

# AXI Bridge for PCI Express Gen3 Subsystem v3.0

## *Product Guide*

Vivado Design Suite

PG194 (v3.0) November 16, 2022

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

<b>Chapter 1: Introduction.....</b>	<b>5</b>
Features.....	5
IP Facts.....	6
<b>Chapter 2: Overview.....</b>	<b>7</b>
Feature Summary.....	8
Unsupported Features.....	9
Limitations.....	9
Licensing and Ordering.....	10
<b>Chapter 3: Product Specification.....</b>	<b>11</b>
Standards.....	12
Performance and Resource Utilization.....	12
Minimum Device Requirements .....	12
AXI Bridge Operations.....	13
Port Descriptions.....	38
Bridge Parameters.....	46
Memory Map.....	55
<b>Chapter 4: Designing with the Core.....</b>	<b>82</b>
General Design Guidelines.....	82
Shared Logic.....	82
Clocking.....	83
Resets.....	86
AXI Transactions for PCIe.....	88
Transaction Ordering for PCIe.....	89
BAR and Address Translation.....	90
Malformed TLP.....	98
Abnormal Conditions.....	98
Endpoint.....	102
Root Port.....	108
Tandem Configuration.....	113

<b>Chapter 5: Design Flow Steps</b> .....	<b>115</b>
Customizing and Generating the Subsystem.....	115
Constraining the Subsystem.....	136
Simulation.....	138
Synthesis and Implementation.....	140
<b>Chapter 6: Example Design</b> .....	<b>141</b>
Endpoint Configuration of the AXI PCIe Block.....	141
Customizing and Generating the Example Design.....	142
Simulation Design Overview.....	143
Implementation Design Overview.....	144
Example Design Elements.....	144
Example Design Output Structure.....	144
<b>Chapter 7: Test Bench</b> .....	<b>146</b>
Root Port Model Test Bench for Endpoint.....	146
Endpoint Model Test Bench for Root Port.....	148
Example Use Mode.....	149
<b>Appendix A: Upgrading</b> .....	<b>151</b>
Migrating from AXI PCIe Gen2 to AXI Bridge for PCIe Gen3.....	151
Migrating from AXI Bridge for PCIe Gen3 to DMA/Bridge Subsystem for PCIe in AXI Bridge Mode.....	152
Upgrading in the Vivado Design Suite.....	153
<b>Appendix B: Debugging</b> .....	<b>156</b>
Finding Help on Xilinx.com.....	156
Debug Tools.....	157
Hardware Debug.....	159
Additional Debug Information.....	163
Interface Debug.....	163
<b>Appendix C: Using the Xilinx Virtual Cable to Debug</b> .....	<b>165</b>
Overview.....	165
Host PC XVC-Server Application.....	166
Host PC XVC-over-PCIe Driver.....	166
XVC-over-PCIe Enabled FPGA Design.....	167
Using the PCIe-XVC-VSEC Example Design.....	173

**Appendix D: CCIX Interface..... 182**

- Supported Configurations..... 182
- Port Descriptions..... 183
- Example Figures..... 194
- CCIX-VC1 Credits..... 195
- CCIX Activation/Deactivation..... 195

**Appendix E: Additional Resources and Legal Notices..... 197**

- Xilinx Resources..... 197
- Documentation Navigator and Design Hubs..... 197
- References..... 197
- Revision History..... 198
- Please Read: Important Legal Notices..... 202

# Introduction

The Xilinx® AXI Bridge for PCI Express Gen3 Subsystem is available for UltraScale™ and Virtex®-7 XT devices. The Xilinx DMA/Bridge Subsystem for PCI Express® in AXI Bridge mode is available for UltraScale+™ devices. These cores bridge AXI4 and PCI Express interfaces.

---

## Features

- AXI Bridge for PCI Express Gen3 supports UltraScale™ architecture and Virtex®-7 XT FPGA Gen3 Integrated Blocks for PCI Express®
- DMA/Bridge Subsystem for PCI Express core in AXI Bridge mode supports UltraScale+ Integrated Blocks for PCI Express
- AXI Bridge for PCI Express Gen3 supports Maximum Payload Size (MPS) up to 512 bytes
- DMA/Bridge Subsystem for PCI Express core in AXI Bridge mode supports Maximum Payload Size (MPS) up to 1024 bytes
- Multiple Vector Messaged Signaled Interrupts (MSIs)
- MSI-X interrupt support
- Legacy interrupt support
- Memory-mapped AXI4 access to PCIe® space
- PCIe access to memory-mapped AXI4 space
- Tracks and manages Transaction Layer Packets (TLPs) completion processing
- Detects and indicates error conditions with interrupts
- Optimal AXI4 pipeline support for enhanced performance
- Compliant with Advanced RISC Machine Arm® Advanced Microcontroller Bus Architecture 4 (AMBA®) AXI4 specification
- Supports up to six PCIe 32-bit or three 64-bit PCIe Base Address Registers (BARs) as Endpoint
- Supports up to two PCIe 32-bit or a single PCIe 64-bit BAR as Root Port

# IP Facts

<b>LogiCORE™ IP Facts Table</b>	
<b>Core Specifics</b>	
Supported Device Family	<b>AXI Bridge for PCIe Gen3:</b> UltraScale, Virtex-7 XT <sup>1</sup> <b>DMA/Bridge Subsystem for PCIe in AXI Bridge mode:</b> UltraScale+
Supported User Interfaces	AXI4
Resources	<a href="#">Performance and Resource Utilization web page</a>
<b>Provided with Core</b>	
Design Files	Verilog
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver	Root Port Driver
<b>Tested Design Flows<sup>2</sup></b>	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the <a href="#">Xilinx Design Tools: Release Notes Guide</a>
Synthesis	Vivado synthesis
<b>Support</b>	
Release Notes and Known Issues	Master Answer Record: <a href="#">61898</a>
All Vivado IP Change Logs	Master Vivado IP Change Logs: <a href="#">72775</a>
<a href="#">Xilinx Support web page</a>	

**Notes:**

1. Except for XC7VX485T, XC7V585T, and XC7V2000T, all Virtex-7 devices are supported.
2. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).

# Overview

Xilinx® provides the following two cores for PCI Express® AXI Bridge:

- AXI Bridge for PCI Express Gen3
- DMA/Bridge Subsystem for PCI Express in AXI Bridge mode

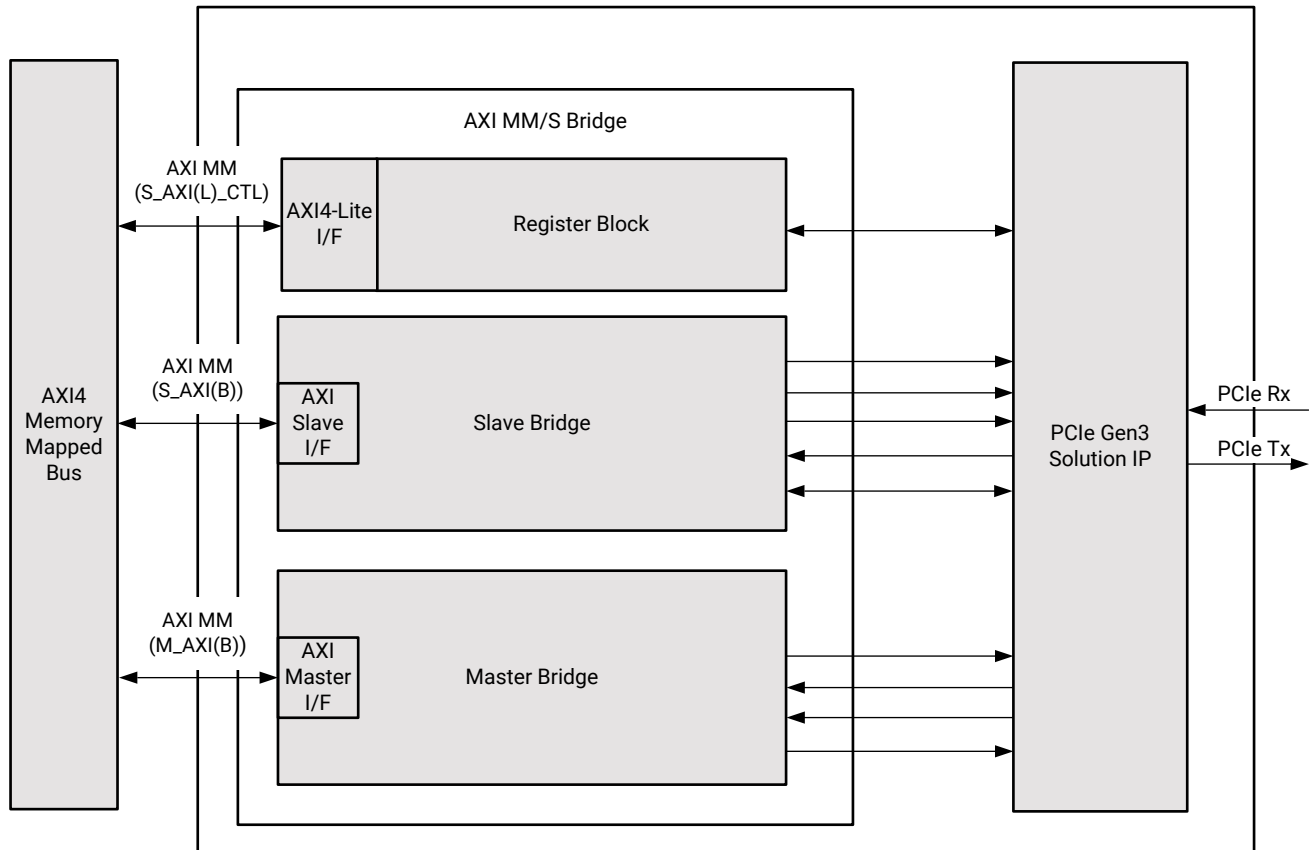
The AXI Bridge for PCI Express Gen3 core supports UltraScale+™ and Virtex®-7 XT devices. The DMA/Bridge Subsystem for PCI Express in AXI Bridge mode core, when configured in Bridge mode, supports UltraScale+ devices.

This document is applicable for both the AXI Bridge for PCI Express Gen3 core, and the DMA/Bridge Subsystem for PCI Express core in AXI Bridge functional mode. See the *DMA/Bridge Subsystem for PCI Express Product Guide* (PG195) for the DMA/Bridge Subsystem for PCI Express core in DMA functional mode. When something applies to both cores together, this document refers to the core as the Bridge core. Otherwise, the specific core name is used.

The AXI Bridge for PCI Express Gen3 is designed for the Vivado® IP integrator in the Vivado® Design Suite. The AXI Bridge for PCI Express Gen3 provides an interface between an AXI4 user logic interface and PCI Express® using the Integrated Block for PCI Express. The Bridge core provides the translation level between the AXI4 embedded system to the PCI Express system. The Bridge core translates the AXI4 memory read or writes to PCI™ Transaction Layer Packets (TLP) packets and translates PCIe memory read and write request TLP packets to AXI4 interface commands.

The architecture of the Bridge is shown in the following figure.

Figure 1: High-Level Bridge Architecture for the PCI Express Gen3 Architecture



X22423-030419

## Feature Summary

The Bridge core is an interface between the AXI4 bus and PCI Express®. The core contains the memory mapped AXI4 to AXI4-Stream Bridge and the AXI4-Stream Enhanced Interface Block for PCIe. The memory-mapped AXI4 to AXI4-Stream Bridge contains a register block and two functional half bridges, referred to as the Slave Bridge and Master Bridge. The slave bridge connects to the AXI4 Interconnect as a slave device to handle any issued AXI4 master read or write requests. The master bridge connects to the AXI4 interconnect as a master to process the PCIe generated read or write TLPs. The core uses a set of interrupts to detect and flag error conditions.

The Bridge core supports both Root Port and Endpoint configurations.

- When configured as an Endpoint, the Bridge core supports up to six 32-bit or three 64-bit PCIe Base Address Registers (BARs).



- When configured as a Root Port, the core supports up to two 32-bit or a single 64-bit PCIe BAR.

The Bridge core is compliant with the *PCI-SIG Specifications* (<https://www.pcisig.com/specifications>) and with the *AMBA AXI and ACE Protocol Specification* (ARM IH10022E).

## Unsupported Features

The following features are not supported in the Bridge core.

- Tandem Configuration solutions, which include Tandem PROM, Tandem PCIe, Tandem with Field Updates, and DFX over PCIe, are not supported for Virtex®-7 XT devices.
- Tandem Configuration is not yet supported for Bridge mode in UltraScale+™ devices.
- In AXI Bridge Root Port mode ASPM is not supported.

## Limitations

1. For this subsystem, the bridge master and bridge slave cannot achieve more than 128 Gb/s.
2. Bridge will be compliant with all MPS and MRRS settings; however, all traffic initiated from the Bridge will be limited to 256 Bytes (max).
3. AXI address width is limited to 48 bits.

## PCIe Transaction Type

The PCIe® transactions generated are those that are compatible with the AXI4 specification. The following table lists the supported PCIe transaction types.

*Table 1: Supported PCIe Transaction Types*

TX	RX
MRd32	MRd32
MRd64	MRd64
MWr32	MWr32
MWr64	MWr64
Msg(INT/Error)	Msg(SSPL,INT,Error)
Cpl	Cpl
CplD	CplD
Cfg Type0/1 (For Root Port)	

## PCIe Capability

Only the following PCIe capabilities are supported because of the AXI4 specification:

- 1 PF
- MSI
- MSI-X
- PM
- Advanced error reporting (AER)

## Others

### AXI Slave

- Only supports the INCR burst type. Other types result in the Slave Illegal Burst (SIB) interrupt.
- No memory type support (AxCACHE)
- No protection type support (AxPROT)
- No lock type support (AxLOCK)
- No non-contiguous byte enable support (WSTRB)

### AXI Master

- Only issues the INCR burst type
- Only issues the data, non-secure, and unprivileged protection type

---

## Licensing and Ordering

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

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

# Product Specification

The Register block contains registers used in the Bridge core for dynamically mapping the AXI4 memory mapped (MM) address range provided using the AXIBAR parameters to an address for PCIe® range.

The slave bridge provides termination of memory-mapped AXI4 transactions from an AXI master device (such as a processor). The slave bridge provides a way to translate addresses that are mapped within the AXI4 memory mapped address domain to the domain addresses for PCIe. Write transactions to the Slave Bridge are converted into one or more `MemWr` TLPs, depending on the configured Max Payload Size setting, which are passed to the integrated block for PCI Express. The slave bridge in AXI Bridge for PCI Express Gen3 core can support up to eight active AXI4 Write requests. The slave bridge in the DMA/Bridge Subsystem for PCIe in AXI Bridge mode core can support up to 32 active AXI4 Write requests. When a remote AXI master initiates a read transaction to the slave bridge, the read address and qualifiers are captured and a `MemRd` request TLP is passed to the core and a completion timeout timer is started. Completions received through the core are correlated with pending read requests and read data is returned to the AXI master. The slave bridge in AXI Bridge for PCI Express Gen3 core can support up to eight active AXI4 Read requests with pending completions. The slave bridge in the DMA/Bridge Subsystem for PCIe in AXI Bridge mode core can support up to 32 active AXI4 Read requests with pending completions.

The master bridge processes both PCIe `MemWr` and `MemRd` request TLPs received from the Integrated Block for PCI Express and provides a means to translate addresses that are mapped within the address for PCIe domain to the memory mapped AXI4 address domain. Each PCIe `MemWr` request TLP header is used to create an address and qualifiers for the memory mapped AXI4 bus and the associated write data is passed to the addressed memory mapped AXI4 Slave. The Master Bridge in AXI Bridge for PCI Express Gen3 core can support up to eight active PCIe `MemWr` request TLPs. The Master Bridge in the DMA/Bridge Subsystem for PCIe in AXI Bridge mode core can support up to 32 active PCIe `MemWr` request TLPs. PCIe `MemWr` request TLPs support is as follows:

- 4 for 64-bit AXI data width
- 8 for 128-bit AXI data width
- 16 for 256-bit AXI data width
- 32 for 512-bit AXI data width

Each PCIe MemRd request TLP header is used to create an address and qualifiers for the memory-mapped AXI4 bus. Read data is collected from the addressed memory mapped AXI4 slave and used to generate completion TLPs which are then passed to the integrated block for PCI Express. The Master Bridge in AXI Bridge for PCI Express Gen3 core can support up to eight active PCIe MemRd request TLPs with pending completions. The Master Bridge in the DMA/Bridge Subsystem for PCIe core in AXI Bridge mode can support up to 32 active PCIe MemRd request TLPs with pending completions for improved AXI4 pipelining performance.

The instantiated AXI4-Stream Enhanced PCIe block contains submodules including the Requester/Completer interfaces to the AXI bridge and the Register block. The Register block contains the status, control, interrupt registers, and the AXI4-Lite interface.

## Standards

The Bridge core is compliant with the [AMBA AXI Protocol Specification](#) and the [PCI-SIG Specifications](#).

## Performance and Resource Utilization

For full details about performance and resource utilization, visit the [Performance and Resource Utilization web page](#).

## Minimum Device Requirements

The following table lists the link widths and supported speed for a given speed grade.

Table 2: Minimum Device Requirements

Capability Link Speed	Capability Link Widths	Supported Speed Grades
<b>UltraScale+™ Architecture (PCIe4)</b>		
Gen1/Gen2	x1, x2, x4, x8, x16	-1, -1L, -1LV, -2, -2L, -2LV, -3 <sup>1</sup>
Gen3	x1, x2, x4	-1, -1L, -1LV, -2, -2L, -2LV, -3 <sup>1</sup>
	x8, x16	-2, -2L, -3 <sup>1</sup>
<b>Virtex® UltraScale+™ Devices with HBM (PCIe4C)<sup>2</sup></b>		
Gen1/Gen2	x1, x2, x4, x8, x16	-1, -2, -2L, -2LV, -3

Table 2: Minimum Device Requirements (cont'd)

Capability Link Speed	Capability Link Widths	Supported Speed Grades
Gen3	x1, x2, x4	-1, -2, -2L, -2LV, -3
	x8	-1, -2, -2L, -2LV, -3
	x16	-1, -2, -2L, -2LV, -3
Gen4 <sup>5</sup>	x1, x2, x4, x8	-2, -2L, -3
<b>UltraScale Family</b>		
Gen1	x1, x2, x4, x8	-1, -2, -3, -1L, -1LV, -1H and -1HV <sup>3</sup>
Gen2	x1, x2, x4, x8	-1, -2, -3, -1L, -1LV, -1H and -1HV <sup>3</sup>
Gen3	x1, x2, x4	-1, -2, -3, -1L, -1LV, -1H and -1HV <sup>3, 4</sup>
Gen3	x8	-2, -3
<b>7 Series Gen3 Family</b>		
Gen1	x1, x2, x4, x8	-3, -2, -1, -2L, -2G, -2I, -1M, -1I
Gen2	x1, x2, x4, x8	-3, -2, -1, -2L, -2G, -2I, -1M, -1I
Gen3	x1, x2, x4, x8	-3, -2, -2L, -2G, -2I

**Notes:**

1. -1L(0.95V), -1LV(0.90V), -2L(0.85V), -2LV(0.72V).
2. Virtex® UltraScale+™ devices with high bandwidth memory (HBM) contain both PCIe4 and PCIe4C blocks. Only the PCIe4C blocks support Gen3 x16 in the -2LV speed grade.
3. -1H and -1HV are available only for Virtex UltraScale devices. -1L and -1LV are available only for Kintex UltraScale devices.
4. The Core Clock Frequency option must be set to 250 MHz for -1, -1LV, -1L, -1H and -1HV speed grades. The Core Clock Frequency option set to 500 MHz is supported for -3 and -2 speed grades only.
5. For Gen4 mode restrictions, see the *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)*.

## AXI Bridge Operations

### AXI Transactions for PCIe

The following tables are the translation tables for AXI4-Stream and memory-mapped transactions.

Table 3: AXI4 Memory-Mapped Transactions to AXI4-Stream PCIe TLPs

AXI4 Memory-Mapped Transaction	AXI4-Stream PCIe TLPs
INCR Burst Read of AXIBAR	MemRd 32 (3DW)
INCR Burst Write to AXIBAR	MemWr 32 (3DW)
INCR Burst Read of AXIBAR	MemRd 64 (4DW)
INCR Burst Write to AXIBAR	MemWr 64 (4DW)

**Table 4: AXI4-Stream PCIe TLPs to AXI4 Memory Mapped Transactions**

<b>AXI4-Stream PCIe TLPs</b>	<b>AXI4 Memory-Mapped Transaction</b>
MemRd 32 (3DW) of PCIEBAR	INCR Burst Read
MemWr 32 (3DW) to PCIEBAR	INCR Burst Write
MemRd 64 (4DW) of PCIEBAR	INCR Burst Read
MemWr 64 (4DW) to PCIEBAR	INCR Burst Write

For PCIe® requests with lengths greater than 1 Dword, the size of the data burst on the Master AXI interface will always equal the width of the AXI data bus even when the request received from the PCIe link is shorter than the AXI bus width.

`slave_axi_wstrb` can be used to facilitate data alignment to an address boundary. `slave_axi_wstrb` may equal 0 in the beginning of a valid data cycle and will appropriately calculate an offset to the given address. However, the valid data identified by `slave_axi_wstrb` must be continuous from the first byte enable to the last byte enable.

All transactions initiated at the Slave Bridge interface will be modified and metered by the IP as necessary. The Slave Bridge interface will conform to AXI4 specification and allow burst size up to 4KB, and the IP will split the transaction automatically according to PCIe Max Read Request Size (MRRS), Max Payload Size (MPS), and Read Completion Boundary (RCB). As a result of this operation, one request at the AXI domain may result in multiple request at the PCIe domain, and the IP will adjust the number of issued PCIe request accordingly to avoid oversubscribing the available Completion buffer.

## Transaction Ordering for PCIe

The Bridge core conforms to PCIe® transaction ordering rules. See the [PCI-SIG Specifications](#) for the complete rule set. The following behaviors are implemented in the Bridge core to enforce the PCIe transaction ordering rules on the highly-parallel AXI bus of the bridge.

- The `brresp` to the remote (requesting) AXI4 master device for a write to a remote PCIe device is not issued until the `MemWr` TLP transmission is guaranteed to be sent on the PCIe link before any subsequent TX-transfers.
- If Relaxed Ordering bit is not set within the TLP header, then a remote PCIe device read to a remote AXI slave is not permitted to pass any previous remote PCIe device writes to a remote AXI slave received by the Bridge core. The AXI read address phase is held until the previous AXI write transactions have completed and `brresp` has been received for the AXI write transactions. If the Relaxed Ordering attribute bit is set within the TLP header, then the remote PCIe device read is permitted to pass.
- Read completion data received from a remote PCIe device are not permitted to pass any remote PCIe device writes to a remote AXI slave received by the Bridge core prior to the read completion data. The `brresp` for the AXI write(s) must be received before the completion data is presented on the AXI read data channel.

**Note:** The transaction ordering rules for PCIe might have an impact on data throughput in heavy bidirectional traffic.

## BAR and Address Translation

### ***BAR Addressing***

`Aperture_Base_Address_n` represents Aperture Base Address of nth BAR in GUI

`Aperture_High_Address_n` represents Aperture High Address of nth BAR in GUI

`AXI to PCIe Translation_n` represents AXI to PCIe\_translation of nth BAR in GUI

`Aperture_Base_Address_n` and `Aperture_High_Address_n` are used to calculate the size of the AXI BAR *n* and during address translation to PCIe address.

- `Aperture_Base_Address_n` provides the low address where AXI BAR *n* starts and will be regarded as address offset 0x0 when the address is translated.
- `Aperture_High_Address_n` is the high address of the last valid byte address of AXI BAR *n*. (For more details on how the address gets translated, see Address Translation.)

The difference between `Aperture_Base_Address_n` and `Aperture_High_Address_n` is your AXI BAR *n* size. These values must be set accordingly such that the AXI BAR *n* size is a power of two and must have at least 4K.

When a packet is sent to the core (outgoing PCIe packets), the packet must have an address that is in the range of `Aperture_Base_Address_n` and `Aperture_High_Address_n`. Any packet that is received by the core that has an address outside of this range will be responded to with a SLVERR. When the IP integrator is used, these parameters are derived from the Address Editor tab within the IP integrator. The Address Editor sets the AXI Interconnect as well as the core so the address range matches, and the packet is routed to the core only when the packet has an address within the valid range.

AXI Address width is limited to 48 bits.

### ***Address Translation***

The address space for PCIe® is different than the AXI address space. To access one address space from another address space requires an address translation process. On the AXI side, the bridge supports mapping to PCIe on up to six 32-bit or 64-bit AXI base address registers (BARs).

Four examples follow:

- Example 1 (32-bit PCIe Address Mapping) demonstrates how to set up three AXI BARs and translate the AXI address to a 32-bit address for PCIe.

- Example 2 (64-bit PCIe Address Mapping) demonstrates how to set up three AXI BARs and translate the AXI address to a 64-bit address for PCIe.
- Example 3 demonstrates how to set up two 64-bit PCIe BARs and translate the address for PCIe to an AXI address.
- Example 4 demonstrates how to set up a combination of two 32-bit AXI BARs and two 64 bit AXI BARs, and translate the AXI address to an address for PCIe.

### **Example 1 (32-bit PCIe Address Mapping)**

This example shows the generic settings to set up three independent AXI BARs and address translation of AXI addresses to a remote 32-bit address space for PCIe. This setting of AXI BARs does not depend on the BARs for PCIe in the Bridge core.

In this example, where C\_AXIBAR\_NUM=3, the following assignments for each range are made:

```
AXI_ADDR_WIDTH=48

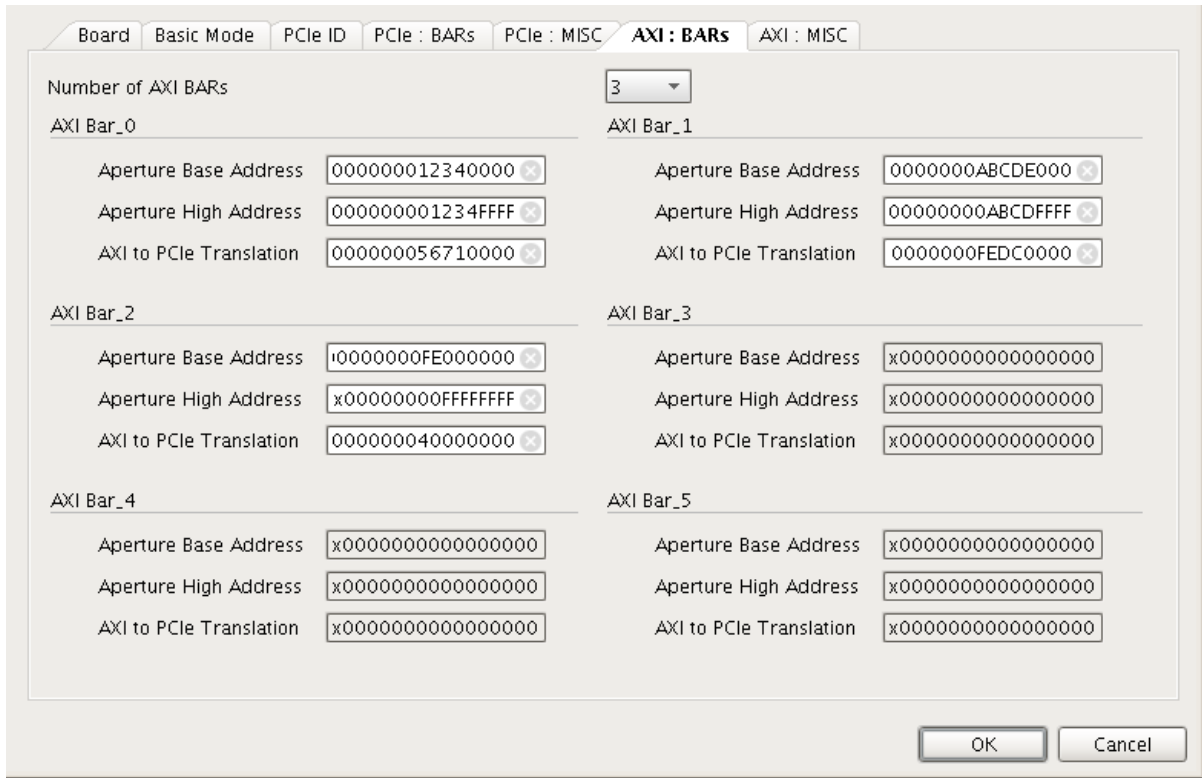
C_AXIBAR_0=0x00000000_12340000
C_AXI_HIGHADDR_0=0x00000000_1234FFFF (64 Kbytes)
C_AXIBAR2PCIEBAR_0=0x00000000_56710000 (Bits 63-32 are zero in order to
produce a
32-bit PCIe TLP. Bits 15-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 16 bits are invalid translation values.)

C_AXIBAR_1=0x00000000_ABCDE000
C_AXI_HIGHADDR_1=0x00000000_ABCDFFFF (8 Kbytes)
C_AXIBAR2PCIEBAR_1=0x00000000_FEDC0000 (Bits 63-32 are zero in order to
produce a
32-bit PCIe TLP. Bits 12-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 13 bits are invalid translation values.)

C_AXIBAR_2=0x00000000_FE000000
C_AXI_HIGHADDR_2=0x00000000_FFFFFFFF (32 Mbytes)
C_AXIBAR2PCIEBAR_2=0x00000000_40000000 (Bits 63-32 are zero in order to
produce a
32-bit PCIe TLP. Bits 24-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 25 bits are invalid translation values.)
```

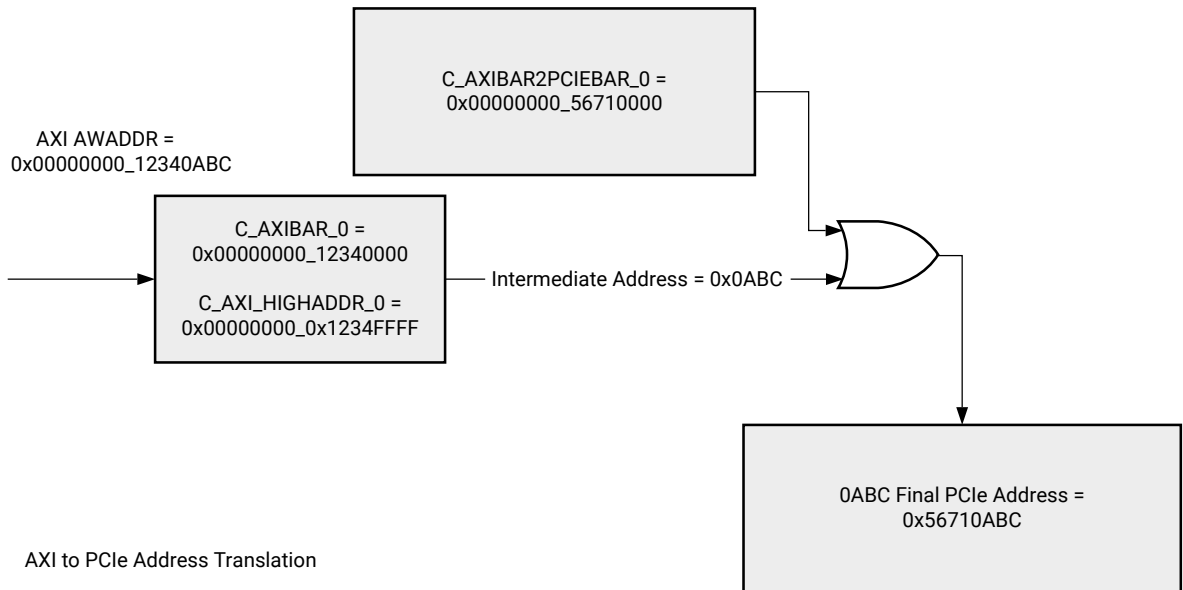


Figure 2: Example 1 Settings



- Accessing the Bridge AXIBAR\_0 with address 0x0000\_12340ABC on the AXI bus yields 0x56710ABC on the bus for PCIe.

Figure 3: AXI to PCIe Address Translation



X20046-032119

- Accessing the Bridge AXIBAR\_1 with address 0x0000\_ABCDF123 on the AXI bus yields 0xFEDC1123 on the bus for PCIe.
- Accessing the Bridge AXIBAR\_2 with address 0x0000\_FFEDCBA on the AXI bus yields 0x41FEDCBA on the bus for PCIe.

### **Example 2 (64-bit PCIe Address Mapping)**

This example shows the generic settings to set up to three independent AXI BARs and address translation of AXI addresses to a remote 64-bit address space for PCIe. This setting of AXI BARs does not depend on the BARs for PCIe within the Bridge.

In this example, where C\_AXIBAR\_NUM=3, the following assignments for each range are made:

```
AXI_ADDR_WIDTH=48
```

```
C_AXIBAR_0=0x00000000_12340000
```

```
C_AXI_HIGHADDR_0=0x00000000_1234FFFF (64 Kbytes)
```

```
C_AXIBAR2PCIEBAR_0=0x50000000056710000 (Bits 63-32 are non-zero in order to  
produce a  
64-bit PCIe TLP. Bits 15-0 must be zero based on the AXI BAR aperture size.  
Non-zero  
values in the lower 16 bits are invalid translation values.)
```

```
C_AXIBAR_1=0x00000000_ABCDE000
```

```
C_AXI_HIGHADDR_1=0x00000000_ABCDFFFF (8 Kbytes)
```

```
C_AXIBAR2PCIEBAR_1=0x60000000_FEDC0000 (Bits 63-32 are non-zero in order to  
produce  
a 64-bit PCIe TLP. Bits 12-0 must be zero based on the AXI BAR aperture  
size. Non-zero  
values in the lower 13 bits are invalid translation values.)
```

```
C_AXIBAR_2=0x00000000_FE000000
```

```
C_AXI_HIGHADDR_2=0x00000000_FFFFFFFF (32 Mbytes)
```

```
C_AXIBAR2PCIEBAR_2=0x70000000040000 (Bits 63-32 are non-zero in order to  
produce a  
64-bit PCIe TLP. Bits 24-0 must be zero based on the AXI BAR aperture size.  
Non-zero  
values in the lower 25 bits are invalid translation values.)
```

Figure 4: Example 2 Settings

- Accessing the Bridge AXIBAR\_0 with address 0x0000\_12340ABC0x5000000056710ABC on the bus for PCIe.
- Accessing the Bridge AXIBAR\_1 with address 0x0000\_ABCDF123 on the AXI bus yields on the AXI bus yields 0x60000000FEDC1123 on the bus for PCIe.
- Accessing the Bridge AXIBAR\_2 with address 0x0000\_FFFEDCBA on the AXI bus yields 0x7000000041FEDCBA on the bus for PCIe.

### Example 3

This example shows the generic settings to set up two independent BARs for PCIe® and address translation of addresses for PCIe to a remote AXI address space. This setting of BARs for PCIe does not depend on the AXI BARs within the bridge.

In this example, where C\_PCIEBAR\_NUM=2, the following range assignments are made.

```
AXI_ADDR_WIDTH=48
```

```
BAR 0 is set to 0x20000000_ABCD8000 by the Root Port. (Since this is a 64-bit BAR
PCIe, BAR1 is disabled.)
PF0_BAR0_APERTURE_SIZE=0x08 (32 Kbytes)
C_PCIEBAR2AXIBAR_0=0x00000000_12340000 (Because the AXI address is 48-bits
wide,
bits 63-48 should be zero. Base on the PCIe Bar Size bits 14-0 should be
```

```

zero.
Non-zero values in these ranges are invalid.)

BAR 2 is set to 0xA000000012000000 by Root Port. (Since this is a 64-bit
BAR PCIe BAR3
is disabled.)
PF0_BAR0_APERTURE_SIZE=0x12 (32 Mbytes)
C_PCIEBAR2AXIBAR_2=0x00000000_FE000000 (Because the AXI address is 48-bits
wide,
bits 63-48 should be zero. Base on the PCIe Bar Size bits 24-0 should be
zero.
Non-zero values in these ranges are invalid.)

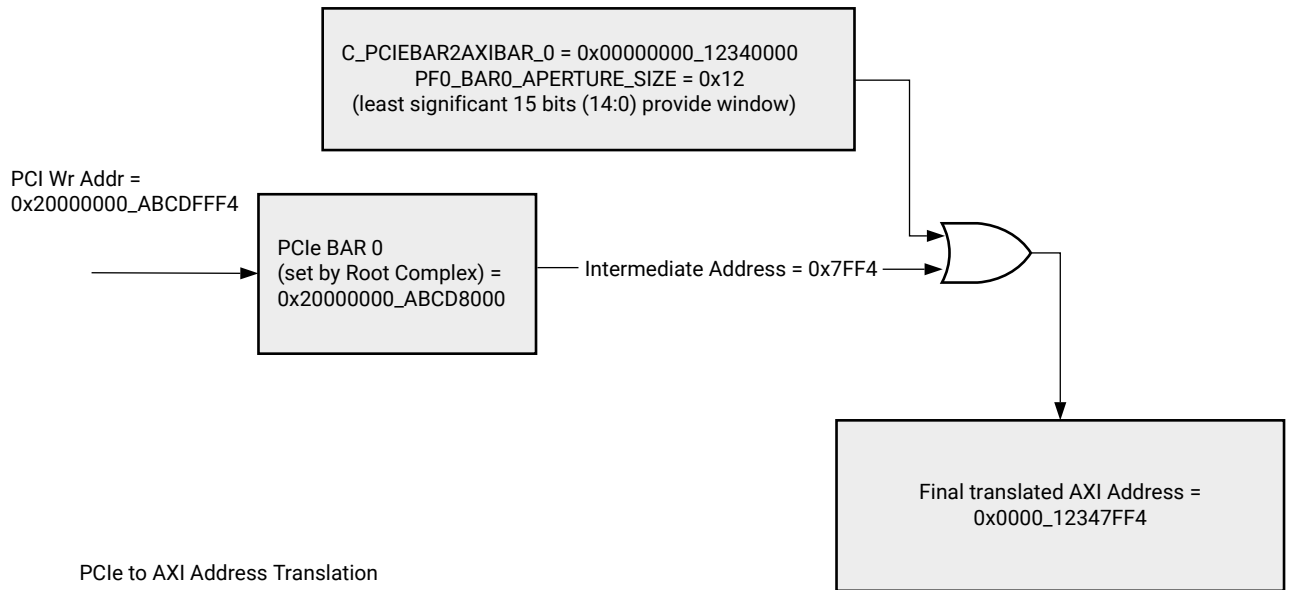
```

Figure 5: Example 3 Settings

The screenshot shows a configuration window for PCIe BARs. The 'PCIe : BARS' tab is selected. There are six sections for PF0\_BAR0 through PF0\_BAR5. Each section has a checkbox to enable it, a 'Type' dropdown (all set to 'Memory'), a 'Size' dropdown, a 'Scale' dropdown, a 'Value' text box, and a 'PCIe to AXI Translation' dropdown. PF0\_BAR0 is enabled with a size of 32 Kilobytes and a value of FFFF8004. PF0\_BAR1 is disabled with a size of 4 Kilobytes and a value of FFFFFFFF. PF0\_BAR2 is enabled with a size of 32 Megabytes and a value of FE000004. PF0\_BAR3 is disabled with a size of 4 Kilobytes and a value of FFFFFFFF. PF0\_BAR4 is disabled with a size of 4 Kilobytes and a value of FFFF0000. PF0\_BAR5 is disabled with a size of 4 Kilobytes and a value of FFFF0000. The 'OK' and 'Cancel' buttons are at the bottom right.

- Accessing the Bridge AXIBAR\_0 with address 0x20000000\_ABCDFFF4 on the bus for PCIe yields 0x0000\_12347FF4 on the AXI bus.

Figure 6: PCIe to AXI Translation



X20047-032119

- Accessing Bridge AXIBAR\_2 with address 0xA00000001235FEDC on the bus for PCIe yields 0x0000\_FE35FEDC on the AXI bus.

### Example 4

This example shows the generic settings of four AXI BARs and address translation of AXI addresses to a remote 32-bit and 64-bit addresses for PCIe®. This setting of AXI BARs do not depend on the BARs for PCIe within the Bridge.

In this example, where number AXI BAR's are 4, the following assignments for each range are made:

```
Aperture_Base_Address_0 =0x00000000_12340000
Aperture_High_Address_0 =0x00000000_1234FFFF (64 KB)
AXI_to_PCIe_Translation_0=0x00000000_56710000 (Bits 63-32 are zero to
produce a 32-bit PCIe
TLP. Bits 15-0 must be zero based on the AXI BAR aperture size. Non-zero
values in
the lower 16 bits are invalid translation values.)

Aperture_Base_Address_1 =0x00000000_ABCDE000
Aperture_High_Address_1 =0x00000000_ABCDFFFF (8 KB)
AXI_to_PCIe_Translation_1=0x50000000_FEDC0000 (Bits 63-32 are non-zero to
produce a 64-bit
PCIe TLP. Bits 12-0 must be zero based on the AXI BAR aperture size. Non-
zero values
in the lower 13 bits are invalid translation values.)

Aperture_Base_Address_2 =0x00000000_FE000000
Aperture_High_Address_2 =0x00000000_FFFFFFFF (32 MB)
```

```
AXI_to_PCIe_Translation_2=0x00000000_40000000 (Bits 63-32 are zero to
produce a 32-bit PCIe
TLP. Bits 24-0 must be zero based on the AXI BAR aperture size. Non-zero
values in
the lower 25 bits are invalid translation values.)

Aperture_Base_Address_3 =0x00000000_00000000
Aperture_High_Address_3 =0x00000000_00000FFF (4 KB)
AXI_to_PCIe_Translation_3=0x60000000_87654000 (Bits 63-32 are non-zero to
produce a 64-bit
PCIe TLP. Bits 11-0 must be zero based on the AXI BAR aperture size. Non-
zero values
in the lower 12 bits are invalid translation values.)
```

Figure 7: Example 4 Settings

- Accessing the Bridge AXI BAR\_0 with address 0x0000\_12340ABC on the AXI bus yields 0x56710ABC on the bus for PCIe.
- Accessing the Bridge AXI BAR\_1 with address 0x0000\_ABCDF123 on the AXI bus yields 0x50000000FEDC1123 on the bus for PCIe.
- Accessing the Bridge AXI BAR\_2 with address 0x0000\_FFFEDCBA on the AXI bus yields 0x41FEDCBA on the bus for PCIe.
- Accessing the Bridge AXI BAR\_3 with address 0x0000\_00000071 on the AXI bus yields 0x6000000087654071 on the bus for PCIe.

## Addressing Checks

When setting the following parameters for PCIe® address mapping, `C_PCIEBAR2AXIBAR_n` and `PF0_BARn_APERTURE_SIZE`, be sure these are set to allow for the addressing space on the AXI system. For example, the following setting is illegal and results in an invalid AXI address.

```
C_PCIEBAR2AXIBAR_n=0x00000000_FFFF000
PF0_BARn_APERTURE_SIZE=0x06 (8 KB)
```

For an 8 Kilobyte BAR, the lower 13 bits must be zero. As a result, the `C_PCIEBAR2AXIBAR_n` value should be modified to be `0x00000000_FFFFE000`. Also, check for a larger value on `PF0_BARn_APERTURE_SIZE` compared to the value assigned to the `C_PCIEBAR2AXIBAR_n` parameter. An example parameter setting follows.

```
C_PCIEBAR2AXIBAR_n=0xFFFF_E000
PF0_BARn_APERTURE_SIZE=0x0D (1 MB)
```

To keep the AXIBAR upper address bits as `0xFFFF_E000` (to reference bits [31:13]), the `PF0_BARn_APERTURE_SIZE` parameter must be set to `0x06` (8 KB).

## Malformed TLP

The integrated block for PCI Express® detects a malformed TLP. For the IP configured as an Endpoint core, a malformed TLP results in a fatal error message being sent upstream if error reporting is enabled in the Device Control register.

## Abnormal Conditions

This section describes how the Slave side and Master side (see the following tables) of the Bridge core handle abnormal conditions.

### Slave Bridge Abnormal Conditions

#### Illegal Burst Type

The slave bridge monitors AXI read and write burst type inputs to ensure that only the INCR (incrementing burst) type is requested. Any other value on these inputs is treated as an error condition and the Slave Illegal Burst (SIB) interrupt is asserted. In the case of a read request, the Bridge asserts SLVERR for all data beats and arbitrary data is placed on the `s_axi_rdata` bus. In the case of a write request, the Bridge asserts SLVERR for the write response and all write data is discarded.

## Completion TLP Errors

Any request to the bus for PCIe (except for a posted Memory write) requires a completion TLP to complete the associated AXI request. The Slave side of the Bridge checks the received completion TLPs for errors and checks for completion TLPs that are never returned (Completion Timeout). Each of the completion TLP error types are discussed in the subsequent sections.

### Unexpected Completion

When the slave bridge receives a completion TLP, it matches the header RequesterID and Tag to the outstanding RequesterID and Tag. A match failure indicates the TLP is an Unexpected Completion which results in the completion TLP being discarded and a Slave Unexpected Completion (SUC) interrupt strobe being asserted. Normal operation then continues.

### Unsupported Request

A device for PCIe might not be capable of satisfying a specific read request. For example, if the read request targets an unsupported address for PCIe, the completer returns a completion TLP with a completion status of `0b001 - Unsupported Request`. The completer that returns a completion TLP with a completion status of `Reserved` must be treated as an unsupported request status, according to the PCI Express Base Specification v3.0. When the slave bridge receives an unsupported request response, the Slave Unsupported Request (SUR) interrupt is asserted and the DECERR response is asserted with arbitrary data on the AXI4 memory mapped bus.

### Completion Timeout

A Completion Timeout occurs when a completion (Cpl) or completion with data (CplD) TLP is not returned after an AXI to PCIe memory read request, or after a PCIe Configuration Read/Write request. For PCIe Configuration Read/Write request, completions must complete within the `C_COMP_TIMEOUT` parameter selected value from the time the request is issued. For PCIe Memory Read request, completions must complete within the value set in the Device Control 2 register in the PCIe Configuration Space register. When a completion timeout occurs, an OKAY response is asserted with all 1s data on the memory mapped AXI4 bus.

### Poison Bit Received on Completion Packet

An Error Poison occurs when the completion TLP EP bit is set, indicating that there is poisoned data in the payload. When the slave bridge detects the poisoned packet, the Slave Error Poison (SEP) interrupt is asserted and the SLVERR response is asserted with arbitrary data on the memory mapped AXI4 bus.



### Completer Abort

A Completer Abort occurs when the completion TLP completion status is 0b100 - Completer Abort. This indicates that the completer has encountered a state in which it was unable to complete the transaction. When the slave bridge receives a completer abort response, the Slave Completer Abort (SCA) interrupt is asserted and the SLVERR response is asserted with arbitrary data on the memory mapped AXI4 bus.

**Table 5: Slave Bridge Response to Abnormal Conditions**

Transfer Type	Abnormal Condition	Bridge Response
Read	Illegal burst type	SIB interrupt is asserted. SLVERR response given with arbitrary read data.
Write	Illegal burst type	SIB interrupt is asserted. Write data is discarded. SLVERR response given.
Read	Unexpected completion	SUC interrupt is asserted. Completion is discarded.
Read	Unsupported Request status returned	SUR interrupt is asserted. DECERR response given with arbitrary read data.
Read	Completion timeout	SCT interrupt is asserted. SLVERR response given with arbitrary read data.
Read	Poison bit in completion	Completion data is discarded. SEP interrupt is asserted. SLVERR response given with arbitrary read data.
Read	Completer Abort (CA) status returned	SCA interrupt is asserted. SLVERR response given with arbitrary read data.

### Master Bridge Abnormal Conditions

The following sections describe the manner in which the master bridge handles abnormal conditions.

#### AXI DECERR Response

When the master bridge receives a DECERR response from the AXI bus, the request is discarded and the Master DECERR (MDE) interrupt is asserted. If the request was non-posted, a completion packet with the Completion Status = Unsupported Request (UR) is returned on the bus for PCIe.

### AXI SLVERR Response

When the master bridge receives a SLVERR response from the addressed AXI slave, the request is discarded and the Master SLVERR (MSE) interrupt is asserted. If the request was non-posted, a completion packet with the Completion Status = Completer Abort (CA) is returned on the bus for PCIe.

### Max Payload Size for PCIe, Max Read Request Size or 4K Page Violated

When the master bridge receives a SLVERR response from the addressed AXI slave, the request is discarded and the Master SLVERR (MSE) interrupt is asserted. If the request was non-posted, a completion packet with the Completion Status = Completer Abort (CA) is returned on the bus for PCIe.

### Completion Packets

When the `MAX_READ_REQUEST_SIZE` is greater than the `MAX_PAYLOAD_SIZE`, a read request for PCIe can ask for more data than the master bridge can insert into a single completion packet. When this situation occurs, multiple completion packets are generated up to `MAX_PAYLOAD_SIZE`, with the Read Completion Boundary (RCB) observed.

### Poison Bit

When the poison bit is set in a transaction layer packet (TLP) header, the payload following the header is corrupted. When the master bridge receives a memory request TLP with the poison bit set, it discards the TLP and asserts the Master Error Poison (MEP) interrupt strobe.

### Zero Length Requests

When the master bridge receives a read request with the `Length = 0x1`, `FirstBE = 0x00`, and `LastBE = 0x00`, it responds by sending a completion with `Status = Successful Completion`.

When the master bridge receives a write request with the `Length = 0x1`, `FirstBE = 0x00`, and `LastBE = 0x00` there is no effect.

**Table 6: Master Bridge Response to Abnormal Conditions**

Transfer Type	Abnormal Condition	Bridge Response
Read	DECERR response	MDE interrupt strobe asserted. Completion returned with Unsupported Request status.
Write	DECERR response	MDE interrupt strobe asserted.
Read	SLVERR response	MSE interrupt strobe asserted. Completion returned with Completer Abort status.
Write	SLVERR response	MSE interrupt strobe asserted.

**Table 6: Master Bridge Response to Abnormal Conditions (cont'd)**

Transfer Type	Abnormal Condition	Bridge Response
Write	Poison bit set in request	MEP interrupt strobe asserted. Data is discarded.
Read	DECERR response	MDE interrupt strobe asserted. Completion returned with Unsupported Request status.
Write	DECERR response	MDE interrupt strobe asserted.

## Link Down Behavior

The normal operation of the Bridge core is dependent on the integrated block for PCIe establishing and maintaining the point-to-point link with an external device for PCIe. If the link has been lost, it must be re-established to return to normal operation.

When a Hot Reset is received by the Bridge core, the link goes down and the PCI Configuration Space must be reconfigured.

Initiated AXI4 write transactions that have not yet completed on the AXI4 bus when the link goes down have a SLVERR response given and the write data is discarded. Initiated AXI4 read transactions that have not yet completed on the AXI4 bus when the link goes down have a SLVERR response given, with arbitrary read data returned.

Any MemWr TLPs for PCIe that have been received, but the associated AXI4 write transaction has not started when the link goes down, are discarded.

## Endpoint

When configured to support Endpoint functionality, the Bridge core fully supports Endpoint operation as supported by the underlying block. There are a few details that need special consideration. The following subsections contain information and design considerations about Endpoint support.

## Interrupts

Interrupt capabilities are provided by the underlying PCI Express® solution IP. For additional information, see the *Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide (PG023)*, the *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)*, and the *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)*.

## AXI Bridge for PCIe Gen3

The Interrupts modes in the following section applies to AXI Bridge for PCIe Gen3 only. For DMA/Bridge Subsystem for PCIe in Bridge Mode, go to the next section.

Multiple interrupt modes can be configured during IP configuration, however only one interrupt mode is used at runtime. If multiple interrupt modes are enabled by the host after PCI bus enumeration at runtime, the core uses the MSI interrupt over Legacy interrupt. Both MSI and Legacy interrupt modes are sent using the same `int_msi_*` interface, and the core automatically picks the best available interrupt mode at runtime. MSI-X is implemented externally to the core and uses a separate `cfg_interrupt_msix_*` interface. The core does not prevent the use of MSI-X at any time even when other interrupt modes are enabled, however Xilinx recommends the use of MSI-X interrupt solely if enabled over other interrupt modes even if they are all enabled at runtime.

### Legacy Interrupts

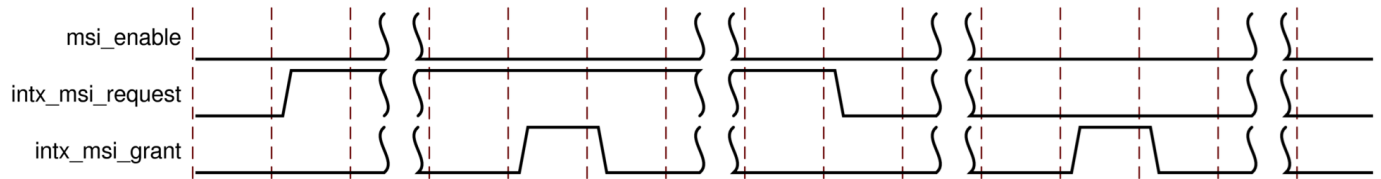
Asserting `intx_msi_request` when legacy interrupts are enabled causes the IP to issue a legacy interrupt over PCIe. After a `intx_msi_grant` signal is asserted, it must remain asserted until the `intx_msi_grant` signal is asserted and the interrupt has been serviced and cleared by the host. The `intx_msi_grant` assertion indicates the requested interrupt has been sent on the PCIe block. The Message sent is based on the value of the `PFO_INTERRUPT_PIN` parameter, which is configurable during core customization in the Vivado Design Suite. Keeping `intx_msi_grant` asserted ensures the interrupt pending register within the IP remains asserted when queried by the host's Interrupt Service Routine (ISR) to determine the source of interrupts. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the host software when an interrupt is serviced.

After the `intx_msi_request` signal is deasserted, it cannot be reasserted until the `intx_msi_grant` signal has been asserted for a second time. This indicates the deassertion message for the legacy interrupt has been sent over PCIe. After a second `intx_msi_grant` has occurred, the `intx_msi_request` signal can be reasserted to generate another legacy interrupt.

The following figure shows the legacy interrupts.

This figure shows only the handshake between `intx_msi_request` and `intx_msi_grant`. The user application might not clear or service the interrupt immediately, in which case, you must keep `intx_msi_request` asserted past `intx_msi_grant`.

Figure 8: Legacy Interrupts



## Related Information

[Bridge Parameters](#)

## MSI Interrupts

Asserting `intx_msi_request` causes the generation of an MSI interrupt if MSI is enabled.

After `intx_msi_request` is asserted, it should be de-asserted immediately in the next clock cycle. The `intx_msi_grant` assertion indicates the requested interrupt has been sent on the PCIe block. The vector number being used in the interrupt packet will be determined based on the value provided at the `msi_vector_num[4:0]` signal. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the host software when an Interrupt is serviced.

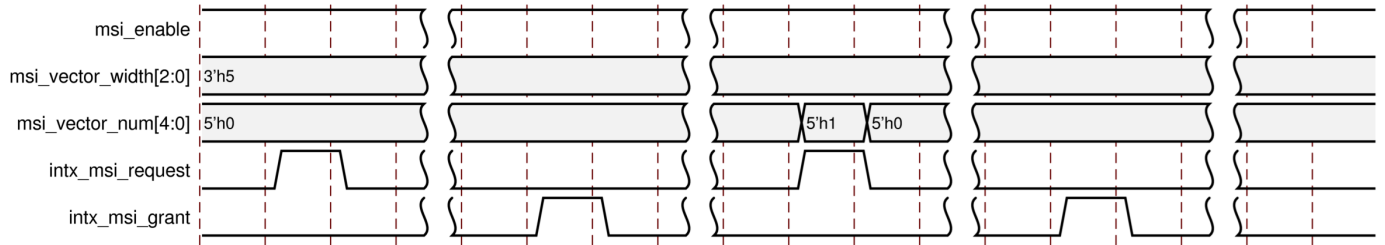
The `intx_msi_request` signal is positive-edge detected and synchronous to `axi_aclk`. `intx_msi_request` must be de-asserted for at least one `axi_aclk` cycle before re-asserting it to send another MSI interrupt on the same vector or on a different vector.

The MSI vector number being used must not exceed the number of MSI vectors being enabled by the host. This information can be queried from `msi_vector_width[2:0]` signal after the host enumerated and enabled MSI interrupt at runtime. The encoding of `msi_vector_width[2:0]` signal is shown in the following table.

Table 7: MSI Vectors Enabled in Message Control Register

Value	Number of Messages Requested	Output Signal, MSI_Vector_Width (2:0)
000	1	000
001	2	001
010	4	010
011	8	011
100	16	100
101	32	101

Figure 9: MSI Interrupts



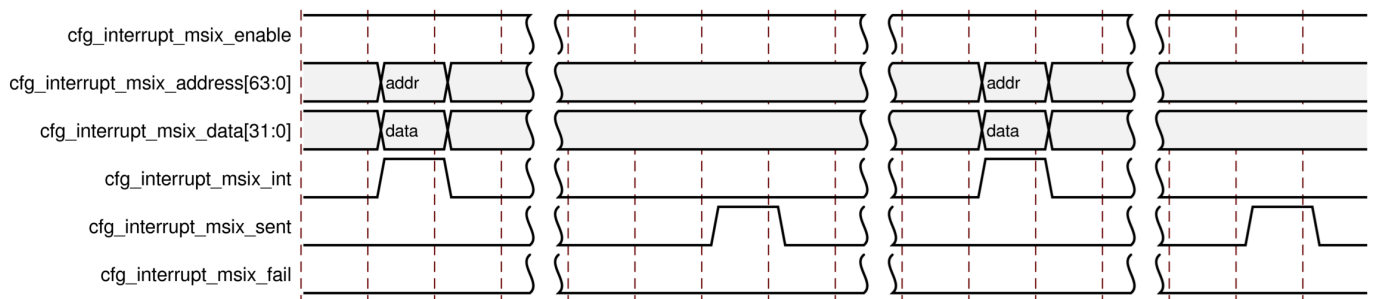
### External MSI-X Interrupts

**Note:** This interrupt mode is only available in AXI Bridge for PCI Express Gen3.

The core supports the MSI-X interrupt and its signaling. The MSI-X vector table and the MSI-X Pending Bit Array need to be implemented as part of the user logic, by claiming a BAR aperture. The External MSI-X interrupts mode is enabled when you set the MSI-X Implementation Location option to External in the PCIe Misc Tab.

To send MSI-X interrupt, user logic must use `cfg_interrupt_msix_*` interface instead of the `intx_msi_*` interface. The signaling requirement is the same as defined in the UltraScale Devices Gen3 Integrated Block for PCIe core as shown below.

Figure 10: External MSI-X Interrupts



### DMA/Bridge Subsystem for PCIe in Bridge Mode

The Interrupts modes in the following section applies to DMA/Bridge Subsystem for PCIe in Bridge Mode only. For AXI Bridge for PCIe Gen3, go to the previous section.

Multiple interrupt modes can be configured during IP configuration, however only one interrupt mode is used at runtime. If multiple interrupt modes are enabled by the host after PCI bus enumeration at runtime, MSI-X interrupt takes precedence over MSI interrupt, and MSI interrupt takes precedence over Legacy interrupt. All of these interrupt modes are sent using the same `usr_irq_*` interface and the core automatically picks the best available interrupt mode at runtime.

## Legacy Interrupts

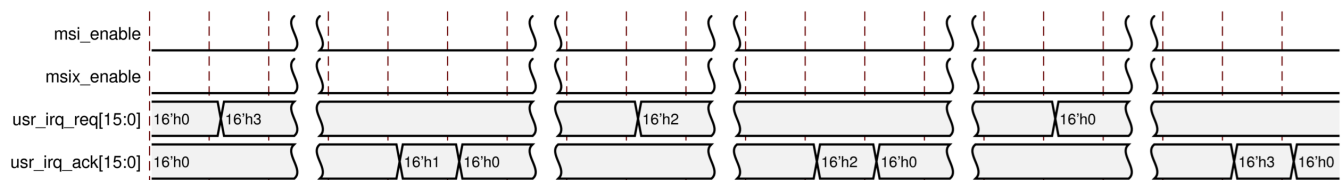
Asserting one or more bits of `usr_irq_req` when legacy interrupts are enabled causes the IP to issue a legacy interrupt over PCIe. Multiple bits may be asserted simultaneously but each bit must remain asserted until the corresponding `usr_irq_ack` bit has been asserted. After a `usr_irq_req` bit is asserted, it must remain asserted until the corresponding `usr_irq_ack` bit is asserted and the interrupt has been serviced and cleared by the Host. The `usr_irq_ack` assertion indicates the requested interrupt has been sent on the PCIe block. This will ensure interrupt pending register within the IP remains asserted when queried by the Host's Interrupt Service Routine (ISR) to determine the source of interrupts. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the Host software when an interrupt is serviced.

After the `usr_irq_req` bit is deasserted, it cannot be reasserted until the corresponding `usr_irq_ack` bit has been asserted for a second time. This indicates the deassertion message for the legacy interrupt has been sent over PCIe. After a second `usr_irq_ack` occurred, the `usr_irq_req` wire can be reasserted to generate another legacy interrupt.

The `usr_irq_req` bit can be mapped to legacy interrupt INTA, INTB, INTC, INTD through the configuration registers. The following figure shows the legacy interrupts.

This figure shows only the handshake between `usr_irq_req` and `usr_irq_ack`. The user application might not clear or service the interrupt immediately, in which case, you must keep `usr_irq_req` asserted past `usr_irq_ack`.

Figure 11: Legacy Interrupts



## MSI and Internal MSI-X Interrupts

**Note:** Internal MSI-X Interrupt mode is only available in DMA/Bridge Subsystem for PCIe in Bridge Mode.

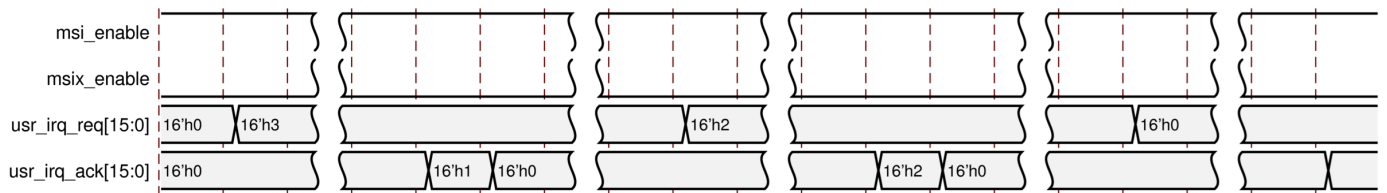
Asserting one or more bits of `usr_irq_req` causes the generation of an MSI or MSI-X interrupt if MSI or MSI-X is enabled. If both MSI and MSI-X capabilities are enabled, an MSI-X interrupt is generated. The Internal MSI-X interrupts mode is enabled when you set the MSI-X Implementation Location option to Internal in the PCIe Misc Tab.

After a `usr_irq_req` bit is asserted, it must remain asserted until the corresponding `usr_irq_ack` bit is asserted and the interrupt has been serviced and cleared by the Host. The `usr_irq_ack` assertion indicates the requested interrupt has been sent on the PCIe block. This will ensure the interrupt pending register within the IP remains asserted when queried by the Host's Interrupt Service Routine (ISR) to determine the source of interrupts. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the Host software when an Interrupt is serviced.

Configuration registers are available to map `usr_irq_req` and DMA interrupts to MSI or MSI-X vectors. For MSI-X support, there is also a vector table and PBA table. The following figure shows the MSI interrupt.

This figure shows only the handshake between `usr_irq_req` and `usr_irq_ack`. Your application might not clear or service the interrupt immediately, in which case, you must keep `usr_irq_req` asserted past `usr_irq_ack`.

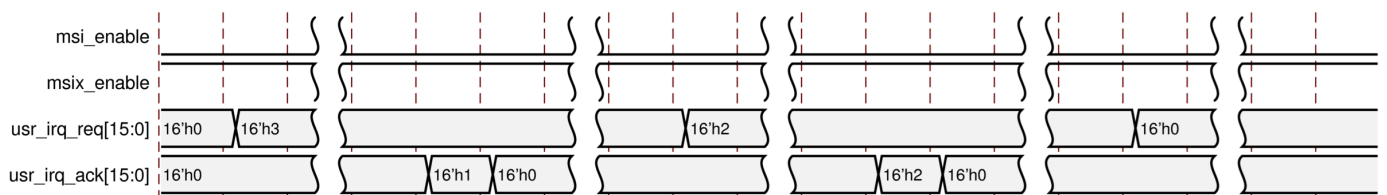
Figure 12: MSI Interrupts



The following figure shows the MSI-X interrupt.

This figure shows only the handshake between `usr_irq_req` and `usr_irq_ack`. Your application might not clear or service the interrupt immediately, in which case, you must keep `usr_irq_req` asserted past `usr_irq_ack`.

Figure 13: MSI-X Interrupts





## Root Port

When configured to support Root Port functionality, the Bridge core fully supports Root Port operation as supported by the underlying block. There are a few details that need special consideration. The following subsections contain information and design considerations about Root Port support.

### ***Power Limit Message TLP***

The Bridge core automatically sends a Power Limit Message TLP when the Master Enable bit of the Command Register is set. The software must set the Requester ID register before setting the Master Enable bit to ensure that the desired Requester ID is used in the Message TLP.

### ***Root Port Configuration Read***

When an ECAM access is performed to the primary bus number, self-configuration of the integrated block for PCIe is performed. A PCIe configuration transaction is not performed and is not presented on the link. When an ECAM access is performed to the bus number that is equal to the secondary bus value in the Enhanced PCIe Type 1 configuration header, then Type 0 configuration transactions are generated.

When an ECAM access is attempted to a bus number that is in the range defined by the secondary bus number and subordinate bus number range (not including secondary bus number), then Type 1 configuration transactions are generated. The primary, secondary and subordinate bus numbers are written and updated by Root Port software to the Type 1 PCI Configuration Header of the Bridge core in the enumeration procedure.

When an ECAM access is attempted to a bus number that is out of the range defined by the secondary bus\_number and subordinate bus number, the bridge does not generate a configuration request and signal a SLVERR response on the AXI4 bus.

When an Unsupported Request (UR) response is received for a configuration read request, all ones are returned on the AXI4 bus to signify that a device does not exist at the requested device address. It is the responsibility of the software to ensure configuration write requests are not performed to device addresses that do not exist. However, the Bridge core asserts SLVERR response on the AXI4 bus when a configuration write request is performed on device addresses that do not exist or a UR response is received.

### ***Root Port BAR***

Root Port BAR does not support packet filtering (all TLPs received from PCIe link are forwarded to the user logic), however Address Translation can be configured to enable or disable, depending on the IP configuration.

During core customization in the Vivado® Design Suite, when there is no BAR enabled, RP passes all received packets to the user application without address translation or address filtering.

When BAR is enabled, by default the BAR address starts at 0x0000\_0000 unless programmed separately. Any packet received from the PCIe® link that hits a BAR is translated according to the PCIe-to-AXI Address Translation rules.

**Note:** The IP must not receive any TLPs outside of the PCIe BAR range from the PCIe link when RP BAR is enabled. If this rule cannot be enforced, it's recommended that the PCIe BAR is disabled and do address filtering and/or translation outside of the IP.

The Root Port BAR customization options in the Vivado Design Suite are found in the PCIe BARs Tab.

**Related Information**

- [PCIe BARs Tab](#)
- [PCIe BARs Tab](#)

**Configuration Transaction Timeout**

Configuration transactions are non-posted transactions. The Bridge core has a timer for timeout termination of configuration transactions that have not completed on the PCIe link. SLVERR is returned when a configuration timeout occurs. Timeouts of configuration transactions are flagged by an interrupt as well.

**Abnormal Configuration Transaction Termination Responses**

Responses to abnormal terminations to configuration transactions are shown in the following table.

*Table 8: Responses of Bridge to Abnormal Configuration Terminations*

Transfer Type	Abnormal Condition	Bridge Response
Config Read or Write	Bus number not in the range of primary bus number through subordinate bus number.	SLVERR response is asserted.
Config Read or Write	Completion timeout.	For PCIe Configuration Read/Write request, an OKAY response and 0s data are given on the AXI4 memory mapped bus.
Config Write	Bus number in the range of secondary bus number through subordinate bus number and UR is returned.	SLVERR response is asserted.

## Receiving Interrupts

In Root Port mode, you can choose one of the two ways to handle incoming interrupts;

- Legacy Interrupt FIFO mode: Legacy Interrupt FIFO mode is the default. It is available in earlier Bridge IP variants and versions, and will continue to be available. Legacy Interrupt FIFO mode is geared towards compatibility for legacy designs.
- Interrupt Decode mode: Interrupt Decode mode is available in the CPM AXI Bridge. Interrupt Decode mode can be used to mitigate Interrupt FIFO overflow condition which can occur in a design that receives interrupts at a high rate and avoids the performance penalty incurred when such condition occurs.

To enable Legacy Interrupt FIFO mode:

```
set_property -dict [list CONFIG.msi_rx_pin_en {false}] [get_ips <ip_name>]
```

To enable Interrupt Decode mode:

```
set_property -dict [list CONFIG.msi_rx_pin_en {true}] [get_ips <ip_name>]
```

If you are customizing and generating the core in the Vivado® IP integrator, replace `get_ips` with `get_bd_cells`.

## Legacy Interrupt FIFO Mode

### Legacy INTx Interrupt

When the IP receives an INTx interrupt, the Interrupt Decode register bit[16] is set. If the Interrupt Mask register bit[16] is also set, the `interrupt_out` pin is asserted. After receiving this interrupt, the user application must follow the following procedure to service the interrupt:

1. *Optional:* Write 0 to the Interrupt Mask register bit [16] to deassert the `interrupt_out` pin while interrupt is being serviced.
2. Read the Root Port Status/Control Register bit [18] to check if it is not empty.
3. Read the Root Port Status/Control Register bit [19] to check if it has overflowed.
4. If the interrupt FIFO is not empty, read Root Port Interrupt FIFO Read Register 1 to check which interrupt line is serviced, and whether this is an Assertion or Deassertion message.
5. Write 1 to the Root Port Interrupt FIFO Read Register 1 bit [31] to remove the interrupt that the user has just read from the FIFO.
6. Repeat from step 2 to step 5 until the FIFO is indicated as empty.
7. If at any time during this process the FIFO is indicated as overflowed (status from step 2), the user application must check any interrupt line that has not been serviced to check for any pending interrupt on that line. Failure to do this before continuing may leave some interrupt line unserved.

8. Write 1 to the Interrupt Decode register bit [16] to clear the INTx interrupt bit.
9. If step 1 is executed, write 1 to the Interrupt Mask register bit [16] to re-enable the `interrupt_out` pin for future INTx interrupt.

### **MSI Interrupt**

The IP will decode the MSI interrupt based on the value programmed in Root Port MSI Base Register 1 and Root Port MSI Base Register 2. Any Memory Write TLP received from the link with an address that falls within a 4 Kb window from the base address programmed in those registers will be treated as an MSI interrupt, and will not be forwarded to the M\_AXI(B) interface. When an MSI interrupt is received, the Interrupt Decode register bit[17] will be set. If the Interrupt Mask register bit[17] is also set, the `interrupt_out` pin is asserted. After receiving this interrupt, the user application must follow the following procedure to service the interrupt:

1. Optional: Write 0 to the Interrupt Mask register bit [17] to deassert the `interrupt_out` pin while the interrupt is being serviced.
2. Read the Root Port Status/Control Register bit [18] to check if it is not empty.
3. Read the Root Port Status/Control Register bit [19] to check if it has overflowed.
4. If the interrupt FIFO is not empty, read the Root Port Interrupt FIFO Read Register 2 to check MSI Message Data from the received MSI interrupt. This is used by the user application to determine the interrupt vector number and can also be used to determine the source of the interrupt.
5. Write 1 to the Root Port Interrupt FIFO Read Register 1 bit [31] to remove the interrupt user has just read from the FIFO.
6. Repeat from step 2 until the FIFO is indicated as empty.
7. If at any time during this process, the FIFO was indicated as overflowed (status from step 2), the user application must check any unserviced interrupt vectors to check for any pending interrupts on that line. Failure to do this before continuing can leave some interrupt vector unserviced.
8. Write 1 to the Interrupt Decode Register bit [17] to clear the MSI interrupt bit.
9. If step 1 was executed, write 1 to the Interrupt Mask Register bit [17] to re-enable the `interrupt_out` pin for future MSI interrupts.

### **MSI-X Interrupt**

All MSI-X interrupts must be decoded by the user application externally to the IP. To do this, set all of their Endpoints to use an MSI-X address that falls outside of the range of the 4Kb window from the base address programmed in the Root Port MSI Base Register 1 and Root Port MSI Base Register 2. All MSI-X interrupts will be forwarded to the M\_AXI(B) interface.

All TLPs forwarded to M\_AXI(B) interface are subject to the PCIe-to-AXI Address translation.

## Interrupt Decode Mode

### Legacy INTx Interrupt

When the IP has received an INTx interrupt, the Root Port Interrupt Decode 2 register is set. If the Root Port Interrupt Decode 2 Mask register is also set, the `interrupt_out` pin is asserted. After receiving this interrupt, the user application must follow this procedure to service the interrupt:

1. Optional: Write 0 to the Interrupt Decode 2 Mask register to deassert an interrupt line while the interrupt is being serviced.
2. Read the Root Port Interrupt Decode 2 register to check which interrupt line is currently asserted.
3. Repeat step 2 until all interrupt lines are deasserted. The interrupt line is automatically cleared when the IP receives the INTx Deassert Message corresponding to that interrupt line.
4. If step 1 was executed, write 1 to the Interrupt Decode 2 Mask register to re-enable an interrupt line for future INTx interrupt.

### MSI Interrupt

The IP decodes the MSI interrupt based on the value programmed in Root Port MSI Base Register 1 and Root Port MSI Base Register 2. Any Memory Write TLPs received from the link with an address that falls within the 4 Kb window from the base address programmed in those registers will be treated as MSI interrupt, and will not be forwarded to the M\_AXI(B) interface.

**Note:** MSI Message Data [5:0] will always be decoded as MSI Message vector regardless of how many vectors are enabled at your Endpoint.

When an MSI interrupt is received, the Root Port MSI Interrupt Decode 1 or Root Port MSI Interrupt Decode 2 register is set. If the Root Port MSI Interrupt Decode 1 or Root Port MSI Interrupt Decode 2 register is also set, the `interrupt_out_msi_vec*` pins are asserted. `interrupt_out_msi_vec0to31` corresponds to MSI vector 0 - 31, and `interrupt_out_msi_vec32to63` corresponds to MSI vector 32 - 63. After receiving this interrupt, the user application must follow this procedure to service the interrupt:

1. Optional: Write 0 to the Root Port MSI Interrupt Decode 1 or 2 Mask register to deassert the `interrupt_out_msi_vec*` pins while the interrupt is being serviced.
2. Read the Root Port MSI Interrupt Decode 1 or 2 register to check which interrupt vector is asserted.
3. Write 1 to the Root Port MSI Interrupt Decode 1 or 2 register to clear the MSI interrupt bit.
4. If step 1 was executed, write 1 to the Root Port MSI Interrupt Decode 1 or 2 Mask register bit to re-enable the `interrupt_out_msi_vec*` pins for future MSI interrupts.

## MSI-X Interrupt

All MSI-X interrupts must be decoded by the user application externally to the IP. To do this, the user application must set all Endpoints to use an MSI-X address that falls outside of the range of the 4Kb window from the base address programmed in the Root Port MSI Base Register 1 and the Root Port MSI Base Register 2. All MSI-X interrupts are forwarded to the M\_AXI(B) interface.

All TLPs forwarded to M\_AXI(B) interface are subject to PCIe-to-AXI Address translation.

# Port Descriptions

The interface signals for the core are described in the following tables.

## Global Signals

The interface signals for the Bridge are described in the following table.

*Table 9: Global Signals*

Signal Name	I/O	Description
refclk	I	AXI Bridge for PCIe Gen3 only: Virtex®-7: PCIe Reference Clock. Should be driven from the O port of reference clock IBUFDS_GTE2. UltraScale™: Dynamic reconfiguration port (DRP) Clock and Internal System Clock (Half frequency from sys_clk_gt frequency if PCIe Reference Clock is 250 MHz, otherwise same frequency as sys_clk_gt frequency). Should be driven by the ODIV2 port of reference clock IBUFDS_GTE3.
sys_clk	I	DMA/Bridge Subsystem for PCIe in AXI Bridge mode only: UltraScale+: Dynamic reconfiguration port (DRP) Clock and Internal System Clock (Half frequency from sys_clk_gt frequency if PCIe Reference Clock is 250 MHz, otherwise same frequency as sys_clk_gt frequency). Should be driven by the ODIV2 port of reference clock IBUFDS_GTE4.
sys_clk_gt	I	PCIe Reference Clock. UltraScale: Should be driven from the O port of reference clock IBUFDS_GTE3. UltraScale+: Should be driven from the O port of reference clock IBUFDS_GTE4.
sys_rst_n	I	Reset from the PCIe edge connector reset signal.

Table 9: Global Signals (cont'd)

Signal Name	I/O	Description
axi_aclk	O	AXI Bridge for PCIe Gen3: PCIe derived clock output for M_AXI, S_AXI interfaces, and all interrupt sideband (MSI, MSI-X, and Bridge Mode Interrupt) signals. DMA/Bridge Subsystem for PCIe in AXI Bridge mode: PCIe derived clock output for M_AXIB, S_AXIB, and S_AXIL interfaces. axi_aclk is a derived clock from the TXOUTCLK pin from the GT block; it is not expected to run continuously while axi_aresetn is asserted.
axi_ctl_aclk	I	AXI Bridge for PCIe Gen3 only: aclk for the S_AXI_CTL interface. Recommended to be driven by the axi_aclk output. axi_ctl_aclk is a derived clock from the TXOUTCLK pin from the GT block; it is not expected to run continuously while axi_ctl_aresetn is asserted.  This pin is for legacy use mode only. By default, new IP generation will have this clock pin internally driven by the IP. Use axi_aclk pin to clock the design.
interrupt_out	O	Interrupt signal. It is asserted for as long as there exists at least one bit asserted in the Interrupt Decode register and is not masked in the Interrupt Mask register, and/or asserted in the Interrupt Decode 2 register and is not masked in the Interrupt Decode 2 Mask register.
interrupt_out_msi_vec0to31	O	Interrupt signal. It is asserted for as long as there exists at least one bit asserted in the Root Port MSI Interrupt Decode 1 register and is not masked in the Root Port MSI Interrupt Decode 1 Mask register. Only available in Root Port configuration with Interrupt Decode mode
interrupt_out_msi_vec32to63	O	Interrupt signal. It is asserted for as long as there exists at least one bit asserted in the Root Port MSI Interrupt Decode 2 register and is not masked in the Root Port MSI Interrupt Decode 2 Mask register. Only available in Root Port configuration with Interrupt Decode mode.
axi_aresetn	O	AXI Bridge for PCIe Gen3: Reset signal for the S_AXI and M_AXI interfaces. axi_aresetn deasserts after a function has transitioned into D0_active power state (configured and enabled). DMA/Bridge Subsystem for PCIe in AXI Bridge mode: In Endpoint configuration, reset signal for the S_AXIB and M_AXIB interfaces. axi_aresetn deasserts after a function has transitioned into D0_active power state (configured and enabled). In Root Port configuration, axi_aresetn deasserts after GT transceivers are initialized (assertion of Phy Ready, independent from PCIe link status).  <b>IMPORTANT!</b> <i>The default value of this pin in the DMA/Bridge Subsystem for PCIe in AXI Bridge mode has changed as mentioned above.</i>

Table 9: Global Signals (cont'd)

Signal Name	I/O	Description
axi_ctl_aresetn	O	<p>AXI Bridge for PCIe Gen3: Reset signal for the S_AXI_CTL interface. axi_ctl_aresetn deasserts after two axi_ctl_aclk (or axi_aclk for newer IP that does not have the axi_ctl_aresetn pin) cycles after sys_rst_n deasserts.</p> <p>DMA/Bridge Subsystem for PCIe in AXI Bridge mode: Reset signal for the S_AXIL interface. In Endpoint configuration, axi_ctl_aresetn deasserts after a function has transitioned into D0_active power state (configured and enabled). In Root Port configuration, axi_ctl_aresetn deasserts after GT transceivers are initialized (assertion of Phy Ready, independent from PCIe link status).</p> <hr/> <p><b>IMPORTANT!</b> The default value of this pin in the DMA/Bridge Subsystem for PCIe in AXI Bridge mode has changed as mentioned above.</p>
dma_bridge_resetn	I	<p>Optional pin available to DMA/Bridge Subsystem for PCIe in AXI Bridge mode only and appears only when SOFT_RESET_EN parameter is set to TRUE. This pin is intended to be user driven reset when link down, Function Level Reset, Dynamic Function eXchange, or another error condition defined by user has occurred. It is not required to be toggled during initial link up operation.</p> <p>When used, all PCIe traffic must be in quiesce state. The signal must be asserted for longer than the Completion Timeout value (typically 50 ms).</p> <p>0: Resets all internal Bridge engines and registers as well as asserts axi_aresetn and axi_ctl_aresetn signals while maintaining PCIe link up.</p> <p>1: Normal operation.</p>

## AXI Slave Interface

For all signals in this interface, s\_axi\_\* is used for the AXI Bridge for PCIe Gen3 core, and s\_axib\_\* is used for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode.

Table 10: AXI Slave Interface Signals

Signal Name	I/O	Description
s_axi(b)_awid[c_s_axi_id_width-1:0]	I	Slave write address ID
s_axi(b)_awaddr[axi_addr_width-1:0]	I	Slave write address
s_axi(b)_awregion[3:0]	I	Slave write region decode
s_axi(b)_awlen[7:0]	I	Slave write burst length
s_axi(b)_awsize[2:0]	I	Slave write burst size
s_axi(b)_awburst[1:0]	I	Slave write burst type
s_axi(b)_awvalid	I	Slave address write valid
s_axi(b)_awready	O	Slave address write ready



Table 10: AXI Slave Interface Signals (cont'd)

Signal Name	I/O	Description
s_axi(b)_wdata[axi_data_width-1:0]	I	Slave write data
s_axi(b)_wstrb[axi_data_width/8-1:0]	I	Slave write strobe
s_axi(b)_wlast	I	Slave write last
s_axi(b)_wvalid	I	Slave write valid
s_axi(b)_wready	O	Slave write ready
s_axi(b)_wuser	I	Reserved. Tie to GND.  <b>Note:</b> This signal is disabled for AXI Bridge for PCI Express Gen3. This signal is also disabled for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode unless CONFIG.parity_settings = Propagate_Parity is set.
s_axi(b)_bid[c_s_axi_id_width-1:0]	O	Slave response ID
s_axi(b)_bresp[1:0]	O	Slave write response
s_axi(b)_bvalid	O	Slave write response valid
s_axi(b)_bready	I	Slave response ready
s_axi(b)_arid[c_s_axi_id_width-1:0]	I	Slave read address ID
s_axi(b)_araddr[axi_addr_width-1:0]	I	Slave read address
s_axi(b)_arregion[3:0]	I	Slave read region decode
s_axi(b)_arlen[7:0]	I	Slave read burst length
s_axi(b)_arsize[2:0]	I	Slave read burst size
s_axi(b)_arburst[1:0]	I	Slave read burst type
s_axi(b)_arvalid	I	Slave read address valid
s_axi(b)_arready	O	Slave read address ready
s_axi(b)_rid[c_s_axi_id_width-1:0]	O	Slave read ID tag
s_axi(b)_rdata[axi_data_width-1:0]	O	Slave read data
s_axi(b)_rresp[1:0]	O	Slave read response
s_axi(b)_rlast	O	Slave read last
s_axi(b)_rvalid	O	Slave read valid
s_axi(b)_rready	I	Slave read ready
s_axi(b)_ruser	O	Slave Read ECC Parity.  <b>Note:</b> This signal is disabled for AXI Bridge for PCI Express Gen3. This signal is also disabled for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode unless CONFIG.parity_settings = Propagate_Parity is set.

## AXI Master Interface

For all signals in this interface, `m_axi_*` is used for the AXI Bridge for PCIe Gen3 core, and `m_axib_*` is used for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode.

Table 11: AXI Master Interface Signals

Signal Name	I/O	Description
m_axi(b)_awaddr[axi_addr_width-1:0]	O	Master write address
m_axi(b)_awlen[7:0]	O	Master write burst length
m_axi(b)_awsize[2:0]	O	<p>Master write burst size</p> <p>All write requests show the requested size is equal to the Master AXI data width except for 1DW request. 1DW request is supported as a special case. As an example, if the IP is configured for 128-bit Master AXI data width, m_axi(b)_awsize[2:0] will show '2' for 1DW request and '4' for 2DW request instead of '3'.</p> <p>Write data must be further masked with m_axi(b)_wstrb signal which will indicate valid byte lanes to avoid writing to disabled byte lanes.</p> <p>The IP doesn't support narrow burst on Master Interface. The above case of 'narrow transfer' is only for single beat transaction.</p>
m_axi(b)_awburst[1:0]	O	Master write burst type
m_axi(b)_awprot[2:0]	O	Master write protection type
m_axi(b)_awvalid	O	Master write address valid
m_axi(b)_awready	I	Master write address ready
m_axi(b)_wdata[axi_data_width-1:0]	O	Master write data
m_axi(b)_wstrb[axi_data_width/8-1:0]	O	Master write strobe
m_axi(b)_wlast	O	Master write last
m_axi(b)_wvalid	O	Master write valid
m_axi(b)_wready	I	Master write ready
m_axi(b)_wuser	O	<p>Reserved. Internally tied to GND.</p> <p><b>Note:</b> This signal is disabled for AXI Bridge for PCI Express Gen3. This signal is also disabled for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode unless CONFIG.parity_settings = Propagate_Parity is set.</p>
m_axi(b)_bresp[1:0]	I	Master write response
m_axi(b)_bvalid	I	Master write response valid
m_axi(b)_bready	O	Master response ready
m_axi(b)_araddr[axi_addr_width-1:0]	O	Master read address
m_axi(b)_arlen[7:0]	O	Master read burst length
m_axi(b)_arsize[2:0]	O	<p>Master read burst size</p> <p>All read requests show the requested size is equal to the Master AXI data width except for 1DW request. 1DW request is supported as a special case. As an example, if the IP is configured for 128-bit Master AXI data width, this signal will show '2' for 1DW request and '4' for 2DW request instead of '3'. This behavior can be ignored if the request is not doing destructive reads; precaution must be taken for destructive register reads.</p> <p>The IP does not support narrow burst on Master Interface. The above case of narrow transfer is only for single beat transaction.</p>

Table 11: AXI Master Interface Signals (cont'd)

Signal Name	I/O	Description
m_axi(b)_arburst[1:0]	O	Master read burst type
m_axi(b)_arprot[2:0]	O	Master read protection type
m_axi(b)_arvalid	O	Master read address valid
m_axi(b)_arready	I	Master read address ready. This signal only responds when Bus Master Enable bit is set in the Command register within the PCI™ Configuration Space.
m_axi(b)_rdata[axi_data_width-1:0]	I	Master read data
m_axi(b)_rresp[1:0]	I	Master read response
m_axi(b)_rlast	I	Master read last
m_axi(b)_rvalid	I	Master read valid
m_axi(b)_rready	O	Master read ready
m_axi(b)_ruser	I	Reserved. Tie to GND.  <b>Note:</b> This signal is disabled for AXI Bridge for PCI Express Gen3. This signal is also disabled for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode unless CONFIG.parity_settings = Propagate_Parity is set.

## AXI4-Lite Control Interface

For all signals in this interface, s\_axi\_ctl\* is used for the AXI Bridge for PCIe Gen3 core, and s\_axil\_\* is used for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode.

Table 12: AXI4-Lite Control Interface Signals

Signal Name	I/O	Description
Endpoint configuration: s_axi_ctl_awaddr[11:0]   s_axil_awaddr[31:0] Root Port configuration: s_axi_ctl_awaddr[27:0]   s_axil_awaddr[31:0]	I	Slave write address. For DMA/Bridge Subsystem for PCIe in AXI Bridge mode, address must always be 512 MB aligned. s_axil_awaddr[28] = 1'b0 selects Bridge Register Memory Map and Enhanced Configuration Access Memory Map. s_axil_awaddr[28] = 1'b1 selects DMA/Bridge Subsystem for PCIe Register Memory Map. For AXI Bridge for PCIe Gen3 core, address must be aligned to 4 KB in Endpoint configuration and 256 MB in Root Port configuration.
s_axi_ctl_awvalid   s_axil_awvalid	I	Slave write address valid
s_axi_ctl_awready   s_axil_awready	O	Slave write address ready
s_axi_ctl_wdata[31:0]   s_axil_wdata[31:0]	I	Slave write data
s_axi_ctl_wstrb[3:0]   s_axil_wstrb[3:0]	I	Slave write strobe
s_axi_ctl_wvalid   s_axil_wvalid	I	Slave write valid
s_axi_ctl_wready   s_axil_wready	O	Slave write ready

Table 12: AXI4-Lite Control Interface Signals (cont'd)

Signal Name	I/O	Description
s_axi_ctl_bresp[1:0]   s_axil_bresp[1:0]	O	Slave write response
s_axi_ctl_bvalid   s_axil_bvalid	O	Slave write response valid
s_axi_ctl_bready   s_axil_bready	I	Slave response ready
Endpoint configuration: s_axi_ctl_araddr[11:0]   s_axil_araddr[31:0] Root Port configuration: s_axi_ctl_araddr[27:0]   s_axil_araddr[31:0]	I	Slave read address. For DMA/Bridge Subsystem for PCIe in AXI Bridge mode, the address must always be 512 MB aligned. s_axil_araddr[28] = 'b0 selects Bridge Register Memory Map and Enhanced Configuration Access Memory Map s_axil_araddr[28] = 'b1 selects DMA/Bridge Subsystem for PCIe Register Memory Map For AXI Bridge for PCIe Gen3 core, address must be aligned to 4 KB in Endpoint configuration and 256 MB in Root Port configuration.
s_axi_ctl_arvalid   s_axil_arvalid	I	Slave read address valid
s_axi_ctl_arready   s_axil_arvalid	O	Slave read address ready
s_axi_ctl_rdata[31:0]   s_axil_arvalid	O	Slave read data
s_axi_ctl_rresp[1:0]   s_axil_arvalid	O	Slave read response
s_axi_ctl_rvalid   s_axil_arvalid	O	Slave read valid
s_axi_ctl_rready   s_axil_arvalid	I	Slave read ready

## AXI Bridge for PCIe Gen3 MSI Signals

Table 13: AXI Bridge for PCIe Gen3 MSI Signals

Signal Name	I/O	Description
intx_msi_request	I	Legacy interrupt input (see pf0_interrupt_pin) when msi_enable = 0. Initiates a MSI write request when msi_enable = 1. intx_msi_request is asserted for one clock period.
intx_msi_grant	O	Indicates legacy interrupt/MSI grant signal. The intx_msi_grant signal is asserted for one clock period when the interrupt is accepted by the PCIe core.
msi_enable	O	Indicates when MSI is enabled.
msi_vector_num [4:0]	I	Indicates MSI vector to send when writing a MSI write request.
msi_vector_width [2:0]	O	Indicates the size of the MSI field (the number of MSI vectors allocated to the device).

## AXI Bridge for PCIe Gen3 MSI-X Signals

Table 14: AXI Bridge for PCIe Gen3 MSI-X Signals

Signal Name	I/O	Description
cfg_interrupt_msix_enable	O	Configuration Interrupt MSI-X Function Enabled. When asserted, indicates that the Message Signaling Interrupt (MSI-X) messaging is enabled, per function.
cfg_interrupt_msix_mask	O	Configuration Interrupt MSI-X Function Mask. Indicates the state of the Function Mask bit in the MSI-X Message Control field. This is a 4 bit signal but only bit 0 is used for this.
cfg_interrupt_msix_address	I	Configuration Interrupt MSI-X Address. When the core is configured to support MSI-X interrupts, this bus is used by the user logic to communicate the address to be used for an MSI-X message.
cfg_interrupt_msix_data	I	Configuration Interrupt MSI-X Data. When the core is configured to support MSI-X interrupts, this bus is used by the user logic to communicate the data to be used for an MSI-X message.
cfg_interrupt_msix_int	I	Configuration Interrupt MSI-X Data Valid. This signal indicates that valid information has been placed on the <code>cfg_interrupt_msix_address[63:0]</code> and <code>cfg_interrupt_msix_data[31:0]</code> buses, and the originating function number has been placed on <code>cfg_interrupt_msi_function_number[3:0]</code> . The core internally registers the associated address and data from <code>cfg_interrupt_msix_address</code> and <code>cfg_interrupt_msix_data</code> on the 0-to-1 transition of this valid signal. The user application must ensure that the <code>cfg_interrupt_msix_enable</code> bit corresponding to function in use is set before asserting <code>cfg_interrupt_msix_int</code> . After asserting an interrupt, the user logic must wait for the <code>cfg_interrupt_msix_sent</code> or <code>cfg_interrupt_msix_fail</code> indication from the core before asserting a new interrupt.
cfg_interrupt_msix_sent	O	Configuration Interrupt MSI-X Interrupt Sent. The core generates a one-cycle pulse on this output to indicate that it has accepted the information placed on the <code>cfg_interrupt_msix_address[63:0]</code> and <code>cfg_interrupt_msix_data[31:0]</code> buses, and an MSI-X interrupt message has been transmitted on the link. The user application must wait for this pulse before signaling another interrupt condition to the core.
cfg_interrupt_msix_fail	O	Configuration Interrupt MSI-X Interrupt Operation Failed. A one-cycle pulse on this output indicates that the interrupt controller has failed to transmit MSI-X interrupt on the link. The user application must retransmit the MSI-X interrupt in this case.

## DMA/Bridge Subsystem for PCIe in Bridge Mode Interrupt Signals

The interface signals for the Bridge are described in the following table.

*Table 15: DMA/Bridge Subsystem for PCIe in Bridge Mode Interrupt Signals*

Signal Name	I/O	Description
usr_irq_req[NUM_USR_IRQ-1:0]	I	Assert to generate an interrupt. Maintain assertion until interrupt is serviced.
usr_irq_ack[NUM_USR_IRQ-1:0]	O	Indicates that the interrupt has been sent on PCIe. Two acks are generated for legacy interrupts. One ack is generated for MSI interrupts.
msi_enable	O	Indicates when MSI is enabled.
msix_enable	O	Indicates when MSI-X is enabled.

## PCIe Interface

*Table 16: PCIe Interface Signals*

Signal Name	I/O	Description
pci_exp_rxp[PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	I	PCIe RX serial interface
pci_exp_rxn[PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	I	PCIe RX serial interface
pci_exp_txp[PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	O	PCIe TX serial interface
pci_exp_txn[PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	O	PCIe TX serial interface

## Bridge Parameters

Because many features in the Bridge core design can be parameterized, you can uniquely tailor the implementation of the core using only the resources required for the desired functionality. This approach also achieves the best possible performance with the lowest resource usage.

The parameters defined for the Bridge are shown in the following table.

Table 17: Top-Level Parameters

Generic	Parameter Name	Description	Allowable Values	Default Value
<b>Bridge Parameters</b>				
	PCIE_BLK_LOCN	PCIe® integrated block location within FPGA	0: X0Y0 1: X0Y1 2: X0Y2 3: X0Y3 4: X0Y4 <sup>1</sup> 5: X0Y5 <sup>1</sup>	0
	PL_UPSTREAM_FACING	Configures the AXI bridge for PCIe to be a Root Port or an Endpoint	TRUE: Endpoint FALSE: Root Port	TRUE
G3	C_COMP_TIMEOUT	Selects the slave bridge completion timeout counter value. This parameter is now deprecated. Completion timeout is now maintained by the PCIe IP.	0: 50 µs 1: 50 ms	0
G6	C_AXIBAR_NUM	Number of AXI address apertures that can be accessed	1: BAR_0 enabled 2: BAR_0, BAR_1 enabled 3: BAR_0, BAR_1, BAR_2 enabled 4: BAR_0 through BAR_3 enabled 5: BAR_0 through BAR_4 enabled 6: BAR_0 through BAR_5 enabled	6
G7	C_AXIBAR_0	AXI BAR_0 aperture low address	Valid AXI address <sup>34</sup>	0x00000000_00000000
G8	C_AXIBAR_HIGHADDR_0	AXI BAR_0 aperture high address	Valid AXI address <sup>34</sup>	0x00000000_00000000
G10	C_AXIBAR2PCIEBAR_0	Initial address translation from an AXI BAR_0 address to a PCI Express address	Valid address for PCIe <sup>2</sup>	0x00000000_00000000
G11	C_AXIBAR_1	AXI BAR_1 aperture low address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G12	C_AXIBAR_HIGHADDR_1	AXI BAR_1 aperture high address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G14	C_AXIBAR2PCIEBAR_1	Initial address translation from an AXI BAR_1 address to a PCI Express address	Valid address for PCIe <sup>2</sup>	0x00000000_00000000
G15	C_AXIBAR_2	AXI BAR_2 aperture low address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G16	C_AXIBAR_HIGHADDR_2	AXI BAR_2 aperture high address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
G18	C_AXIBAR2PCIEBAR_2	Initial address translation from an AXI BAR_2 address to a PCI Express address	Valid address for PCIe <sup>2</sup>	0x00000000_00000000
G19	C_AXIBAR_3	AXI BAR_3 aperture low address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G20	C_AXIBAR_HIGHADDR_3	AXI BAR_3 aperture high address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G22	C_AXIBAR2PCIEBAR_3	Initial address translation from an AXI BAR_3 address to a PCI Express address	Valid address for PCIe <sup>2</sup>	0x00000000_00000000
G23	C_AXIBAR_4	AXI BAR_4 aperture low address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G24	C_AXIBAR_HIGHADDR_4	AXI BAR_4 aperture high address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G26	C_AXIBAR2PCIEBAR_4	Initial address translation from an AXI BAR_4 address to a PCI Express address	Valid address for PCIe <sup>2</sup>	0x00000000_00000000
G27	C_AXIBAR_5	AXI BAR_5 aperture low address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G28	C_AXIBAR_HIGHADDR_5	AXI BAR_5 aperture high address	Valid AXI address <sup>3, 4</sup>	0x00000000_00000000
G30	C_AXIBAR2PCIEBAR_5	Initial address translation from an AXI BAR_5 address to a PCI Express address	Valid address for PCIe <sup>2</sup>	0x00000000_00000000
G31	PCIEBAR_NUM	Number of address for PCIe apertures that can be accessed	1: BAR_0 enabled 2: BAR_[0:1] enabled or BAR_1 as 64-bit 3: BAR_[0-2] enabled 4: BAR_[0-3] enabled or BAR_2 as 64-bit 5: BAR_[0-4] enabled or 6: BAR_[0-5] enabled or BAR_4 as 64-bit	3
G33	PF0_BAR0_APERTURE_SIZE	Specifies the size of the PCIe BAR	0x05: 4 Kilobytes 0x06: 8 Kilobytes ... 0x0C: 512 Kilobytes 0x0D: 1 Megabyte ... 0x16: 512 Megabytes 0x17: 1 Gigabyte ... 0x1F: 256 Gigabytes	0x05



Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
	PF0_BAR0_CONTROL	PCI Express control settings for BAR0	bit 0: 32-bit addressable = 0 64-bit addressable = 1 bit 1: Prefetchable off = 0 Prefetchable on = 1 bit 2: IO BAR = 0 Memory bar = 1	100
G34	C_PCIEBAR2AXIBAR_0	Initial address translation from an AXI BAR_0 address to a PCI Express address	Valid AXI address	0x00000000_00000000
G35	PF0_BAR1_APERTURE_SIZE	Specifies the size of the PCIe BAR.	0x05: 4 Kilobytes 0x06: 8 Kilobytes ... 0x0C: 512 Kilobytes 0x0D: 1 Megabyte ... 0x16: 512 Megabytes 0x17: 1 Gigabyte ... 0x1F: 256 Gigabytes	0x05
	PF0_BAR1_CONTROL	PCI Express control settings for BAR1	bit 0: 32-bit addressable = 0 64-bit addressable = 1 bit 1: Prefetchable off = 0 Prefetchable on = 1 bit 2: IO bar = 0 Memory bar = 1	100
G36	C_PCIEBAR2AXIBAR_1	Initial address translation from an AXI BAR_1 address to a PCI Express address	Valid AXI address	0x00000000_00000000
G37	PF0_BAR2_APERTURE_SIZE	Specifies the size of the PCIe BAR.	0x05: 4 Kilobytes 0x06: 8 Kilobytes ... 0x0C: 512 Kilobytes 0x0D: 1 Megabyte ... 0x16: 512 Megabytes 0x17: 1 Gigabyte ... 0x1F: 256 Gigabytes	0x05

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
	PF0_BAR2_CONTROL	PCI Express control settings for BAR2	bit 0: 32-bit addressable = 0 64-bit addressable = 1 bit 1: Prefetchable off = 0 Prefetchable on = 1 bit 2: IO bar = 0 Memory bar = 1	100
G38	C_PCIEBAR2AXIBAR_2	Initial address translation from an AXI BAR_2 address to a PCI Express address.	Valid AXI address	0x00000000_00000000
	PF0_BAR3_APERTURE_SIZE	Specifies the size of the PCIe BAR.	0x05: 4 Kilobytes 0x06: 8 Kilobytes ... 0x0C: 512 Kilobytes 0x0D: 1 Megabyte ... 0x16: 512 Megabytes 0x17: 1 Gigabyte ... 0x1F: 256 Gigabytes	0x05
	PF0_BAR3_CONTROL	PCI Express control settings for BAR3	bit 0: 32-bit addressable = 0 64-bit addressable = 1 bit 1: Prefetchable off = 0 Prefetchable on = 1 bit 2: IO bar = 0 Memory bar = 1	100
	C_PCIEBAR2AXIBAR_3	Initial address translation from an AXI BAR_3 address to a PPCI Express address.	Valid AXI address	0x00000000_00000000
	PF0_BAR4_APERTURE_SIZE	Specifies the size of the PCIe BAR.	0x05: 4 Kilobytes 0x06: 8 Kilobytes ... 0x0C: 512 Kilobytes 0x0D: 1 Megabyte ... 0x16: 512 Megabytes 0x17: 1 Gigabyte ... 0x1F: 256 Gigabytes	0x05

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
	PF0_BAR4_CONTROL	PCI Express control settings for BAR4	bit 0: 32-bit addressable = 0 64-bit addressable = 1 bit 1: Prefetchable off = 0 Prefetchable on = 1 bit 2: IO bar = 0 Memory bar = 1	100
	C_PCIEBAR2AXIBAR_4	Initial address translation from an AXI BAR_4 address to a PCI Express address.	Valid AXI address	0x00000000_00000000
	PF0_BAR5_APERTURE_SIZE	Specifies the size of the PCIe BAR.	0x05: 4 Kilobytes 0x06: 8 Kilobytes ... 0x0C: 512 Kilobytes 0x0D: 1 Megabyte ... 0x16: 512 Megabytes 0x17: 1 Gigabyte ... 0x1F: 256 Gigabytes	0x05
	PF0_BAR5_CONTROL	PCI Express control settings for BAR5	bit 0: 32-bit addressable = 0 64-bit addressable = 1 bit 1: Prefetchable off = 0 Prefetchable on = 1 bit 2: IO bar = 0 Memory bar = 1	100
	C_PCIEBAR2AXIBAR_5	Initial address translation from an AXI BAR_5 address to a PCI Express address.	Valid AXI address	0x00000000_00000000
	C_MSI_RX_PIN_EN	DMA/Bridge Subsystem for PCIe in AXI Bridge Mode only. Selects Legacy Interrupt FIFO mode or Interrupt Decode mode for Root Port configuration.	0: Legacy Interrupt FIFO mode 1: Interrupt Decode mode	0
	INTERRUPT_OUT_WIDTH	DMA/Bridge Subsystem for PCIe in AXI Bridge Mode only. Interrupt_out signal bus width.	1: 1-bit for Legacy Interrupt FIFO mode or Endpoint configuration 3: 3-bits for Interrupt Decode mode	1

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
	SOFT_RESET_EN	DMA/Bridge Subsystem for PCIe in AXI Bridge Mode only. Enables the <code>dma_bridge_resetn</code> pin which allow user reset of all internal Bridge engines and registers as well as all peripherals in AXI domain while maintaining PCIe link up.	TRUE: Enables <code>dma_bridge_resetn</code> pin FALSE: Disables <code>dma_bridge_resetn</code> pin	FALSE
<b>AXI4-Lite Parameters</b>				
G39	C_BASEADDR	Device base address AXI Bridge for PCIe Gen3: When configured as an Root Port (RP), the minimum alignment granularity must be 256MB. Bit [27:0] is used for Bus Number, Device Number, Function number. For Endpoint, the minimum alignment granularity is 4KB. C_BASEADDR is deprecated for DMA/Bridge Subsystem for PCIe in AXI Bridge Mode.	Valid AXI address	0xFFFF_FFFF
G40	C_HIGHADDR	Device high address	Valid AXI address	0x0000_0000
<b>PCIe Core Configuration Parameters</b>				
G41	PL_LINK_CAP_MAX_LINK_WIDTH	Number of PCIe Lanes	1, 2, 4, 8	1
G42	PF0_DEVICE_ID	Device ID	16-bit vector	0x0000
G43	PF0_VENDOR_ID	Vendor ID	16-bit vector	0x0000
G44	PF0_CLASS_CODE	Class Code	24-bit vector	0x00_0000
G45	PF0_REVISION_ID	Rev ID	8-bit vector	0x00
G46	C_SUBSYSTEM_ID	Subsystem ID	16-bit vector	0x0000
G47	C_SUBSYSTEM_VENDOR_ID	Subsystem Vendor ID	16-bit vector	0x0000
G49	REF_CLK_FREQ	Reference Clock input frequency. <code>refclk</code> for Virtex®-7 devices. <code>sys_clk_gt</code> for UltraScale devices.	0: 100 MHz 1: 125 MHz 2: 250 MHz	0
G55	PL_LINK_CAP_MAX_LINK_SPEED	Maximum PCIe link speed supported	0: 2.5 GT/s 1: 5.0 GT/s 4: 8.0 GT/s	0

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
	TL_LEGACY_MODE_ENABLE	Selects between Legacy PCIe Endpoint Device and regular PCIe Endpoint Device	TRUE: Legacy PCIe Endpoint Device FALSE: PCIe Endpoint Device	FALSE
	CORE_CLK_FREQ	Core Clock Frequency. See CORE CLOCK Frequency for valid configurations	1: 250 MHz 2: 500 MHz	2
	PLL_TYPE	Specifies PLL being used.	0: CPLL 1: QPLL0 2: QPLL1	0
	USER_CLK_FREQ	AXI Clock Frequency	1: 62.5 MHz 2: 125 MHz 3: 250 MHz	1
	PIPE_SIM	PIPE Simulation Enable/Disable	TRUE: Enable PIPE Simulation FALSE: Disable PIPE Simulation	FALSE
	EXT_CH_GT_DRP	Enable/Disable Transceiver DRP Ports	TRUE: Enable Transceiver Block DRP Ports FALSE: Disable Transceiver Block DRP Ports	FALSE
	PCIE3_DRP	Enable/Disable PCIe DRP Ports	TRUE: Enable PCIe Block DRP Ports FALSE: Disable PCIe Block DRP Ports	FALSE
	DEDICATE_PERST	Use the dedicated PERST routing resources	TRUE: Use dedicated PERST signal routing FALSE: Use fabric routing for PERST signal	TRUE
	SYS_RESET_POLARITY	System Reset Polarity. An Active-Low reset should be selected for designs utilizing PCIe edge connector	0: Active-Low 1: Active-High	0
	EN_TRANSCEIVER_STATUS_PORTS	Enable Additional Transceiver Control and Status Ports	False: Do not add Transceiver debug ports TRUE: Add Transceiver debug ports	FALSE
	MSI_ENABLED	Enable/Disable MSI support	TRUE: Enable MSI-Support FALSE: Disable MSI Support	TRUE
	DEV_PORT_TYPE	Selects PCI Express Device Type	0: PCI Express Endpoint Device 1: Legacy PCI Express Endpoint Device	0
	MSIX_EN	Enable/Disable MSI-X support	TRUE: Enable MSI-X Support FALSE: Disable MSI-X Support	FALSE

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
	pf0_msix_cap_pba_bir	Value of Pending Bit Array BAR Indicator Register. Indicates which BAR is used for MSI-X Pending Bit Array.	0: BAR0 1: BAR1 2: BAR2 3: BAR3 4: BAR4 5: BAR5	0
	pf0_msix_cap_pba_offset	Value of Pending Bit Array Offset Register. Indicates the address offset within the BAR where MSI-X Pending Bit Array starts.	'h0000_0000 to 'h1FFF_FFFF	'h0000_0000
	pf0_msix_cap_table_bir	Value of MSI-X Table BAR Indicator Register. Indicates which BAR is used for MSI-X table.	0: BAR0 1: BAR1 2: BAR2 3: BAR3 4: BAR4 5: BAR5	0
	pf0_msix_cap_table_offset	Value of MSI-X Table Offset Register. Indicates the address offset within the BAR where MSI-X table starts.	'h0000_0000 to 'h1FFF_FFFF	'h0000_0000
	pf0_msix_cap_table_size	Value of MSI-X Table Size Register. Indicates the size of MSI-X table.	'h000 to 'h7FF	'h000
<b>Memory Mapped AXI4 Parameters</b>				
G50	AXI_DATA_WIDTH	AXI Bus Data width	64 128 256	64
G51	AXI_ADDR_WIDTH	AXI Bus Address width	32-64	32
G52	C_S_AXI_ID_WIDTH	AXI Slave Bus ID width	4	4
G56	PF0_INTERRUPT_PIN	Legacy INTX pin support/select	0: No INTX support (setting for Root Port). 1: INTA selected 2: INTB selected 3: INTC selected 4: INTD selected 1,2,3,4: (only selectable when core in Endpoint configuration).	0
	C_S_AXI_SUPPORTS_NARROW_BURST	Enable/Disable Narrow Burst support on S_AXI interface.	0: Narrow Burst support disabled 1: Narrow Burst support enabled	0

Table 17: Top-Level Parameters (cont'd)

Generic	Parameter Name	Description	Allowable Values	Default Value
<b>AXI4 Slave Interconnect Parameters<sup>5</sup></b>				
G57	C_S_AXI_NUM_WRITE	AXI Interconnect Slave Port Write Pipeline Depth	8 Size of pipeline for active AXI AWADDR values to be stored in AXI slave bridge PCIe.	8
G58	C_S_AXI_NUM_READ	AXI Interconnect Slave Port Read Pipeline Depth	8 Size of pipeline for active AXI ARADDR values to be stored in AXI slave bridge PCIe.	8
	EN_AXI_SLAVE_IF	AXI4 Slave Bus Enable/Disable	TRUE: AXI4 Slave Bus is active FALSE: AXI4 Slave Bus is disabled	TRUE
<b>AXI4 Master Interconnect Parameters<sup>5</sup></b>				
G59	C_M_AXI_NUM_WRITE	AXI Interconnect master bridge write address issue depth	8 Number of actively issued AXI AWADDR values on the AXI Interconnect to the target slave device(s).	8
G60	C_M_AXI_NUM_READ	AXI Interconnect master bridge read address issue depth	8 Number of actively issued AXI ARADDR values on the AXI Interconnect to the target slave device(s).	8
	EN_AXI_MASTER_IF	AXI4 Master Bus Enable/Disable	TRUE: AXI4 Master Bus is active FALSE: AXI4 Master Bus is disabled	TRUE

**Notes:**

1. X0Y4 and X0Y5 are only available on select UltraScale devices.
2. The width of this should match the address width of the PCIe BARs.
3. The range specified must comprise a complete, contiguous power of two range, such that the range =  $2^n$  and the  $n$  least significant bits of the Base Address are zero.
4. The difference between C\_AXIBAR\_n and C\_AXIBAR\_HIGHADDR\_n must be greater than or equal to 0x00000000\_00000FFF.
5. These are the user parameters of the AXI4 Interconnect. By default, the slave bridge handles up to two AXI4 write requests and eight AXI4 read requests. The master bridge handles up to four PCIe write/read requests.

## Memory Map

There are three distinct register spaces, which are described in this section:

- Bridge Register Memory Map

- DMA/Bridge Subsystem for PCIe® Register Memory Map
- Enhanced Configuration Access Memory Map

In AXI Bridge for PCI Express Gen3 IP, only Bridge Register Memory Map and Enhanced Configuration Access Memory Map are used. In DMA/Bridge Subsystem for PCI Express® (in AXI Bridge mode), all three register spaces are used. These registers are described in more detail in the following section. During reset, all registers return to default values.

In AXI Bridge for PCI Express Gen3 IP, all registers are accessed through the AXI4-Lite Control Interface and are offset from the AXI Base Address assigned to this interface (`C_BASEADDR` parameter).

In DMA/Bridge Subsystem for PCI Express in AXI Bridge mode, all registers are accessed through the AXI4-Lite Control Interface and are offset from the AXI base address assigned to this interface. `C_BASEADDR` parameter has been deprecated in this particular IP and the IP is no longer doing address filtering for the AXI4-Lite Control interface. However, if AXI Interconnect IP is used in the design, AXI Crossbar IP within it will do address filtering based on the AXI Address assigned to each interface. The DMA/Bridge Subsystem for PCI Express in AXI Bridge mode uses bit 28 of the address bus to select between each register space:

- If the AXI4-Lite Slave address bit 28 is set to 0, access is directed to the Bridge Register Memory Map and Enhanced Configuration Access Memory Map.
- If the AXI4-Lite Slave address bit 28 is set to 1, access is directed to the DMA/Bridge Subsystem for PCIe Register Memory Map.

**Note:** Registers that are marked as Reserved may appear writable or have a value other than 0. If your application does data compare after the registers are being written, ensure that it ignores the value being returned on the reserved bit position.

Configuration register attributes are defined in the following table.

**Table 18: Configuration Register Attribute Definitions**

Register Attribute	Description
RO	Read-Only: Register bits are read-only and cannot be modified by the software.
RW	Read-Write: Register bits are read-write and are permitted to be either Set or Cleared by the software to the desired state.
RW1C	Write-1-to-clear-status: Register bits indicate status when read. A Set bit indicates a status event which is Cleared by writing a 1b. Writing a 0b to RW1C bits has no effect.
RWC	Write-any-to-clear-status: Register bits indicate status when read. Write of any value clears that entire register set.
W1C	Non-readable-write-1-to-clear-status: Register returns 0 when read. Writing 1b clears the status for that bit index. Writing a 0b to W1C bits has no effect.



Table 18: Configuration Register Attribute Definitions (cont'd)

Register Attribute	Description
W1S	Non-readable-write-1-to-set: Register returns 0 when read. Writing 1b sets the control set for that bit index. Writing a 0b to W1S bits has no effect.

## Bridge Register Memory Map

Table 19: Bridge Register Memory Map

Accessibility	Offset	Contents	Location
RO - EP	0x000 -0xFF	<p>PCIe® Configuration Space Header</p> <p>For more details of the register layout for addresses in this address range:</p> <p>For Virtex-7 XT, see <i>Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide (PG023)</i></p> <p>For UltraScale, see <i>UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)</i></p> <p>For UltraScale+, see <i>UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)</i></p>	Part of the integrated PCIe® configuration space.
RO - EP	0x100-0x124	<p>PCIe® Extended Configuration Space Header</p> <p>For more details of the register layout for addresses in this address range:</p> <p>For Virtex-7 XT, see <i>Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide (PG023)</i></p> <p>For UltraScale, see <i>UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)</i></p> <p>For UltraScale+, see <i>UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)</i></p>	Part of the integrated PCIe® configuration space.

Table 19: Bridge Register Memory Map (cont'd)

Accessibility	Offset	Contents	Location
RO	0x128	PCIe® Vendor Specific Extended Capability Header for Bridge register	AXI bridge defined memory-mapped register space.
RO	0x12C	PCIe® Vendor Specific Header for Bridge register	
RO	0x130	Bridge Info	
RO - EP	0x134	Bridge Status and Control	
R/W	0x138	Interrupt Decode	
R/W	0x13C	Interrupt Mask	
RO - EP	0x140	Bus Location	
RO	0x144	Physical-Side Interface (PHY) Status/Control	
RO - EP, R/W - RC	0x148	Root Port Status/Control	
RO - EP, R/W - RC	0x14C	Root Port MSI Base 1	
RO - EP, R/W - RC	0x150	Root Port MSI Base 2	
RO - EP, R/W - RC	0x154	Root Port Error FIFO Read	
RO - EP, R/W - RC	0x158	Root Port Interrupt FIFO Read 1	
RO - EP, R/W - RC	0x15C	Root Port Interrupt FIFO Read 2	

Table 19: Bridge Register Memory Map (cont'd)

Accessibility	Offset	Contents	Location
RO	0x160	Reserved for AXI PCIe® Bridge. Interrupt Decode 2 for the DMA/Bridge Subsystem for PCI Express, when configured in Bridge mode (UltraScale+™).	AXI bridge defined memory-mapped register space.
RO	0x164	Reserved for AXI PCIe® Bridge. Interrupt Decode 2 for the DMA/Bridge Subsystem for PCI Express, when configured in Bridge mode (UltraScale+™).	
R/W	0x168	Configuration Control	
R/W	0x170	Reserved for AXI PCIe 3 Bridge. Root Port MSI Interrupt Decode 1 Register for the DMA/Bridge Subsystem for PCI Express core when configured in Bridge mode (UltraScale+).	
R/W	0x174	Reserved for AXI PCIe® 3 Bridge. Root Port MSI Interrupt Decode 2 Register for the DMA/Bridge Subsystem for PCI Express, when configured in Bridge mode (UltraScale+).	
R/W	0x178	Reserved for AXI PCIe® 3 Bridge. Root Port MSI Interrupt Decode 1 Mask Register for the DMA/Bridge Subsystem for PCI Express, when configured in Bridge mode core (UltraScale+).	
R/W	0x17C	Reserved for AXI PCIe® 3 Bridge. Root Port MSI Interrupt Decode 2 Mask Register for the DMA/Bridge Subsystem for PCI Express, when configured in Bridge mode (UltraScale+).	
RO	0x180 - 0x1FF	Reserved (zeros returned on read)	
RO	0x200	VSEC Capability 2	
RO	0x204	VSEC Header 2	
R/W	0x208 - 0x234	AXI Base Address Translation Configuration Registers	
RO	0x238 - 0xFFF	Reserved (zeros returned on read)	

The Bridge register is mapped within the PCIe® Extended Configuration Space Header memory region. The AXI Bridge VSEC registers are only accessible from the Bridge IP AXI4-Lite Control Interface, therefore at the start of the PCIe® Extended Configuration Space (offset 0x100), the capability header will point to address 0x128 as the next capability pointer. When the PCIe® Extended Configuration Space is accessed from the PCIe link, the next capability pointer field will point to the next enabled PCIe® capability instead of the AXI Bridge VSEC register.

## PCIe Configuration Space Header

The PCI™ Configuration Space Header is a memory aperture for accessing the core for PCIe configuration space. This area is read-only when configured as an Endpoint. Writes are permitted for some registers when configured as a Root Port. Special access modes can be enabled using the PHY Status/Control register. All reserved or undefined memory-mapped addresses must return zero and writes have no effect.

## Bridge Info Register (Offset 0x130)

The Bridge Info register (described in the following table) provides general configuration information about the AXI4-Stream Bridge. Information in this register is static and does not change during operation.

Table 20: Bridge Info Register

Bits	Name	Core Access	Reset Value	Description
0	Gen2 Capable	RO	0	If set, underlying integrated block supports PCIe® Gen2 speed.
1	Root Port Present	RO	0	Indicates the underlying integrated block is a Root Port when this bit is set. If set, Root Port registers are present in this interface.
2	Reserved	RO	0	Reserved
3	Gen3 Capable	RO	0	If set, underlying integrated block supports PCIe Gen3 speed.
31:4	Reserved	RO	0	Reserved

## Bridge Status and Control Register (Offset 0x134)

The Bridge Status and Control register (described in the following table) provides information about the current state of the AXI4-Stream Bridge. It also provides control over how reads and writes to the Core Configuration Access aperture are handled.

Table 21: Bridge Status and Control Register

Bits	Name	Core Access	Reset Value	Description
7:0	Reserved	RO	0	Reserved
8	Global Disable	RW	0	When set, disables interrupt line from being asserted. Does not prevent bits in Interrupt Decode register from being set.
31:9	Reserved	RO	0	Reserved

## Interrupt Decode Register (Offset 0x138)

The Interrupt Decode register (described in the following table) provides a single location where the host processor interrupt service routine can determine what is causing the interrupt to be asserted and how to clear the interrupt. Writing a 1'b1 to any bit of the Interrupt Decode register clears that bit except for the Correctable, Non-Fatal, and Fatal bits.

Follow this sequence to clear the Correctable, Non-Fatal, and Fatal bits:

1. Clear the Root Port Error FIFO (0x154) by performing first a read, followed by write-back of the same register.
2. Read Root Port Status/Control Register (0x148) bit 16, and ensure that the Error FIFO is empty.  
**Note:** If the error FIFO is still not empty, repeat step 1 and step 2 until the Error FIFO is empty.
3. Write to the Interrupt Decode Register (0x138) with 1 to the appropriate error bit to clear it.



**IMPORTANT!** An asserted bit in the Interrupt Decode register does not cause the interrupt line to assert unless the corresponding bit in the Interrupt Mask register is also set.

Table 22: Interrupt Decode Register

Bits	Name	Core Access	Reset Value	Description
0	Link Down	RW1C	0	Indicates that Link-Up on the PCI Express link was lost. Not asserted unless link-up had previously been seen.
1	Reserved	RO	0	Reserved
2	Reserved	RO	0	Reserved
3	Hot Reset	RW1C	0	Indicates a Hot Reset was detected (Only as Endpoint).
4	Reserved	RO	0	Reserved
7:5	Cfg Completion Status	RW1C	0	Indicates config completion status.
8	Cfg Timeout	RW1C	0	Indicates timeout on an enhanced configuration access mechanism (ECAM) access. (Only applicable to Root Port cores.)
9	Correctable	RW1C	0	Indicates a correctable error message was received. Requester ID of error message should be read from the Root Port FIFO. (Only applicable to Root Port cores.)
10	Non-Fatal	RW1C	0	Indicates a non-fatal error message was received. Requester ID of error message should be read from the Root Port FIFO. (Only applicable to Root Port cores.)

Table 22: Interrupt Decode Register (cont'd)

Bits	Name	Core Access	Reset Value	Description
11	Fatal	RW1C	0	Indicates a fatal error message was received. Requester ID of error message should be read from the Root Port FIFO. (Only applicable to Root Port cores.)
15:12	Reserved	RO	0	Reserved
16	INTx Interrupt Received	RW1C	0	Indicates an INTx interrupt was received. Interrupt details should be read from the Root Port FIFO. (Only applicable to Root Port cores.)
17	MSI Interrupt Received	RW1C	0	Indicates an MSI(x) interrupt was received. Interrupt details should be read from the Root Port FIFO. (Only applicable to Root Port cores.)
19:18	Reserved	RO	0	Reserved
20	Slave Unsupported Request	RW1C	0	Indicates that a completion TLP was received with a status of 0b001 - Unsupported Request.
21	Slave Unexpected Completion	RW1C	0	Indicates that a completion TLP was received that was unexpected.
22	Reserved	RO	0	Reserved. This bit previously indicates Slave Completion Timeout in the Bridge IP. This functionality is now maintained by the PCIe IP. See Slave Bridge Abnormal Conditions Completion Timeout section for details. The read request that has timed out.
23	Slave Error Poison	RW1C	0	Indicates the error poison (EP) bit was set in a completion TLP.
24	Slave Completer Abort	RW1C	0	Indicates that a completion TLP was received with a status of 0b100 - Completer Abort.
25	Slave Illegal Burst	RW1C	0	Indicates that a burst type other than INCR was requested by the AXI master.
26	Master DECERR	RW1C	0	Indicates a Decoder Error (DECERR) response was received.
27	Master SLVERR	RW1C	0	Indicates a Slave Error (SLVERR) response was received.
28	Reserved	RO	0	Reserved
31:29	Reserved	RO	0	Reserved

## Interrupt Mask Register (Offset 0x13C)

The Interrupt Mask register controls whether each individual interrupt source can cause the interrupt line to be asserted. A one in any location allows the interrupt source to assert the interrupt line. The Interrupt Mask register initializes to all zeros. Therefore, by default no interrupt is generated for any event. The following table describes the Interrupt Mask register bits and values.

Table 23: Interrupt Mask Register

Bits	Name	Core Access	Reset Value	Description
0	Link Down	RW	0	Enables interrupts for Link Down events when bit is set.
1	Reserved	RO	0	Reserved
2	Reserved	RO	0	Reserved
3	Hot Reset	RW	0	Enables interrupts for Hot Reset events when bit is set. (Only writable for EP configurations, otherwise = 0)
4	Reserved	RO	0	Reserved
7:5	Cfg Completion Status	RW	0	Enables interrupts for config completion status. (Only writable for Root Port Configurations, otherwise = 0)
8	Cfg Timeout	RO	0	Enables interrupts for Config (Cfg) Timeout events when bit is set. (Only writable for Root Port Configurations, otherwise = 0)
9	Correctable	RO	0	Enables interrupts for Correctable Error events when bit is set. (Only writable for Root Port Configurations, otherwise = 0)
10	Non-Fatal	RO	0	Enables interrupts for Non-Fatal Error events when bit is set. (Only writable for Root Port Configurations, otherwise = 0)
11	Fatal	RO	0	Enables interrupts for Fatal Error events when bit is set. (Only writable for Root Port Configurations, otherwise = 0)
15:12	Reserved	RO	0	Reserved
16	INTx Interrupt Received	RO	0	Enables interrupts for INTx Interrupt events when bit is set. (Only writable for Root Port Configurations, otherwise = 0)
17	MSI Interrupt Received	RO	0	Enables interrupts for MSI Interrupt events when bit is set. (Only writable for Root Port Configurations, otherwise = 0)
19:18	Reserved	RO	0	Reserved
20	Slave Unsupported Request	RW	0	Enables the Slave Unsupported Request interrupt when bit is set.
21	Slave Unexpected Completion	RW	0	Enables the Slave Unexpected Completion interrupt when bit is set.
22	Reserved	RW	0	Reserved. This bit is previously mask Slave Completion Timeout in the Bridge IP. This functionality is now maintained by the PCIe IP.
23	Slave Error Poison	RW	0	Enables the Slave Error Poison interrupt when bit is set.
24	Slave Completer Abort	RW	0	Enables the Slave Completer Abort interrupt when bit is set.

Table 23: Interrupt Mask Register (cont'd)

Bits	Name	Core Access	Reset Value	Description
25	Slave Illegal Burst	RW	0	Enables the Slave Illegal Burst interrupt when bit is set.
26	Master DECERR	RW	0	Enables the Master DECERR interrupt when bit is set.
27	Master SLVERR	RW	0	Enables the Master SLVERR interrupt when bit is set.
28	Reserved	RO	0	Reserved
31:29	Reserved	RO	0	Reserved

## Bus Location Register (Offset 0x140)

The Bus Location register reports the Bus, Device, and Function number, and the Port number for the PCIe port (see the following table).

Table 24: Bus Location Register

Bits	Name	Core Access	Reset Value	Description
2:0	Function Number	RO	0	Function number of the port for PCIe. Hard-wired to 0.
7:3	Device Number	RO	0	Device number of port for PCIe. For Endpoint, this register is RO and is set by the Root Port.
15:8	Bus Number	RO	0	Bus number of port for PCIe. For Endpoint, this register is RO and is set by the external Root Port.
23:16	Port Number	RW	0	Sets the Port number field of the Link Capabilities register. EP: Always Read 0 and is not writeable. RP: Is writeable.
31:24	Reserved	RO	0	Reserved

## PHY Status/Control Register (Offset 0x144)

The PHY Status/Control register (described in the following table) provides the status of the current PHY state, as well as control of speed and rate switching for Gen2-capable cores.

Table 25: PHY Status/Control Register

Bits	Name	Core Access	Reset Value	Description
0	Link Rate is Gen2	RO	0	Reports the current link rate. 0b = 2.5 GT/s (if bit[12] = 0), or 8.0GT/s (if bit[12] = 1) 1b = 5.0 GT/s



Table 25: PHY Status/Control Register (cont'd)

Bits	Name	Core Access	Reset Value	Description
2:1	Link Width	RO	0	Reports the current link width. 00b = x1 01b = x2 10b = x4 11b = x8
8:3	LTSSM State	RO	0	Reports the current Link Training and Status State Machine (LTSSM) state. Encoding is specific to the underlying integrated block.
10:9	Reserved	RO	0	Reserved
11	Link Up	RO	0	Reports the current PHY Link-up state. 1b: Link up 0b: Link down Link up indicates the core has achieved link up status, meaning the LTSSM is in the L0 state and the core can send/receive data packets.
12	Link Rate is Gen3	RO	0	Reports the current link rate. 0b = see bit[0] 1b = 8.0 GT/s
13	Link Width is x16	RO	0	Reports the current link width. 0b = See bit[2:1] 1b = x16
31:13	Reserved	RO	0	Reserved

## Root Port Status/Control Register (Offset 0x148)

The Root Port Status/Control register provides access to the Root Port specific status and control. This register is only implemented for Root Port cores. For non-Root Port cores, reads return 0 and writes are ignored (described in the following table).

Table 26: Root Port Status/Control Register

Bits	Name	Core Access	Reset Value	Description
0	Bridge Enable	RW	0	When set, allows the reads and writes to the AXIBARs to be presented on the PCIe bus. Root Port software needs to write a 1 to this bit when enumeration is done. AXI Enhanced PCIe Bridge clears this location when link up to link down transition occurs. Default is set to 0.
15:1	Reserved	RO	0	Reserved.
16	Error FIFO Not Empty	RO	0	Indicates that the Root Port Error FIFO has data to read.

*Table 26: Root Port Status/Control Register (cont'd)*

Bits	Name	Core Access	Reset Value	Description
17	Error FIFO Overflow	RW1C	0	Indicates that the Root Port Error FIFO overflowed and an error message was dropped. Writing a 1 clears the overflow status.
18	Interrupt FIFO Not Empty	RO	0	Indicates that the Root Port Interrupt FIFO has data to read.
19	Interrupt FIFO Overflow	RW1C	0	Indicates that the Root Port Interrupt FIFO overflowed and an interrupt message was dropped. Writing a 1 clears the overflow status.
31:20	Reserved	RO	0	Reserved.

## Root Port MSI Base Register 1 (Offset 0x14C)

The Root Port MSI Base register contains the upper 32-bits of the 64-bit MSI address (described in the following table). For EP configurations, read returns zero.

*Table 27: Root Port MSI Base Register 1*

Bits	Name	Core Access	Reset Value	Description
31:0	MSI Base	RW	0	4 Kb-aligned address for MSI interrupts. In case of 32-bit MSI, it returns 0 but captures the upper 32-bits of the MSI address in case of 64-bit MSI.

## Root Port MSI Base Register 2 (Offset 0x150)

The Root Port MSI Base register 2 (described in the following table) sets the address window in Root Port cores used for MSI interrupts. MemWr TLPs to addresses in this range are interpreted as MSI interrupts. MSI TLPs are interpreted based on the address programmed in this register. The window is always 4 Kb, beginning at the address indicated in this register. For EP configurations, a read returns zero. However, the Bridge core does not support MSI-X and multiple vector address, only single MSI is supported.

*Table 28: Root Port MSI Base Register 2*

Bits	Name	Core Access	Reset Value	Description
11:0	Reserved	RO	0	Reserved
31:12	MSI Base	RW	0	4 Kb-aligned address for MSI interrupts.

## Root Port Error FIFO Read Register (Offset 0x154)

The Root Port Error FIFO Read Register reads from this location return queued error (Correctable/Non-fatal/Fatal) messages. Data from each read follows the format shown in the following table. For EP configurations, read returns zero.

Reads are non-destructive. Removing the message from the FIFO requires a write. The write value is ignored.

Table 29: Root Port Error FIFO Read Register

Bits	Name	Core Access	Reset Value	Description
15:0	Requester ID	RWC	0	Requester ID belonging to the requester of the error message.
17:16	Error Type	RWC	0	Indicates the type of the error. 00b: Correctable 01b: Non-Fatal 10b: Fatal 11b: Reserved
18	Error Valid	RWC	0	Indicates whether read succeeded. 1b: Success 0b: No message to read
31:19	Reserved	RO	0	Reserved

## Root Port Interrupt FIFO Read Register 1 (Offset 0x158)

The Root Port Interrupt FIFO Read Register 1 reads from this location return queued interrupt messages. Data from each read follows the format shown in the following table. For MSI interrupts, the message payload is presented in the Root Port Interrupt FIFO Read 2 register. The interrupt-handling flow should be to read this register first, immediately followed by the Root Port Interrupt FIFO Read 2 register. For non-Root Port cores, reads return zero.

**Note:** Reads are non-destructive. Removing the message from the FIFO requires a write to either this register or the Root Port Interrupt FIFO Read 2 register. The write value is ignored.

Table 30: Root Port Interrupt FIFO Read Register 1

Bits	Name	Core Access	Reset Value	Description
15:0	Requester ID	RWC	0	Requester ID belonging to the requester of the error message.
26:16	MSI Address	RWC	0	For MSI interrupts, contains address bits 12:2 from the TLP address field.

Table 30: Root Port Interrupt FIFO Read Register 1 (cont'd)

Bits	Name	Core Access	Reset Value	Description
28:27	Interrupt Line	RWC	0	Indicates interrupt line used. 00b: INTA 01b: INTB 10b: INTC 11b: INTD For MSI, this field is set to 00b and should be ignored.
29	Interrupt Assert	RWC	0	Indicates assert or deassert for INTx. 1b: Assert 0b: Deassert For MSI, this field is set to 0b and should be ignored.
30	MSI Interrupt	RWC	0	Indicates whether interrupt is MSI or INTx. 1b = MSI 0b = INTx
31	Interrupt Valid	RWC	0	Indicates whether read succeeded. 1b: Success 0b: No interrupt to read

## Root Port Interrupt FIFO Read Register 2 (Offset 0x15C)

The Root Port Interrupt FIFO Read Register 2 reads from this location return message payload for MSI Interrupts. The header information is presented in the Root Port Interrupt FIFO Read 1 register. The interrupt-handling flow is to read the Root Port Interrupt FIFO Read 1 register first, immediately followed by this register. For non-Root Port cores, reads return 0. For INTx interrupts, reads return zero.

Reads are non-destructive. Removing the message from the FIFO requires a write to either this register or the Root Port Interrupt FIFO Read 1 register (write value is ignored).

Table 31: Root Port Interrupt FIFO Read Register 2

Bits	Name	Core Access	Reset Value	Description
15:0	Message Data	RWC	0	Payload for MSI messages.
31:16	Reserved	RO	0	Reserved

## Root Port Interrupt Decode 2 Register (Offset 0x160)



**IMPORTANT!** This register is valid for XDMA Root Port Bridge only, and reserved for AXI PCIe3 Bridge.

The Root Port Interrupt Decode 2 Register reads from this location return INTx interrupt source status. Data from each read follows the format shown in the following table. For non-Root Port cores, reads return 0.

**Table 32: Root Port Interrupt Decode 2 Register**

Bits	Name	Core Access	Reset Value	Description
15:0	Reserved	RO	0	Reserved
16	INTA status	RO	0	1 - INTA is asserted. Keeps 1 till INTA Deassert is received. 0 - INTA is deserted. Keep 0 till INTA Assert is received.
17	INTB status	RO	0	1 - INTB is asserted. Keeps 1 till INTB Deassert is received. 0 - INTB is deserted. Keep 0 till INTB Assert is received.
18	INTC status	RO	0	1 - INTC is asserted. Keeps 1 till INTC Deassert is received. 0 - INTC is deserted. Keep 0 till INTC Assert is received.
19	INTD status	RO	0	1 - INTD is asserted. Keeps 1 till INTD Deassert is received. 0 - INTD is deserted. Keep 0 till INTD Assert is received.
31:20	Reserved	RO	0	Reserved

## Root Port Interrupt Decode 2 Mask Register (Offset 0x164)

**Note:** This register is valid for XDMA Root Port Bridge only, and reserved for AXI PCIe3 Bridge.

The RootPort Interrupt Decode 2 Mask register controls whether INTx interrupt is checked by Interrupt decode bit 16 and also forwarded to `interrupt_out` in Interrupt Decode mode. The Root Port Interrupt Decode 2 Mask Register initializes to all zeros. The following table describes the register bits.

**Table 33: Root Port Interrupt Decode 2 Mask Register**

Bits	Name	Core Access	Reset Value	Description
15:0	Reserved	RO	0	Reserved
16	INTA status	RW	0	1 - INTA is checked by Interrupt Decode [16] 0 - INTA is not checked by Interrupt Decode [16]
17	INTB status	RW	0	1 - INTB is checked by Interrupt Decode [16] 0 - INTB is not checked by Interrupt Decode [16]

*Table 33: Root Port Interrupt Decode 2 Mask Register (cont'd)*

Bits	Name	Core Access	Reset Value	Description
18	INTC status	RW	0	1 - INTC is checked by Interrupt Decode [16] 0 - INTC is not checked by Interrupt Decode [16]
19	INTD status	RW	0	1 - INTD is checked by Interrupt Decode [16] 0 - INTD is not checked by Interrupt Decode [16]
31:20	Reserved	RO	0	Reserved

## Root Port MSI Interrupt Decode 1 Register (Offset 0x170)

The Root Port MSI Interrupt Decode 1 register reads from this location return MSI vector 0-31 source status. Data from each read follows the format shown in the following table. For non-Root Port cores, reads return 0.

*Table 34: Root Port MSI Interrupt Decode 1 Register*

Bits	Name	Core Access	Reset Value	Description
31:0	MSI Vector Number	RW1C	0	MSI vector number. Bit index [x] indicates MSI vector x.

## Root Port MSI Interrupt Decode 2 Register (Offset 0x174)

The Root Port MSI Interrupt Decode 2 register reads from this location return MSI vector 32-63 source status. Data from each read follows the format shown in the following table. For non-Root Port cores, reads return 0.

*Table 35: Root Port MSI Interrupt Decode 2 Register*

Bits	Name	Core Access	Reset Value	Description
31:0	MSI Vector Number	RW1C	0	MSI vector number. Bit index [x] indicates MSI vector x+32.

## Root Port MSI Interrupt Decode 1 Mask Register (Offset 0x178)

The Root Port MSI Interrupt Decode 1 Mask register controls whether MSI interrupt vector 0-31 are forwarded to `interrupt_out_msi_vec0to31` signal in Interrupt Decode mode. The Root Port MSI Interrupt Decode 1 Mask Register initializes to all zeros. The following table describes the register bits.

Table 36: Root Port MSI Interrupt Decode 1 Mask Register

Bits	Name	Core Access	Reset Value	Description
31:0	MSI Vector Status	RW	0	1 - This MSI vector is indicated in <code>interrupt_out_msi_vec0to31</code> signal. 0 - This MSI vector is not indicated in <code>interrupt_out_msi_vec0to31</code> signal. Bit index [x] indicates MSI vector x.

## Root Port MSI Interrupt Decode 2 Mask Register (Offset 0x17C)

The Root Port MSI Interrupt Decode 2 Mask register controls whether MSI interrupt vector 32-63 are forwarded to the `interrupt_out_msi_vec32to63` signal in Interrupt Decode mode. The Root Port MSI Interrupt Decode 2 Mask Register initializes to all zeros. The following table describes the register bits.

Table 37: Root Port MSI Interrupt Decode 2 Mask Register

Bits	Name	Core Access	Reset Value	Description
31:0	MSI Vector Status	RW	0	1 - This MSI vector is indicated in <code>interrupt_out_msi_vec32to63</code> signal. 0 - This MSI vector is not indicated in <code>interrupt_out_msi_vec32to63</code> signal. Bit index [x] indicates MSI vector x+32.

## Configuration Control Register (Offset 0x168)

Configuration Control register (described in the following table) allows the user application to indicate if a correctable or uncorrectable error has occurred and report it in the respective AER Error Status Register.

Table 38: Configuration Control Register

Bits	Name	Core Access	Reset Value	Description
0	Uncorrectable Error	RW	0	Uncorrectable Error Detected. The user application writes a 1 to this bit to indicate an Uncorrectable error was 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.  This bit only asserts for 1 clock cycle and automatically resets to 0 in the next clock cycle.
1	Correctable Error	RW	0	Correctable Error Detected. The user application writes a 1 to this bit to indicate a Correctable error was 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 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.  This bit only asserts for 1 clock cycle and automatically resets to 0 in the next clock cycle.
31:2	Reserved	RO	0	Reserved

## VSEC Capability Register 2 (Offset 0x200)

The VSEC capability register (described in the following table) allows the memory space for the core to appear as though it is a part of the underlying integrated block PCIe configuration space. The VSEC is inserted immediately following the last enhanced capability structure in the underlying block. VSEC is defined in the PCI Express Base Specification, v1.1 (7.19 of v2.0), available from *PCI-SIG Specifications* (<https://www.pcisig.com/specifications>).

Table 39: VSEC Capability Register 2

Bits	Name	Core Access	Reset Value	Description
15:0	VSEC Capability ID	RO	0x000B	PCI-SIG defined ID identifying this Enhanced Capability as a Vendor-Specific capability. Hardcoded to 0x000B.
19:16	Capability Version	RO	0x1	Version of this capability structure. Hardcoded to 0x1.



Table 39: VSEC Capability Register 2 (cont'd)

Bits	Name	Core Access	Reset Value	Description
31:20	Next Capability Offset	RO	0x000	Offset to next capability.

## VSEC Header Register 2 (Offset 0x204)

The VSEC Header Register 2 (described in the following table) provides a unique (within a given vendor) identifier for the layout and contents of the VSEC structure, as well as its revision and length. VSEC Header Register 2 is part of the Bridge core that contains AXI Base Address Translation Configuration Registers which start immediately after VSEC Header Register 2 (Offset 0x208).

Table 40: VSEC Header Register 2

Bits	Name	Core Access	Reset Value	Description
15:0	VSEC ID	RO		ID value uniquely identifying the nature and format of this VSEC structure.
19:16	VSEC REV	RO	0x0	Version of this capability structure. Hardcoded to 0x0.
31:20	VSEC Length	RO	0x038	Length of the entire VSEC Capability structure, in bytes, including the VSEC Capability register. Hardcoded to 0x038 (56 decimal).

## AXI Base Address Translation Configuration Registers (Offset 0x208 - 0x234)

The AXI Base Address Translation Configuration registers and their offsets are shown in the following tables, and the register bits are described in the subsequent table. This set of registers can be used in two configurations based on the address width of the PCIe BARs. When the PCIe BAR is set to a 32-bit address space, then the translation vector should be placed into the AXIBAR2PCIEBAR\_nL register where n is the PCIe BAR number. When the BAR is set to a 64-bit address space, then the most significant 32 bits are written into the AXIBAR2PCIEBAR\_nU and the least significant 32 bits are written into AXIBAR2PCIEBAR\_nL. Care should be taken so that invalid values are not written to the address translation registers.

Table 41: AXI Base Address Translation Configuration Registers

Offset	Bits	Register Mnemonic
0x208	31-0	AXIBAR2PCIEBAR_0U
0x20C	31-0	AXIBAR2PCIEBAR_0L
0x210	31-0	AXIBAR2PCIEBAR_1U
0x214	31-0	AXIBAR2PCIEBAR_1L
0x218	31-0	AXIBAR2PCIEBAR_2U

**Table 41: AXI Base Address Translation Configuration Registers (cont'd)**

Offset	Bits	Register Mnemonic
0x21C	31-0	AXIBAR2PCIEBAR_2L
0x220	31-0	AXIBAR2PCIEBAR_3U
0x224	31-0	AXIBAR2PCIEBAR_3L
0x228	31-0	AXIBAR2PCIEBAR_4U
0x22C	31-0	AXIBAR2PCIEBAR_4L
0x230	31-0	AXIBAR2PCIEBAR_5U
0x234	31-0	AXIBAR2PCIEBAR_5L

**Table 42: AXI Base Address Translation Configuration Register Bit Definitions**

Bits	Name	Core Access	Reset Value	Description
31-0	Lower Address	R/W	C_AXIBAR2PCIEBAR_0(31 to 0)	To create the address for PCIe—this is the value substituted for the least significant 32 bits of the AXI address.
31-0	Upper Address	R/W	if (C_AXIBAR2PCIEBAR_0 = 64 bits), then reset value = C_AXIBAR2PCIEBAR_0(63 to 32) if (C_AXIBAR2PCIEBAR_0 = 32 bits), then reset value = 0x00000000	To create the address for PCIe—this is the value substituted for the most significant 32 bits of the AXI address.
31-0	Lower Address	R/W	C_AXIBAR2PCIEBAR_1(31 to 0)	To create the address for PCIe—this is the value substituted for the least significant 32 bits of the AXI address.
31-0	Upper Address	R/W	if (C_AXIBAR2PCIEBAR_1 = 64 bits), then reset value = C_AXIBAR2PCIEBAR_1(63 to 32) if (C_AXIBAR2PCIEBAR_1 = 32 bits), then reset value = 0x00000000	To create the address for PCIe—this is the value substituted for the most significant 32 bits of the AXI address.
31-0	Lower Address	R/W	C_AXIBAR2PCIEBAR_2(31 to 0)	To create the address for PCIe—this is the value substituted for the least significant 32 bits of the AXI address.
31-0	Upper Address	R/W	if (C_AXIBAR2PCIEBAR_2 = 64 bits), then reset value = C_AXIBAR2PCIEBAR_2(63 to 32) if (C_AXIBAR2PCIEBAR_2 = 32 bits), then reset value = 0x00000000	To create the address for PCIe—this is the value substituted for the most significant 32 bits of the AXI address.
31-0	Lower Address	R/W	C_AXIBAR2PCIEBAR_3(31 to 0)	To create the address for PCIe—this is the value substituted for the least significant 32 bits of the AXI address.
31-0	Upper Address	R/W	if (C_AXIBAR2PCIEBAR_3 = 64 bits) then reset value = C_AXIBAR2PCIEBAR_3(63 to 32) if (C_AXIBAR2PCIEBAR_3 = 32 bits) then reset value = 0x00000000	To create the address for PCIe—this is the value substituted for the most significant 32 bits of the AXI address.

**Table 42: AXI Base Address Translation Configuration Register Bit Definitions (cont'd)**

Bits	Name	Core Access	Reset Value	Description
31-0	Lower Address	R/W	C_AXIBAR2PCIEBAR_4(31 to 0)	To create the address for PCIe—this is the value substituted for the least significant 32 bits of the AXI address.
31-0	Upper Address	R/W	if (C_AXIBAR2PCIEBAR_4 = 64 bits), then reset value = C_AXIBAR2PCIEBAR_4(63 to 32) if (C_AXIBAR2PCIEBAR_4 = 32 bits), then reset value = 0x00000000	To create the address for PCIe—this is the value substituted for the most significant 32 bits of the AXI address.
31-0	Lower Address	R/W	C_AXIBAR2PCIEBAR_5(31 to 0)	To create the address for PCIe—this is the value substituted for the least significant 32 bits of the AXI address.
31-0	Upper Address	R/W	if (C_AXIBAR2PCIEBAR_5 = 64 bits), then reset value = C_AXIBAR2PCIEBAR_5(63 to 32) if (C_AXIBAR2PCIEBAR_5 = 32 bits), then reset value = 0x00000000	To create the address for PCIe—this is the value substituted for the most significant 32 bits of the AXI address.

## DMA/Bridge Subsystem for PCIe Register Memory Map

This section lists Interrupt registers that are unique to DMA/Bridge Subsystem for PCIe in AXI Bridge mode only. These registers listed in this section are accessible through the AXI4-Lite Control interface when address bit[28] is set to 1'b1. When address bit[28] is set to 1'b0, all of these register fields are re-purposed for Bridge operation, which contains the same Bridge Memory map layout as listed in the previous section.

**Table 43: DMA/Bridge Subsystem for PCIe Address Format**

31:29	28	27:16	15:12	11:8	7:0
Reserved	Register Table Select	Reserved	Target	Reserved	Byte Offset

**Table 44: PCIe to DMA/Bridge Address Field Descriptions**

Bit Index	Field	Description
31:29	Reserved	Reserved
28	Register Table Select	1'b0: Select Bridge and ECAM registers listed in Bridge Memory Map in the previous section. 1'b1: Select Interrupt registers listed in this section.
27:16	Reserved	When bit[28] = 1'b0: Refer to Bridge Memory map. When bit[28] = 1'b1: Reserved

Table 44: PCIe to DMA/Bridge Address Field Descriptions (cont'd)

Bit Index	Field	Description
15:12	Target	The destination submodule within the DMA 4'h0: Reserved 4'h1: Reserved 4'h2: IRQ Block 4'h3: Reserved 4'h4: Reserved 4'h5: Reserved 4'h6: Reserved 4'h8: MSI-X
11:8	Reserved	This field must be 0.
7:0	Byte Offset	The byte address of the register to be accessed within the target. Bits[1:0] must be 0.

## IRQ Block Registers (0x2)

The IRQ Block registers are described in this section.

Table 45: IRQ Block Register Space

Address (hex)	Register Name
0x00	IRQ Block Identifier (0x00)
0x04	IRQ Block User Interrupt Enable Mask (0x04)
0x08	IRQ Block User Interrupt Enable Mask (0x08)
0x0C	IRQ Block User Interrupt Enable Mask (0x0C)
0x10	Reserved
0x14	Reserved
0x18	Reserved
0x40	IRQ Block User Interrupt Request (0x40)
0x44	Reserved
0x48	IRQ Block User Interrupt Pending (0x48)
0x4C	Reserved
0x80	IRQ Block User Vector Number (0x80)
0x84	IRQ Block User Vector Number (0x84)
0x88	IRQ Block User Vector Number (0x88)
0x8C	IRQ Block User Vector Number (0x8C)
0xA0	Reserved
0xA4	Reserved

**Table 46: IRQ Block Identifier (0x00)**

Bit Index	Default	Access Type	Description
31:20	12'h1fc	RO	DMA Subsystem for PCIe identifier
19:16	4'h2	RO	IRQ Identifier
15:8	8'h0	RO	Reserved
7:0	8'h04	RO	Version 8'h01: 2015.3 and 2015.4 8'h02: 2016.1 8'h03: 2016.2 8'h04: 2016.3 8'h05: 2016.4 8'h06: 2017.1, 2017.2 and 2017.3

**Table 47: IRQ Block User Interrupt Enable Mask (0x04)**

Bit Index	Default	Access Type	Description
[NUM_USR_INT-1:0]	AXI Bridge Functional Mode: {NUM_USR_INT}'b1 DMA Functional Mode: 'h0	RW	user_int_enmask User Interrupt Enable Mask 0: Prevents an interrupt from being generated when the user interrupt source is asserted. 1: Generates an interrupt on the rising edge of the user interrupt source. If the Enable Mask is set and the source is already set, a user interrupt will be generated also.

**Table 48: IRQ Block User Interrupt Enable Mask (0x08)**

Bit Index	Default	Access Type	Description
		W1S	user_int_enmask Bit descriptions are the same as in the following table.

**Table 49: IRQ Block User Interrupt Enable Mask (0x0C)**

Bit Index	Default	Access Type	Description
		W1C	user_int_enmask Bit descriptions are the same as in table: IRQ Block User Interrupt Enable Mask (0x04).

**Table 50: IRQ Block User Interrupt Request (0x40)**

Bit Index	Default	Access Type	Description
[NUM_USR_INT-1:0]	'h0	RO	user_int_req User Interrupt Request This register reflects the interrupt source AND'd with the enable mask register.

**Table 51: IRQ Block User Interrupt Pending (0x48)**

Bit Index	Default	Access Type	Description
[NUM_USR_INT-1:0]	'h0	RO	user_int_pend User Interrupt Pending. This register indicates pending events. The pending events are cleared by removing the event cause condition at the source component.

If MSI is enabled, this register specifies the MSI or MSI-X vector number of the MSI. In Legacy interrupts only the two LSBs of each field should be used to map to INTA, B, C, or D.

**Table 52: IRQ Block User Vector Number (0x80)**

Bit Index	Default	Access Type	Description
28:24	5'h0	RW	vector 3 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[3].
20:16	5'h0	RW	vector 2 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[2].
12:8	5'h0	RW	vector 1 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[1].
4:0	5'h0	RW	vector 0 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[0].

If MSI is enabled, this register specifies the MSI or MSI-X vector number of the MSI. In Legacy interrupts only the 2 LSB of each field should be used to map to INTA, B, C, or D.

**Table 53: IRQ Block User Vector Number (0x84)**

Bit Index	Default	Access Type	Description
28:24	5'h0	RW	vector 7 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[7].
20:16	5'h0	RW	vector 6 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[6].
12:8	5'h0	RW	vector 5 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[5].
4:0	5'h0	RW	vector 4 The vector number that is used when an interrupt is generated by the user IRQ usr_irq_req[4].

If MSI is enabled, this register specifies the MSI or MSI-X vector number of the MSI. In Legacy interrupts only the 2 LSB of each field should be used to map to INTA, B, C, or D.

**Table 54: IRQ Block User Vector Number (0x88)**

Bit Index	Default	Access Type	Description
28:24	5'h0	RW	vector 11 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[11]</code> .
20:16	5'h0	RW	vector 10 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[10]</code> .
12:8	5'h0	RW	vector 9 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[9]</code> .
4:0	5'h0	RW	vector 8 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[8]</code> .

If MSI is enabled, this register specifies the MSI or MSI-X vector number of the MSI. In Legacy interrupts only the 2 LSB of each field should be used to map to INTA, B, C, or D.

**Table 55: IRQ Block User Vector Number (0x8C)**

Bit Index	Default	Access Type	Description
28:24	5'h0	RW	vector 15 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[15]</code> .
20:16	5'h0	RW	vector 14 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[14]</code> .
12:8	5'h0	RW	vector 13 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[13]</code> .
4:0	5'h0	RW	vector 12 The vector number that is used when an interrupt is generated by the user IRQ <code>usr_irq_req[12]</code> .

## MSI-X Vector Table and PBA (0x8)

The MSI-X Vector table and PBA are described in the following table.

**Table 56: MSI-X Vector Table and PBA (0x00–0xFE0)**

Byte Offset	Bit Index	Default	Access Type	Description
0x00	31:0	32'h0	RW	<code>MSIX_Vector0_Address[31:0]</code> MSI-X vector0 message lower address.
0x04	31:0	32'h0	RW	<code>MSIX_Vector0_Address[63:32]</code> MSI-X vector0 message upper address.
0x08	31:0	32'h0	RW	<code>MSIX_Vector0_Data[31:0]</code> MSI-X vector0 message data.

Table 56: MSI-X Vector Table and PBA (0x00–0xFE0) (cont'd)

Byte Offset	Bit Index	Default	Access Type	Description
0x0C	31:0	32'h0	RW	MSIX_Vector0_Control[31:0] MSI-X vector0 control. Bit Position: 31:1: Reserved. 0: Mask. When set to one, this MSI-X vector is not used to generate a message.
0x1F0	31:0	32'h0	RW	MSIX_Vector31_Address[31:0] MSI-X vector31 message lower address.
0x1F4	31:0	32'h0	RW	MSIX_Vector31_Address[63:32] MSI-X vector31 message upper address.
0x1F8	31:0	32'h0	RW	MSIX_Vector31_Data[31:0] MSI-X vector31 message data.
0x1FC	31:0	32'h0	RW	MSIX_Vector31_Control[31:0] MSI-X vector31 control. Bit Position: 31:1: Reserved. 0: Mask. When set to one, this MSI-X vector is not used to generate a message.
0xFE0	31:0	32'h0	RW	Pending_Bit_Array[31:0] MSI-X Pending Bit Array. There is one bit per vector. Bit 0 corresponds to vector0, etc.

## Enhanced Configuration Access Memory Map

When the Bridge core is configured as a Root Port, configuration traffic is generated by using the PCI Express enhanced configuration access mechanism (ECAM). ECAM functionality is available only when the core is configured as a Root Port. Reads and writes to a certain memory aperture are translated to configuration reads and writes, as specified in the PCI Express Base Specification (v3.0), §7.2.2.

The address breakdown is defined in the following table. ECAM is used in conjunction with the Bridge Register Memory Map only when used in both AXI Bridge for PCIe Gen3 core as well as DMA/Bridge Subsystem for PCIe in AXI Bridge mode core. The DMA/Bridge Subsystem for PCIe Register Memory Map does not have ECAM functionality.



When an ECAM access is attempted to the primary bus number, which defaults as bus 0 from reset, then access to the type 1 PCI Configuration Header of the integrated block in the Enhanced Interface for PCIe is performed. When an ECAM access is attempted to the secondary bus number, then type 0 configuration transactions are generated. When an ECAM access is attempted to a bus number that is in the range defined by the secondary bus number and subordinate bus number (not including the secondary bus number), then type 1 configuration transactions are generated. The primary, secondary, and subordinate bus numbers are written by Root Port software to the type 1 PCI Configuration Header of the Enhanced Interface for PCIe in the beginning of the enumeration procedure.

When an ECAM access is attempted to a bus number that is out of the bus\_number and subordinate bus number range, the bridge does not generate a configuration request and signal SLVERR response on the AXI4-Lite bus. When the Bridge is configured for EP (PL\_UPSTREAM\_FACING = TRUE), the underlying Integrated Block configuration space and the core memory map are available at the beginning of the memory space. The memory space looks like a simple PCI Express® configuration space. When the Bridge is configured for RC (PL\_UPSTREAM\_FACING = FALSE), the same is true, but it also looks like an ECAM access to primary bus, Device 0, Function 0.

When the Bridge core is configured as a Root Port, the reads and writes of the local ECAM are Bus 0. Because the FPGA only has a single Integrated Block for PCIe core, all local ECAM operations to Bus 0 return the ECAM data for Device 0, Function 0.

Configuration write accesses across the PCI Express bus are non-posted writes and block the AXI4-Lite interface while they are in progress. Because of this, system software is not able to service an interrupt if one were to occur. However, interrupts due to abnormal terminations of configuration transactions can generate interrupts. ECAM read transactions block subsequent Requester read TLPs until the configuration read completions packet is returned to allow unique identification of the completion packet.

**Table 57: ECAM Addressing**

Bits	Name	Description
1:0	Byte Address	Ignored for this implementation. The s<n>_axi_wstrb signals define byte enables for ECAM accesses.
7:2	Register Number	Register within the configuration space to access.
11:8	Extended Register Number	Along with Register Number, allows access to PCI Express Extended Configuration Space.
14:12	Function Number	Function Number to completer.
19:15	Device Number	Device Number to completer.
27:20	Bus Number	Bus Number to completer.

# Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

---

## General Design Guidelines

The Xilinx® Vivado® Design Suite has been optimized to provide a starting point for designing with the Bridge core.

---

## Shared Logic

The AXI Bridge for AXI Bridge for PCI Express Gen3 and DMA/Bridge Subsystem for PCIe in AXI Bridge mode support Shared Logic and Shared clocking features that are available in the PCIe subcore IPs. Note that each device family contain different Shared Logic and/or Shared Clocking features that can be supported based on the IP configuration. More details about these features can be found in the “Designing with the Core” chapter in the following documents.

- For the Virtex-7 XT device, see *Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG023](#)).
- For UltraScale devices, see *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG156](#)).
- For UltraScale+ devices, see *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#)).

# Clocking

The reference clock input is used to generate the internal clocks used by the core and the output clock. Note that the reference clock is `refclk` in Virtex®-7 devices, and `sys_clk_gt` in UltraScale™ and UltraScale+™ devices. This clock must be provided at the reference clock frequency selected in the Vivado® Integrated Design Environment (IDE) during IP generation. This port should be driven by the PCI Express edge connector clock pins through an `IBUFDSGTE` primitive.

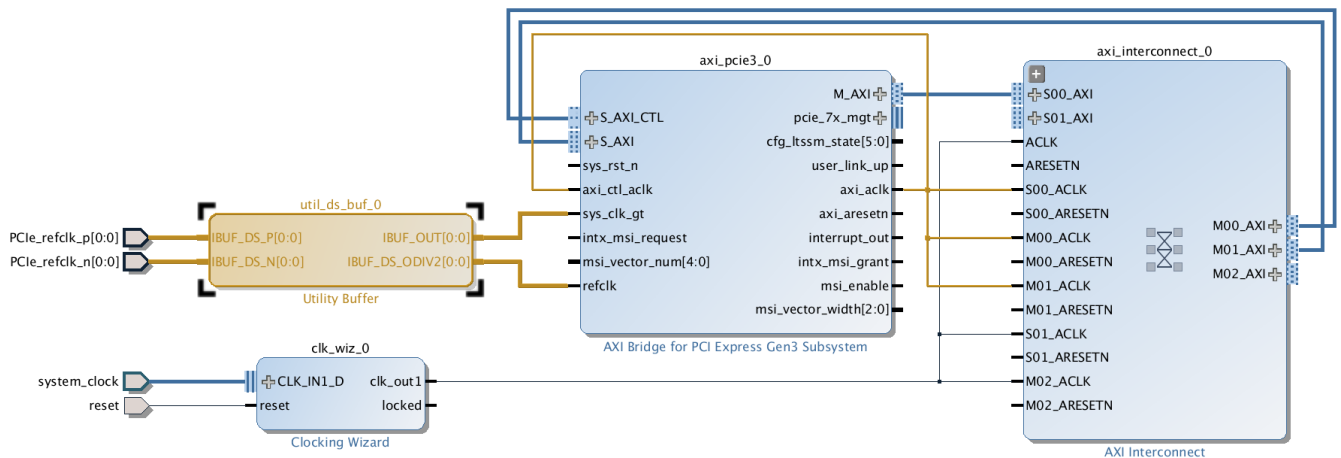
The `axi_aclk` output is the clock used for all AXI interfaces and should drive all corresponding AXI Interconnect `aclk` signals as well as the `axi_ctl_aclk` input port when using the older version of the AXI Bridge for PCI Express Gen3 core with the `axi_ctl_aclk` port available externally.

**Note:** The `axi_aclk` output should not be used for the system clock for your design. The `axi_aclk` is not a free-run clock output. As noted, `axi_aclk` may not be present at all times.

For additional information about how the source of the `aclk` clock might impact your designs, see the *7 Series FPGAs Clocking Resources User Guide (UG472)*, or the *UltraScale Architecture Clocking Resources User Guide (UG572)*.

The following figure shows the clocking diagram for the core in an UltraScale device.

Figure 14: Clocking Diagram (UltraScale Devices)



The following figure shows the clocking diagram for the core in an UltraScale™ device.

Figure 15: Clocking Diagram (UltraScale+ Devices)

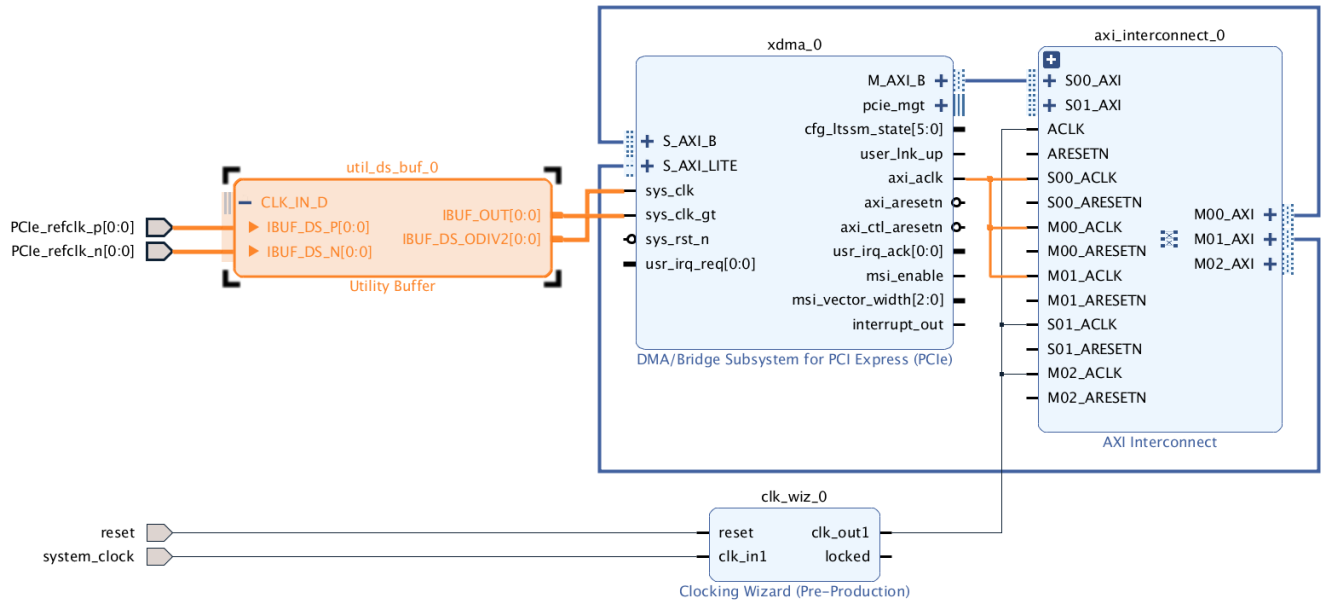


Table 58: Clock Frequencies and Interface Widths Supported For Various Configurations

PCIe Link Speed Capability	PCIe Link Width Capability	AXI4 Memory Mapped Interface Data Width (bits)	user_clk2 Frequency (MHz) (AXI4-Stream)
Gen1	X1	64	62.5
		64	125
		64	250
	X2	64	62.5
		64	125
		64	250
	X4	64	125
		64	250
	X8	64	250
128		125	
X16	128	250	

**Table 58: Clock Frequencies and Interface Widths Supported For Various Configurations (cont'd)**

PCIe Link Speed Capability	PCIe Link Width Capability	AXI4 Memory Mapped Interface Data Width (bits)	user_clk2 Frequency (MHz) (AXI4-Stream)
Gen2	X1	64	62.5
		64	125
		64	250
	X2	64	125
		64	250
	X4	64	250
		128	125
	X8	128	250
		256	125
	X16	256	250
Gen3	X1	64	125
		64	250
	X2	64	250
		128	125
	X4	128	250
		256	125
	X8	256	250
		256	250
	X16	512	250
	Gen4 <sup>2</sup>	X1	64
128			125
X2		128	250
		256	125
X4		256	250
X8		512	250

**Notes:**

1. All X16 rows (Gen1/2/3) exist in DMA/Bridge Subsystem for PCIe in AXI Bridge mode only.
2. All Gen4 rows are only applicable to Virtex® UltraScale+™ devices with high bandwidth memory (HBM) (PCIe4C).

## Reference Clock for PCIe Frequency Value

The reference clock input used by the serial transceiver for PCI™ must be 100 MHz, 125 MHz, or 250 MHz. Note that the reference clock is `refclk` in Virtex®-7 devices, and `sys_clk_gt` in UltraScale™ and UltraScale+™ devices. The `REF_CLK_FREQ` parameter is used to set this value, as defined in [Bridge Parameters](#). The reference clock input must be fed in from a clock source that is external to the chip.

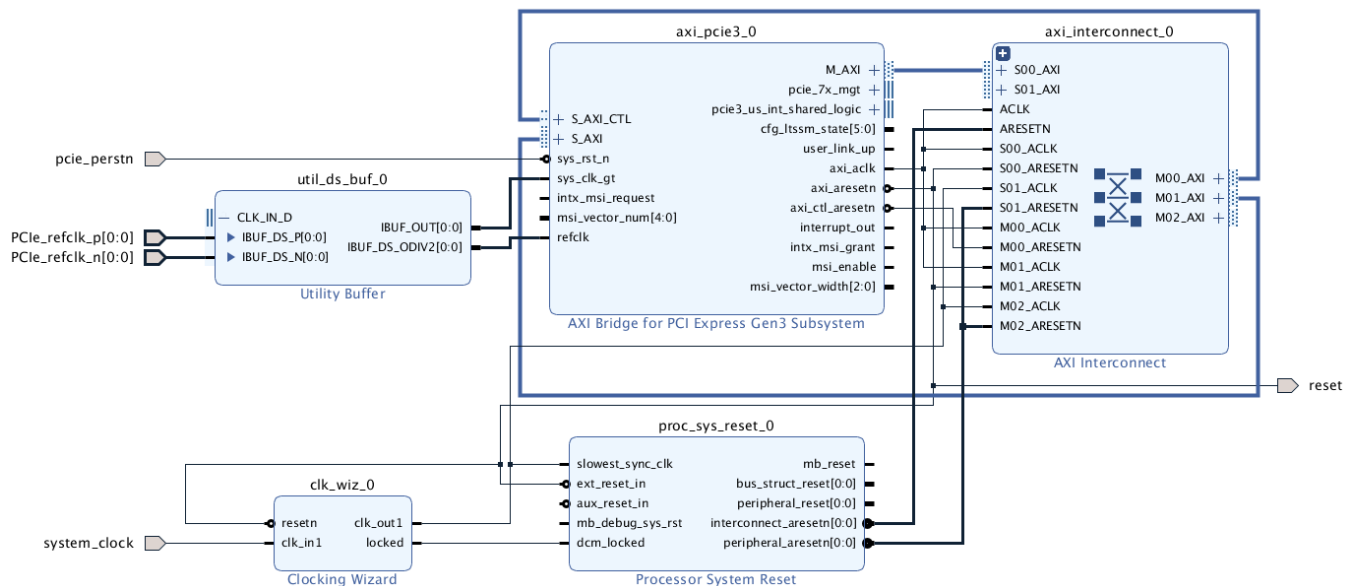
## Resets

For endpoint configurations, the `sys_rst_n` signal should be driven by the PCI Express® edge connector reset (`perstn`). This serves as the reset for the PCI Express interface.

The `axi_aresetn` output the AXI reset signal synchronous with the clock provided on the `axi_aclk` output. This reset should drive all corresponding AXI Interconnect `aresetn` signals.

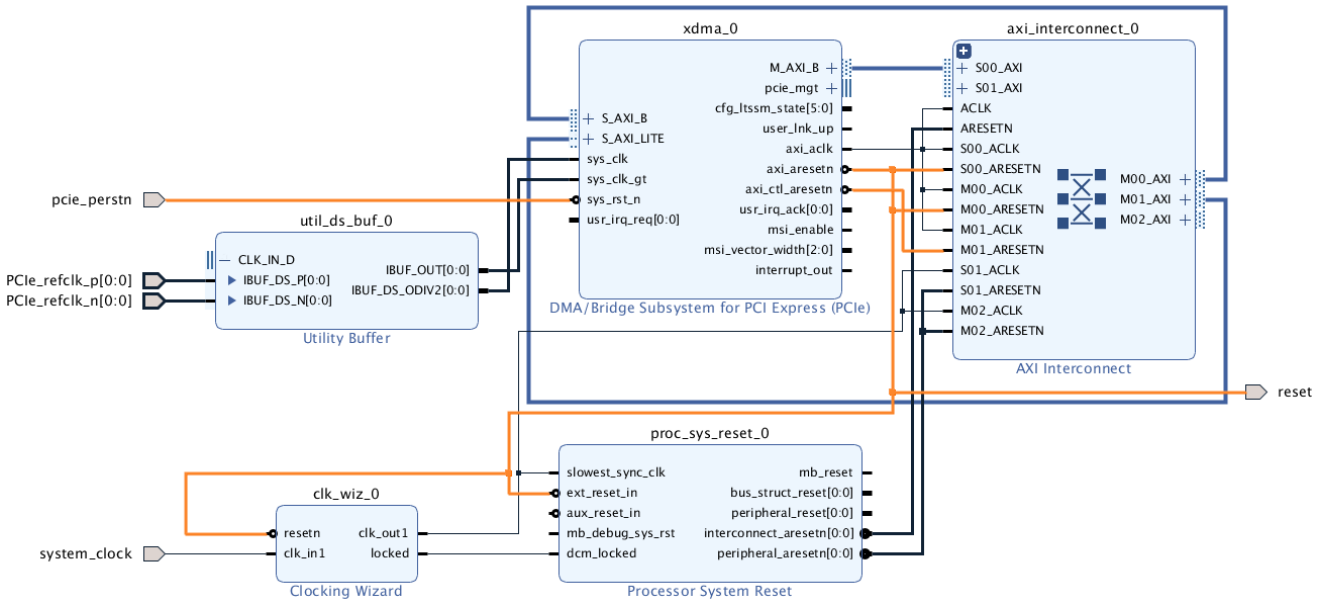
The following figure shows the Endpoint system reset connection for the core in an UltraScale™ device.

Figure 16: Endpoint System Reset Connection (UltraScale Devices)



The following figure shows the Endpoint system reset connection for the core in an UltraScale+ device.

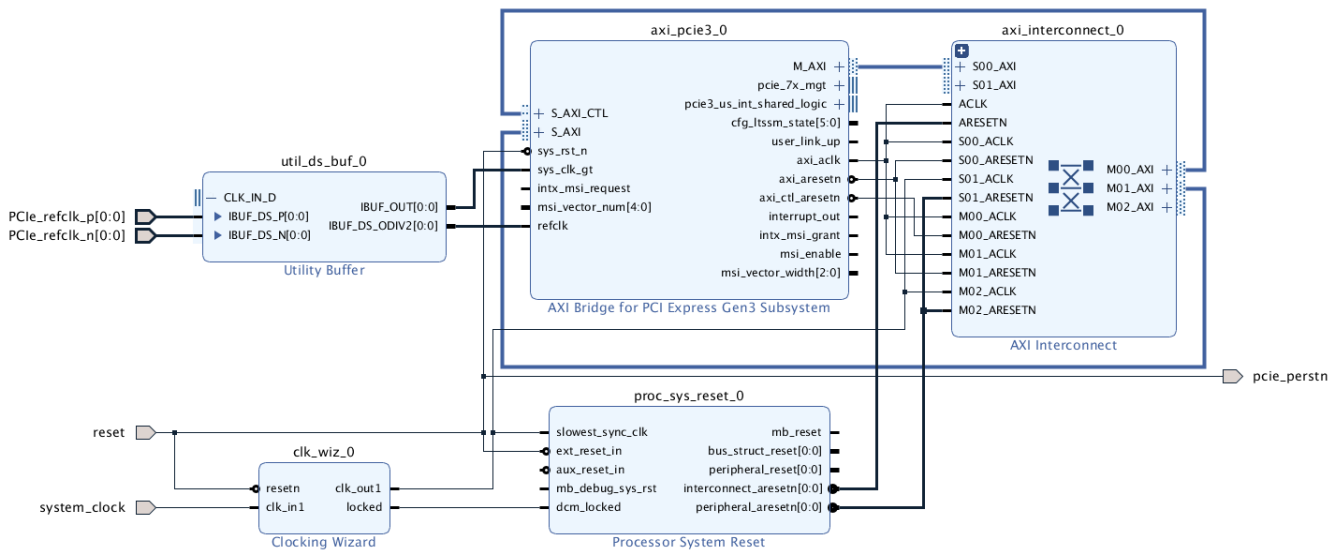
Figure 17: Endpoint System Reset Connection (UltraScale+ Devices)



For Root Port configurations, the `sys_rst_n` signal is internally generated by the user design. This serves as the reset to the PCI Express slot connector reset (`perstn`).

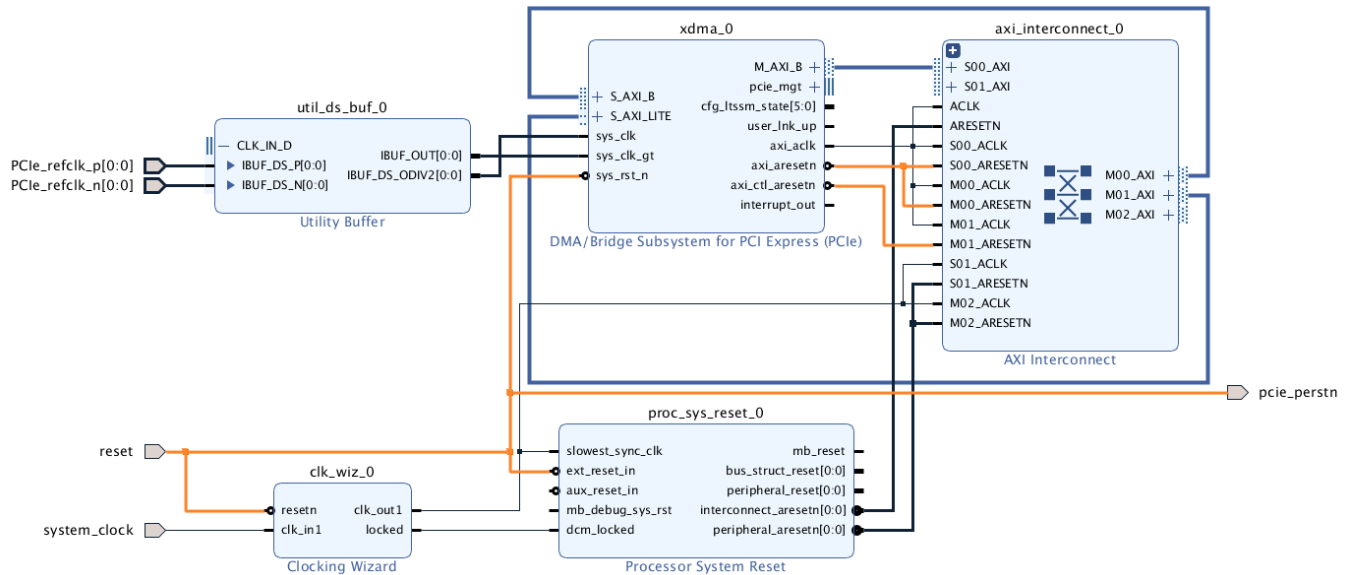
The following figure shows the Root Port system reset connection for the core in an UltraScale device.

Figure 18: Root Port System Reset Connection (UltraScale Devices)



The following figure shows the Root Port system reset connection for the core in an UltraScale+ device

Figure 19: Root Port System Reset Connection (UltraScale+ Devices)



Available for the DMA/Bridge Subsystem for PCIe in AXI Bridge mode, there is an optional `dma_bridge_resetn` input pin which allows you to reset all internal Bridge engines and registers as well as all AXI peripherals driven by `axi_aresetn` and `axi_ctl_aresetn` pins. When the following parameter is set, `dma_bridge_resetn` does not need to be asserted during initial link up operation because it will be done automatically by the IP. You must terminate all transactions before asserting this pin. After being asserted, the pin must be kept asserted for a minimum duration of at least equal to the Completion Timeout value (typically 50 ms) to clear any pending transfer that may currently be queued in the data path. To set this parameter, type the following command at the Tcl command line:

```
set_property -dict [list CONFIG.soft_reset_en {true}] [get_ips <ip-name>]
```

## AXI Transactions for PCIe

The following tables are the translation tables for AXI4-Stream and memory-mapped transactions.

Table 59: AXI4 Memory-Mapped Transactions to AXI4-Stream PCIe TLPs

AXI4 Memory-Mapped Transaction	AXI4-Stream PCIe TLPs
INCR Burst Read of AXIBAR	MemRd 32 (3DW)
INCR Burst Write to AXIBAR	MemWr 32 (3DW)
INCR Burst Read of AXIBAR	MemRd 64 (4DW)
INCR Burst Write to AXIBAR	MemWr 64 (4DW)



Table 60: AXI4-Stream PCIe TLPs to AXI4 Memory Mapped Transactions

AXI4-Stream PCIe TLPs	AXI4 Memory-Mapped Transaction
MemRd 32 (3DW) of PCIEBAR	INCR Burst Read
MemWr 32 (3DW) to PCIEBAR	INCR Burst Write
MemRd 64 (4DW) of PCIEBAR	INCR Burst Read
MemWr 64 (4DW) to PCIEBAR	INCR Burst Write

For PCIe® requests with lengths greater than 1 Dword, the size of the data burst on the Master AXI interface will always equal the width of the AXI data bus even when the request received from the PCIe link is shorter than the AXI bus width.

`slave_axi_wstrb` can be used to facilitate data alignment to an address boundary. `slave_axi_wstrb` may equal 0 in the beginning of a valid data cycle and will appropriately calculate an offset to the given address. However, the valid data identified by `slave_axi_wstrb` must be continuous from the first byte enable to the last byte enable.

All transactions initiated at the Slave Bridge interface will be modified and metered by the IP as necessary. The Slave Bridge interface will conform to AXI4 specification and allow burst size up to 4KB, and the IP will split the transaction automatically according to PCIe Max Read Request Size (MRRS), Max Payload Size (MPS), and Read Completion Boundary (RCB). As a result of this operation, one request at the AXI domain may result in multiple request at the PCIe domain, and the IP will adjust the number of issued PCIe request accordingly to avoid oversubscribing the available Completion buffer.

## Transaction Ordering for PCIe

The Bridge core conforms to PCIe® transaction ordering rules. See the [PCI-SIG Specifications](#) for the complete rule set. The following behaviors are implemented in the Bridge core to enforce the PCIe transaction ordering rules on the highly-parallel AXI bus of the bridge.

- The `bresp` to the remote (requesting) AXI4 master device for a write to a remote PCIe device is not issued until the `MemWr` TLP transmission is guaranteed to be sent on the PCIe link before any subsequent TX-transfers.
- If Relaxed Ordering bit is not set within the TLP header, then a remote PCIe device read to a remote AXI slave is not permitted to pass any previous remote PCIe device writes to a remote AXI slave received by the Bridge core. The AXI read address phase is held until the previous AXI write transactions have completed and `bresp` has been received for the AXI write transactions. If the Relaxed Ordering attribute bit is set within the TLP header, then the remote PCIe device read is permitted to pass.

- Read completion data received from a remote PCIe device are not permitted to pass any remote PCIe device writes to a remote AXI slave received by the Bridge core prior to the read completion data. The `bresp` for the AXI write(s) must be received before the completion data is presented on the AXI read data channel.

**Note:** The transaction ordering rules for PCIe might have an impact on data throughput in heavy bidirectional traffic.

---

## BAR and Address Translation

### BAR Addressing

`Aperture_Base_Address_n` represents Aperture Base Address of nth BAR in GUI

`Aperture_High_Address_n` represents Aperture High Address of nth BAR in GUI

`AXI to PCIe Translation_n` represents AXI to PCIe\_translation of nth BAR in GUI

`Aperture_Base_Address_n` and `Aperture_High_Address_n` are used to calculate the size of the AXI BAR *n* and during address translation to PCIe address.

- `Aperture_Base_Address_n` provides the low address where AXI BAR *n* starts and will be regarded as address offset 0x0 when the address is translated.
- `Aperture_High_Address_n` is the high address of the last valid byte address of AXI BAR *n*. (For more details on how the address gets translated, see Address Translation.)

The difference between `Aperture_Base_Address_n` and `Aperture_High_Address_n` is your AXI BAR *n* size. These values must be set accordingly such that the AXI BAR *n* size is a power of two and must have at least 4K.

When a packet is sent to the core (outgoing PCIe packets), the packet must have an address that is in the range of `Aperture_Base_Address_n` and `Aperture_High_Address_n`. Any packet that is received by the core that has an address outside of this range will be responded to with a SLVERR. When the IP integrator is used, these parameters are derived from the Address Editor tab within the IP integrator. The Address Editor sets the AXI Interconnect as well as the core so the address range matches, and the packet is routed to the core only when the packet has an address within the valid range.

AXI Address width is limited to 48 bits.

## Address Translation

The address space for PCIe® is different than the AXI address space. To access one address space from another address space requires an address translation process. On the AXI side, the bridge supports mapping to PCIe on up to six 32-bit or 64-bit AXI base address registers (BARs).

Four examples follow:

- Example 1 (32-bit PCIe Address Mapping) demonstrates how to set up three AXI BARs and translate the AXI address to a 32-bit address for PCIe.
- Example 2 (64-bit PCIe Address Mapping) demonstrates how to set up three AXI BARs and translate the AXI address to a 64-bit address for PCIe.
- Example 3 demonstrates how to set up two 64-bit PCIe BARs and translate the address for PCIe to an AXI address.
- Example 4 demonstrates how to set up a combination of two 32-bit AXI BARs and two 64 bit AXI BARs, and translate the AXI address to an address for PCIe.

### Example 1 (32-bit PCIe Address Mapping)

This example shows the generic settings to set up three independent AXI BARs and address translation of AXI addresses to a remote 32-bit address space for PCIe. This setting of AXI BARs does not depend on the BARs for PCIe in the Bridge core.

In this example, where C\_AXIBAR\_NUM=3, the following assignments for each range are made:

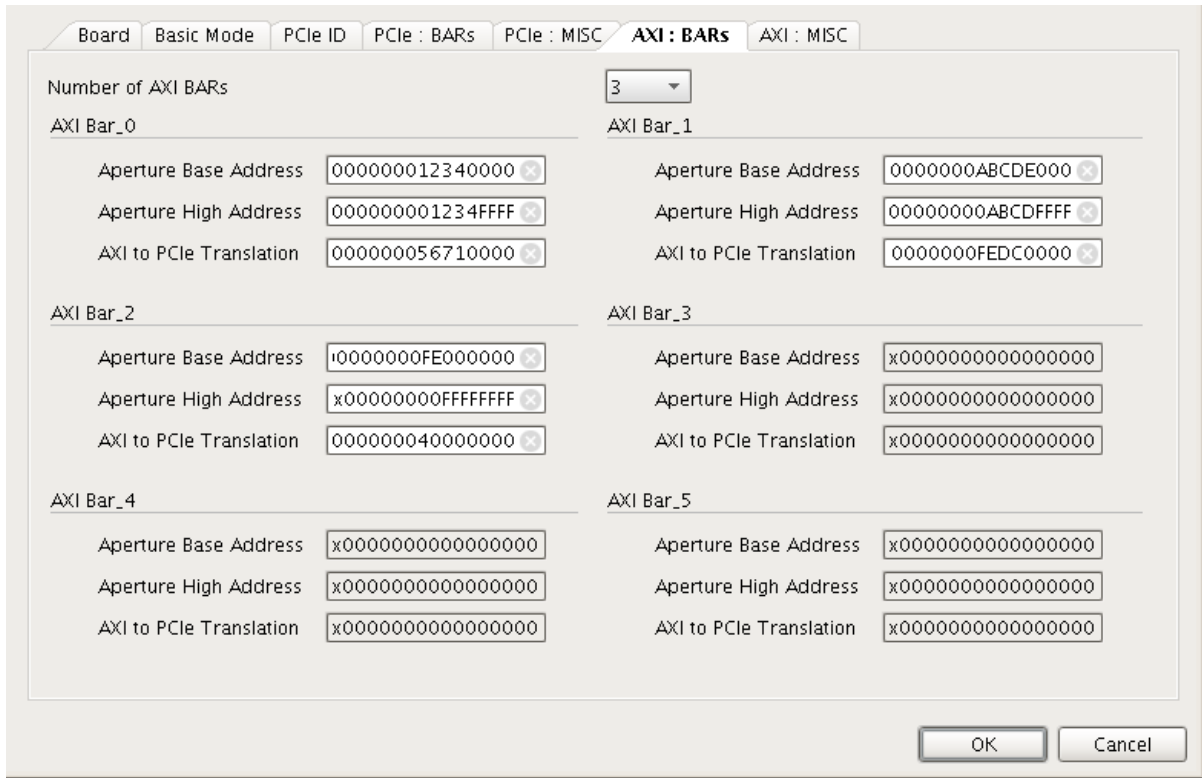
```
AXI_ADDR_WIDTH=48

C_AXIBAR_0=0x00000000_12340000
C_AXI_HIGHADDR_0=0x00000000_1234FFFF (64 Kbytes)
C_AXIBAR2PCIEBAR_0=0x00000000_56710000 (Bits 63-32 are zero in order to
produce a
32-bit PCIe TLP. Bits 15-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 16 bits are invalid translation values.)

C_AXIBAR_1=0x00000000_ABCDE000
C_AXI_HIGHADDR_1=0x00000000_ABCDFFFF (8 Kbytes)
C_AXIBAR2PCIEBAR_1=0x00000000_FEDC0000 (Bits 63-32 are zero in order to
produce a
32-bit PCIe TLP. Bits 12-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 13 bits are invalid translation values.)

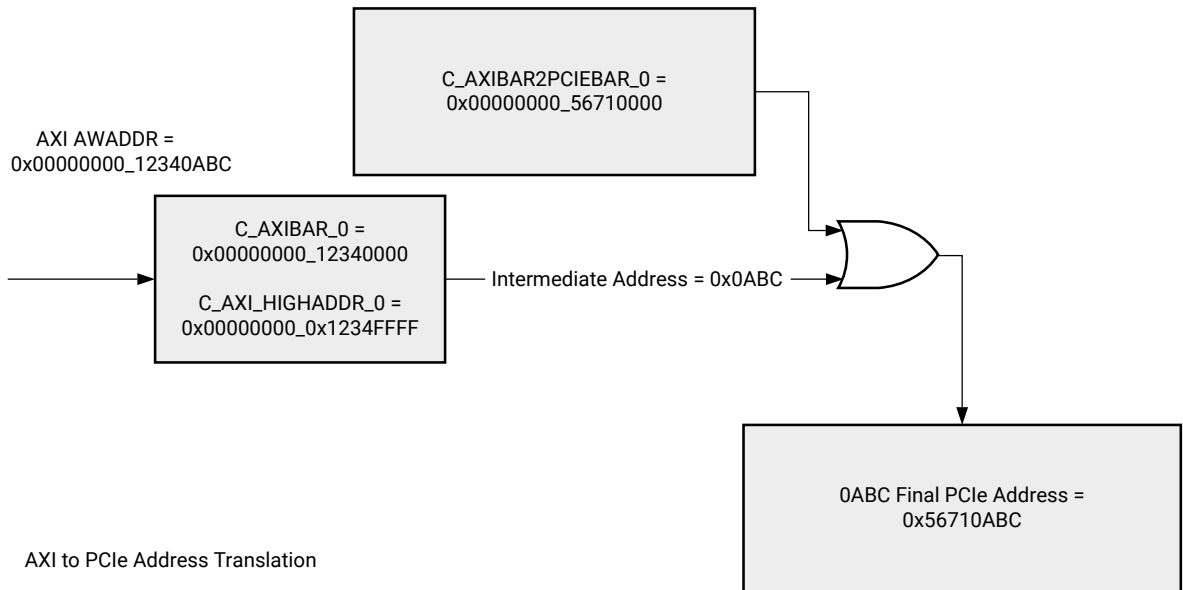
C_AXIBAR_2=0x00000000_FE000000
C_AXI_HIGHADDR_2=0x00000000_FFFFFFFF (32 Mbytes)
C_AXIBAR2PCIEBAR_2=0x00000000_40000000 (Bits 63-32 are zero in order to
produce a
32-bit PCIe TLP. Bits 24-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 25 bits are invalid translation values.)
```

Figure 20: Example 1 Settings



- Accessing the Bridge AXIBAR\_0 with address 0x0000\_12340ABC on the AXI bus yields 0x56710ABC on the bus for PCIe.

Figure 21: AXI to PCIe Address Translation



X20046-032119

- Accessing the Bridge AXIBAR\_1 with address 0x0000\_ABCDF123 on the AXI bus yields 0xFEDC1123 on the bus for PCIe.
- Accessing the Bridge AXIBAR\_2 with address 0x0000\_FFEDCBA on the AXI bus yields 0x41FEDCBA on the bus for PCIe.

## Example 2 (64-bit PCIe Address Mapping)

This example shows the generic settings to set up to three independent AXI BARs and address translation of AXI addresses to a remote 64-bit address space for PCIe. This setting of AXI BARs does not depend on the BARs for PCIe within the Bridge.

In this example, where C\_AXIBAR\_NUM=3, the following assignments for each range are made:

```
AXI_ADDR_WIDTH=48
```

```
C_AXIBAR_0=0x00000000_12340000
```

```
C_AXI_HIGHADDR_0=0x00000000_1234FFFF (64 Kbytes)
```

```
C_AXIBAR2PCIEBAR_0=0x50000000056710000 (Bits 63-32 are non-zero in order to
produce a
64-bit PCIe TLP. Bits 15-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 16 bits are invalid translation values.)
```

```
C_AXIBAR_1=0x00000000_ABCDE000
```

```
C_AXI_HIGHADDR_1=0x00000000_ABCDFFFF (8 Kbytes)
```

```
C_AXIBAR2PCIEBAR_1=0x60000000_FEDC0000 (Bits 63-32 are non-zero in order to
produce
a 64-bit PCIe TLP. Bits 12-0 must be zero based on the AXI BAR aperture
size. Non-zero
values in the lower 13 bits are invalid translation values.)
```

```
C_AXIBAR_2=0x00000000_FE000000
```

```
C_AXI_HIGHADDR_2=0x00000000_FFFFFFFF (32 Mbytes)
```

```
C_AXIBAR2PCIEBAR_2=0x7000000040000 (Bits 63-32 are non-zero in order to
produce a
64-bit PCIe TLP. Bits 24-0 must be zero based on the AXI BAR aperture size.
Non-zero
values in the lower 25 bits are invalid translation values.)
```

Figure 22: Example 2 Settings

- Accessing the Bridge AXIBAR\_0 with address 0x0000\_12340ABC0x5000000056710ABC on the bus for PCIe.
- Accessing the Bridge AXIBAR\_1 with address 0x0000\_ABCDF123 on the AXI bus yields on the AXI bus yields 0x60000000FEDC1123 on the bus for PCIe.
- Accessing the Bridge AXIBAR\_2 with address 0x0000\_FFFEDCBA on the AXI bus yields 0x7000000041FEDCBA on the bus for PCIe.

### Example 3

This example shows the generic settings to set up two independent BARs for PCIe® and address translation of addresses for PCIe to a remote AXI address space. This setting of BARs for PCIe does not depend on the AXI BARs within the bridge.

In this example, where C\_PCIEBAR\_NUM=2, the following range assignments are made.

```
AXI_ADDR_WIDTH=48
```

```
BAR 0 is set to 0x20000000_ABCD8000 by the Root Port. (Since this is a 64-bit BAR
PCIe, BAR1 is disabled.)
PFO_BAR0_APERTURE_SIZE=0x08 (32 Kbytes)
C_PCIEBAR2AXIBAR_0=0x00000000_12340000 (Because the AXI address is 48-bits
wide,
```

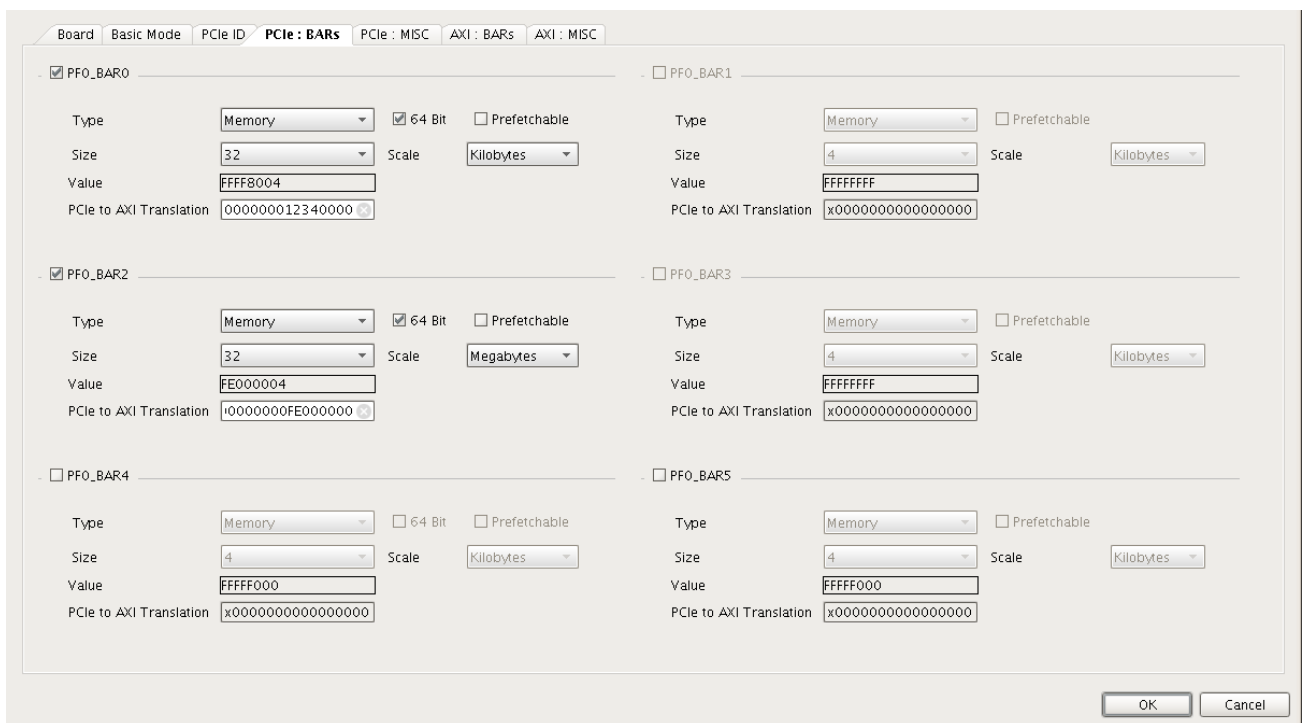
bits 63-48 should be zero. Base on the PCIe Bar Size bits 14-0 should be zero. Non-zero values in these ranges are invalid.)

BAR 2 is set to 0xA000000012000000 by Root Port. (Since this is a 64-bit BAR PCIe BAR3 is disabled.)

PFO\_BAR0\_APERTURE\_SIZE=0x12 (32 Mbytes)

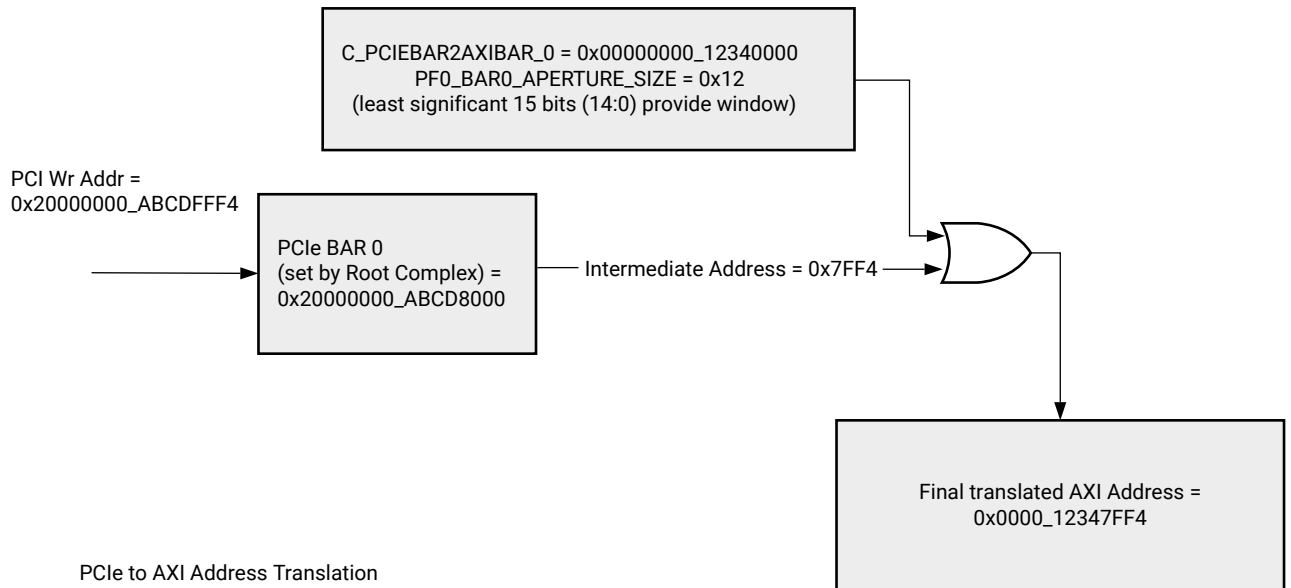
C\_PCIEBAR2AXIBAR\_2=0x00000000\_FE000000 (Because the AXI address is 48-bits wide, bits 63-48 should be zero. Base on the PCIe Bar Size bits 24-0 should be zero. Non-zero values in these ranges are invalid.)

Figure 23: Example 3 Settings



- Accessing the Bridge AXIBAR\_0 with address 0x20000000\_ABCDFFF4 on the bus for PCIe yields 0x0000\_12347FF4 on the AXI bus.

Figure 24: PCIe to AXI Translation



X20047-032119

- Accessing Bridge AXIBAR\_2 with address 0xA00000001235FEDC on the bus for PCIe yields 0x0000\_FE35FEDC on the AXI bus.

## Example 4

This example shows the generic settings of four AXI BARs and address translation of AXI addresses to a remote 32-bit and 64-bit addresses for PCIe®. This setting of AXI BARs do not depend on the BARs for PCIe within the Bridge.

In this example, where number AXI BAR's are 4, the following assignments for each range are made:

```
Aperture_Base_Address_0 =0x00000000_12340000
Aperture_High_Address_0 =0x00000000_1234FFFF (64 KB)
AXI_to_PCIe_Translation_0=0x00000000_56710000 (Bits 63-32 are zero to
produce a 32-bit PCIe
TLP. Bits 15-0 must be zero based on the AXI BAR aperture size. Non-zero
values in
the lower 16 bits are invalid translation values.)

Aperture_Base_Address_1 =0x00000000_ABCDE000
Aperture_High_Address_1 =0x00000000_ABCDFFFF (8 KB)
AXI_to_PCIe_Translation_1=0x50000000_FEDC0000 (Bits 63-32 are non-zero to
produce a 64-bit
PCIe TLP. Bits 12-0 must be zero based on the AXI BAR aperture size. Non-
zero values
in the lower 13 bits are invalid translation values.)

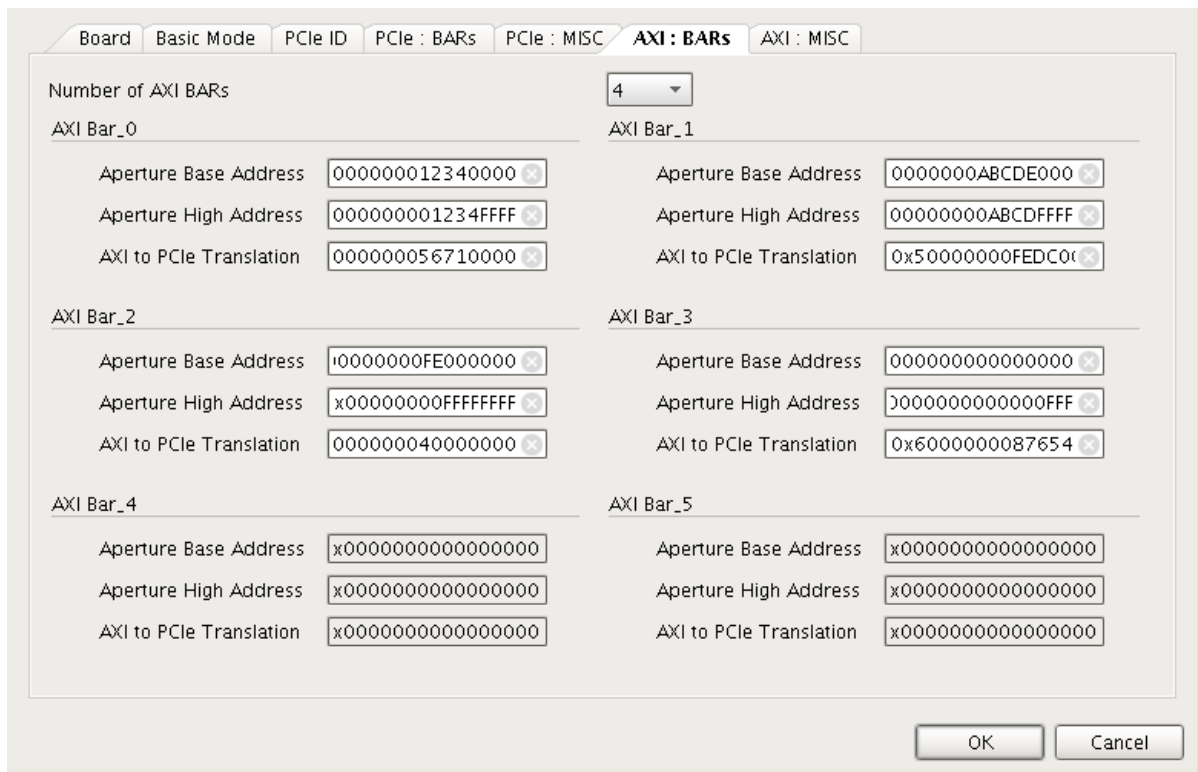
Aperture_Base_Address_2 =0x00000000_FE000000
Aperture_High_Address_2 =0x00000000_FFFFFFFF (32 MB)
```



```
AXI_to_PCIe_Translation_2=0x00000000_40000000 (Bits 63-32 are zero to
produce a 32-bit PCIe
TLP. Bits 24-0 must be zero based on the AXI BAR aperture size. Non-zero
values in
the lower 25 bits are invalid translation values.)

Aperture_Base_Address_3 =0x00000000_00000000
Aperture_High_Address_3 =0x00000000_00000FFF (4 KB)
AXI_to_PCIe_Translation_3=0x60000000_87654000 (Bits 63-32 are non-zero to
produce a 64-bit
PCIe TLP. Bits 11-0 must be zero based on the AXI BAR aperture size. Non-
zero values
in the lower 12 bits are invalid translation values.)
```

Figure 25: Example 4 Settings



- Accessing the Bridge AXI BAR\_0 with address 0x0000\_12340ABC on the AXI bus yields 0x56710ABC on the bus for PCIe.
- Accessing the Bridge AXI BAR\_1 with address 0x0000\_ABCDF123 on the AXI bus yields 0x50000000FEDC1123 on the bus for PCIe.
- Accessing the Bridge AXI BAR\_2 with address 0x0000\_FFFEDCBA on the AXI bus yields 0x41FEDCBA on the bus for PCIe.
- Accessing the Bridge AXI BAR\_3 with address 0x0000\_00000071 on the AXI bus yields 0x6000000087654071 on the bus for PCIe.

## Addressing Checks

When setting the following parameters for PCIe® address mapping, `C_PCIEBAR2AXIBAR_n` and `PF0_BARn_APERTURE_SIZE`, be sure these are set to allow for the addressing space on the AXI system. For example, the following setting is illegal and results in an invalid AXI address.

```
C_PCIEBAR2AXIBAR_n=0x00000000_FFFF000  
PF0_BARn_APERTURE_SIZE=0x06 (8 KB)
```

For an 8 Kilobyte BAR, the lower 13 bits must be zero. As a result, the `C_PCIEBAR2AXIBAR_n` value should be modified to be `0x00000000_FFFFE000`. Also, check for a larger value on `PF0_BARn_APERTURE_SIZE` compared to the value assigned to the `C_PCIEBAR2AXIBAR_n` parameter. An example parameter setting follows.

```
C_PCIEBAR2AXIBAR_n=0xFFFF_E000  
PF0_BARn_APERTURE_SIZE=0x0D (1 MB)
```

To keep the AXIBAR upper address bits as `0xFFFF_E000` (to reference bits [31:13]), the `PF0_BARn_APERTURE_SIZE` parameter must be set to `0x06` (8 KB).

---

## Malformed TLP

The integrated block for PCI Express® detects a malformed TLP. For the IP configured as an Endpoint core, a malformed TLP results in a fatal error message being sent upstream if error reporting is enabled in the Device Control register.

---

## Abnormal Conditions

This section describes how the Slave side and Master side (see the following tables) of the Bridge core handle abnormal conditions.

### Slave Bridge Abnormal Conditions

## ***Illegal Burst Type***

The slave bridge monitors AXI read and write burst type inputs to ensure that only the INCR (incrementing burst) type is requested. Any other value on these inputs is treated as an error condition and the Slave Illegal Burst (SIB) interrupt is asserted. In the case of a read request, the Bridge asserts SLVERR for all data beats and arbitrary data is placed on the `s_axi_rdata` bus. In the case of a write request, the Bridge asserts SLVERR for the write response and all write data is discarded.

## ***Completion TLP Errors***

Any request to the bus for PCIe (except for a posted Memory write) requires a completion TLP to complete the associated AXI request. The Slave side of the Bridge checks the received completion TLPs for errors and checks for completion TLPs that are never returned (Completion Timeout). Each of the completion TLP error types are discussed in the subsequent sections.

### **Unexpected Completion**

When the slave bridge receives a completion TLP, it matches the header RequesterID and Tag to the outstanding RequesterID and Tag. A match failure indicates the TLP is an Unexpected Completion which results in the completion TLP being discarded and a Slave Unexpected Completion (SUC) interrupt strobe being asserted. Normal operation then continues.

### **Unsupported Request**

A device for PCIe might not be capable of satisfying a specific read request. For example, if the read request targets an unsupported address for PCIe, the completer returns a completion TLP with a completion status of `0b001 - Unsupported Request`. The completer that returns a completion TLP with a completion status of `Reserved` must be treated as an unsupported request status, according to the PCI Express Base Specification v3.0. When the slave bridge receives an unsupported request response, the Slave Unsupported Request (SUR) interrupt is asserted and the DECERR response is asserted with arbitrary data on the AXI4 memory mapped bus.

### **Completion Timeout**

A Completion Timeout occurs when a completion (Cpl) or completion with data (CplD) TLP is not returned after an AXI to PCIe memory read request, or after a PCIe Configuration Read/Write request. Completions must be received within the value set in the Device Control 2 register in the PCIe configuration space register. When a completion timeout occurs for a PCIe memory read request, AXI Bridge for PCIe Gen3 IP provides SLVERR response, while DMA/Bridge Subsystem for PCIe in Bridge Mode IP provides OKAY response with 0s data on the AXI4 memory mapped bus.

## Poison Bit Received on Completion Packet

An Error Poison occurs when the completion TLP EP bit is set, indicating that there is poisoned data in the payload. When the slave bridge detects the poisoned packet, the Slave Error Poison (SEP) interrupt is asserted and the SLVERR response is asserted with arbitrary data on the memory mapped AXI4 bus.

## Completer Abort

A Completer Abort occurs when the completion TLP completion status is 0b100 - Completer Abort. This indicates that the completer has encountered a state in which it was unable to complete the transaction. When the slave bridge receives a completer abort response, the Slave Completer Abort (SCA) interrupt is asserted and the SLVERR response is asserted with arbitrary data on the memory mapped AXI4 bus.

*Table 61: Slave Bridge Response to Abnormal Conditions*

Transfer Type	Abnormal Condition	Bridge Response
Read	Illegal burst type	SIB interrupt is asserted. SLVERR response given with arbitrary read data.
Write	Illegal burst type	SIB interrupt is asserted. Write data is discarded. SLVERR response given.
Read	Unexpected completion	SUC interrupt is asserted. Completion is discarded.
Read	Unsupported Request status returned	SUR interrupt is asserted. DECERR response given with arbitrary read data.
Read	Completion timeout	For PCIe Memory Read request, AXI Bridge for PCIe Gen3 IP provides SLVERR response, while DMA/Bridge Subsystem for PCIe in Bridge Mode IP provides OKAY response with 0s data on the AXI4 memory mapped bus.
Read	Poison bit in completion	Completion data is discarded. SEP interrupt is asserted. SLVERR response given with arbitrary read data.
Read	Completer Abort (CA) status returned	SCA interrupt is asserted. SLVERR response given with arbitrary read data.

## Master Bridge Abnormal Conditions

The following sections describe the manner in which the master bridge handles abnormal conditions.

### ***AXI DECERR Response***

When the master bridge receives a DECERR response from the AXI bus, the request is discarded and the Master DECERR (MDE) interrupt is asserted. If the request was non-posted, a completion packet with the Completion Status = Unsupported Request (UR) is returned on the bus for PCIe.

### ***AXI SLVERR Response***

When the master bridge receives a SLVERR response from the addressed AXI slave, the request is discarded and the Master SLVERR (MSE) interrupt is asserted. If the request was non-posted, a completion packet with the Completion Status = Completer Abort (CA) is returned on the bus for PCIe.

### ***Max Payload Size for PCIe, Max Read Request Size or 4K Page Violated***

When the master bridge receives a SLVERR response from the addressed AXI slave, the request is discarded and the Master SLVERR (MSE) interrupt is asserted. If the request was non-posted, a completion packet with the Completion Status = Completer Abort (CA) is returned on the bus for PCIe.

### ***Completion Packets***

When the `MAX_READ_REQUEST_SIZE` is greater than the `MAX_PAYLOAD_SIZE`, a read request for PCIe can ask for more data than the master bridge can insert into a single completion packet. When this situation occurs, multiple completion packets are generated up to `MAX_PAYLOAD_SIZE`, with the Read Completion Boundary (RCB) observed.

### ***Poison Bit***

When the poison bit is set in a transaction layer packet (TLP) header, the payload following the header is corrupted. When the master bridge receives a memory request TLP with the poison bit set, it discards the TLP and asserts the Master Error Poison (MEP) interrupt strobe.

### ***Zero Length Requests***

When the master bridge receives a read request with the `Length = 0x1`, `FirstBE = 0x00`, and `LastBE = 0x00`, it responds by sending a completion with `Status = Successful Completion`.

When the master bridge receives a write request with the `Length = 0x1`, `FirstBE = 0x00`, and `LastBE = 0x00` there is no effect.

Table 62: Master Bridge Response to Abnormal Conditions

Transfer Type	Abnormal Condition	Bridge Response
Read	DECERR response	MDE interrupt strobe asserted. Completion returned with Unsupported Request status.
Write	DECERR response	MDE interrupt strobe asserted.
Read	SLVERR response	MSE interrupt strobe asserted. Completion returned with Completer Abort status.
Write	SLVERR response	MSE interrupt strobe asserted.
Write	Poison bit set in request	MEP interrupt strobe asserted. Data is discarded.
Read	DECERR response	MDE interrupt strobe asserted. Completion returned with Unsupported Request status.
Write	DECERR response	MDE interrupt strobe asserted.

## Link Down Behavior

The normal operation of the Bridge core is dependent on the integrated block for PCIe establishing and maintaining the point-to-point link with an external device for PCIe. If the link has been lost, it must be re-established to return to normal operation.

When a Hot Reset is received by the Bridge core, the link goes down and the PCI Configuration Space must be reconfigured.

Initiated AXI4 write transactions that have not yet completed on the AXI4 bus when the link goes down have a SLVERR response given and the write data is discarded. Initiated AXI4 read transactions that have not yet completed on the AXI4 bus when the link goes down have a SLVERR response given, with arbitrary read data returned.

Any MemWr TLPs for PCIe that have been received, but the associated AXI4 write transaction has not started when the link goes down, are discarded.

## Endpoint

When configured to support Endpoint functionality, the Bridge core fully supports Endpoint operation as supported by the underlying block. There are a few details that need special consideration. The following subsections contain information and design considerations about Endpoint support.

## Interrupts

Interrupt capabilities are provided by the underlying PCI Express® solution IP. For additional information, see the *Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide (PG023)*, the *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)*, and the *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)*.

## AXI Bridge for PCIe Gen3

The Interrupts modes in the following section applies to AXI Bridge for PCIe Gen3 only. For DMA/Bridge Subsystem for PCIe in Bridge Mode, go to the next section.

Multiple interrupt modes can be configured during IP configuration, however only one interrupt mode is used at runtime. If multiple interrupt modes are enabled by the host after PCI bus enumeration at runtime, the core uses the MSI interrupt over Legacy interrupt. Both MSI and Legacy interrupt modes are sent using the same `int_msi_*` interface, and the core automatically picks the best available interrupt mode at runtime. MSI-X is implemented externally to the core and uses a separate `cfg_interrupt_msix_*` interface. The core does not prevent the use of MSI-X at any time even when other interrupt modes are enabled, however Xilinx recommends the use of MSI-X interrupt solely if enabled over other interrupt modes even if they are all enabled at runtime.

### Legacy Interrupts

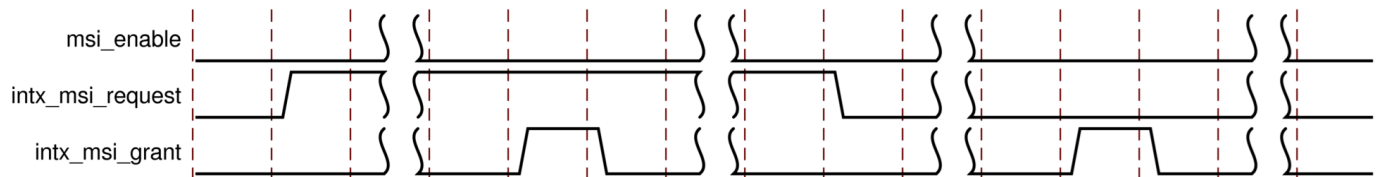
Asserting `intx_msi_request` when legacy interrupts are enabled causes the IP to issue a legacy interrupt over PCIe. After a `intx_msi_grant` signal is asserted, it must remain asserted until the `intx_msi_grant` signal is asserted and the interrupt has been serviced and cleared by the host. The `intx_msi_grant` assertion indicates the requested interrupt has been sent on the PCIe block. The Message sent is based on the value of the `PFO_INTERRUPT_PIN` parameter, which is configurable during core customization in the Vivado Design Suite. Keeping `intx_msi_grant` asserted ensures the interrupt pending register within the IP remains asserted when queried by the host's Interrupt Service Routine (ISR) to determine the source of interrupts. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the host software when an interrupt is serviced.

After the `intx_msi_request` signal is deasserted, it cannot be reasserted until the `intx_msi_grant` signal has been asserted for a second time. This indicates the deassertion message for the legacy interrupt has been sent over PCIe. After a second `intx_msi_grant` has occurred, the `intx_msi_request` signal can be reasserted to generate another legacy interrupt.

The following figure shows the legacy interrupts.

This figure shows only the handshake between `intx_msi_request` and `intx_msi_grant`. The user application might not clear or service the interrupt immediately, in which case, you must keep `intx_msi_request` asserted past `intx_msi_grant`.

Figure 26: Legacy Interrupts



## Related Information

[Bridge Parameters](#)

## MSI Interrupts

Asserting `intx_msi_request` causes the generation of an MSI interrupt if MSI is enabled.

After `intx_msi_request` is asserted, it should be de-asserted immediately in the next clock cycle. The `intx_msi_grant` assertion indicates the requested interrupt has been sent on the PCIe block. The vector number being used in the interrupt packet will be determined based on the value provided at the `msi_vector_num[4:0]` signal. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the host software when an Interrupt is serviced.

The `intx_msi_request` signal is positive-edge detected and synchronous to `axi_aclk`. `intx_msi_request` must be de-asserted for at least one `axi_aclk` cycle before re-asserting it to send another MSI interrupt on the same vector or on a different vector.

The MSI vector number being used must not exceed the number of MSI vectors being enabled by the host. This information can be queried from `msi_vector_width[2:0]` signal after the host enumerated and enabled MSI interrupt at runtime. The encoding of `msi_vector_width[2:0]` signal is shown in the following table.

Table 63: MSI Vectors Enabled in Message Control Register

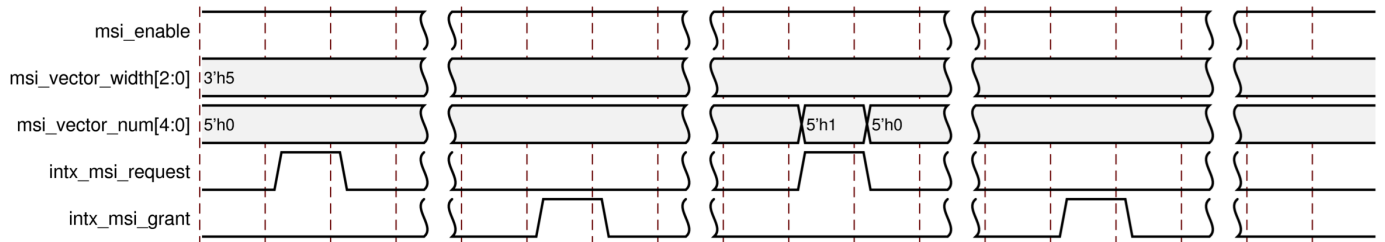
Value	Number of Messages Requested	Output Signal, MSI_Vector_Width (2:0)
000	1	000
001	2	001



Table 63: MSI Vectors Enabled in Message Control Register (cont'd)

Value	Number of Messages Requested	Output Signal, MSI_Vector_Width (2:0)
010	4	010
011	8	011
100	16	100
101	32	101

Figure 27: MSI Interrupts



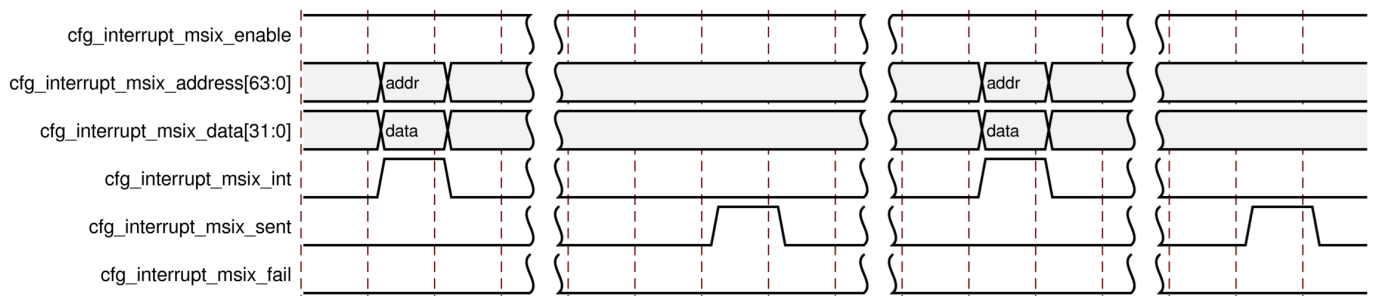
## External MSI-X Interrupts

**Note:** This interrupt mode is only available in AXI Bridge for PCI Express Gen3.

The core supports the MSI-X interrupt and its signaling. The MSI-X vector table and the MSI-X Pending Bit Array need to be implemented as part of the user logic, by claiming a BAR aperture. The External MSI-X interrupts mode is enabled when you set the MSI-X Implementation Location option to External in the PCIe Misc Tab.

To send MSI-X interrupt, user logic must use `cfg_interrupt_msix_*` interface instead of the `intx_msi_*` interface. The signaling requirement is the same as defined in the UltraScale Devices Gen3 Integrated Block for PCIe core as shown below.

Figure 28: External MSI-X Interrupts



## DMA/Bridge Subsystem for PCIe in Bridge Mode

The Interrupts modes in the following section applies to DMA/Bridge Subsystem for PCIe in Bridge Mode only. For AXI Bridge for PCIe Gen3, go to the previous section.

Multiple interrupt modes can be configured during IP configuration, however only one interrupt mode is used at runtime. If multiple interrupt modes are enabled by the host after PCI bus enumeration at runtime, MSI-X interrupt takes precedence over MSI interrupt, and MSI interrupt takes precedence over Legacy interrupt. All of these interrupt modes are sent using the same `usr_irq_*` interface and the core automatically picks the best available interrupt mode at runtime.

### Legacy Interrupts

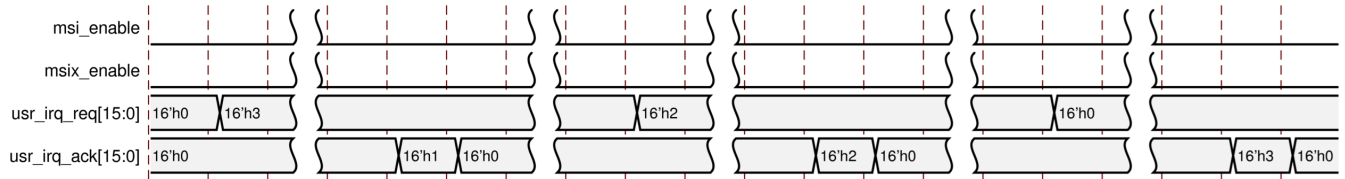
Asserting one or more bits of `usr_irq_req` when legacy interrupts are enabled causes the IP to issue a legacy interrupt over PCIe. Multiple bits may be asserted simultaneously but each bit must remain asserted until the corresponding `usr_irq_ack` bit has been asserted. After a `usr_irq_req` bit is asserted, it must remain asserted until the corresponding `usr_irq_ack` bit is asserted and the interrupt has been serviced and cleared by the Host. The `usr_irq_ack` assertion indicates the requested interrupt has been sent on the PCIe block. This will ensure interrupt pending register within the IP remains asserted when queried by the Host's Interrupt Service Routine (ISR) to determine the source of interrupts. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the Host software when an interrupt is serviced.

After the `usr_irq_req` bit is deasserted, it cannot be reasserted until the corresponding `usr_irq_ack` bit has been asserted for a second time. This indicates the deassertion message for the legacy interrupt has been sent over PCIe. After a second `usr_irq_ack` occurred, the `usr_irq_req` wire can be reasserted to generate another legacy interrupt.

The `usr_irq_req` bit can be mapped to legacy interrupt INTA, INTB, INTC, INTD through the configuration registers. The following figure shows the legacy interrupts.

This figure shows only the handshake between `usr_irq_req` and `usr_irq_ack`. The user application might not clear or service the interrupt immediately, in which case, you must keep `usr_irq_req` asserted past `usr_irq_ack`.

Figure 29: Legacy Interrupts



### MSI and Internal MSI-X Interrupts

**Note:** Internal MSI-X Interrupt mode is only available in DMA/Bridge Subsystem for PCIe in Bridge Mode.

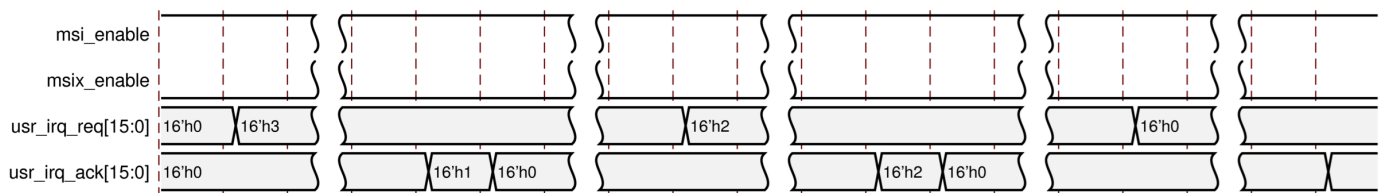
Asserting one or more bits of `usr_irq_req` causes the generation of an MSI or MSI-X interrupt if MSI or MSI-X is enabled. If both MSI and MSI-X capabilities are enabled, an MSI-X interrupt is generated. The Internal MSI-X interrupts mode is enabled when you set the MSI-X Implementation Location option to Internal in the PCIe Misc Tab.

After a `usr_irq_req` bit is asserted, it must remain asserted until the corresponding `usr_irq_ack` bit is asserted and the interrupt has been serviced and cleared by the Host. The `usr_irq_ack` assertion indicates the requested interrupt has been sent on the PCIe block. This will ensure the interrupt pending register within the IP remains asserted when queried by the Host's Interrupt Service Routine (ISR) to determine the source of interrupts. You must implement a mechanism in the user application to know when the interrupt routine has been serviced. This detection can be done in many different ways depending on your application and your use of this interrupt pin. This typically involves a register (or array of registers) implemented in the user application that is cleared, read, or modified by the Host software when an Interrupt is serviced.

Configuration registers are available to map `usr_irq_req` and DMA interrupts to MSI or MSI-X vectors. For MSI-X support, there is also a vector table and PBA table. The following figure shows the MSI interrupt.

This figure shows only the handshake between `usr_irq_req` and `usr_irq_ack`. Your application might not clear or service the interrupt immediately, in which case, you must keep `usr_irq_req` asserted past `usr_irq_ack`.

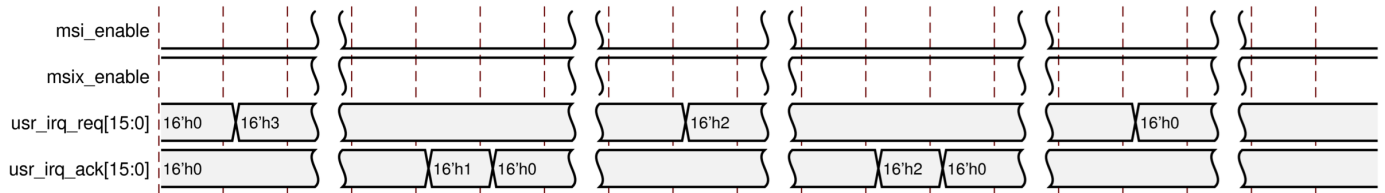
Figure 30: MSI Interrupts



The following figure shows the MSI-X interrupt.

This figure shows only the handshake between `usr_irq_req` and `usr_irq_ack`. Your application might not clear or service the interrupt immediately, in which case, you must keep asserted past `usr_irq_ack`.

Figure 31: MSI-X Interrupts



## Root Port

When configured to support Root Port functionality, the Bridge core fully supports Root Port operation as supported by the underlying block. There are a few details that need special consideration. The following subsections contain information and design considerations about Root Port support.

### Power Limit Message TLP

The Bridge core automatically sends a Power Limit Message TLP when the Master Enable bit of the Command Register is set. The software must set the Requester ID register before setting the Master Enable bit to ensure that the desired Requester ID is used in the Message TLP.

### Root Port Configuration Read

When an ECAM access is performed to the primary bus number, self-configuration of the integrated block for PCIe is performed. A PCIe configuration transaction is not performed and is not presented on the link. When an ECAM access is performed to the bus number that is equal to the secondary bus value in the Enhanced PCIe Type 1 configuration header, then Type 0 configuration transactions are generated.

When an ECAM access is attempted to a bus number that is in the range defined by the secondary bus number and subordinate bus number range (not including secondary bus number), then Type 1 configuration transactions are generated. The primary, secondary and subordinate bus numbers are written and updated by Root Port software to the Type 1 PCI Configuration Header of the Bridge core in the enumeration procedure.

When an ECAM access is attempted to a bus number that is out of the range defined by the secondary bus\_number and subordinate bus number, the bridge does not generate a configuration request and signal a SLVERR response on the AXI4 bus.

When an Unsupported Request (UR) response is received for a configuration read request, all ones are returned on the AXI4 bus to signify that a device does not exist at the requested device address. It is the responsibility of the software to ensure configuration write requests are not performed to device addresses that do not exist. However, the Bridge core asserts SLVERR response on the AXI4 bus when a configuration write request is performed on device addresses that do not exist or a UR response is received.

## Root Port BAR

Root Port BAR does not support packet filtering (all TLPs received from PCIe link are forwarded to the user logic), however Address Translation can be configured to enable or disable, depending on the IP configuration.

During core customization in the Vivado® Design Suite, when there is no BAR enabled, RP passes all received packets to the user application without address translation or address filtering.

When BAR is enabled, by default the BAR address starts at 0x0000\_0000 unless programmed separately. Any packet received from the PCIe® link that hits a BAR is translated according to the PCIE-to-AXI Address Translation rules.

**Note:** The IP must not receive any TLPs outside of the PCIe BAR range from the PCIe link when RP BAR is enabled. If this rule cannot be enforced, it's recommended that the PCIe BAR is disabled and do address filtering and/or translation outside of the IP.

The Root Port BAR customization options in the Vivado Design Suite are found in the PCIe BARs Tab.

### Related Information

[PCIe BARs Tab](#)

[PCIe BARs Tab](#)

## Configuration Transaction Timeout

Configuration transactions are non-posted transactions. The Bridge core has a timer for timeout termination of configuration transactions that have not completed on the PCIe link. SLVERR is returned when a configuration timeout occurs. Timeouts of configuration transactions are flagged by an interrupt as well.

## Abnormal Configuration Transaction Termination Responses

Responses to abnormal terminations to configuration transactions are shown in the following table.

*Table 64: Responses of Bridge to Abnormal Configuration Terminations*

Transfer Type	Abnormal Condition	Bridge Response
Config Read or Write	Bus number not in the range of primary bus number through subordinate bus number.	SLVERR response is asserted.
Config Read or Write	Completion timeout.	For PCIe Configuration Read/Write request, an OKAY response and 0s data are given on the AXI4 memory mapped bus.
Config Write	Bus number in the range of secondary bus number through subordinate bus number and UR is returned.	SLVERR response is asserted.

## Receiving Interrupts

In Root Port mode, you can choose one of the two ways to handle incoming interrupts;

- Legacy Interrupt FIFO mode: Legacy Interrupt FIFO mode is the default. It is available in earlier Bridge IP variants and versions, and will continue to be available. Legacy Interrupt FIFO mode is geared towards compatibility for legacy designs.
- Interrupt Decode mode: Interrupt Decode mode is available in the CPM AXI Bridge. Interrupt Decode mode can be used to mitigate Interrupt FIFO overflow condition which can occur in a design that receives interrupts at a high rate and avoids the performance penalty incurred when such condition occurs.

To enable Legacy Interrupt FIFO mode:

```
set_property -dict [list CONFIG.msi_rx_pin_en {false}] [get_ips <ip_name>]
```

To enable Interrupt Decode mode:

```
set_property -dict [list CONFIG.msi_rx_pin_en {true}] [get_ips <ip_name>]
```

If you are customizing and generating the core in the Vivado® IP integrator, replace `get_ips` with `get_bd_cells`.

## Legacy Interrupt FIFO Mode

### *Legacy INTx Interrupt*

When the IP receives an INTx interrupt, the Interrupt Decode register bit[16] is set. If the Interrupt Mask register bit[16] is also set, the `interrupt_out` pin is asserted. After receiving this interrupt, the user application must follow the following procedure to service the interrupt:

1. *Optional:* Write 0 to the Interrupt Mask register bit [16] to deassert the `interrupt_out` pin while interrupt is being serviced.
2. Read the Root Port Status/Control Register bit [18] to check if it is not empty.
3. Read the Root Port Status/Control Register bit [19] to check if it has overflowed.
4. If the interrupt FIFO is not empty, read Root Port Interrupt FIFO Read Register 1 to check which interrupt line is serviced, and whether this is an Assertion or Deassertion message.
5. Write 1 to the Root Port Interrupt FIFO Read Register 1 bit [31] to remove the interrupt that the user has just read from the FIFO.
6. Repeat from step 2 to step 5 until the FIFO is indicated as empty.
7. If at any time during this process the FIFO is indicated as overflowed (status from step 2), the user application must check any interrupt line that has not been serviced to check for any pending interrupt on that line. Failure to do this before continuing may leave some interrupt line unserved.
8. Write 1 to the Interrupt Decode register bit [16] to clear the INTx interrupt bit.
9. If step 1 is executed, write 1 to the Interrupt Mask register bit [16] to re-enable the `interrupt_out` pin for future INTx interrupt.

## MSI Interrupt

The IP will decode the MSI interrupt based on the value programmed in Root Port MSI Base Register 1 and Root Port MSI Base Register 2. Any Memory Write TLP received from the link with an address that falls within a 4 Kb window from the base address programmed in those registers will be treated as an MSI interrupt, and will not be forwarded to the M\_AXI(B) interface. When an MSI interrupt is received, the Interrupt Decode register bit[17] will be set. If the Interrupt Mask register bit[17] is also set, the `interrupt_out` pin is asserted. After receiving this interrupt, the user application must follow the following procedure to service the interrupt:

1. *Optional:* Write 0 to the Interrupt Mask register bit [17] to deassert the `interrupt_out` pin while the interrupt is being serviced.
2. Read the Root Port Status/Control Register bit [18] to check if it is not empty.
3. Read the Root Port Status/Control Register bit [19] to check if it has overflowed.

4. If the interrupt FIFO is not empty, read the Root Port Interrupt FIFO Read Register 2 to check MSI Message Data from the received MSI interrupt. This is used by the user application to determine the interrupt vector number and can also be used to determine the source of the interrupt.
5. Write 1 to the Root Port Interrupt FIFO Read Register 1 bit [31] to remove the interrupt user has just read from the FIFO.
6. Repeat from step 2 until the FIFO is indicated as empty.
7. If at any time during this process, the FIFO was indicated as overflowed (status from step 2), the user application must check any unserved interrupt vectors to check for any pending interrupts on that line. Failure to do this before continuing can leave some interrupt vector unserved.
8. Write 1 to the Interrupt Decode Register bit [17] to clear the MSI interrupt bit.
9. If step 1 was executed, write 1 to the Interrupt Mask Register bit [17] to re-enable the `interrupt_out` pin for future MSI interrupts.

### ***MSI-X Interrupt***

All MSI-X interrupts must be decoded by the user application externally to the IP. To do this, set all of their Endpoints to use an MSI-X address that falls outside of the range of the 4Kb window from the base address programmed in the Root Port MSI Base Register 1 and Root Port MSI Base Register 2. All MSI-X interrupts will be forwarded to the M\_AXI(B) interface.

All TLPs forwarded to M\_AXI(B) interface are subject to the PCIe-to-AXI Address translation.

## **Interrupt Decode Mode**

### ***Legacy INTx Interrupt***

When the IP has received an INTx interrupt, the Root Port Interrupt Decode 2 register is set. If the Root Port Interrupt Decode 2 Mask register is also set, the `interrupt_out` pin is asserted. After receiving this interrupt, the user application must follow this procedure to service the interrupt:

1. Optional: Write 0 to the Interrupt Decode 2 Mask register to deassert an interrupt line while the interrupt is being serviced.
2. Read the Root Port Interrupt Decode 2 register to check which interrupt line is currently asserted.
3. Repeat step 2 until all interrupt lines are deasserted. The interrupt line is automatically cleared when the IP receives the INTx Deassert Message corresponding to that interrupt line.
4. If step 1 was executed, write 1 to the Interrupt Decode 2 Mask register to re-enable an interrupt line for future INTx interrupt.



## MSI Interrupt

The IP decodes the MSI interrupt based on the value programmed in Root Port MSI Base Register 1 and Root Port MSI Base Register 2. Any Memory Write TLPs received from the link with an address that falls within the 4 Kb window from the base address programmed in those registers will be treated as MSI interrupt, and will not be forwarded to the M\_AXI(B) interface.

**Note:** MSI Message Data [5:0] will always be decoded as MSI Message vector regardless of how many vectors are enabled at your Endpoint.

When an MSI interrupt is received, the Root Port MSI Interrupt Decode 1 or Root Port MSI Interrupt Decode 2 register is set. If the Root Port MSI Interrupt Decode 1 or Root Port MSI Interrupt Decode 2 register is also set, the `interrupt_out_msi_vec*` pins are asserted. `interrupt_out_msi_vec0to31` corresponds to MSI vector 0 - 31, and `interrupt_out_msi_vec32to63` corresponds to MSI vector 32 - 63. After receiving this interrupt, the user application must follow this procedure to service the interrupt:

1. Optional: Write 0 to the Root Port MSI Interrupt Decode 1 or 2 Mask register to deassert the `interrupt_out_msi_vec*` pins while the interrupt is being serviced.
2. Read the Root Port MSI Interrupt Decode 1 or 2 register to check which interrupt vector is asserted.
3. Write 1 to the Root Port MSI Interrupt Decode 1 or 2 register to clear the MSI interrupt bit.
4. If step 1 was executed, write 1 to the Root Port MSI Interrupt Decode 1 or 2 Mask register bit to re-enable the `interrupt_out_msi_vec*` pins for future MSI interrupts.

## MSI-X Interrupt

All MSI-X interrupts must be decoded by the user application externally to the IP. To do this, the user application must set all Endpoints to use an MSI-X address that falls outside of the range of the 4Kb window from the base address programmed in the Root Port MSI Base Register 1 and the Root Port MSI Base Register 2. All MSI-X interrupts are forwarded to the M\_AXI(B) interface.

All TLPs forwarded to M\_AXI(B) interface are subject to PCIe-to-AXI Address translation.

---

# Tandem Configuration

Tandem Configuration utilizes a two-stage methodology that enables the IP to meet the configuration time requirements indicated in the PCI Express Specification. Multiple use cases are supported with this technology:

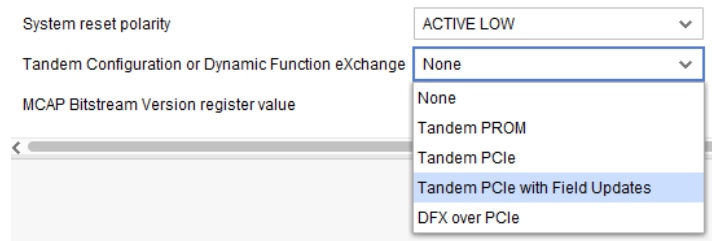
- Tandem PROM: Load the single two-stage bitstream from the flash.
- Tandem PCIe: Load the first stage bitstream from flash, and deliver the second stage bitstream over the PCIe link to the MCAP.

- **Tandem (PCIe) with Field Updates:** After a Tandem PROM (UltraScale only) or Tandem PCIe (UltraScale or UltraScale+) initial configuration, update the entire user design while the PCIe link remains active. The update region (floorplan) and design structure are predefined, and Tcl scripts are provided.
- **Tandem + Dynamic Function eXchange:** This is a more general case of Tandem Configuration followed by Dynamic Function eXchange (DFX) of any size or number of dynamic regions.
- **Dynamic Function eXchange over PCIe:** This is a standard configuration followed by DFX, using the PCIe / MCAP as the delivery path of partial bitstreams.

To enable any of these capabilities, select the appropriate option when customizing the core. In the Basic tab:

1. Change the Mode to **Advanced**.
2. Change the Tandem Configuration or Dynamic Function eXchange option according to your particular case:
  - **Tandem** for Tandem PROM, Tandem PCIe or Tandem + Dynamic Function eXchange use cases.
  - **Tandem with Field Updates ONLY** for the predefined Field Updates use case.
  - **DFX over PCIe** to enable the MCAP link for Dynamic Function eXchange, without enabling Tandem Configuration.

Figure 32: Tandem Configuration or Dynamic Function eXchange Option



Tandem Configuration features are available for the Bridge core for all supported UltraScale and UltraScale+ devices.

For complete information about Tandem Configuration, including supported devices, required PCIe block locations, design flow examples, requirements, restrictions and other considerations, see [Tandem Configuration](#) in the *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)* and *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)*. For information on Dynamic Function eXchange, see the *Vivado Design Suite User Guide: Dynamic Function eXchange (UG909)*.

# Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the Vivado® design flows and the Vivado 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 Subsystem

This section includes information about using the Vivado® Design Suite to customize and generate the core.

If you are customizing and generating the core in the Vivado® IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#)) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP, or select the **Customize IP** command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* ([UG896](#)) and the *Vivado Design Suite User Guide: Getting Started* ([UG910](#)).

Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

## Customizing the Core

The Bridge core customization parameters are described in the following sections.

- For Virtex®-7 XT and UltraScale™ devices, see [AXI Bridge for PCIe Gen3](#).
- For UltraScale+™ devices, see [DMA/AXI Bridge Subsystem for PCI Express in AXI Bridge Mode](#).

## AXI Bridge for PCIe Gen3

### Basic Mode Tab

The initial customization screen shown in the following figure is used to define the basic parameters for the core, including the component name, PCIe® configuration, AXI parameters, and reference clock. The following figures show the parameters available for the Advanced Mode.

In the Basic Mode tab, the additional parameters available in Advanced Mode are applicable to UltraScale™ architecture devices only.

Figure 33: Basics Parameter Settings: Basic Mode Selected

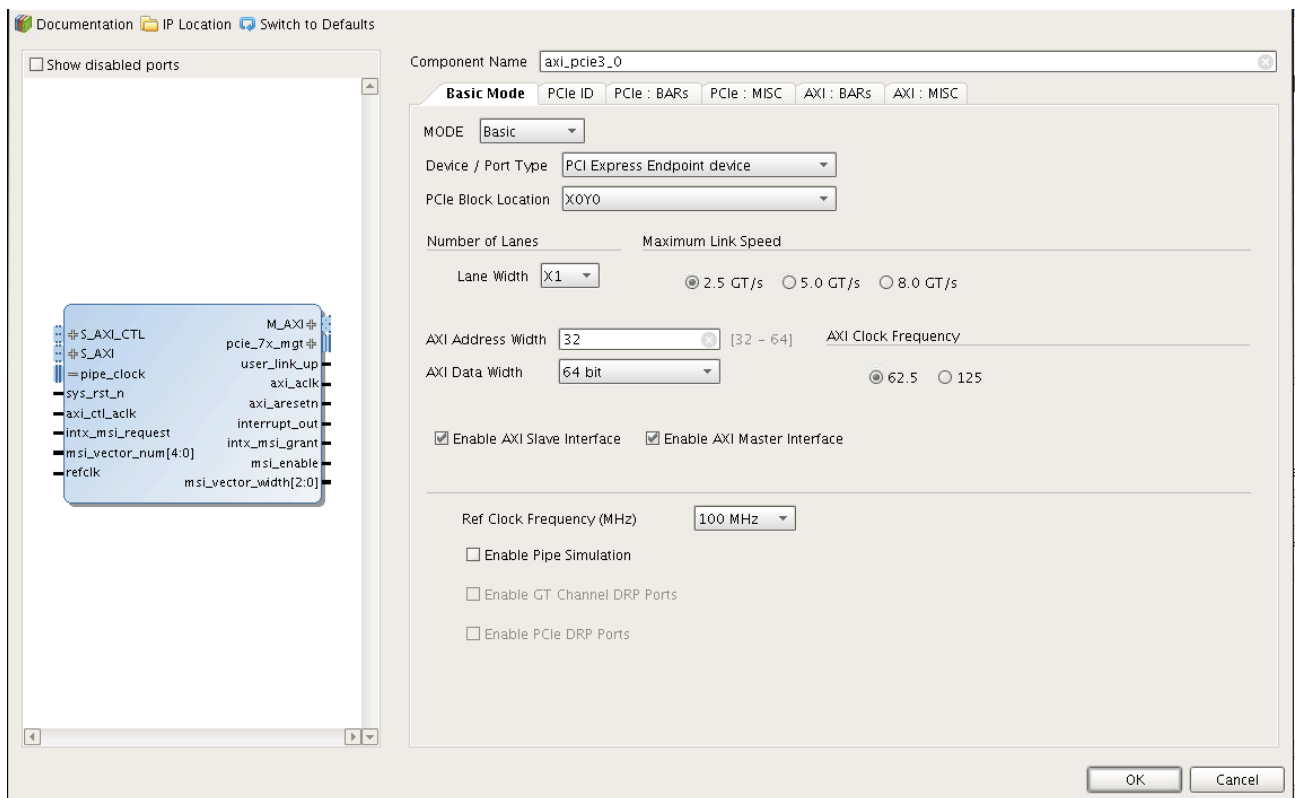


Figure 34: Basic Parameter Settings: Advanced Mode Selected

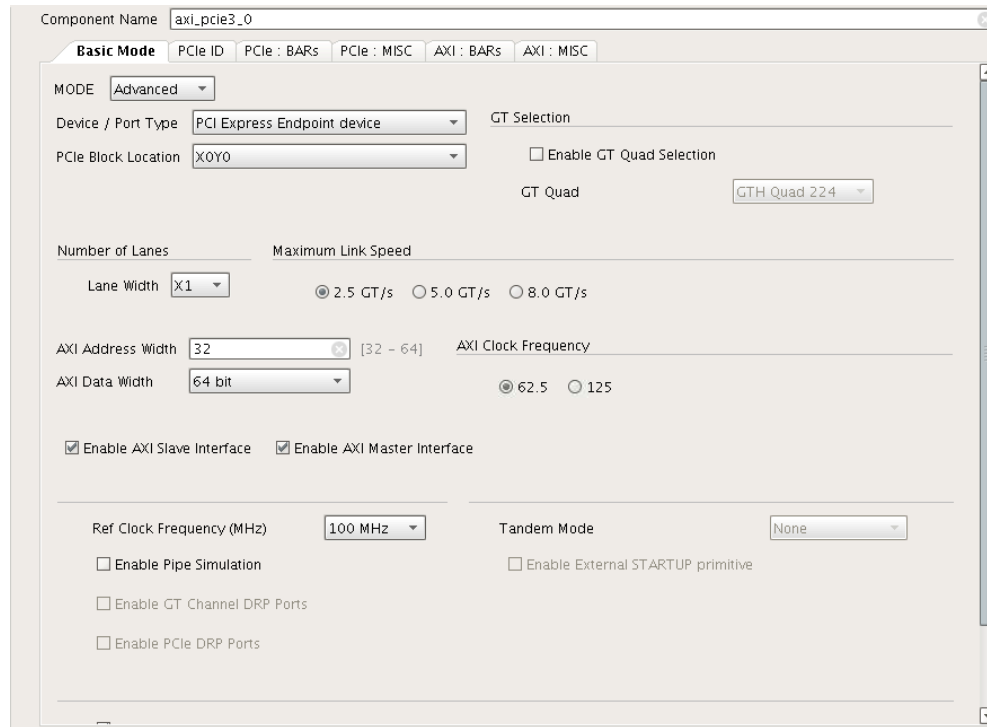


Figure 35: Additional Basic Parameter Settings: Advanced Mode Selected



- **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 “\_.”

**Note:** The name cannot be the same as a core module name; for example, "axi\_pcie" is a reserved name.

- **Mode:** Allows you to select the Basic or Advanced mode of the configuration of core.
- **Device / Port Type:** Indicates the PCI Express logical device type.
- **PCIe Block Location:** Selects from the available integrated blocks to enable generation of location-specific constraint files and pinouts. This selection is used in the default example design scripts.
- **Enable GT Quad Selection:** This parameter is used to enable the device/package migration. Applicable to UltraScale™ devices only.
- **Number of Lanes:** The core requires the selection of the initial lane width. Wider lane width cores can train down to smaller lane widths and consume more FPGA resource.

- **Maximum Link Speed:** Indicates the maximum link speed supported by the design. Higher link speed cores are capable of training to lower link speeds and run at a higher clock frequency.
- **AXI Address Width:** Indicates the AXI address width for the `S_AXI` and `M_AXI` interfaces, but does not affect the address width of the `S_AXI_CTL` interface.
- **AXI Data Width:** Indicates the AXI data width for the `S_AXI` and `M_AXI` interfaces, but does not affect the data width of the `S_AXI_CTL` interface.
- **AXI Clock Frequency:** Indicates the clock frequency that will be generated on the `axi_aclk` output. All AXI interfaces and a majority of the core outputs are synchronous to this clock.
- **Enable AXI Slave Interface:** Allows the slave bridge to be enabled or disabled as desired by the system design. If only the master bridge is used the slave bridge can be disabled to conserve FPGA resources.
- **Enable AXI Master Interface:** Allows the master bridge to be enabled or disabled as desired by the system design. If only the slave bridge is used, the master bridge can be disabled to conserve FPGA resources.
- **Reference Clock Frequency:** Selects the frequency of the reference clock provided on the `refclk` reference clock input. This reference clock input corresponds to `refclk` for Virtex®-7 devices and `sys_clk_gt` for UltraScale™ devices.
- **Enable Pipe Simulation:** When selected, this option generates the core that can be simulated with PIPE interfaces connected.
- **Enable GT Channel DRP Ports:** When checked, enables the GT channel DRP interface.
- **Enable PCIe DRP Ports:** When checked, enables the PCIe DRP interface.
- **Tandem Mode:** For supported devices only, this option allow you to choose the Tandem Configuration mode: None, Tandem PROM and Tandem PCIe.
- **Enable External STARTUP primitive:** When checked, generates the STARTUP primitive external to the IP.
- **Use the dedicated PERST routing resources:** Enables `sys_rst` dedicated routing for applicable UltraScale PCIe locations. This option is not applicable for Virtex-7 XT and UltraScale+ devices.
- **System Reset polarity :** This parameter is used to set the polarity of the `sys_rst` `ACTIVE_HIGH` or `ACTIVE_LOW`.
- **CORE CLOCK Frequency:** Available only when an UltraScale device is selected.

This parameter allows you to select the core clock frequencies.

For Gen3 link speed:

- The values of 250 MHz and 500 MHz are available for selection for speed grade -2 or -3 and link width other than x8. For this configuration, this parameter is available when Advanced mode is selected.
- For speed grades -2 or -3 and link width of x8, this parameter defaults to 500 MHz and is not available for selection.
- For -1 speed grade (-1, -1L, -1LV, -1H and -1HV) and link width other than x8, this parameter defaults to 250 MHz and is not available for selection.

For Gen1 and Gen2 link speeds:

- This parameter defaults to 250 MHz and is not available for selection.

**Note:** When -1 or -1L, -1LV, -1H and -1HV speed grade is selected and non production parts of XCKU060 (ES2), XCKU115 (ES2) and VU440 (ES2) are selected, this parameter defaults to 250 MHz and is not available for selection.

## PCIe ID Tab

The PCIe® Identity parameters are shown in the following figure. These settings customize the IP initial values and device class code.

Figure 36: PCIe ID Settings

Component Name: axi\_pcie3\_0

Board Basic **PCIe ID** PCIe : BARs PCIe : MISC AXI : BARs AXI : MISC

ID Initial Values

Vendor ID: 10EE

Device ID: 8011

Revision ID: 00

Subsystem Vendor ID: 10EE

Subsystem ID: 0007

Class Code Lookup Assistant

Use Class Code Lookup Assistant

Base Class Menu: Simple communication controllers

Base Class Value: 05 Range: 00..FF

Sub Class Interface Menu: Generic XT compatible serial controller

Sub Class Value: 80 Range: 00..FF

Interface Value: 00 Range: 00..FF

Class Code: 058000 Range: 000000..

OK Cancel

## 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 your vendor identification number here. `FFFFh` is reserved.

- **Device ID:**

A unique identifier for the application; the default value, which depends on the configuration selected, is `70<link speed><link width>h`. This field can be any value; change this value for the application.

The Device ID parameter is evaluated based on:

1. the device family (9 for UltraScale+, 8 for UltraScale, 7 for 7 series devices),
2. EP or RP mode,
3. Link width, and
4. Link speed.

If any of the above values are changed, the Device ID value will be re-evaluated, replacing the previous set value.

It is always recommended that the link width, speed and Device Port type be changed first and then the Device ID value.

- **Revision ID:** Indicates the revision of the device or application; an extension of the Device ID. The default value is `00h`; enter values 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. The Vivado allows you to either enter the 24-bit value manually (default) by either selecting the **Enter Class Code Manually** checkbox or using the Class Code lookup assistant to populate the field. De-select the checkbox to enable the Class Code assistant.

- **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 in the [PCI-SIG Specifications](#)

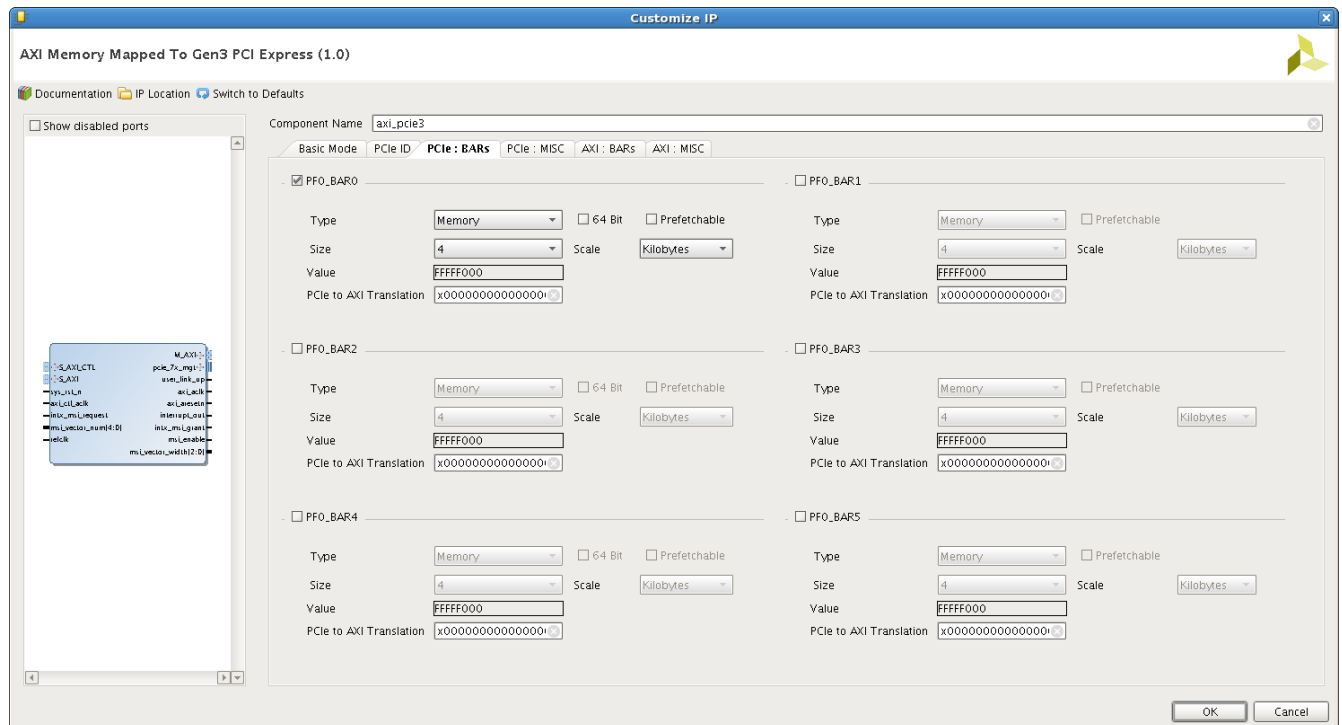
### Class Code Look-up Assistant

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.

### PCIe BARs Tab

The PCI Express Base Address Registers (BARs) parameters shown in the following figure set 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, as described previously.

Figure 37: PCIe Base Address Register



### Base Address Register Overview

The Bridge core in Endpoint configuration supports up to six 32-bit BARs or three 64-bit BARs. The AXI Bridge for PCI Express in Root Port configuration supports up to two 32-bit BARs or one 64-bit BAR.

BARs can be one of two sizes. BARs 0, 2, and 4 can be either 32-bit or 64-bit addressable. BARs 1, 3, and 5 can only be 32-bit addressable and are disabled if the previous BAR is enabled as 64-bit.

- **32-bit BARs:** The address space can be as small as 4 kilobytes or as large as 2 gigabytes. Used for Memory to I/O.
- **64-bit BARs:** The address space can be as small as 4 kilobytes or as large as 256 gigabytes. Used for Memory only.

All BAR registers share these options:

- **Checkbox:** Click the checkbox to enable BAR. Deselect the checkbox to disable BAR.
- **Type:** BARs can be Memory apertures only. Memory BARs can be either 64-bit or 32-bit. Prefetch is enabled for 64-bit and not enabled for 32-bit. When a BAR is set as 64 bits, it uses the next BAR for the extended address space, making it 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 65: BAR Size Ranges for Device Configuration*

PCIe Device/Port Type	BAR Type	BAR Size Range
PCI Express Endpoint	32-bit Memory	4 Kilobytes - 2 Gigabytes
	64-bit Memory	4 Kilobytes - 256 Gigabytes

- **Prefetchable:** Identifies the ability of the memory space to be prefetched. This can only be enabled for 64-bit addressable bars.
- **Value:** The value assigned to BAR.
- **PCIe to AXI Translation:** This text field should be set to the appropriate value to perform the translation from the PCI Express base address to the desired AXI Base Address.

For more information about managing the Base Address Register settings, see Managing Base Address Register Settings.

### Managing Base Address Register Settings

Memory indicates that the address space is defined as memory aperture. The base address register only responds to commands that access the specified address space.

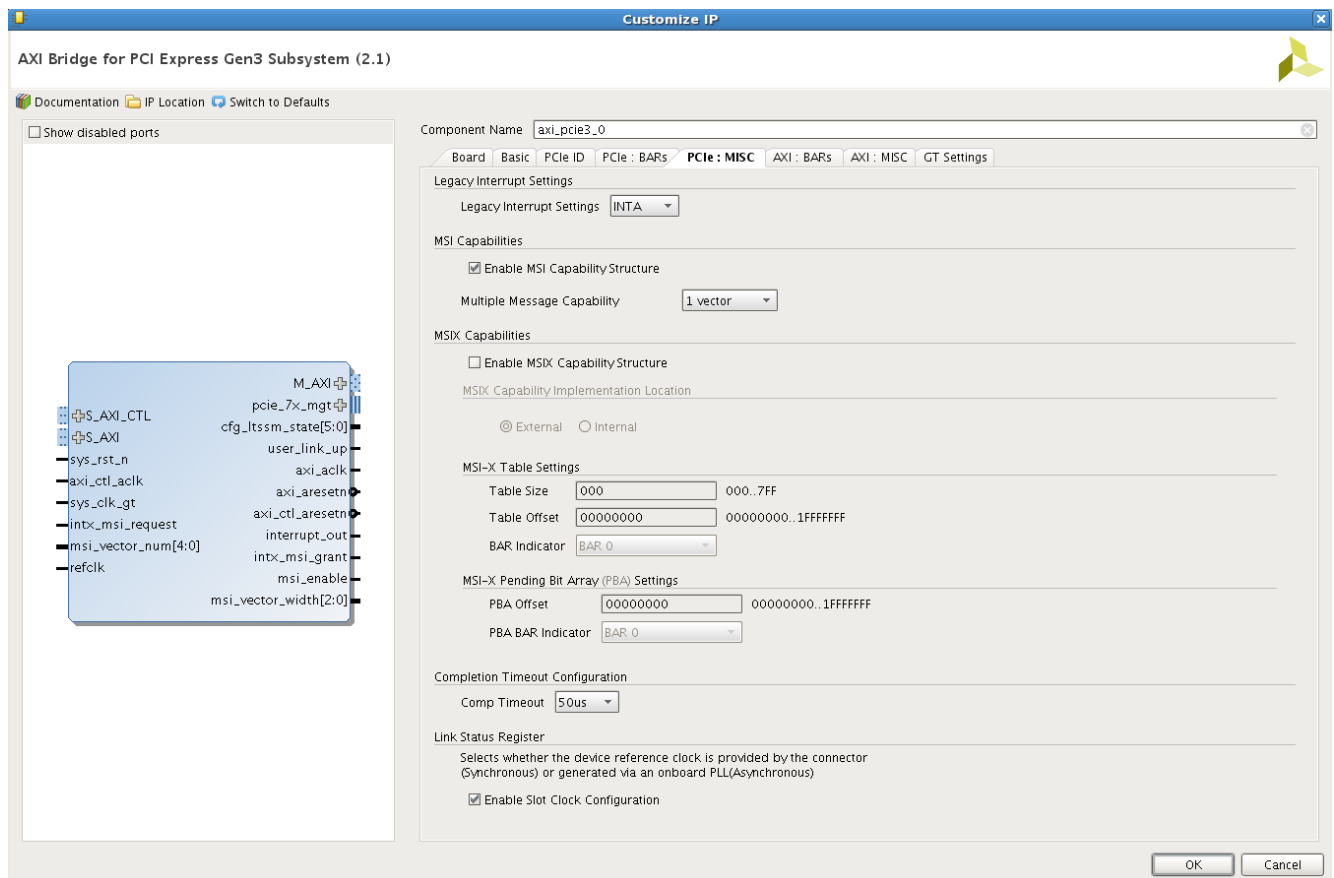
### Disabling Unused Resources

For best results, disable unused base address registers to conserve system resources. A base address register is disabled by deselecting unused BARs in the Vivado IDE.

## PCIe Miscellaneous Tab

The PCIe Miscellaneous parameters are shown in the following figure.

Figure 38: PCIe Miscellaneous Setting



### Legacy Interrupt Settings

Indicates the usage of Legacy interrupts. The Bridge core implements INTA only.

### Enable MSI Capability Structure

Indicates that the MSI capability structure exists. Cannot be enabled when MSI-X Capability Structure is enabled.

### Enable MSIX Capability Structure

Indicates that the MSI-X capability structure exists. Cannot be enabled when MSI Capability Structure is enabled.

### MSIX Table Settings

Defines the MSI-X Table structure.

- **Table Size:** Specifies the MSI-X Table size.
- **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.

### Multiple Message Capable

Indicates the number of message signals interrupt vectors that this endpoint could request.

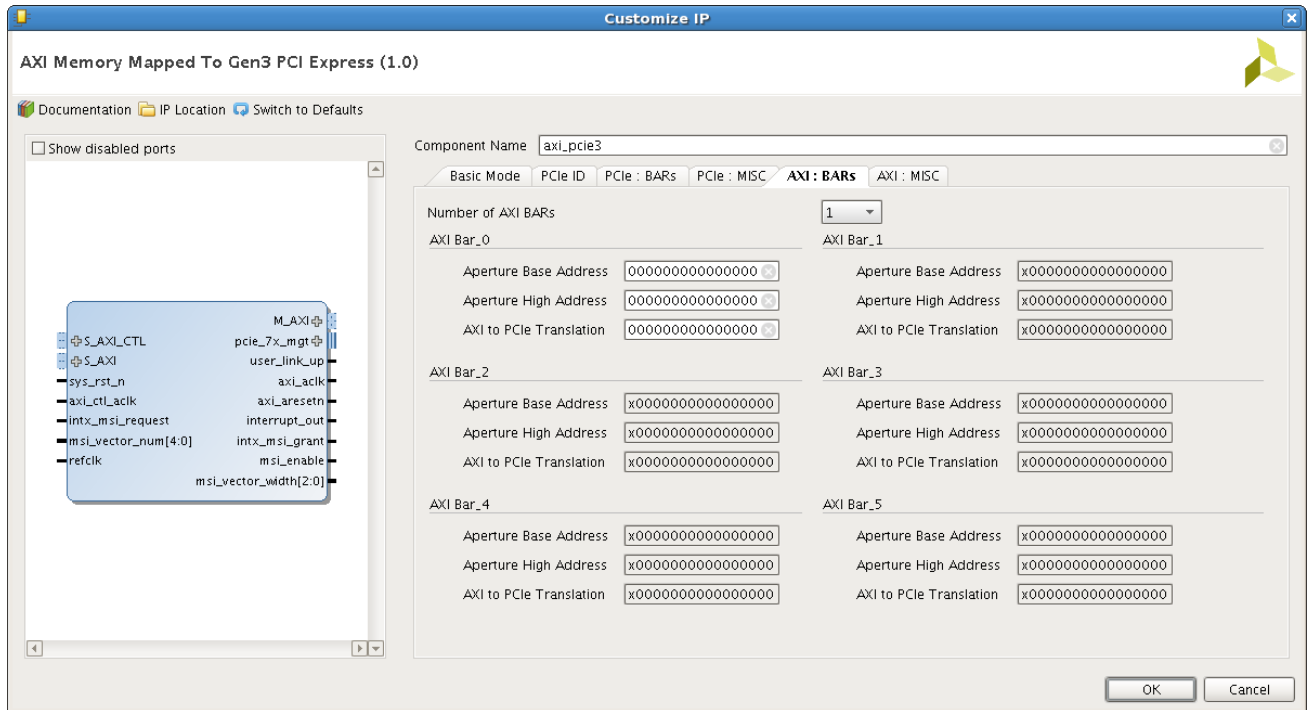
### Completion Timeout Configuration

Indicates the completion timeout value for incoming completions due to outstanding memory read requests. This option is deprecated and does not have any effect. This option is now maintained from the Device Control 2 register in the PCIe Configuration Space register.

### ***AXI BARs Tab***

The AXI Base Address Registers (BARs) parameters shown in the following figure set the AXI base address registers and the translation between AXI Memory space and PCI Express® Memory space. Each BAR has a Base Address, High Address, and translation field which can be configured through the Vivado IDE.

Figure 39: AXI Base Address Registers



- **Number of BARS:** Indicates the number of AXI BARS enabled. The BARS are enabled sequentially.
- **Aperture Base Address:** Sets the base address for the address range of BAR. You should edit this parameter to fit design requirements.

In the Vivado® IDE, this parameter is handled in the Address Manager, and does not appear in the Core Customization dialog box.

- **Aperture High Address:** Sets the upper address threshold for the address range of BAR. You should edit this parameter to fit design requirements.

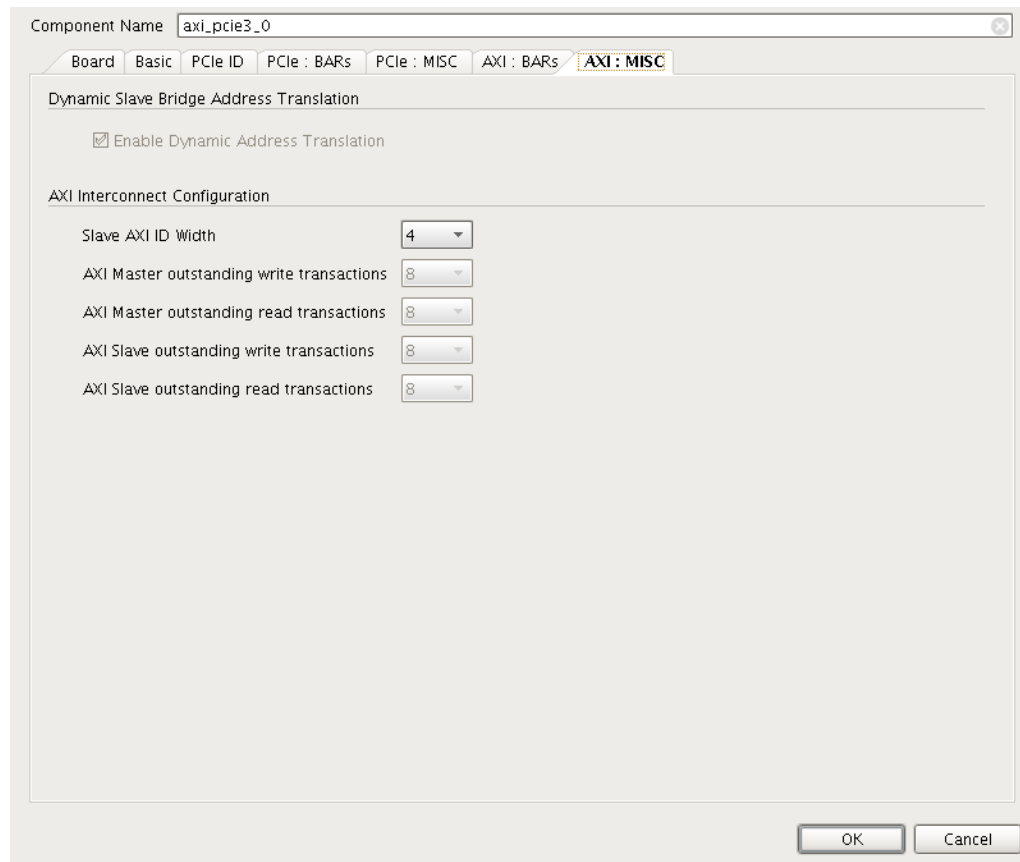
In the Vivado IDE, this parameter is handled in the Address Manager, and does not appear in the Core Customization dialog box.

- **AXI to PCIe Translation:** Configures the translation mapping between AXI and PCI Express address space. You should edit this parameter to fit design requirements.

## AXI Miscellaneous Tab

The AXI Miscellaneous parameters shown in the following figure set the additional AXI parameters.

Figure 40: AXI Miscellaneous Settings



- **S AXI ID WIDTH:** Sets the ID width for the AXI Slave Interface.

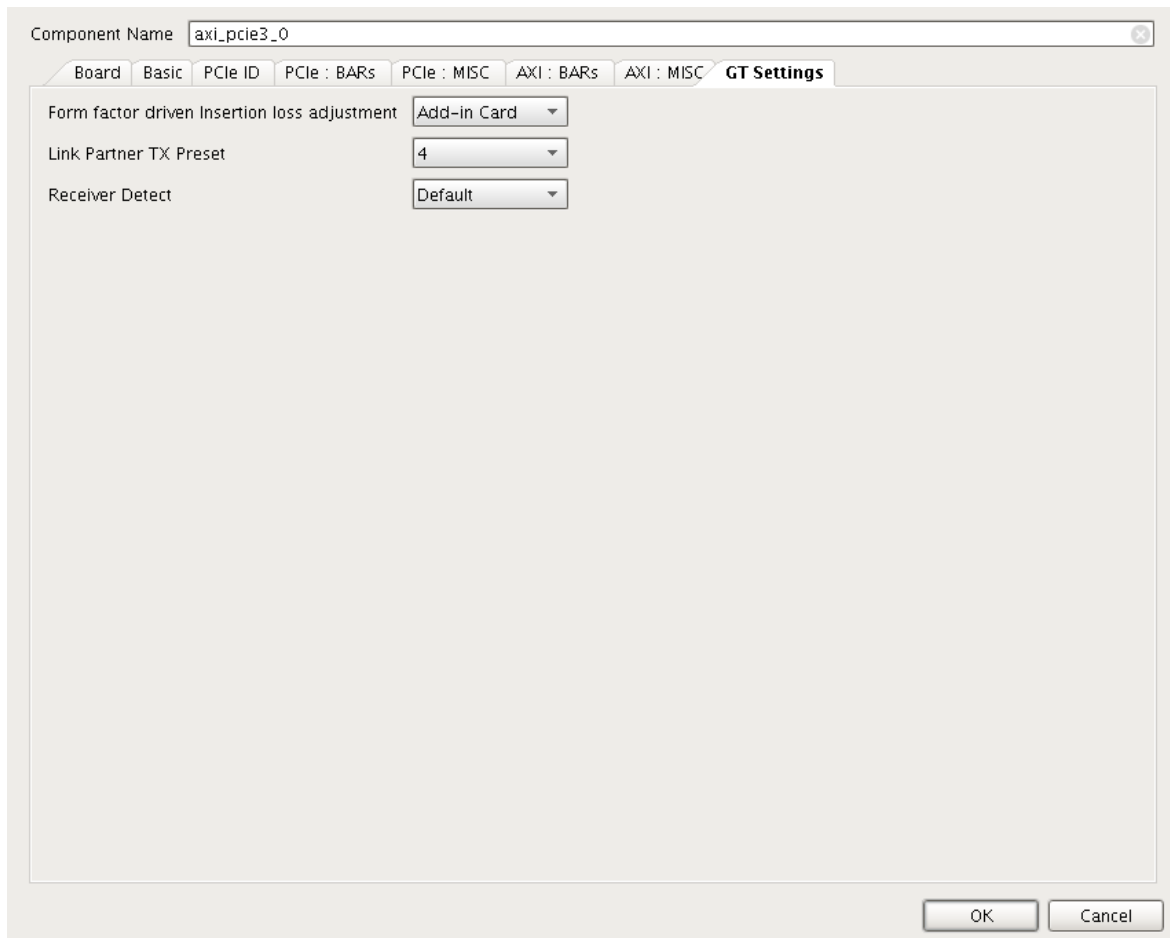
Multiple IDs are not supported for AXI Master Interface. Therefore, all signals concerned with ID are not available at AXI Master Interface.

- **AXI Master outstanding write transactions:** Indicates the number of outstanding write transactions that are allowable for the AXI Master interface.
- **AXI Master outstanding read transactions:** Indicates the number of outstanding read transactions that are allowable for the AXI Master interface.
- **AXI Slave outstanding write transactions:** Indicates the number of outstanding write transactions that are allowable for the AXI Slave interface.
- **AXI Slave outstanding read transactions:** Indicates the number of outstanding read transactions that are allowable for the AXI Slave interface.

## GT Settings Tab

The following figure shows the GT Settings tab.

Figure 41: GT Settings



- **Form factor driven Insertion loss adjustment:** Indicates the insertion loss profile options. There are three options provided.

1. Chip-to-Chip (5 dB)
2. Add-in Card (15 dB)
3. Backplane (20 dB)

- **Link Partner TX Preset:** Available only when an UltraScale™ device is selected.

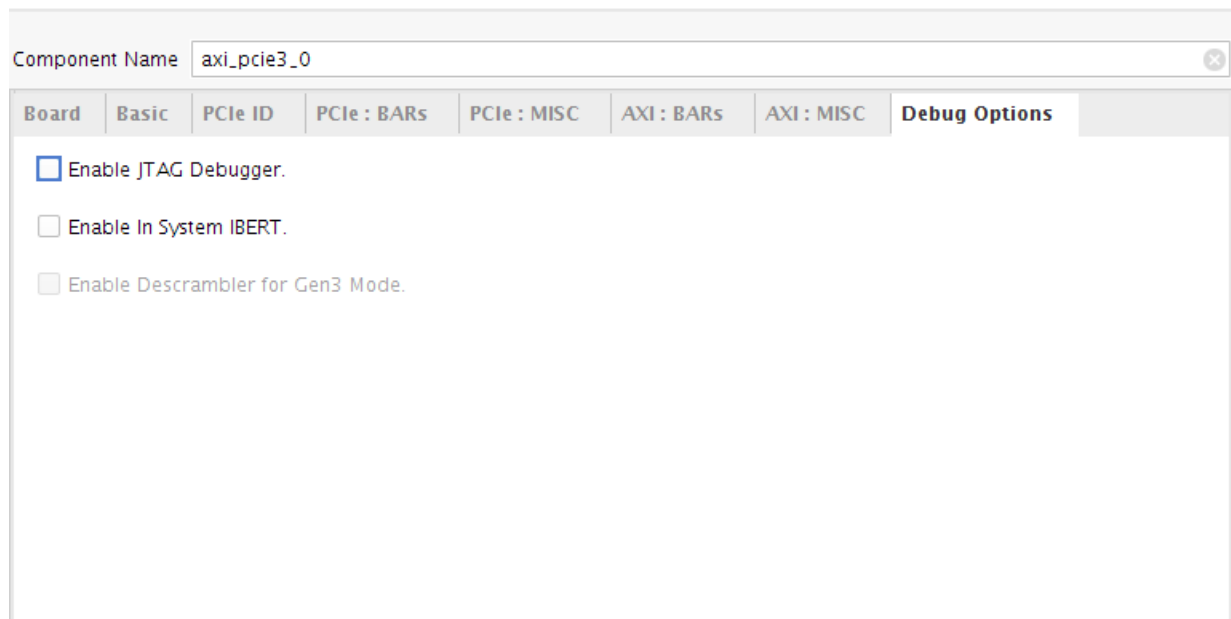
The default value is 4. It is not advisable to change the default value unless recommended by Xilinx® Technical Support.

- **Receiver Detect:** Indicates the type of Receiver Detect: Default or Falling Edge. This parameter is available in the GT Settings Tab when Advanced mode is selected. This parameter is available only for Production devices. When the Falling Edge option is selected, the GT Channel DRP Parameter in the Basic tab (in Advanced mode) is disabled. For more information on the receiver falling edge detect, see the applicable GT User Guide for your targeted device (*7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)*), or *UltraScale Architecture GTH Transceivers User Guide (UG576)*.

## Debug Options Tab

The following figure shows the debug options, which enable you to use of the debug utilities of the UltraScale™ and 7 series PCI Express® subcores.

Figure 42: Debug Options



- **Enable JTAG Debugger:** When selected, the JTAG debugger option of the base IP is set to true which will enable the debugging capability of the subcore through the JTAG interface. For more details, see *Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide (PG023)*, and *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)*.
- **Enable In System IBERT:** This debug option is used to view the eye diagram of the serial link at the desired link speed. This option is available for UltraScale devices only, and not supported for 7 series devices. For more details, see *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)*.
- **Enable Descrambler for Gen3 Mode:** This debug option integrates an encrypted version of the descrambler module inside the PCIe core, which will be used to descrambler the PIPE data to/from the PCIe integrated block in Gen3 link speed mode. This provides hardware-only support to debug on the board.



# DMA/AXI Bridge Subsystem for PCI Express in AXI Bridge Mode

The following tabs and options are available when the Functional Mode option (Basic Tab) is set to AXI Bridge.

## Basic Tab

The Basic tab options for the AXI Bridge mode (Functional Mode option) are shown in the following figure.

Figure 43: Basic Tab for AXI Bridge Functional Mode

The screenshot displays the configuration interface for the AXI Bridge Functional Mode. The 'Basic' tab is selected, and the 'Functional Mode' is set to 'AXI Bridge'. The 'Mode' is set to 'Advanced'. The 'Device / Port Type' is 'PCI Express Endpoint device' and the 'PCIe Block Location' is 'X1Y0'. Under 'GT Selection', 'Enable GT Quad Selection' is unchecked, and 'GT Quad' is set to 'GTY Quad 227'. The 'PCIe Interface' section shows 'Lane Width' as 'X1', 'Maximum Link Speed' as '2.5 GT/s', 'Reference Clock Frequency (MHz)' as '100 MHz', and 'Reset Source' as 'User Reset'. The 'AXI Interface' section shows 'AXI Address Width' as '32', 'AXI Data Width' as '64 bit', and 'AXI Clock Frequency' as '250'. Both 'Enable AXI Slave Interface' and 'Enable AXI Master Interface' are checked. At the bottom, 'System Reset polarity' is 'ACTIVE LOW' and 'Tandem Configuration or Partial Reconfiguration' is 'None'. There are also four unchecked checkboxes: 'Enable PIPE Simulation', 'Enable GT Channel DRP Ports', 'Enable PCIe DRP Ports', and 'Additional Transceiver Control and Status Ports'.

The options are defined as follows:

- **Device / Port Type:** PCI Express Endpoint device or Root Port of PCI Express® Root complex can be selected.
- **Enable AXI Slave Interface:** This interface is selected by default, user can de-select this interface.
- **Enable AXI Master Interface:** This interface is selected by default, user can de-select this interface.

All other options are the same as those for the DMA Subsystem mode. For a description of these options, see the “Basic Tab” options in Chapter 4: “Design Flow Steps” in the *DMA/Bridge Subsystem for PCI Express Product Guide* (PG195).

## PCIe ID Tab

For a description of these options, see PCIe ID Tab.

## PCIe BARs Tab

The PCIe BARs tab options for the AXI Bridge mode (Functional Mode option) is shown in the following figure.

Figure 44: PCIe BARs Tab for AXI Bridge Functional Mode

The screenshot displays the 'PCIe : BARs' configuration window. It features five sections for configuring PF0\_BAR0 through PF0\_BAR5. Each section contains the following fields:

- PF0\_BAR0:** Type: Memory; 64 Bit: ; Prefetchable: ; Size: 128; Scale: Kilobytes; Value: FFFE0000; PCIe to AXI Translation: 0x0000000000000000
- PF0\_BAR1:** Type: Memory; 64 Bit: ; Prefetchable: ; Size: 4; Scale: Kilobytes; Value: FFFF0000; PCIe to AXI Translation: 0x0000000000000000
- PF0\_BAR2:** Type: Memory; 64 Bit: ; Prefetchable: ; Size: 4; Scale: Kilobytes; Value: FFFF0000; PCIe to AXI Translation: 0x0000000000000000
- PF0\_BAR3:** Type: Memory; 64 Bit: ; Prefetchable: ; Size: 4; Scale: Kilobytes; Value: FFFF0000; PCIe to AXI Translation: 0x0000000000000000
- PF0\_BAR4:** Type: Memory; 64 Bit: ; Prefetchable: ; Size: 4; Scale: Kilobytes; Value: FFFF0000; PCIe to AXI Translation: 0x0000000000000000
- PF0\_BAR5:** Type: Memory; 64 Bit: ; Prefetchable: ; Size: 4; Scale: Kilobytes; Value: FFFF0000; PCIe to AXI Translation: 0x0000000000000000

## Base Address Register Overview

The Bridge core in Endpoint configuration supports up to six 32-bit BARs or three 64-bit BARs. The AXI Bridge for PCI Express® in Root Port configuration supports up to two 32-bit BARs or one 64-bit BAR.

BARs can be one of two sizes. BARs 0, 2, and 4 can be either 32-bit or 64-bit addressable. BARs 1, 3, and 5 can only be 32-bit addressable and are disabled if the previous BAR is enabled as 64-bit.

- **32-bit BARs:** The address space can be as small as 4 kilobytes or as large as 2 gigabytes. Used for Memory to I/O.
- **64-bit BARs:** The address space can be as small as 4 kilobytes or as large as 256 gigabytes. Used for Memory only.

All BAR registers share these options:

- **Checkbox:** Click the checkbox to enable BAR. Deselect the checkbox to disable BAR.
- **Type:** BARs can be Memory apertures only. Memory BARs can be either 64-bit or 32-bit. Prefetch is enabled for 64-bit and not enabled for 32-bit. When a BAR is set as 64 bits, it uses the next BAR for the extended address space, making it 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 66: BAR Size Ranges for Device Configuration

PCIe Device/Port Type	BAR Type	BAR Size Range
PCI Express Endpoint	32-bit Memory	4 Kilobytes - 2 Gigabytes
	64-bit Memory	4 Kilobytes - 8 Exabytes

- **Prefetchable:** Identifies the ability of the memory space to be prefetched. This can only be enabled for 64-bit addressable bars.
- **Value:** The value assigned to BAR.
- **PCIe to AXI Translation:** This text field should be set to the appropriate value to perform the translation from the PCI Express base address to the desired AXI Base Address.

## Managing Base Address Register Settings

Memory indicates that the address space is defined as memory aperture. The base address register only responds to commands that access the specified address space. If MSI-X Capability Structure is enabled and MSI-X Internal implementation location is selected, there will be a 64 KB reserved address space in one of the enabled PCIe BAR. See the BAR Indicator option in the PCIe Misc Tab.

## 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 Vivado® IDE.

## PCIe Miscellaneous Tab

The PCIe Miscellaneous tab options for the AXI Bridge mode (Functional Mode option) are shown in the following figure.

Figure 45: PCIe Miscellaneous Tab for AXI Bridge Functional Mode

The screenshot shows the 'PCIe : MISC' tab selected in the Vivado IDE. The configuration options are as follows:

- Bar Indicator Settings:** Bar Indicator is set to BAR 0.
- User Interrupts:** Number of User Interrupts Request (1-16) is set to 1.
- Legacy Interrupt Settings:** Legacy Interrupt Settings is set to INTA.
- MSI Capabilities:**
  - Enable MSI Capability Structure:
  - Multiple Message Capability: 1 vector
- MSI-X Capabilities:**
  - Enable MSI-X Capability Structure:
- MSI-X Implementation Location:**
  - Internal:  External:
- Completion Timeout Configuration:** Completion Timeout is set to 50ms.
- Miscellaneous:**
  - Config Extended Interface:
- Link Status Register:**
  - Enable Slot Clock Configuration:

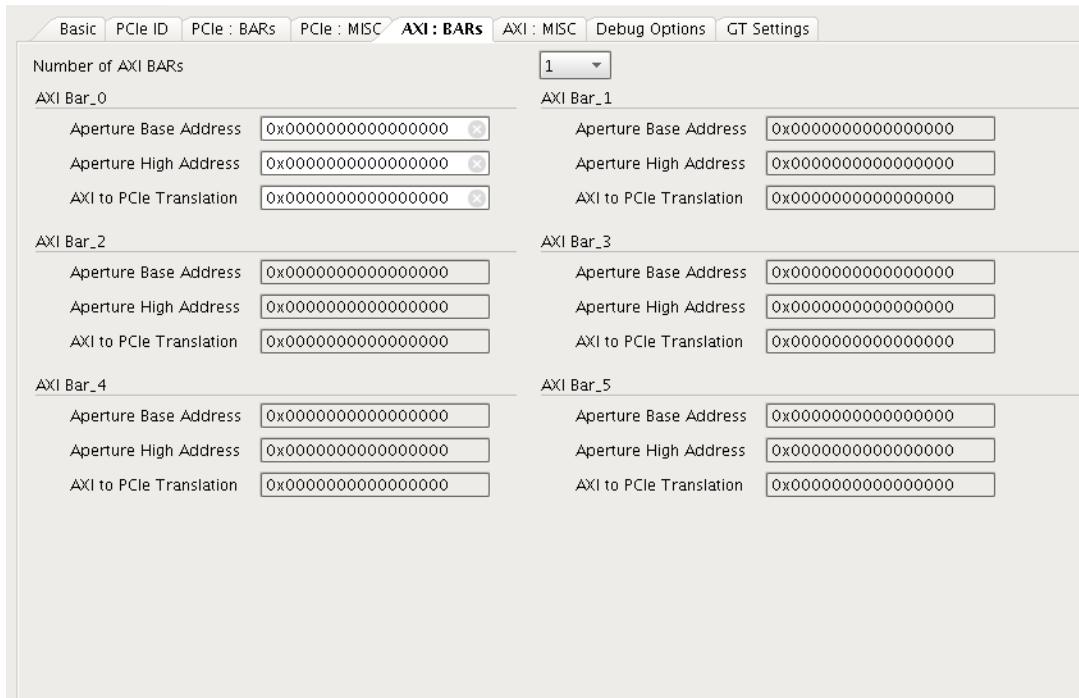
The options are defined as follows:

- **Bar Indicator:** This is the space allocated for Interrupt processing registers that are shared between AXI Bridge and AXI DMA. This register space includes an MSI-X table, and when MSI-X Internal is selected, 64KB address space from the BAR indicated here will be reserved for the MSI-X table. Depending on PCIe BAR selection this register space can be allocated to any selected BAR space (BAR0 to BAR5). This options is valid only when MSI-X Internal option is selected. For all other interrupt options there is no allocated space.
- **User Interrupts:** Select number of user interrupts.
- **Legacy Interrupt Settings:** Select one of the Legacy Interrupts: INTA, INTB, INTC, or INTD.
- **Number of User Interrupt Request:** Up to 16 user interrupt requests can be selected.
- **MSI Capabilities:** By default, MSI Capabilities is enabled, and 1 vector is enabled. You can choose up to 16 vectors. In general, Linux uses only 1 vector for MSI. This option can be disabled.
- **MSI-X Capabilities:** Select a MSI-X event. For more information, see [MSI-X Vector Table and PBA \(0x8\)](#).
- **MSI-X Implementation Location:** For MSI-X, there are two options: Internal and External. When Internal is selected, MSI-X table is internal to the IP and the table can be accessed depending on BAR Indicator selection. When External is selected, MSI-X related ports are brought out of IP and user is responsible to make table outside of IP. In this case, Bar Indicator is not used.
- **Completion Timeout Configuration:** By default, completion timeout is set to 50 ms. Option of 50 us is also available. This option is deprecated and does not have any effect. This option is now maintained from the Device Control 2 register in the PCIe Configuration Space register.
- **Config Extended Interface:** PCIe extended interface can be selected for more configuration space. When Configuration Extend Interface is selected, you are responsible for adding logic to extend the interface to make it work properly.

## ***AXI BARs Tab***

The AXI BARs tab options for the AXI Bridge mode (Functional Mode option) are shown in the following figure.

Figure 46: AXI BARs Tab for AXI Bridge Functional Mode



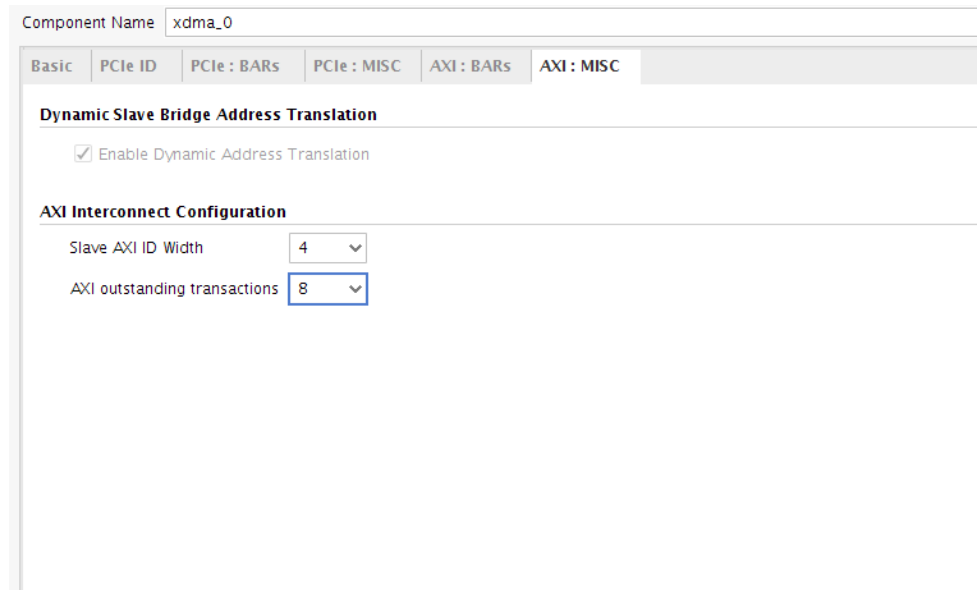
The options are defined as follows:

- **Number of BARs:** Indicates the number of AXI BARs enabled. The BARs are enabled sequentially.
- **Aperture Base Address:** Sets the base address for the address range of BAR. You should edit this parameter to fit design requirements.
- **Aperture High Address:** Sets the upper address threshold for the address range of BAR. You should edit this parameter to fit design requirements.
- **AXI to PCIe Translation:** Configures the translation mapping between AXI and PCI Express® address space. You should edit this parameter to fit design requirements.

### AXI Misc Tab

The AXI Miscellaneous tab options for the AXI Bridge mode (Functional Mode option) are shown in the following.

Figure 47: AXI Misc Tab for AXI Bridge Functional Mode



The options are defined as follows:

- **Dynamic Slave Bridge Address Translation:** Is always enabled.
- **Slave AXI ID Width:** Sets the ID width for the AXI Slave Interface.
- **AXI outstanding transactions:** Select from the options: 8, 16 or 32 outstanding transactions. This options is common for AXI Master Write, AXI Master Read, AXI Slave Write and AXI Slave Read transaction. The number of AXI Master Write transactions is based on the AXI data width size (4 for 64-bit, 8 for 128-bit, 16 for 256 bits or 32 for 512-bits).

### **Debug Options Tab**

For a description of these options, see [Debug Options Tab](#).

### **GT Settings Tab**

For a description of these options, see [GT Settings Tab](#).

## **Output Generation**

For details, see the *Vivado Design Suite User Guide: Designing with IP (UG896)*.

For information regarding the example design, see Example Design Output Structure.

### **Related Information**

[Example Design Output Structure](#)

# Constraining the Subsystem

This section contains information about constraining the core in the Vivado® Design Suite.

## Required Constraints

The Bridge core requires a clock period constraint for the reference clock input that agrees with the REF\_CLK\_FREQ parameter setting. In addition, pin-placement (LOC) constraints are needed that are board/part/package specific.

See Placement Constraints for more details on the constraint paths for FPGA architectures.

Additional information on clocking can be found in the Xilinx® Solution Center for PCI Express (see [Solution Centers](#)).

**Note:** The reference clock input is `refclk` in Virtex®-7 devices and `sys_clk_gt` in UltraScale devices.

## System Integration

A typical embedded system including the Bridge core is shown in . Some additional components to this system in the Vivado® IP integrator can include the need to connect the MicroBlaze™ processor or Zynq® device Arm® processor peripheral to communicate with PCI Express® (in addition to the AXI4-Lite register port on the PCIe bridge). The AXI Interconnect provides this capability and performs the necessary conversions for the various AXI ports that might be connected to the AXI Interconnect IP (see AXI to AXI Connector Data Sheet ([DS803](#))).

The Bridge core can be configured with each port connection for an AXI Vivado IP integrator system topology. When instantiating the core, ensure the following bus interface tags are defined.

```
BUS_INTERFACE M_AXI
BUS_INTERFACE S_AXI
BUS_INTERFACE S_AXI_CTL
```

## PCIe Clock Integration

The PCIe® differential clock input in the system might need to use a differential input buffer (that is instantiated separately) from the Bridge core. The Vivado® IP integrator automatically inserts the appropriate clock buffer.



## Placement Constraints

The Bridge core provides a Xilinx® design constraint (XDC) file for all supported PCIe, Part, and Package permutations. You can find the generated XDC file in the Sources tab of the Vivado® IDE after generating the IP in the Customize IP dialog box.

For design platforms, it might be necessary to manually place and constrain the underlying blocks of the Bridge core. The modules to assign a LOC constraint include:

- Embedded integrated block for PCIe
- GTH transceivers (for each channel)
- PCIe differential clock input

The following subsection describes the example location constraints.

## Location Constraints

This section highlights the LOC constraints to be specified in the XDC file for the Bridge core for design implementations.

For placement/path information on the integrated block for PCIe® itself, use the following constraint:

```
# 7 Series Constraint
set_property LOC PCIE_X*Y* [get_cells {axi_pcie3_0_i/inst/pcie3_ip_i/inst/
pcie_top_i/pcie_7vx_i/PCIE_3_0_i}]
# Ultrascale Constraint
set_property LOC PCIE_X*Y* [get_cells {axi_pcie3_0_i/inst/pcie3_ip_i/inst/
pcie3_uscale_top_inst/pcie3_uscale_wrapper_inst/PCIE_3_1_inst}]
```

For placement/path information of the GTH transceivers, use the following constraint:

```
# 7 Series Constraint
set_property LOC GTXE2_CHANNEL_X*Y* [get_cells {axi_pcie3_0_i/inst/
pcie3_ip_i/inst/
gt_top_i/pipe_wrapper_i/pipe_lane[0].gt_wrapper_i/
gth_channel.gthe2_channel_i}]
# Ultrascale Constraint
set_property LOC GTXE2_CHANNEL_X*Y* [get_cells {axi_pcie3_0_i/inst/
pcie3_ip_i/inst/
gt_top_i/gt_wizard.gtwizard_top_i/axi_pcie3_0_pcie3_ip_gt_i/inst/
gen_gtwizard_gthe3_top.axi_pcie3_0_pcie3_ip_gt_gtwizard_gthe3_inst/
gen_gtwizard_gthe3.gen_channel_container[1].gen_enabled_channel.gthe3_chan
nel_wrapp
er_inst/channel_inst/
gthe3_channel_gen.gen_gthe3_channel_inst[0].GTHE3_CHANNEL_PRIM_INST}]
```

For placement/path constraints of the input PCIe differential clock source (using the example provided in System Integration), use the following constraint:

```
set_property LOC IBUFDS_GTE2_X*Y* [get_cells {refclk_ibuf}]
```

## Clock Frequencies

The AXI Memory Mapped to PCI Express® Bridge supports reference clock frequencies of 100 MHz, 125 MHz, and 250 MHz and is configurable within the Vivado® IDE.

## Clock Management

Clock management is covered in the core clocking section.

### Related Information

[Clocking](#)

## Clock Placement

For details, see [Placement Constraints](#).

## Banking

This section is not applicable for this IP core.

## Transceiver Placement

The Transceiver primitives adjacent to the PCIe hard block should be used to aid in the place and route of the solution IP. The adjacent Transceiver banks one above or one below the desired PCIe hard block can also be used. Transceivers outside this range are not likely to meet the timing requirements for the PCI Express® Solution IP and should not be used.

## I/O Standard and Placement

The `sys_reset_n` input should be driven directly by FPGA I/O pins. The pin should be driven by the PCI Express® edge connector reset signal (`perstn`). As described in the PCI Express® Specification, the PCI Express edge connector reset is driven at 3.3V. If the bank voltage for the `sys_reset_n` pin differs from 3.3V, and external level shifter is required to convert the 3.3V PCI Express edge connector reset (`perstn`) to the desired FPGA bank voltage.

---

## Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third party tools, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)*.

**Note:** For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx® IP is tested and qualified with UNISIM libraries only.

## Post-Synthesis/Post-Implementation Netlist Simulation

The DMA/Bridge Subsystem core for UltraScale+™ devices does not support post-synthesis/post-implementation netlist functional simulations. The AXI Bridge for PCIe® core supports post-synthesis/post-implementation netlist functional simulations. However, some configurations do not support this feature in this release. See the following table for the configuration support of netlist functional simulations.

Post-synthesis/implementation netlist *timing* simulations are not supported for any of the configurations in this release.

**Table 67: Configuration Support for Functional Simulation**

Configuration	Verilog	VHDL	PIPE Mode Option/ External PIPE Interface	Shared Logic in Core	Shared Logic in Example Design
Endpoint	Yes	N/A	No		N/A

### Post-Synthesis Netlist Functional Simulation

To run a post-synthesis netlist functional simulation:

1. Generate the core with required configuration
2. Open the example design and run Synthesis
3. After synthesis is completed, in the Flow Navigator, right-click the **Run Simulation** option and select **Run Post-Synthesis Functional Simulation**.

### Post-Implementation Netlist Functional Simulation

To run post-implementation netlist functional simulations:

1. Complete the above steps post-synthesis netlist function simulation.
2. Run the implementation for the generated example design.
3. After implementation is completed, in the Flow Navigator, right-click the **Run Simulation** option and select **Run Post-Implementation Functional Simulation**.

---

## Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* ([UG896](#)).

# Example Design

This chapter contains information about the example design provided with the generation of the IP in the Vivado® Design Suite.

---

## Endpoint Configuration of the AXI PCIe Block

The example simulation design for the Endpoint configuration of the AXI PCIe® block consists of two parts.

- Root Port Model: a test bench that generates, consumes, and checks PCI Express® bus traffic
- AXI Block RAM Controller

### Xilinx AXI Verification IP Attached to the AXI Slave Interface

For DMA/Bridge Subsystem for PCIe in AXI Bridge mode, an alternative example design with Xilinx® AXI Verification IP attached to the AXI Slave interface is available. AXI Verification IP allows you to initiate AXI transfer from an Endpoint configured bridge or generates a larger more complex AXI transfer from a Root Port configured bridge. For more details, see the *AXI Verification IP LogiCORE IP Product Guide (PG267)*.

To enable AXI Verification IP example design option:

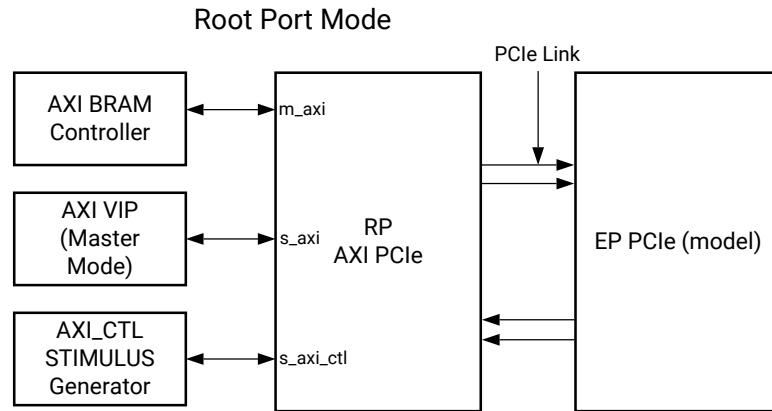
1. Add the DMA/Bridge Subsystem for PCIe IP to the design.
2. Run the following Tcl command in Vivado® console:

```
set_property CONFIG.axi_vip_in_exdes true [get_ips <ip_name>]
```

3. Open the IP Example Design.

The following figure shows the DMA/Bridge Subsystem for PCIe IP in Root Port configuration.

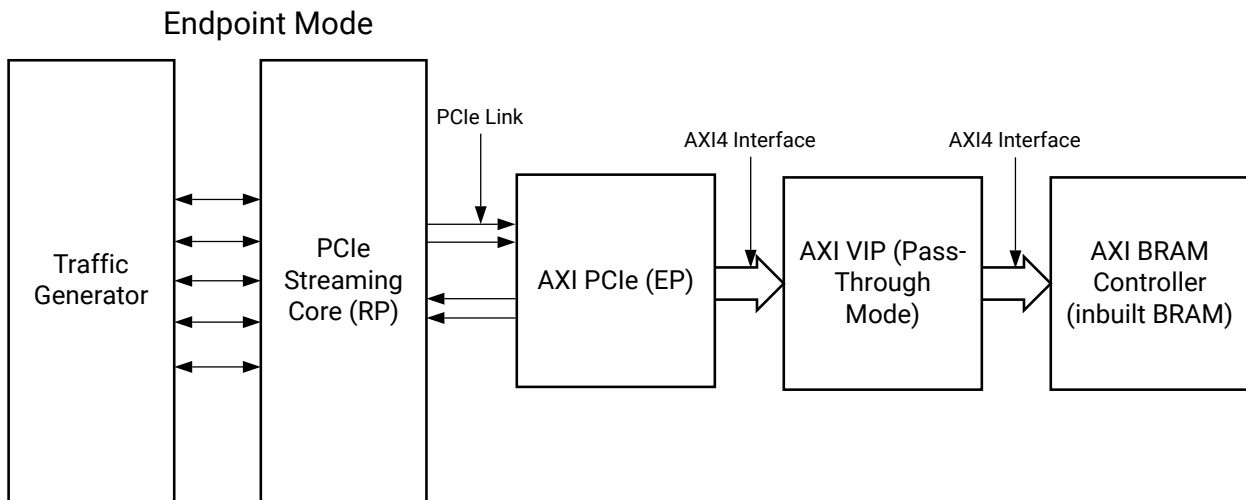
Figure 48: Root Port Configuration



X20166-042919

The following figure shows the DMA/Bridge Subsystem for PCIe IP in Endpoint configuration.

Figure 49: Endpoint Configuration



X20167-042919

## Customizing and Generating the Example Design

In the Customize IP dialog box, use the default core parameter values for the IP example design. In particular:

- In the PCIE:Basics tab, the example design supports only an Endpoint (EP) device.

- In the AXI:BARS tab, the Base Address, High Address, and AXI to PCIe® Translation default values are used.

After reviewing the core parameters:

1. Right-click the component name.
2. Select **Open IP Example Design**.

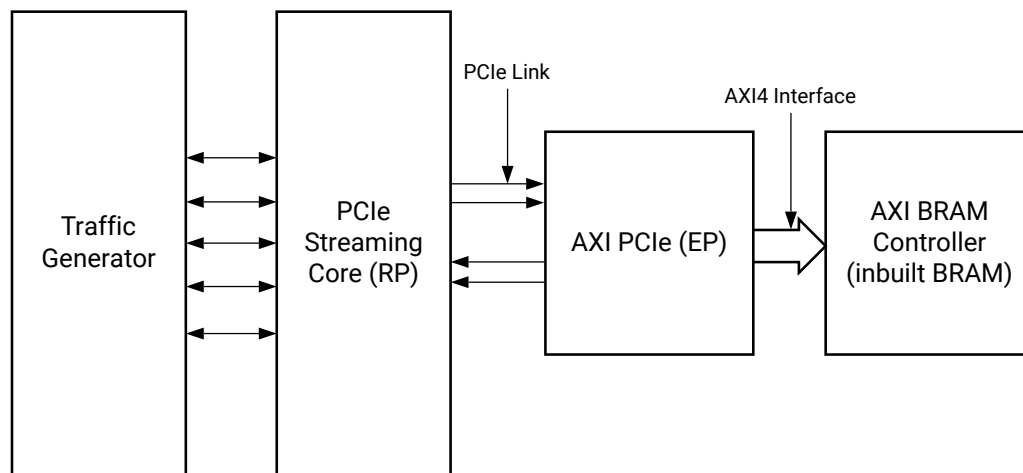
This opens a separate example design.

## Simulation Design Overview

For the simulation design, transactions are sent from the Root Port Model to the Bridge core configured as an Endpoint and processed inside the AXI Block RAM controller design.

The following figure illustrates the simulation design provided with the Bridge core.

Figure 50: Example Design Block Diagram



X20048-042919

The example design supports Verilog as the target language.

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)*.

## Implementation Design Overview

For implementation design, the AXI Block RAM controller can be used as a scratch pad memory to write and read to Block RAM locations.

## Example Design Elements

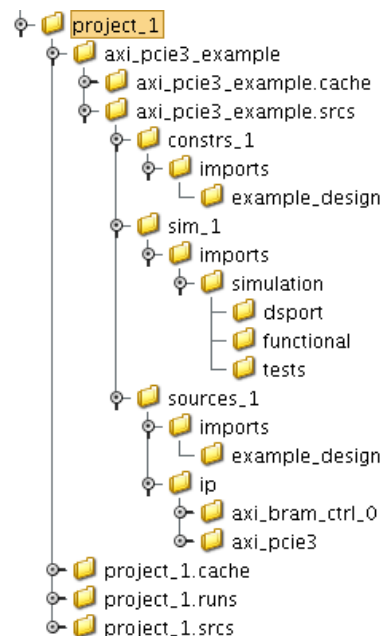
The core wrapper includes:

- An example Verilog HDL or VHDL wrapper (instantiates the cores and example design).
- A customizable demonstration test bench to simulate the example design.

## Example Design Output Structure

The following figure shows the output structure of the example design.

Figure 51: Example Design Output Structure



The following table provides a description of the contents of the example design directories.



Directory	Description
project_1/axi_pcie3_example	Contains all example design files.
project_1/axi_pcie3_example/axi_pcie3_example.srcs/sources_1/imports/example_design/	Contains the top module for the example design, <code>xilinx_axi_pcie3_ep.v</code> .
project_1/axi_pcie3_example/axi_pcie3_example.srcs/sources_1/ip/axi_pcie3	Contains the XDC file based on device selected, all design files and subcores used in axi_pcie, and the top modules for simulation and synthesis.
project_1/axi_pcie3_example/axi_pcie3_example.srcs/sources_1/ip/axi_bram_ctrl_0	Contains block RAM controller files used in example design.
project_1/axi_pcie3_example/axi_pcie3_example.srcs/sim_1/imports/simulation/dsport	Contains all RP files, cgator and PIO files.
project_1/axi_pcie3_example/axi_pcie3_example.srcs/sim_1/imports/simulation/functional	Contains the test bench file.
project_1/axi_pcie3_example/axi_pcie3_example.srcs/constrs_1/imports/example_design	Contains the example design XDC file.

# Test Bench

This chapter contains information about the test benches provided in the Vivado® Design Suite environment.

---

## Root Port Model Test Bench for Endpoint

The PCI Express® Root Port Model is a robust test bench environment that provides a test program interface that can be used with the provided PIO design or with a user design. The purpose of the Root Port Model is to provide a source mechanism for generating downstream PCI Express TLP traffic to stimulate your design, and a destination mechanism for receiving upstream PCI Express TLP traffic from your design in a simulation environment.

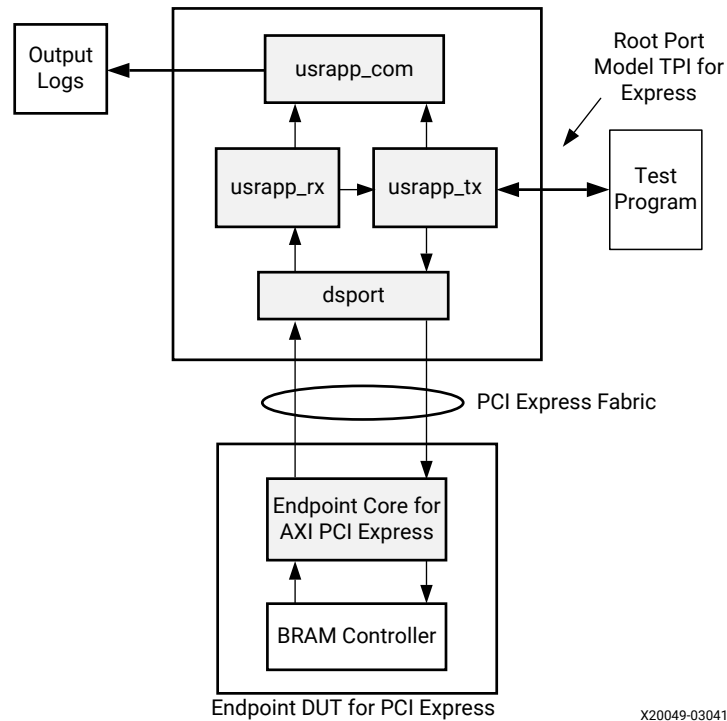
Source code for the Root Port Model is included to provide the model for a starting point for your test bench. All the significant work for initializing the configuration space, creating TLP transactions, generating TLP logs, and providing an interface for creating and verifying tests are complete, allowing you to dedicate efforts to verifying the correct functionality of the design rather than spending time developing an Endpoint core test bench infrastructure.

The Root Port Model consists of:

- Test Programming Interface (TPI), which allows you to stimulate the Endpoint device for the model.
- Example tests that illustrate how to use the test program TPI.

The following figure illustrates the Root Port Model coupled with the PIO design.

Figure 52: Root Port Model for AXI\_PCIE Endpoint



X20049-030419

## Architecture

The Root Port Model consists of these blocks, illustrated in the previous figure:

- dsport (Root Port)
- usrapp\_tx
- usrapp\_rx
- usrapp\_com (Verilog only)

The `usrapp_tx` and `usrapp_rx` blocks interface with the `dsport` block for transmission and reception of TLPs to/from the Endpoint Design Under Test (DUT). The Endpoint DUT consists of the Endpoint for AXI-PCIe® and the Block RAM controller design (displayed) or customer design.

The `usrapp_tx` block sends TLPs to the `dsport` block for transmission across the PCI Express Link to the Endpoint DUT. In turn, the Endpoint DUT device transmits TLPs across the PCI Express Link to the `dsport` block, which are subsequently passed to the `usrapp_rx` block. The `dsport` and core are responsible for the data link layer and physical link layer processing when communicating across the PCI Express logic. Both `usrapp_tx` and `usrapp_rx` utilize the `usrapp_com` block for shared functions, for example, TLP processing and log file outputting.

Transaction sequences or test programs are initiated by the `usrapp_tx` block to stimulate the logic interface of the Endpoint device. TLP responses from the Endpoint device are received by the `usrapp_rx` block. Communication between the `usrapp_tx` and `usrapp_rx` blocks allow the `usrapp_tx` block to verify correct behavior and act accordingly when the `usrapp_rx` block has received TLPs from the Endpoint device.

## Simulating the Example Design

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)*.

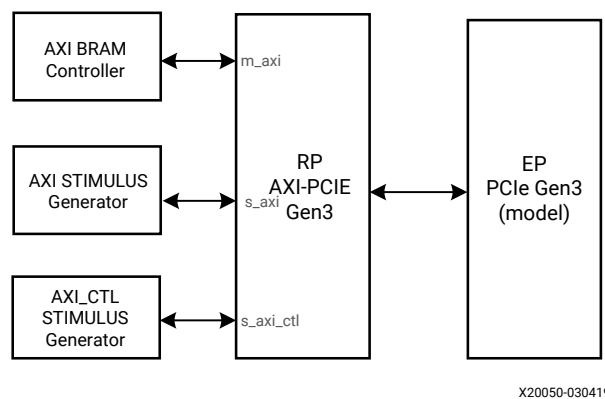
# Endpoint Model Test Bench for Root Port

The Endpoint model test bench for the Bridge core in Root Port configuration is a simple example test bench that connects the Configurator example design and the PCI Express Endpoint model allowing the two to operate like two devices in a physical system. Because the Configurator example design consists of logic that initializes itself and generates and consumes bus traffic, the example test bench only implements logic to monitor the operation of the system and terminate the simulation.

The Endpoint model test bench consists of:

- Verilog source code for all Endpoint model components
- PIO slave design

Figure 53: Endpoint Model for AXI\_PCIE Root Port



## Architecture

The Endpoint model consists of these blocks:

- PCI Express Endpoint (Bridge in Endpoint configuration) model.
- PIO slave design, consisting of:
  - `pio_rx_engine`
  - `pio_tx_engine`
  - `pio_ep_mem`
  - `pio_to_ctrl`

The `pio_rx_engine` and `pio_tx_engine` blocks interface with the `ep` block for reception and transmission of TLPs from/to the Root Port Design Under Test (DUT). The Root Port DUT consists of the core configured as a Root Port and the Block RAM controller along with `s_axi` and `s_axi_ctl` models to drive traffic on `s_axi` and `s_axi_ctl`.

## Simulating the Example Design

To simulate the design, see the [Chapter 6: Example Design](#) chapter.

### Related Information

[Example Design](#)

---

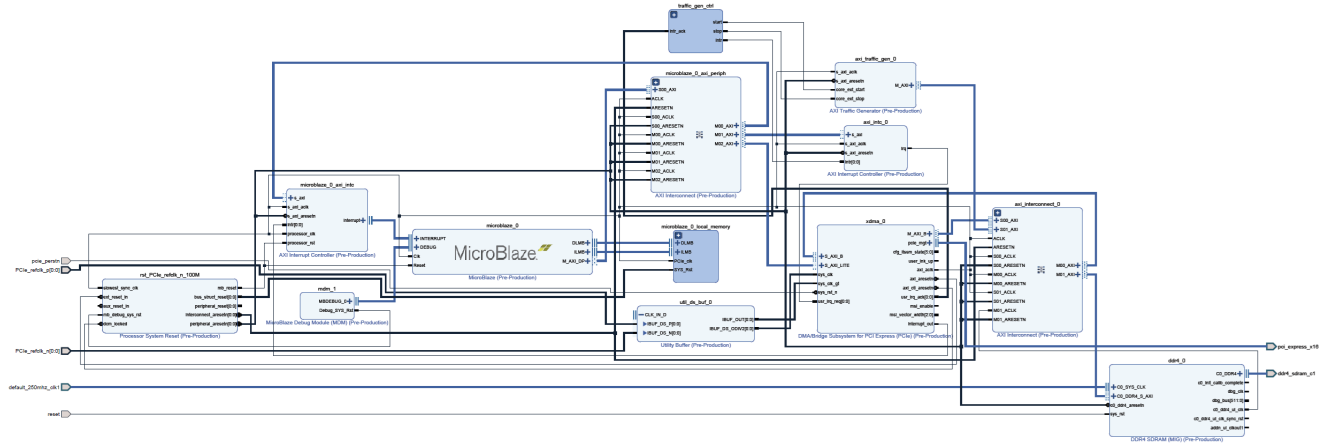
## Example Use Mode

In Root Port configuration, the slave AXI4-Lite and slave AXI4 interfaces are typically driven by a PS system or a MicroBlaze™. This microprocessor runs a kernel driver software that will do PCIe enumeration process. The Master AXI interface is connected if there is a packet generator module at the Endpoint side, such as a DMA engine. The `interrupt_out` signal is typically connected to the Processor Interrupt Controller and multiplexed together with other interrupt sources before being forwarded to the kernel driver software running at the microprocessor.

In Endpoint configuration, the Slave AXI interface is typically driven by a DMA module or a packet generator, such as an AXI exerciser. The master AXI4 interface is used to receive packets from the host, such as the DMA configuration file or PIO, and accesses directly to a Memory module, such as MIG. The slave AXI4-Lite interface is not typically used in an Endpoint configuration but can be utilized to check some Bridge/Link status registers. The User Interrupt input signals can be driven by an Interrupt controller to signal the Host for any important event at the Endpoint side, such as a DMA transfer complete and an error event.

The following figure shows an example of where the Bridge cores can be used.

**Figure 54: Example Use Mode**



# Upgrading

This appendix contains information about migrating from other cores to this core, and upgrading the core to most recent version of the IP core.

## Migrating from AXI PCIe Gen2 to AXI Bridge for PCIe Gen3

When you move from the AXI PCIe Gen2 core to the AXI Bridge for PCIe Gen3 core, design modifications are necessary.

### Parameter Changes

The following parameters have changed from the AXI PCIe Gen2 core to the AXI Bridge for PCIe Gen3 core.

Table 68: Parameter Name Changes

AXI PCIe Gen2 Parameter	AXI Bridge for PCIe Gen3 Parameter
C_PCIEBAR_LEN_0	PF0_BAR0_APERTURE_SIZE
C_PCIEBAR_LEN_1	PF0_BAR1_APERTURE_SIZE
C_PCIEBAR_LEN_2	PF0_BAR2_APERTURE_SIZE
	PF0_BAR3_APERTURE_SIZE
	PF0_BAR4_APERTURE_SIZE
	PF0_BAR5_APERTURE_SIZE
C_SUPPORTS_NARROW_BURST	C_S_AXI_SUPPORTS_NARROW_BURST <sup>1</sup>
C_NO_OF_LANES	PL_LINK_CAP_MAX_LINK_WIDTH
C_DEVICE_ID	PF0_DEVICE_ID
C_VENDOR_ID	PF0_VENDOR_ID
C_CLASS_CODE	PF0_CLASS_CODE
C_REV_ID	PF0_REVISION_ID
C_REF_CLK_FREQ	REF_CLK_FREQ

**Notes:**

1. The core supports Narrow Burst starting in 2016.4.

# Migrating from AXI Bridge for PCIe Gen3 to DMA/Bridge Subsystem for PCIe in AXI Bridge Mode

When you move from the AXI Bridge for PCIe Gen3 core to the DMA/Bridge Subsystem for PCIe in AXI Bridge mode, design modifications are necessary.

## Parameter Changes

The following parameters have changed from the AXI Bridge for PCIe Gen3 core to the DMA/Bridge Subsystem for PCIe in AXI Bridge mode.

Table 69: Parameter Changes

User Parameter Name	Display Name	New/Changed/Removed	Details	Default Value
functional_mode	Functional Mode	New	Selects DMA or AXI Bridge functional mode. Valid values: DMA, AXI Bridge	DMA
set_finite_credit	Finite Completion Credits	New	Endpoint mode only. FALSE: Infinite Completion credit is advertised to Root Complex. TRUE: Finite Completion credit is advertised to Root Complex. WARNING: PCIe specification requires Endpoint to advertise Infinite Completion credit only. Most Root Complex can obey Finite Completion credit advertisement from Endpoint, however use this option with caution.	FALSE
axi_vip_in_exdes	AXI Verification IP in Example Design	New	Adds AXI VIP IP in example design when enabled	FALSE
ext_xvc_vsec_enable	Add the PCIe XVC-VSEC to the Example Design	New	Xilinx Virtual Cable enablement. Must be used in conjunction with <code>cfg_ext_if</code> parameter	FALSE



## Port Changes

The following parameters have changed from the AXI Bridge for PCIe Gen3 core to the DMA/Bridge Subsystem for PCIe in AXI Bridge mode.

*Table 70: Port Changes*

AXI Bridge for PCIe Gen3 Port Name	DMA/Bridge Subsystem for PCIe in AXI Bridge Mode Port Name	New/Changed/Removed
refclk	sys_clk	Changed
m_axi_*	m_axib_*	Changed
m_axi_wuser		Removed
m_axi_ruser		Removed
s_axi_*	s_axib_*	Changed
s_axi_wuser		Removed
s_axi_ruser		Removed
s_axi_ctl_*	s_axil_*	Changed
intx_msi_request	usr_irq_req	Changed
intx_msi_grant	usr_irq_ack	Changed
cfg_interrupt_msix_sent	cfg_interrupt_msi_sent	Changed
cfg_interrupt_msix_vf_enable		Removed
cfg_interrupt_msix_vf_mask		Removed

## Memory Map Changes

In DMA/Bridge Subsystem for PCIe in AXI Bridge mode, the Bridge Memory Map can be accessed through AXI4-Lite Control interface when address bit [28] is set to 1'b0. When address bit [28] is set to 1'b1, interrupt registers in DMA/Bridge Subsystem for PCIe Memory map is accessed.

*Table 71: Memory Map Changes*

AXI Bridge for PCIe Gen3 Memory Map	DMA/Bridge Subsystem for PCIe in AXI Bridge mode	New/Changed/Removed
Memory map	Interrupt Decode 2	New
	Interrupt Decode 2 Mask	New

## Upgrading in the Vivado Design Suite

This section provides the parameter and port changes that occur when you upgrade from AXI Bridge for PCIe Gen3 core v2.1 to 3.0 in the Vivado® Design Suite.

## Parameter Changes

The following table shows the new parameter added to the core in the current release.

*Table 72: Parameter Changes*

User Parameter Name	Display Name	New/ Change/ Removed	Details	Default Value
device_port_type	Device/Port Type	Change	Removed the Legacy_PCI_Express_Endpoint_device option from available options.	PCI_Express_Endpoint_device
tandem_mode	Tandem Configuration or Dynamic Function eXchange	Removed	This parameter has been replaced by the mcap_enablement parameter	None
mcap_enablement	Tandem Configuration or Dynamic Function eXchange	New	This is the new parameter added for Tandem mode selection. The valid values are None, Tandem, and Tandem with Field Updates.	None
en_ext_startup	Use an external STARTUP primitive	Removed	This is replaced by the ext_startup_primitive parameter.	False
ext_startup_primitive	Use an external STARTUP primitive	New	When this parameter unselected the STARTUP block will be generated internal to the IP when Tandem modes are selected.	False
pf0_base_class_menu	Base Class Menu	Change	Changed the default value.	Memory Controller
pf0_sub_class_interface_menu	Subclass Interface Menu	Change	Changed the default value.	Other Memory Controller
enable_jtag_dbg	Enable JTAG Debugger	New	When checked, this option enables the JTAG Debugging option of the subcore.	Unchecked
en_dbg_descramble	Enable Descrambler for Gen3 mode	New	This option enables the Enable Descrambler for Gen3 Mode option of the subcore and is valid only for Gen3 speeds.	Unchecked
enable_ibert	Enable In system IBERT	New	This option enables the 'In System IBERT' feature of the subcore.	Unchecked
en_l23_entry	Support PM_L23 Entry	New	When checked, supports PM_L23 Entry, and the PM_ENABLE_L23_ENTRY attribute will be set to TRUE.	Unchecked

## Port Changes

The following table shows the new port added to the core in the current release.

**Table 73: Port Changes**

Name	Direction	Width
common_commands_out	O	26 Bits <sup>1</sup>
pipe_tx_0_sigs	O	84 Bits <sup>2</sup>
pipe_tx_1_sigs	O	84 Bits <sup>2</sup>
pipe_tx_2_sigs	O	84 Bits <sup>2</sup>
pipe_tx_3_sigs	O	84 Bits <sup>2</sup>
pipe_tx_4_sigs	O	84 Bits <sup>2</sup>
pipe_tx_5_sigs	O	84 Bits <sup>2</sup>
pipe_tx_6_sigs	O	84 Bits <sup>2</sup>
pipe_tx_7_sigs	O	84 Bits <sup>2</sup>
gt_dmoniforeset <sup>3</sup>	I	(PL_LINK_CAP_MAX_LINK_WIDTH-1):0
gt_dmonitorclk <sup>3</sup>	I	(PL_LINK_CAP_MAX_LINK_WIDTH-1):0

**Notes:**

1. This signal width has changed to match common\_command\_in.
2. This external pipe interface signal width have changed to match pipe\_rx\*\_sigs.
3. This signal was added to transceiver debug and status ports interface and is applicable only for UltraScale variants.

The following table shows the `cfg_ext_if` signals which are available when the `CFG_EXT_IF` parameter is set to true in the AXI Bridge for PCIe Gen3 only.

**Table 74: New Ports**

Name	Direction	Width
cfg_ext_function_number	O	8 Bits
cfg_ext_read_data	I	32 Bits
cfg_ext_read_data_valid	I	1 Bit
cfg_ext_read_received	O	1 Bit
cfg_ext_register_number	O	10 Bits
cfg_ext_write_byte_enable	O	4 Bits
cfg_ext_write_data	O	32 Bits
cfg_ext_write_received	O	1 Bit

# Debugging

This appendix provides information for using the resources available on the Xilinx® Support website, debug tools, and other step-by-step processes for debugging designs that use the AXI Bridge for PCIe core.

---

## Finding Help on Xilinx.com

To help in the design and debug process when using the Bridge core, 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.

### Documentation

This product guide is the main document associated with the core. 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, see 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 PCI Express® Solution Center is located at [Xilinx Solution Center for PCI Express](#). Extensive debugging collateral is available in AR: [56802](#).

### 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 core are listed below, and can be located by using the Search Support box on the main [Xilinx support web page](#). To maximize your search results, use proper keywords, such as:

- the product name
- tool messages
- summary of the issue encountered

### ***Master Answer Record for the Subsystem***

AR: [61898](#)

## **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](#).

---

## **Debug Tools**

There are many tools available to address Bridge design issues. It is important to know which tools are useful for debugging various situations.

### **Vivado Design Suite Debug Feature**

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used to interact with the logic debug cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#)).

## Reference Boards

Various Xilinx development boards support the core. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series evaluation boards
  - VC709
- UltraScale™ evaluation boards
  - VCU108

## Third-Party Tools

This section describes third-party software tools that can be useful in debugging.

### LSPCI (Linux)

LSPCI is available on Linux platforms and allows you to view the PCI Express® device configuration space. LSPCI is usually found in the `/sbin` directory. LSPCI displays a list of devices on the PCI buses in the system. See the LSPCI manual for all command options. Some useful commands for debugging include:

- `lspci -x -d [<vendor>]: [<device>]`

This displays the first 64 bytes of configuration space in hexadecimal form for the device with vendor and device ID specified (omit the `-d` option to display information for all devices). The default Vendor/Device ID for Xilinx cores is 10EE:6012. Here is a sample of a read of the configuration space of a Xilinx device:

```
> lspci -x -d 10EE:6012
81:00.0 Memory controller: Xilinx Corporation: Unknown device 6012
00: ee 10 12 60 07 00 10 00 00 00 80 05 10 00 00 00
10: 00 00 80 fa 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 ee 10 6f 50
30: 00 00 00 00 40 00 00 00 00 00 00 00 05 01 00 00
```

Included in this section of the configuration space are the Device ID, Vendor ID, Class Code, Status and Command, and Base Address Registers.

- `lspci -xxxx -d [<vendor>]: [<device>]`

This displays the extended configuration space of the device. It can be useful to read the extended configuration space on the root and look for the Advanced Error Reporting (AER) registers. These registers provide more information on why the device has flagged an error (for example, it might show that a correctable error was issued because of a replay timer timeout).

- `lspci -k`

Shows kernel drivers handling each device and kernel modules capable of handling it (works with kernel 2.6 or later).

## Hardware Debug

### Transceiver Debug



**IMPORTANT!** The ports in the Transceiver Control And Status Interface must be driven in accordance with the appropriate GT user guide. Using the input signals listed below may result in unpredictable behavior of the IP core.

The following table describes the ports used to debug transceiver related issues when a 7 series device is targeted.

Table 75: Ports Used for Transceiver Debug

Port	I/O	Width	Description
pipe_txprbssel	I	3	PRBS input
pipe_rxprbssel	I	3	PRBS input
pipe_rxprbsforceerr	I	1	PRBS input
pipe_rxprbscntreset	I	1	PRBS input
pipe_loopback	I	1	PIPE loopback
pipe_rxprbserr	O	1	PRBS output
pipe_rst_fsm	O		Should be examined if PIPE_RST_IDLE is stuck at 0.
pipe_qrst_fsm	O		Should be examined if PIPE_RST_IDLE is stuck at 0.
pipe_sync_fsm_tx	O		Should be examined if PIPE_RST_FSM is stuck at 11'b10000000000, or PIPE_RATE_FSM is stuck at 24'b000100000000000000000000.
pipe_sync_fsm_rx	O		Deprecated.
pipe_drp_fsm	O		Should be examined if PIPE_RATE_FSM is stuck at 100000000.
pipe_rst_idle	O		Wrapper is in IDLE state if PIPE_RST_IDLE is High.
pipe_qrst_idle	O		Wrapper is in IDLE state if PIPE_QRST_IDLE is High.
pipe_rate_idle	O		Wrapper is in IDLE state if PIPE_RATE_IDLE is High.

Table 75: Ports Used for Transceiver Debug (cont'd)

Port	I/O	Width	Description
PIPE_DEBUG_0/gt_txresetdone	O		Generic debug ports to assist debug. These are generic debug ports to bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_1/gt_rxresetdone	O		Generic debug ports to assist debug. These are generic debug ports to bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_2/gt_phystatus	O		Generic debug ports to assist debug. These are generic debug ports to bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_3/gt_rxvalid	O		Generic debug ports to assist debug. These are generic debug ports to bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_4/gt_txphaligndone	O		Generic debug ports to assist debug. These generic debug ports bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_5/gt_rxphaligndone	O		Generic debug ports to assist debug. These generic debug ports bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_6/gt_rxcommadet	O		Generic debug ports to assist debug. These generic debug ports bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.



**Table 75: Ports Used for Transceiver Debug (cont'd)**

Port	I/O	Width	Description
PIPE_DEBUG_7/gt_rdy	O		Generic debug ports to assist debug. These generic debug ports bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_8/user_rx_converge	O		Generic debug ports to assist debug. These generic debug ports bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
PIPE_DEBUG_9/PIPE_TXELECIDLE	O		Generic debug ports to assist debug. These generic debug ports bring out internal PIPE Wrapper signals, such as raw GT signals. DEBUG_0 to DEBUGT_9 are intended for per lane signals. The bus width of these generic debug ports depends on the number of lanes configured in the wrapper.
pipe_txinhibit	I	1	Connects to TXINHIBIT on transceiver channel primitives.

The following table describes the ports used to debug transceiver related issues when an UltraScale™ device is targeted.

**Table 76: Ports Used for Transceiver Debug**

Port	I/O	Width	Description
gt_pcieuserratedone	I	1	Connects to PCIEUSERRATEDONE on transceiver channel primitives
gt_loopback	I	3	Connects to LOOPBACK on transceiver channel primitives
gt_txprbsforceerr	I	1	Connects to TXPRBSFORCEERR on transceiver channel primitives
gt_txinhibit	I	1	Connects to TXINHIBIT on transceiver channel primitives
gt_txprbssel	I	4	PRBS input
gt_rxprbssel	I	4	PRBS input
gt_rxprbscntreset	I	1	Connects to RXPRBSCNTRESET on transceiver channel primitives
gt_txelecidle	O	1	Connects to TXELECIDLE on transceiver channel primitives
gt_txresetdone	O	1	Connects to TXRESETDONE on transceiver channel primitives
gt_rxresetdone	O	1	Connects to RXRECLKOUT on transceiver channel primitives

Table 76: Ports Used for Transceiver Debug (cont'd)

Port	I/O	Width	Description
gt_rxpmaresetdone	O	1	Connects to TXPMARESETDONE on transceiver channel primitives
gt_txphaligndone	O	1	Connects to TXPHALIGNDONE of transceiver channel primitives
gt_txphinitdone	O	1	Connects to TXPHINITDONE of transceiver channel primitives
gt_txdlysresetdone	O	1	Connects to TXDLYSRESETDONE of transceiver channel primitives
gt_rxphaligndone	O	1	Connects to RXPHALIGNDONE of transceiver channel primitives
gt_rxdlysresetdone	O	1	Connects to RXDLYSRESETDONE of transceiver channel primitives
gt_rxsyncdone	O	1	Connects to RXYNCDONE of transceiver channel primitives
gt_eyescandataerror	O	1	Connects to EYESCANDATAERROR on transceiver channel primitives
gt_rxprbserr	O	1	Connects to RXPRBSERR on transceiver channel primitives
gt_dmonitorout	O	17	Connects to DMONITOROUT on transceiver channel primitives
gt_rxcommadet	O	1	Connects to RXCOMMADETEN on transceiver channel primitives
gt_phystatus	O	1	Connects to PHYSTATUS on transceiver channel primitives
gt_rxvalid	O	1	Connects to RXVALID on transceiver channel primitives
gt_rxcdrlock	O	1	Connects to RXCDRLOCK on transceiver channel primitives
gt_pcieerateidle	O	1	Connects to PCIERATEIDLE on transceiver channel primitives
gt_pcieuseratestart	O	1	Connects to PCIEUSERATESTART on transceiver channel primitives
gt_gtpowergood	O	1	Connects to GTPOWERGOOD on transceiver channel primitives
gt_cpplllock	O	1	Connects to CPLLLOCK on transceiver channel primitives
gt_rxoutclk	O	1	Connects to RXOUTCLK on transceiver channel primitives
gt_rxreclckout	O	1	Connects to RXRECLCKOUT on transceiver channel primitives
gt_qpll1lock	O	1	Connects to QPLL1LOCK on transceiver common primitives
gt_rxstatus	O	3	Connects to RXSTATUS on transceiver channel primitives
gt_rxbufstatus	O	3	Connects to RXBUFSTATUS on transceiver channel primitives
gt_bufgtdiv	O	9	Connects to BUFGTDIV on transceiver channel primitives

Table 76: Ports Used for Transceiver Debug (cont'd)

Port	I/O	Width	Description
phy_txeq_ctrl	O	2	PHY TX Equalization control bits
phy_txeq_prese	O	4	PHY TX Equalization Preset bits
phy_rst_fsm	O	4	PHY RST FSM state bits
phy_txeq_fsm	O	3	PHY RX Equalization FSM state bits (Gen3)
phy_rxeq_fsm	O	3	PHY TX Equalization FSM state bits (Gen3)
phy_rst_idle	O	1	PHY is in IDLE state
phy_rrst_n	O	1	Synchronized reset generation by <code>sys_clk</code>
phy_prst_n	O	1	Synchronized reset generation by <code>pipe_clk</code>

See *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)* for Transceiver Debug interface ports for Ultrascale+ devices when DMA/Bridge subsystem is enabled.

## Additional Debug Information

Additional debugging information can be found in the product guide for your device:

- *Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide (PG023)*.
- *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide (PG156)*.
- *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide (PG213)*.

## Interface Debug

### AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive ensure that the following conditions are met.

- For older versions of the AXI Bridge for PCIe Gen3 core with the `axi_ctl_aclk` port, ensure the `axi_ctl_aclk` and `axi_ctl_aclk_out` pins are connected to the design and are pulsing out of the IP.

- The interface is not being held in reset, and `axi_aresetn` is an active-Low reset.
- Ensure that the main core clocks are toggling and that the enables are also asserted.
- Has a simulation been run? Verify in simulation and/or a Vivado® Design Suite debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.


# Using the Xilinx Virtual Cable to Debug

The Xilinx® Virtual Cable (XVC) allows the Vivado® Design Suite to connect to FPGA debug cores through non-JTAG interfaces. The standard Vivado® Design Suite debug feature uses JTAG to connect to physical hardware FPGA resources and perform debug through Vivado. This section focuses on using XVC to perform debug over a PCIe® link rather than the standard JTAG debug interface. This is referred to as XVC-over-PCIe and allows for Vivado ILA waveform capture, VIO debug control, and interaction with other Xilinx debug cores using the PCIe link as the communication channel.

XVC-over-PCIe should be used to perform FPGA debug remotely using the Vivado Design Suite debug feature when JTAG debug is not available. This is commonly used for data center applications where the FPGA is connected to a PCIe Host system without any other connections to the hardware device.

Using debug over XVC requires software, driver, and FPGA hardware design components. Since there is an FPGA hardware design component to XVC-over-PCIe debug, you cannot perform debug until the FPGA is already loaded with an FPGA hardware design that implements XVC-over-PCIe and the PCIe link to the Host PC is established. This is normally accomplished by loading an XVC-over-PCIe enabled design into the configuration flash on the board prior to inserting the card into the data center location. Since debug using XVC-over-PCIe is dependent on the PCIe communication channel this should not be used to debug PCIe link related issue.

---

 **IMPORTANT!** XVC only provides connectivity to the debug cores within the FPGA. It does not provide the ability to program the device or access device JTAG and configuration registers. These operations can be performed through other standard Xilinx interfaces or peripherals such as the PCIe MCAP VSEC and HWICAP IP.

---

---

## Overview

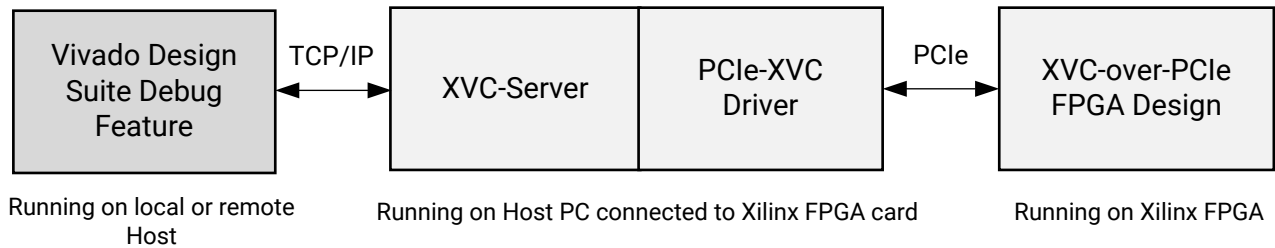
The main components that enable XVC-over-PCIe debug are as follows:

- Host PC XVC-Server application
- Host PC PCIe-XVC driver

- XVC-over-PCIe enabled FPGA design

These components are provided as a reference on how to create XVC connectivity for Xilinx FPGA designs. These three components are shown in the following figure and connect to the Vivado Design Suite debug feature through a TCP/IP socket.

Figure 55: XVC-over-PCIe Software and Hardware Components



X18837-032119

## Host PC XVC-Server Application

The `hw_server` application is launched by Vivado Design Suite when using the debug feature. Through the Vivado IDE you can connect `hw_server` to local or remote FPGA targets. This same interface is used to connect to local or remote PCIe-XVC targets as well. The Host PCIe XVC-Server application connects to the Xilinx `hw_server` using TCP/IP socket. This allows Vivado (using `hw_server`) and the XVC-Server application to be running on the same PC or separate PCs connected through Ethernet. The XVC-Server application needs to be run on a PC that is directly connected to the FPGA hardware resource. In this scenario the FPGA hardware is connected through PCIe® to a Host PC. The XVC-Server application connects to the FPGA hardware device through the PCIe-XVC driver that is also running on the Host PC.

## Host PC XVC-over-PCIe Driver

The XVC-over-PCIe driver provides connectivity to the PCIe enabled FPGA hardware resource that is connected to the Host PC. As such this is provided as a Linux kernel mode driver to access the PCIe hardware device, which is available in the following location,

`<Vivado_Installation_Path>/data/xicom/driver/pcie/xvc_pcie.zip`. The necessary components of this driver must be added to the driver that is created for a specific FPGA platform. The driver implements the basic functions needed by the XVC-Server application to communicate with the FPGA via PCIe.

## XVC-over-PCIe Enabled FPGA Design

Traditionally Vivado® debug is performed over JTAG. By default, Vivado tool automation connects the Xilinx debug cores to the JTAG BSCAN resource within the FPGA to perform debug. In order to perform XVC-over-PCIe debug, this information must be transmitted over the PCIe link rather than over the JTAG cable interface. The Xilinx Debug Bridge IP allows you to connect the debug network to PCIe through either the PCIe extended configuration interface (PCIe-XVC-VSEC) or through a PCIe BAR via an AXI4-Lite Memory Mapped interface (AXI-XVC).

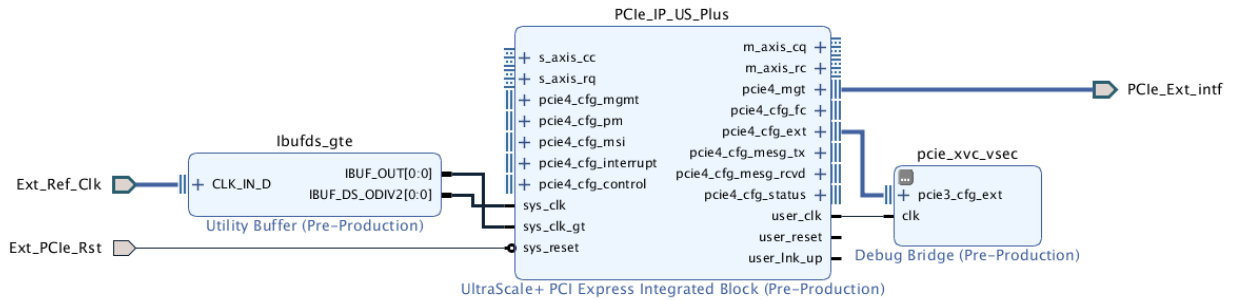
The Debug Bridge IP, when configured for **From PCIe to BSCAN** or **From AXI to BSCAN**, provides a connection point for the Xilinx® debug network from either the PCIe Extended Capability or AXI4-Lite interfaces respectively. Vivado tool automation connects this instance of the Debug Bridge to the Xilinx debug cores found in the design rather than connecting them to the JTAG BSCAN interface. There are design trade-offs to connecting the debug bridge to the PCIe Extended Configuration Space or AXI4-Lite. The following sections describe the implementation considerations and register map for both implementations.

### XVC-over-PCIe Through PCIe Extended Configuration Space (PCIe-XVC-VSEC)

Using the PCIe-XVC-VSEC approach, the Debug Bridge IP uses a PCIe Vendor Specific Extended Capability (VSEC) to implement the connection from PCIe to the Debug Bridge IP. The PCIe extended configuration space is set up as a linked list of extended capabilities that are discoverable from a Host PC. This is specifically valuable for platforms where one version of the design implements the PCIe-XVC-VSEC and another design implementation does not. The linked list can be used to detect the existence or absence of the PCIe-XVC-VSEC and respond accordingly.

The PCIe Extended Configuration Interface uses PCIe configuration transactions rather than PCIe memory BAR transactions. While PCIe configuration transactions are much slower, they do not interfere with PCIe memory BAR transactions at the PCIe IP boundary. This allows for separate data and debug communication paths within the FPGA. This is ideal if you expect to debug the datapath. Even if the datapath becomes corrupt or halted, the PCIe Extended Configuration Interface can remain operational to perform debug. The following figure describes the connectivity between the PCIe IP and the Debug Bridge IP to implement the PCIe-XVC-VSEC.

Figure 56: XVC-over-PCIe with PCIe Extended Capability Interface

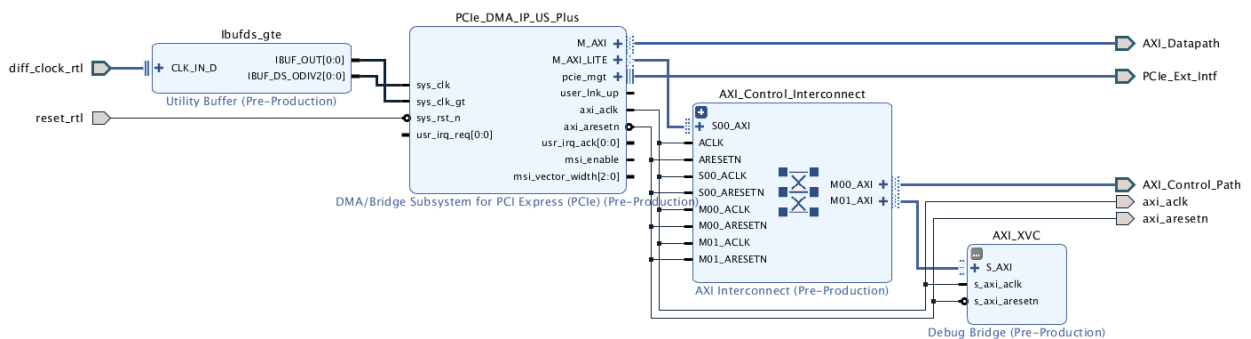


**Note:** Although the previous figure shows the UltraScale+™ Devices Integrated Block for PCIe IP, other PCIe IP (that is, the UltraScale™ Devices Integrated Block for PCIe, AXI Bridge for PCIe, or PCIe DMA IP) can be used interchangeably in this diagram.

## XVC-over-PCIe Through AXI (AXI-XVC)

Using the AXI-XVC approach, the Debug Bridge IP connects to the PCIe IP through an AXI Interconnect IP. The Debug Bridge IP connects to the AXI Interconnect like other AXI4-Lite Slave IPs and similarly requires that a specific address range be assigned to it. Traditionally the `debug_bridge` IP in this configuration is connected to the control path network rather than the system datapath network. The following figure describes the connectivity between the DMA Subsystem for PCIe IP and the Debug Bridge IP for this implementation.

Figure 57: XVC over PCIe with AXI4-Lite Interface



**Note:** Although the previous figure shows the PCIe DMA IP, any AXI-enabled PCIe IP can be used interchangeably in this diagram.

The AXI-XVC implementation allows for higher speed transactions. However, XVC debug traffic passes through the same PCIe ports and interconnect as other PCIe control path traffic, making it more difficult to debug transactions along this path. As result the AXI-XVC debug should be used to debug a specific peripheral or a different AXI network rather than attempting to debug datapaths that overlap with the AXI-XVC debug communication path.



## XVC-over-PCIe Register Map

The PCIe-XVC-VSEC and AXI-XVC have a slightly different register map that must be taken into account when designing XVC drivers and software. The register maps in the following tables show the byte-offset from the base address.

- The PCIe-XVC-VSEC base address must fall within the valid range of the PCIe Extended Configuration space. This is specified in the Debug Bridge IP configuration.
- The base address of an AXI-XVC Debug Bridge is the offset for the Debug Bridge IP peripheral that was specified in the Vivado Address Editor.

The following tables describe the register map for the Debug Bridge IP as an offset from the base address when configured for the **From PCIe-Ext to BSCAN** or **From AXI to BSCAN** modes.

**Table 77: Debug Bridge for XVC-PCIe-VSEC Register Map**

Register Offset	Register Name	Description	Register Type
0x00	PCIe Ext Capability Header	PCIe defined fields for VSEC use.	Read Only
0x04	PCIe VSEC Header	PCIe defined fields for VSEC use.	Read Only
0x08	XVC Version Register	IP version and capabilities information.	Read Only
0x0C	XVC Shift Length Register	Shift length.	Read Write
0x10	XVC TMS Register	TMS data.	Read Write
0x14	XVC TDIO Register	TDO/TDI data.	Read Write
0x18	XVC Control Register	General control register.	Read Write
0x1C	XVC Status Register	General status register.	Read Only

**Table 78: Debug Bridge for AXI-XVC Register Map**

Register Offset	Register Name	Description	Register Type
0x00	XVC Shift Length Register	Shift length.	Read Write
0x04	XVC TMS Register	TMS data.	Read Write
0x08	XVC TDI Register	TDI data.	Read Write
0x0C	XVC TDO Register	TDO data.	Read Only
0x10	XVC Control Register	General control register.	Read Write
0x14	XVC Status Register	General status register.	Read Only
0x18	XVC Version Register	IP version and capabilities information.	Read Only

## PCIe Ext Capability Header

This register is used to identify the PCIe-XVC-VSEC added to a PCIe design. The fields and values in the PCIe Ext Capability Header are defined by PCI-SIG and are used to identify the format of the extended capability and provide a pointer to the next extended capability, if applicable. When used as a PCIe-XVC-VSEC, the appropriate PCIe ID fields should be evaluated prior to interpretation. These can include PCIe Vendor ID, PCIe Device ID, PCIe Revision ID, Subsystem Vendor ID, and Subsystem ID. The provided drivers specifically check for a PCIe Vendor ID that matches Xilinx (0x10EE) before interpreting this register. The following table describes the fields within this register.

Table 79: PCIe Ext Capability Header Register Description

Bit Location	Field	Description	Initial Value	Type
15:0	PCIe Extended Capability ID	This field is a PCI-SIG defined ID number that indicates the nature and format of the Extended Capability. The Extended Capability ID for a VSEC is 0x000B	0x000B	Read Only
19:16	Capability Version	This field is a PCI-SIG defined version number that indicates the version of the capability structure present. Must be 0x1 for this version of the specification.	0x1	Read Only
31:20	Next Capability Offset	This field is passed in from the user and contains the offset to the next PCI Express Capability structure or 0x000 if no other items exist in the linked list of capabilities. For Extended Capabilities implemented in the PCIe extended configuration space, this value must always be within the valid range of the PCIe Extended Configuration space.	0x000	Read Only

## PCIe VSEC Header (PCIe-XVC-VSEC only)

This register is used to identify the PCIe-XVC-VSEC when the Debug Bridge IP is in this mode. The fields are defined by PCI-SIG, but the values are specific to the Vendor ID (0x10EE for Xilinx). The PCIe Ext Capability Header register values should be qualified prior to interpreting this register.

Table 80: PCIe XVC VSEC Header Register Description

Bit Location	Field	Description	Initial Value	Type
15:0	VSEC ID	This field is the ID value that can be used to identify the PCIe-XVC-VSEC and is specific to the Vendor ID (0x10EE for Xilinx).	0x0008	Read Only
19:16	VSEC Rev	This field is the Revision ID value that can be used to identify the PCIe-XVC-VSEC revision.	0x0	Read Only
31:20	VSEC Length	This field indicates the number of bytes in the entire PCIe-XVC-VSEC structure, including the PCIe Ext Capability Header and PCIe VSEC Header registers.	0x020	Read Only

### ***XVC Version Register (PCIe-XVC-VSEC only)***

This register is populated by the Xilinx tools and is used by the Vivado Design Suite to identify the specific features of the Debug Bridge IP that is implemented in the hardware design.

### ***XVC Shift Length Register***

This register is used to set the scan chain shift length within the debug scan chain.

### ***XVC TMS Register***

This register is used to set the TMS data within the debug scan chain.

### ***XVC TDO/TDI Data Register(s)***

This register is used for TDO/TDI data access. When using PCIe-XVC-VSEC, these two registers are combined into a single field. When using AXI-XVC, these are implemented as two separate registers.

### ***XVC Control Register***

This register is used for XVC control data.

### ***XVC Status Register***

This register is used for XVC status information.

## **XVC Driver and Software**

Example XVC driver and software has been provided with the Vivado Design Suite installation, which is available at the following location: `<Vivado_Installation_Path>/data/xicom/driver/pcie/xvc_pcie.zip`. This should be used for reference when integrating the XVC capability into Xilinx FPGA platform design drivers and software. The provided Linux kernel mode driver and software implement XVC-over-PCIe debug for both PCIe-XVC-VSEC and AXI-XVC debug bridge implementations.

When operating in PCIe-XVC-VSEC mode, the driver will initiate PCIe configuration transactions to interface with the FPGA debug network. When operating in AXI-XVC mode, the driver will initiate 32-bit PCIe Memory BAR transactions to interface with the FPGA debug network. By default, the driver will attempt to discover the PCIe-XVC-VSEC and use AXI-XVC if the PCIe-XVC-VSEC is not found in the PCIe configuration extended capability linked list.

The driver is provided in the data directory of the Vivado installation as a .zip file. This .zip file should be copied to the Host PC connected through PCIe to the Xilinx FPGA and extracted for use. README.txt files have been included; review these files for instructions on installing and running the XVC drivers and software.

## Special Considerations for Tandem or Dynamic Function eXchange Designs

Tandem Configuration and Dynamic Function eXchange (DFX) designs may require additional considerations as these flows partition the physical resources into separate regions. These physical partitions should be considered when adding debug IPs to a design, such as VIO, ILA, MDM, and MIG-IP. A Debug Bridge IP configured for **From PCIe-ext to BSCAN** or **From AXI to BSCAN** should only be placed into the static partition of the design. When debug IPs are used inside of a DFX or Tandem Field Updates region, an additional debug BSCAN interface should be added to the dynamic region module definition and left unconnected in the dynamic region module instantiation.

To add the BSCAN interface to the Reconfigurable Partition definition the appropriate ports and port attributes should be added to the Reconfigurable Partition definition. The sample Verilog provided below can be used as a template for adding the BSCAN interface to the port declaration.

```

...
// BSCAN interface definition and attributes.
// This interface should be added to the DFX module definition
// and left unconnected in the DFX module instantiation.
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN drck" *)
(* DEBUG="true" *)
input S_BSCAN_drck,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN shift" *)
(* DEBUG="true" *)
input S_BSCAN_shift,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN tdi" *)
(* DEBUG="true" *)
input S_BSCAN_tdi,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN update" *)
(* DEBUG="true" *)
input S_BSCAN_update,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN sel" *)
(* DEBUG="true" *)
input S_BSCAN_sel,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN tdo" *)
(* DEBUG="true" *)
output S_BSCAN_tdo,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN tms" *)
(* DEBUG="true" *)
input S_BSCAN_tms,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN tck" *)
(* DEBUG="true" *)
input S_BSCAN_tck,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN runtest" *)
(* DEBUG="true" *)
input S_BSCAN_runtest,

```

```
( * X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN reset" * )
( * DEBUG="true" * )
input S_BSCAN_reset,
( * X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN capture" * )
( * DEBUG="true" * )
input S_BSCAN_capture,
( * X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN bscanid_en" * )
( * DEBUG="true" * )
input S_BSCAN_bscanid_en,
....
```

When `link_design` is run, the exposed ports are connected to the static portion of the debug network through tool automation. The ILAs are also connected to the debug network as required by the design. There might also be an additional `dbg_hub` cell that is added at the top level of the design. For Tandem PCIe with Field Updates designs, the `dbg_hub` and tool inserted clock buffer(s) must be added to the appropriate design partition. The following is an example of the Tcl commands that can be run after `opt_design` to associate the `dbg_hub` primitives with the appropriate design partitions.

```
# Add the inserted dbg_hub cell to the appropriate design partition.
set_property HD.TANDEM_IP_PBLOCK Stage1_Main [get_cells dbg_hub]
# Add the clock buffer to the appropriate design partition.
set_property HD.TANDEM_IP_PBLOCK Stage1_Config_IO [get_cells
dma_pcie_0_support_i/
pcie_ext_cap_i/vsec_xvc_inst/vsec_xvc_dbg_bridge_inst/inst/bsip/ins
t/USE_SOFTBSCAN.U_TAP_TCKBUFG]
```

---

## Using the PCIe-XVC-VSEC Example Design

The PCIe-XVC-VSEC has been integrated into the PCIe example design as part of the Advanced settings for the UltraScale+™ Integrated Block for PCIe IP. This section provides instruction of how to generate the PCIe example design with the PCIe-XVC-VSEC, and then debug the FPGA through PCIe using provided XVC drivers and software. This is an example for using XVC in customer applications. The FPGA design, driver, and software elements will need to be integrated into customer designs.

### Generating a PCIe-XVC-VSEC Example Design

The PCIe-XVC-VSEC can be added to the UltraScale+™ PCIe example design by selecting the following options.

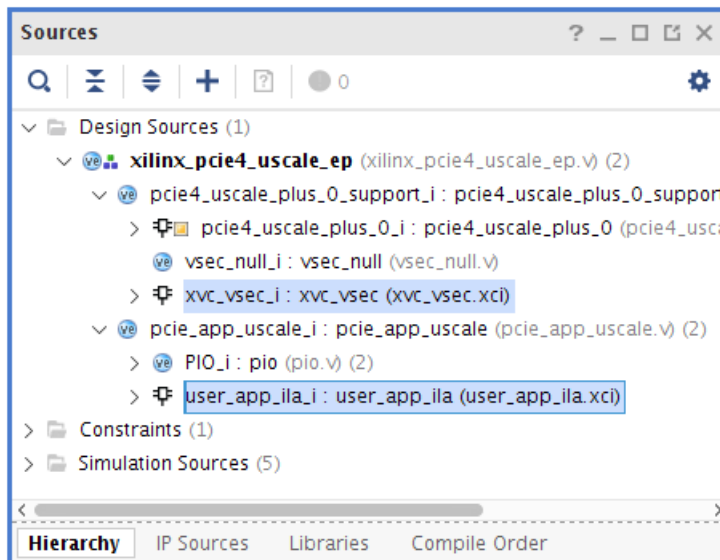
1. Configure the core to the desired configuration.
2. On the Basic tab, select the **Advanced Mode**.

3. On the Adv. Options-3 tab:
  - a. Select the **PCI Express Extended Configuration Space Enable** checkbox to enable the PCI Express extended configuration interface. This is where additional extended capabilities can be added to the PCI Express core.
  - b. Select the **Add the PCIe-XVC-VSEC to the Example Design** checkbox to enable the PCIe-XVC-VSEC in the example design generation.
4. Verify the other configuration selections for the PCIe IP. The following selections are needed to configure the driver for your hardware implementation:
  - PCIe Vendor ID (0x10EE for Xilinx)
  - PCIe Device ID (dependent on user selection)
5. Click **OK** to finalize the selection and generate the IP.
6. Generate the output products for the IP as desired for your application.
7. In the Sources window, right-click the IP and select **Open IP Example Design**.
8. Select a directory for generating the example design, and select **OK**.

After being generated, the example design shows that:

- the PCIe IP is connected to `xvc_vsec` within the support wrapper, and
- an ILA IP is added to the user application portion of the design.

This demonstrates the desired connectivity for the hardware portion of the FPGA design. Additional debug cores can be added as required by your application.



**Note:** Although the previous figure shows to the UltraScale+ Devices Integrated Block for PCIe IP, the example design hierarchy is the same for other PCIe IPs.

9. Double-click the Debug Bridge IP identified as `xvc_vsec` to view the configuration option for this IP. Make note of the following configuration parameters because they will be used to configure the driver.
  - PCIe XVC VSEC ID (default `0x0008`)
  - PCIe XVC VSEC Rev ID (default `0x0`)



**IMPORTANT!** Do not modify these parameter values when using a Xilinx Vendor ID or provided XVC drivers and software. These values are used to detect the XVC extended capability. (See the PCIe specification for additional details.)

10. In the Flow Navigator, click **Generate Bitstream** to generate a bitstream for the example design project. This bitstream will be then be loaded onto the FPGA board to enable XVC debug over PCIe.

After the XVC-over-PCIe hardware design has been completed, an appropriate XVC enabled PCIe driver and associated XVC-Server software application can be used to connect the Vivado Design Suite to the PCIe connected FPGA. Vivado can connect to an XVC-Server application that is running local on the same Machine or remotely on another machine using a TCP/IP socket.

## System Bring-Up

The first step is to program the FPGA and power on the system such that the PCIe link is detected by the host system. This can be accomplished by either:

- Programming the design file into the flash present on the FPGA board, or
- Programming the device directly via JTAG.

If the card is powered by the Host PC, it will need to be powered on to perform this programming using JTAG and then re-started to allow the PCIe link to enumerate. After the system is up and running, you can use the Linux `lspci` utility to list out the details for the FPGA-based PCIe device.

## Compiling and Loading the Driver

The provided PCIe drivers and software should be customized to a specific platform. To accomplish this, drivers and software are normally developed to verify the Vendor ID, Device ID, Revision ID, Subsystem Vendor ID, and Subsystem ID before attempting to access device-extended capabilities or peripherals like the PCIe-XVC-VSEC or AXI-XVC. Because the provided driver is generic, it only verifies the Vendor ID and Device ID for compatibility before attempting to identify the PCIe-XVC-VSEC or AXI-XVC peripheral.

The XVC driver and software are provide as a ZIP file included with the Vivado Design Suite installation.

1. Copy the ZIP file from the Vivado install directory to the FPGA connected Host PC and extract (unzip) its contents. This file is located at the following path within the Vivado installation directory.

```
XVC Driver and SW Path: .../data/xicom/driver/pcie/xvc_pcie.zip
```

The `README.txt` files within the `driver_*` and `xvcserver` directories identify how to compile, install, and run the XVC drivers and software, and are summarized in the following steps. Follow the following steps after the driver and software files have been copied to the Host PC and you are logged in as a user with root permissions.

2. Modify the variables within the `driver_*/xvc_pcie_user_config.h` file to match your hardware design and IP settings. Consider modifying the following variables:

- **PCIE\_VENDOR\_ID:** The PCIe Vendor ID defined in the PCIe® IP customization.
- **PCIE\_DEVICE\_ID:** The PCIe Device ID defined in the PCIe® IP customization.
- **Config\_space:** Allows for the selection between using a PCIe-XVC-VSEC or an AXI-XVC peripheral. The default value of AUTO first attempts to discover the PCIe-XVC-VSEC, then attempts to connect to an AXI-XVC peripheral if the PCIe-XVC-VSEC is not found. A value of CONFIG or BAR can be used to explicitly select between PCIe®-XVC-VSEC and AXI-XVC implementations, as desired.
- **config\_vsec\_id:** The PCIe XVC VSEC ID (default `0x0008`) defined in the Debug Bridge IP when the Bridge Type is configured for From PCIe to BSCAN. This value is only used for detection of the PCIe®-XVC-VSEC.
- **config\_vsec\_rev:** The PCIe XVC VSEC Rev ID (default `0x0`) defined in the Debug Bridge IP when the Bridge Type is configured for From PCIe to BSCAN. This value is only used for detection of the PCIe-XVC-VSEC.
- **bar\_index:** The PCIe BAR index that should be used to access the Debug Bridge IP when the Bridge Type is configured for From AXI to BSCAN. This BAR index is specified as a combination of the PCIe IP customization and the addressable AXI peripherals in your system design. This value is only used for detection of an AXI-XVC peripheral.
- **bar\_offset:** PCIe BAR Offset that should be used to access the Debug Bridge IP when the Bridge Type is configured for From AXI to BSCAN. This BAR offset is specified as a combination of the PCIe IP customization and the addressable AXI peripherals in your system design. This value is only used for detection of an AXI-XVC peripheral.

3. Move the source files to the directory of your choice. For example, use:

```
/home/username/xil_xvc or /usr/local/src/xil_xvc
```

4. Make sure you have root permissions and change to the directory containing the driver files.

```
# cd /driver_*/
```

5. Compile the driver module:

```
# make install
```

The kernel module object file will be installed as:



```
/lib/modules/[KERNEL_VERSION]/kernel/drivers/pci/pcie/Xilinx/  
xil_xvc_driver.ko
```

6. Run the `depmod` command to pick up newly installed kernel modules:

```
# depmod -a
```

7. Make sure no older versions of the driver are loaded:

```
# modprobe -r xil_xvc_driver
```

8. Load the module:

```
# modprobe xil_xvc_driver
```

If you run the `dmesg` command, you will see the following message:

```
kernel: xil_xvc_driver: Starting...
```

**Note:** You can also use `insmod` on the kernel object file to load the module:

```
# insmod xil_xvc_driver.ko
```

However, this is not recommended unless necessary for compatibility with older kernels.

9. The resulting character file, `/dev/xil_xvc/cfg_ioc0`, is owned by user `root` and group `root`, and it will need to have permissions of `660`. Change permissions on this file if it does not allow the application to interact with the driver.

```
# chmod 660 /dev/xil_xvc/cfg_ioc0
```

10. Build the simple test program for the driver:

```
# make test
```

11. Run the test program:

```
# ./driver_test/verify_xil_xvc_driver
```

You should see various successful tests of differing lengths, followed by the following message:

```
"XVC PCIE Driver Verified Successfully!"
```

## Compiling and Launching the XVC-Server Application

The XVC-Server application provides the connection between the Vivado HW server and the XVC enabled PCIe device driver. The Vivado Design Suite connects to the XVC-Server using TCP/IP. The desired port number will need to be exposed appropriately through the firewalls for your network. The following steps can be used to compile and launch the XVC software application, using the default port number of `10200`.

1. Make sure the firewall settings on the system expose the port that will be used to connect to the Vivado Design Suite. For this example, port `10200` is used.

2. Make note of the host name or IP address. The host name and port number will be required to connect Vivado to the `xvcserver` application. See the OS help pages for information regarding the firewall port settings for your OS.

3. Move the source files to the directory of your choice. For example, use:

```
/home/username/xil_xvc or /usr/local/src/xil_xvc
```

4. Change to the directory containing the application source files:

```
# cd ./xvcserver/
```

5. Compile the application:

```
# make
```

6. Start the XVC-Server application:

```
# ./bin/xvc_pcie -s TCP::10200
```

After the Vivado Design Suite has connected to the XVC-server application you should see the following message from the XVC-server.

```
Enable verbose by setting VERBOSE env var.  
Opening /dev/xil_xvc/cfg_ioc0
```

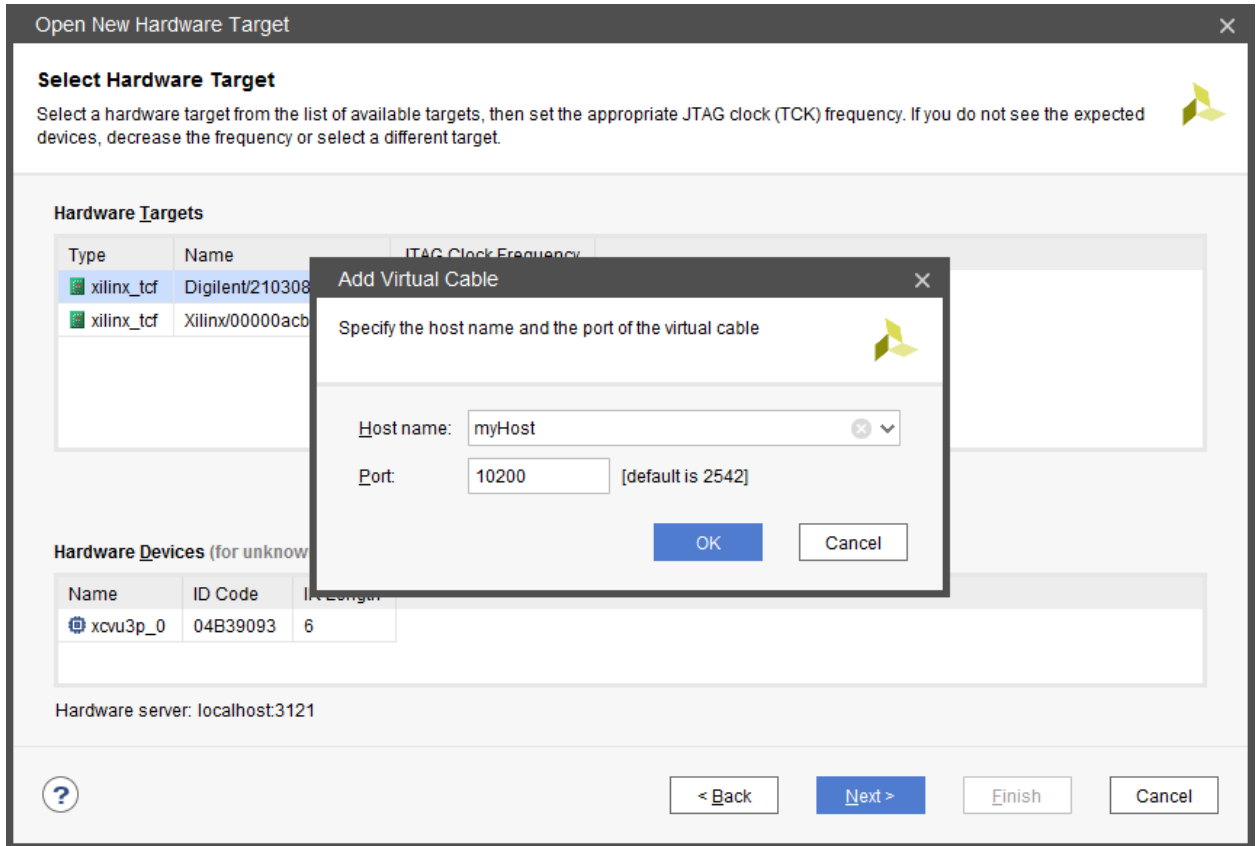
## Connecting the Vivado Design Suite to the XVC-Server Application

The Vivado Design Suite can be run on the computer that is running the XVC-server application, or it can be run remotely on another computer that is connected over an Ethernet network. The port however must be accessible to the machine running Vivado. To connect Vivado to the XVC-Server application follow the steps should be used and are shown using the default port number.

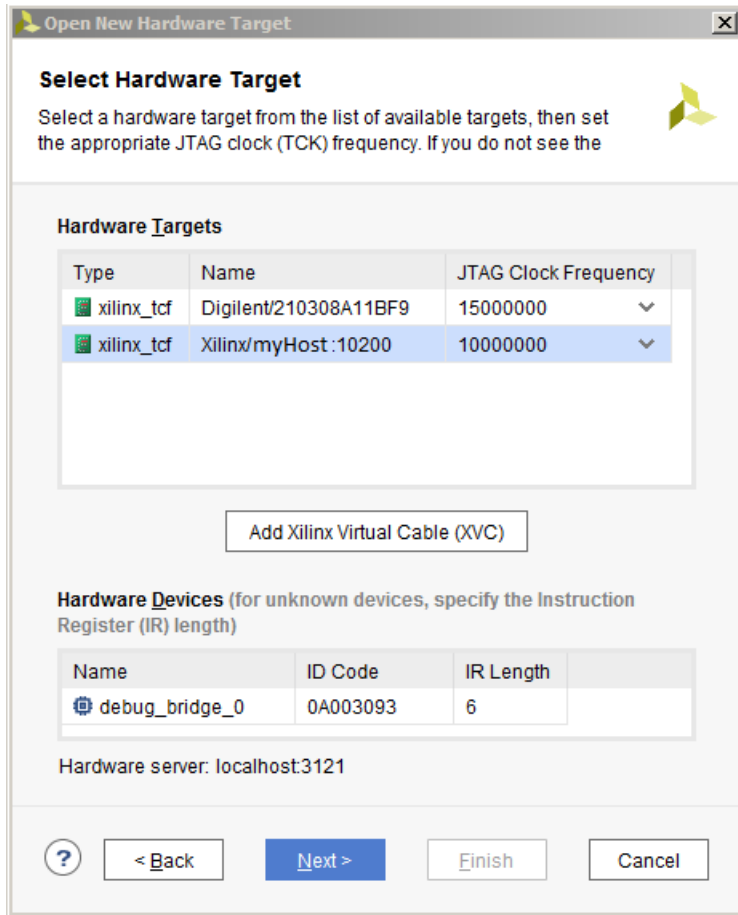
1. Launch the Vivado Design Suite.
2. Select **Open HW Manager**.
3. In the Hardware Manager, select **Open target** → **Open New Target**.
4. Click **Next**.
5. Select **Local server**, and click **Next**.

This launches `hw_server` on the local machine, which then connects to the `xvcserver` application.

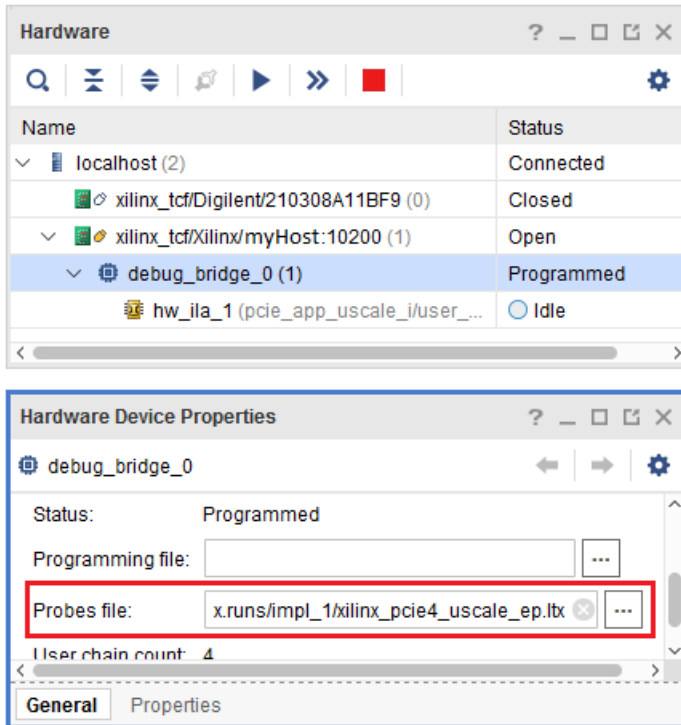
6. Select **Add Xilinx Virtual Cable (XVC)**.
7. In the Add Virtual Cable dialog box, type in the appropriate Host name or IP address, and Port to connect to the `xvcserver` application. Click **OK**.



8. Select the newly added XVC target from the Hardware Targets table, and click **Next**.



9. Click **Finish**.
10. In the Hardware Device Properties panel, select the debug bridge target, and assign the appropriate probes .1tx file.



Vivado now recognizes your debug cores and debug signals, and you can debug your design through the Vivado hardware tools interface using the standard debug approach.

This allows you to debug Xilinx FPGA designs through the PCIe connection rather than JTAG using the Xilinx Virtual Cable technology. You can terminate the connection by closing the hardware server from Vivado using the right-click menu. If the PCIe connection is lost or the XVC-Server application stops running, the connection to the FPGA and associated debug cores will also be lost.

## Run Time Considerations

The Vivado connection to an XVC-Server Application should not be running when a device is programmed. The XVC-Server Application along with the associated connection to Vivado should only be initiated after the device has been programmed and the hardware PCIe interface is active.

For DFX designs, it is important to terminate the connection during DFX operations. During a DFX operation where debug cores are present inside the dynamic region, a portion of the debug tree is expected to be reprogrammed. Vivado debug tools should not be actively communicating with the FPGA through XVC during a DFX operation.

# CCIX Interface

This appendix covers the following aspects of the CCIX interface:

- Supported Configurations
- Port Descriptions
- CCIX Activation/Deactivation
- CCIX-VC1 Credits
- Example Figures

## Supported Configurations

The following table lists the link widths, the required core clock frequency and speed grade.

*Table 81: Minimum Device Requirements*

Capability Link Speed	Capability Link Width	usr_clk (MHz)	Supported Speed Grades	CXS Width (bits)
Gen3	x8	250	All	256
	x16	250	3E (VCCINT = 0.90V), -2E (VCCINT = 0.85V), and -2I (VCCINT = 0.85V)	512
Gen4	x4	250	All	256
	x8	250	-3E (VCCINT = 0.90V), -2E (VCCINT = 0.85V), and -2I (VCCINT = 0.85V)	512

# Port Descriptions

## CCIX Core Interfaces

In addition to status, control and AXI4-Stream interfaces, the core has two CCIX interfaces used to transfer and receive transactions.

The 256-bit interfaces are:

- Transmit CCIX Core Interface (256-Bit)
- Receive CCIX Core Interface (256-Bit)

The 512-bit interfaces are:

- Transmit CCIX Core Interface (512-Bit)
- Receive CCIX Core Interface (512-Bit)

### ***Transmit CCIX Core Interface (256-Bit)***

The TX CCIX interface contains ports through which all user application CCIX data is transmitted. The following table defines the ports in the 256-bit transmit CCIX interface.

**Table 82: 256-Bit Transmit CCIX Interface Port Descriptions**

Port	Direction	Width	Description
s_axis_ccix_tx_tdata	Input	256	Receive data to the core from CCIX application.
s_axis_ccix_tx_tuser	Input	45	This set of signals contain sideband information on the TLP being transferred. These signals are valid when s_axis_ccix_tx_tvalid is High. The individual signals are defined in the following table.
s_axis_ccix_tx_tvalid	Input	1	The CCIX application asserts this signal whenever it presents valid data on s_axis_ccix_tx_tdata.
ccix_tx_credit_gnt	Output	1	<p>This signal is asserted by the core to release credits. The core has a total of eight TX credits. After the VC1 link becomes active, the core first releases all the eight credits to the CCIX application, and then the core releases a credit as available. The CCIX application can transmit valid data when the application has credits available.</p> <p><b>Note:</b> If the CCIX application drops valid between transmitting TLP packets, the application must grantee the continuation of TLP packets after ccix_tx_credit_gnt goes High at the core interface.</p> <p>Or</p> <p>The CCIX application can wait to accumulate the needed amount of credits before transmitting a TLP packet in order to avoid valid drop between packet.</p>

Table 82: 256-Bit Transmit CCIX Interface Port Descriptions (cont'd)

Port	Direction	Width	Description
ccix_tx_credit_rtn	Input	1	The CCIX application asserts this signal to return the credit to the core when the CCIX link is going to deactivate state.
ccix_tx_active_req	Input	1	This signal is asserted by CCIX application to move out of deactive (stop) state and into active state. This signal must remain High as long as the CCIX application requires the link to be in active state.
ccix_tx_active_ack	Output	1	The core asserts this signal in response to the ccix_tx_active_req from the CCIX application when the core is ready to accept packets being sent.
ccix_tx_deact_hint	Output	1	The core asserts this signal when the PCIe link want to go through a link reset, the CCIX application must use this signal to return the credits and move to link to deactivate state.

Table 83: Sideband Signals in s\_axis\_ccix\_tx\_tuser (256-Bit Interface)

Bit Index	Name	Width	Description
1:0	is_sop[1:0]	2	Signals the start of a TLP in this beat. The encoding is as follows: <ul style="list-style-type: none"> <li>• 00: No new TLP starting in this beat.</li> <li>• 01: A single new TLP starts in this beat. Its start position is indicated by is_sop0_ptr.</li> <li>• 11: Two new TLPs are starting in this beat. The is_sop0_ptr signal provides the start position of the first TLP. The is_sop1_ptr signal provides the start position of the second TLP.</li> <li>• 10: Reserved.</li> </ul>
2	is_sop0_ptr	1	Indicates the position of the first byte of the first TLP starting in this beat: <ul style="list-style-type: none"> <li>• 0: Byte 0</li> <li>• 1: Byte 16</li> </ul>
3	is_sop1_ptr	1	Indicates the position of the first byte of the second TLP starting in this beat: <ul style="list-style-type: none"> <li>• 0: Reserved</li> <li>• 1: Byte 16</li> </ul>



**Table 83: Sideband Signals in s\_axis\_ccix\_tx\_tuser (256-Bit Interface) (cont'd)**

Bit Index	Name	Width	Description
5:4	is_eop[1:0]	2	<p>Signals that a TLP is ending in this beat. These inputs are set in the final beat of a TLP. The encoding are as follows:</p> <ul style="list-style-type: none"> <li>• 00: No TLPs ending in this beat.</li> <li>• 01: A single TLP is ending in this beat. The is_eop0_ptr[2:0] signal provides the offset of the last Dword of this TLP.</li> <li>• 11: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[2:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[2:0] provides the offset of the last Dword of the second TLP.</li> </ul> </li> <li>• 10: Reserved.</li> </ul>
7:6	discontinue[1:0]	2	<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. 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.</p> <p>The discontinue signal can be asserted only when s_axis_ccix_tx_tvalid is High. Thus, once asserted, it should not be deasserted until the end of the packet.</p>
10:8	is_eop0_ptr[2:0]	3	Indicates the offset of the last Dword of the first TLP ending in this beat. This input is valid when is_eop[0] is asserted.
13:11	is_eop1_ptr[2:0]	3	Indicates the offset of the last Dword of the second TLP ending in this beat. This input is valid when is_eop[1] is asserted.
45:14	data_parity[31:0]	32	<p>Odd parity for the 256-bit data. When parity checking is enabled in the core, the user logic must set bit i of this bus to the odd parity computed for byte i of s_axis_ccix_tx_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>

### Receive CCIX Core Interface (256-Bit)

The RX CCIX interface are the ports through which the core receives CCIX data from the user application. The following defines the ports in the 256-bit receive CCIX interface.

**Table 84: 256-Bit Receive CCIX Interface Port Descriptions**

Port	Direction	Width	Description
m_axis_ccix_rx_tdata	Output	256	The transmit data from core to CCIX application.
m_axis_ccix_rx_tuser	Output	45	This set of signals contain sideband information on TLP being transferred. These signals are valid when m_axis_ccix_rx_tvalid is High. The individual signals are defined in the following table.
m_axis_ccix_rx_tvalid	Output	1	The core asserts this signal whenever it presents valid data on m_axis_ccix_rx_tdata.

**Table 84: 256-Bit Receive CCIX Interface Port Descriptions (cont'd)**

Port	Direction	Width	Description
ccix_rx_credit_gnt	Input	1	This signal is asserted by the user application to release credits to the core. After the VC1 link becomes active, the user application first releases all the available credits to the core, and then it releases credits as available. The core receives valid data when there are credits available. Note: The core can drop valid between packets if there are no available credits.
ccix_rx_credit_rtn	Output	1	The core asserts this signal to return the credits to the CCIX application when the CCIX link is going to deactivate state.
ccix_rx_active_req	Output	1	This signal is asserted by the core to move out of deactive(stop) state and into active state. This signal must remain High as long the core requires the link to be in active state.
ccix_rx_active_ack	Input	1	The CCIX application asserts this signal in response to the ccix_rx_active_req from the core when the application is ready to accept TLP packets
ccix_rx_deact_hint	Input	1	The CCIX application asserts this signal when the application wants to go through a link reset. The core must use this signal to return the credits and move to link deactivate state.

**Table 85: Sideband Signals in s\_axis\_ccix\_rx\_tuser (256-Bit Interface)**

Bit Index	Name	Width	Description
1:0	is_sop[1:0]	2	Signals the start of a TLP in this beat. The encoding is as follows <ul style="list-style-type: none"> <li>• 00: No new TLP starting in this beat</li> <li>• 01: A single new TLP starts in this beat. The start position is indicated by is_sop0_ptr.</li> <li>• 11: Two new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> </ul> </li> <li>• 10: Resrved</li> </ul>
2	is_sop0_ptr	1	Indicates the position of the first byte of the first TLP starting in this beat: <ul style="list-style-type: none"> <li>• 0: Byte 0</li> <li>• 1: Byte 16</li> </ul>
3	is_sop1_ptr	1	Indicates the position of the first byte of the second TLP starting in this beat: <ul style="list-style-type: none"> <li>• 0: Reserved</li> <li>• 1: Byte 16</li> </ul>

Table 85: Sideband Signals in s\_axis\_ccix\_rx\_tuser (256-Bit Interface) (cont'd)

Bit Index	Name	Width	Description
5:4	is_eop[1:0]	2	<p>Signals that a TLP is ending in this beat. These outputs are set in the final beat of a TLP. The encoding are as follows:</p> <ul style="list-style-type: none"> <li>• 00: No TLPs ending in this beat.</li> <li>• 01: A single TLP is ending in this beat. The is_eop0_ptr[2:0] signal provides the offset of the last Dword of this TLP.</li> <li>• 11: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[2:0] signal provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[2:0] provides the offset of the last Dword of the second TLP.</li> </ul> </li> <li>• 10: Reserved.</li> </ul>
7:6	discontinue[1:0]	2	<p>This signal can be asserted by the core during a transfer if it has detected an error in the data being transferred and needs to abort the packet. The user logic nullifies the corresponding TLP on the link to avoid data corruption.</p> <p>The core asserts this signal in any beat of a 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.</p> <p>The discontinue signal can be asserted only when s_axis_ccix_rx_tvalid is high.</p>
10:8	is_eop0_ptr[2:0]	3	Indicates the offset of the last Dword of the first TLP ending in this beat. This output is valid when is_eop[0] is asserted.
13:11	is_eop1_ptr[2:0]	3	Indicates the offset of the last Dword of the second TLP ending in this beat. This output is valid when is_eop[1] is asserted.
45:14	data_parity[31:0]	32	<p>Odd parity for the 256-bit data. Core sets bit i of this bus to the odd parity computed for byte i of s_axis_ccix_rx_tdata.</p> <p>On detection of a parity error, the user application can nullifies the corresponding TLP on the link and treat it as an Uncorrectable Internal Error.</p>

### Transmit CCIX Core Interface (512-Bit)

The TX CCIX interface contains the ports through which all user application CCIX data is transmitted. The following table defines the ports in the 512-bit transmit CCIX interface.

Table 86: 512-Bit Transmit CCIX Interface Port Descriptions

Port	Direction	Width	Description
s_axis_ccix_tx_tdata	Input	512	Receives data to the core from the CCIX application.
s_axis_ccix_tx_tuser	Input	100	This set of signals contains sideband information on TLP being transferred. These signals are valid when s_axis_ccix_tx_tvalid is High. Below The individual signals are defined in the following table.
s_axis_ccix_tx_tvalid	Input	1	The CCIX application asserts this signal whenever it presents valid data on s_axis_ccix_tx_tdata.

**Table 86: 512-Bit Transmit CCIX Interface Port Descriptions (cont'd)**

Port	Direction	Width	Description
ccix_tx_credit_gnt	Output	1	This signal is asserted by core to release credits. The core has a total of eight TX credits. After VC1 link becomes active, the core first releases all the eight credits to the CCIX application, and then the core releases each credit as available. The CCIX application can transmit valid data when the application has credits available.  Note: If CCIX application drops valid between transmitting a TLP packet, the application must guarantee the continuation of TLP packet after ccix_tx_credit_gnt goes High at the core interface. Or The CCIX application waits to accumulate the needed amount of credits before transmitting TLP packet in order to avoid valid drop during the packet.
ccix_tx_credit_rtn	Input	1	The CCIX application asserts this signal to return the credit to the core when the CCIX link goes to the deactivate state.
ccix_tx_active_req	Input	1	This signal is asserted by the CCIX application to come out of deactive(stop) state and into active state. This signal needs to remain High for as long as CCIX application requires the link to be in the active state.
ccix_tx_active_ack	Output	1	The core asserts this signal in response to the ccix_tx_active_req signal from the CCIX application when the core is ready to accept packets being sent.
ccix_tx_deact_hint	Output	1	The core asserts this signal when the PCIe link goes through a link reset. The CCIX application must use this signal to return the credits and move to link deactivate state.

**Table 87: Sideband Signals in s\_axis\_ccix\_tx\_tuser (512-Bit Interface)**

Bit Index	Name	Width	Description
3:0	is_sop[3:0]	4	Signals the start of a TLP in this beat. The encoding is as follows <ul style="list-style-type: none"> <li>• 0000: No new TLP starting in this beat.</li> <li>• 0001: A single new TLP starts in this beat. The start position is indicated by is_sop0_ptr.</li> <li>• 0011: Two new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> </ul> </li> <li>• 0111: Three new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> <li>◦ is_sop2_ptr provides the start position of the third TLP.</li> </ul> </li> <li>• 1111: Two new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> <li>◦ is_sop2_ptr provides the start position of the third TLP.</li> <li>◦ is_sop3_ptr provides the start position of the forth TLP.</li> </ul> </li> <li>• All other values: Reserved.</li> </ul>

Table 87: Sideband Signals in s\_axis\_ccix\_tx\_tuser (512-Bit Interface) (cont'd)

Bit Index	Name	Width	Description
5:4	is_sop0_ptr[1:0]	2	Indicates the position of the first byte of the first TLP starting in this beat: <ul style="list-style-type: none"> <li>• 00: Byte 0</li> <li>• 01: Byte 16</li> <li>• 10: Byte 32</li> <li>• 11: Byte 48</li> </ul>
7:6	is_sop1_ptr[1:0]	2	Indicates the position of the first byte of the second TLP starting in this beat: <ul style="list-style-type: none"> <li>• 00: Reserved</li> <li>• 01: Byte 16</li> <li>• 10: Byte 32</li> <li>• 11: Byte 48</li> </ul>
9:8	is_sop2_ptr[1:0]	2	Indicates the position of the first byte of the second TLP starting in this beat: <ul style="list-style-type: none"> <li>• 00: Reserved</li> <li>• 01: Reserved</li> <li>• 10: Byte 32</li> <li>• 11: Byte 48</li> </ul>
11:10	is_sop3_ptr[1:0]	2	Indicates the position of the first byte of the second TLP starting in this beat: <ul style="list-style-type: none"> <li>• 00: Reserved</li> <li>• 01: Reserved</li> <li>• 10: Reserved</li> <li>• 11: Byte 48</li> </ul>

Table 87: Sideband Signals in s\_axis\_ccix\_tx\_tuser (512-Bit Interface) (cont'd)

Bit Index	Name	Width	Description
15:12	is_eop[3:0]	4	<p>Signals that a TLP is ending in this beat. These inputs are set in the final beat of a TLP. The encoding are as follows:</p> <ul style="list-style-type: none"> <li>• 0000: No TLPs ending in this beat.</li> <li>• 0001: A single TLP is ending in this beat. The is_eop0_ptr[3:0] signal provides the offset of the last Dword of this TLP.</li> <li>• 0011: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP.</li> </ul> </li> <li>• 0111: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP.</li> <li>◦ is_eop2_ptr[3:0] provides the offset of the last Dword of the third TLP</li> </ul> </li> <li>• 1111: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP.</li> <li>◦ is_eop2_ptr[3:0] provides the offset of the last Dword of the third TLP.</li> <li>◦ is_eop3_ptr[3:0] provides the offset of the last Dword of the fourth TLP.</li> </ul> </li> <li>• All other values: Reserved.</li> </ul>
19:16	discontinue[3:0]	4	<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. 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.</p> <p>The discontinue signal can be asserted only when s_axis_ccix_tx_tvalid is High. Thus, once asserted, it should not be deasserted until the end of the packet.</p>
23:20	is_eop0_ptr[3:0]	4	Indicates the offset of the last Dword of the first TLP ending in this beat. This input is valid when is_eop[0] is asserted.
27:24	is_eop1_ptr[3:0]	4	Indicates the offset of the last Dword of the second TLP ending in this beat. This input is valid when is_eop[1] is asserted.
31:28	is_eop2_ptr[3:0]	4	Indicates the offset of the last Dword of the third TLP ending in this beat. This input is valid when is_eop[2] is asserted.
35:32	is_eop3_ptr[3:0]	4	Indicates the offset of the last Dword of the fourth TLP ending in this beat. This input is valid when is_eop[3] is asserted.
99:36	data_parity[63:0]	64	<p>Odd parity for the 512-bit data. When parity checking is enabled in the core, the user logic must set bit i of this bus to the odd parity computed for byte i of s_axis_ccix_tx_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>

## Receive CCIX Core Interface (512-Bit)

The RX CCIX interface contains the ports through which the core receives CCIX data from the user application. The following table defines the ports in the 512-bit receive CCIX interface.

**Table 88: 512-Bit Receive CCIX Interface Port Descriptions**

Port	Direction	Width	Description
m_axis_ccix_rx_tdata	Output	512	Transmits data from core to CCIX application.
m_axis_ccix_rx_tuser	Output	100	This set of signals contains sideband information on TLP being transferred. These signals are valid when m_axis_ccix_rx_tvalid is High. The individual signals are defined in the following table.
m_axis_ccix_rx_tvalid	Output	1	The core asserts this signal whenever it presents valid data on m_axis_ccix_rx_tdata.
ccix_rx_credit_gnt	Input	1	This signal is asserted by the user application to release credits to the core. After the VC1 link becomes active, the user application first release all available credits to the core, and then it releases credits as available. The core transmits valid data when there are credits available. Note: The core can drop valid between packets if there are no available credits.
ccix_rx_credit_rtn	Output	1	The core asserts this signal to return the credits to CCIX application when the CCIX link goes to the deactivate state.
ccix_rx_active_req	Output	1	This signal is asserted by the core to come out of deactivate(stop) state and into active state. This signal needs to remain High as long as the core requires the link to be in active state.
ccix_rx_active_ack	Input	1	The CCIX application asserts this signal in response to the ccix_rx_active_req from the core when the application is ready to accept TLP packets.
ccix_rx_deact_hint	Input	1	The CCIX application asserts this signal when the application goes through a link reset. The core must use this signal to return the credits and move to link deactivate state.

**Table 89: Sideband Signals in s\_axis\_ccix\_rx\_tuser (512-Bit Interface)**

Bit Index	Name	Width	Description
3:0	is_sop[3:0]	4	<p>Signals the start of a TLP in this beat. The encoding is as follows:</p> <ul style="list-style-type: none"> <li>• 0000: No new TLP starting in this beat.</li> <li>• 0001: A single new TLP starts in this beat. The start position is indicated by is_sop0_ptr.</li> <li>• 0011: Two new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> </ul> </li> <li>• 0111: Three new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> <li>◦ is_sop2_ptr provides the start position of the third TLP.</li> </ul> </li> <li>• 1111: Two new TLPs are starting in this beat. <ul style="list-style-type: none"> <li>◦ is_sop0_ptr provides the start position of the first TLP.</li> <li>◦ is_sop1_ptr provides the start position of the second TLP.</li> <li>◦ is_sop2_ptr provides the start position of the third TLP.</li> <li>◦ is_sop3_ptr provides the start position of the fourth TLP.</li> </ul> </li> <li>• All other values: Reserved.</li> </ul>
5:4	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"> <li>• 00: Byte 0</li> <li>• 01: Byte 16</li> <li>• 10: Byte 32</li> <li>• 11: Byte 48</li> </ul>
7:6	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"> <li>• 00: Reserved</li> <li>• 01: Byte 16</li> <li>• 10: Byte 32</li> <li>• 11: Byte 48</li> </ul>
9:8	is_sop2_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"> <li>• 00: Reserved</li> <li>• 01: Reserved</li> <li>• 10: Byte 32</li> <li>• 11: Byte 48</li> </ul>



Table 89: Sideband Signals in s\_axis\_ccix\_rx\_tuser (512-Bit Interface) (cont'd)

Bit Index	Name	Width	Description
11:10	is_sop3_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"> <li>• 00: Reserved</li> <li>• 01: Reserved</li> <li>• 10: Reserved</li> <li>• 11: Byte 48</li> </ul>
15:12	is_eop[3:0]	4	<p>Signals that a TLP is ending in this beat. These outputs are set in the final beat of a TLP. The encoding are as follows:</p> <ul style="list-style-type: none"> <li>• 0000: No TLPs ending in this beat.</li> <li>• 0001: A single TLP is ending in this beat. is_eop0_ptr[3:0] provides the offset of the last Dword of this TLP.</li> <li>• 0011: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP.</li> </ul> </li> <li>• 0111: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP.</li> <li>◦ is_eop2_ptr[3:0] provides the offset of the last Dword of the third TLP.</li> </ul> </li> <li>• 1111: Two TLPs are ending in this beat. <ul style="list-style-type: none"> <li>◦ is_eop0_ptr[3:0] provides the offset of the last Dword of the first TLP.</li> <li>◦ is_eop1_ptr[3:0] provides the offset of the last Dword of the second TLP.</li> <li>◦ is_eop2_ptr[3:0] provides the offset of the last Dword of the third TLP.</li> <li>◦ is_eop3_ptr[3:0] provides the offset of the last Dword of the fourth TLP.</li> </ul> </li> <li>• All other values: Reserved.</li> </ul>
19:16	discontinue[3:0]	4	<p>This signal can be asserted by the core during a transfer if it has detected an error in the data being transferred and needs to abort the packet. The user logic nullifies the corresponding TLP on the link to avoid data corruption.</p> <p>The core asserts this signal in any beat of a 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.</p> <p>The discontinue signal can be asserted only when s_axis_ccix_rx_tvalid is High.</p>
23:20	is_eop0_ptr[3:0]	4	<p>Indicates the offset of the last Dword of the first TLP ending in this beat. This output is valid when is_eop[0] is asserted.</p>
27:24	is_eop1_ptr[3:0]	4	<p>Indicates the offset of the last Dword of the second TLP ending in this beat. This output is valid when is_eop[1] is asserted.</p>

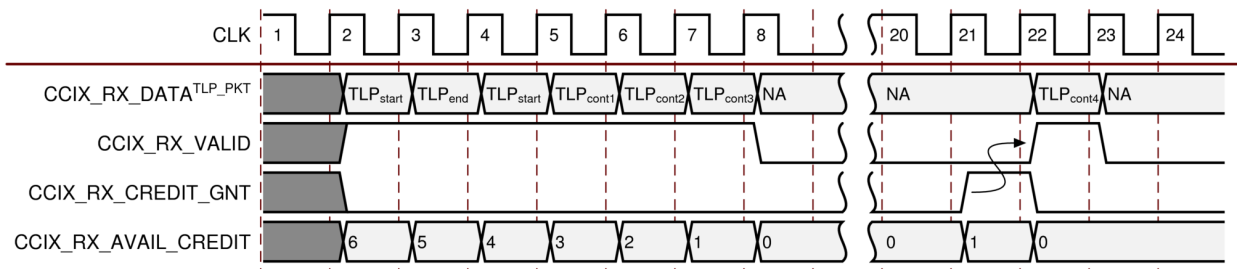
Table 89: Sideband Signals in s\_axis\_ccix\_rx\_tuser (512-Bit Interface) (cont'd)

Bit Index	Name	Width	Description
31:28	is_eop2_ptr[3:0]	4	Indicates the offset of the last Dword of the third TLP ending in this beat. This output is valid when is_eop[2] is asserted.
35:32	is_eop3_ptr[3:0]	4	Indicates the offset of the last Dword of the fourth TLP ending in this beat. This output is valid when is_eop[3] is asserted.
99:36	data_parity[63:0]	64	Odd parity for the 512-bit data. The core sets bit i of this bus to the odd parity computed for byte i of s_axis_ccix_rx_tdata. On detection of a parity error, the user application can nullifies the corresponding TLP on the link and treat it as an Uncorrectable Internal Error.

## Example Figures

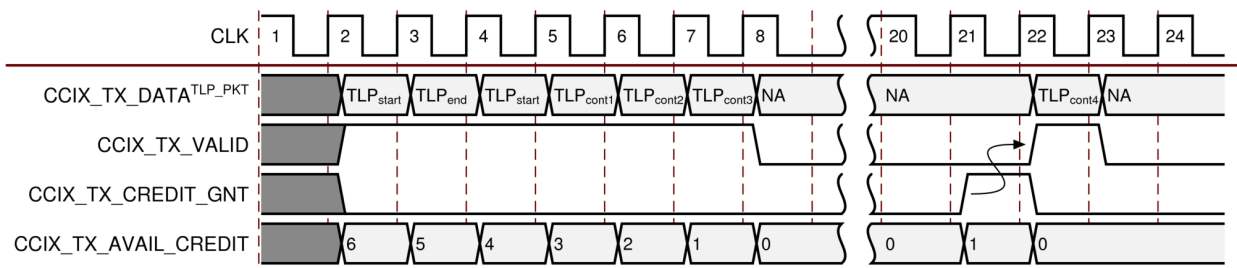
The following figure shows how the CCIX RX core interface transfers data based on available credits from the user application. The interface will deassert `ccix_rx_valid` for a TLP that is in the process of being presented, because it has no credit available. The core takes one user clock cycle to restart the TLP presentation after the user application sends a credit.

Figure 58: CCIX RX Core Interface



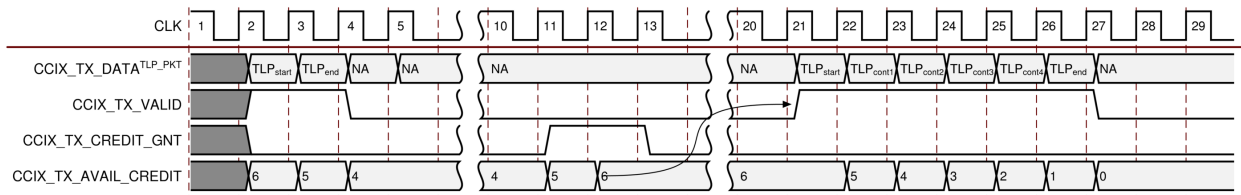
The following figure shows the requirements of the user application on the Transmit CCIX core interface, when the `ccix_tx_valid` signal is deasserted during presentation of a TLP packet (option 1).

Figure 59: CCIX PCIe TX interface - Option 1



The following figure shows options 2, where the user accumulates enough credits before transmitting the TLP packet, which ensures that the user does not need to deassert `ccix_tx_valid` during the presentation of the TLP.

Figure 60: CCIX PCIe TX interface - Option 2



## CCIX-VC1 Credits

The following table shows the credits information for CCIX core interface 256 bits and 512 bits.

Table 90: Credits

Credits	256 Bits	512 Bits
Transmit CCIX Core Interface Credits	8	12
PCI Express VC1 Receiver Flow Control Posted Credits	Posted Header = 64D Posted Data = 1C0H (7168B)	Posted Header = 64D Posted Data = C0H (3072B) (Default) or 1C0H (7168B)

## CCIX Activation/Deactivation

Four states define how a CCIX link moves from deactivation to activation, and activation to deactivation.

**Note:** When moving activation to deactivation, it is important that the credit exchange is carefully controlled to avoid loss of data or credits in the CCIX link. The CCIX link must be in quiesce state.

The following table describes the four state transitions.

Table 91: Four State Transitions

State Name	Description
Stop	The CCIX link is in reset state or low power state. All the credits are with receiver and transmitter, and are not permitted to send any valid data.
Run	The CCIX link is in operation state. Both components can active transfer valid data and credits.
Activate	Intermediate state when moving from Stop to Run.

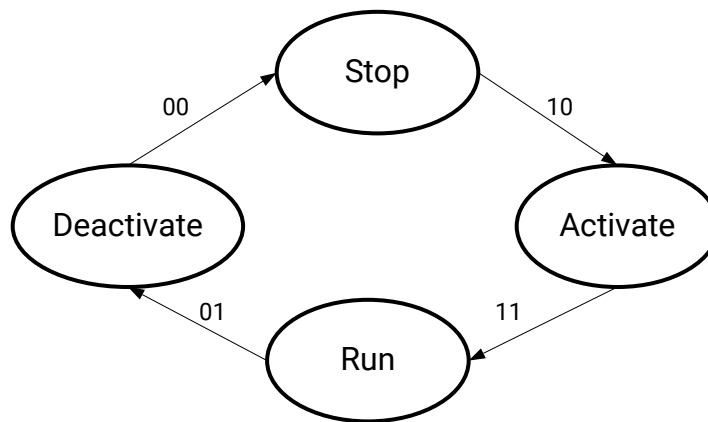
Table 91: Four State Transitions (cont'd)

State Name	Description
Deactivate	Intermediate state when moving from Run to Stop.

The following figure shows the four state transitions that are based on a pair of interface signals used as a concatenation:

- `ccix_tx_active_req` and `ccix_tx_active_ack`, or
- `ccix_rx_active_req` and `ccix_rx_active_ack`.

Figure 61: CCIX Link State Transitions



X22414-030119

# 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

This section provides links to supplemental material useful to this document:

1. *Vivado Design Suite: AXI Reference Guide* ([UG1037](#))
2. *7 Series FPGAs GTX/GTH Transceivers User Guide* ([UG476](#))
3. *UltraScale Architecture GTH Transceivers User Guide* ([UG576](#))
4. *Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG023](#))
5. *UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG156](#))
6. *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#))
7. *AXI Memory Mapped to PCI Express (PCIe) Gen2 LogiCORE IP Product Guide* ([PG055](#))
8. *DMA/Bridge Subsystem for PCI Express Product Guide* ([PG195](#))
9. *AXI Verification IP LogiCORE IP Product Guide* ([PG267](#))
10. *7 Series FPGAs Clocking Resources User Guide* ([UG472](#))
11. *UltraScale Architecture Clocking Resources User Guide* ([UG572](#))
12. *AMBA AXI4-Stream Protocol Specification* ([ARM IHI 0051A](#))
13. *PCI-SIG Specifications* (<https://www.pcisig.com/specifications>)
14. *ISE to Vivado Design Suite Migration Guide* ([UG911](#))
15. *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
16. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
17. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
18. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
19. *Vivado Design Suite User Guide: Dynamic Function eXchange* ([UG909](#))
20. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))

## Revision History

The following table shows the revision history for this document.

Section	Revision Summary
<b>11/16/2022 Version 3.0</b>	
<a href="#">Bridge Parameters</a>	Updated Top-Level Parameters table.
<a href="#">Interrupt Decode Register (Offset 0x138)</a>	Updated Interrupt Decode Register table.
<a href="#">Interrupt Mask Register (Offset 0x13C)</a>	Updated Interrupt Mask Register table.
<a href="#">Completion Timeout</a>	Updated Completion Timeout section.
<a href="#">Completer Abort</a>	Updated Completer Abort section.

Section	Revision Summary
<b>07/22/2020 Version 3.0</b>	
AXI Bridge for PCIe Gen3 and DMA/Bridge Subsystem for PCIe in Bridge Mode	Updated descriptions for Legacy Interrupts, MSI Interrupts, and MSI-X Interrupts
Root Port MSI Interrupt Decode 2 Mask Register (Offset 0x17C) and Root Port MSI Interrupt Decode 1 Mask Register (Offset 0x178)	Updated interrupt_out_msi_vec* signal names in descriptions.
Tandem Configuration	Updated Partial Reconfiguration to Dynamic Function eXchange.
<b>07/16/2019 Version 3.0</b>	
Port Descriptions	Reorganized port description information into individual tables per interface.
AXI Slave Interface and AXI Master Interface	Corrected port name and updated description for s_axi(b)_wuser and s_axi(b)_ruser.
Transaction Ordering for PCIe	Updated details regarding relaxed ordering.
Completion Timeout	Updated details regarding PCIe configuration read/write requests.
Basic Tab (DMA/AXI Bridge Subsystem for PCI Express in AXI Bridge Mode)	Added MPSOC PL RootPort solution.
Appendix D: CCIX Interface	Added appendix.
<b>12/05/2018 Version 3.0</b>	
Chapter 3: Product Specification	Added Gen4 support to the Minimum Device Requirements table. Added clarifying note to axi_ctl_aclk signal description. Added Configuration Register Attribute Definitions table to Memory Map section.
Chapter 4: Designing with the Core	Added Gen4 support to the Clock Frequencies and Interface Widths table. Added Root Port System Reset Connection diagrams to the Reset section. Added clarifying note to Receiving Interrupts section.
<b>04/04/2018 Version 3.0</b>	
General Updates	Clarified that Tandem Configuration is not yet supported for Bridge mode in UltraScale+™ devices.
Chapter 3: Product Specification	Added more detail to interrupt_out port description (in the Top-Level Interface Signals table in the Product Specification chapter). Updated the axi_ctl_aresetn port default values Please refer to the description for more details and the available options. Updated signal name and description for s_axi_ctl_awaddr/s_axil_awaddr, and s_axi_ctl_araddr/s_axil_araddr. Added the msi_enable, msix_enable, and dma_bridge_resetn signals. Major rewrite in the Memory Map section. Added C_MSI_RX_PIN_EN, INTERRUPT_OUT_WIDTH, and SOFT_RESET_EN bridge parameters. Updated Reserved addresses in the IRQ Block Register Space in the "DMA/Bridge Subsystem for PCIePCIe Register Memory Map" section. Added Virtex® UltraScale+™ Devices with HBM (PCIe4C) minimum device requirements information.

Section	Revision Summary
<a href="#">Chapter 4: Designing with the Core</a>	Changed C_PCIE2AXIBAR_n to C_PCIEBAR2AXIBAR_n (in the BAR and Address Translation section). Added Interrupts section to Endpoint section, and Receiving Interrupts section to the Root Port section.
<b>12/20/2017 Version 3.0</b>	
<a href="#">Chapter 4: Designing with the Core</a>	Added Clock Frequencies and Interface Widths Supported For Various Configurations figure to the . Updated the Clocking Diagram (UltraScale+ Devices).
<a href="#">Chapter 6: Example Design</a>	Added example design block diagrams.
<a href="#">Chapter 7: Test Bench</a>	Added Example Use Mode section.
<b>10/04/2017 Version 3.0</b>	
General Updates	Added the DMA/Bridge Subsystem for PCI Express in AXI Bridge mode information (UltraScale+UltraScale+ devices only) to this document (formerly in PG195). Added two new register descriptions: Root Port Interrupt Decode 2 Register (Offset 0x160), and Root Port Interrupt Decode 2 Mask Register (Offset 0x164). Added example design details: Xilinx AXI Verification IP attached to the AXI Slave Interface.
<a href="#">Appendix A: Upgrading</a>	Updated details.
<a href="#">Appendix C: Using the Xilinx Virtual Cable to Debug</a>	New appendix added.
<b>04/05/2017 Version 3.0</b>	
<a href="#">Customizing the Core</a>	Added the new Debug Options parameters.
<b>11/30/2016 Version 3.0</b>	
General Updates	Updated to reflect that the core supports Narrow Bust in 2016.4. Added the Minimum Device Requirements section.
<b>10/12/2016 Version 3.0</b>	
General Updates	Updated that the slave bridge is capable of handling up to 8 memory mapped AXI4 read requests with pending completions.
<b>10/05/2016 Version 3.0</b>	
General Updates	Updated Bit[2] and Bit[28] to reserved bits in the Interrupt Decode Register and Interrupt Mask Register tables. Added Bit[13] row to PHY Status/Control Register table. Minor updates to the Link Down Behavior section.
Migrating and Upgrading	Added the New Ports table to the Port Changes section (Upgrading in the Vivado Design Suite)
<b>06/08/2016 Version 2.1</b>	
IP Facts	Updated Maximum Payload Size to 512 in IP Facts.
Top-Level Interface Signals	Added axi_aresetn, s_axi_ctl_awaddr[31:0], and s_axi_ctl_araddr[31:0] to the table..
Top-Level Parameters	Removed C_INCLUDE_BAROFFSET_REG, added pf0_msix_cap_pba_bir to pf0_msix_cap_table_size to the table.
Updated Note	Address Translation
General Updates	Updated PCIe Miscellaneous Setting figure and GT Settings figure.



Section	Revision Summary
PCIe Miscellaneous	Added MSIx Table Settings and MSIx Pending Bit Array Settings section.
<b>04/06/2016 Version 2.1</b>	
General Updates	Small editorial update to the Root Port Error FIFO Read Register (Offset 0x154) section. Added the Tandem Configuration section. New Parameter Changes and Port Changes tables for the release added to the Migrating and Updating appendix.
<b>11/18/2015 Version 2.0</b>	
General Updates	Updated the supported speed grades. Updated the Limitations section in the Overview chapter. Updated the description for m_axi_arready. Added VCU108 to the Reference Boards section in the Debugging chapter.
<b>09/30/2015 Version 2.0</b>	
General Updates	Updated for the core version 2.0. Added support for Root Port configurations. Added MSI-X signal support. Updated the clock and reset diagrams. Added the Configuration Control Register. Clarified the Reference Clock input frequency as refclk for Virtex-7 devices, and sys_clk_gt for UltraScale devices. Updated supported values and the defaults for the following parameters: <ul style="list-style-type: none"> <li>• C_S_AXI_NUM_WRITE</li> <li>• C_S_AXI_NUM_READ</li> <li>• C_M_AXI_NUM_WRITE</li> <li>• C_M_AXI_NUM_READ</li> </ul> Added the PL_UPSTREAM_FACING bridge parameter. Added the RX_Detect parameter. Added the cfg_ltssm_state port. Updated the Register Memory Map and Bridge Info Register tables. Added the Enhanced Configuration Access table (ECAM).
<b>06/24/2015 Version 1.1</b>	
General Updates	Moved performance and resource utilization data to Xilinx website. Corrected C_NUM_MSI_REQ from attribute to signal. Corrected the documented parameters: added top-level parameters, removed erroneous entries, and corrected parameter names, and descriptions. Added the BAR Addressing section. Corrected Example 2 (64-bit PCIe Address Mapping). Added ports enabled with the Enable MSIX Capability Structure options selected to the Migrating and Upgrading chapter Updated Vivado® Lab Edition to Vivado® Design Suite Debug Feature.

Section	Revision Summary
<b>05/07/2015 Version 1.1</b>	
General Updates	Minor editorial update: Corrected page footers.
<b>04/01/2015 Version 1.1</b>	
General Updates	Specified that the narrow burst feature is not supported. To Vivado IDE description, added new GT Settings tab, and PLL Selection, CORE CLOCK Frequency, PPM Offset between receiver and transmitter, Spread Spectrum clocking, Insertion loss at Nyquist, and Link Partner TX Preset parameters. Added post-synthesis and post-implementation netlist simulation details. Added Transceiver Debug information. Updated Vivado lab tools to Vivado Lab Edition.
<b>11/19/2014 Version 1.0</b>	
General Updates	Added UltraScale architecture placement constraint examples. Updated the simulation procedures for Cadence Incisive Enterprise Simulator (IES), and Verilog Compiler Simulator (VCS). Added important note regarding the recommended version of Mentor Graphics simulator to use to avoid simulation failure. Minor edits.
<b>10/01/2014 Version 1.0</b>	
Initial 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://](https://www.xilinx.com/legal)

[www.xilinx.com/legal.htm#tos](http://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 2014–2022 Advanced Micro Devices, 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. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.