used by the core only when the straddle option is disabled. When the straddle option is enabled, the core ignores the setting of this input, using instead the is_sop/ is_eop signals in the pcie(n)_s_axis_cc_tuser bus to determine the start and end of TLPs.</p>

Table 12: Completer Completion Interface Port Descriptions (512-bit Interface) (cont'd)

Name	I/O	Width	Description
pcie0_s_axis_cc_tkeep pcie1_s_axis_cc_tkeep	I	16	<p>The assertion of bit <i>i</i> of this bus during a transfer indicates to the core that Dword <i>i</i> of the pcie(n)_s_axis_cc_tdata bus contains valid data. The user logic must set this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_s_axis_cc_tdata must be set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (both in Dwords). This is true for both Dword-aligned and 128b address-aligned modes of payload transfer.</p> <p>The tkeep bits are valid only when straddle is not enabled on the CC interface. When straddle is enabled, the core ignores the setting of these bits when receiving data across the interface. The user logic must set the is_sop/is_eop signals in the pcie(n)_s_axis_cc_tuser bus in that case to signal the start and end of TLPs transferred over the interface.</p>
pcie0_s_axis_cc_tvalid pcie1_s_axis_cc_tvalid	I	1	<p>The user application must assert this output whenever it is driving valid data on the pcie(n)_s_axis_cc_tdata bus. The user application must keep the valid signal asserted during the transfer of a packet. The core paces the data transfer using the pcie(n)_s_axis_cc_tready signal.</p>
pcie0_s_axis_cc_tready pcie1_s_axis_cc_tready	O	4	<p>Activation of this signal by the PCIe core indicates that it is ready to accept data. Data is transferred across the interface when both pcie(n)_s_axis_cc_tvalid and pcie(n)_s_axis_cc_tready are asserted in the same cycle. If the core deasserts the ready signal when the valid signal is High, the user logic must maintain the data on the bus and keep the valid signal asserted until the core has asserted the ready signal.</p> <p>With this output port, each bit indicates the same value, so the user logic can use any of the bit.</p>

Table 13: Sideband Signals in pcie(n)_s_axis_cc_tuser

Bit Index	Name	Width	Description
1:0	is_sop[1:0]	2	<p>Signals the start of a new TLP in this beat. These outputs are set in the first beat of a TLP. When straddle is disabled, only is_sop[0] is valid. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> 00: No new TLP starting in this beat. 01: A single new TLP starts in this beat. Its start position is indicated by is_sop0_ptr[1:0]. 11: Two new TLPs are starting in this beat. is_sop0_ptr[1:0] provides the start position of the first TLP and is_sop1_ptr[1:0] provides the start position of the second TLP. 10: Reserved. <p>This field is used by the core only when the straddle option is enabled. When straddle is disabled, the core uses tlast to determine the first beat of an incoming TLP.</p>

Table 13: Sideband Signals in `pcie(n)_s_axis_cc_tuser` (cont'd)

Bit Index	Name	Width	Description
3:2	<code>is_sop0_ptr[1:0]</code>	2	<p>Indicates the position of the first byte of the first TLP starting in this beat:</p> <ul style="list-style-type: none"> 00: Byte lane 0 10: Byte lane 32 01, 11: Reserved <p>This field is used by the core only when the straddle option is enabled. When straddle is disabled, the user logic must always start a TLP in byte lane 0.</p>
5:4	<code>is_sop1_ptr[1:0]</code>	2	<p>Indicates the position of the first byte of the second TLP starting in this beat:</p> <ul style="list-style-type: none"> 10: Byte lane 32 00, 01, 11: Reserved. <p>This input is used only when the straddle option is enabled on the CC interface. The user can then straddle two TLPs in the same beat.</p>
7:6	<code>is_eop[1:0]</code>	2	<p>Signals that a TLP is ending in this beat. These outputs are set in the final beat of a TLP. When straddle is disabled, only <code>is_eop[0]</code> is valid. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> 00: No TLPs ending in this beat. 01: A single TLP is ending in this beat. <code>is_eop_ptr[3:0]</code> provides the offset of the last Dword of this TLP. 11: Two TLPs are ending in this beat. <code>is_eop_ptr[3:0]</code> provides the offset of the last Dword of the first TLP and <code>is_eop1_ptr[3:0]</code> provides the offset of the last Dword of the second TLP. 10: Reserved. <p>This field is used by the core only when the straddle option is enabled. When straddle is disabled, the core uses <code>tlast</code> and <code>tkeep</code> to determine the ending beat and position of EOP.</p>
11:8	<code>is_eop0_ptr[3:0]</code>	4	<p>Offset of the last Dword of the first TLP ending in this beat. This output is valid when <code>is_eop[0]</code> is asserted. This field is used by the core only when the straddle option is enabled.</p>
15:12	<code>is_eop1_ptr[3:0]</code>	4	<p>Offset of the last Dword of the second TLP ending in this beat. This output is valid when <code>is_eop[1]</code> is asserted. This field is used by the core only when the straddle option is enabled.</p>

Table 13: Sideband Signals in `pcie(n)_s_axis_cc_tuser` (cont'd)

Bit Index	Name	Width	Description
16	discontinue	1	<p>This signal can be asserted by the user application during a transfer if it has detected an error (such as an uncorrectable ECC error while reading the payload from memory) in the data being transferred and needs to abort the packet. The core nullifies the corresponding TLP on the link to avoid data corruption.</p> <p>The user logic can assert this signal in any beat during the transfer except the first beat of the TLP. It can either choose to terminate the packet prematurely in the cycle where the error was signaled, or continue until all bytes of the payload are delivered to the core. In the latter case, the core treats the error as sticky for the following beats of the packet, even if the user logic deasserts the discontinue signal before the end of the packet.</p> <p>The discontinue signal can be asserted only when <code>pcie(n)_s_axis_cc_tvalid</code> is High. The core samples this signal only when <code>pcie(n)_s_axis_cc_tready</code> is High. Thus, once asserted, it should not be deasserted until <code>pcie(n)_s_axis_cc_tready</code> is High.</p> <p>When the straddle option is enabled on the CC interface, the user should not start a new TLP in the same beat when a TLP is ending with discontinue asserted.</p> <p>When the core is configured as an Endpoint, this error is also reported by the core to the Root Complex it is attached to, using Advanced Error Reporting (AER).</p>
80:17	parity	64	<p>Odd parity for the 256-bit data. When parity checking is enabled in the core, user logic must set bit <i>i</i> of this bus to the odd parity computed for byte <i>i</i> of <code>pcie(n)_s_axis_cc_tdata</code>.</p> <p>On detection of a parity error, the core nullifies the corresponding TLP on the link and reports it as an Uncorrectable Internal Error.</p> <p>The parity bits can be permanently tied to 0 if parity check is not enabled in the core.</p>

Requester Request Interface

Table 14: Requester Request Interface Port Descriptions (512-bit Interface)

Name	Width	I/O	Description
<code>pcie0_s_axis_rq_tdata</code> <code>pcie1_s_axis_rq_tdata</code>	512	I	Requester-side request data from the user application to the PCIe core.
<code>pcie0_s_axis_rq_tuser</code> <code>pcie1_s_axis_rq_tuser</code>	183	I	This is a set of signals containing sideband information for the TLP being transferred. These signals are valid when <code>pcie(n)_s_axis_rq_tvalid</code> is High. The individual signals in this set are described in the following table.
<code>pcie0_s_axis_rq_tlast</code> <code>pcie1_s_axis_rq_tlast</code>	1	I	<p>The user application must assert this signal in the last cycle of a TLP to indicate the end of the packet. When the TLP is transferred in a single beat, the user logic must set this bit in the first cycle of the transfer.</p> <p>This input is used by the core only when the straddle option is disabled. When the straddle option is enabled, the core ignores the setting of this input, using instead the <code>is_sop/</code> <code>is_eop</code> signals in the <code>s_axis_rq_tuser</code> bus to determine the start and end of TLPs.</p>

Table 14: Requester Request Interface Port Descriptions (512-bit Interface) (cont'd)

Name	Width	I/O	Description
pcie0_s_axis_rq_tkeep pcie1_s_axis_rq_tkeep	16	I	<p>The assertion of bit <i>i</i> of this bus during a transfer indicates to the core that Dword <i>i</i> of the pcie(n)_s_axis_rq_tdata bus contains valid data. The user logic must set this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_s_axis_rq_tdata must be set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (both in Dwords). This is true for both Dword-aligned and 128b address-aligned modes of payload transfer.</p> <p>The tkeep bits are valid only when straddle is not enabled on the RQ interface. When straddle is enabled, the core ignores the setting of these bits when receiving data across the interface. The user logic must set the is_sop/is_eop signals in the pcie(n)_s_axis_rq_tuser bus in that case to signal the start and end of TLPs transferred over the interface.</p>
pcie0_s_axis_rq_tvalid pcie1_s_axis_rq_tvalid	1	I	<p>The user application must assert this output whenever it is driving valid data on the pcie(n)_s_axis_rq_tdata bus. The user application must keep the valid signal asserted during the transfer of a packet. The core paces the data transfer using the pcie(n)_s_axis_rq_tready signal.</p>
pcie0_s_axis_rq_tready pcie1_s_axis_rq_tready	4	O	<p>Activation of this signal by the PCIe core indicates that it is ready to accept data. Data is transferred across the interface when both pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are asserted in the same cycle. If the core deasserts the ready signal when the valid signal is High, the user logic must maintain the data on the bus and keep the valid signal asserted until the core has asserted the ready signal.</p> <p>With this output port, each bit indicates the same value, so the user logic can use any of the bit.</p>
pcie0_rq_tag_vld0 pcie1_rq_tag_vld0	1	O	<p>The core asserts this output for one cycle when it has allocated a tag to an incoming Non-Posted request from the requester request interface and placed it on the pcie(n)_rq_tag0 output. The bit is encoded as follows:</p> <ul style="list-style-type: none"> • 0: No tags being provided in this cycle. • 1: A tag is presented on pcie(n)_rq_tag0.
pcie0_rq_tag_vld1 pcie1_rq_tag_vld1	1	O	<p>The core asserts this output for one cycle when it has allocated a tag to an incoming Non-Posted request from the requester request interface and placed it on the pcie(n)_rq_tag1 output. The bit is encoded as follows:</p> <ul style="list-style-type: none"> • 0: No tag is provided on pcie(n)_rq_tag1 in this cycle. • 1: A tag is presented on pcie(n)_rq_tag1.

Table 14: Requester Request Interface Port Descriptions (512-bit Interface) (cont'd)

Name	Width	I/O	Description
pcie0_rq_tag0 pcie1_rq_tag0	8	O	<p>When tag management for Non-Posted requests is performed by the core (Enable Client Tag is unchecked in the IP customization GUI), this output is used by the core to communicate the allocated tag for each Non-Posted request received from the client. The tag value on pcie(n)_rq_tag0 is valid for one cycle when pcie(n)_rq_tag_vld0 is High. The client must copy this tag and use it to associate the completion data with the pending request.</p> <p>There can be a delay of several cycles between the transfer of the request on the pcie(n)_s_axis_rq_tdata bus and the assertion of pcie(n)_rq_tag_vld0 by the core to provide the allocated tag for the request. The client can, meanwhile, continue to send new requests. The tags for requests are communicated on this bus in FIFO order. Therefore, the user application must associate the allocated tags with the requests in the order in which the requests were transferred over the interface.</p> <p>When pcie(n)_rq_tag0 and pcie(n)_rq_tag1 are both valid in the same cycle, the value on pcie(n)_rq_tag0 corresponds to the earlier of the two requests transferred over the interface.</p>
pcie0_rq_tag1 pcie1_rq_tag1	8	O	<p>The description of this signal is the same as pcie(n)_rq_tag0, except the tag value on pcie(n)_rq_tag1 is valid for one cycle when pcie(n)_rq_tag_vld1 is asserted.</p>
pcie0_rq_seq_num0 pcie1_rq_seq_num0	6	O	<p>The user may optionally use this output to keep track of the progress of the request in the core's transmit pipeline. To use this feature, the user application must provide a sequence number for each request on the pcie(n)_s_axis_rq_seq_num0[5:0] bus. The core outputs this sequence number on the pcie(n)_rq_seq_num0[5:0] output when the request TLP has progressed to a point in the pipeline where a Completion TLP from the client will not be able to pass it. This mechanism enables the client to maintain ordering between Completions sent to the completer completion interface of the core and Posted requests sent to the requester request interface.</p> <p>Data on the pcie(n)_rq_seq_num0[5:0] output is valid when pcie(n)_rq_seq_num_vld0 is High.</p>
pcie0_rq_seq_num1 pcie1_rq_seq_num1	6	O	<p>This output is identical in function to that of pcie(n)_rq_seq_num0. It is used to provide a second sequence number in the same cycle when a first sequence number is being presented on pcie(n)_rq_seq_num0.</p> <p>Data on the pcie(n)_rq_seq_num1[5:0] output is valid when pcie(n)_rq_seq_num_vld1 is High.</p>
pcie0_rq_seq_num_vld0 pcie1_rq_seq_num_vld0	1	O	<p>This output is asserted by the core for one cycle when it has placed valid data on pcie(n)_rq_seq_num0[5:0].</p>
pcie0_rq_seq_num_vld1 pcie1_rq_seq_num_vld1	1	O	<p>This output is asserted by the core for one cycle when it has placed valid data on pcie(n)_rq_seq_num1[5:0].</p>

When PASID_CAP_ON is enabled then pcie(n)_s_axis_rq_tuser [182:137] pins are shared with cfg* ports. The following tables provides more details.

Table 15: Sideband Signals in pcie(n)_s_axis_rq_tuser (512-bit Interface)

Bit Index	Name	Width	Description
7:0	first_be[7:0]	8	<p>Byte enables for the first Dword. This field must be set based on the desired value of the first_be bits in the Transaction-Layer header of the request TLP. first_be[3:0] corresponds to the byte enables for the first TLP starting in this beat, and first_be[7:4] corresponds to the byte enables for the second TLP starting in this beat (if present).</p> <p>For Memory Reads, I/O Reads and Configuration Reads, these 4 bits indicate the valid bytes to be read in the first Dword. For Memory Writes, I/O Writes and Configuration Writes, these bits indicate the valid bytes in the first Dword of the payload.</p> <p>The core samples this field in the first beat of a packet, when pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are both High.</p>
15:8	last_be[7:0]	8	<p>Byte enables for the last Dword.</p> <p>This field must be set based on the desired value of the last_be bits in the Transaction-Layer header of the TLP. last_be[3:0] corresponds to the byte enables for the first TLP starting in this beat, and last_be[7:4] corresponds to the byte enables for the second TLP starting in this beat (if present).</p> <p>For Memory Reads and Writes of one DW transfers and zero length transfers, these bits should be 0s.</p> <p>For Memory Reads of 2 Dwords or more, these 4 bits indicate the valid bytes to be read in the last Dword of the block of data. For Memory Writes of 2 Dwords or more, these bits indicate the valid bytes in the last Dword of the payload.</p> <p>The core samples this field in the first beat of a packet, when pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are both High.</p>
19:16	addr_offset[3:0]	4	<p>When 128b the address-aligned mode is in use on this interface, the user application must provide the offset where the payload data begins (in multiples of 4 bytes) on the data bus on this sideband bus. This enables the core to determine the alignment of the data block being transferred.</p> <p>addr_offset[1:0] corresponds to the offset for the first TLP starting in this beat, and addr_offset[3:2] is reserved for future use.</p> <p>The core samples this field in the first beat of a packet, when pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are both High.</p> <p>When the requester request interface is configured in the Dword-alignment mode, these bits must always be set to 0.</p>

Table 15: Sideband Signals in `pcie(n)_s_axis_rq_tuser` (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
21:20	<code>is_sop[1:0]</code>	2	<p>Signals the start of a new TLP in this beat. These outputs are set in the first beat of a TLP. When straddle is disabled, only <code>is_sop[0]</code> is valid. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> 00: No new TLP starting in this beat. 01: A single new TLP starts in this beat. Its start position is indicated by <code>is_sop0_ptr[1:0]</code>. 11: Two new TLPs are starting in this beat. <code>is_sop0_ptr[1:0]</code> provides the start position of the first TLP and <code>is_sop1_ptr[1:0]</code> provides the start position of the second TLP. 10: Reserved. <p>Use of this signal is optional for the user logic when the straddle option is not enabled, because a new TLP always starts in the beat following <code>tlast</code> assertion.</p>
23:22	<code>is_sop0_ptr[1:0]</code>	2	<p>Indicates the position of the first byte of the first TLP starting in this beat:</p> <ul style="list-style-type: none"> 00: Byte lane 0 10: Byte lane 32 01, 11: Reserved
25:24	<code>is_sop1_ptr[1:0]</code>	2	<p>Indicates the position of the first byte of the second TLP starting in this beat:</p> <ul style="list-style-type: none"> 10: Byte lane 32 00, 01, 11: Reserved. <p>This output is used only when the straddle option is enabled on the interface.</p>
27:26	<code>is_eop[1:0]</code>	2	<p>Signals that a TLP is ending in this beat. These outputs are set in the final beat of a TLP. When straddle is disabled, only <code>is_eop[0]</code> is valid. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> 00: No TLPs ending in this beat. 01: A single TLP is ending in this beat. <code>is_eop0_ptr[3:0]</code> provides the offset of the last Dword of this TLP. 11: Two TLPs are ending in this beat. <code>is_eop0_ptr[3:0]</code> provides the offset of the last Dword of the first TLP and <code>is_eop1_ptr[3:0]</code> provides the offset of the last Dword of the second TLP. 10: Reserved. <p>Use of this signal is optional for the user logic when the straddle option is not enabled, because <code>tlast</code> is asserted in the final beat of a TLP.</p>
31:28	<code>is_eop0_ptr[3:0]</code>	4	<p>Offset of the last Dword of the first TLP ending in this beat. This output is valid when <code>is_eop[0]</code> is asserted.</p>
35:32	<code>is_eop1_ptr[3:0]</code>	4	<p>Offset of the last Dword of the second TLP ending in this beat. This output is valid when <code>is_eop[1]</code> is asserted.</p>

Table 15: Sideband Signals in pcie(n)_s_axis_rq_tuser (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
36	discontinue	1	<p>This signal can be asserted by the user application during a transfer if it has detected an error in the data being transferred and needs to abort the packet. The core nullifies the corresponding TLP on the link to avoid data corruption.</p> <p>The user logic can assert this signal in any beat of a TLP except the first beat during its transfer. It can either choose to terminate the packet prematurely in the cycle where the error was signaled, or continue until all bytes of the payload are delivered to the core. In the latter case, the core treats the error as sticky for the following beats of the packet, even if the user logic deasserts the discontinue signal before the end of the packet.</p> <p>The discontinue signal can be asserted only when pcie(n)_s_axis_rq_tvalid is High. The core samples this signal only when pcie(n)_s_axis_rq_tready is High. Thus, once asserted, it should not be deasserted until pcie(n)_s_axis_rq_tready is High.</p> <p>When the straddle option is enabled on the RQ interface, the user should not start a new TLP in the same beat when a TLP is ending with discontinue asserted.</p> <p>When the core is configured as an Endpoint, this error is also reported by the core to the Root Complex it is attached to, using Advanced Error Reporting (AER).</p>
66:61	seq_num0[5:0]	6	<p>The user logic can optionally supply a 6-bit sequence number in this field to keep track of the progress of the request in the core's transmit pipeline. The core outputs this sequence number on its pcie(n)_rq_seq_num0 or pcie(n)_rq_seq_num1 output when the request TLP has progressed to a point in the pipeline where a Completion TLP from the user logic is not able to pass it.</p> <p>The core samples this field in the first beat of a packet, when pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are both High.</p> <p>This input can be hardwired to 0 when the user logic is not monitoring the pcie(n)_rq_seq_num* outputs of the core.</p>
72:67	seq_num1[5:0]	6	<p>If there is a second TLP starting in the same beat, the user logic can optionally provide a 6-bit sequence number for this TLP on this input. This sequence number is used in the same manner as seq_num0.</p> <p>The core samples this field in the first beat of a packet, when pcie(n)_s_axis_rq_tvalid and pcie(n)_s_axis_rq_tready are both High.</p> <p>This input can be hardwired to 0 when the user logic is not monitoring the pcie(n)_rq_seq_num* outputs of the core.</p>
136:73	parity	64	<p>Odd parity for the 512-bit data. When parity checking is enabled in the core, user logic must set bit <i>i</i> of this bus to the odd parity computed for byte <i>i</i> of pcie(n)_s_axis_rq_tdata.</p> <p>On detection of a parity error, the core nullifies the corresponding TLP on the link and reports it as an Uncorrectable Internal Error.</p> <p>These bits can be set to 0 if parity checking is disabled in the core.</p>
137	PASID TLP Valid 0	1	<p>Indicates PASID TLP is valid for packet 0. pcie_posted_req_delivered is repurposed to pass PASID TLP VALID0 information from the user design.</p>

Table 15: Sideband Signals in pcie(n)_s_axis_rq_tuser (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
138	PASID TLP Valid 1	1	Indicates PASID TLP is valid for packet 1. pcie_cq_pipeline_empty is repurposed to pass PASID TLP VALID1 information from the user design.
158:139	PASID 0	20	Indicates PASID TLP Prefix for packet 0. Repurpose the following signals to pass PASID 0 information from the user design. [139]: pcie_cq_np_user_credit_rcvd. [141:140]: pcie_compl_delivered. [149:142]: pcie_compl_delivered_tag0. [157:150]: pcie_compl_delivered_tag1. user_spare_in[0]: tl_rx_posted_credit_released.
178:159	PASID 1	20	Indicates PASID TLP Prefix for packet 1. Repurpose the following list of signals to pass PASID 1 information from the user design. user_spare_in[20:1] indexed as below: [165:159]: tl_rx_posted_payload_credit_released_value. [166]: tl_rx_nonposted_credit_released. [168:167]: tl_rx_nonposted_payload_credit_released_value. [169]: tl_rx_compl_credit_released. [171:170]: tl_rx_compl_header_credit_released_value. [178:172]: tl_rx_compl_payload_credit_released_value.
179	Execute Requested 0	1	Indicates Execute Requested 0.
180	Execute Requested 1	1	Indicates Execute Requested 1.
181	Privileged Mode Requested 0	1	Indicates Privileged Mode Requested 0.
182	Privileged Mode Requested 1	1	Indicates Privileged Mode Requested 1.

Requester Completion Interface

Table 16: Requester Completion Interface Port Descriptions (512-bit Interface)

Name	I/O	Width	Description
pcie0_m_axis_rc_tdata pcie1_m_axis_rc_tdata	O	512	Transmit data from the PCIe requester completion interface to the user application.
pcie0_m_axis_rc_tuser pcie1_m_axis_rc_tuser	O	161	This is a set of signals containing sideband information for the TLP being transferred. These signals are valid when pcie(n)_m_axis_rc_tvalid is High. The individual signals in this set are described in the following table.
pcie0_m_axis_rc_tlast pcie1_m_axis_rc_tlast	O	1	The core asserts this signal in the last beat of a packet to indicate the end of the packet. When a TLP is transferred in a single beat, the core sets this bit in the first beat of the transfer. This output is used only when the straddle option is disabled. When the straddle option is enabled, the core sets this output permanently to 0.

Table 16: Requester Completion Interface Port Descriptions (512-bit Interface) (cont'd)

Name	I/O	Width	Description
pcie0_m_axis_rc_tkeep pcie1_m_axis_rc_tkeep	O	16	<p>The assertion of bit <i>i</i> of this bus during a transfer indicates to the user logic that Dword <i>i</i> of the pcie(n)_m_axis_rc_tdata bus contains valid data. The core sets this bit to 1 contiguously for all Dwords starting from the first Dword of the descriptor to the last Dword of the payload. Thus, pcie(n)_m_axis_rc_tkeep is set to all 1s in all beats of a packet, except in the final beat when the total size of the packet is not a multiple of the width of the data bus (both in Dwords). This is true for both Dword-aligned and address-aligned modes of payload transfer.</p> <p>These outputs are permanently set to all 1s when the straddle option is enabled. The user logic must use the signals in pcie(n)_m_axis_rc_tuser in that case to determine the start and end of Completion TLPs transferred over the interface.</p>
pcie0_m_axis_rc_tvalid pcie1_m_axis_rc_tvalid	O	1	<p>The core asserts this output whenever it is driving valid data on the pcie(n)_m_axis_rc_tdata bus. The core keeps the valid signal asserted during the transfer of a packet. The user application can pace the data transfer using the pcie(n)_m_axis_rc_tready signal.</p>
pcie0_m_axis_rc_tready pcie1_m_axis_rc_tready	I	1	<p>Activation of this signal by the user logic indicates to the PCIe core that the user logic is ready to accept data. Data is transferred across the interface when both pcie(n)_m_axis_rc_tvalid and pcie(n)_m_axis_rc_tready are asserted in the same cycle.</p> <p>If the user logic deasserts the ready signal when the valid signal is High, the core maintains the data on the bus and keep the valid signal asserted until the user logic has asserted the ready signal.</p>

Table 17: Sideband Signals in pcie(n)_m_axis_rc_tuser (512-bit Interface)

Bit Index	Name	Width	Description
63:0	byte_en	64	<p>The client logic may optionally use these byte enable bits to determine the valid bytes in the payload of a packet being transferred. The assertion of bit <i>i</i> of this bus during a transfer indicates to the client that byte <i>i</i> of the pcie(n)_m_axis_cq_tdata bus contains a valid payload byte. This bit is not asserted for descriptor bytes.</p> <p>Although the byte enables can be generated by client logic from information in the request descriptor (address and length), the client has the option of using these signals directly instead of generating them from other interface signals. The 1 bits in this bus for the payload of a TLP are always contiguous.</p>

Table 17: Sideband Signals in pcie(n)_m_axis_rc_tuser (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
67:64	is_sop[3:0]	4	<p>Signals the start of a new TLP in this beat. These outputs are set in the first beat of a TLP. When straddle is disabled, only is_sop[0] is valid and is_sop[3:1] are permanently set to 0. When straddle is enabled, the settings are as follows:</p> <ul style="list-style-type: none"> • 0000: No new TLP starting in this beat. • 0001: A single new TLP starts in this beat. ts start position is indicated by is_sop0_ptr[1:0]. • 0011: Two new TLPs are starting in this beat. is_sop0_ptr[1:0] provides the start position of the first TLP and is_sop1_ptr[1:0] provides the start position of the second TLP. • 0111: Three new TLPs are starting in this beat. is_sop0_ptr[1:0] provides the start position of the first TLP, is_sop1_ptr[1:0] provides the start position of the second TLP, and is_sop2_ptr[1:0] provides the start position of the third TLP. • 1111: Four new TLPs are starting in this beat. is_sop0_ptr[1:0] provides the start position of the first TLP, is_sop1_ptr[1:0] provides the start position of the second TLP, is_sop2_ptr[1:0] provides the start position of the third TLP, and is_sop3_ptr[1:0] provides the start position of the fourth TLP. • All other settings are reserved. <p>Use of this signal is optional for the client when the straddle option is not enabled, because a new TLP always starts in the beat following pcie(n)_m_axis_rc_tlast assertion.</p>
69:68	is_sop0_ptr[1:0]	2	<p>Indicates the position of the first byte of the first TLP starting in this beat:</p> <ul style="list-style-type: none"> • 00: Byte lane 0 • 01: Byte lane 16 • 10: Byte lane 32 • 11: Byte lane 48 <p>This field is valid only when the straddle option is enabled on the RC interface. Otherwise, it is set to 0 permanently, as a TLP can only start in byte lane 0.</p>
71:70	is_sop1_ptr[1:0]	2	<p>Indicates the position of the first byte of the second TLP starting in this beat:</p> <ul style="list-style-type: none"> • 00: Reserved • 01: Byte lane 16 • 10: Byte lane 32 • 11: Byte lane 48 <p>This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.</p>

Table 17: Sideband Signals in pcie(n)_m_axis_rc_tuser (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
73:72	is_sop2_ptr[1:0]	2	<p>Indicates the position of the first byte of the third TLP starting in this beat:</p> <ul style="list-style-type: none"> 00: Reserved 01: Reserved 10: Byte lane 32 11: Byte lane 48 <p>This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.</p>
75:74	is_sop3_ptr[1:0]	2	<p>Indicates the position of the first byte of the fourth TLP starting in this beat:</p> <ul style="list-style-type: none"> 00, 01, 10: Reserved 11: Byte lane 48 <p>This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.</p>
79:76	is_eop[3:0]	4	<p>Signals that one or more TLPs are ending in this beat only when straddle is enabled. These outputs are set in the final beat of a TLP. The settings are as follows:</p> <ul style="list-style-type: none"> 0000: No TLPs ending in this beat. 0001: A single TLP is ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of this TLP. 0011: Two TLPs are ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP and is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP. 0111: Three TLPs are ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP, is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP, and is_eop2_ptr[3:0] provides the offset of the last Dword of the third TLP. 1111: Four TLPs are ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP, is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP, is_eop2_ptr[3:0] provides the offset of the last Dword of the third TLP, and is_eop3_ptr[3:0] provides the offset of the last Dword of the fourth TLP. All other settings are reserved. <p>When the straddle option is disabled, pcie(n)_m_axis_rc_tlast indicates the final beat of a TLP.</p>
83:80	is_eop0_ptr[3:0]	4	<p>Offset of the last Dword of the first TLP ending in this beat. This output is valid when is_eop[0] is asserted.</p> <p>This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.</p>
87:84	is_eop1_ptr[3:0]	4	<p>Offset of the last Dword of the second TLP ending in this beat. This output is valid when is_eop[1] is asserted.</p> <p>This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.</p>

Table 17: Sideband Signals in `pcie(n)_m_axis_rc_tuser` (512-bit Interface) (cont'd)

Bit Index	Name	Width	Description
91:88	<code>is_eop2_ptr[3:0]</code>	4	Offset of the last Dword of the third TLP ending in this beat. This output is valid when <code>is_eop[2]</code> is asserted. This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.
95:92	<code>is_eop3_ptr[3:0]</code>	4	Offset of the last Dword of the fourth TLP ending in this beat. This output is valid when <code>is_eop[3]</code> is asserted. This output is used only when the straddle option is enabled on the RC interface. The output is permanently set to 0 when straddle is disabled.
96	<code>discontinue</code>	1	This signal is asserted by the core in the last beat of a TLP, if it has detected an uncorrectable error while reading the TLP payload from its internal FIFO memory. The client application must discard the entire TLP when such an error is signaled by the core. This signal is never asserted when the TLP has no payload. It is asserted only in the last beat of the payload transfer, that is when <code>is_eop[0]</code> is High. When the straddle option is enabled, the core does not start a new TLP if it has asserted <code>discontinue</code> in a beat. When the core is configured as an Endpoint, the error is also reported by the core to the Root Complex it is attached to, using Advanced Error Reporting (AER).
160:97	<code>parity</code>	64	Odd parity for the 512-bit transmit data. Bit <i>i</i> provides the odd parity computed for byte <i>i</i> of <code>pcie(n)_m_axis_cq_tdata</code> .

Clock and Reset Interface

Fundamental to the operation of the core, the Clock and Reset interface provides the system-level clock and reset to the core as well as the user application clock and reset signal. The table below defines the ports in the Clock and Reset interface of the core.

The `pcie(n)_user_clk` signal is the clock available in the fabric region. For more information on clock and reset interface, refer to 'Clocking' and 'Resets' in "Designing with the Core" chapter.

Note: The `pcie0*` signals map to PCIe Controller 0, and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 18: Clock and Reset Interface Port Descriptions

Port	I/O	Width	Description
<code>pcie0_user_clk</code> <code>pcie1_user_clk</code>	O	1	User clock output (62.5, 125, or 250 MHz) This clock has a fixed frequency and is configured in the Vivado® Integrated Design Environment (IDE).
<code>pcie0_user_reset</code> <code>pcie1_user_reset</code>	O	1	This signal is deasserted synchronously with respect to <code>pcie(n)_user_clk</code> . It is asserted asynchronously with <code>perst</code> assertion.

Related Information

[Clocking](#)

[Resets](#)

Configuration Management Interface

The Configuration Management interface is used to read and write to the Configuration Space Registers. The following table defines the ports in the Configuration Management interface of the core.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 19: Configuration Management Interface Port Descriptions

Port	I/O	Width	Description
<code>pcie0_cfg_mgmt_addr</code> <code>pcie1_cfg_mgmt_addr</code>	I	10	Read/Write Address Configuration Space Dword-aligned address.
<code>pcie0_cfg_mgmt_function_number</code> <code>pcie1_cfg_mgmt_function_number</code>	I	8	PCI Function Number Selects the PCI function number for the configuration register read/write.
<code>pcie0_cfg_mgmt_write</code> <code>pcie1_cfg_mgmt_write</code>	I	1	Write Enable Asserted for a write operation. Active-High.
<code>pcie0_cfg_mgmt_write_data</code> <code>pcie1_cfg_mgmt_write_data</code>	I	32	Write data Write data is used to configure the Configuration and Management registers.
<code>pcie0_cfg_mgmt_byte_enable</code> <code>pcie1_cfg_mgmt_byte_enable</code>	I	4	Byte Enable Byte enable for write data, where <code>pcie(n)_cfg_mgmt_byte_enable[0]</code> corresponds to <code>pcie(n)_cfg_mgmt_write_data[7:0]</code> , and so on.
<code>pcie0_cfg_mgmt_read</code> <code>pcie1_cfg_mgmt_read</code>	I	1	Read Enable Asserted for a read operation. Active-High.
<code>pcie0_cfg_mgmt_read_data</code> <code>pcie1_cfg_mgmt_read_data</code>	O	32	Read data out Read data provides the configuration of the Configuration and Management registers.
<code>pcie0_cfg_mgmt_read_write_done</code> <code>pcie1_cfg_mgmt_read_write_done</code>	O	1	Read/Write operation complete Asserted for 1 cycle when operation is complete. Active-High.
<code>pcie0_cfg_mgmt_debug_access</code> <code>pcie1_cfg_mgmt_debug_access</code>	I	1	Type 1 RO, Write When the core is configured in the Root Port mode, asserting this input during a write to a Type-1 configuration space register forces a write into certain read-only fields of the register (see description of RC-mode Config registers). This input has no effect when the core is in the Endpoint mode, or when writing to any register other than a Type-1 configuration space register.

Configuration Status Interface

The Configuration Status interface provides information on how the core is configured, such as the negotiated link width and speed, the power state of the core, and configuration errors. The following table defines the ports in the Configuration Status interface of the core.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 20: Configuration Status Interface Port Descriptions

Port	I/O	Width	Description
<code>pcie0_cfg_phy_link_down</code> <code>pcie1_cfg_phy_link_down</code>	O	1	Configuration Link Down Status of the PCI Express link based on the Physical Layer LTSSM. <ul style="list-style-type: none"> 1b: Link is Down (LinkUp state variable is 0b) 0b: Link is Up (LinkUp state variable is 1b) Note: Per the PCI Express Base Specification, rev. 3.0, LinkUp is 1b in the Recovery, L0, L0s, L1, and L2 <code>cfg_ltssm</code> states. In the Configuration state, LinkUp can be 0b or 1b. It is always 0b when the Configuration state is reached using Detect > Polling > Configuration. LinkUp is 1b if the configuration state is reached through any other state transition. Note: While reset is asserted, the output of this signal are 0b until reset is released.
<code>pcie0_cfg_phy_link_status</code> <code>pcie1_cfg_phy_link_status</code>	O	2	Configuration Link Status Status of the PCI Express link. <ul style="list-style-type: none"> 00b: No receivers detected 01b: Link training in progress 10b: Link up, DL initialization in progress 11b: Link up, DL initialization completed
<code>pcie0_cfg_negotiated_width</code> <code>pcie1_cfg_negotiated_width</code>	O	3	Negotiated Link Width This output indicates the negotiated width of the given PCI Express Link and is valid when <code>cfg_phy_link_status[1:0] == 11b</code> (DL Initialization is complete). Negotiated Link Width values: <ul style="list-style-type: none"> 000b = x1 001b = x2 010b = x4 011b = x8 100b = x16 Other values are reserved.

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_current_speed pcie1_cfg_current_speed	O	2	Current Link Speed This signal outputs the current link speed of the given PCI Express Link. <ul style="list-style-type: none"> • 00b: 2.5 GT/s PCI Express Link Speed • 01b: 5.0 GT/s PCI Express Link Speed • 10b: 8.0 GT/s PCI Express Link Speed • 11b: 16.0 GT/s PCI Express Link Speed
pcie0_cfg_max_payload	O	2	Max_Payload_Size This signal outputs the maximum payload size from Device Control register bits 7 down to 5. This field sets the maximum TLP payload size. As a Receiver, the logic must handle TLPs as large as the set value. As a Transmitter, the logic must not generate TLPs exceeding the set value. <ul style="list-style-type: none"> • 00b: 128 bytes maximum payload size • 01b: 256 bytes maximum payload size • 10b: 512 bytes maximum payload size • 11b: 1024 bytes maximum payload size
pcie0_cfg_max_read_req	O	3	Max_Read_Request_Size This signal outputs the maximum read request size from Device Control register bits 14 down to 12. This field sets the maximum Read Request size for the logic as a Requester. The logic must not generate Read Requests with size exceeding the set value. <ul style="list-style-type: none"> • 000b: 128 bytes maximum Read Request size • 001b: 256 bytes maximum Read Request size • 010b: 512 bytes maximum Read Request size • 011b: 1024 bytes maximum Read Request size • 100b: 2048 bytes maximum Read Request size • 101b: 4096 bytes maximum Read Request size • Other values are reserved

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_function_status	O	16	<p>Configuration Function Status</p> <p>These outputs indicate the states of the Command register bits in the PCI configuration space of each function. These outputs are used to enable requests and completions from the host logic. The assignment of bits is as follows:</p> <ul style="list-style-type: none"> • Bit 0: Function 0 I/O Space Enable • Bit 1: Function 0 Memory Space Enable • Bit 2: Function 0 Bus Master Enable • Bit 3: Function 0 INTx Disable • Bit 4: Function 1 I/O Space Enable • Bit 5: Function 1 Memory Space Enable • Bit 6: Function 1 Bus Master Enable • Bit 7: Function 1 INTx Disable • Bit 8: Function 2 I/O Space Enable • Bit 9: Function 2 Memory Space Enable • Bit 10: Function 2 Bus Master Enable • Bit 11: Function 2 INTx Disable • Bit 12: Function 3 I/O Space Enable • Bit 13: Function 3 Memory Space Enable • Bit 14: Function 3 Bus Master Enable • Bit 15: Function 3 INTx Disable
pcie0_cfg_vf_status pcie1_cfg_vf_status	O	504	<p>Configuration Virtual Function Status</p> <ul style="list-style-type: none"> • Bit 0: Virtual function 0: Configured/Enabled by the software. • Bit 1: Virtual function 0: PCI Command register, Bus Master Enable. • Bit 2: Virtual function 1: Configured/Enabled by software. • Bit 3: Virtual function 1: PCI Command register, Bus Master Enable.
pcie0_cfg_function_power_state	O	12	<p>Configuration Function Power State</p> <p>These outputs indicate the current power state of the physical functions. Bits [2:0] capture the power state of function 0, and bits [5:3] capture that of function 1, and so on. The possible power states are:</p> <ul style="list-style-type: none"> • 000: D0_uninitialized • 001: D0_active • 010: D1 • 100: D3_hot • Other values are reserved.

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_vf_power_state pcie1_cfg_vf_power_state	O	756	Configuration Virtual Function Power State These outputs indicate the current power state of the virtual functions. Bits [2:0] capture the power state of virtual function 0, and bits [5:3] capture that of virtual function 1, and so on. The possible power states are: <ul style="list-style-type: none"> • 000: D0_uninitialized • 001: D0_active • 010: D1 • 100: D3_hot • Other values are reserved.
pcie0_cfg_link_power_state pcie1_cfg_link_power_state	O	2	Current power state of the PCI Express link, and is valid when <code>cfg_phy_link_status[1:0] == 11b</code> (DL Initialization is complete). <ul style="list-style-type: none"> • 00: L0 • 01: TX L0s • 10: L1 • 11: L2/3 Ready

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_local_error_out pcie1_cfg_local_error_out	O	5	<p>Local Error Conditions: Error priority is noted and Priority 0 has the highest priority.</p> <ul style="list-style-type: none"> • 00000b - Reserved • 00001b - Physical Layer Error Detected (Priority 16) • 00010b - Link Replay Timeout (Priority 12) • 00011b - Link Replay Rollover (Priority 13) • 00100b - Link Bad TLP Received (Priority 10) • 00101b - Link Bad DLLP Received (Priority 11) • 00110b - Link Protocol Error (Priority 9) • 00111b - Replay Buffer RAM Correctable ECC Error (Priority 22) • 01000b - Replay Buffer RAM Uncorrectable ECC Error (Priority 3) • 01001b - Receive Posted Request RAM Correctable ECC Error (Priority 20) • 01010b - Receive Posted Request RAM Uncorrectable ECC Error (Priority 1) • 01011b - Receive Completion RAM Correctable ECC Error (Priority 21) • 01100b - Receive Completion RAM Uncorrectable ECC Error (Priority 2) • 01101b - Receive Posted Buffer Overflow Error (Priority 5) • 01110b - Receive Non Posted Buffer Overflow Error (Priority 6) • 01111b - Receive Completion Buffer Overflow Error (Priority 7) • 10000b - Flow Control Protocol Error (Priority 8) • 10001b - Transmit Parity Error Detected (Priority 4) • 10010b - Unexpected Completion Received (Priority 15) • 10011b - Completion Timeout Detected (Priority 14) • 10100b - AXI4ST RQ INTFC Packet Drop (Priority 17) • 10101b - AXI4ST CC INTFC Packet Drop (Priority 18) • 10110b - AXI4ST CQ Poisoned Drop (Priority 19) • 10111b - User Signaled Internal Correctable Error (Priority 23) • 11000b - User Signaled Internal Uncorrectable Error (Priority 0) • 11001b - 11111b - Reserved

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_local_error_valid pcie1_cfg_local_error_valid	O	1	Local Error Conditions Valid: Block activates this output for one cycle when any of the errors in <code>cfg_local_error_out[4:0]</code> are encountered. When driven 1b <code>cfg_local_error_out[4:0]</code> indicates local error type. Priority of error reporting (for the case of concurrent errors) is noted. Note: This signal may not work for all PCIe Link Width/Speed configurations. Do not rely solely on this signal to indicate an error. Alternatively, you can decode AER register to accurately detect errors.
pcie0_cfg_rx_pm_state pcie1_cfg_rx_pm_state	O	2	Current RX Active State Power Management L0s State: Encoding is listed below and valid when <code>cfg_ltssm_state</code> is indicating L0: <ul style="list-style-type: none"> • RX_NOT_IN_L0s = 0 • RX_L0s_ENTRY = 1 • RX_L0s_IDLE = 2 • RX_L0s_FTS = 3
pcie0_cfg_tx_pm_state pcie1_cfg_tx_pm_state	O	2	Current TX Active State Power Management L0s State: Encoding is listed below and valid when <code>cfg_ltssm_state</code> is indicating L0: <ul style="list-style-type: none"> • TX_NOT_IN_L0s = 0 • TX_L0s_ENTRY = 1 • TX_L0s_IDLE = 2 • TX_L0s_FTS = 3

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_ltssm_state pcie1_cfg_ltssm_state	O	6	LTSSM State. Shows the current LTSSM state: <ul style="list-style-type: none"> • 00: Detect.Quiet • 01: Detect.Active • 02: Polling.Active • 03: Polling.Compliance • 04: Polling.Configuration • 05: Configuration.Linkwidth.Start • 06: Configuration.Linkwidth.Accept • 07: Configuration.Lanenum.Accept • 08: Configuration.Lanenum.Wait • 09: Configuration.Complete • 0A: Configuration.Idle • 0B: Recovery.RcvrLock • 0C: Recovery.Speed • 0D: Recovery.RcvrCfg • 0E: Recovery.Idle • 10: L0 • 11-16: Reserved • 17: L1.Entry • 18: L1.Idle • 19-1A: Reserved • 20: Disabled • 21: Loopback_Entry_Master • 22: Loopback_Active_Master • 23: Loopback_Exit_Master • 24: Loopback_Entry_Slave • 25: Loopback_Active_Slave • 26: Loopback_Exit_Slave • 27: Hot_Reset • 28: Recovery_Equalization_Phase0 • 29: Recovery_Equalization_Phase1 • 2a: Recovery_Equalization_Phase2 • 2b: Recovery_Equalization_Phase3
pcie0_cfg_rcb_status	O	4	RCB Status. Provides the setting of the Read Completion Boundary (RCB) bit in the Link Control register of each physical function. In Endpoint mode, bit 0 indicates the RCB for Physical Function 0 (PF 0), bit 1 indicates the RCB for PF 1, and so on. In RC mode, bit 0 indicates the RCB setting of the Link Control register of the RP, bit 1 is reserved. For each bit, a value of 0 indicates an RCB of 64 bytes and a value of 1 indicates 128 bytes.

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_pl_status_change pcie1_cfg_pl_status_change	O	1	This output is used by the core in Root Port mode to signal one of the following link training-related events: <ul style="list-style-type: none"> The link bandwidth changed as a result of the change in the link width or operating speed and the change was initiated locally (not by the link partner), without the link going down. This interrupt is enabled by the Link Bandwidth Management Interrupt Enable bit in the Link Control register. The status of this interrupt can be read from the Link Bandwidth Management Status bit of the Link Status register; or The link bandwidth changed autonomously as a result of the change in the link width or operating speed and the change was initiated by the remote node. This interrupt is enabled by the Link Autonomous Bandwidth Interrupt Enable bit in the Link Control register. The status of this interrupt can be read from the Link Autonomous Bandwidth Status bit of the Link Status register; or The Link Equalization Request bit in the Link Status 2 register was set by the hardware because it received a link equalization request from the remote node. This interrupt is enabled by the Link Equalization Interrupt Enable bit in the Link Control 3 register. The status of this interrupt can be read from the Link Equalization Request bit of the Link Status 2 register. The pl_interrupt output is not active when the core is configured as an Endpoint.
pcie0_cfg_ext_tag_enable	O	1	Extended Tag Enable: Per function state of Device Control Register Ext Tag (8-Bit) Enable bit.
pcie0_cfg_atomic_requester_enable	O	4	Atomic Operation Requester Enable: Per function state of Device Control2 Register AtomicOp Requester Enable bit.
pcie0_cfg_10b_tag_requester_enable	O	4	10b Tag Requester Enable: Per function state of Device Control2 Register 10-Bit Tag Requester Enable bit.
pcie0_pcie_tfc_nph_av pcie1_pcie_tfc_nph_av	O	4	This output provides an indication of the currently available header credit for Non-Posted TLPs on the transmit side of the core. The user logic can check this output before transmitting a Non-Posted request on the requester request interface, to avoid blocking the interface when no credit is available. The encodings are: <ul style="list-style-type: none"> 0000: No credit available 0001: 1 credit available 0010: 2 credits available ... 1110: 14 credits available 1111: 15 or more credits available Because of pipeline delays, the value on this output can not include the credit consumed by the Non-Posted requests in the last eight cycles or less. The user logic must adjust the value on this output by the credit consumed by the Non-Posted requests it sent in the previous clock cycles, if any.

Table 20: Configuration Status Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_pcie_tfc_npd_av pcie1_pcie_tfc_npd_av	O	4	This output provides an indication of the currently available payload credit for Non-Posted TLPs on the transmit side of the core. The user logic checks this output before transmitting a Non-Posted request on the requester request interface, to avoid blocking the interface when no credit is available. The encodings are: <ul style="list-style-type: none"> • 0000: No credit available • 0001: 1 credit available • 0010: 2 credits available • ... • 1110: 14 or more credits available • 1111: 15 or more credits available Because of pipeline delays, the value on this output does not include the credit consumed by the Non-Posted requests sent by the user logic in the last eight clock cycles or less. The user logic must adjust the value on this output by the credit consumed by the Non-Posted requests it sent in the previous clock cycles, if any.
pcie0_rq_tag_av pcie1_rq_tag_av	O	4	This output provides an indication of the number of free tags available for allocation to Non-Posted requests on the PCIe master side of the core. The user logic checks this output before transmitting a Non-Posted request on the requester request interface, to avoid blocking the interface when no tags are available. The encodings are: <ul style="list-style-type: none"> • 0000: No tags available • 0001: 1 tag available • 0010: 2 tags available • ... • 1110: 14 tags available • 1111: 15 or more tags available Because of pipeline delays, the value on this output does not include the tags consumed by the Non-Posted requests sent by the user logic in the last 8 clock cycles or less. The user logic must adjust the value on this output by the number of Non-Posted requests it sent in the previous clock cycles, if any.

Configuration Received Message Interface

The Configuration Received Message interface indicates to the logic that a decodable message from the link, the parameters associated with the data, and the type of message have been received. The following table defines the ports in the Configuration Received Message interface of the core.

Note: Configuration Received Message interface is available only for PCIe Controller 0.

Table 21: Configuration Received Message Interface

Port	I/O	Width	Description
pcie0_cfg_msg_received	O	1	<p>Configuration Received a Decodable Message.</p> <p>The core asserts this output for one or more consecutive clock cycles when it has received a decodable message from the link. The duration of its assertion is determined by the type of message. The core transfers any parameters associated with the message on the pcie0_cfg_msg_data[7:0] output in one or more cycles when pcie0_cfg_msg_received is High. The following table lists the number of cycles of pcie0_cfg_msg_received assertion, and the parameters transferred on cfg_msg_data[7:0] in each cycle, for each type of message.</p> <p>The core inserts at least a one-cycle gap between two consecutive messages delivered on this interface when the pcie0_cfg_msg_received interface is enabled.</p> <p>The Configuration Received Message interface must be enabled during core configuration in the Vivado IDE.</p>
pcie0_cfg_msg_received_data	O	8	<p>This bus is used to transfer any parameters associated with the Received Message. The information it carries in each cycle for various message types is listed in the previous table.</p>
pcie0_cfg_msg_received_type	O	5	<p>Received message type.</p> <p>When pcie0_cfg_msg_received is High, these five bits indicate the type of message being signaled by the core. The various message types are listed in the previous table.</p>

Table 22: Message Type Encoding on Receive Message Interface

cfg_msg_received_type[4:0]	Message Type
0	ERR_COR
1	ERR_NONFATAL
2	ERR_FATAL
3	Assert_INTA
4	Deassert_INTA
5	Assert_INTB
6	Deassert_INTB
7	Assert_INTC
8	Deassert_INTC
9	Assert_INTD
10	Deassert_INTD
11	PM_PME
12	PME_TO_Ack
13	PME_Turn_Off
14	PM_Active_State_Nak
15	Set_Slot_Power_Limit
16	Latency Tolerance Reporting (LTR)

Table 22: Message Type Encoding on Receive Message Interface (cont'd)

cfg_msg_received_type[4:0]	Message Type
17	Reserved
18	Unlock
19	Vendor_Defined Type 0
20	Vendor_Defined Type 1
25 – 31	Reserved

Table 23: Message Parameters on Receive Message Interface

Message Type	Number of cycles of pcie0_cfg_msg_received assertion	Parameter transferred on pcie0_cfg_msg_received_data[7:0]
ERR_COR, ERR_NONFATAL, ERR_FATAL	2	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number
Assert_INTx, Deassert_INTx	2	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number
PM_PME, PME_TO_Ack, PME_Turn_off, PM_Active_State_Nak	2	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number
Set_Slot_Power_Limit	6	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number Cycle 3: bits [7:0] of payload Cycle 4: bits [15:8] of payload Cycle 5: bits [23:16] of payload Cycle 6: bits [31:24] of payload
Latency Tolerance Reporting (LTR)	6	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number Cycle 3: bits [7:0] of Snoop Latency Cycle 4: bits [15:8] of Snoop Latency Cycle 5: bits [7:0] of No-Snoop Latency Cycle 6: bits [15:8] of No-Snoop Latency
Unlock	2	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number
Vendor_Defined Type 0	4 cycles when no data present, 8 cycles when data present.	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number Cycle 3: Vendor ID[7:0] Cycle 4: Vendor ID[15:8] Cycle 5: bits [7:0] of payload Cycle 6: bits [15:8] of payload Cycle 7: bits [23:16] of payload Cycle 8: bits [31:24] of payload

Table 23: Message Parameters on Receive Message Interface (cont'd)

Message Type	Number of cycles of <code>pcie0_cfg_msg_received</code> assertion	Parameter transferred on <code>pcie0_cfg_msg_received_data[7:0]</code>
Vendor_Defined Type 1	4 cycles when no data present, 8 cycles when data present.	Cycle 1: Requester ID, Bus Number Cycle 2: Requester ID, Device/Function Number Cycle 3: Vendor ID[7:0] Cycle 4: Vendor ID[15:8] Cycle 5: bits [7:0] of payload Cycle 6: bits [15:8] of payload Cycle 7: bits [23:16] of payload Cycle 8: bits [31:24] of payload

Configuration Transmit Message Interface

The Configuration Transmit Message interface is used by the user application to transmit messages to the core. The user application supplies the transmit message type and data information to the core, which responds with the `done` signal. The following table defines the ports in the Configuration Transmit Message interface of the core.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 24: Configuration Transmit Message Interface

Port	I/O	Width	Description
<code>pcie0_cfg_msg_transmit</code> <code>pcie1_cfg_msg_transmit</code>	I	1	Configuration Transmit Encoded Message. This signal is asserted together with <code>cfg_msg_transmit_type</code> , which supplies the encoded message type and <code>cfg_msg_transmit_data</code> , which supplies optional data associated with the message, until <code>cfg_msg_transmit_done</code> is asserted in response.
<code>pcie0_cfg_msg_transmit_type</code> <code>pcie1_cfg_msg_transmit_type</code>	I	3	Configuration Transmit Encoded Message Type. Indicates the type of PCI Express message to be transmitted. Encodings supported are: <ul style="list-style-type: none"> • 000b: Latency Tolerance Reporting (LTR) • 001b: Optimized Buffer Flush/Fill (OBFF) • 010b: Set Slot Power Limit (SSPL) • 011b: Power Management (PM PME) • 100b -111b: Reserved

Table 24: Configuration Transmit Message Interface (cont'd)

Port	I/O	Width	Description
pcie0_cfg_msg_transmit_data pcie1_cfg_msg_transmit_data	I	32	Configuration Transmit Encoded Message Data. Indicates message data associated with particular message type. 000b: LTR <ul style="list-style-type: none"> cfg_msg_transmit_data[31] < Snoop Latency Req cfg_msg_transmit_data[30:29] <= Repurposing to pass PASID information. assigned to s_axis_rq_tuser[182:181] inside the IP. cfg_msg_transmit_data[28:26] < Snoop Latency Scale cfg_msg_transmit_data[25:16] < Snoop Latency Value cfg_msg_transmit_data[15] < No-Snoop Latency Requirement cfg_msg_transmit_data[14:13] <= Repurposing to pass PASID information. assigned to s_axis_rq_tuser[180:179] inside the IP. cfg_msg_transmit_data[12:10] < No-Snoop Latency Scale cfg_msg_transmit_data[9:0] < No-Snoop Latency Value 001b: Reserved. 010b: SSPL <ul style="list-style-type: none"> cfg_msg_transmit_data[9:0] < {Slot Power Limit Scale, Slot Power Limit Value} 011b: PM_PME <ul style="list-style-type: none"> cfg_msg_transmit_data[7:0] <= 8'h00-8'hFF PF0-3 - 8'h00-8'h03 Other encodings are reserved 100b - 111b: Reserved
pcie0_cfg_msg_transmit_done	O	1	Configuration Transmit Encoded Message Done. Asserted in response to cfg_msg_transmit assertion, for 1 cycle after the request is complete.

Configuration Flow Control Interface

The following table defines the ports in the Configuration Flow Control interface of the core.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 25: Configuration Flow Control Interface

Port	I/O	Width	Description
pcie0_cfg_fc_ph pcie1_cfg_fc_ph	O	8	Posted Header Flow Control Credits. This output provides the number of Posted Header Flow Control Credits. This multiplexed output can be used to bring out various flow control parameters and variables related to Posted Header Credit maintained by the core. The flow control information to bring out on this core is selected by the <code>cfg_fc_sel[2:0]</code> input.

Table 25: Configuration Flow Control Interface (cont'd)

Port	I/O	Width	Description
pcie0_cfg_fc_ph_scale pcie1_cfg_fc_ph_scale	O	2	Posted Header Flow Control Credits Scale This output provides the scale of Posted Header Flow Control Credit number (cfg_fc_ph). The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_pd pcie1_cfg_fc_pd	O	12	Posted Data Flow Control Credits. This output provides the number of Posted Data Flow Control Credits. This multiplexed output can be used to bring out various flow control parameters and variables related to Posted Data Credit maintained by the core. The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_pd_scale pcie1_cfg_fc_pd_scale	O	2	Posted Data Flow Control Credits Scale This output provides the scale of Posted Data Flow Control Credit number (cfg_fc_pd). The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_nph pcie1_cfg_fc_nph	O	8	Non-Posted Header Flow Control Credits. This output provides the number of Non-Posted Header Flow Control Credits. This multiplexed output can be used to bring out various flow control parameters and variables related to Non-Posted Header Credit maintained by the core. The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_nph_scale pcie1_cfg_fc_nph_scale	O	2	Non-Posted Header Flow Control Credits Scale This output provides the scale of Non-Posted Header Flow Control Credit number (cfg_fc_nph). The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_npd pcie1_cfg_fc_npd	O	12	Non-Posted Data Flow Control Credits. This output provides the number of Non-Posted Data Flow Control Credits. This multiplexed output can be used to bring out various flow control parameters and variables related to Non-Posted Data Credit maintained by the core. The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_npd_scale pcie1_cfg_fc_npd_scale	O	2	Non-Posted Data Flow Control Credits Scale This output provides the scale of Non-Posted Data Flow Control Credit number (cfg_fc_npd). The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_cplh	O	8	Completion Header Flow Control Credits. This output provides the number of Completion Header Flow Control Credits. This multiplexed output can be used to bring out various flow control parameters and variables related to Completion Header Credit maintained by the core. The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.
pcie0_cfg_fc_cplh_scale	O	2	Completion Header Flow Control Credit Scale This output provides the scale of Completion Header Flow Control Credit number (cfg_fc_cplh). The flow control information to bring out on this core is selected by the cfg_fc_sel[2:0] input.

Table 25: Configuration Flow Control Interface (cont'd)

Port	I/O	Width	Description
pcie0_cfg_fc_cpld	O	12	Completion Data Flow Control Credits. This output provides the number of Completion Data Flow Control Credits. This multiplexed output can be used to bring out various flow control parameters and variables related to Completion Data Credit maintained by the core. The flow control information to bring out on this core is selected by the <code>cfg_fc_sel[2:0]</code> .
pcie0_cfg_fc_cpld_scale	O	2	Completion Data Flow Control Credit Scale This output provides the scale of Completion Data Flow Control Credit number (<code>cfg_fc_cpld</code>). The flow control information to bring out on this core is selected by the <code>cfg_fc_sel[2:0]</code> input.
pcie0_cfg_fc_vc_sel pcie1_cfg_fc_vc_sel	I	1	Flow Control Informational Virtual Channel Select: Selects the Virtual Channel for the type of flow control information presented on the <code>trn_fc_*</code> signals. Possible values: 0b = VC0 1b = VC1
pcie0_cfg_fc_sel pcie1_cfg_fc_sel	I	3	Flow Control Informational Select. These inputs select the type of flow control to bring out on the <code>cfg_fc_*</code> outputs of the core. The various flow control parameters and variables that can be accessed for the different settings of these inputs are: <ul style="list-style-type: none"> • 000: Receive credit limit to link partner (<code>rlimit</code>) • 001: Transmit credits consumed (<code>tconsumed</code>) & wide posted and completion • 010: Receive credits consumed by link partner (<code>rconsumed</code>) • 011: Transmit credits consumed (<code>tconsumed</code>) wide nonposted and completion • 100: Transmit user credits available (<code>tlimit - tconsumed</code>) • 101: Transmit credit limit (<code>tlimit</code>) • 110: Transmit credits consumed (<code>tconsumed</code>) • 111: Receive free 32B word count This value represents the actual unused credits in the receiver FIFO, and the recommendation is to use it only as an approximate indication of receiver FIFO fullness, relative to the initial credit limit value advertized, such as, ¼ full, ½ full, ¾ full, full. Infinite credit for transmit credits available (<code>cfg_fc_sel == 3'b100</code>) is signaled as 8'h80, 12'h800 for header and data credits, respectively. For all other <code>cfg_fc_sel</code> selections, infinite credit is signaled as 8'h00, 12'h000, respectively, for header and data categories.

Configuration Control Interface

The Configuration Control interface signals allow a broad range of information exchange between the user application and the core. The user application uses this interface to do the following:

- Set the configuration space.
- Indicate if a correctable or uncorrectable error has occurred.
- Set the device serial number.
- Set the downstream bus, device, and function number.
- Receive per function configuration information.

This interface also provides handshaking between the user application and the core when a Power State change or function level reset occurs.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 26: Configuration Control Interface Port Descriptions

Port	I/O	Width	Description
pcie0_cfg_hot_reset_in pcie1_cfg_hot_reset_in	I	1	Configuration Hot Reset In In RP mode, assertion transitions LTSSM to hot reset state, active-High.
pcie0_cfg_hot_reset_out pcie1_cfg_hot_reset_out	O	1	Configuration Hot Reset Out In EP mode, assertion indicates that EP has transitioned to the hot reset state, active-High.
pcie0_cfg_power_state_change_ack pcie1_cfg_power_state_change_ack	I	1	Configuration Power State Ack You must assert this input to the core for one cycle in response to the assertion of <code>pcie(n)_cfg_power_state_change_interrupt</code> , when it is ready to transition to the low-power state requested by the configuration write request. The user application can permanently hold this input High if it does not need to delay the return of the completions for the configuration write transactions, causing power-state changes.
pcie0_cfg_power_state_change_interrupt pcie1_cfg_power_state_change_interrupt	O	1	Power State Change Interrupt The core asserts this output when the power state of a physical or virtual function is being changed to the D1 or D3 states by a write into its Power Management Control register. The core holds this output High until the user application asserts the <code>pcie(n)_cfg_power_state_change_ack</code> input to the core. While <code>pcie(n)_cfg_power_state_change_interrupt</code> remains High, the core does not return completions for any pending configuration read or write transaction received by the core. The purpose is to delay the completion for the configuration write transaction that caused the state change until the user application is ready to transition to the low-power state. When <code>pcie(n)_cfg_power_state_change_interrupt</code> is asserted, the function number associated with the configuration write transaction is provided on the <code>pcie(n)_cfg_ext_function_number[7:0]</code> output. When the user application asserts <code>pcie(n)_cfg_power_state_change_ack</code> , the new state of the function that underwent the state change is reflected on <code>pcie(n)_cfg_function_power_state</code> (for PFs) or the <code>pcie(n)_cfg_vf_power_state</code> (for VFs) outputs of the core.

Table 26: Configuration Control Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_err_cor_in pcie1_cfg_err_cor_in	I	1	<p>Correctable Error Detected</p> <p>The user application activates this input for one cycle to indicate a correctable error detected within the user logic that needs to be reported as an internal error through the PCI Express Advanced Error Reporting (AER) mechanism. In response, the core sets the Corrected Internal Error Status bit in the AER Correctable Error Status register of all enabled functions, and also sends an error message if enabled to do so. This error is not considered function-specific.</p>
pcie0_cfg_err_cor_out pcie1_cfg_err_cor_out	O	1	<p>Correctable Error Detected</p> <p>In the Endpoint mode, the Block activates this output for one cycle when it has detected a correctable error and its reporting is not masked. When multiple functions are enabled, this is the logical OR of the correctable error status bits in the Device Status Registers of all functions.</p>
pcie0_cfg_err_fatal_out pcie1_cfg_err_fatal_out	O	1	<p>Fatal Error Detected</p> <p>In the Endpoint mode, the block activates this output for one cycle when it has detected a fatal error and its reporting is not masked. When multiple functions are enabled, this output is the logical OR of the fatal error status bits in the Device Status Registers of all functions.</p> <p>In the Root Port mode, this output is activated on detection of a local fatal error, when its reporting is not masked. This output does not respond to any errors signaled by remote devices using PCI Express error messages. These error messages are delivered to the user through the message interface.</p>
pcie0_cfg_err_nonfatal_out pcie1_cfg_err_nonfatal_out	O	1	<p>Non Fatal Error Detected</p> <p>In the Endpoint mode, the block activates this output for one cycle when it has detected a non fatal error and its reporting is not masked. When multiple functions are enabled, this output is the logical OR of the non fatal error status bits in the Device Status Registers of all functions.</p> <p>In the Root Port mode, this output is activated on detection of a local non fatal error, when its reporting is not masked. This output does not respond to any errors signaled by remote devices using PCI Express error messages. These error messages are delivered through the message interface.</p>
pcie0_cfg_err_uncor_in pcie1_cfg_err_uncor_in	I	1	<p>Uncorrectable Error Detected</p> <p>The user application activates this input for one cycle to indicate an uncorrectable error detected within the user logic that needs to be reported as an internal error through the PCI Express Advanced Error Reporting mechanism. In response, the core sets the uncorrected Internal Error Status bit in the AER Uncorrectable Error Status register of all enabled functions, and also sends an error message if enabled to do so. This error is not considered function-specific.</p>

Table 26: Configuration Control Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_flr_done pcie1_cfg_flr_done	I	4	Function Level Reset Complete The user application must assert this input when it has completed the reset operation of the Virtual Function. This causes the core to deassert <code>pcie(n)_cfg_flr_in_process</code> for physical function <i>i</i> and to re-enable configuration accesses to the physical function. The core will issue CRS to configuration requests to a particular Physical Function till <code>pcie(n)_cfg_flr_done</code> is not asserted when <code>pcie(n)_cfg_flr_in_process = 1</code> for that Physical Function.
pcie0_cfg_vf_flr_done pcie1_cfg_vf_flr_done	I	1	Function Level Reset for Virtual Function is Complete The user application must assert this input when it has completed the reset operation of the Virtual Function. This causes the core to deassert <code>pcie(n)_cfg_vf_flr_in_process</code> for function <i>i</i> and to re-enable configuration accesses to the virtual function. The core will issue CRS to configuration requests to a particular Virtual Function till <code>pcie(n)_cfg_vf_flr_done</code> is not asserted when <code>pcie(n)_cfg_vf_flr_in_process = 1</code> for that Virtual Function.
pcie0_cfg_vf_flr_func_num pcie1_cfg_vf_flr_func_num	I	8	Function Level Reset for Virtual Function <i>i</i> is Complete. This user application drives a valid Virtual Function number on this input along with asserting <code>pcie(n)_cfg_vf_flr_done</code> when the reset operation of Virtual Function <i>i</i> completes. Valid entries are 8'h04-8'hFF for VF0-VF251. Values 8'h00-8'h03 are reserved.
pcie0_cfg_flr_in_process	O	4	Function Level Reset In Process The core asserts bit <i>i</i> of this bus when the host initiates a reset of physical function <i>i</i> through its FLR bit in the configuration space. The core continues to hold the output High until the user sets the corresponding <code>pcie(n)_cfg_flr_done</code> input for the corresponding physical function to indicate the completion of the reset operation.
pcie0_cfg_vf_flr_in_process pcie1_cfg_vf_flr_in_process	O	252	Function Level Reset In Process for Virtual Function The core asserts bit <i>i</i> of this bus when the host initiates a reset of virtual function <i>i</i> through its FLR bit in the configuration space. The core continues to hold the output High until the user sets the <code>pcie(n)_cfg_vf_flr_done</code> input and drives <code>pcie(n)_cfg_vf_flr_func_num</code> with the corresponding function to indicate the completion of the reset operation.

Configuration Interrupt Controller Interface

The Configuration Interrupt Controller interface allows the user application to set Legacy PCIe interrupts, MSI interrupts, or MSI-X interrupts. The core provides the interrupt status on the configuration interrupt sent and fail signals. The following tables define the interface ports associated with the Configuration Interrupt Controller interface of the core.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Legacy Interrupt Interface

Table 27: Legacy Interrupt Interface Port Descriptions

Name	I/O	Width	Description
pcie0_cfg_interrupt_int pcie1_cfg_interrupt_int	I	4	Configuration INTx Vector: When the core is configured as EP, these four inputs are used by the user application to signal an interrupt from any of its PCI Functions to the RC using the Legacy PCI Express Interrupt Delivery mechanism of PCI Express. These four inputs correspond to INTA, INTB, INTC, and INTD of the PCI bus, respectively. Asserting one of these signals causes the core to send out an Assert_INTx message, and deasserting the signal causes the core to transmit a Deassert_INTx message.
pcie0_cfg_interrupt_sent pcie1_cfg_interrupt_sent	O	1	Configuration INTx Sent: A pulse on this output indicates that the core has sent an INTx Assert or Deassert message in response to a change in the state of one of the pcie(n)_cfg_interrupt_int inputs.
pcie0_cfg_interrupt_pending pcie1_cfg_interrupt_pending	I	4	Configuration INTx Interrupt Pending: Per Function indication of a pending interrupt from the user. pcie(n)_cfg_interrupt_pending[0] corresponds to Function #0. Each of these inputs is connected to the Interrupt Pending bits of the PCI Status Register of the corresponding Function.

MSI Interrupt Interface

Table 28: MSI Interrupt Interface Port Descriptions

Name	I/O	Width	Description
pcie0_cfg_msi_enable	O	4	Configuration Interrupt MSI Function Enabled Indicates that the Message Signaling Interrupt (MSI) messaging is enabled, per Function. These outputs reflect the setting of the MSI Enable bits in the MSI Control Register of Physical Functions 0 - 3.
pcie0_cfg_msi_mint_vector pcie1_cfg_msi_mint_vector	I	32	Configuration Interrupt MSI Vector When configured in the Endpoint mode to support MSI interrupts, these inputs are used to signal the 32 distinct interrupt conditions associated with a PCI Function (Physical or Virtual) from the user logic to the core. The Function number must be specified on the input pcie(n)_cfg_msi_function_number. After placing the Function number on the input pcie(n)_cfg_msi_function_number, the user logic must activate one of these signals for one cycle to transmit an interrupt. The user logic must not activate more than one of the 32 interrupt inputs in the same cycle. The core internally registers the interrupt condition on the 0-to-1 transition of any bit in pcie(n)_cfg_msi_mint_vector. After asserting an interrupt, the user logic must wait for the pcie0_cfg_msi_sent or pcie0_cfg_msi_fail indication from the core before asserting a new interrupt.

Table 28: MSI Interrupt Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msi_function_number pcie1_cfg_msi_function_number	I	8	Configuration MSI Initiating Function Indicates the Endpoint Function # initiating the MSI interrupt. <ul style="list-style-type: none"> 8'h00 – 8'h03: PF 0 – PF 3 8'h04 – 8'hFF: VF 0 – VF 252 Other encodings are reserved.
pcie0_cfg_msi_sent	O	1	Configuration Interrupt MSI Interrupt Sent The core generates a one-cycle pulse on this output to signal that an MSI interrupt message has been transmitted on the link. The user logic must wait for this pulse before signaling another interrupt condition to the core.
pcie0_cfg_msi_fail	O	1	Configuration Interrupt MSI Interrupt Operation Failed A one-cycle pulse on this output indicates that an MSI interrupt message was aborted before transmission on the link. The user logic must retransmit the MSI or MSI-X interrupt in this case.
pcie0_cfg_msi_mmenable	O	12	Configuration Interrupt MSI Function Multiple Message Enable When the core is configured in the Endpoint mode to support MSI interrupts, these outputs are driven by the 'Multiple Message Enable' bits of the MSI Control Register associated with Physical Functions. These bits encode the number of allocated MSI interrupt vectors for the corresponding Function. Bits [2:0] correspond to Physical Function 0, bits [5:3] correspond to PF 1, and so on. The valid encodings of the 3 bits are: <ul style="list-style-type: none"> 000b: 1 vector 001b: 2 vectors 010b: 4 vectors 011b: 8 vectors 100b: 16 vectors 101b: 32 vectors
pcie0_cfg_msi_pending_status pcie1_cfg_msi_pending_status	I	32	Configuration MSI Interrupt Pending Status These inputs are provided for the user to indicate the interrupt pending status of the MSI interrupts associated with the Physical Functions. When the status of a MSI interrupt associated with a PF changes, the user must place the new interrupt status on these inputs, along with the corresponding Function number on the pcie(n)_cfg_msi_pending_status_function_num input, and activate the pcie(n)_cfg_msi_pending_status_data_enable input for one cycle. The core then latches the new status in its MSI Pending Bits Register of the corresponding Physical Function.

Table 28: MSI Interrupt Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msi_pending_status_function_num pcie1_cfg_msi_pending_status_function_num	I	2	Configuration Interrupt MSI Pending Target Function Number <ul style="list-style-type: none"> • 00 = PF 0 • 01 = PF 1 • 10 = PF 2 • 11 = PF 3 This input is used to identify the Function number when the user places interrupt status on the pcie(n)_cfg_msi_pending_status inputs.
pcie0_cfg_msi_pending_status_data_enable pcie1_cfg_msi_pending_status_data_enable	Input	1	Configuration Interrupt MSI Pending Data Valid The user application asserts this signal together with pcie(n)_cfg_msi_pending_status and pcie(n)_cfg_msi_pending_status_function_num values to update the MSI Pending Bits in the corresponding function.
pcie0_cfg_msi_mask_update	O	1	Configuration Interrupt MSI Function Mask Updated The SR-IOV core asserts this for 1 cycle when the MSI Mask Register of any enabled PFs has changed its value. The user can then read the new mask settings from the pcie(n)_cfg_msi_data outputs.
pcie0_cfg_msi_select pcie1_cfg_msi_select	Input	2	Configuration Interrupt MSI Select These inputs are used to select the Function number for reading the MSI Mask Register setting from the core. Values 0 – 3 correspond to Physical Functions 0 – 3, respectively. The mask MSI Mask Register contents of the selected PF appear on the output pcie(n)_cfg_msi_data after one cycle.
pcie0_cfg_msi_data pcie1_cfg_msi_data	O	32	Configuration Interrupt MSI Data These output reflect the MSI Mask Register setting of the Physical Function selected by the pcie(n)_cfg_msi_select input.
pcie0_cfg_msi_attr pcie1_cfg_msi_attr	I	3	Configuration Interrupt MSI/MSI-X TLP Attribute These bits enable you to set the Attribute bits that are used for both MSI and MSI-X interrupt requests. <ul style="list-style-type: none"> • Bit 0 is the No Snoop bit. • Bit 1 is the Relaxed Ordering bit. • Bit 2 is the ID-Based Ordering bit. The core samples these bits on a 0-to-1 transition on pcie(n)_cfg_msi_mint_vector bits (when using MSI) or pcie(n)_cfg_msix_int (when using MSI-X).

MSI-X Interrupt Interface

Table 29: MSI-X Interrupt External Interface Port Descriptions

Name	I/O	Width	Description
pcie0_cfg_msix_enable	O	4	Configuration Interrupt MSI-X Function Enabled These outputs reflect the setting of the MSI-X Enable bits of the MSI-X Control Register of Physical Functions 0 – 3.

Table 29: MSI-X Interrupt External Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msix_function_number pcie1_cfg_msix_function_number	I	8	Configuration MSI-X Initiating Function Indicates the Endpoint Function # initiating the MSI-X interrupt. <ul style="list-style-type: none"> 8'h00- 8'h03: PF 0 – PF 3 8'h04- 8'hFF: VF 0 – VF 252 Other encodings are reserved.
pcie0_cfg_msix_sent	O	1	Configuration Interrupt MSI-X Interrupt Sent The core generates a one-cycle pulse on this output to signal that an MSI-X interrupt message has been transmitted on the link. The user logic must wait for this pulse before signaling another interrupt condition to the core.
pcie0_cfg_msix_fail	O	1	Configuration Interrupt MSI-X Interrupt Operation Failed A one-cycle pulse on this output indicates that an MSI-X interrupt message was aborted before transmission on the link. The user logic must retransmit the MSI-X interrupt in this case.
pcie0_cfg_msix_mask	O	4	Configuration Interrupt MSI-X Function Mask These outputs reflect the setting of the MSI-X Function Mask bits of the MSI-X Control Register of Physical Functions 0 – 3.
pcie0_cfg_msix_vf_enable pcie1_cfg_msix_vf_enable	O	252	Configuration Interrupt MSI-X Enable from VFs These outputs reflect the setting of the MSI-X Enable bits of the MSI-X Control Register of Virtual Functions 0 – 251.
pcie0_cfg_msix_vf_mask pcie1_cfg_msix_vf_mask	O	252	Configuration Interrupt MSI-X VF Mask These outputs reflect the setting of the MSI-X Function Mask bits of the MSI-X Control Register of Virtual Functions 0 – 251.
pcie0_cfg_msix_address pcie1_cfg_msix_address	I	64	Configuration Interrupt MSI-X Address When the core is configured to support MSI-X interrupts and when the MSI-X Table is implemented in user memory, this bus is used by the user logic to communicate the address to be used to generate an MSI-X interrupt.
pcie0_cfg_msix_data pcie1_cfg_msix_data	I	32	Configuration Interrupt MSI-X Data When the core is configured to support MSI-X interrupts and when the MSI-X Table is implemented in user memory, this bus is used by the user logic to communicate the data to be used to generate an MSI-X interrupt.

Table 29: MSI-X Interrupt External Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msix_int_vector pcie1_cfg_msix_int_vector	I	1	<p>Configuration Interrupt MSI-X Data Valid</p> <p>The assertion of this signal by the user indicates a request from the user to send an MSI-X interrupt. The user must place the identifying information on the designated inputs before asserting this interrupt.</p> <p>When the MSI-X Table and Pending Bit Array are implemented in user memory, the identifying information consists of the memory address, data, and the originating Function number for the interrupt.</p> <p>These must be placed on the pcie(n)_cfg_msix_address[63:0], pcie(n)_cfg_msix_data[31:0], and pcie(n)_cfg_msix_function_number[7:0], respectively. The core internally registers these parameters on the 0-to-1 transition of pcie(n)_cfg_msix_int_vector.</p> <p>When the MSI-X Table and Pending Bit Array are implemented by the core, the identifying information consist so the originating Function number for the interrupt and the interrupt vector.</p> <p>These must be placed on pcie(n)_cfg_msix_function_number[7:0] and pcie(n)_cfg_msix_int_vector[31:0], respectively.</p> <p>Bit i of pcie(n)_cfg_msix_mint_vector[31:0] represents interrupt vector i, and only one of the bits of this bus can be set to 1 when asserting pcie(n)_cfg_msix_int_vector.</p> <p>After asserting an interrupt, the user logic must wait for the pcie(n)_cfg_msix_sent or pcie0_cfg_msix_fail indication from the core before asserting a new interrupt.</p>

Table 30: MSI-X Interrupt Internal Interface Port Descriptions

Name	I/O	Width	Description
pcie0_cfg_msix_mint_vector pcie1_cfg_msix_mint_vector	I	32	<p>Configuration Interrupt MSI-X Vector</p> <p>When configured in the Endpoint mode to support MSI-X interrupts, these inputs are used to signal the 32 distinct interrupt conditions associated with a PCI Function (Physical or Virtual) from the user logic to the core. The Function number must be specified on the input pcie(n)_cfg_msix_function_number. After placing the Function number on the input pcie(n)_cfg_msix_function_number, the user logic must activate one of these signals for one cycle to transmit an interrupt. The user logic must not activate more than one of the 32 interrupt inputs in the same cycle. The core internally registers the interrupt condition on the 0-to-1 transition of any bit in pcie(n)_cfg_msix_mint_vector. After asserting an interrupt, the user logic must wait for the pcie0_cfg_msix_sent or pcie0_cfg_msix_fail indication from the core before asserting a new interrupt.</p>
pcie0_cfg_msix_function_number pcie1_cfg_msix_function_number	I	8	<p>Configuration MSI-X Initiating Function</p> <p>Indicates the Endpoint Function # initiating the MSI-X interrupt.</p> <ul style="list-style-type: none"> 8'h00– 8'h03: PF 0 – PF 3 8'h04– 8'hFF: VF 0 – VF 252 Other encodings are reserved.

Table 30: MSI-X Interrupt Internal Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msix_sent	O	1	Configuration Interrupt MSI-X Interrupt Sent The core generates a one-cycle pulse on this output to signal that an MSI-X interrupt message has been transmitted on the link. The user logic must wait for this pulse before signaling another interrupt condition to the core.
pcie0_cfg_msix_fail	O	1	Configuration Interrupt MSI-X Interrupt Operation Failed A one-cycle pulse on this output indicates that an MSI-X interrupt message was aborted before transmission on the link. The user logic must retransmit the MSI-X interrupt in this case.
pcie0_cfg_msix_int_vector pcie1_cfg_msix_int_vector	I	1	Configuration Interrupt MSI-X Data Valid The assertion of this signal by the user indicates a request from the user to send an MSI-X interrupt. The user must place the identifying information on the designated inputs before asserting this interrupt. When the MSI-X Table and Pending Bit Array are implemented in user memory, the identifying information consists of the memory address, data, and the originating Function number for the interrupt. These must be placed on the pcie(n)_cfg_msix_address[63:0], pcie(n)_cfg_msix_data[31:0], and pcie(n)_cfg_msix_function_number[7:0], respectively. The core internally registers these parameters on the 0-to-1 transition of pcie(n)_cfg_msix_int_vector. When the MSI-X Table and Pending Bit Array are implemented by the core, the identifying information consist so the originating Function number for the interrupt and the interrupt vector. These must be placed on pcie(n)_cfg_msix_function_number[7:0] and pcie(n)_cfg_msix_int_vector[31:0], respectively. Bit i of pcie(n)_cfg_msix_mint_vector[31:0] represents interrupt vectori, and only one of the bits of this bus can be set to 1 when asserting pcie(n)_cfg_msix_int_vector. After asserting an interrupt, the user logic must wait for the pcie(n)_cfg_msix_sent or pcie0_cfg_msix_fail indication from the core before asserting a new interrupt.
pcie0_cfg_msix_enable	O	4	Configuration Interrupt MSI-X Function Enabled These outputs reflect the setting of the MSI-X Enable bits of the MSI-X Control Register of Physical Functions 0 – 3.
pcie0_cfg_msix_mask	O	4	Configuration Interrupt MSI-X Function Mask These outputs reflect the setting of the MSI-X Function Mask bits of the MSI-X Control Register of Physical Functions 0 – 3.
pcie0_cfg_msix_vf_enable pcie1_cfg_msix_vf_enable	O	252	Configuration Interrupt MSI-X Enable from VFs These outputs reflect the setting of the MSI-X Enable bits of the MSI-X Control Register of Virtual Functions 0 – 251.
pcie0_cfg_msix_vf_mask pcie1_cfg_msix_vf_mask	O	252	Configuration Interrupt MSI-X VF Mask These outputs reflect the setting of the MSI-X Function Mask bits of the MSI-X Control Register of Virtual Functions 0 – 251.

Table 30: MSI-X Interrupt Internal Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msix_vec_pending pcie1_cfg_msix_vec_pending	I	2	<p>Configuration Interrupt MSI-X Pending Bit Query/Clear</p> <p>These mode bits are used only when the core is configured to include the MSI-X Table and Pending Bit Array. These two bits are set when asserting <code>pcie(n)_cfg_msix_int_vector</code> to send an MSI-X interrupt, to perform certain actions on the MSI-X Pending Bit associated with the selected Function and interrupt vector. The various modes are:</p> <ul style="list-style-type: none"> • 00b: Normal interrupt generation. If the Mask bit associated with the vector was 0 when <code>pcie(n)_cfg_msix_int_vector</code> was asserted, the core transmits the MSI-X request TLP on the link. If the Mask bit was 1, the core does not immediately send the interrupt, but instead sets the Pending Bit associated with the interrupt vector in its MSI-X Pending Bit Array (and subsequently transmits the MSI-X request TLP when the Mask clears). In both cases, the core asserts <code>pcie(n)_cfg_msix_sent</code> for one cycle to indicate that the interrupt request was accepted. The user can distinguish these two cases by sampling the <code>pcie(n)_cfg_msix_vec_pending_status</code> output, which reflects the current setting of the MSI-X Pending Bit corresponding to the interrupt vector. • 01b: Pending Bit Query. In this mode, the core treats the assertion of one of the bits of <code>pcie(n)_cfg_msix_mint_vector</code> as a query for the status of its Pending Bit. The user must also place the Function number of the Pending Bit being queried on the <code>pcie(n)_cfg_msix_function_number</code> input. The core does not transmit a MSI-X request in response, but asserts <code>pcie(n)_cfg_msix_sent</code> for one cycle, along with the status of the MSI-X Pending Bit on the <code>pcie(n)_cfg_msix_vec_pending_status</code> output. • 10b: Pending Bit Clear. In this mode, the core treats the assertion of one of the bits of <code>pcie(n)_cfg_msix_int_vector</code> as a request to clear its Pending Bit. The user must also place the Function number of the Pending Bit being queried on the <code>pcie(n)_cfg_msix_function_number</code> input. The core does not transmit a MSI-X request in response, but clears the MSI-X Pending Bit of the vector (if it is set), and activates <code>pcie(n)_cfg_msix_sent</code> for one cycle as the acknowledgment. The core also provides the previous state of the MSI-X Pending Bit on the <code>pcie(n)_cfg_msix_vec_pending_status</code> output, which can be sampled by the user to determine if the Pending Bit was cleared by the core before the user request (because the pending interrupt was actually transmitted). This mode can be used to implement a polling mode for MSI-X interrupts, where the interrupt is permanently masked and the software polls the Pending Bit to detect and service the interrupt. After each interrupt is serviced, the Pending Bit can be cleared through this interface.

Table 30: MSI-X Interrupt Internal Interface Port Descriptions (cont'd)

Name	I/O	Width	Description
pcie0_cfg_msix_vec_pending_status	O	1	Configuration Interrupt MSI-X Pending Bit Status This output provides the status of the Pending Bit associated with an MSI-X interrupt, in response to query using the pcie(n)_cfg_msix_vec_pending input. It is active only when the core is configured to include the MSI-X Table and Pending Bit Array.
pcie0_cfg_msi_attr pcie1_cfg_msi_attr	I	3	Configuration Interrupt MSI/MSI-X TLP Attribute These bits enable you to set the Attribute bits that are used for both MSI and MSI-X interrupt requests. <ul style="list-style-type: none"> • Bit 0 is the No Snoop bit. • Bit 1 is the Relaxed Ordering bit. • Bit 2 is the ID-Based Ordering bit. The core samples these bits on a 0-to-1 transition on pcie(n)_cfg_msi_mint_vector bits (when using MSI) or pcie(n)_cfg_msix_int_vector (when using MSI-X).
pcie0_cfg_msi_tph_present pcie1_cfg_msi_tph_present	I	1	Configuration Interrupt MSI/MSI-X TPH Present Indicates the presence of an optional Transaction Processing Hint (TPH) in the MSI/MSI-X interrupt request. The user application must set this bit while asserting pcie(n)_cfg_msi_mint_vector bits (when using MSI), or pcie(n)_cfg_msix_int_vector (when using MSI-X), if it is including a TPH in the MSI or MSI-X transaction.
pcie0_cfg_msi_tph_type pcie1_cfg_msi_tph_type	I	2	Configuration Interrupt MSI/MSI-X TPH Type When pcie(n)_cfg_msi_tph_present is 1'b1, these two bits are used to supply the 2-bit type associated with the Hint. The user application must set these bits on 0-to-1 transition on any bit of pcie(n)_cfg_msi_mint_vector or pcie(n)_cfg_msix_int_vector, depending on whether MSI or MSI-X interrupts are being used.
pcie0_cfg_msi_tph_st_tag pcie1_cfg_msi_tph_st_tag	I	8	Configuration Interrupt MSI/MSI-X TPH Steering Tag When pcie(n)_cfg_msi_tph_present is asserted, the SteeringTag associated with the Hint must be placed on pcie(n)_cfg_msi_tph_st_tag[7:0]. The core samples these bits on 0-to-1 transition on any bit of pcie(n)_cfg_msi_mint_vector or pcie(n)_cfg_msix_int_vector, depending on whether MSI or MSI-X interrupts are being used.

Configuration Extend Interface

The Configuration Extend interface allows the core to transfer configuration information with the user application when externally implemented configuration registers are implemented. The following table defines the ports in the Configuration Extend interface of the core.

Note: The pcie0* signals map to PCIe Controller 0 and pcie1* signals map to PCIe Controller 1 in the port descriptions below.

Table 31: Configuration Extend Interface Port Descriptions

Port	I/O	Width	Description
pcie0_cfg_ext_read_received pcie1_cfg_ext_read_received	O	1	Configuration Extend Read Received. The Block asserts this output when it has received a configuration read request from the link. Set when PCI Express Extended Configuration Space Enable is selected in User Defined Configuration Capabilities in core configuration in the Vivado IDE. All received configuration reads with pcie(n)_cfg_ext_register_number in the following ranges are considered to be the PCIe Extended Configuration Space: <ul style="list-style-type: none"> • 0xB0 - 0xBF: When Legacy Extended Configuration Space Enable is set in Vivado IP catalog. • E00H - FFCH: When Extended Small is selected in the PCI Express Extended Configuration Space Enable option in the IP catalog. • 600H - FFCH: When Extended Large is selected in the PCI Express Extended Configuration Space Enable option. All received configuration reads regardless of address will be indicated by 1 cycle assertion of pcie(n)_cfg_ext_read_received, and valid data is driven on pcie(n)_cfg_ext_register_number and pcie(n)_cfg_ext_function_number. Only received configuration reads within the aforementioned ranges need to be responded by User Application outside of the IP.
pcie0_cfg_ext_write_received pcie1_cfg_ext_write_received	O	1	Configuration Extend Write Received. The Block asserts this output when it has received a configuration write request from the link. Set when PCI Express Extended Configuration Space Enable is selected in User Defined Configuration Capabilities in the core configuration in the Vivado IDE. Data corresponding to all received configuration writes with pcie(n)_cfg_ext_register_number in the range 0xb0-0xbf is presented on pcie(n)_cfg_ext_register_number, pcie(n)_cfg_ext_function_number, pcie(n)_cfg_ext_write_data and pcie(n)_cfg_ext_write_byte_enable. All received configuration writes with pcie(n)_cfg_ext_register_number in the 0xE80-0xFFFF range are presented on pcie(n)_cfg_ext_register_number, pcie(n)_cfg_ext_function_number, pcie(n)_cfg_ext_wrt_data and pcie(n)_cfg_ext_write_byte_enable.
pcie0_cfg_ext_register_number pcie1_cfg_ext_register_number	O	10	Configuration Extend Register Number The 10-bit address of the configuration register being read or written. The data is valid when pcie(n)_cfg_ext_read_received or pcie(n)_cfg_ext_write_received is High.
pcie0_cfg_ext_function_number pcie1_cfg_ext_function_number	O	8	Configuration Extend Function Number The 8-bit function number corresponding to the configuration read or write request. The data is valid when pcie(n)_cfg_ext_read_received or pcie(n)_cfg_ext_write_received is High.

Table 31: Configuration Extend Interface Port Descriptions (cont'd)

Port	I/O	Width	Description
pcie0_cfg_ext_write_data pcie1_cfg_ext_write_data	O	32	Configuration Extend Write Data Data being written into a configuration register. This output is valid when pcie(n)_cfg_ext_write_received is High.
pcie0_cfg_ext_write_byte_enable pcie1_cfg_ext_write_byte_enable	O	4	Configuration Extend Write Byte Enable Byte enables for a configuration write transaction.
pcie0_cfg_ext_read_data pcie1_cfg_ext_read_data	I	32	Configuration Extend Read Data You can provide data from an externally implemented configuration register to the core through this bus. The core samples this data on the next positive edge of the clock after it sets pcie(n)_cfg_ext_read_received High, if you have set pcie(n)_cfg_ext_read_data_valid.
pcie0_cfg_ext_read_data_valid pcie1_cfg_ext_read_data_valid	I	1	Configuration Extend Read Data Valid The user application asserts this input to the core to supply data from an externally implemented configuration register. The core samples this input data on the next positive edge of the clock after it sets pcie(n)_cfg_ext_read_received High. The core expects the assertions of this signal within 262144 ('h4_0000) clock cycles of user clock after receiving the read request on pcie(n)_cfg_ext_read_received signal. If no response is received by this time, the core will send auto-response with 'h0 payload, and the user application must discard the response and terminate that particular request immediately.

Configuration VC1 Status Interface

The Configuration VC1 Status interface is used to determine the VC1 enablement or disablement by the software, and determine the VC1 resource status.

Note: *cfg_vc1* ports are applicable only for PCIe Controller 0.

Table 32: Configuration VC1 Status Interface Port Descriptions

Port	I/O	Width	Description
pcie0_cfg_vc1_enable	O	1	Configuration VC1 Enable: VC1 Resource Control Register:VC Enable bit. When 1b, indicates software has enabled VC1 operation. When 0b, indicates that VC1 is disabled by software.
pcie0_cfg_vc1_negotiation_pending	O	1	Configuration VC1 Negotiation Pending: VC1 Resource Status Register: VC Negotiation Pending bit. When 1b, indicates VC1 negotiation (initialization or disabling) is in pending state.

Configuration PASID Interface

The Configuration PASID interface is used to determine the enablement of the PASID per function status by the software.

Note: The `pcie0*` signals map to PCIe Controller 0 and `pcie1*` signals map to PCIe Controller 1 in the port descriptions below.

Table 33: Configuration PASID Interface Port Descriptions

Port	I/O	Width	Description
<code>pcie0_cfg_pasid_enable</code> <code>pcie1_cfg_pasid_enable</code>	O	4	Configuration PASID Enable: Per Function PASID Enable.
<code>pcie0_cfg_pasid_exec_permission_enable</code> <code>pcie1_cfg_pasid_exec_permission_enable</code>	O	4	Configuration PASID Exec Permission Enable: Per Function PASID Exec Permission Enable.
<code>pcie0_cfg_pasid_privil_mode_enable</code> <code>pcie1_cfg_pasid_privil_mode_enable</code>	O	4	Configuration PASID Privil Mode Enable: Per Function PASID Privil Mode Enable.

Ports Not Available in PCIe Controller 1

The following output ports are *not* available in the PCIe Controller 1. A similarly named port is available only for the PCIe Controller 0 (starting with `pcie0_`).

- `pcie1_cfg_max_payload`
- `pcie1_cfg_max_read_req`
- `pcie1_cfg_function_status`
- `pcie1_cfg_function_power_state`
- `pcie1_cfg_rcb_status`
- `pcie1_cfg_atomic_requester_enable`
- `pcie1_cfg_10b_tag_requester_enable`
- `pcie1_cfg_ext_tag_enable`
- `pcie1_cfg_vc1_enable`
- `pcie1_cfg_vc1_negotiation_pending`
- `pcie1_cfg_msg_received`
- `pcie1_cfg_msg_received_data`
- `pcie1_cfg_msg_received_type`
- `pcie1_cfg_msg_transmit_done`
- `pcie1_cfg_fc_cplh`
- `pcie1_cfg_fc_cplh_scale`
- `pcie1_cfg_fc_cpld`
- `pcie1_cfg_fc_cpld_scale`
- `pcie1_cfg_flr_in_process`

- pcie1_cfg_interrupt_msi_enable
- pcie1_cfg_interrupt_msi_sent
- pcie1_cfg_interrupt_msi_fail
- pcie1_cfg_interrupt_msi_mmenable
- pcie1_cfg_interrupt_msi_mask_update
- pcie1_cfg_interrupt_msi_data
- pcie1_cfg_interrupt_msix_enable
- pcie1_cfg_interrupt_msix_mask
- pcie1_cfg_interrupt_msix_vec_pending_status

Register Space

The configuration space is a register space defined by the *PCI Express Base Specification 4.0* (<https://www.pcisig.com/specifications>). The Versal® ACAP CPM Mode for PCIe supports Xilinx proprietary read/write configuration interfaces into this register space, and supports up to four Physical Functions (PFs) and 252 Virtual Functions (VFs).

The PCI configuration space consists of the following primary parts.

Legacy PCI v4.0 Type 0/1 Configuration Space Header

- Type 0 Configuration Space Header supported for Endpoint configuration
- Type 1 Configuration Space Header supported for Root, Switch Port configuration

Legacy Extended Capability Items

- PCIe Capability
- Power Management Capability
- Message Signaled Interrupt (MSI) Capability
- MSI-X Capability
- Legacy Extend Capabilities

PCIe Extended Capabilities

- Advanced Error Reporting Capability
- Function Level Reset
- ASPM L1 Support

- ASPM L0s Support (supported in Gen1 and Gen2 configurations only)
- Device Serial Number Capability
- Virtual Channel Capability
- ARI Capability (optional)
- SR-IOV Extended Capability Structure
- Configuration Space Extend Capabilities
- Address Translation Services (ATS)
- Page Request Interface (PRI)
- Feature DLLP
- CCIX Transport DVSEC through configuration space extension
- CCIX Protocol DVSEC through configuration space extension
- Transaction Tag Scaling as Requester and Completer
- Flow Control Scaling
- MCAP Interface for Staged Configuration and Dynamic Function eXchange per PCI Express port
- PASID Capability
- Lane Margining at the Receiver Capability
- Physical Layer 16 GT/s Capability

Designing with the Core

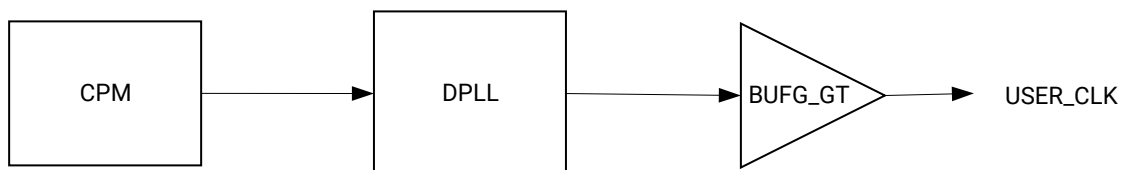
This section includes guidelines and additional information to facilitate designing with the core.

Clocking

Note: USER_CLK (`user_clk`) in this section refers to `pcie(n)_user_clk`, which is also described in the Clock and Reset Interface section.

The CPM requires a 100, 125, or 250 MHz reference clock input. The following figure shows the clocking architecture. The `user_clk` clock can be used as the system clock.

Figure 6: USER_CLK Clocking Architecture



X22710-071520

All user interface signals are timed with respect to the same clock (`user_clk`) which can have a frequency of 62.5, 125, or 250 MHz depending on the configured link speed and width. The `user_clk` should be used to interface with the CPM. With the user logic, any available clocks can be used.

Each link partner device shares the same reference clock source. The following figures show a system using a 100 MHz reference clock. Even if the device is part of an embedded system, if the system uses commercial PCI Express® root complexes or switches along with typical motherboard clocking schemes, synchronous clocking should be used.

Note: The following figures are high-level representations of the board layout. Ensure that coupling, termination, and details are correct when laying out a board.

Figure 7: Embedded System Using 100 MHz Reference Clock

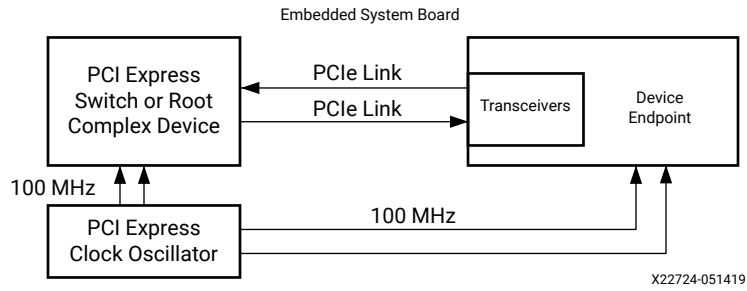
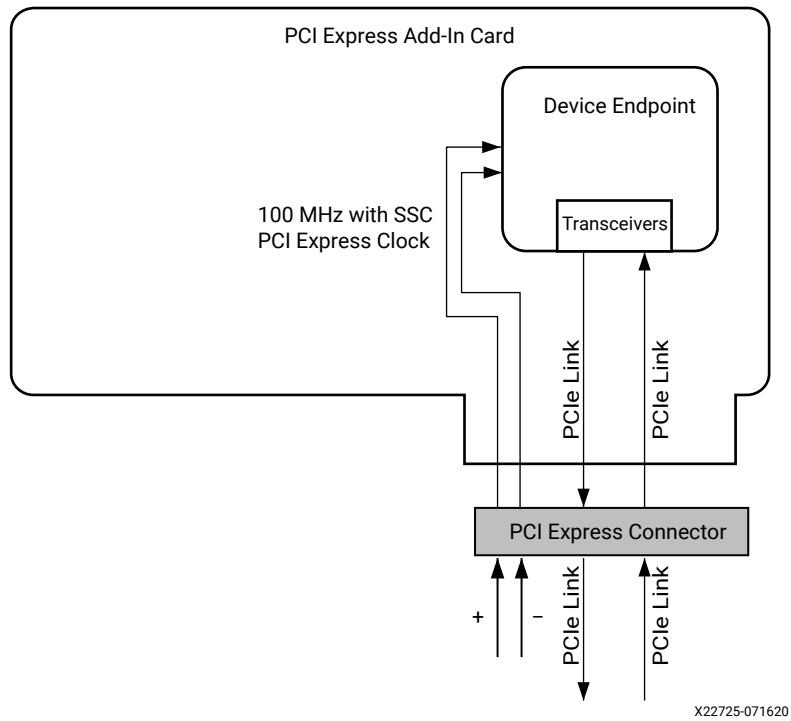


Figure 8: Open System Add-In Card Using 100 MHz Reference Clock



Related Information

[Clock and Reset Interface](#)

Resets

The fundamental resets for the CPM PCIe® controllers and associated GTs are `perst0n` and `perst1n`. The resets are driven by the I/O inside the PS. In addition, there is a power-on-reset for CPM driven by the platform management controller (PMC). When both PS and the power-on-reset from PMC are released, CPM PCIe controllers and the associated GTs will come out of reset.

After the reset is released, the core attempts to link train and resumes normal operation.

In addition, there is a `pcie(n)_user_reset` given from the CPM PCIe controller to the user design present in the fabric logic. Whenever CPM PCIe controller goes through a reset, or there is a link down, the CPM PCIe controller issues a `pcie(n)_user_reset` to the user design in the programmable logic (PL) region. After the PCIe link is up, `pcie(n)_user_reset` is released for the user design to come out of reset.

Tandem Configuration

Overview

PCI Express® is a plug-and-play protocol, meaning that at power up the PCIe® host will enumerate the system. This process consists of the host enumerating PCIe devices and assigning them base addresses. As such, PCIe interfaces must be ready when the host queries them or they will not get assigned a base address. The PCI Express specification states that PERST# can deassert 100 ms after the power good of the systems has occurred, and a PCI Express port must be ready to link train 20 ms after PERST# has deasserted. This is commonly referred to as the 100 ms boot time requirement, even though 120 ms is the fundamental goal.

Xilinx devices can meet this 120 ms link training requirement by using Tandem Configuration, a solution that splits the programming image into two stages. The first stage quickly configures the PCIe endpoint(s) so the endpoint is ready for link training within 120 ms. The second stage then configures the rest of the device. Two variants are supported:

- **Tandem PROM:** Loads both stages from a single programming image from a standard primary boot interface.
- **Tandem PCIe:** Loads the first stage from a primary boot interface, then the second stage is delivered via one of the CPM PCIe controllers.

Within Xilinx Versal® ACAP, the CPM consists of two PCIe controllers, DMA features, CCIX features, and network on chip (NoC) integration. The Versal® ACAP CPM Mode for PCI Express enables direct access to the two high-performance, independently customizable PCIe controllers. You can select to have one or both of these controllers enumerate within 120 ms using Tandem Configuration.

While the term **Tandem Configuration** has been carried forward from prior iterations of this technology applied in 7 series, UltraScale™, and UltraScale+™ silicon, the solution is fundamentally different in Versal given how the PCIe controllers are built and configured. No programmable logic is needed to activate an endpoint, so only CPM, XPIPE, NoC, and GTY resources are included along with the PMC in the first stage.



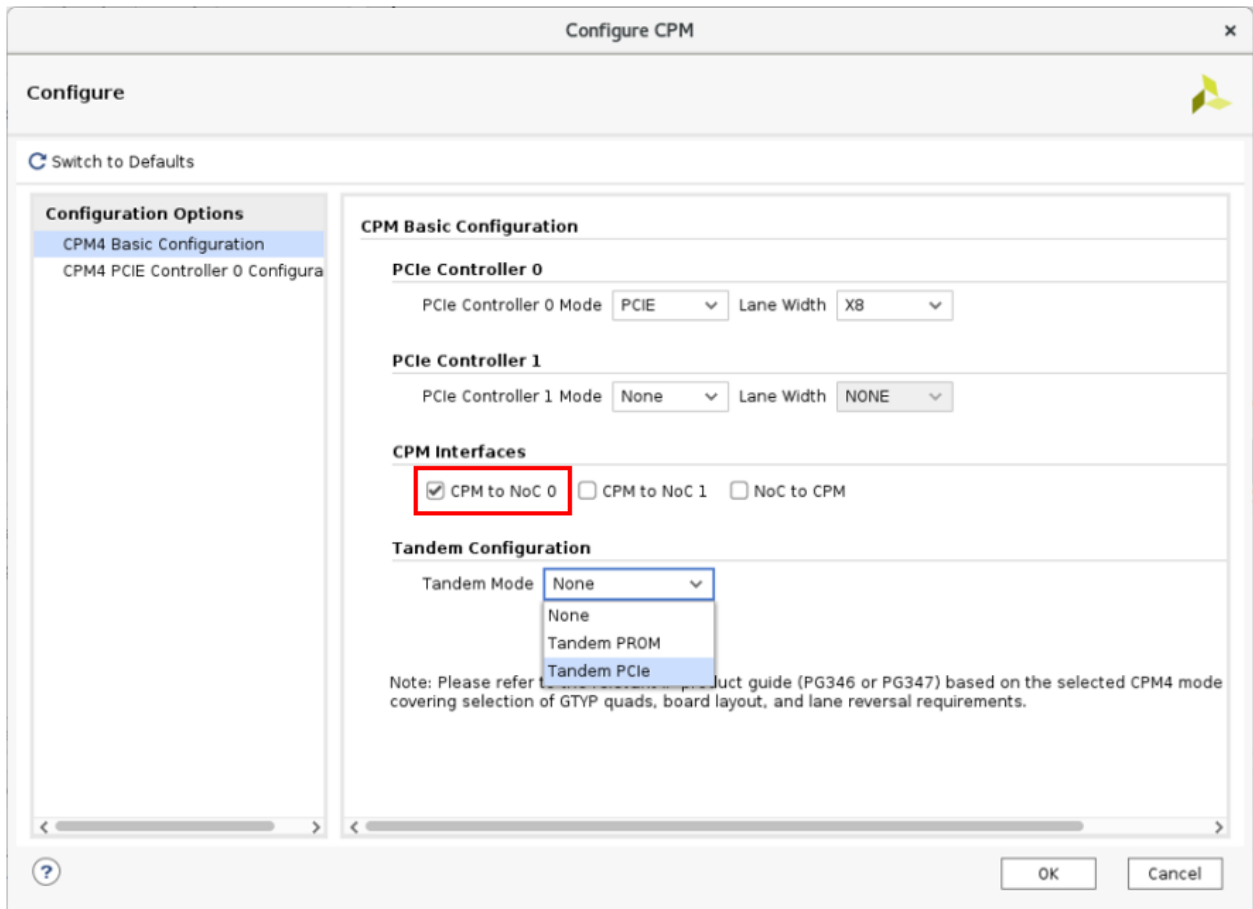
IMPORTANT! Only production Versal silicon is supported for Tandem Configuration. Do NOT use this solution on engineering silicon (ES1).

Enable the Tandem Configuration Solution

Starting with Vivado 2021.1, an option in the CIPS customization GUI allows you to pick which option suits their needs. In the CPM options, a Tandem Configuration selection will be enabled when a PCIe Endpoint is selected for PCIe Controller 0 or Controller 1. The three available options are:

- Tandem PROM
- Tandem PCIe
- None

Figure 9: Customizing the PCIe Controllers



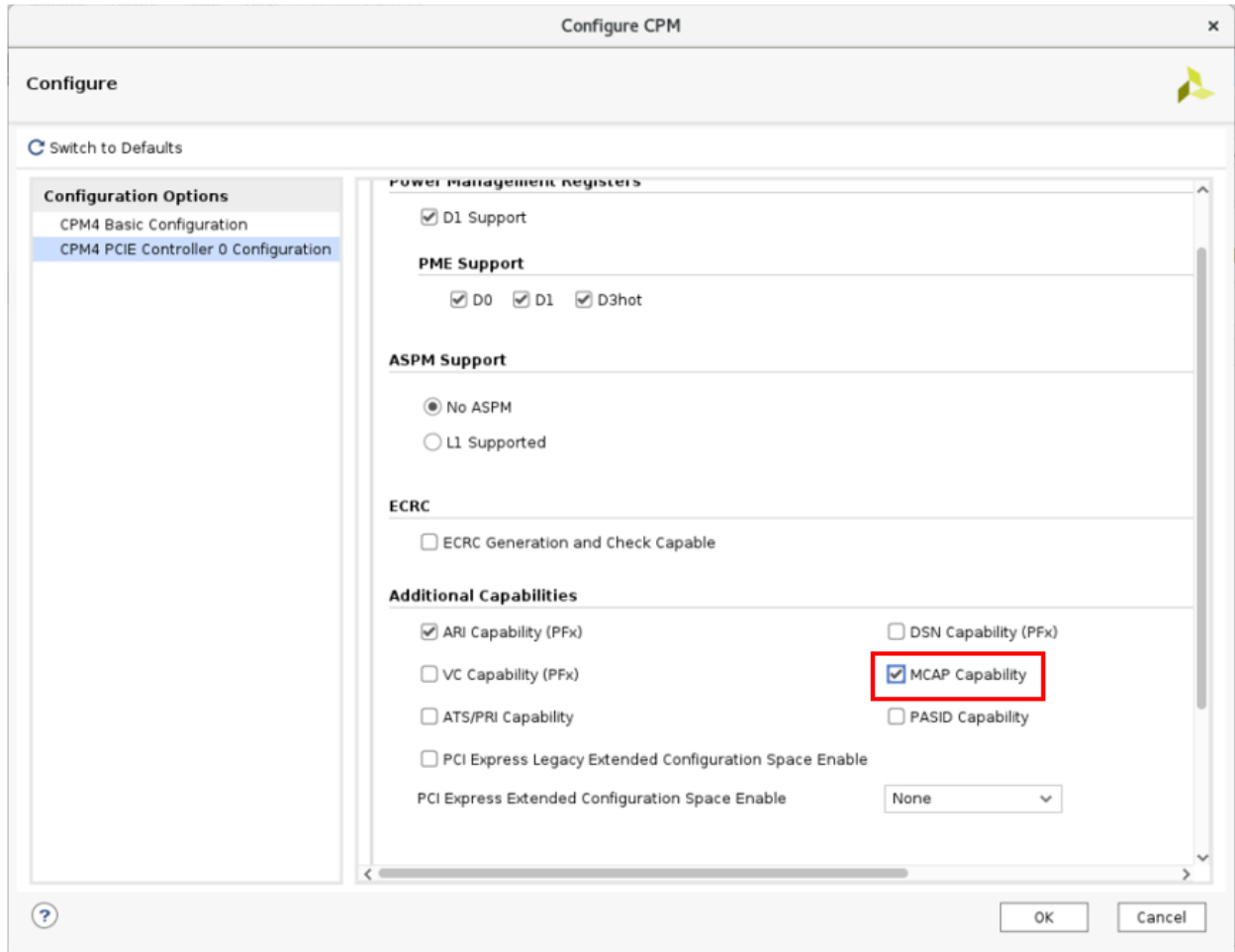
Tandem PROM is the simpler mode for Tandem Configuration, where both stages reside in a single programming image. If 120 ms enumeration is required the selection of this option essentially comes for free, as there is no overhead in design complexity or requirement for programmable logic to build. The programming ordering simply starts with the CPM and other necessary elements before moving on to the rest of the device.

Because Tandem PCIe uses the PCIe link to program the stage 2 portion of the design, the design must include connectivity from the enable CPM Master(s) to the PMC Slave. This should be accomplished through the block design connectivity -- note that in the figure above, the **CPM to NoC 0** interface must be selected before the Tandem PCIe option appears --and appropriate mapping of the slave into the CPM master address space(s). This includes enabling the CPM to NoC 0 Interface by checking the appropriate box on the CPM Basic Configuration customization page. This is done automatically for DMA mode but must be selected for CCIX or PCIe modes. The specific aperture within the PMC slave that must be accessible from the host is the Slave Boot Interface (SBI) which is available at the following Versal address.

Versal Slave Boot Interface Address: 0x10210000.

To deliver stage 2 images using MCAP VSEC, this advanced feature must be enabled during IP customization. Within the Configuration options for the Controller instance to be used:

1. Under the Basic tab, set the PCIe[0/1] Basic/Advanced mode selection option to **Advanced**.
2. Under the Advanced Options tab, check the box for **MCAP Capability**.



To deliver stage 2 images using PCIe DMA, the DMA BAR must be set to BAR0. The driver will probe BAR0 to find the DMA BAR. If this goes to the PL the transaction will not complete, because the PL is not yet configured. For more information on Tandem PCIe using QDMA or XDMA, see *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide* (PG347).

Confirmation that Vivado parameters and Tandem Configuration in general have been applied can be seen in the log when `write_device_image` is run. Following is a snippet of the log for a Tandem PROM run during the `write_device_image` step:

```

Creating bitstream...
Tandem stage1 bitstream contains 23552 bits.
Writing CDO partition ./design_1_wrapper_tandem1.rcdo...
Writing NPI partition ./design_1_wrapper_tandem1.rnpi...
Creating bitstream...
    
```

```
Tandem stage2 bitstream contains 5939712 bits.
Writing CDO partition ./design_1_wrapper_tandem2.rcdo...
Writing NPI partition ./design_1_wrapper_tandem2.rnpi...
Writing NPI partition ./design_1_wrapper_shutdown.txt...
Generating bif file design_1_wrapper_tandemPROM.bif for Tandem PROM.
```

The resulting run creates (in addition to the files mentioned above) a single `.pdi` image for this design called `design_1_wrapper.pdi`.


When Tandem PCIe is enabled through CIPS IP customization, two `.pdi` files will be generated:

- `tandem1.pdi`: This file should be added to the device configuration flash.
- `tandem2.pdi`: This file should be programmed into the device through the PCIe link once it becomes active.

The resulting report in the `write_device_image` log looks nearly identical, but the file name for the `.bif` will be slightly different:

```
Creating bitstream...
Tandem stage1 bitstream contains 23552 bits.
Writing CDO partition ./design_1_wrapper_tandem1.rcdo...
Writing NPI partition ./design_1_wrapper_tandem1.rnpi...
Creating bitstream...
Tandem stage2 bitstream contains 5939712 bits.
Writing CDO partition ./design_1_wrapper_tandem2.rcdo...
Writing NPI partition ./design_1_wrapper_tandem2.rnpi...
Writing NPI partition ./design_1_wrapper_shutdown.txt...
Generating bif file ./design_1_wrapper_tandem1.bif for Tandem stage-1.
```

In addition to the files mentioned above, the resulting run creates two `.pdi` images for this design called `design_1_wrapper_tandem1.pdi` and `design_1_wrapper_tandem2.pdi`. The `_tandem1` and `_tandem2` suffixes are automatically added to differentiate the stages.

 **IMPORTANT!** *Stage 1 and stage 2 bitstreams must remain paired. While this is trivial for Tandem PROM because both stages are stored in a single PDI image, this is a critical consideration for Tandem PCIe. If any part of the design is modified such that a full recompilation is triggered, both stage 1 and stage 2 images must be updated. Always update both stages when any change is made.*

In UltraScale+, the Field Updates solution enables you to build “Reconfigurable Stage Twos” where one could not only pick a stage 2 image from a list of compatible images, but then also reconfigure that stage 2 area with another stage 2 image to provide dynamic field updates. In Versal, the first part (for the initial boot of a device) may be supported in the future to allow you to lock a stage 1 image in a small local boot flash; the second part (dynamic reconfiguration) will require a Tandem + DFX-based approach to allow for dynamic reconfiguration of a subsection of the PL.

For test and debug purposes the `HD.TANDEM_BITSTREAMS` property can be set on the implemented design before `.pdi` file generation to separate a single Tandem PROM `.pdi` file into separate `tandem1.pdi` and `tandem2.pdi` files.

```
set_property HD.TANDEM_BITSTREAMS Separate [current_design]
```

Similarly, the behavior of Tandem PROM or Tandem PCIe file generation can be disabled entirely by using the `HD.TANDEM_BITSTREAMS` property on the implemented design before `.pdi` file generation. The following command can be used to do this.

```
set_property HD.TANDEM_BITSTREAMS NONE [current_design]
```

Deliver Programming Images to Silicon

The purpose of Tandem Configuration is to be able to program the PCIe endpoint(s) within 120 ms before link training starts. The stage 1 file configures the CPM, GTs, and NoC connection required for stage 1 operation. These blocks then become active and can interact with the host to perform PCIe enumeration.

When using Tandem PROM, configuration of the remainder of the device continues to load from the same programming image on the same primary boot device while the PCIe endpoint(s) are up and running. There are no requirements for PERSIST or similar restrictions as was the case with UltraScale+ and prior FPGA architectures.

When using Tandem PCIe, the remainder of the device programming is done by delivering the stage 2 `.pdi` from the PCIe host as a secondary boot device. There are four data paths in the CPM through which the `tandem2.pdi` file can be loaded from PCIe into the PMC Slave Boot Interface. The specific path you select is determined based on the IP configuration desired for your specific application. These are enumerated below.

- QDMA MM Data Path:** If the QDMA Memory Mapped data path is enabled, it can be used to download through PCIe into the FPGA Slave Boot Interface at a maximum rate of 3.2 Gbytes/s. This is limited by the programming rate of the Slave Boot Interface. Xilinx provides sample QDMA drivers and software (see Note below). This data path can only be used with PCIe controller 0 because CPM controller 1 does not support hardened QDMA operation.
- XDMA MM Data Path:** If the XDMA Memory Mapped data path is enabled, it can be used to download through PCIe into the FPGA Slave Boot Interface at a maximum rate of 3.2 Gbytes/s. This is limited by the programming rate of the FPGA Slave Boot Interface. Xilinx provides sample XDMA drivers and software (see Note below). This data path can only be enabled with PCIe controller 0 because CPM controller 1 does not support hardened XDMA operation.
- AXI Master Bridge:** If the AXI Master bridge data path is enabled, it can be used to download through PCIe into the FPGA Slave Boot Interface at a maximum rate limited by the host's ability to generate PCIe transactions (typically around 700 Mbytes/s for 64-byte transfers from the host). Xilinx does not provide sample drivers and software for this because either the XDMA or QDMA data paths are typically enabled with this mode and allow for higher transfer rates. This data path can only be enabled with PCIe controller 0 because CPM controller 1 does not support hardened AXI Master Bridge operation.

- **Versal MCAP VSEC:** Both controllers in CPM can enable the MCAP Vendor Specific Extended Capability. This capability functions differently from UltraScale and UltraScale+ solutions. It can be used to download through PCIe into the FPGA Slave Boot Interface at a maximum rate limited by the host's ability to generate 32-bit PCIe configuration transactions (typically lower than 1 Mbytes/s). This mode of configuration should only be used when the other three data paths are not available. This includes PCIe Streaming and CCIX modes on either controller.

Note: Xilinx provides sample drivers and software to enable stage 2 programming. These drivers can be found at https://github.com/Xilinx/dma_ip_drivers.

Tandem Configuration Performance

The initial programming image is stored in and retrieved from a primary boot device. For Tandem PROM, this image contains both stages, and for Tandem PCIe, this image contains only stage 1. To calculate configuration performance, use the stage 1 image size as reported in the `write_device_image` log files (shown above).

When using Tandem PCIe, the stage 2 image is delivered over a secondary boot device. Depending on the delivery path and PCIe controller options, configuration performance can be up to 3.2 GB/s while configuring the programmable logic.

- CPM QDMA-MM to SBI: Gen3x16+ expected bandwidth 3.2 GB/s
- CPM XDMA-MM to SBI: Gen3x16+ expected bandwidth 3.2 GB/s
- CPM AXI-Bridge to SBI: Maximum expected Bandwidth 700 MB/s for 512-bit 64-byte transfers
- CPM MCAP to SBI: Maximum expected bandwidth 1 MB/s or slower

For more information regarding Versal Configuration and Boot, please consult *Versal ACAP System Software Developers Guide* ([UG1304](#)).

Design Operation

Though the CPM is a hardened integrated block, many features and options that can be selected during CIPS configuration will require implementation in programmable soft logic (PL). Any part of the design that has been implemented in the PL will be configured in stage 2. Design configurations that require PL resources during PCIe enumeration should not be used with Tandem PROM or Tandem PCIe. Specifically, the PCIe extended capability interface should not be enabled for Tandem modes because these registers are addressed during enumeration and are not present in the stage 1 portion of the design. Moreover, any other resource in the Versal device, such as the R5 or A72 processors in the Scalar Engines, will be programmed after the CPM and its PCI Express endpoint(s). While future enhancements to the Tandem Configuration solution may open opportunities to quickly booting other dedicated parts of a target device, the current solution focuses exclusively on PCI Express end points in the CPM only for the sole purpose of meeting the 100 ms boot time requirement.

Known Issues and Limitations

- Engineering Silicon (ES) is not supported for VC1902, VC1802 and VM1802 devices. Only production silicon is supported for these devices.
- PCIe features incompatible with Tandem Configuration:
 - PCIe Extended Configuration Space, as this requires PL logic.
 - QDMA multi-function is not supported. This feature uses PL mailbox which will be probed during driver load.
- Stage 1 and stage 2 PDI images must remain linked. If you update one, update the other to ensure both stages have been generated from the same implemented design.
- Do not reload the same or a new stage 2 image on the fly. Isolation circuitry on the periphery of the CPM is designed to ensure safe operation of the PCI Express endpoints prior to the rest of the device becoming active. Upon stage 2 completion, this isolation is released and it is not re-enabled for a dynamic reload of the stage 2 image.
- For the Versal architecture, CPM Tandem PROM configuration can be used with post configuration flash update. This is different from previous (UltraScale+ and older) architectures that did not support Tandem PROM post configuration flash update. Given the lack of dual-mode configuration pins, there is no PERSIST requirement to keep a configuration port active.
- Tandem Configuration is compatible with most Dynamic Function eXchange (DFX) solutions. Care must be taken to keep any DFX Pblock away from the PS-PL boundary where soft logic associated with the CIPS is placed.
- At this point there is no “Tandem with Field Updates” predefined use case. You can create a “Tandem + DFX” solution noted above where both features are enabled in the same design, but creation of the design hierarchy and floorplan as well as insertion of any decoupling logic are the responsibility of the designer. The dynamic (DFX) portion of the solution would be limited to programmable logic and NoC resources, and not parts of the Scalar Engines (processors).
- The CPM itself (and therefore Tandem Configuration) is not compatible with the Classic SoC Boot flow, which utilizes DFX to separate PS and PL configuration events. This is because the CPM solution requires soft logic for most configurations and thus the DFX region cannot be constructed to include all of the programmable logic.

Design Flow Steps

This section describes customizing and generating the Versal[®] ACAP CPM Mode for PCIe, constraining the Versal[®] ACAP CPM Mode for PCIe, and the simulation, synthesis, and implementation steps that are specific to this IP Versal[®] ACAP CPM Mode for PCIe. More detailed information about the standard Vivado[®] design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

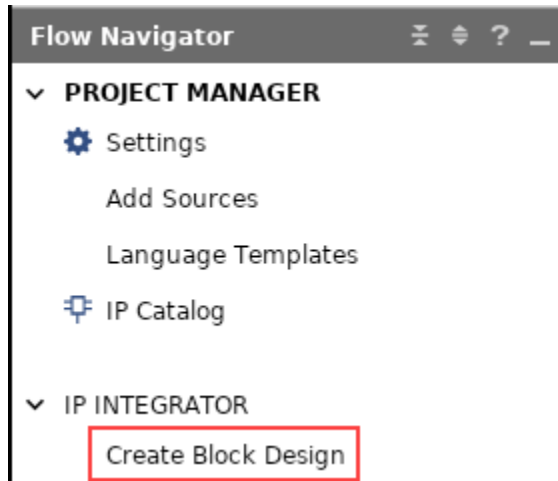
- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))
- *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
- *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
- *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))

Customizing and Generating the CIPS IP Core

This section includes information about using the Vivado[®] Design Suite to customize and generate the Control, Interfaces and Processing System IP core. This section configures the CIPS IP core to access the CPM PCIe controllers directly. For extended information about the CIPS IP core, see the *Control, Interface and Processing System LogiCORE IP Product Guide* ([PG352](#)).

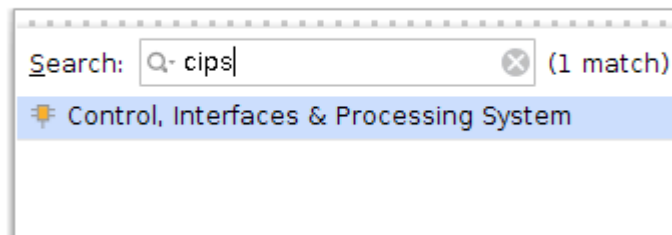
Configuring the CIPS IP Core

1. In the Vivado IDE, select **IP Integrator** → **Create Block Design** from the Flow Navigator, as shown in the following figure.

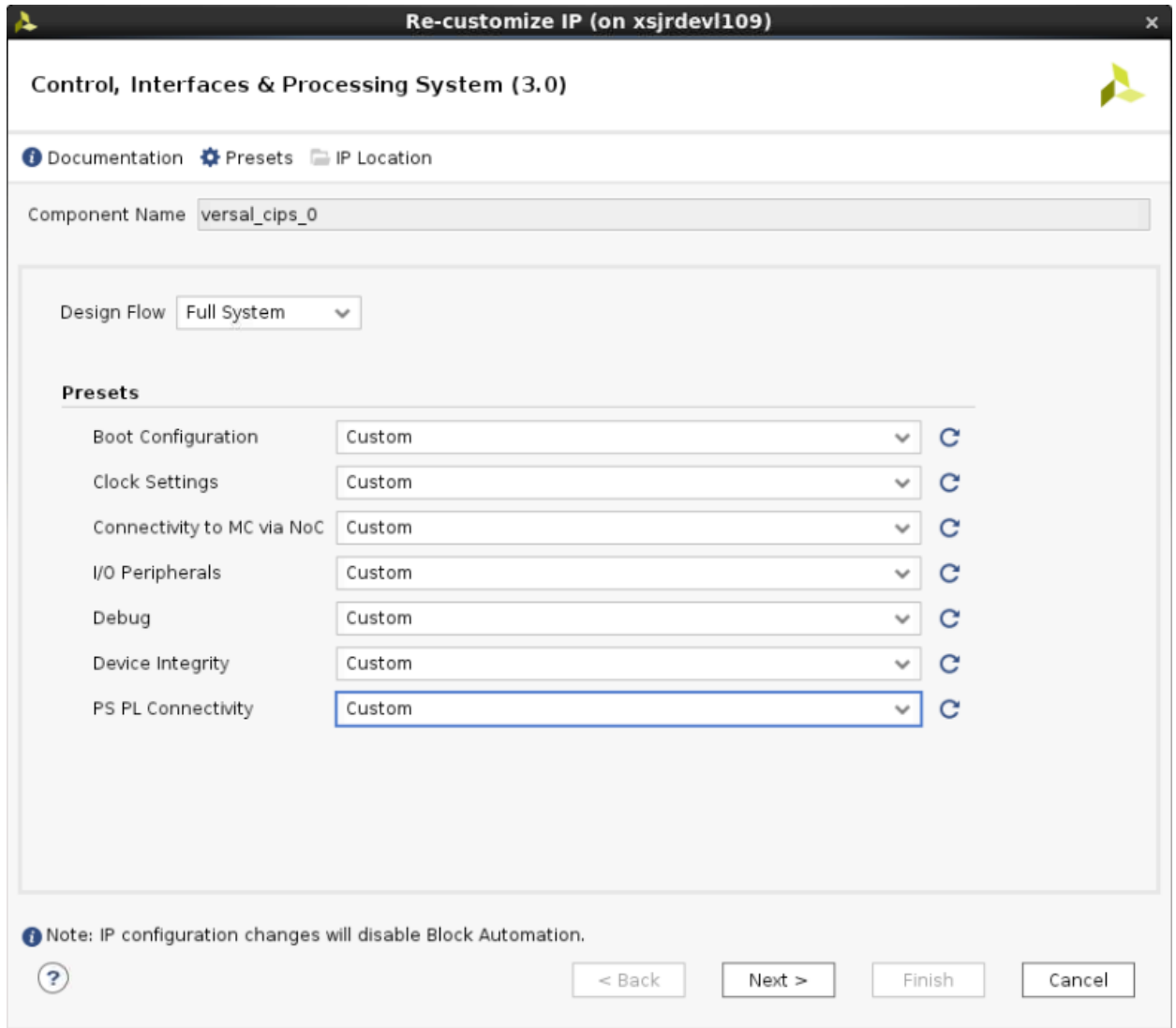


A popup dialog displays to create the block design.

2. Click **OK**. An empty block design diagram canvas opens in the IP integrator.
3. Right-click on the block design canvas and from the context menu select **Add IP**.
4. Search for `cips`.

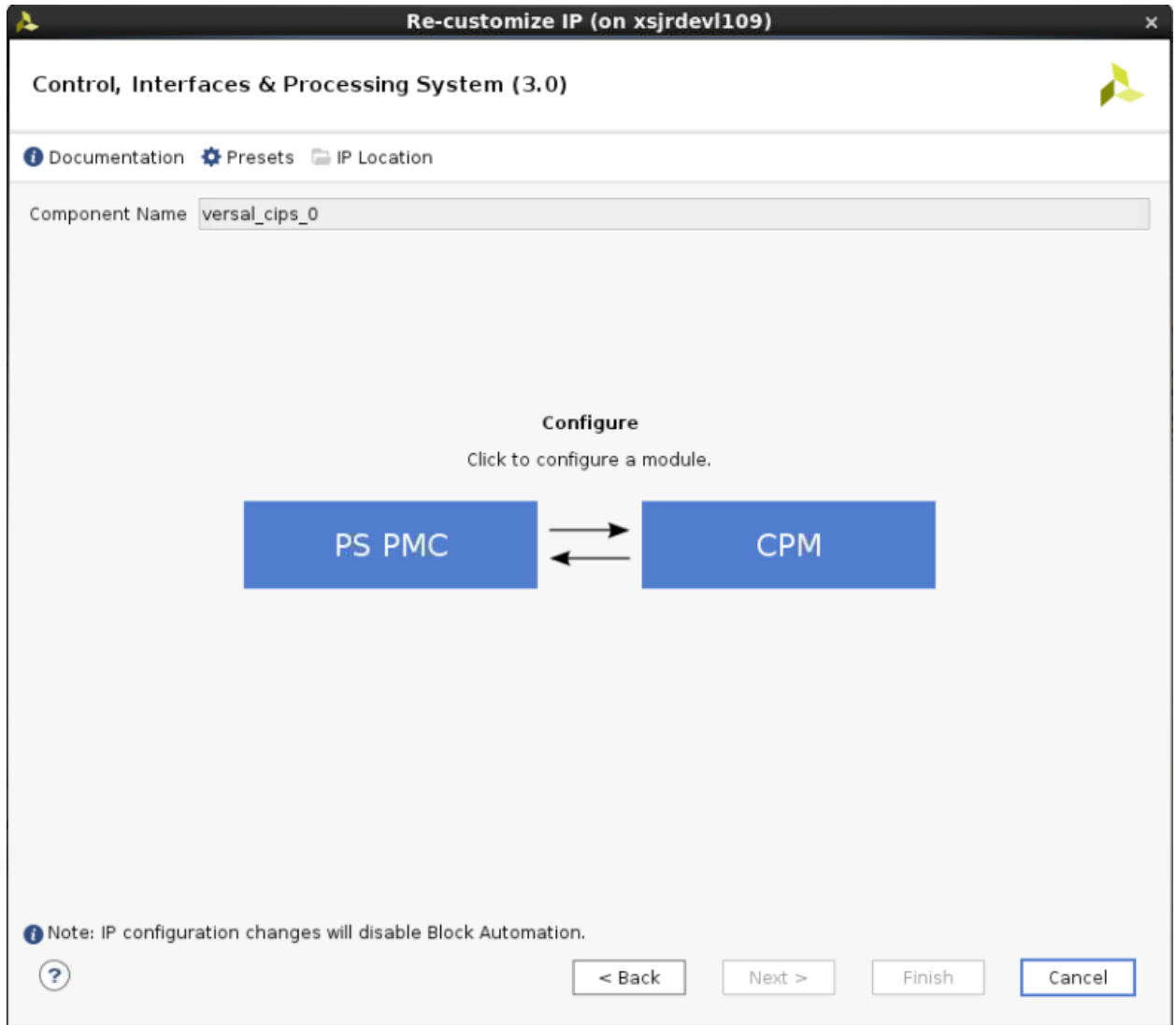


5. Double-click the **Control, Interface, and Processing System** IP core to customize it.
6. In the Configuration Options pane, leave the settings as default or select appropriate options required for presets and click **Next**.

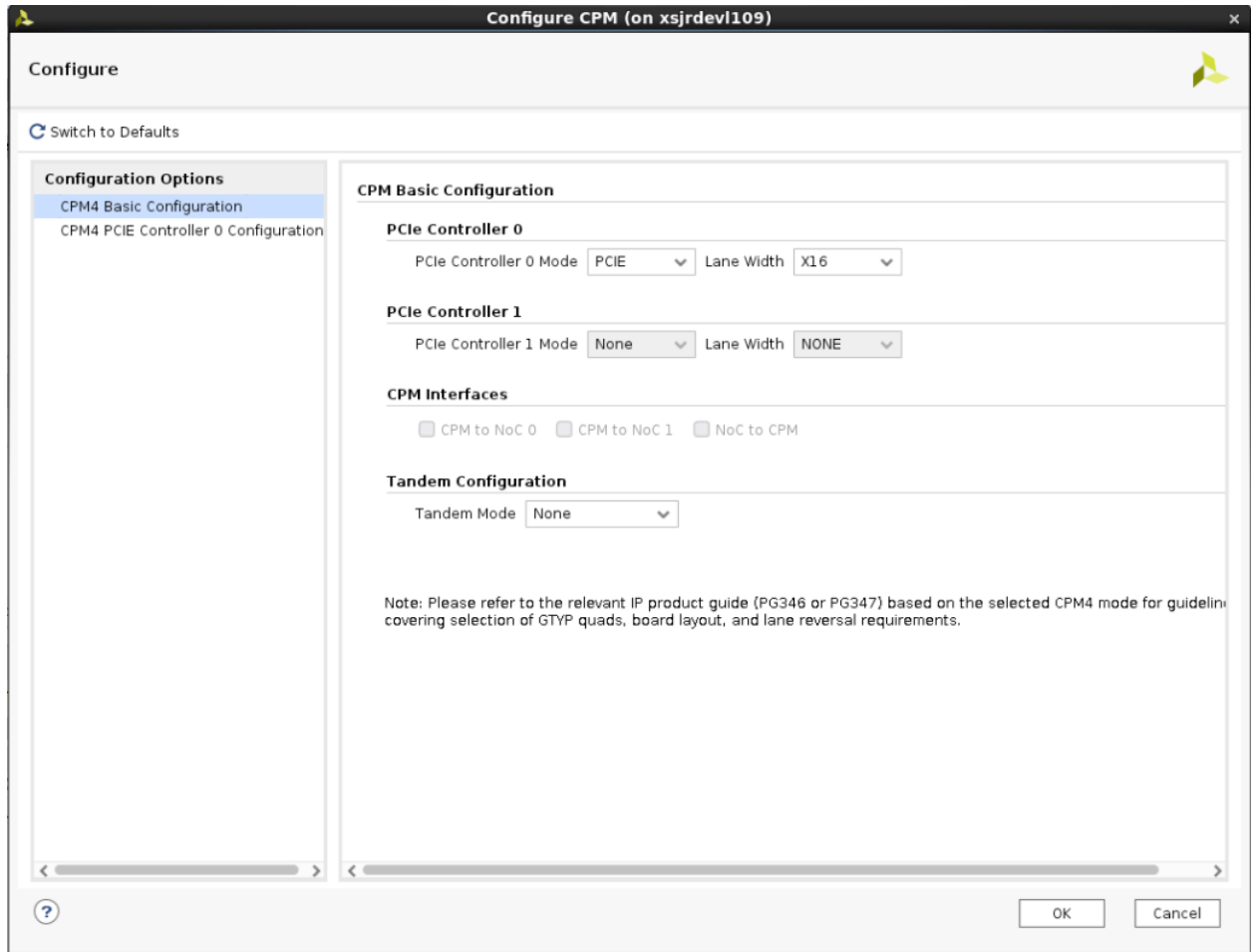


Note: For information about the option presets, see *Control, Interface and Processing System LogiCORE IP Product Guide (PG352)*.

7. Click the **CPM** block to configure the core.



The CPM Basic Configuration page displays.



8. Set the PCIe Controller 0 Mode to **PCIe**, and select the lane width. This setting enables the PCIe Port 0, and in a later step, you will be configuring the PCIe Port 0.
9. If you require the PCIe Port 1, set the PCIe Controller 1 Mode to **PCIe**, and select the lane width.

Table 34: Available Lane Width Combinations

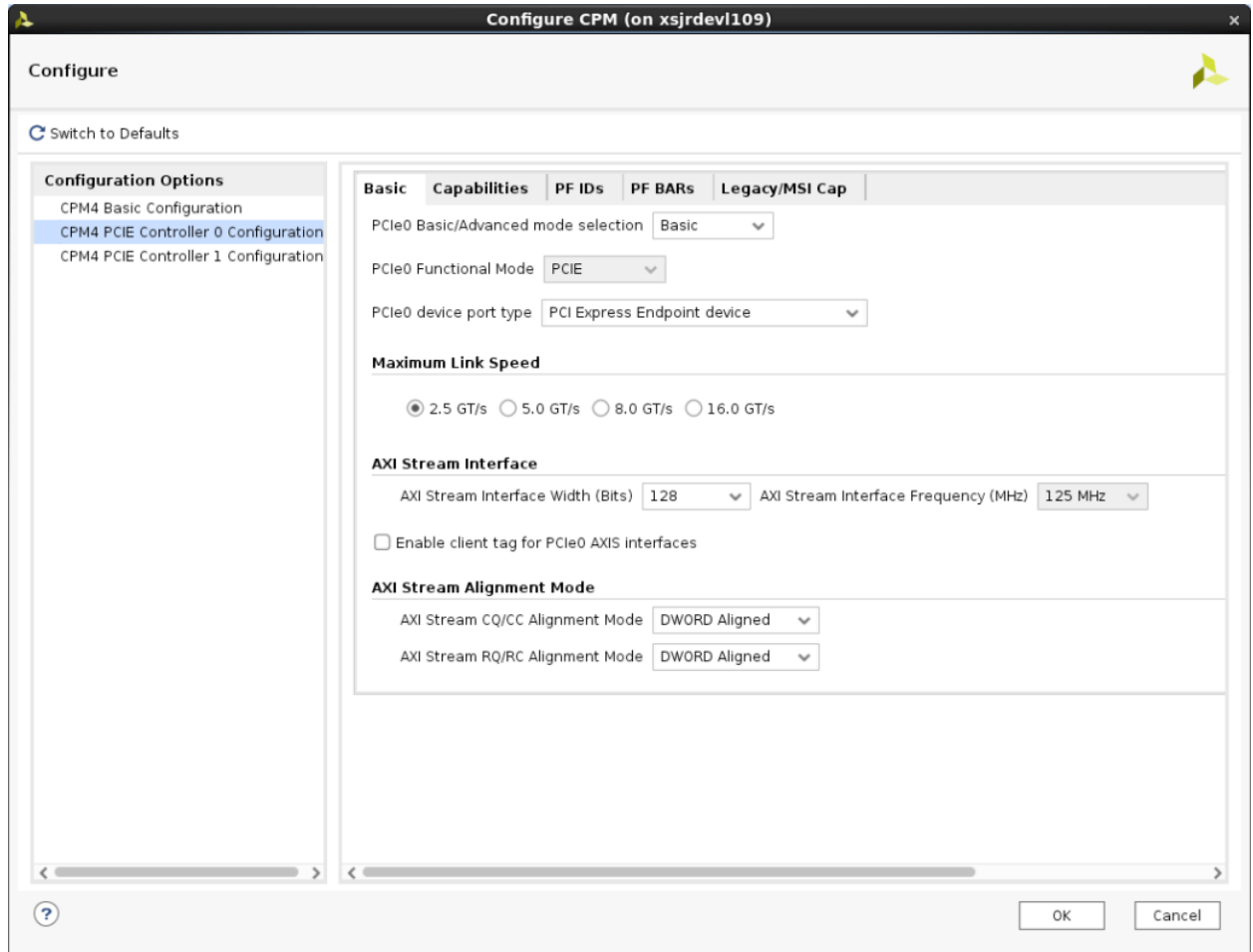
PCIe Port 0	PCIe Port 1
X1, X2, X4, or X8	X1, X2, X4, or X8
X16	Not available

Note: There is no PCIe Port 1-only option available.

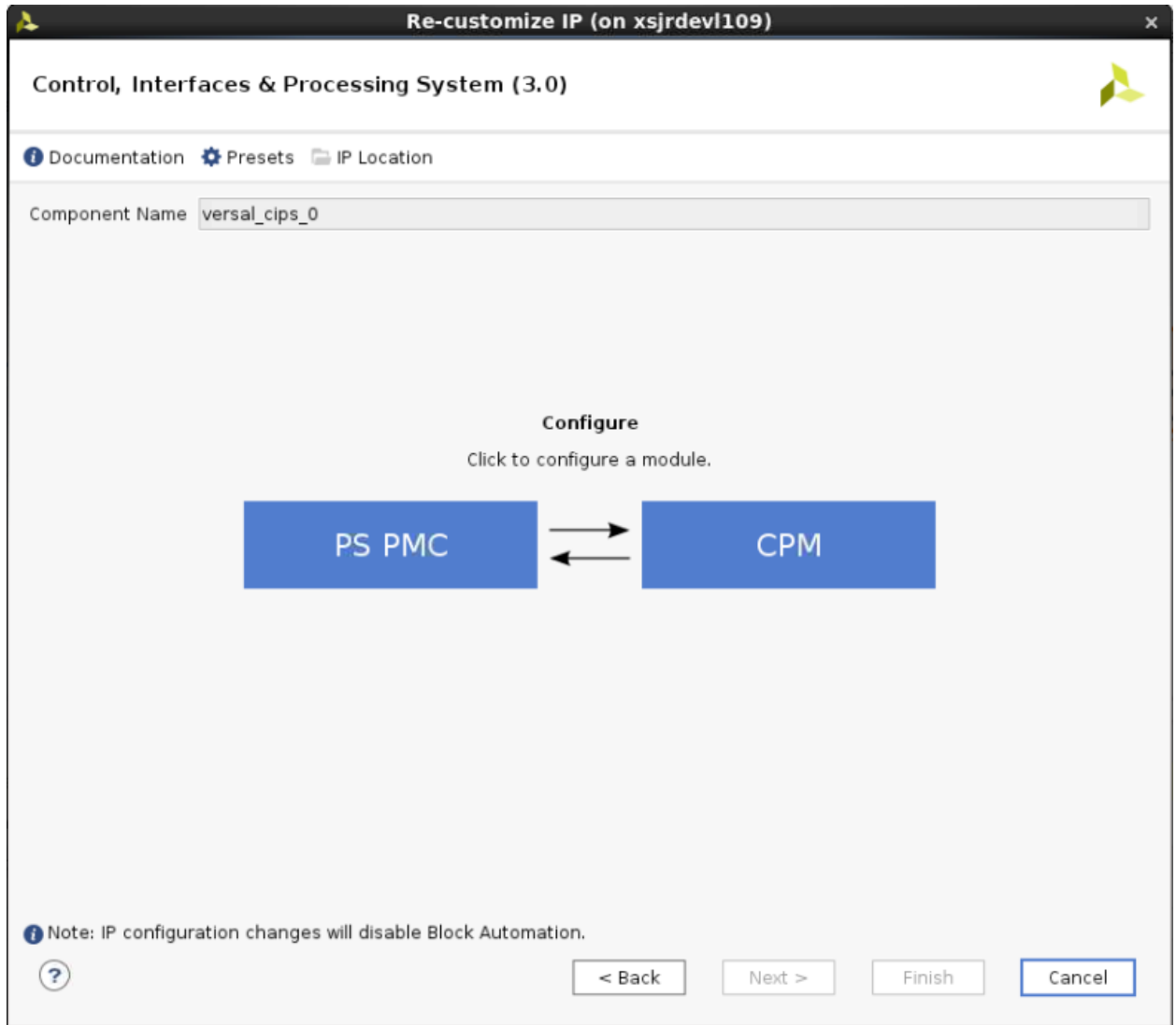
Note: PCIe Port 1 is available only if the lane width of PCIe Port 0 is less than or equal to X8.

Note: PCIe Port 1 supports up to X8 when PCIe Port 0 is configured up to X8.

- In the Configuration Options pane, expand **CPM4**, and click **PCIE Controller 0 Configuration** to customize the PCIe Port 0 for the Versal ACAP CPM Mode for PCI Express core. It offers two modes: Basic, and Advanced. To select a mode, use the CPM Modes drop-down list on the first page of the Customize IP dialog box. Next section will explain the parameters available in each mode.

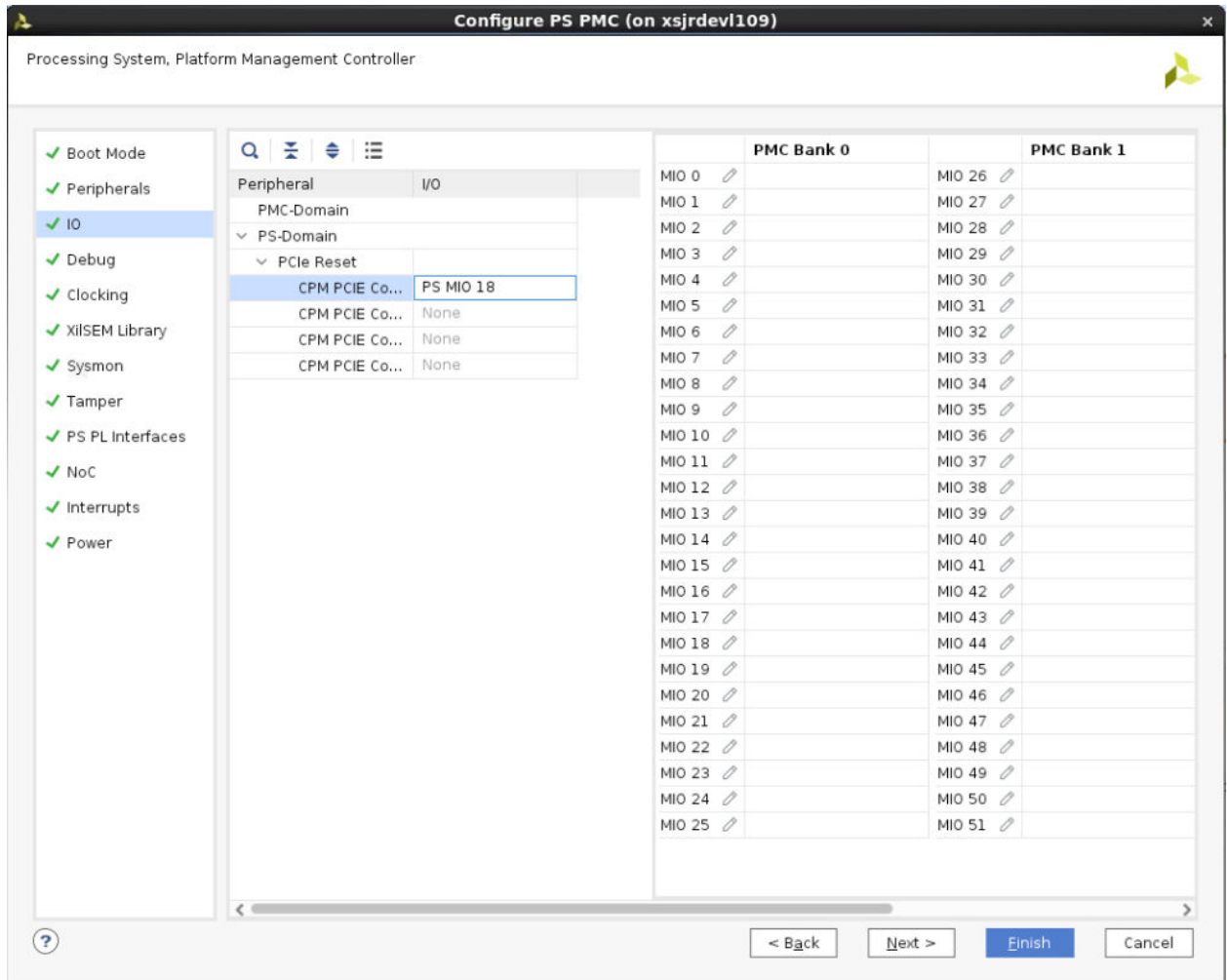


- If applicable, in the Configuration Options pane, expand **CPM**, and click **PCIE Controller 1 Configuration** to customize PCIe Port 1.
- After configuring the PCIe controller, click **OK** to return to the Configure screen, as shown below.



13. Click **PS PMC**, and click **IO** configuration.

The IO Configuration tab has a list of options to configure the external PCIe Reset options.



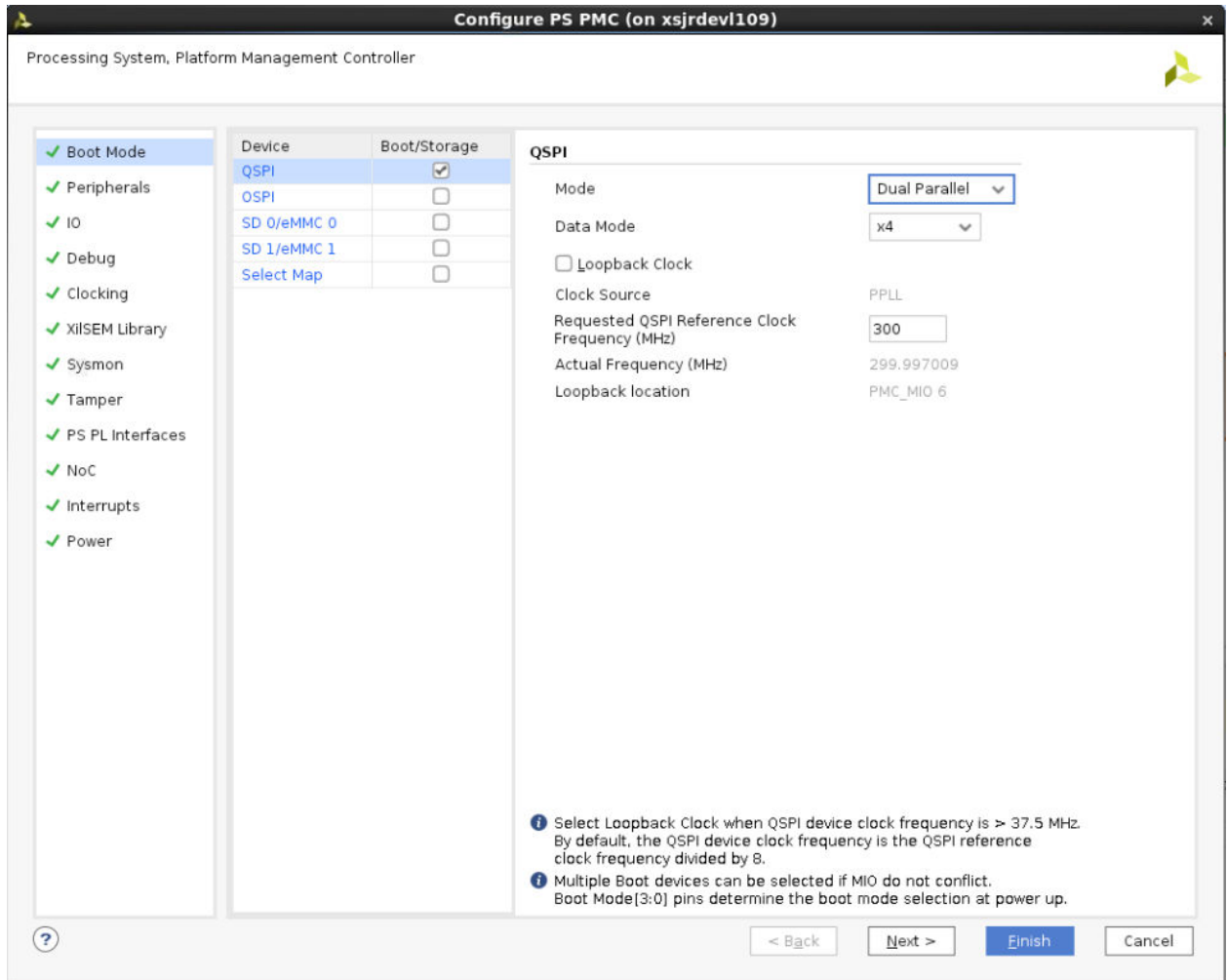
14. Select the **PCIe Reset** option located in the Peripheral column.

Notice that the MIO pin selected in the PCIe reset is automatically connected to the PCIe reset I/O. In the figure below, MIO 38 is connected to the PCIe reset I/O.

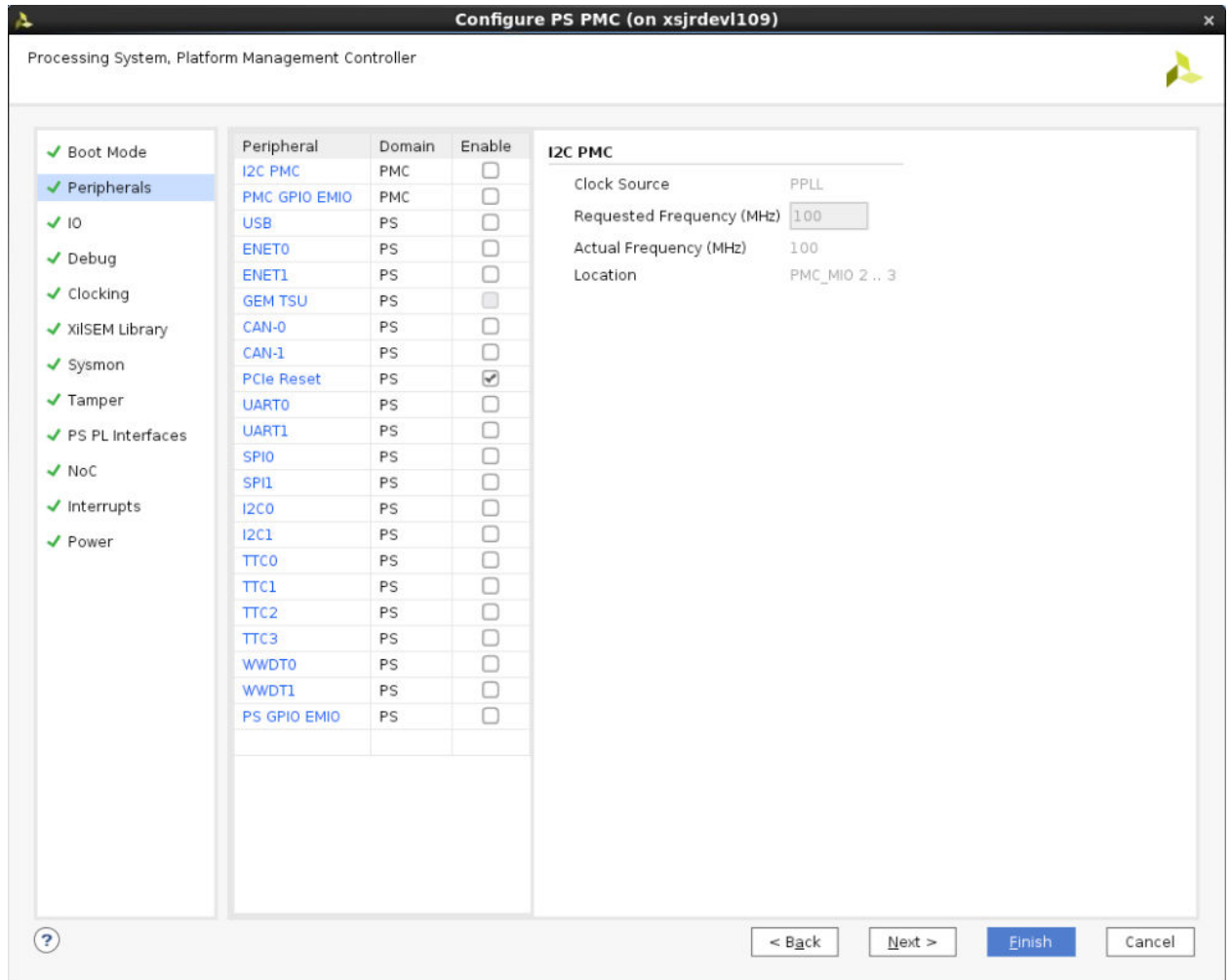
- For PCIe Port 0 Endpoint configuration: Next to A0 Endpoint, select one of the available MIOs: **PS MIO 18**, **PMC MIO 24**, or **PMC MIO 38**. This selection should match the MIO that is connected to the PCIe reset I/O in the board schematic. In the figure below, PMC MIO 38 is selected to correspond to PCIe reset MIO 38.
- If PCIe Port 1 is enabled: Next to A1 Endpoint, select one of the available MIOs (**PS MIO 19**, **PMC MIO 25**, or **PMC MIO 39**) based on which MIO is connected to the PCIe reset I/O in the board schematic.

15. If the board will boot from serial NOR flash, select the a **QSPI** or **OSPI** option in **Boot Mode** options to enable programming of the flash on the board. Select the appropriate option based on availability to match the board schematic.

To set up the boot device, see the *Versal ACAP Technical Reference Manual* (AM011). If a serial NOR flash boot device will be used, the correct options must be selected to enable the correct MIOs.



16. To enable additional I/O interfaces, such as UART, 12C, and USB IOs, select them in the **Peripherals** section in a similar manner. See the *Versal ACAP Technical Reference Manual* (AM011) for more details on these interfaces.



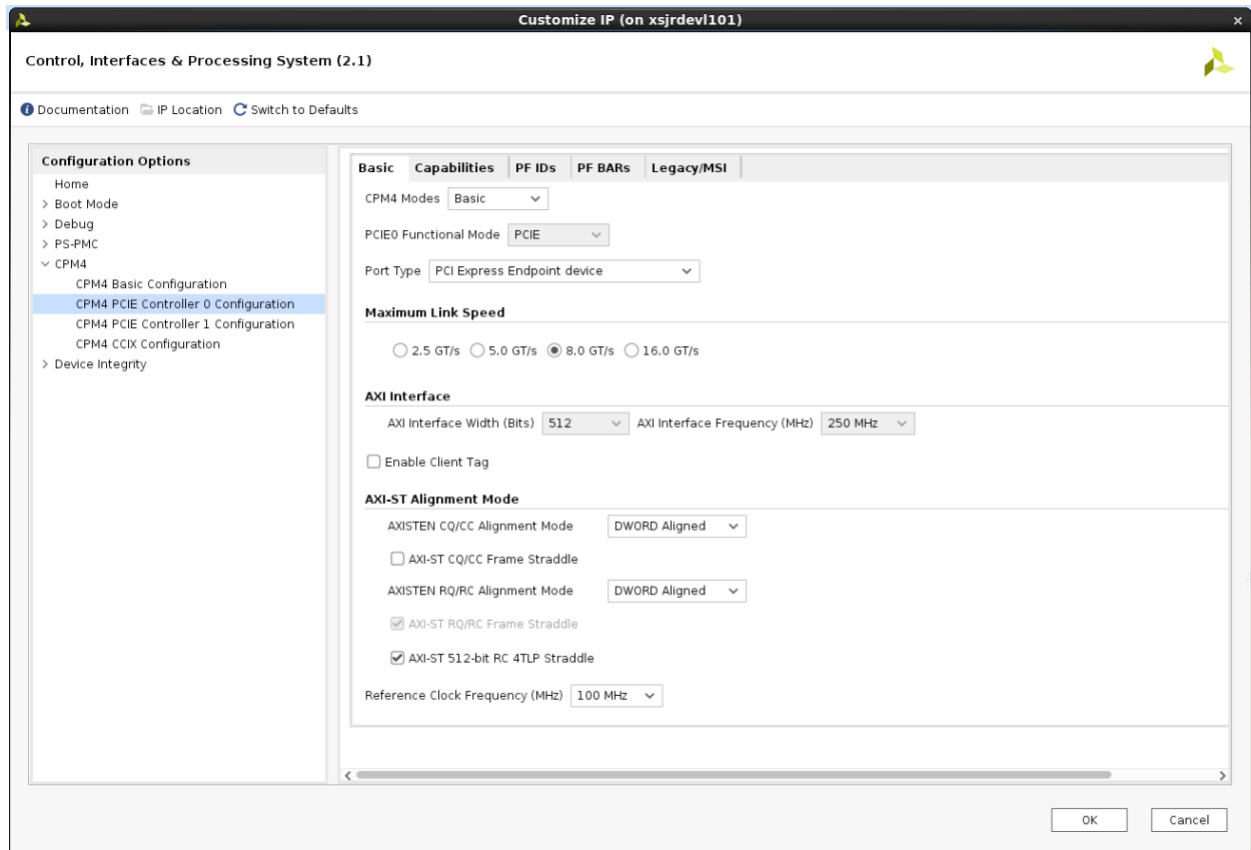
Basic Mode Parameters

The Basic mode parameters are explained in this section.

Basic Tab

The following figure shows the initial customization page, used to set the Basic mode parameters.

Figure 10: Basic Tab



- **Component Name:** Base name of the output files generated for the core. The name must begin with a letter and can be composed of these characters: a to z, 0 to 9, and “_.”
- **CPM Mode:** Allows you to select the Basic or Advanced mode configuration of the core.
- **Port Type:** Indicates the PCI Express logical device type.
- **Maximum Link Speed:** The core allows you to select the Maximum Link Speed supported by the device. Higher link speed cores are capable of training to a lower link speed if connected to a lower link speed capable device.
- **AXI-ST Interface Frequency:** Enables you to specify the AXI-ST Interface frequency.
- **AXI-ST Interface Width:** The core allows you to select the Interface Width. The default interface width set in the Customize IP dialog box is the lowest possible interface width.
- **AXI-ST Alignment Mode:** When a payload is present, there are two options for aligning the first byte of the payload with respect to the datapath. The options are provided to select the CQ/CC and RQ/RC interfaces.
- **Enable AXI-ST Frame Straddle:** The core provides an option to straddle packets on the requester completion interface when the interface width is 256 bits.

- **AXI-ST CQ/CC Frame Straddle and AXI-ST RQ/RC Frame Straddle:** When 512-bit AXI-ST interface width is selected AXI-ST frame Straddle is supported for CQ, CC, RQ and RC AXI-ST interfaces. Option to select CQ and CC AXI-ST frame straddle together and for RQ and RC interfaces.
- **Enable Client Tag:** Enables you to use the client tag.
- **Reference Clock Frequency:** Selects the frequency of the reference clock provided on `gt_refclk(n)`, where *n* refers to the selected PCIe Controller.

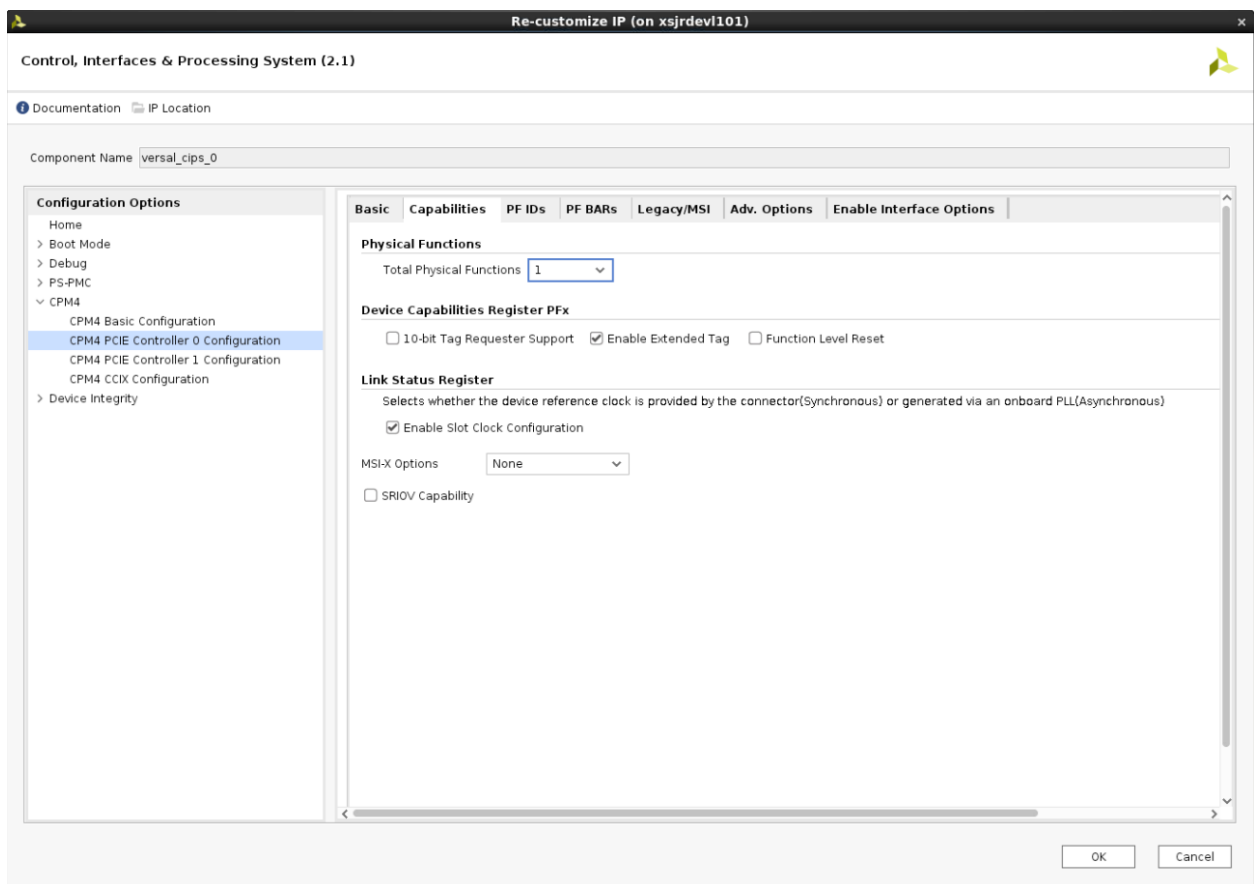
Related Information

[Clocking](#)

Capabilities Tab

The Capabilities settings are explained in this section as shown in the following figure.

Figure 11: Capabilities Tab



- **Total Physical Functions:** Enables you to select the number of physical functions. The number of physical functions supported is 4.

- **PFx Max Payload Size:** This field indicates the maximum payload size that the device or function can support for TLPs. This is the value advertised to the system in the Device Capabilities Register.
- **Extended Tag Field:** This field indicates the maximum supported size of the Tag field. The options are:
 - When selected, 8-bit Tag field support (256 tags)
 - When deselected, 5-bit Tag field support (32 tags)
- **10-bit Tag Requester Support:** This field indicates the maximum supported size of the Tag field as a Requester. The options are:
 - When selected, 10-bit Requester Tag field support (768 tags)
 - When deselected, 8-/5-bit Tag supported, depending on Extended Tag Field selection
- **Enable Slot Clock Configuration:**

Enables the Slot Clock Configuration bit in the Link Status register.

 - When this option is selected, the link is synchronously clocked.
 - When this option is deselected, asynchronous clock in SRNS mode is supported. SRNS refers to a separate reference clock with No SSC (an asynchronous clock without SRIS support).

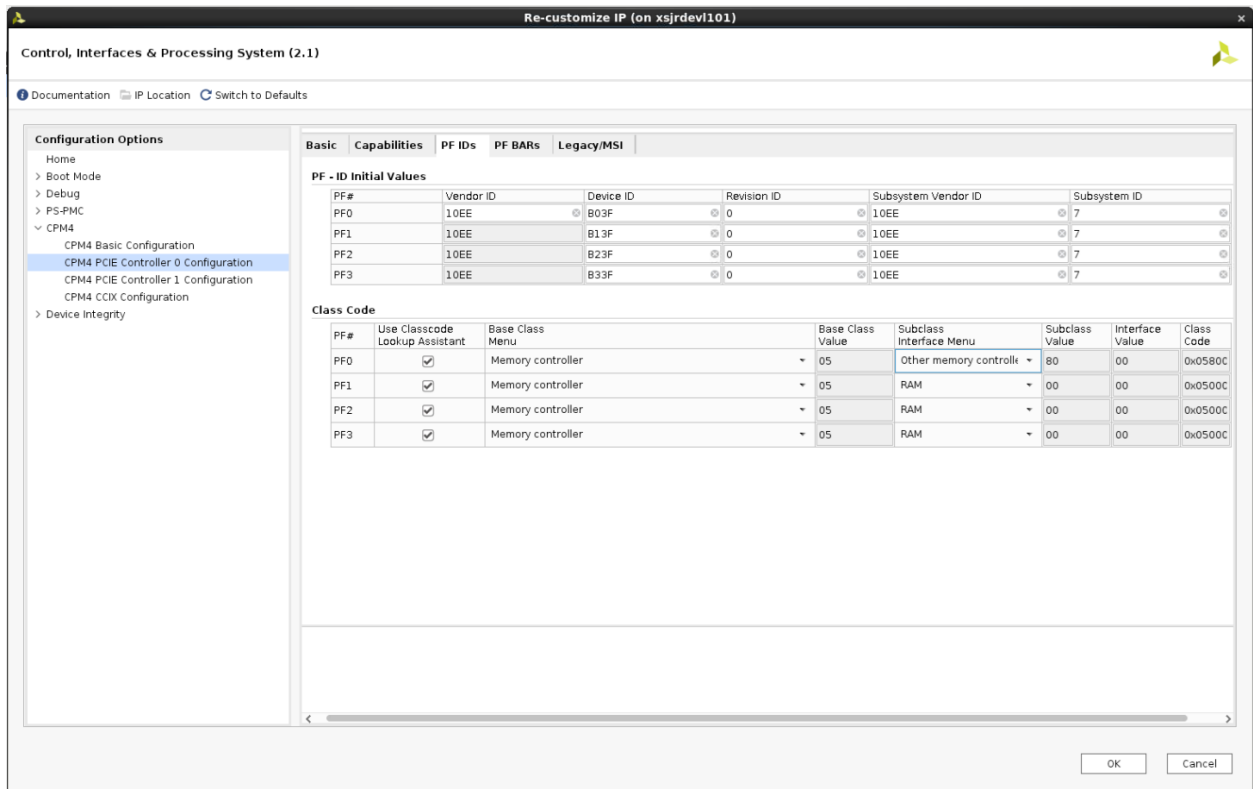
Related Information

[Clocking](#)

PF IDs Tab

The following figure shows the Identity Settings parameters.

Figure 12: PF IDs Tab



- **PF0 ID Initial Values:**

- **Vendor ID:** Identifies the manufacturer of the device or application. Valid identifiers are assigned by the PCI Special Interest Group to guarantee that each identifier is unique. The default value, 10EEh, is the Vendor ID for Xilinx. Enter a vendor identification number here. FFFFh is reserved.
- **Device ID:** A unique identifier for the application; the default value depends on the configuration selected.
- **Revision ID:** Indicates the revision of the device or application; an extension of the Device ID. The default value is 00h; enter a value appropriate for the application.
- **Subsystem Vendor ID:** Further qualifies the manufacturer of the device or application. Enter a Subsystem Vendor ID here; the default value is 10EEh. Typically, this value is the same as Vendor ID. Setting the value to 0000h can cause compliance testing issues.
- **Subsystem ID:** Further qualifies the manufacturer of the device or application. This value is typically the same as the Device ID; the default value depends on the lane width and link speed selected. Setting the value to 0000h can cause compliance testing issues.
- **Class Code:** The Class Code identifies the general function of a device, and is divided into three byte-size fields:

- **Base Class:** Broadly identifies the type of function performed by the device.
- **Sub-Class:** More specifically identifies the device function.
- **Interface:** Defines a specific register-level programming interface, if any, allowing device-independent software to interface with the device.

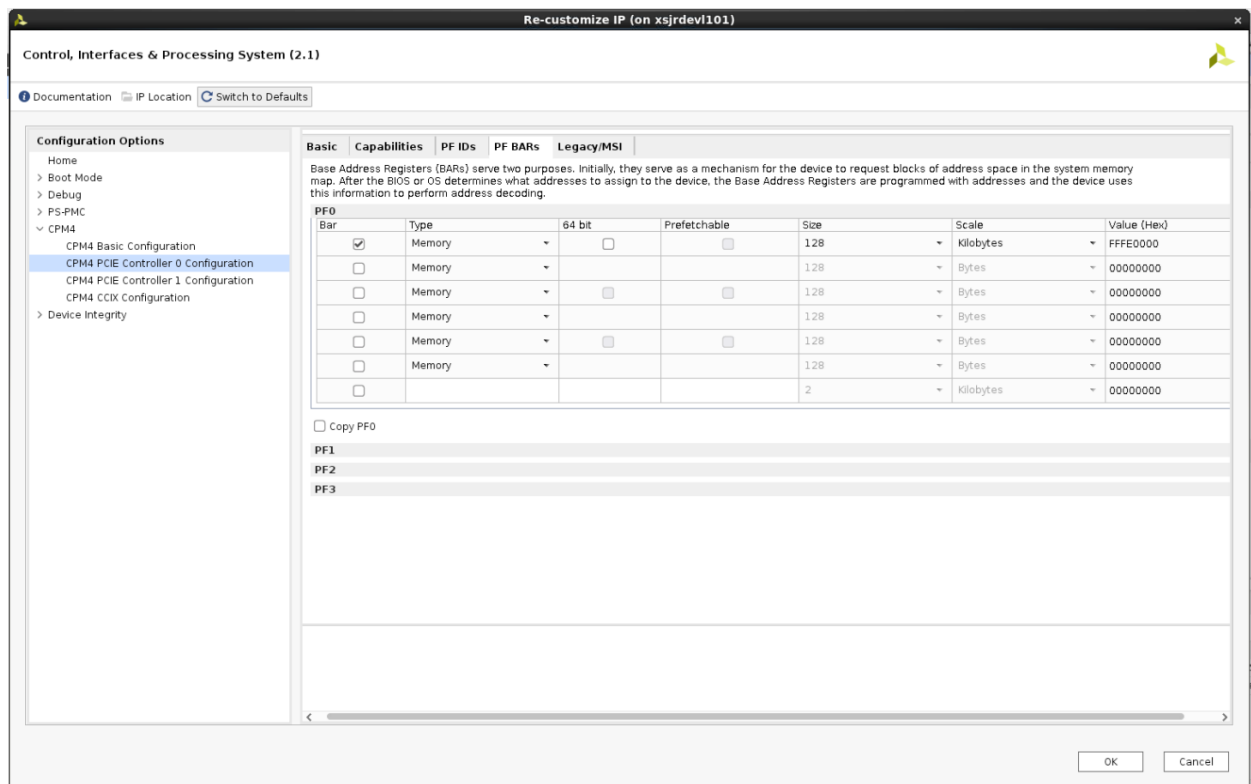
Class code encoding can be found at the [PCI-SIG website](#).

- **Class Code Look-up Assistant:** The Class Code Look-up Assistant provides the Base Class, Sub-Class and Interface values for a selected general function of a device. This Look-up Assistant tool only displays the three values for a selected function. You must enter the values in Class Code for these values to be translated into device settings.

PF BARs Tab

The PF BARs tab, shown in the following figure, sets the base address register space for the Endpoint configuration. Each BAR (0 through 5) configures the BAR Aperture Size and Control attributes of the physical function.

Figure 13: PF BARs Tab Showing PF0 and PF1 Only



- Base Address Register Overview:** In Endpoint configuration, the core supports up to six 32-bit BARs or three 64-bit BARs, and the Expansion read-only memory (ROM) BAR. In Root Port configuration, the core supports up to two 32-bit BARs or one 64-bit BAR, and the Expansion ROM BAR. BARs can be one of two sizes:
 - 32-bit BARs:** The address space can be as small as 128 bytes or as large as 2 gigabytes. Used for Memory or I/O.
 - 64-bit BARs:** The address space can be as small as 128 bytes or as large as 8 Exabytes. Used for Memory only.

All BAR registers share these options:

- Checkbox:** Click the checkbox to enable the BAR; deselect the checkbox to disable the BAR.
- Type:** Bars can either be I/O or Memory.
 - I/O:** I/O BARs can only be 32-bit; the Prefetchable option does not apply to I/O BARs. I/O BARs are only enabled for a Legacy PCI Express Endpoint.
 - Memory:** Memory BARs can be either 64-bit or 32-bit and can be prefetchable. When a BAR is set as 64 bits, it uses the next BAR for the extended address space and makes the next BAR inaccessible.
- Size:** The available Size range depends on the PCIe Device/Port Type and the Type of BAR selected. The following table lists the available BAR size ranges.

Table 35: **BAR Size Ranges for Device Configuration**

PCIe Device / Port Type	BAR Type	BAR Size Range
PCI Express Endpoint	32-bit Memory	128 bytes (B) – 2 gigabytes (GB)
	64-bit Memory	128 B – 8 Exabytes
Legacy PCI Express Endpoint	32-bit Memory	128 B – 2 GB
	64-bit Memory	128 B – 8 Exabytes
	I/O	16 B – 2 GB

- Prefetchable:** Identifies the ability of the memory space to be prefetched.
- Value:** The value assigned to the BAR based on the current selections.
- Expansion ROM Base Address Register:** If selected, the Expansion ROM is activated and can be sized from 2 KB to 4 GB. According to the PCI Local Bus Specification Revision 3.0 on the [PCI-SIG website](#), the maximum size for the Expansion ROM BAR should be no larger than 16 MB. Selecting an address space larger than 16 MB can cause compliance testing issues.
- Managing Base Address Register Settings:** Memory, I/O, Type, and Prefetchable settings are handled by setting the appropriate settings for the desired base address register.

Memory or I/O settings indicate whether the address space is defined as memory or I/O. The base address register only responds to commands that access the specified address space. Generally, memory spaces less than 4 KB in size should be avoided. The minimum I/O space allowed is 16 bytes; use of I/O space should be avoided in all new designs.

Prefetchability is the ability of memory space to be prefetched. A memory space is prefetchable if there are no side effects on reads (that is, data is not destroyed by reading, as from a RAM). Byte-write operations can be merged into a single double word write, when applicable.

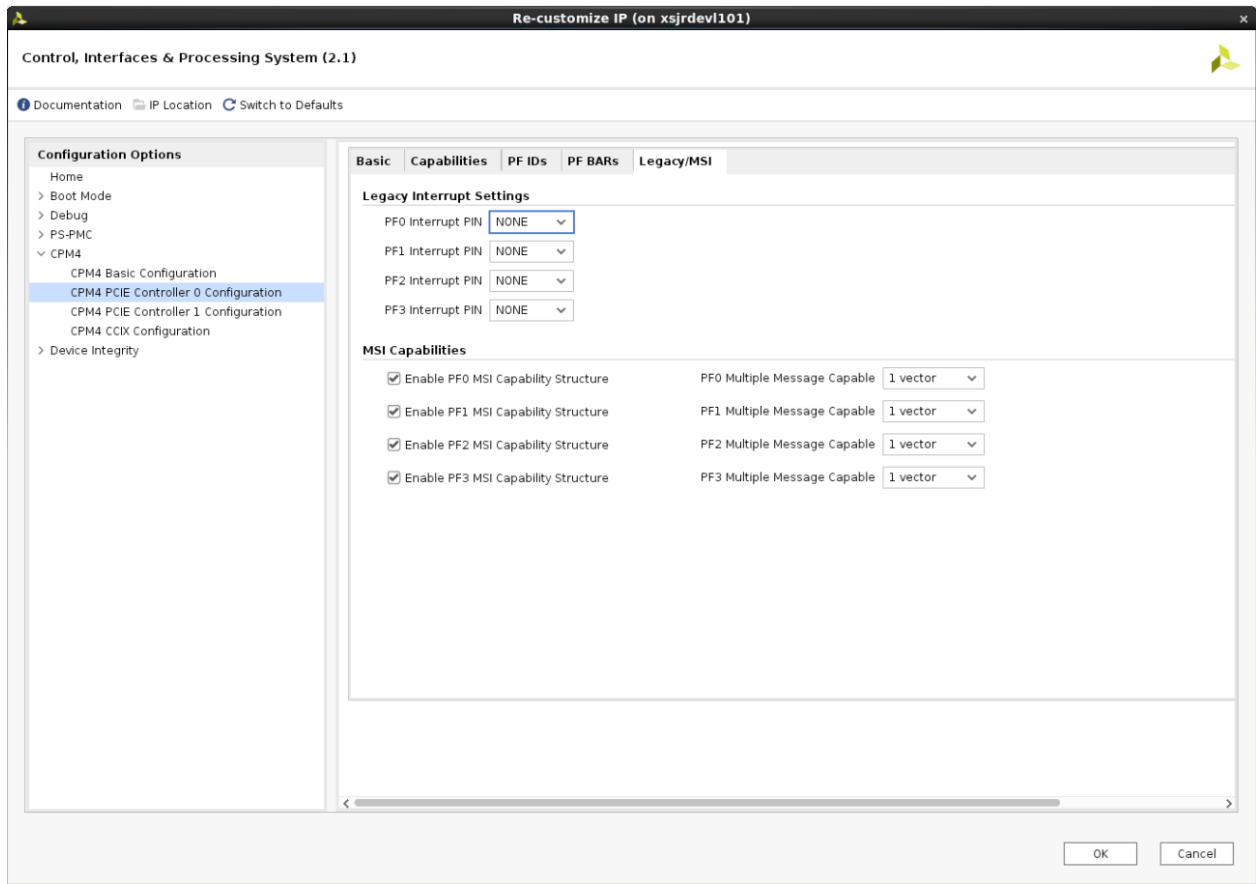
When configuring the core as an Endpoint for PCIe (non-Legacy), 64-bit addressing must be supported for all BARs (except BAR5) that have the prefetchable bit set. 32-bit addressing is permitted for all BARs that do not have the prefetchable bit set. The prefetchable bit-related requirement does not apply to a Legacy Endpoint. The minimum memory address range supported by a BAR is 128 bytes for a PCI Express Endpoint and 16 bytes for a Legacy PCI Express Endpoint.

- **Disabling Unused Resources:** For best results, disable unused base address registers to conserve system resources. A base address register is disabled by deselecting unused BARs in the Customize IP dialog box.
- **Copy PF0:** When set, the Copy PF0 option allows you to set all BARs settings of the remaining PFs to the same values as PF0. Applicable when there are more than one total Physical Function (PF).

Legacy/MSI Cap Tab

On this page, you set the Legacy Interrupt Settings and MSI Capabilities for all applicable physical and virtual functions. This page is not visible when the **SRIOV Capability** parameter is selected on the Capabilities page.

Figure 14: Legacy/MSI Cap Tab



- **Legacy Interrupt Settings:**

- **PF0/PF1/PF2/PF3 Interrupt PIN:** Indicates the mapping for Legacy Interrupt messages. A setting of None indicates that no Legacy Interrupts are used.

Note: When PASID is enabled, legacy interrupts cannot be used and are disabled.

- **MSI Capabilities:**

- **PF0/PF1/PF2/PF3 Enable MSI Capability Structure:** Indicates that the MSI Capability structure exists.

Note: Although it is possible to not enable MSI or MSI-X, the result would be a non-compliant core. The PCI Express Base Specification requires that MSI, MSI-X, or both be enabled. No MSI capabilities are supported when **MSI-X Internal** is enabled in the [MSI-X Capabilities Tab](#) (Advanced mode), because MSI-X Internal uses some of the MSI interface signals.

- **PF0/PF1/PF2/PF3 Multiple Message Capable:** Selects the number of MSI vectors to request from the Root Complex.
- **Enable MSI Per Vector Masking:** Enables MSI Per Vector Masking Capability of all the Physical functions enabled.

Note: Enabling this option for individual physical functions is not supported.

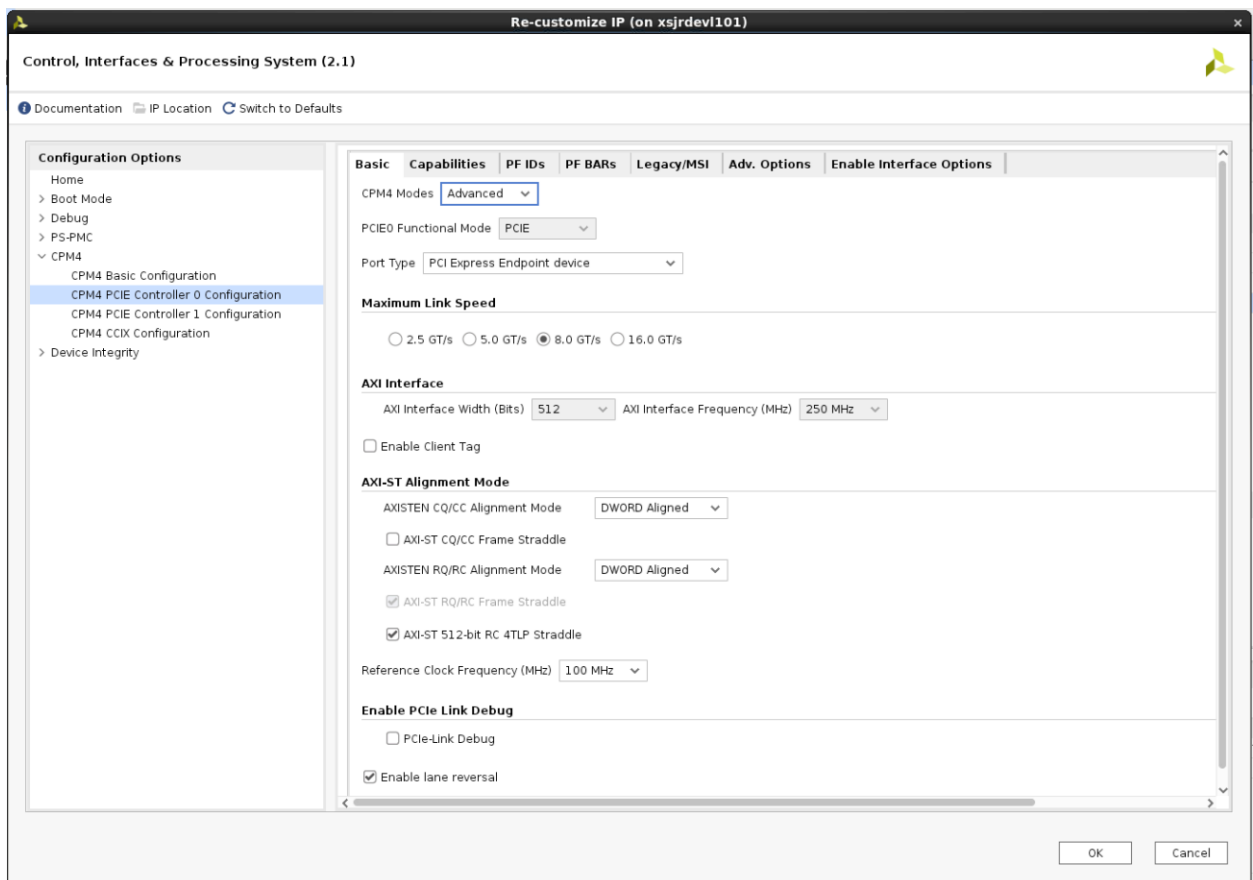
Advanced Mode Parameters

The following parameters appear on different pages of the IP catalog when **Advanced** mode is selected for Mode on the Basic page.

Basic Tab

The Basic page with Advanced mode selected (shown in the following figure) includes additional settings. The following parameters are visible on the Basic page when the **Advanced** mode is selected.

Figure 15: Basic Tab, Advanced Mode

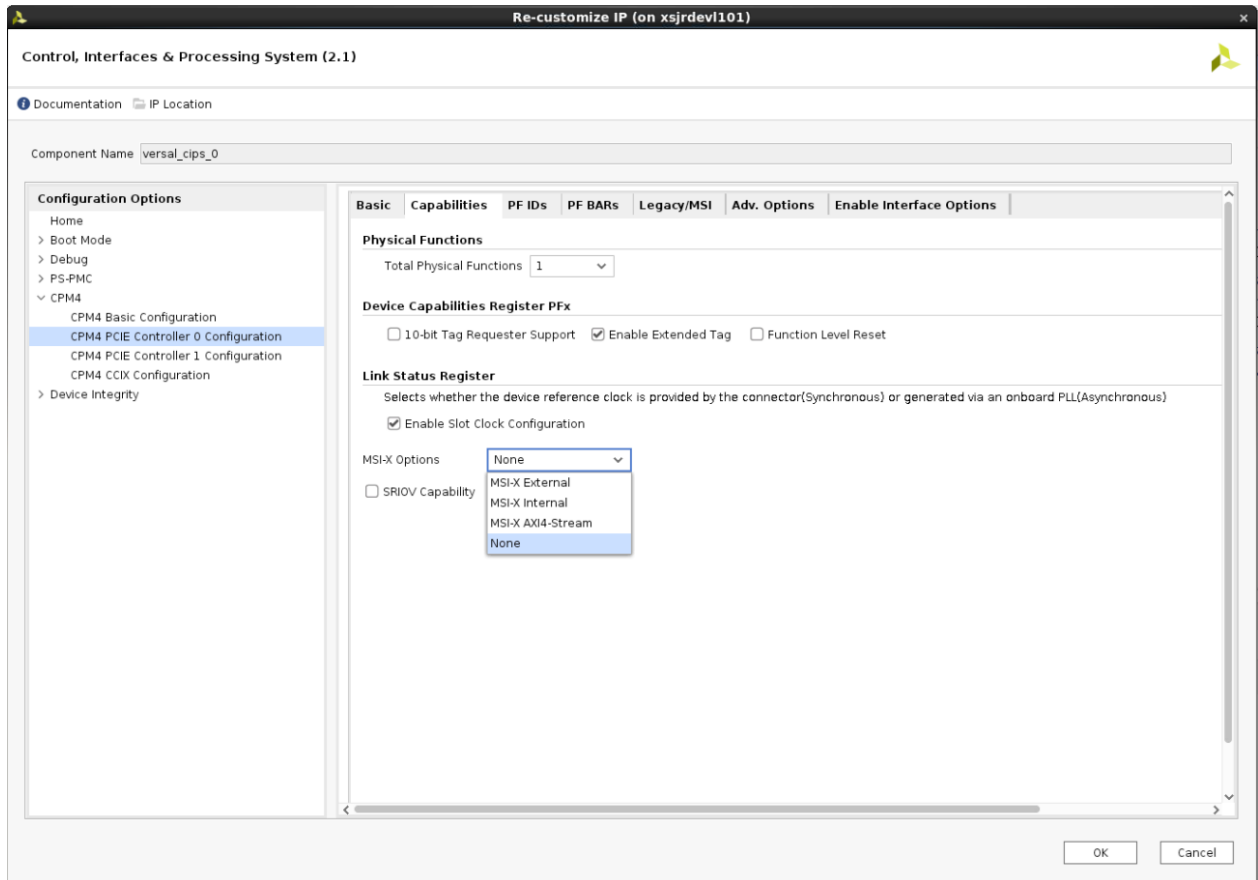


- **PCIe Link Debug:** This enables the link debug option to be activated.
- **Enable Lane Reversal:** This enables the lane reversal feature.

Capabilities Tab

The Capabilities settings for Advanced mode (as shown in the following figure) contains two additional parameters to those for Basic mode and are described below.

Figure 16: Capabilities Tab, Advanced Mode



- **Function Level Reset:** Enable Function Level Reset (FLR). FLR is supported when the PCIe IP is configured as Endpoint.
- **SRIOV Capabilities:** Enables Single Root Port I/O Virtualization (SR-IOV) capabilities. The integrated block implements extended Single Root Port I/O Virtualization PCIe. When this is enabled, SR-IOV is implemented on all the selected physical functions. When SR-IOV capabilities are enabled MSI support is disabled and you can use MSI-X support as shown in the above figure.

Note: When SR-IOV capabilities are enabled, MSI support is disabled and you can use MSI-X support.

- **MSI-X Options:** To enable MSI-X capabilities, select **Advanced** mode and then select the required options on the Capabilities tab. There are four options to choose from:
 - **MSI-X External:** In this mode you need to implement MSI-X External interface driving logic, MSI-X Table and PBA buffers outside the PCIe core. You can configure the MSI-X BARs.

- **MSI-X Internal:** In this mode you need to implement the MSI-X Internal interface driving logic only. MSI-X Table and PBA buffers are built into the PCIe core. You can configure the MSI-X BARs.
- **MSI-X AXI4-Stream:** In this mode user is expected to drive MSI-X interrupts on the AXI4-Stream interface. You can configure the MSI-X BARs.
- **None:** No MSI-X is supported.

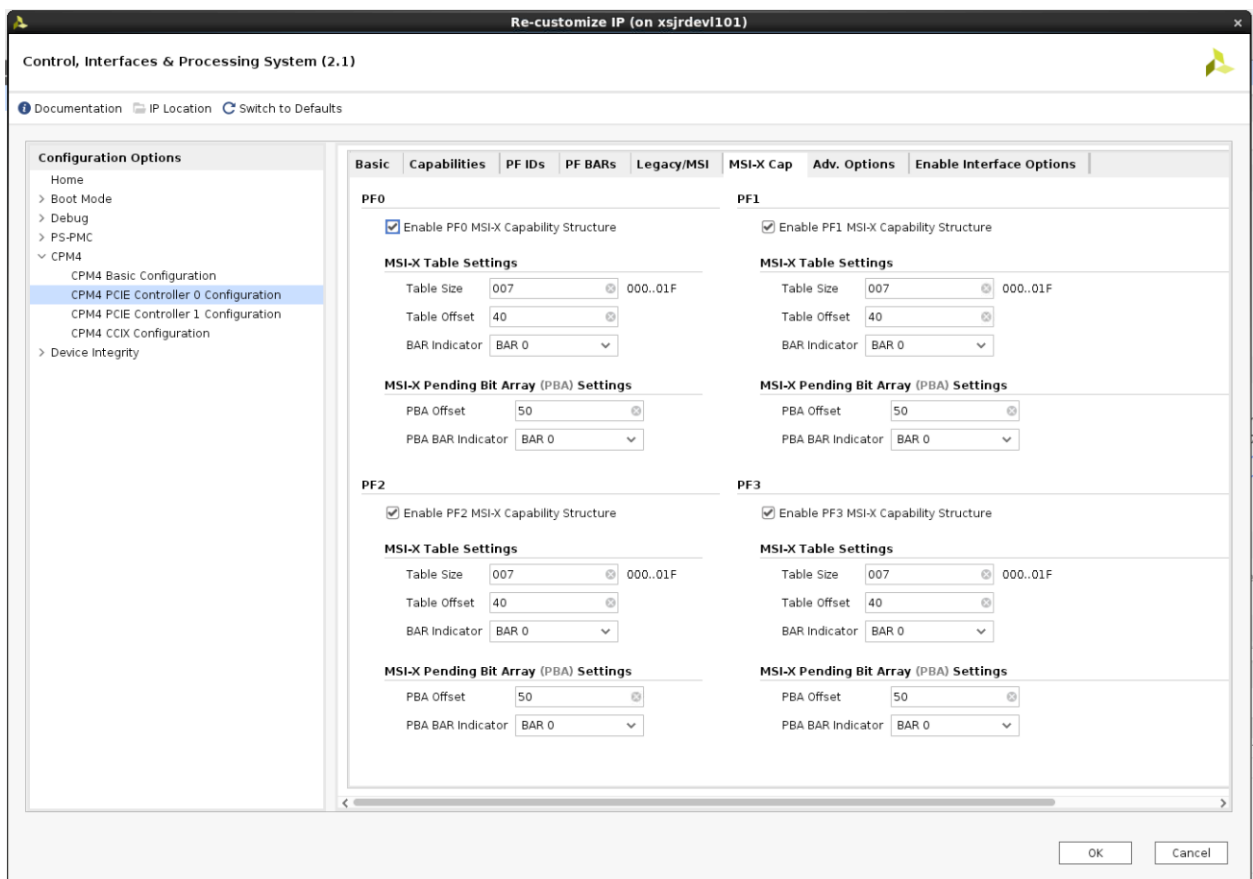
The same MSI-X options are applicable when SRIOV capability is selected.

MSI-X Capabilities Tab

The MSI-X Capabilities parameters, shown in the following figure, are available in Advanced mode only. To enable MSI-X capabilities, select **Advanced** mode and then select the required options on the Capabilities page.

The same MSI-X options are applicable when SRIOV capability is selected.

Figure 17: MSI-X Cap Tab, Advanced Mode



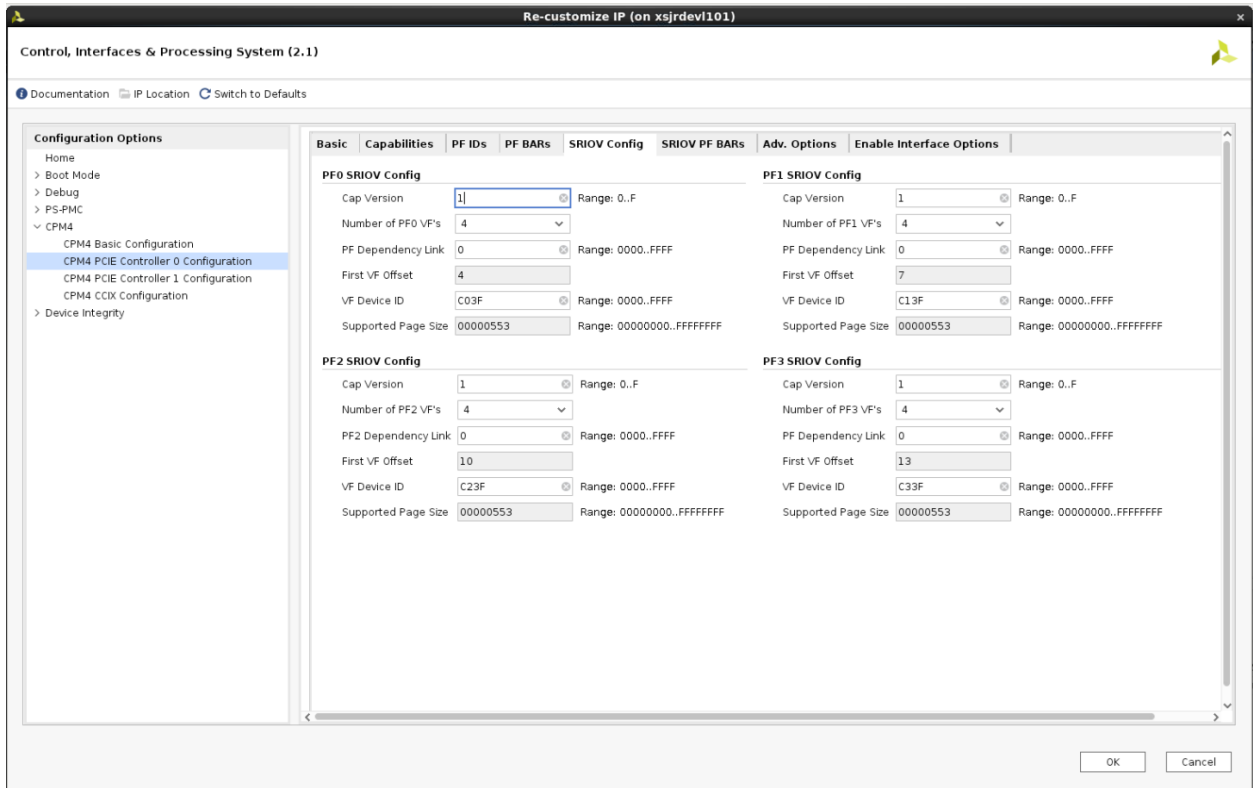
- **Enable MSI-X Capability Structure:** Indicates that the MSI-X Capabilities structure exists.

Note: The Capability Structure needs at least one Memory BAR to be configured. You must maintain the MSI-X Table and Pending Bit Array in the application.

- **MSI-X Table Settings:** Defines the MSI-X Table structure.
 - **Table Size:** Specifies the MSI-X Table size. Table Size field is expecting N-1 interrupts (0x0F will configure a table count of 16).
 - **Table Offset:** Specifies the offset from the Base Address Register that points to the base of the MSI-X Table.
 - **BAR Indicator:** Indicates the Base Address Register in the Configuration Space used to map the function in the MSI-X Table onto memory space. For a 64-bit Base Address Register, this indicates the lower DWORD.
- **MSIx Pending Bit Array (PBA) Settings:** Defines the MSI-X Pending Bit Array (PBA) structure.
 - **PBA Offset:** Specifies the offset from the Base Address Register that points to the base of the MSI-X PBA.
 - **PBA BAR Indicator:** Indicates the Base Address Register in the Configuration Space used to map the function in the MSI-X PBA onto Memory Space.

SRIOV Config Tab

The SRIOV Configuration parameters, as shown in the following figure, are described in this section.



- **Cap Version:** Indicates the 4-bit SR-IOV Capability version for the physical function.
- **Number of PFx VF's:** Indicates the number of virtual functions associated to the physical function. A total of 252 virtual functions are available that can be flexibly used across the four physical functions.
- **PFx Dependency Link:** Indicates the SR-IOV Functional Dependency Link for the physical function. The programming model for a device can have vendor-specific dependencies between sets of functions. The Function Dependency Link field is used to describe these dependencies.
- **First VF Offset:** Indicates the offset of the first virtual function (VF) for the physical function (PF). PFx offset is always fixed. PF0 resides at offset 0, PF1 resides at offset 1, PF2 resides at offset 2, and PF3 resides at offset 3.

A total of 252 virtual functions are available. They reside at the function number range 4 to 255.

When ARI is enabled, allowed value is 'd4', and the total number of VF in all PFs plus this field must not be greater than 256. When ARI is disabled, this field will be set to 1 to support 1 PF plus 7 VF non-ARI SR-IOV configurations only.

Virtual functions are mapped sequentially with VFs with PFs taking precedence. For example, if PF0 has two virtual functions and PF1 has three, the following mapping occurs:

The `PFx_FIRST_VF_OFFSET` is calculated by taking the first offset of the virtual function and subtracting that from the offset of the physical function.

```
PFx_FIRST_VF_OFFSET = (PFx first VF offset - PFx offset)
```

In the example above, the following offsets are used:

```
PF0_FIRST_VF_OFFSET = (4 - 0) = 4
PF1_FIRST_VF_OFFSET = (6 - 1) = 5
```

The initial offset for PF1 is a function of how many VFs are attached to PF0 and is defined in the following pseudo code:

```
PF1_FIRST_VF_OFFSET = FIRST_VF_OFFSET + NUM_PF0_VFs - 1
```

Similarly, for other PFs:

```
PF2_FIRST_VF_OFFSET = FIRST_VF_OFFSET + NUM_PF0_VFs + NUM_PF1_VFs - 2
PF3_FIRST_VF_OFFSET =
    FIRST_VF_OFFSET + NUM_PF0_VFs + NUM_PF1_VFs + NUM_PF2_VFs - 3
```

- **VF Device ID:** Indicates the 16-bit Device ID for all virtual functions associated with the physical function.
- **SRIOV Supported Page Size:** Indicates the page size supported by the physical function. This physical function supports a page size of $2^{(n+12)}$, if bit n of the 32-bit register is set.

SRIOV PF BARs Tab

The SRIOV Base Address Registers (BARs) set the base address register space for the Endpoint configuration. Each BAR (0 through 5) configures the SR-IOV BAR aperture size and SR-IOV control attributes.

Figure 18: SRIOV BARs Tab, Advanced Mode

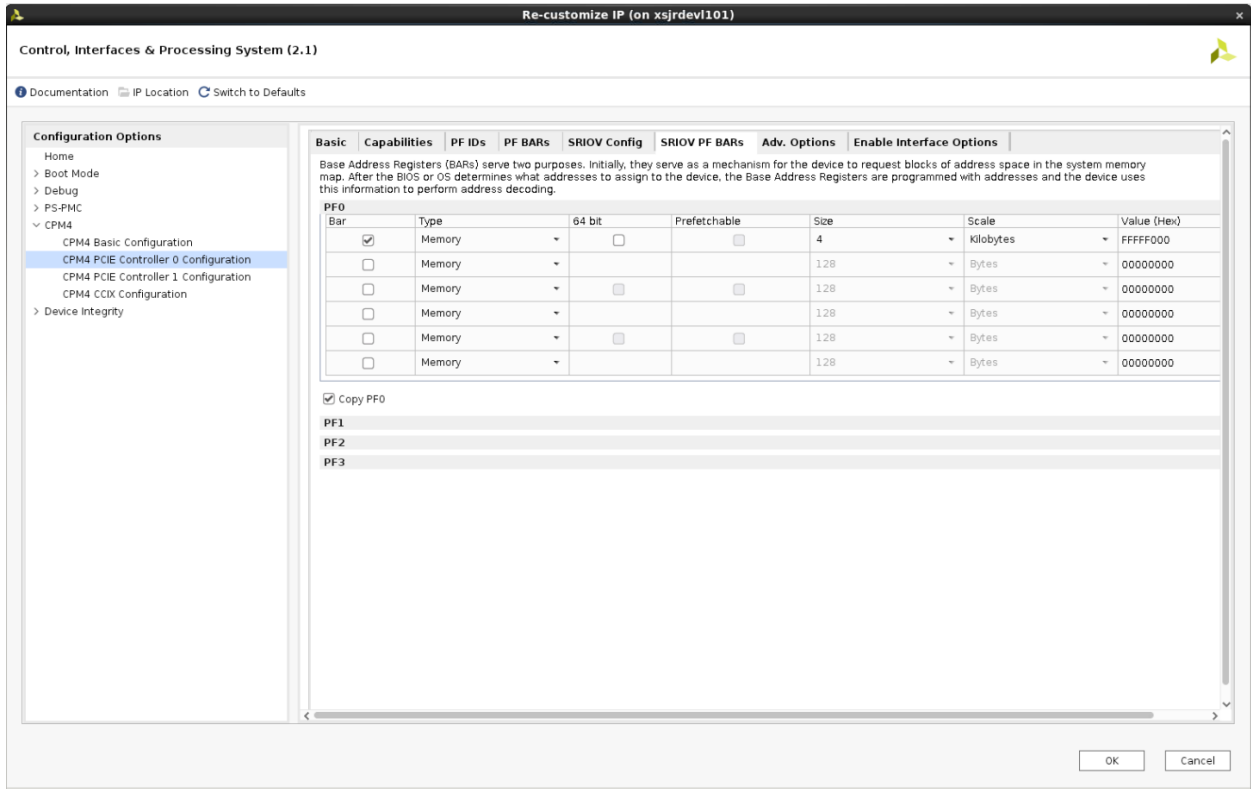


Table 36: Example Virtual Function Mappings

Physical Function	Virtual Function	Function Number Range
PF0	VF0	64
PF0	VF1	65
PF1	VF0	68
PF1	VF1	69
PF1	VF1	70

- SRIOV Base Address Register Overview:** In Endpoint configuration, the core supports up to six 32-bit BARs or three 64-bit BARs. In Root Port configuration, the core supports up to two 32-bit BARs or one 64-bit BAR. SR-IOV BARs can be one of two sizes:
 - 32-bit BARs:** The address space can be as small as 16 bytes or as large as 3 gigabytes. Used for memory to I/O.
 - 64-bit BARs:** The address space can be as small as 128 bytes or as large as 256 gigabytes. Used for memory only.

All SR-IOV BAR registers have these options:

- **Checkbox:** Click the checkbox to enable the BAR; deselect the checkbox to disable the BAR.
- **Type:** SR-IOV BARs can be either I/O or Memory.
 - **I/O:** I/O BARs can only be 32-bit; the Prefetchable option does not apply to I/O BARs. I/O BARs are only enabled for a Legacy PCI Express Endpoint.
 - **Memory:** Memory BARs can be either 64-bit or 32-bit and can be prefetchable. When a BAR is set to 64-bits, it uses the next BAR for the extended address space and makes the next BAR inaccessible.
- **Size:** The available size range depends on the PCIe device/port type and the type of BAR selected. The following table lists the available BAR size ranges.

Table 37: SRIOV BAR Size Ranges for Device Configuration

PCIe Device / Port Type	BAR Type	BAR Size Range
PCI Express Endpoint	32-bit Memory	128 bytes – 2 gigabytes
	64-bit Memory	128 bytes – 8 exabytes
Legacy PCI Express Endpoint	32-bit Memory	16 bytes – 2 gigabytes
	64-bit Memory	16 bytes – 8 exabytes
	I/O	16 bytes – 2 gigabytes

- **Prefetchable:** Identifies the ability of the memory space to be prefetched.
- **Value:** The value assigned to the BAR based on the current selections.
- **Managing SRIOV Base Address Register Settings:** Memory, I/O, Type, and Prefetchable settings are handled by setting the appropriate Customize IP dialog box settings for the desired base address register.

Memory or I/O settings indicate whether the address space is defined as memory or I/O. The base address register only responds to commands that access the specified address space. Generally, memory spaces less than 4 KB in size should be avoided. The minimum I/O space allowed is 16 bytes. I/O space should be avoided in all new designs.

A memory space is prefetchable if there are no side effects on reads (that is, data is not destroyed by reading, as from RAM). Byte-write operations can be merged into a single double-word write, when applicable.

When configuring the core as an Endpoint for PCIe (non-Legacy), 64-bit addressing must be supported for all SR-IOV BARs (except BAR5) that have the prefetchable bit set. 32-bit addressing is permitted for all SR-IOV BARs that do not have the prefetchable bit set. The prefetchable bit related requirement does not apply to a Legacy Endpoint. The minimum memory address range supported by a BAR is 128 bytes for a PCI Express Endpoint and 16 bytes for a Legacy PCI Express Endpoint.

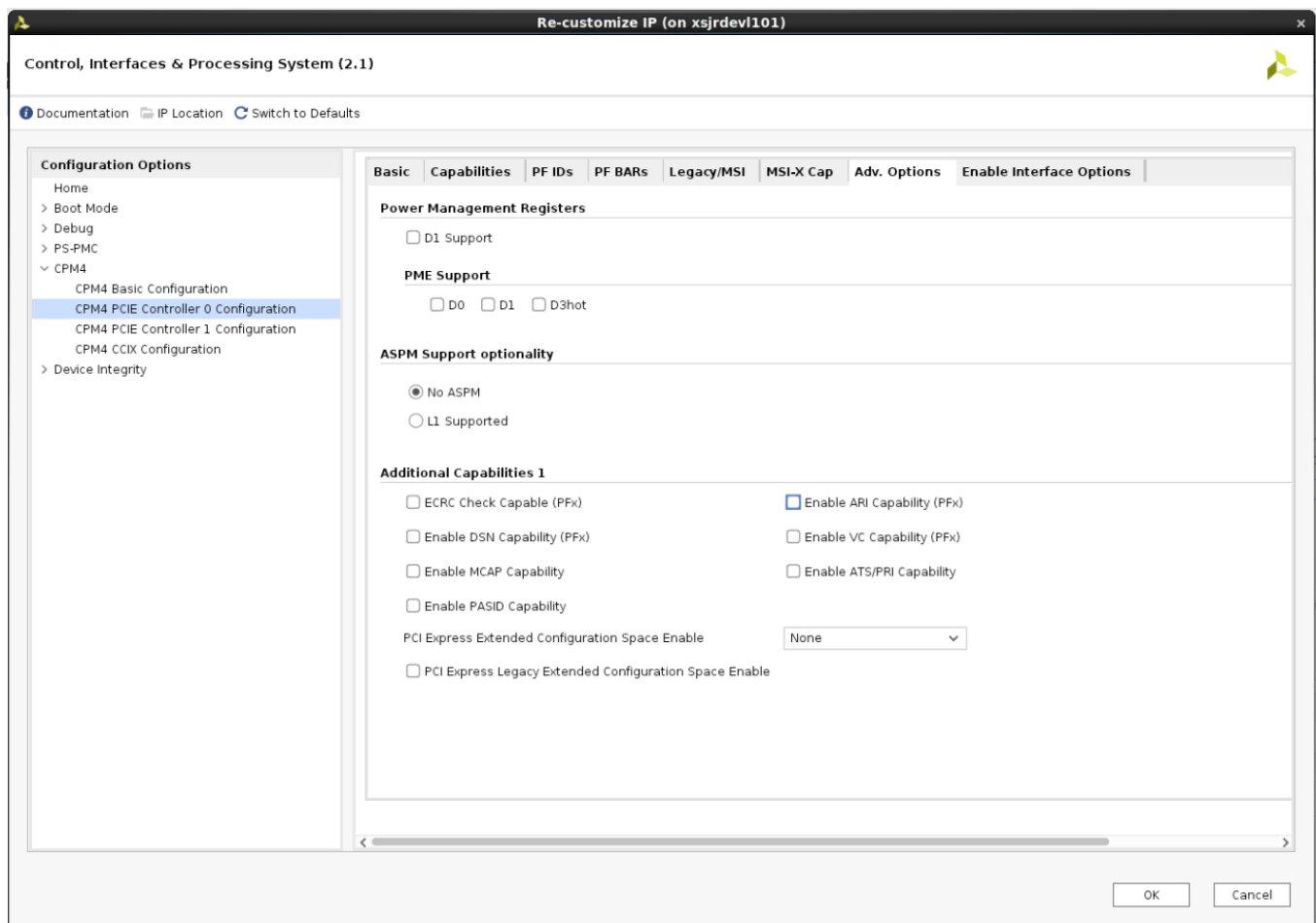
- **Disabling Unused Resources:** For best results, disable unused base address registers to conserve system resources. Disable base address register by deselecting unused BARs in the Customize IP dialog box.
- **Copy PF0:** When set, the Copy PF0 option allows you to set all BAR settings of the remaining PF groups to the same values as PF0 group. Applicable when there are more than one total Physical Function (PF).

Adv. Options

The Advanced Options tab enables Power Management Registers and ASPM. There is no L0s support when the link speed is not 2.5 GT/s and 5.0 GT/s.

The Advanced Options tab also enables you to choose ECRC check in AER Capability, ARI, DSN, Virtual Channel, MCAP, ATS/PRI, PASID, and PCI Express Extended Config space and PCI Legacy Extended space.

Figure 19: Adv. Options Tab



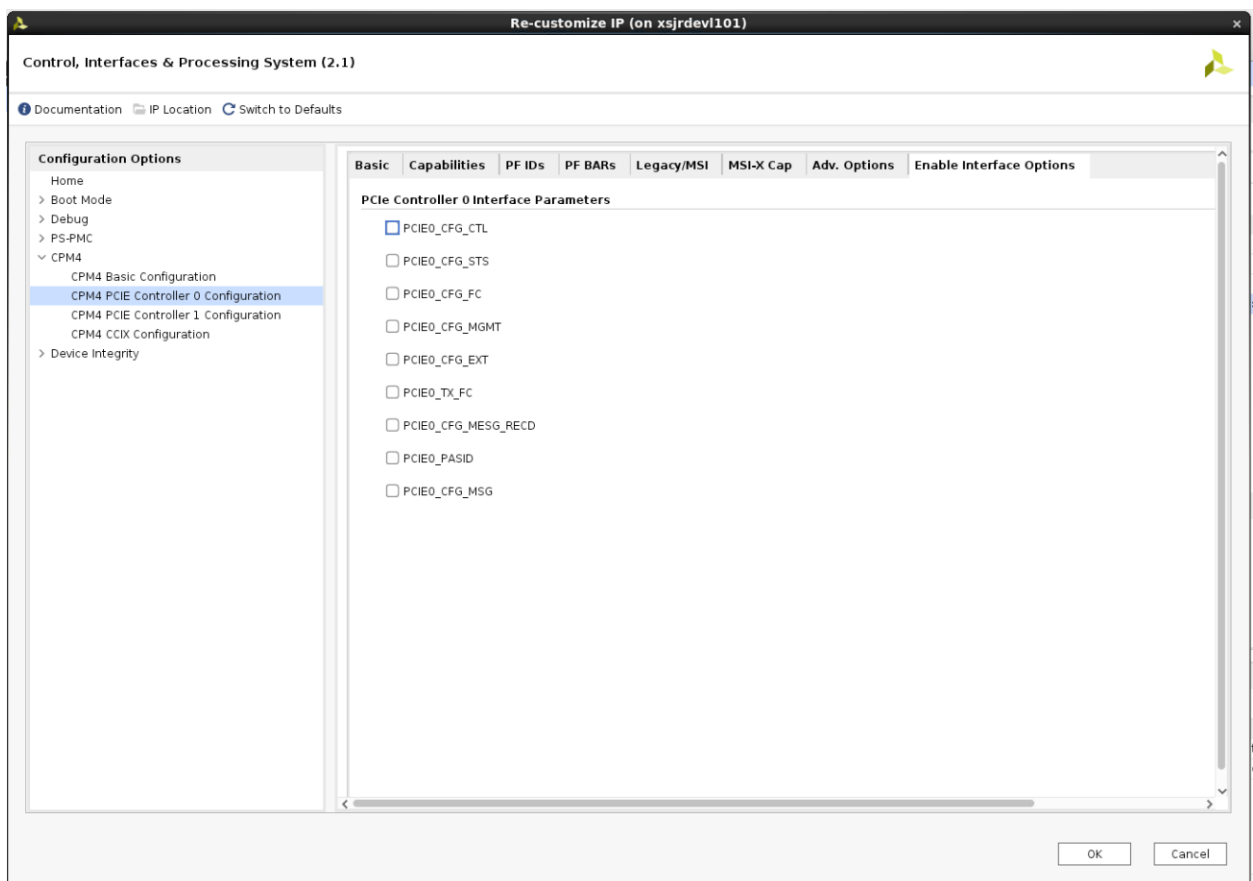
PCI Express Extended Configuration Space has three options:

- **None:** No Extended Configuration space required.
- **Extended Small:** PCI Express Extended space available from 0xE00 - 0xFFC.
- **Extended Large:** PCI Express Extended space available from 0x600 - 0xFFC.

Interface Parameters

The Interface Parameters tab enables you to enable/disable interfaces that are not required for your application.

Figure 20: Interface Parameters Tab



GT Selection and Pin Planning for CPM4

This appendix provides guidance on gigabit transceiver (GT) selection for Versal® devices and some key recommendations that should be considered when selecting the GT locations. The GT locations for Versal devices can be customized through the IP customization wizard. This appendix provides guidance for CPM, PL PCIe® and PHY IP based solutions. In this guide, the CPM related guidance is of primary importance, while the other related guidance might be relevant and is provided for informational purposes.

A GT Quad is comprised of four GT lanes. GT Quad and ref clock locations for CPM4 are in fixed locations depending on the desired link configuration (see [GT Quad Locations](#)). When selecting GT Quads for the PL PCIe-based solution Xilinx® recommends that you use the GT Quad most adjacent to the integrated block. While this is not required, it will improve place, route, and timing for the design.

- Link widths of x1, x2, and x4 require one bonded GT Quad and should not split lanes between two GT Quads.
- A link width of x8 requires two adjacent GT Quads that are bonded and are in the same SLR.
- A link width of x16 requires four adjacent GT Quads that are bonded and are in the same SLR.
- PL PCIe blocks should use GTs adjacent to the PCIe block where possible.
- CPM has a fixed connectivity to GTs based on the CPM configuration.

For GTs on the **left side of the device**, PCIe lane 0 is placed in the bottom-most GT of the bottom-most GT Quad. Subsequent lanes use the next available GTs moving vertically up the device as the lane number increments. This means that the highest PCIe lane number uses the top-most GT in the top-most GT Quad that is used for PCIe.

For GTs on the **right side of the device**, PCIe lane 0 is placed in the top-most GT of the top-most GT Quad. Subsequent lanes use the next available GTs moving vertically down the device as the lane number increments. This means that the highest PCIe lane number uses the bottom-most GT in the bottom-most GT Quad that is used for PCIe.

The PCIe reference clock uses GTREFCLK0 in the PCIe lane 0 GT Quad for x1, x2, x4, and x8 configurations. For x16 configurations the PCIe reference clock should use GTREFCLK0 on a GT Quad associated with lanes 8-11. This allows the clock to be forwarded to all 16 PCIe lanes.

The PCIe reset pins for CPM designs must connect to one of specified pins for each of the two PCIe controllers. The PCIe reset pin for PL PCIe and PHY IP designs can be connected to any compatible PL pin location, or the CPM PCIe reset pins when the corresponding CPM PCIe controller is not in use. This is summarized in the table below.

Table 38: PCIe Controller Reset Pin Locations

Versal PCIe Controller	Versal Reset Pin Location
CPM PCIe Controller 0	PS MIO 18
	PMC MIO 24
	PMC MIO 38
CPM PCIe Controller 1	PS MIO 19
	PMC MIO 25
	PMC MIO 39
PL PCIe Controllers	Any compatible single-ended PL I/O pin.
Versal ACAP PHY IP	Any compatible single-ended PL I/O pin.

CPM4 GT Selection

The CPM block within Versal devices has a fixed set of GTs that can be used for each of the two PCIe controllers. These GTs are shared between the two PCIe controllers and High Speed Debug Port (HSDP) as such x16 link widths are only supported when a single PCIe controller is in use and HSDP is disabled. When two CPM PCIe controllers or one PCIe controller and HSDP are enabled each link will be limited to a x8 link width. GT Quad allocation for CPM happens at GT Quad granularity and must include all GT Quads from the most adjacent to the CPM to the top-most GT Quad that is in use by the CPM. GT Quads that are used or between GT Quads that are used by the CPM (for either PCIe or HSDP) cannot be shared with PL resources even if GTs within the quad are not in use.

CPM in Single Controller Mode

When a single PCIe controller in the CPM is being used and HSDP is disabled, PCIe x1, x2, x4, x8, and x16 link widths are supported. PCIe lane0 is placed at the bottom-most GT of the bottom-most GT Quad that is directly above the CPM. Subsequent lanes use the next available GTs moving vertically up the device as the lane number increments. This means the highest PCIe lane number uses the top-most GT in the top-most GT Quad that is used for PCIe. Because the GT locations and lane ordering for CPM is fixed it cannot be modified through IP customization.

As stated previously GT Quad allocation happens at GT Quad granularity and cannot share unused GT Quad resources with the PL. This means that CPM PCIe controller 0 configurations that use x1 or x2 link widths will not use all the GTs within the Quad and that these GTs cannot be used in the PL for additional GT connectivity. Unused GT Quads in this configuration can be used by the PL to implement PL GT based solutions.

When CPM PCIe controller 0 and High Speed Debug Port (HSDP) is enabled, a PCIe link width of x16 cannot be used and the CPM will use all three GT Quads that are directly above the CPM regardless of PCIe link width. In this configuration, these GT Quads are allocated to CPM and cannot be shared with PL resources. CPM PCIe lanes 0-7 will be unchanged in their GT selection and lane ordering. HSDP will use the bottom-most GT that is the third GT Quad away from CPM. This corresponds to the same location as PCIe lane 8 for a x16 link configuration. The fourth GT Quad in this configuration is not used by CPM and can be used to implement PL GT based solutions.

CPM in Dual Controller Mode

When the CPM is configured to use two PCIe controllers, High Speed Debug Port (HSDP) cannot be used because it shares GTs with the two PCIe controllers. Each PCIe controller can support x1, x2, x4 and x8 link widths in this configuration. This configuration will use at least the bottom three GT Quads closest to the CPM. These GT Quads cannot be used by PL resources. If CPM PCIe controller 1 is using a link width of x1, x2, or x4; then CPM uses three GT Quads. In this case the fourth GT Quad can be used by PL resources to implement GT based solutions. If CPM PCIe controller 1 is using a x8 link width, all four GT Quads will be used by the CPM and cannot be used by PL resources.

CPM PCIe controller 0 lane0 is placed at the bottom-most GT of the bottom-most GT Quad that is directly above the CPM. Subsequent lanes use the next available GTs moving vertically up the device as the lane number increments. CPM PCIe controller 0 lane7 connects to the top-most GT in the second GT Quad away from the CPM.

CPM PCIe controller 1 lane0 is placed at the bottom-most GT of third GT Quad above the CPM. Subsequent lanes use the next available GTs moving vertically up the device as the lane number increments. CPM PCIe controller 1 lane7 connects to the top-most GT in the fourth GT Quad away from the CPM.

High Speed Debug Port (HSDP) Only Modes

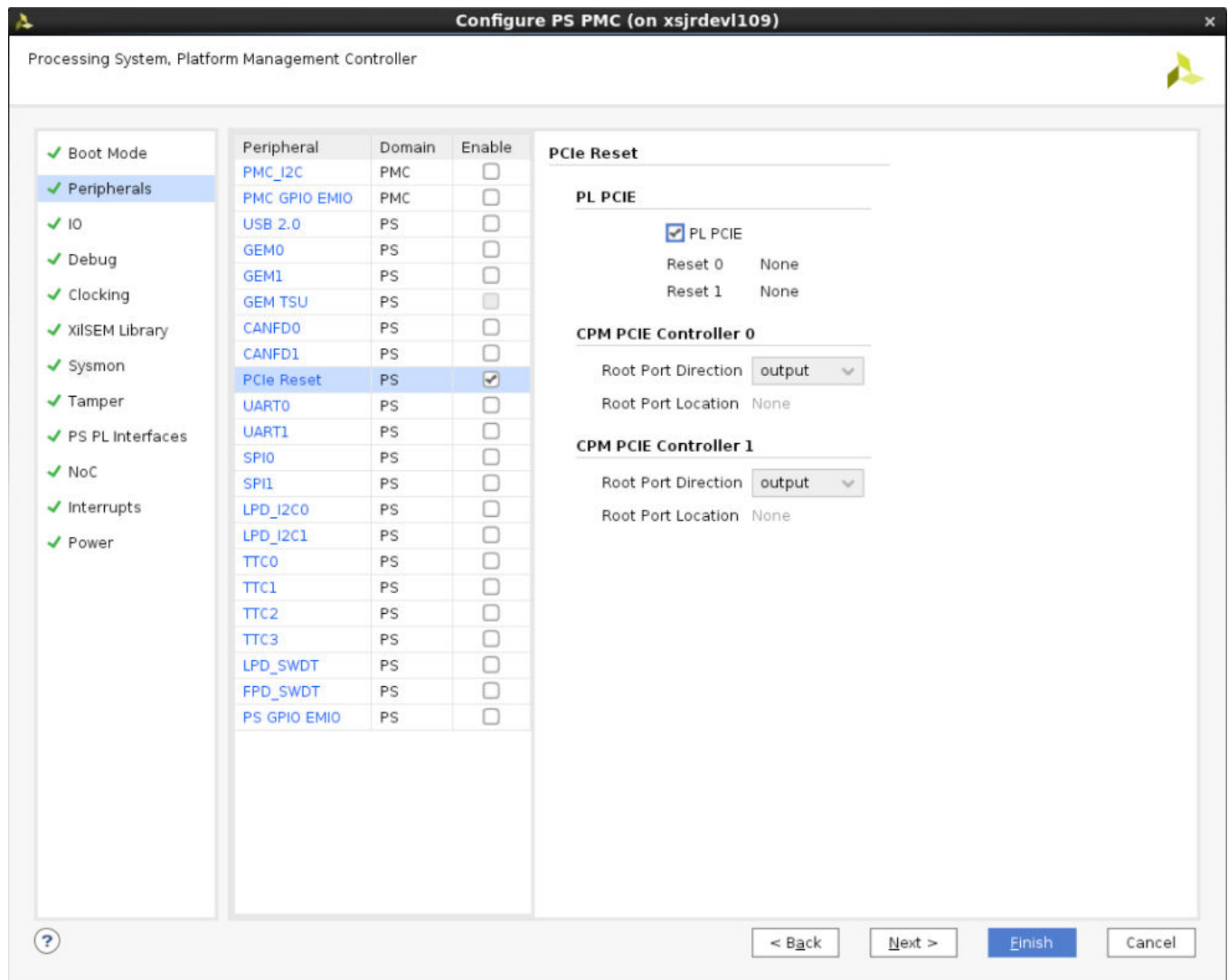
When the CPM is configured to use the High Speed Debug Port (HSDP) without enabling either PCIe controller, the bottom-most GT in the bottom-most GT Quad closest to CPM should be used. This will allow the CPM to use only one GT Quad and allow the next three GT Quads to be used by PL resources.

HSDP can also be enabled for the bottom-most GT in the third GT Quad up from CPM. In this scenario CPM will use three GT Quads and only use one GT. The remaining unused GTs cannot be used or shared by PL resources. As result typically HSDP will not be used in this configuration.

CPM4 Additional Considerations

To facilitate migration from UltraScale™ or UltraScale+™ designs, boards may be designed to use either CPM4 or PL PCIe integrated blocks to implement PCIe solutions. When designing a board to use either CPM4 or the PL PCIe hardblock, the CPM4 pin selection and planning guidelines should be followed because they are more restrictive. By doing this a board can be designed that will work for either a CPM4 or PL PCIe implementation. To route the PCIe reset from the CPM4 to the PL for use with a PL PCIe implementation the following option will need to be enabled under PS-PMC in the CIPS IP customization GUI.

Figure 21: Configuring the PS PMC



When this option is enabled the PCIe reset for each disabled CPM4 PCIe controller will be routed to the PL. The same CPM4 pin selection limitations will apply and the additional PCIe reset output pins will be exposed at the boundary of the CIPS IP. If the CPM4 PCIe controller is enabled, the PCIe reset is used internal to the CPM4 and is not routed to the PL for connectivity to PL PCIe controllers.

GT Locations

Assigning GT Locations

Unlike in UltraScale+™ and previous devices implementations where direct assignment of GTs are not possible in the user constraints, in Versal ACAP implementations the GTs are external to the PCIe® IP and hence the GT assignment can be done in the user constraints. The GT locations are assigned in Quad granularity. (Details of lane 0 placement and lane ordering is found in the GT Selection section. Changing the lane 0 location or lane ordering from the default setting is not supported. Below is an example of assigning GT locations in a user constraint file.

Note: The `gt_quad` instances should be assigned contiguously.

```
set_property LOC GTY_QUAD_X0Y6 [get_cells $gt_quads -filter NAME=~*/
gt_quad_3/*]
set_property LOC GTY_QUAD_X0Y5 [get_cells $gt_quads -filter NAME=~*/
gt_quad_2/*]
set_property LOC GTY_QUAD_X0Y4 [get_cells $gt_quads -filter NAME=~*/
gt_quad_1/*]
set_property LOC GTY_QUAD_X0Y3 [get_cells $gt_quads -filter NAME=~*/
gt_quad_0/*]
```

Related Information

[CPM4 GT Selection](#)

GT Quad Locations

The following table identifies the PCIe lane0 GT Quad(s) that can be used for each PCIe controller location. The Quad shown in bold is the most adjacent or suggested GT Quad for each PCIe lane0 location.

Table 39: GT Locations

Device	Package	PCIe Blocks	GT QUAD (X16)	GT QUAD (X8)	GT QUAD (X4 and Below)
XCVC1902	VIVA1596	CPM Controller 0	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y3
		CPM Controller 1	N/A	GT_Y_QUAD_X0Y5	GT_Y_QUAD_X0Y5
		X0Y2	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y6 GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3
		X0Y1	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y6 GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3
		X1Y2	GT_Y_QUAD_X1Y5	GT_Y_QUAD_X1Y5 GT_Y_QUAD_X1Y4 GT_Y_QUAD_X1Y3	GT_Y_QUAD_X1Y5 GT_Y_QUAD_X1Y4 GT_Y_QUAD_X1Y3 GT_Y_QUAD_X1Y2
		X1Y0	GT_Y_QUAD_X1Y5	GT_Y_QUAD_X1Y5 GT_Y_QUAD_X1Y4 GT_Y_QUAD_X1Y3	GT_Y_QUAD_X1Y5 GT_Y_QUAD_X1Y4 GT_Y_QUAD_X1Y3 GT_Y_QUAD_X1Y2
XCVC1902	VSVA2197	CPM Controller 0	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y3
		CPM Controller 1	N/A	GT_Y_QUAD_X0Y5	GT_Y_QUAD_X0Y5
		X0Y2	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y6 GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3
		X0Y1	GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3	GT_Y_QUAD_X0Y6 GT_Y_QUAD_X0Y5 GT_Y_QUAD_X0Y4 GT_Y_QUAD_X0Y3
		X1Y2	GT_Y_QUAD_X1Y6	GT_Y_QUAD_X1Y6 GT_Y_QUAD_X1Y5 GT_Y_QUAD_X1Y4	GT_Y_QUAD_X1Y6 GT_Y_QUAD_X1Y5 GT_Y_QUAD_X1Y4 GT_Y_QUAD_X1Y3
		X1Y0	GT_Y_QUAD_X1Y3	GT_Y_QUAD_X1Y3 GT_Y_QUAD_X1Y2 GT_Y_QUAD_X1Y1	GT_Y_QUAD_X1Y3 GT_Y_QUAD_X1Y2 GT_Y_QUAD_X1Y1 GT_Y_QUAD_X1Y0

Table 39: GT Locations (cont'd)

Device	Package	PCIe Blocks	GT QUAD (X16)	GT QUAD (X8)	GT QUAD (X4 and Below)
XCVC1902	VSVD1760	CPM Controller 0	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3
		CPM Controller 1	N/A	GTY_QUAD_X0Y5	GTY_QUAD_X0Y5
		X0Y2	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X0Y1	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X1Y2	N/A	GTY_QUAD_X1Y4	GTY_QUAD_X1Y4 GTY_QUAD_X1Y3
		X1Y0	N/A	GTY_QUAD_X1Y4	GTY_QUAD_X1Y4 GTY_QUAD_X1Y3
XCVM1802	VFVC1760	CPM Controller 0	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3
		CPM Controller 1	N/A	GTY_QUAD_X0Y5	GTY_QUAD_X0Y5
		X0Y2	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X0Y1	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X1Y2	GTY_QUAD_X1Y6	GTY_QUAD_X1Y6 GTY_QUAD_X1Y5 GTY_QUAD_X1Y4	GTY_QUAD_X1Y6 GTY_QUAD_X1Y5 GTY_QUAD_X1Y4 GTY_QUAD_X1Y3
		X1Y0	GTY_QUAD_X1Y3	GTY_QUAD_X1Y3 GTY_QUAD_X1Y2 GTY_QUAD_X1Y1	GTY_QUAD_X1Y3 GTY_QUAD_X1Y2 GTY_QUAD_X1Y1 GTY_QUAD_X1Y0

Table 39: GT Locations (cont'd)

Device	Package	PCIe Blocks	GT QUAD (X16)	GT QUAD (X8)	GT QUAD (X4 and Below)
XCVM1802	VSVA2197	CPM Controller 0	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3
		CPM Controller 1	N/A	GTY_QUAD_X0Y5	GTY_QUAD_X0Y5
		X0Y2	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X0Y1	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X1Y2	GTY_QUAD_X1Y6	GTY_QUAD_X1Y6 GTY_QUAD_X1Y5 GTY_QUAD_X1Y4	GTY_QUAD_X1Y6 GTY_QUAD_X1Y5 GTY_QUAD_X1Y4 GTY_QUAD_X1Y3
		X1Y0	GTY_QUAD_X1Y3	GTY_QUAD_X1Y3 GTY_QUAD_X1Y2 GTY_QUAD_X1Y1	GTY_QUAD_X1Y3 GTY_QUAD_X1Y2 GTY_QUAD_X1Y1 GTY_QUAD_X1Y0
XCVM1802	VSVD1760	CPM Controller 0	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3	GTY_QUAD_X0Y3
		CPM Controller 1	N/A	GTY_QUAD_X0Y5	GTY_QUAD_X0Y5
		X0Y2	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X0Y1	GTY_QUAD_X0Y3	GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3	GTY_QUAD_X0Y6 GTY_QUAD_X0Y5 GTY_QUAD_X0Y4 GTY_QUAD_X0Y3
		X1Y2	N/A	GTY_QUAD_X1Y4	GTY_QUAD_X1Y4 GTY_QUAD_X1Y3
		X1Y0	N/A	GTY_QUAD_X1Y4	GTY_QUAD_X1Y4 GTY_QUAD_X1Y3

Debugging

This appendix includes details about resources available on the Xilinx[®] Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Versal[®] ACAP CPM Mode for PCIe, the [Xilinx Support web page](#) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The [Xilinx Community Forums](#) are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the Versal[®] ACAP CPM Mode for PCIe. This guide, along with documentation related to all products that aid in the design process, can be found on the [Xilinx Support web page](#) or by using the Xilinx[®] Documentation Navigator. Download the Xilinx Documentation Navigator from the [Downloads page](#). For more information about this tool and the features available, open the online help after installation.

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

The Solution Center specific to the Versal[®] ACAP CPM Mode for PCIe is listed below.

- [Xilinx Solution Center for PCI Express](#)

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this Versal® ACAP CPM Mode for PCIe can be located by using the Search Support box on the main [Xilinx support web page](#). To maximize your search results, use keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Core

AR [73083](#).

Technical Support

Xilinx provides technical support on the [Xilinx Community Forums](#) for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the [Xilinx Community Forums](#).

PCIe Link Debug Enablement

The Versal® ACAP CPM Mode for PCI Express customization provides an option to enable PCIe® Link Debug. Enabling this option will insert a debug core inside the IP core that will be recognized by the Vivado® Hardware Manager and provide PCIe specific debug information and view. The debug view provides information relating to the current link speed, current link width, and LTSSM state transitions, which can facilitate debug of PCIe link related issues.

Note: This appendix provides guidance for both CPM and PL PCIe based solutions. For this core, the CPM related guidance is of primary importance, while the PL PCIe related guidance might be relevant and is provided for informational purposes.

Enabling PCIe Link Debug

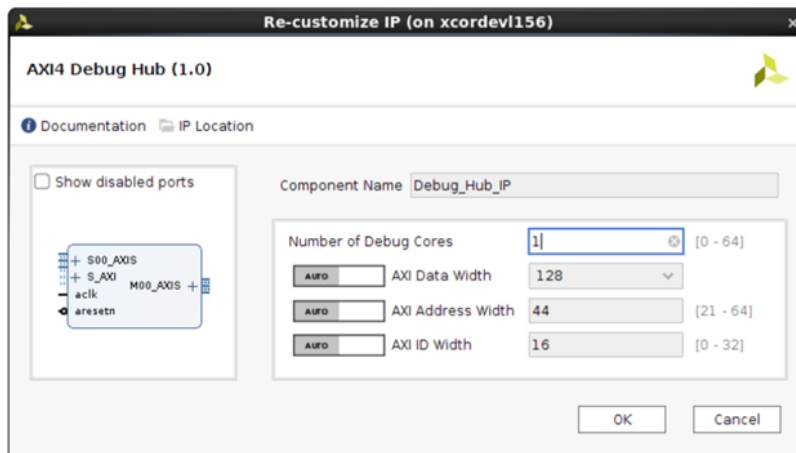
Use this guide to enable and connect PCIe Link Debug in a Vivado IP integrator design. This section only describes the additional connections that should be added to enable PCIe Link Debug in a design. It does not discuss how to properly connect the PCIe enabled IPs to create a working design. Block automation can be used, or the connectivity and connections described below should be added to an existing design and IP configuration.

1. Enable this option in the core customization wizard, and select the options in the customization GUI, as shown below. The CPM PCIe cores are customized through the CIPS IP and for PL PCIe cores are customized through the Versal ACAP Integrated Block for PCIe IP.

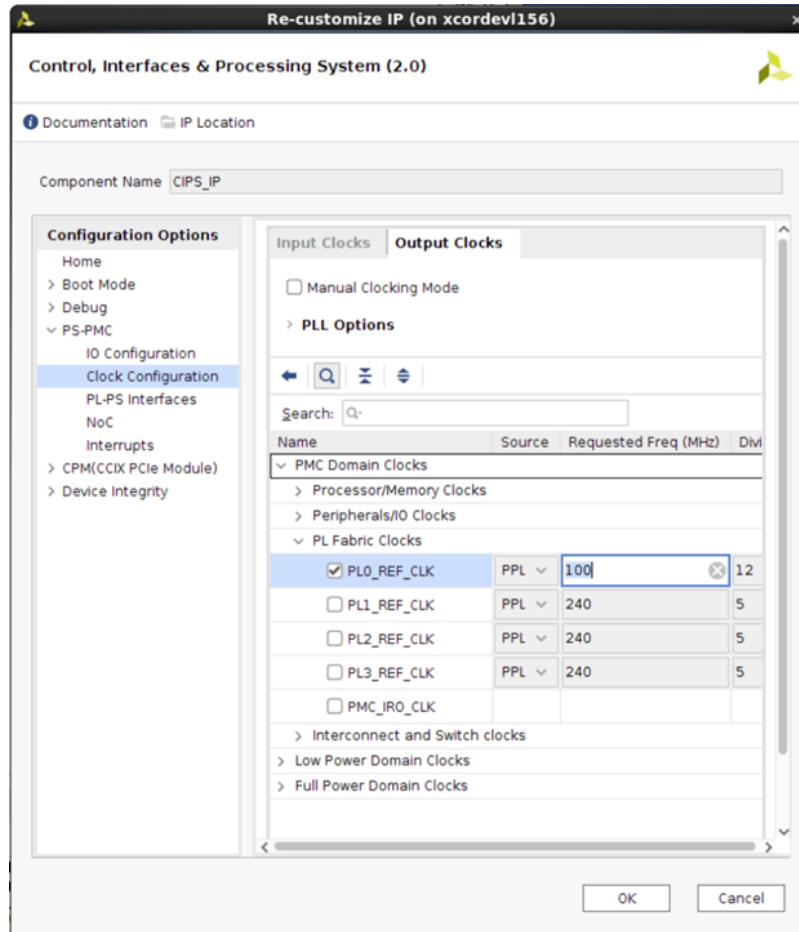
- PCIe-Link Debug
- Enable Debug AXI4 Stream Interfaces

This adds the PCIe debug core to the PCIe IP and exposes the debug AXI4-Stream interfaces and ports. The debug AXI4-Stream and interface ports should be connected to a Debug Hub IP within the Versal design to enable debug for the design. The PCIe example design provides one implementation of how the Debug Hub IP can be connected in Versal designs. This is also detailed in the description below.

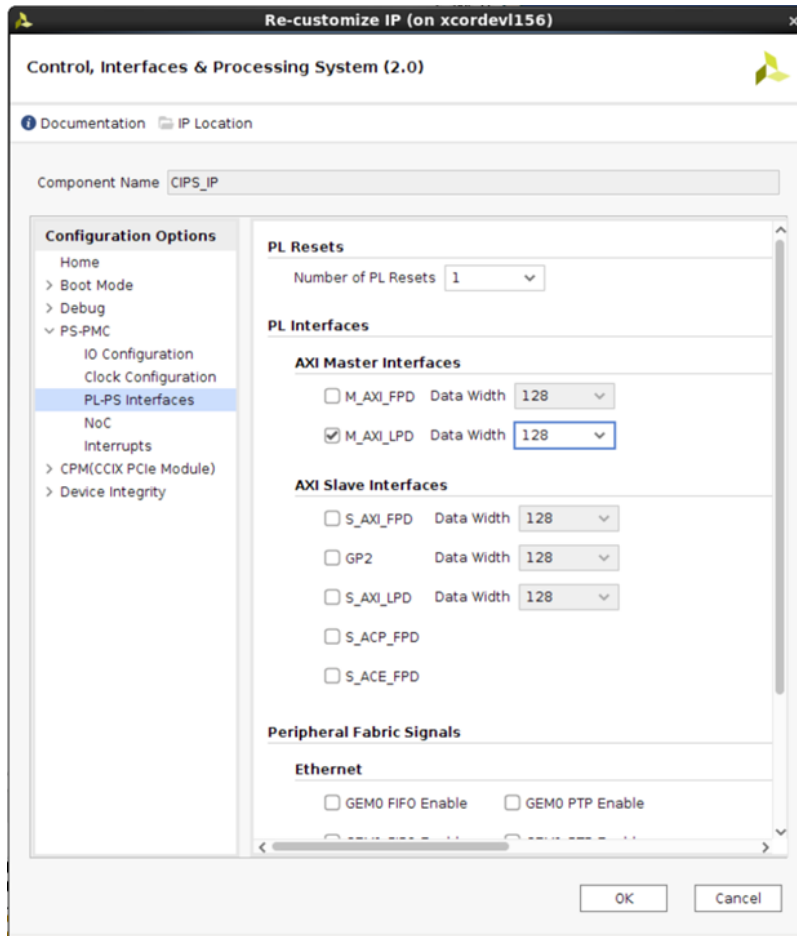
2. Add the Debug Hub IP to the design and use the following configuration options to enable the Debug Hub AXI Memory Mapped interface along with one set of AXI4-Stream interfaces. Additional AXI4-Stream interfaces can be enabled and connected in your design as desired.



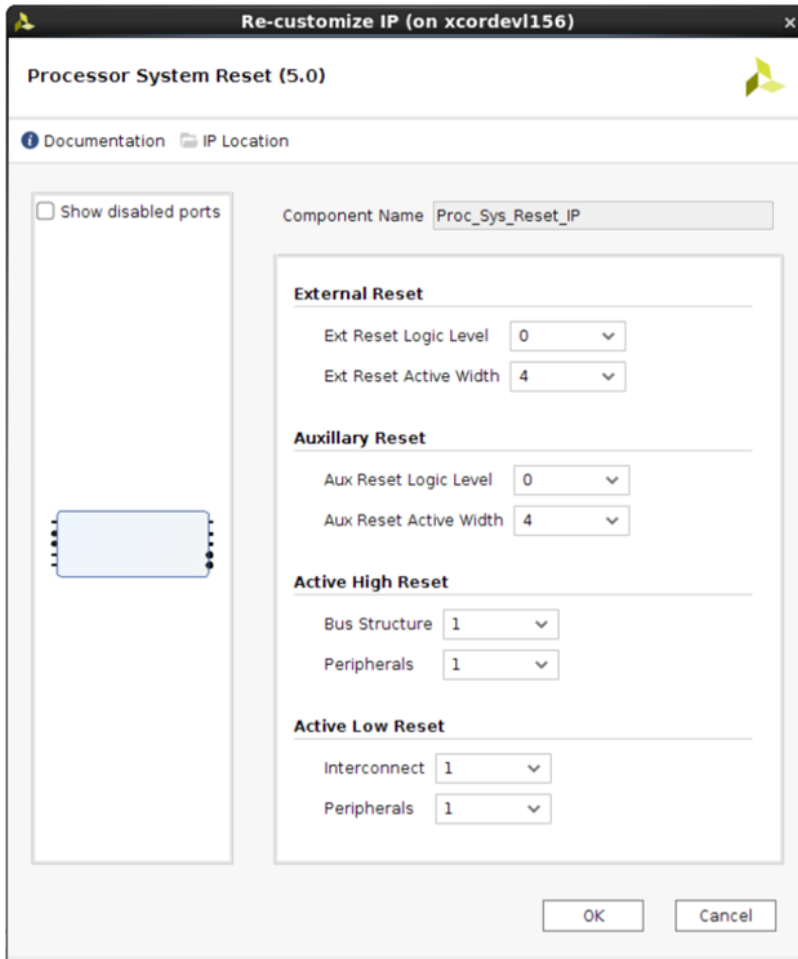
3. Add the CIPS IP to the design or configure the existing CIPS IP and include the following configuration options. These options will enable an AXI Master, clock, and reset that can be connected to the Debug Hub IP. To do so:
 - a. Select **PS-PMC** → **Clock Configuration** → **Output Clocks** → **PMC Domain Clocks** → **PL Fabric Clocks** selection enable a 100 MHz or similar output clock.



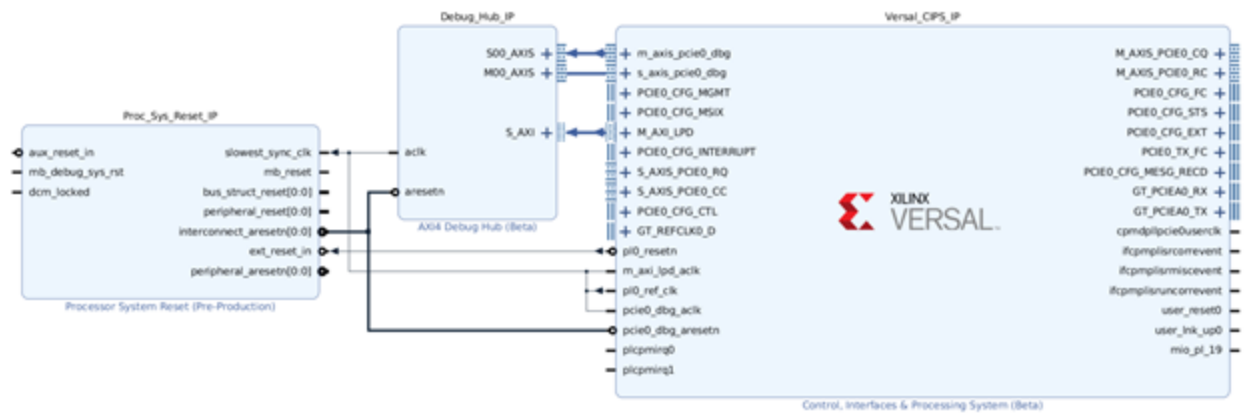
- b. Select **PS-PMC** → **PL-PS Interfaces**, and enable at least one PL reset in **Number of PL Resets**, and the M_AXI_LPD AXI master.



4. Add and configure the Processor System Reset IP.



5. Connect the IPs as shown in the following figures. This may need to be customized to fit with existing design connectivity.



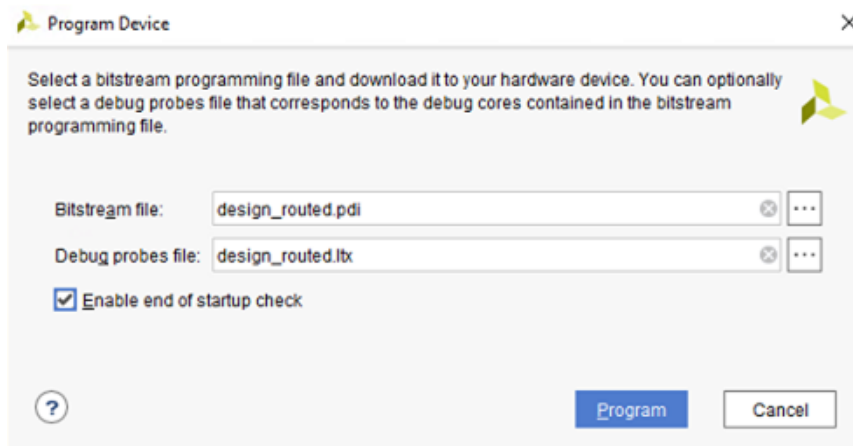
After the debug connections have been added to an Vivado IP integrator design, as shown above, PCIe Link debug is enabled in the generated `.pdi` image. The connections shown above should be added to a full design and are not sufficient to create a working design alone. The PCIe IP ports and the remainder of the design must be created and configured as per the desired operation of the PCIe-enabled IP.

Connecting to PCIe Link Debug in Vivado

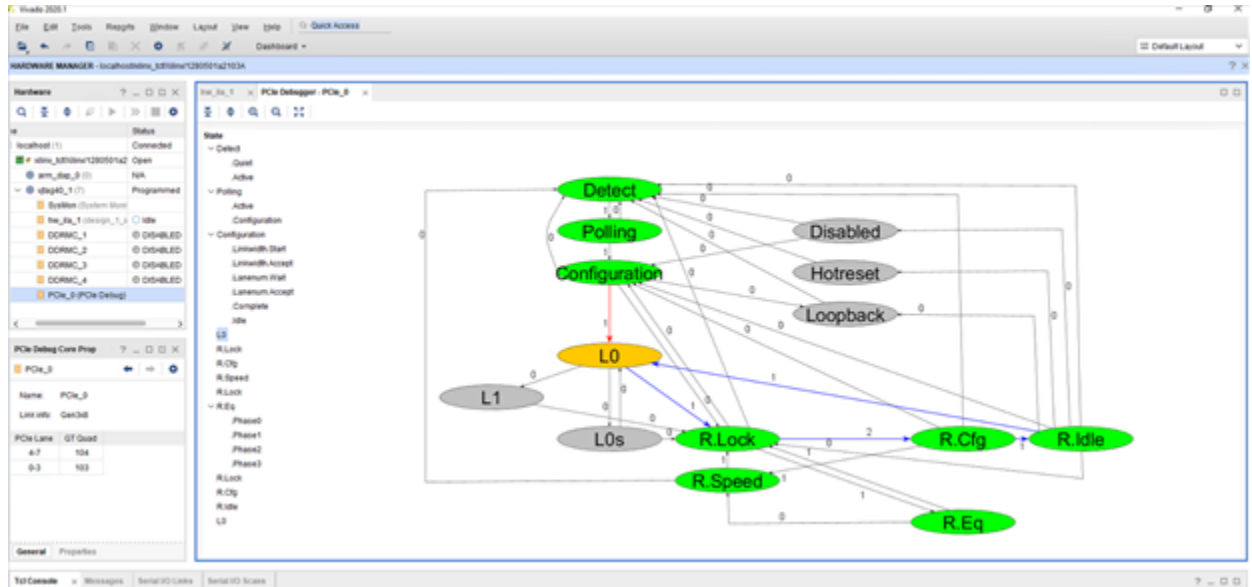
Use the following steps to connect Vivado Hardware Manager to the FPGA device and associated PCIe Link Debug enabled design.

1. Open the Hardware Manager.
2. Select the device from the **Tools** → **Program Device...** drop-down menu.
3. Select the `.pdi` and `.ltx` files for programming the device, and select **Program**.

Note: You should not load the `.ltx` file and refresh the target until after the `.pdi` file has been programmed.



4. Select the PCIe Debug core in the Hardware window. You will see three main views that include the PCIe Debug Core Properties, PCIe Link LTSSM State Trace, and the PCIe Link LTSSM State Diagram with transitions.



Using this view, you can observe the active PCIe link status and state transitions. In the PCIe Debug Core Properties window, you can see the name of the PCIe debug core (PCIe_0), the current link status (Gen3x8), and the connected GTs (Quads 103 and 104). The PCIe LTSSM State Trace view shows a hierarchical view of the PCIe LTSSM state machine transitions. The PCIe LTSSM State Diagram provides a graphical display of the PCIe LTSSM states transitions that were traversed during the PCIe link up process. Visited LTSSM states are shown in green, the final or current LTSSM state is shown in yellow and the number of times each transition was traversed is identified on the arcs between states.

In addition to the graphical display, the `report_hw_pcie` command can be used to generate a console text report that contains the PCIe debug information. This information can be shared with others to aid in debugging PCIe Link issues. For this example, the name of the debug core is PCIe_0, and is inserted into the command.

```
report_hw_pcie PCIe_0
```

This information helps determine where in the PCIe link-up process an issue occurred and can guide further debug of link related issues.

Migrating

This section provides information for migrating from the UltraScale+™ device PCIe core to the core.

Ports

New Ports

All of the ports in the CPM4 PCIE have `pcie0` or `pcie1` (as per the core selected) as prefix to the port names used in UltraScale+™. For example, for the `s_axis_rq_tuser` port in UltraScale+ is `pcie0_s_axis_rq_tuser` in Versal® CPM4.

The following table lists the new ports in the Versal CPM4 Controllers relative to the UltraScale+ device integrated block for PCIe IP.

Note: The following tables mentions only `pcie0*` ports, but apply to both `pcie0*` and `pcie1*` ports.

Table 40: New Ports for Versal CPM4

Name	I/O	Notes
<code>pcie0_cfg_10b_tag_requester_enable[3:0]</code>	O	Per function state of Device Control2 Register 10-Bit Tag Requester Enable bit.
<code>pcie0_cfg_atomic_requester_enable[3:0]</code>	O	Per function state of Device Control2 Register AtomicOp Requester Enable bit.
<code>pcie0_cfg_ats_control_enable[3:0]</code>	O	Per function State of ATS Control Register Enable bit.
<code>pcie0_cfg_ext_tag_enable</code>	O	State of Device Control Register Ext Tag (8-Bit) Enable bit.
<code>pcie0_cfg_fc_vc_sel</code>	I	Selects the Virtual Channel for the type of flow control information presented on the <code>cfg_fc_*</code> signals.
<code>pcie0_cfg_vc1_enable</code>	O	VC1 Resource Control Register: VC Enable bit.
<code>pcie0_cfg_vc1_negotiation_pending</code>	O	VC1 Resource Status Register: VC Negotiation Pending bit.
<code>pcie0_cfg_pasid_enable[3:0]</code>	O	Per Function PASID Enable.
<code>pcie0_cfg_pasid_exec_permission_enable[3:0]</code>	O	Per Function PASID Exec Permission Enable.
<code>pcie0_cfg_pasid_privil_mode_enable[3:0]</code>	O	Per Function PASID Privil Mode Enable.
<code>pcie0_cfg_fc_ph_scale[1:0]</code>	O	Please refer to port list for details.
<code>pcie0_cfg_fc_pd_scale[1:0]</code>	O	Please refer to port list for details.

Table 40: New Ports for Versal CPM4 (cont'd)

Name	I/O	Notes
pcie0_cfg_fc_nph_scale[1:0]	O	Please refer to port list for details.
pcie0_cfg_fc_npd_scale[1:0]	O	Please refer to port list for details.
pcie0_cfg_fc_cpdl_scale[1:0]	O	Please refer to port list for details.
pcie0_cfg_fc_cplh_scale[1:0]	O	Please refer to port list for details.

Port Changes

The following table lists the ports for which the widths were changed in the Versal CPM4 Controllers relative to the UltraScale+ device integrated block for PCIe IP.

Table 41: Port Width Changes For Current Ports

Name	I/O
pcie0_s_axis_rq_tuser[182:0]	I
pcie0_s_axis_cq_tuser[228:0]	O
pcie0_rq_tag0[9:0]	O
pcie0_rq_tag1[9:0]	O

Ports Not Available

The following table lists the ports which were deprecated in the Versal CPM4 Controllers relative to the UltraScale+ device integrated block for PCIe IP.

Table 42: Deprecated Ports

Name	I/O	Notes
cfg_pm_aspm_l1_entry_reject	I	Now part of the register table.
cfg_pm_aspm_tx_l0s_entry_disable	I	Now part of the register table.
cfg_config_space_enable	I	Now part of the register table.
cfg_dsn	I	Now part of the register table.
cfg_dev_id_pf0	I	Now part of the register table.
cfg_dev_id_pf1	I	Now part of the register table.
cfg_dev_id_pf2	I	Now part of the register table.
cfg_dev_id_pf3	I	Now part of the register table.
cfg_vend_id	I	Now part of the register table.
cfg_rev_id_pf0	I	Now part of the register table.
cfg_rev_id_pf1	I	Now part of the register table.
cfg_rev_id_pf2	I	Now part of the register table.
cfg_rev_id_pf3	I	Now part of the register table.
cfg_subsys_id_pf0	I	Now part of the register table.
cfg_subsys_id_pf1	I	Now part of the register table.

Table 42: Deprecated Ports (cont'd)

Name	I/O	Notes
cfg_subsys_id_pf2	I	Now part of the register table.
cfg_subsys_id_pf3	I	Now part of the register table.
cfg_subsys_vend_id	I	Now part of the register table.
cfg_ds_port_number	I	Now part of the register table.
cfg_ds_bus_number	I	Now part of the register table.
cfg_ds_device_number	I	Now part of the register table.
cfg_ds_function_number	I	Now part of the register table.
cfg_link_training_enable	I	Now part of the register table.
cfg_req_pm_transition_l23_ready	I	Now part of the register table.
cfg_bus_number	I	Now part of the register table.
cfg_dpa_substate_change	O	Not available
cfg_obff_enable	O	Not available
phy_rdy_out	O	Not available
sys_reset	I	Reset now routed via PS.

Related Information

[Resets](#)

GT Locations

CPM has a fixed connectivity to GTs based on the CPM configuration. You cannot change the GT locations in Versal ACAP as was possible in UltraScale+ devices.

Related Information

[GT Selection and Pin Planning for CPM4](#)

Clocking

CPM4 requires the same reference clock as UltraScale+ devices on the input side. On the output side, only `pcie(n)_user_clk` is available for the fabric (user logic). The `pcie(n)_user_clk` signal can have a frequency of 62.5, 125, or 250 MHz depending on the configured link speed and width.

Related Information

[Clocking](#)

Reset

Fundamental reset for the PCIe controller is driven by the I/O inside the PS which should be configured in CIPS.

Related Information

[Resets](#)

Features

New Features

PASID

PASID Extended Capability structure has been added which can be enabled via GUI with which the core supports sending and receiving TLPs containing a PASID TLP Prefix.

10-Bit Tag

The CPM4 PCIe controller supports 10-bit Tag feature, it inherently supports 10-bit tag on the completer interface while requestor interface can be enabled via GUI option. When enabled management of up to 768 tags is possible compared to max of 256 tags in UltraScale+ devices.

Feature DLLP

Data Link Feature Extended Capability structure has been added for link speed of 16.0 GT/s. It contains programmable control/status information about the local and peer support of the “Data Link Feature Support”.

Lane Margining

Lane Margining at the Receiver Extended Capability structure has been added for link speed of 16.0 GT/s.

Physical Layer 16.0 GT/s Extended Capability

Physical Layer 16.0 GT/s Extended Capability structure has been added for link speed of 16.0 GT/s with which Gen4 equalization status can be read.

Retimers Supported

Link Extension devices (retimers) is supported to interoperate with CPM4 PCIe block for link speed of 16.0 GT/s.

Flow Control Informational Select

All combinations of `cfg_fc_sel` values are supported relative to UltraScale+, refer port description for more details.

MSIX – Additional Vectors

When configured as MSIX-internal can support up to 32 vectors per physical function compared to 8 in UltraScale+, vectors per function for VF and total number of vectors (2048) remains the same.

Switch Mode

CPM4 PCIe controller can be configured as Upstream or Downstream Port of a Switch design.

Features Not Available, or Limited Usage Features

- No Example Design - There is no example design support for this release.
- There was a core clock frequency selection available in UltraScale+ for some of the configurations which is not available in Versal CPM4.
- Legacy Interrupts cannot be supported when PASID is enabled as some of the ports are re-purposed.
- Pipe mode is currently not supported for Versal CPM4.
- Descrambler is currently not supported for Versal CPM4.
- Some of the advanced GT Settings present in UltraScale+ devices, such as Auto RX Equalization, form factor driver insertion loss adjustment. TX preset selection is not supported for Versal CPM4.
- TPH capability is not supported.

Attributes

A complete list of Versal attributes is available in the *Versal ACAP Register Reference* ([AM012](#)).

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on DocNav, see the [Documentation Navigator](#) page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *Control, Interface and Processing System LogiCORE IP Product Guide* ([PG352](#))
2. *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide* ([PG347](#))
3. *Versal ACAP Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG343](#))
4. *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#))
5. *Versal ACAP Technical Reference Manual* ([AM011](#))
6. *Versal ACAP Register Reference* ([AM012](#))
7. *Versal ACAP CPM CCIX Architecture Manual* ([AM016](#))
8. *PCI Express Base Specification 4.0* (<https://www.pcisig.com/specifications>)
9. *PCI Express Base Specification 5.0* (<https://www.pcisig.com/specifications>)
10. Cache Coherent Interconnect for Accelerators (CCIX) Transport Specification (available at <http://www.ccixconsortium.com/>; membership required).
11. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))
12. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
13. *Versal ACAP System Software Developers Guide* ([UG1304](#))

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/09/2021 Version 3.0	
General, and Customizing and Generating the CIPS IP Core	Updated for CIPS IP version 3.0.
Minimum Device Requirements	New information added.
Tandem Configuration	New section added, and tandem configuration use case details updated throughout.
Limitations	New information added.
04/29/2021 Version 2.1	
Limitations	Added topic to report known issues in the release.
01/08/2021 Version 2.1	
Features	Updated lane operations.

Section	Revision Summary
PCI Express Endpoint Use Modes	Added the Tandem PROM and Tandem PCIe use modes (not supported in this release).
Port Descriptions	Added new section.
Chapter 4: Design Flow Steps	Updated with new parameters throughout.
Appendix C: Migrating	Added new appendix.
07/24/2020 Version 1.0	
Initial Xilinx release.	N/A

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING

OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2020-2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.