PCIe Debug K-Map documentation

Release 1.0

Xilinx, Inc.

Jul 28, 2021

Chapter 1

PCIe Collaterals

1.1 PCIe LFARs (Long Form Answer Records)

S.No.	Title
1	Virtex-5 Integrated PCI Express Block Plus - Debugging Guide for Link Training Issues
2	Virtex-5 Endpoint Block Plus for PCI Express - Debugging and Packet Analysis Guide with Downstream Port Model and PIO Example Design
3	Virtex-6 Integrated PCIe Block Wrapper - Debugging and Packet Analysis Guide
4	7 Series Integrated Block for PCI Express in Vivado
5	Generating Quick Test Cases for Xilinx Integrated PCI Express Block and Serial RAPIDIO Cores Verilog Simulation
6	7 Series Integrated Block for PCI Express - Link Training Debug Guide
7	Virtex-7 FPGA Gen3 Integrated Block for PCI Express core SRIOV Example Design Simulation
8	Xilinx PCI Express Interrupt Debugging Guide
9	Vivado ILA Usage Guide for 7 Series Integrated Block for PCI Express
10	AXI Memory Mapped for PCI Express Address Mapping
11	UltraScale and UltraScale+ FPGA Gen3 Integrated Block for PCI Express - Integrated Debugging Features and Usage Guide
12	Vivado ILA Usage Guide for UltraScale FPGA Gen3 Integrated Block for PCI Express
13	Reading AXI PCIe Gen3/XDMA Internal Registers using JTAG to AXI Master IP
14	Xilinx PCI Express (PS-PCIe/PL-PCIe) Drivers Debug Guide
15	DMA Subsystem for PCI Express - Driver and IP Debug Guide
16	PetaLinux Image Generation and System Example Design with ZC706 as Root Complex and KC705 as Endpoint
17	PetaLinux Image Generation and System Example Design with ZCU102 PS-PCIe as Root Complex and ZC706 as Endpoint
18	Example design with PL-PCIe Root Port in ZCU106 and PS-PCIe Endpoint in UltraZed
19	UltraScale+ FPGA Gen3 Integrated Block for PCI Express (Vivado 2019.1) - Integrated Debugging Features and Usage Guide
20	Queue DMA subsystem for PCI Express (Vivado 2019.1) - QDMA Linux Kernel Driver Usage and Debug Guide
21	Xilinx PCI Express Gen3 Link Training Debugging Guide for UltraScale and UltraScale+ Devices
22	System Example Design with ZCU102 PS-PCIe as Root Complex and Intel SSD 750 Series NVMe Device as an Endpoint
23	UltraScale/UltraScale+ PCI Express Integrated Block - Interrupt debug guide

1.2 PCIe Debug Tips and Techniques Blogs

S.No.	Title
1	Demystifying PIPE interface packets using the in-built descrambler module in UltraScale+ Devices Integrated Block for PCI Express Gen3
2	QDMA Linux Kernel Driver Usage and Debug Guide
3	Debugging PCIe Issues using lspci and setpci
4	Debugging PCI Express Link Training Issues with Integrated Debugging Features in the IP
5	Debugging PCIe Issues Using Python
6	Debugging Versal ACAP Integrated Block for PCIe Express link issues using in-built “”PCIe Link Debug”” feature
7	Understanding the new PL PCIE IP Generation flow for Versal ACAP Devices
8	Debugging Versal ACAP CPM Mode for PCI Express Designs using Vivado ILA
9	Register based debugging of Versal ACAP CPM Mode for PCI Express Designs

1.3 PCIe Release Notes

AR Number	Title	Remarks
Xilinx Answer 75397	Versal ACAP DMA and Bridge Subsystem for PCI Express	
Xilinx Answer 75396	Versal ACAP CPM DMA and Bridge Mode for PCI Express	
Xilinx Answer 75350	Versal ACAP CPM Mode for PCI Express	
Xilinx Answer 73083	Versal ACAP Integrated Block for PCI Express	
Xilinx Answer 72289	Versal ACAP PHY for PCI Express	
Xilinx Answer 70927	Queue DMA subsystem for PCI Express (PCIe)	
Xilinx Answer 66988	UltraScale Architecture PHY for PCI Express	
Xilinx Answer 65751	UltraScale+ PCI Express Integrated Block	
Xilinx Answer 65443	DMA Subsystem for PCI Express	
Xilinx Answer 57945	UltraScale FPGA Gen3 Integrated Block for PCI Express	
Xilinx Answer 61898	AXI Bridge for PCI Express Gen3	
Xilinx Answer 54645	Virtex-7 FPGA Gen3 Integrated Block for PCI Express (v2.0 and onwards)	For v2.0 and onwards in Vivado Design Suite.
Xilinx Answer 47441	Virtex-7 FPGA Gen3 Integrated Block for PCI Express (before v2.0)	For all versions in ISE Design Suite and versions prior to v2.0 in Vivado Design Suite.
Xilinx Answer 54643	7 Series Integrated Block for PCI Express (v2.0 and onwards)	For v2.0 and onwards in Vivado Design Suite.

Xilinx Answer 40469	7 Series Integrated Block for PCI Express (before v2.0)	For all versions in ISE Design Suite and versions prior to v2.0 in Vivado Design Suite.
Xilinx Answer 54646	AXI PCI Express (v2.0 and onwards)	For v2.0 and onwards in Vivado Design Suite.
Xilinx Answer 44969	AXI PCI Express (before v2.0)	For all versions in ISE Design Suite (EDK) and versions prior to v2.0 in Vivado Design Suite.
Xilinx Answer 65178	Virtex-6 FPGA Integrated Block for PCI Express	Master Answer Record
Xilinx Answer 65177	Spartan-6 FPGA Integrated Endpoint Block for PCI Express	Master Answer Record
Xilnx Answer 51597	Endpoint Block Plus Wrapper for PCI Express	Master Answer Record

1.4 PCIe Application Notes

PCIe Application Notes
XAPP1177: Designing with SR-IOV Capability of Xilinx Virtex-7 PCI Express Gen3 Integrated Block
http://www.xilinx.com/support/documentation/application_notes/xapp1177-pcie-gen3-sriov.pdf
XAPP1179: Using Tandem Configuration for PCIe in the Kintex-7 Connectivity TRD
http://www.xilinx.com/support/documentation/application_notes/xapp1179-tandem-config-pcie.pdf
XAPP1184: PIPE Mode Simulation Using Integrated Endpoint PCI Express Block in Gen2 x8 and Gen3 x8 Configurations
http://www.xilinx.com/support/documentation/application_notes/xapp1184-PIPE-mode-PCIe.pdf
XAPP1171: PCI Express Endpoint-DMA Initiator Subsystem
http://www.xilinx.com/support/documentation/application_notes/xapp1171-pcie-central-dma-subsystem.pdf
XAPP1201: Virtex-7 (XT and HT) and UltraScale FPGAs Gen3 Integrated Block for PCI Express to AXI4-Lite Bridge
http://www.xilinx.com/support/documentation/application_notes/xapp1201.pdf
XAPP1198: In-System Eye Scan of a PCI Express Link with Vivado IP Integrator and AXI4
http://www.xilinx.com/support/documentation/application_notes/xapp1198-eye-scan.pdf
XAPP859: Virtex-5 FPGA Integrated Endpoint Block for PCI Express Designs: DDR2 SDRAM DMA Initiator Demonstration Platform
http://www.xilinx.com/support/documentation/application_notes/xapp859.pdf
XAPP1002: Using ChipScope Pro to Debug Endpoint Block Plus Wrapper, Endpoint, and Endpoint PIPE
http://www.xilinx.com/support/documentation/application_notes/xapp1002.pdf
XAPP 1022: Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores
http://www.xilinx.com/support/documentation/application_notes/xapp1022.pdf
XAPP 1052: Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions
http://www.xilinx.com/support/documentation/application_notes/xapp1052.pdf
XAPP518: In-System Programming of BPI PROM for Virtex-6 FPGAs Using PCI Express Technology
http://www.xilinx.com/support/documentation/application_notes/xapp518-isp-bpi-prom-virtex-6-pcie.pdf
XAPP883: Fast Configuration of PCI Express Technology through Partial Reconfiguration
http://www.xilinx.com/support/documentation/application_notes/xapp883_Fast_Config_PCIe.pdf

XAPP1201: Virtex-7 (XT and HT) and UltraScale FPGAs Gen3 Integrated Block for PCI Express to AXI4-Lite Bridge
https://www.xilinx.com/support/documentation/application_notes/xapp1201.pdf
XAPP1198: In-System Eye Scan of a PCI Express Link with Vivado IP Integrator and AXI4
https://www.xilinx.com/support/documentation/application_notes/xapp1198-eye-scan.pdf
XAPP1286: 7 Series FPGAs Gen2 Integrated Block for PCIe to AXI4-Lite Bridge
https://www.xilinx.com/support/documentation/application_notes/xapp1286-pcie-axi4-lite-bridge.pdf

1.5 PCIe Videos

PCIe Videos
Xilinx UltraScale+ PCIe Gen3 x16 hardened IP passes PCI SIG compliance test: See it now on video running at 100Gbps+
https://forums.xilinx.com/t5/Xcell-Daily-Blog/Xilinx-UltraScale-PCIe-Gen3-x16-hardened-IP-passes-PCI-SIG/ba-p/
AXI PCIe with MIG on a KCU105 using WinDriver from Jungo Connectivity
http://www.xilinx.com/video/fpga/ultrascale-designs-fast-ipi-ddr4-pcie-windriver.html
How to create a PCI Express Design in an UltraScale FPGA:
https://www.youtube.com/watch?v=1YgviyNfLYY
UltraScale PCI Express - The Power of 4:
https://www.youtube.com/watch?v=G8n86wvh2ig
Ultrascale PIPE Simulation with Mentor BFM
https://www.youtube.com/watch?v=VWnkg01rJEY
AXI PCI Express MIG Subsystem Built in IPI:
https://www.youtube.com/watch?v=0KnvW_6Bgu0
Zynq PCI Express Root Complex Made Simple:
https://www.youtube.com/watch?v=D1vOFBSuWAc
Virtex-7 PCI Express Gen3 Demo:
https://www.youtube.com/watch?v=IOHgltR11QY
Xilinx Virtex-6 FPGA PCI Express Demo:
http://www.youtube.com/watch?v=wxD71xdmmkE
PCIe x8 Gen3 Running on a Xilinx Kintex-7 FPGA:
https://www.youtube.com/watch?v=mAw7Ao6P6zU
Inserting Debug Cores into the Design:
https://www.youtube.com/watch?v=bU8BsPuIyOo&index=29&list=PL35626FEF3D5CB8F2
Programming and Debugging a Design in Hardware:
https://www.youtube.com/watch?v=i8axs4hw2f4&list=PL35626FEF3D5CB8F2&index=30
Debugging at Device Startup:
https://www.youtube.com/watch?v=dt3YTlWfeHw&list=PL35626FEF3D5CB8F2&index=104
Tandem Configuration of 7 Series Devices:
https://www.youtube.com/watch?v=N5OVPtSTwuA
Xilinx QDMA Linux Kernel Drive Usage Demo
https://youtu.be/c2J89liXHYA
Xilinx QDMA DPDK Driver Usage Demo
https://youtu.be/RYozp-DmwSk
Generating and Implementing Xilinx PCIe Example Design for VCU118 Development Board in Vivado 2019.2
https://youtu.be/HJUARBawyqw

Generating Xilinx DMA Subsystem for PCI Express (XDMA) Example Design for VCU118 in Vivado 2019.2
https://youtu.be/x0NjX-Zzg4k
Generating QDMA Subsystem for PCI Express v4.0 Example Design for U200 Board in Vivado 2020.1
https://youtu.be/eSJc6TWGAFI
Versal ACAP: PCI Express
https://www.xilinx.com/video/acap/versal-acap-live-pci-express.html
Versal Premium series: PCIe Gen5
https://www.xilinx.com/video/acap/pcie-gen5.html

1.6 PCIe White Papers

PCIe White Papers
WP464: PCI Express for UltraScale Architecture-Based Devices
http://www.xilinx.com/support/documentation/white_papers/wp464-PCIe-ultrascale.pdf
WP350: Understanding Performance of PCI Express Systems
http://www.xilinx.com/support/documentation/white_papers/wp350.pdf
WP384: PCI Express for the 7 Series FPGAs
http://www.xilinx.com/support/documentation/white_papers/wp384_PCIe_7Series.pdf
WP363: Spartan-6 FPGA Connectivity Targeted Reference Design Performance
http://www.xilinx.com/support/documentation/white_papers/wp363.pdf

Chapter 2

PCIe Common Issues

2.1 Enumeration shows no PCIe device (lspci)

- Check using ILA if the cfg_ltssm_state signal shows an L0 state ('h10).
- If in the L0 state, check if it consistently stays in the L0 state or is going through recovery state continuously.
- **Repeatedly going through recovery state indicates a link integrity issue.**
    - If it is stable in L0 state, check if PCIe Config Request TLP's are exchanged and that each Completion TLP is returned.
    - This is part of the PCI enumeration process and must be done within 100ms of the Power Good indication.
    - Try doing a warm reboot and if the device is now detected after a warm reboot, this requirement was violated.
    - Certain server systems might not do another PCIe discovery after a PCIe slot/device failure, requiring a Cold Boot (power cycle) to recover.
    - If the cfg_ltssm_state signal shows state 00 indefinitely,ensure that cfg_link_-training_enable input pin is driven to 1'b1.
    - Check using AXI JTAG if the GT reset FSM has completed and is back to 00 state. If so, check if the phy_status_rst pin is connected to the PCIe reset_done pin.

2.2 Missing DMA read data for certain read requests

- **Cross check the read/write request size against the max payload size register in the config space of the endpoint, as there is a chance of the host machine not scanning the config register correctly.**
    - The issue was see when programming the device on board first and then placing the board in the PCIe slot and rescanning by the host PCIe bus.
    - Host bridge did not enumerate the EP config registers properly.

2.3 Missing payload in TLP

Make sure the control signals (tready, tvalid, tlast) are correctly driven from the user application.

2.4 Unsupported Requests/Completion Timeout

Use the command "lspci -vvv -d [Vendor ID]:[Device ID]" and check to ensure that the BAR exists:

The list should have something similar to the following (or more than one if you have multiple BARs):

Region 0: Memory at <address>.

Make sure that this line does not have words such as disabled or virtual.

Note Virtual means that you have lost the BAR information at the Endpoint card, but that it was previously visible. This can be caused by an unexpected link down.

Disabled means that the Memory Enable bit at the PCIe Command register is not set. Set them through your driver or using the setpci utility.

If the BAR does not show up at all, but lspci does indicate that your device is detected, then your system might be running out of memory during memory allocation. This can be caused by large PCIe BARs in your design. Use the pci=realloc directive in the Kernel to re-map your MMIO or use 64-bit BAR instead of 32-bit BAR. Typically this is caused by Missing BAR information or the Command Register (Memory Enable bit) not being set.

2.5 Receiver Overflow

- Check whether the lspci AER RxOF+ is set
- Ensure that the system sets the relaxed ordering bit in the PCIe config space. If not, it might cause an overflow
- Check if sufficient credits have been assigned

Chapter 3

PCIe General Debug Techniques

3.1 First thing to check

- If possible, check whether the issue could be reproduced with the generated example design in Gen1x1 configuration.
- In the case of a user design, check with Gen1x1 configuration if the IP is configured for a higher speed and lane width.

3.2 How to check the LTSSM status?

See:

- https://www.xilinx.com/support/answers/72471.html (For US+ Devices)
- https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-Integrated-B (For Versal Devices)

3.3 How to check the endpoint was successfully detected and enumerated ?

See https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCIe-Issues-using-lspci-and-setpci/ba-p/1148199

3.4 How to enumerate the endpoint when FPGA is configured after enumeration?

See https://www.xilinx.com/support/answers/37406.html

3.5 How to debug link training issues?

See https://www.xilinx.com/support/answers/73361.html

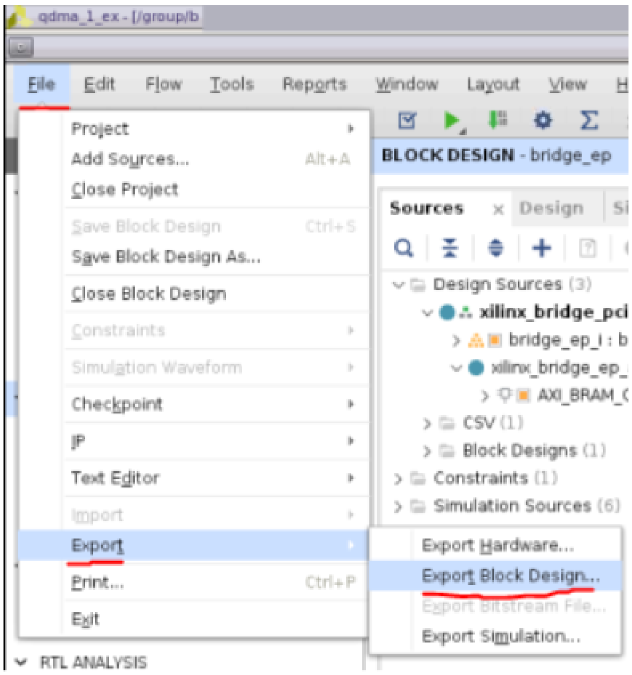
3.6 Generating IP Block Design from the Example Design

The below instructions provide a technique to isolate the Versal PL PCIe IP IPI block design from a full design. In Versal ACAP devices, it introduces the concept of Split IP where the full PL PCIe IP is an integrated IPI block consisting of various components that make up the IP. For more details on the Split IP concept, see the blog in the link below:

https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Understanding-the-new-PL-PCIE-IP-Generation-f

If you have a design and wish to extract only the PL PCIe block design, you can follow the steps below.

1. Create project (e.g. project_1 as project name) and generate QDMA IP from the IP catalog
2. Configure the QDMA IP as an endpoint
3. Right click on the IP and generate the example design
4. In the example design, there will be two block designs generated. Close the block design ending with "_rp"
5. Export block design with the name ending with "_ep" as .tcl file. To do this, click "File -> Export -> Export Block Design" as shown in the snapshot below.



6. The generated .tcl file could be sourced in a new project to create only the QDMA IPI block design.

Chapter 4

Link Training Issue

4.1 Common Link Training Issue Reasons

- Unable to retain L0, going to recovery
- Incorrect Pinouts – Clock, GTs, Reset
- Lane is reversed and neither EP or RP can do lane reversal
- BAR is too big or wrong type – Host run out of contiguous memory space
- Link is disabled by Host – maybe missed enumeration time, driver directed to this, surprise link down, etc.
- VM issue – device not assigned to proper IOMMU for VM use
- Host is bifurcated – Card doesn't match Host and vice versa
- Link width/speed not supported by Host slot
- Power issue on the card
- Wrong Vendor / Device ID
- Invalid capability pointers or extended config not terminated
- Incorrect / bad Tandem image
- Timing failures / Wrong constraints
- Incorrect IP use case – no credits, stalled/throttled data pipelines, cfg_space not enabled, etc.
- Existing Errata with the link partner
- Incorrect BIOS revision

4.2 General Debug Questions

4.2.1 Regression

- Did the issue occur in previous Vivado versions too?
- Do other link width configurations show similar behavior?
- Have you tried with Gen1x1 configuration?
- Do you have a different board that you could try on? If you do, do you see the same issue on that board?

- Have you tried on a different machine?

4.2.2 System Configuration

- Indicate what board you are using: is it a Xilinx Development Board or a Customer Board? If it is a Xilinx development board, please provide the board revision ID. Indicate motherboard description:
- What is the link partner? Is it a switch, or a PC? Who is the manufacturer of that switch or PC? - Which Chipset are you using?
- Did the failure occur in Gen1, Gen2, and/or Gen3?
- Did the failure occur as RP (Root Port) and/or EP (Endpoint)?
- Describe the channel between the FPGA and the link partner, and estimated channel loss at desired link speed.
- Is there any passive hardware (Interposer for analyzer, retimer) in the path?
- How is the Xilinx part connected to the system / link partner (Chip-to-chip on single board, add-in card, backplane, cabling) ?

4.2.3 Regression

- Did the issue occur in previous Vivado versions too?
- Do other link width configurations show similar behavior?
- Have you tried with Gen1x1 configuration?
- Do you have a different board that you could try on? If you do, do you see the same issue on that board?
- Have you tried on a different machine?

4.2.4 Clocking

- Did the clock lock?
- What is the clocking architecture? Synchronous or Asynchronous?
- Is SSC enabled?
- Have you checked by disabling SSC?
- What frequency are you using for the reference clock?

4.2.5 Design Implementation

- Were there any implementation (synthesis, routing) errors?
- Were there any timing errors?
- Have all of the IP constraints been verified by comparing with the Example Design XDC file?

4.2.6 Failing Behavior

- What is the frequency of the error? For example, does it happen immediately or after 1 hour?
- Can the error be cleared? If cleared, does the error come back?
- Is this failure observed on multiple parts?

- Does failure occur immediately after reset?
- Does failure occur immediately, after first rate change, after multiple rate changes?
- How long after successful rate change does it fail?
- Does the issue occur with the Example Design as well or only in your design?

4.2.7 Debug Capability

- Was a protocol link analyzer used? If so, provide protocol link analyzer captures with the details of your analysis of the captures.
- Do you have a high-speed oscilloscope available, to probe clock / power / data lines?
- Do you have a free-running clock available to the FPGA? (A clock that is separate from the PCIe reference clock, and not tied to the PCIe reset)
- Do you have the ability to insert a clean clock in place of the on-board reference clock?
- Have you captured an LTSSM graph by enabling the JTAG Debugger feature in the GUI?
- Have you run Eye Scan by enabling the In-system IBERT feature in the GUI?

4.2.8 SI Debug Info

- Has it been confirmed whether Clock Jitter and Power Noise are within the specification?
- Power integrity measurements
- REFCLK jitter measurements
- Channel loss data
- Eye Scan plots
- Confirm whether DFE, LPM or AutoRxEq are selected in the core configuration.

4.3 General Debug Checklist

- Check with the example design that comes with the generation of the IP.
- Try Gen1x1 configuration
- Instead of regenerating the core, tape off the lanes if possible (See: https://www.xilinx.com/support/answers/38988.html)
- Check with a different host system if possible.
- Check with a different card if possible.
- **Check Integrated EoU Debug Features:**
    - https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCI-Express-Lin
    - https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-Int
- **Check Board Design Guidelines in GTH/GTY User Guide**
    - https://www.xilinx.com/support/documentation/user_guides/ug576-ultrascale-gth-transceive
    - https://www.xilinx.com/support/documentation/user_guides/ug578-ultrascale-gty-transceiver

- **Check the link training debug guide in the following links:**
    - https://www.xilinx.com/support/answers/56616.html
    - https://www.xilinx.com/support/answers/73361.html

- **Check ltssm status in Vivado ILA; track the ltssm transition by using store_ltssm signal**
    - (See: https://www.xilinx.com/support/answers/71355.html)
    - Run example design simulation for the IP configuration of your design.
    - Observe ltssm transition in example design simulation. Compare this with the ltssm transition in Vivado ILA. If the expected transition is not happening, identify the erroneous ltssm state transition and consult PCIe specification to understand why such transition is occurring.

- **Signal Integrity Check:**
    - As defined in the specification, it is required to put AC coupling capacitors at the transmitter lanes differential signal pair. The value of AC coupling capacitor is between 75 nF and 200 nF. The user should make sure that the PCI express card has an AC coupling capacitor placed in the close proximity of the transmitter lane. Check if the correct AC capacitor value has been put in place or not. There might be a possibility for a cracked capacitor.
    - Review the reference clock data sheet and/or hardware measurement of reference clock, confirm the phase noise specification is below the phase noise mask of the target GT.
    - Confirm the hardware has taken care of "GT Reference Clock Checklist" specified in the respective GT user guide.
    - Ensure Reference clock hardware schematic is as per the requirement of the IO standard of the clock
    - Review the power supply decoupling capacitors on the board and make sure it is as per the guidelines mentioned in GT user guide
    - Check the power supply noise measurement and confirm that peak-to-peak noise as measured at the input pin is according to the value specified in the corresponding GT user guide.
    - Confirm that Termination Resistor Calibration Circuit on the board is as per the reference circuit in the corresponding GT user guide and layout guidelines from the user guide are followed.
    - Fulfill the oscillator vendor's requirement regarding power supply, board layout, and noise specification.
    - Provide AC coupling between the oscillator output pins and the dedicated clock input pins.
    - Meet or exceed the reference clock characteristics as specified in the corresponding datasheets of the device used.
    - Provide a dedicated point-to-point connection between the oscillator and clock input pins.
    - Keep impedance discontinuities on the differential transmission lines to a minimum (impedance discontinuities generate jitter).

- **For Gen3 Link Training Issues:**
  - Check by setting 'Enable Auto RxEq' option to 'True' in the IP Configuration GUI if it is available for the device being used.
  - Sometimes the issue may be related to CPLL vs QPLL. The IP Configuration GUI allows to select PLL option for Gen2 mode only. If the link training issue is related to Gen3, check by configuring the IP as Gen2 by selecting QPLL option.
  - Check by selecting 'Link Partner TX Preset' value to '5'. The default value is 4; this can be selected in the IP configuration GUI.
  - Check if the Link Status 2 register in the PCIe Configuration Space to see if Link Equalization phases were attempted.
  - Try bypassing Phase2/3
  - Use the PCIe PIPE descrambler module in Xilinx PCIe MAC to check for lane-to-lane skew at Gen3 speed. See: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Demystifying-PIPE-interface-packets-using-the-in-built/ba-p/
- If Third party MAC is used, try using the Xilinx example design first to rule out any board or setup issues. The most common issue is the MAC-GT integration issue, please review all connections on mandatory ports as stated in (PG239).
- Check if it is possible to change TX drive parameters on the host.
- Check if PERSIST is enabled as a Bitstream setting. This option is not supported for non-tandem designs while using SPI/BPS Flash and has been known to cause link training issues.
- Ensure that there is no skew between lanes on the board

Note The list above provides a general checklist for debugging link training issues across all devices.

4.4 Documents and Debug Collaterals

Description	URL
https://www.xilinx.com/support/answers/73361.html	Xilinx PCI Express Gen3 Link Training Debugging Guide for UltraScale and UltraScale+ Devices

https://www.xilinx.com/support/answers/72471.html	UltraScale+ FPGA Gen3 Integrated Block for PCI Express (Vivado 2019.1) - Integrated Debugging Features and Usage Guide
https://www.xilinx.com/support/answers/56616.html	7 Series Integrated Block for PCI Express - Link Training Debug Guide
https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Demystifying-PIPE-interface-packets-using-the-in-built/ba-	Demystifying PIPE interface packets using the in-built descrambler module in UltraScale+ Devices Integrated Block for PCI Express Gen3

4.5 Useful Links

Description	URL
Troubleshooting PCI Express Link Training and Protocol Issues	https://pcisig.com/sites/default/files/files/02_01_Troubleshooting_PCI_Express_Link_Training_and_Protocol_Issues_FRO

Chapter 5

Simulation Issue

5.1 General Debug Checklist

- Check by bypassing EQ Phase 2 and 3 using PL_EQ_BYPASS_PHASE23 if there is an issue in simulation with Gen3 design
- For PCIe DMA simulation issues, always ensure that the IP is generated in Vivado with target language set to Verilog. A Timeout error might be seen if the target language is set to VHDL
- When using third party simulators, always ensure that the corresponding supported version of the simulator is used for a given Vivado version
- Check all reset and clock signals. Are these working as expected? Are the frequencies and polarities correct?
- Check the top level connections. Are TX and RX connected as expected?
- Are the Unisim_ver models being called first? If not, put them first in the library call list of the simulation launch.
- Is the script calling any FAST libraries?
- Are both sides expecting PIPE or serial (i.e. is one sending via txp/txn, and the other pipe_txdata)?
- Are the transactions on the user interface synchronous to the user clocks?
- Are all top level inputs driven?

5.2 Documents and Debug Collaterals

Description	URL
PIPE Mode Simulation Using Integrated Endpoint PCI Express Block in Gen3 x8 and Gen2 x8 Configurations	https://www.xilinx.com/support/documentation/application_notes/xapp1184-PIPE-mode-PCIe.pdf

5.3 Useful Links

Description	URL
Vivado Design Suite Tutorial (Logic Simulation)	https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug937-vivado-design-suite-simulation-tutoria

Chapter 6

Interrupt Issue

6.1 General Debug Checklist

- **Check the Interrupt Enable bit in the PCIe Configuration Space.**
    - If you are using MSI, the MSI Control register has this Enable bit.
    - If you are using MSI-X, the MSI-X Control register has this Enable bit.
    - **If you are using Legacy Interrupt, there is no enable, but check the Interrupt Disable bit in the Command Register within the PCI Configuration Space, and Interrupt Pin and the Interrupt Line register of the PCI Configuration Space. Ensure that cfg_interrupt_int is held until cfg_interrupt_done/fail is received.**
        - Certain IP only need this signal asserted for a clock cycle, and some require it to be held steady until the IP responds with a done or fail indicator.
        - Some are also one hot encoded, so check the Product Guide for a given IP.

6.2 Documents and Debug Collaterals

Description	URL
Xilinx PCI Express Interrupt Debugging Guide	https://www.xilinx.com/support/answers/58495.html
UltraScale/UltraScale+ PCI Express Integrated Block - Interrupt debug guide	https://www.xilinx.com/support/answers/72702.html

6.3 Useful Links

Description	URL
The MSI Driver Guide HOWTO	https://www.kernel.org/doc/html/latest/PCI/msi-howto.html

Chapter 7

Versal ACAP CPM Mode for PCI Express

7.1 General Debug Checklist

- **Check if the PCIe IP is detected in lspci**
  - Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCIe-Issues-using-lspci-and-setpci/ba-p/1148199
- **If the PCIe is not detected in lspci, do a warm reboot of the host machine**
  - Re-run lspci
- **If the PCIe IP is still not detected, check if the ltssm state is at L0. Enable PCIe Link Debug Feature.**
  - **Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-Integrated-Block-for-PCIe-Express-link/ba-p/1203707**
    - See "PCIe Link Debug Enablement" section in PG346
- **If the ltssm is not in L0 or the ltssm goes to recovery multiple times, check the eye diagram as discussed in the link below.**
  - https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-Integrate
- **If the eye diagram is not wide enough and the link issues persist, see the document in the following answer record. The document provided in the answer record is for UltraScale+ device but the debug theory applies to Versal ACAP devices as well.**
  - **https://www.xilinx.com/support/answers/73361.html**
    - Key things to check are: noise in power and clock jitter. Confirm both are as per the spec.

- **If the link is up but the packets are not getting returned or written into memory, capture relevant interfaces in Vivado ILA as described in the link below:**
    - https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-CPM-Mo
- **The above design could also be used as an example design.**
    - Start with this design to check if the link and general read/write work or not before doing the debug on the user design.
- **Read the status of phy_rdy**
    - Ref: https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pcie0_attr___phy_rdy.html
- **For Gen3 configuration, check by bypassing phase-2 and phase-3**
    - Ref: https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pcie0_attr___pl_eq_bypass_phase23.html
- **Check if the BARs have been enumerated or not:**
    - https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pci
    - https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pci
    - https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pci
    - https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pci
- For MSI/MSI-X related issue, see: https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html#cpm4_pcie0_attr___cfg_interrupt.html

---

Note

- **To read CPM registers, use xsdb. Check the link below for the details on reading CPM registers using xsdb**
    - https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Register-based-debugging-of-Versa

---

7.2 Issues and Debug Tips/Questions

- **The host is not able to detect VMK180/VCK190 boards with ES1 device**
    - Make sure the SSC is turned off. This can be done in the BIOS

- **PDI programming fails in VMK180/VCK190 boards with ES1 device**
  - **In PS-PMC IO Configuration, make sure the following properties are set for PMC_MIO_37**
    - External Usage: GPIO
    - Output Data: high
    - Drive Strength: 8mA
    - Slew: slow
    - Pull: pullup
    - Direction: out

7.3 Documents and Debug Collaterals

Description	URL
Versal ACAP CPM Mode for PCI Express	https://www.xilinx.com/support/documentation/ip_documentation/cpm_wrapper/v1_0/pg346-cpm-pcie.pdf
Versal CCIX-PCIe Module (CPM) Root port Linux driver	https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/184287255/Versal+CCIX-PCIe+Module+CPM+Root+port+Linux+d
Versal ACAP Register Reference	https://www.xilinx.com/html_docs/registers/am012/am012-versal-register-reference.html
Versal ACAP Technical Reference Manual	https://www.xilinx.com/support/documentation/architecture-manuals/am011-versal-acap-trm.pdf

7.4 Useful Links

Description	URL
Versal ACAP Prime Series Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/versal-prime-product-selection-guide.pdf
Versal ACAP Premium Series Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/versal-premium-psg.pdf
Versal ACAP AI Core Series Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/versal-ai-core-product-selection-guide.pdf

Chapter 8

Versal ACAP Integrated Block for PCI Express

8.1 General Debug Checklist

- **Check if the PCIe IP is detected in lspci**
    - Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCIe-Issues-using-lspci-and-setpci/ba-p/1148199

- **If the PCIe is not detected in lspci, do a warm reboot of the host machine**
    - Re-run lspci

- **If the PCIe IP is still not detected, check if the ltssm state is at L0. Enable PCIe Link Debug Feature.**
    - **Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-Integrated-Block-for-PCIe-Express-link/ba-p/1203707**
        - See "PCIe Link Debug Enablement" section in PG343

- **If the ltssm is not in L0 or the ltssm goes to recovery multiple times, check the eye diagram as discussed in the link below.**
    - https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-Versal-ACAP-Integrate

- **If the eye diagram is not wide enough and the link issues persist, see the document in the following answer record. The document provided in the answer record is for UltraScale+ device but the debug theory applies to Versal ACAP devices as well.**
    - **https://www.xilinx.com/support/answers/73361.html**
        - Key things to check are: noise in power and clock jitter. Confirm both are as per the spec.

8.2 General FAQs

- **The Versal ACAP Integrated Block for PCI Express now comes with BMD example design with the generation of the IP. Is it possible to generate PIO example design instead?**
    - Yes, the PIO example design can be generated. To do so, enter the command below at the Vivado Tcl console prompt after theVersal ACAP Integrated Block for PCIe® IP is generated
    - set_property config.bmd_pio_mode false [get_ips pcie_versal_0]
- **Can I use CPM4 pin MIO38 to reset PL PCIe?**
    - Yes this is possible. Before opening the example design, set the following Tcl property to sue the reset on MIO38 pin: * set_property CONFIG.insert_cips {true} [get_ips pcie_versal_0]
- **In UltraScale+ and previous devices, direct assignment of GTs are not possible in the user constraints. Does similar limitation still exist with Versal devices?**
    - In Versal, the GT locations assignment can be done in the user constraints, while changing GT locations in GT customization IP is not available.
- **The PL PCIE IP Generation flow in Versal ACAP devices is different from previous devices such as UltraScale and UltraScale+. What are the major changes that went into PL based PCIe IP in Versal ACAP devices?**
    - **Below are the major changes:**
        - GT components are updated from Common/Channel to a quad granularity.
        - GT wizard flows are modified to use the Vivado® IP integrator.
        - Only the Vivado IP integrator-based block design flow is currently supported with manual or automatic connectivity.
        - The required GT and PHY IP blocks for Versal ACAP PL PCIe interfaces are outside of the Versal ACAP PL PCIe4 IP.
    - For more information see: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Understanding-the-new-PL-PCIE-IP-Generation-flow-for-Versal-ACAP/ba-p/1215986

8.3 Debug Gotchas

- **Check cfg_function_status signal.**
    - The output of this signal indicates the states of Command Register bits in the PCI Configuration Space of each function: I/O Space Enable, Memory Space Enable, Bus Master Enable and INTx Disable; for details, see: PG343.

- **Use the following signals to trigger ILA when the width or speed change happen.**
  - cfg_negotiated_width
  - cfg_current_speed
- **Check cfg_local_error_out signal to see if there is any local errors e.g. Link Replay Timeout, Link Reply Rollover etc.**
  - For more details, see PG343.
- **If you need to trigger ILA at certain ltssm state, use cfg_ltssm_state signal.**
  - For encoding value for each state, see PG343.

8.4 Documents and Debug Collaterals

Description	URL
Versal ACAP Integrated Block for PCI Express v2.0	https://www.xilinx.com/support/documentation/ip_documentation/pcie_versal/v2_0/pg343-pcie-versal.pdf
Versal ACAP Integrated Block for PCI Express - Release Notes and Known Issues	https://www.xilinx.com/support/answers/73083.html

8.5 Useful Links

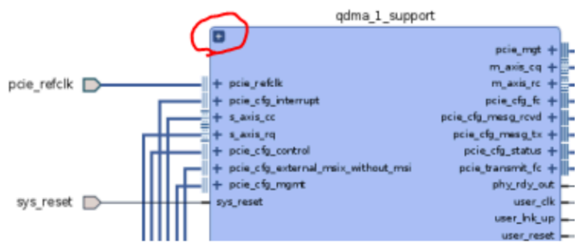
Description	URL
Versal ACAP Prime Series Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/versal-prime-product-selection-guide.pdf

Versal ACAP Premium Series Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/versal-premium-psg.pdf
Versal ACAP AI Core Series Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/versal-ai-core-product-selection-guide.pdf

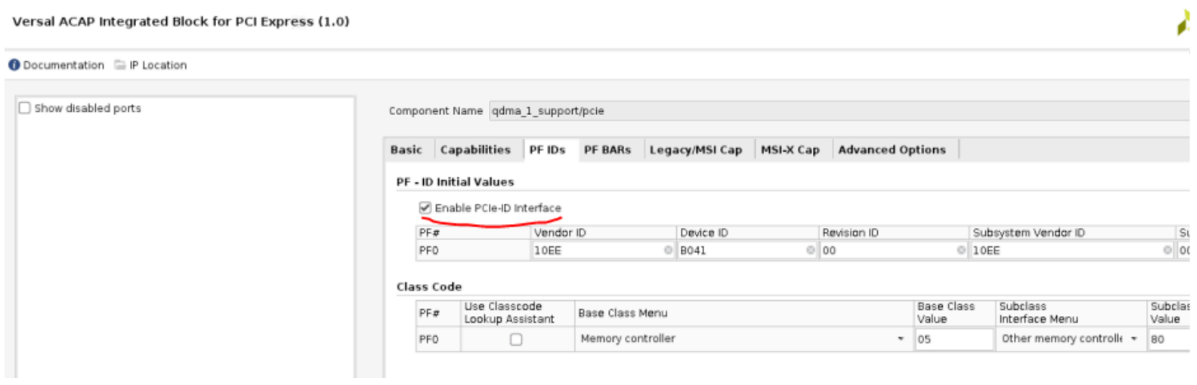
8.6 Specific Issues

- **How to enable PCIe-ID interface option in QDMA IP example design?**
  - The PCIe-IP interface can be enabled as described below:

1. Click '+' sign on the block ending with '_support'.



2. Double click on the PCIe IP and enable PCIe id interface option in 'PF IDs' tab of the GUI.



Component Name qdma_1_support/pcie

PF - ID Initial Values

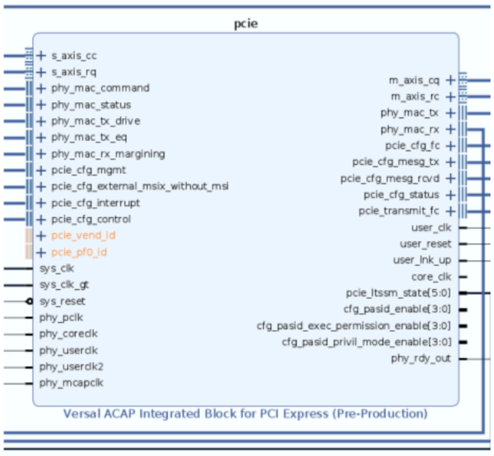
[✓] Enable PCIe-ID Interface

PF#	Vendor ID	Device ID	Revision ID	Subsystem Vendor ID	Su
PF0	10EE	B041	00	10EE	00

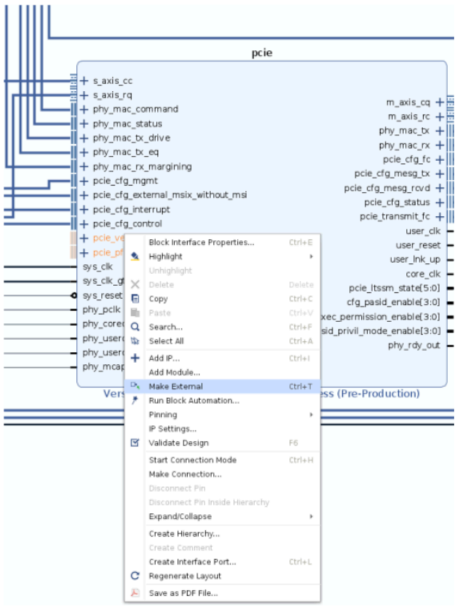
Class Code

PF#	Use Classcode Lookup Assistant	Base Class Menu	Base Class Value	Subclass Interface Menu	Subclas Value
PF0	☐	Memory controller	05	Other memory controlle	80

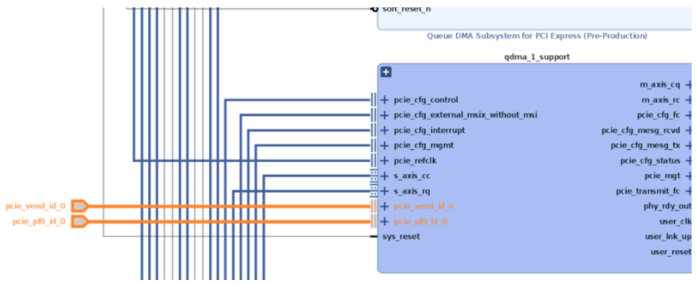
3. This will enable two new interfaces with the PCIe IP as shown below:



4. Select both ports and make them external ports.



5. The interface ports will be brought out in the top level design as shown in the snapshot below.



Chapter 9

UltraScale+ Devices Integrated Block for PCIExpress

9.1 General Debug Checklist

- **Check if the PCIe IP is detected in lspci**
  - Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCIe-Issues-using-lspci-and-setpci/ba-p/1148199
- **If the PCIe is not detected in lspci, do a warm reboot of the host machine**
  - Re-run lspci
- **If the PCIe IP is still not detected, check if the ltssm state is at L0. Enable PCIe Link Debug Feature.**
  - Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCI-Express-Link-Training-Issues-with-Integrated/ba-p/1097525
  - **LTSSM can also be checked by capturing the following signals**
    - *Configuration Status Interface Port Descriptions LTSSM*
- **If the ltssm is not in L0 or the ltssm goes to recovery multiple times, check the eye diagram as discussed in the link below.**
  - Ref: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Debugging-PCI-Express-Link-Training-Issues-with-Integrated/ba-p/1097525
- **If the eye diagram is not wide enough and the link issues persist, see the document in the following answer record.**
  - Ref: https://www.xilinx.com/support/answers/73361.html

- **Check if the phy_link is asserted**
  - *Configuration Status Interface Port Descriptions Phy Link*
- **To check what speed and width the link is negotiated to, capture the following signals in Vivado ILA**
  - *Configuration Status Interface Port Descriptions Negotiated Width and Speed*
- **If the memory read/write is failing and the upstream Memory Read/Write are failing, check if Memory Space Enable and Bus Master Enable bits in the command register are asserted. Capture the following signals in Vivado ILA to check the status.**
  - *Configuration Status Interface Port Descriptions Function Status*
- **Check if there are any local errors occurred by checking the following signal. This signal many not work for all PCIe Link Width/Speed configurations. This signal should be used just as a reference.**
  - *Configuration Status Interface Port Descriptions Local Error*
- **Make sure the following signal is asserted**
  - *Configuration Status Interface Port Descriptions Link Training*
- **If the issue is related to completions, check the following signals**
  - *Requester Completion Descriptor Fields Byte Count*
  - *Requester Completion Descriptor Fields DWORD Count*
  - *Requester Completion Descriptor Fields Error Code*

Note The signals referenced above are from product guide PG213.Please refer to the latest version of the document for new updates and more details.

9.2 General FAQs

- **Which Data Alignment option should be used for better performance?**
  - For performance critical applications, Dword Aligned mode should be used instead of Address-Aligned mode.
- **Does the PIO example design generated with the UltraScale+ Devices Integrated Block for PCI Express IP support more than 1 DW TLP?**
  - In Address Align Mode, the PIO design supports single Dword payload Read and Write PCI Express transactions to 32-/64-bit address memory spaces and I/O space with support for completionTLPs.

- In the case of Dword Align Mode, the PIO Design supports multiple Dword payload (Up to 256 DW) read and write PCI Express transactions to 32-bit Address Memory Spaces with support for completion TLPs.

- **Does UltraScale+ Devices Integrated Block for PCI Express IP support Loopback Master Capability?**
    - The Loopback Master Capability is supported in Root Port mode only

- **Is Lane Reversal Supported in UltraScale+ Devices Integrated Block for PCI Express IP?**
    - Lane Reversal is supported. However, with UltraScale+ Integrated Block for PCI Express IP as an Endpoint, Lane reversal must not be enabled if the Link Partner has the Lane reversal capability. By default EndPoint configuration has Lane Reversal Support disabled through the attribute DISABLE_LANE_REVERSAL.

9.3 Debug Gotchas

- **When the Requester Completion (RC) interface is configured for a width of 256 or 512 bits, depending on the type of TLP and Payload size, there can be significant interface utilization inefficiencies, if a maximum of 1 TLP for 256 bits or 2 TLPs for 512 bits is allowed to start or end per interface beat. This inefficient use of RC interface can lead to overflow of the completion FIFO when Infinite Receiver Credits are advertized. You must either:**
    - Restrict the number of outstanding Non Posted requests, so as to keep the total number of completions received less than 64 and within the completion of the FIFO size selected, or
    - Use the RC interface straddle option.

- **The user application must keep the s_axis_cc_tvalid signal asserted over the duration of the packet.**
    - The integrated block treats the deassertion of s_axis_cc_tvalid during the packet transfer as an error, and nullifies the corresponding Completion TLP transmitted on the link to avoid data corruption.

- **The user application must also assert the s_axis_cc_tlast signal in the last beat of the packet. The integrated block can deassert s_axis_cc_tready in any cycle if it is not ready to accept data.**
    - The user application must not change the values on the CC interface during a clock cycle that the integrated block has deasserted s_axis_cc_tready.

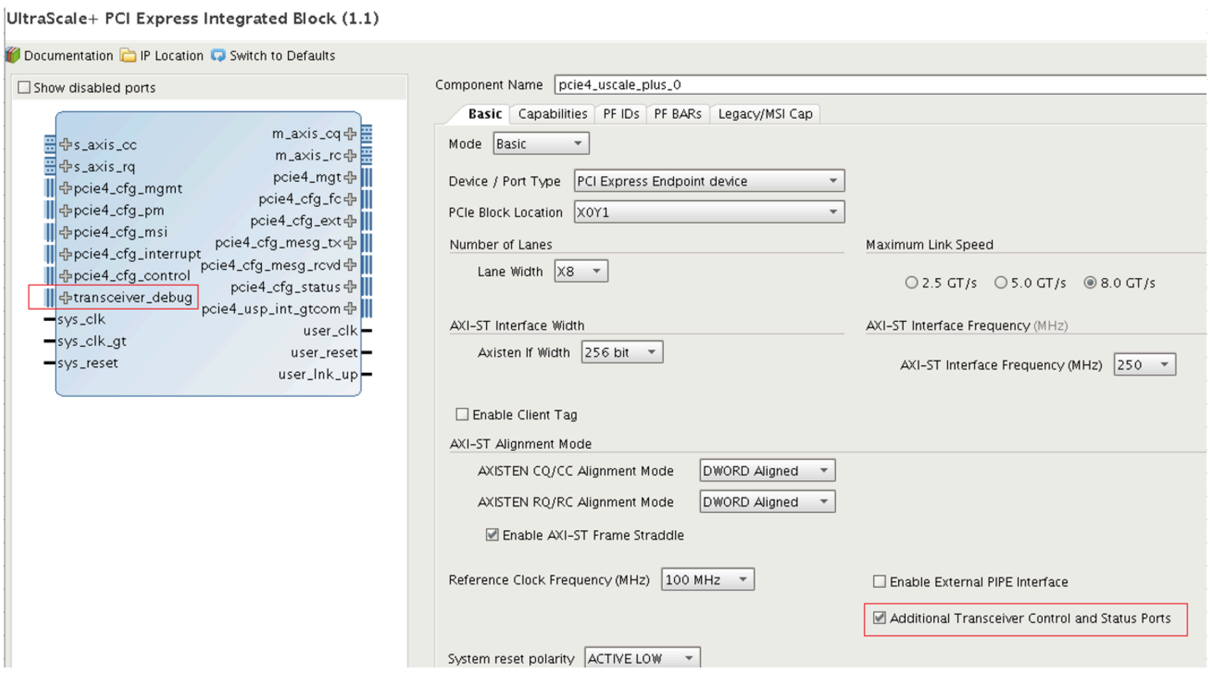
- **The discontinue signal can be asserted only when s_axis_cc_tvalid is active-High.**
    - The integrated block samples this signal when s_axis_cc_tvalid and s_axis_cc_tready are both asserted. Thus, after assertion, the discontinue signal should not be deasserted until s_axis_cc_tready is asserted.

- **s_axis_cc_tvalid: The user application must assert this signal whenever it is driving valid data on the s_axis_cc_tdata bus.**
  - The user application must keep the valid signal asserted during the transfer of a packet.
  - The core paces the data transfer using the s_axis_cc_tready signal.

Note The above debug gotchas are taken from QDMA Product Guide PG213.Please refer to the latest version of the document for more details and new updates.

9.4 Issues/Debug Tips/Questions

- **Enable Transciever Debug Port in Ultrascale PCI Express Integrated Block IP coinfiguration GUI to expose GT debug ports**
  -



UltraScale+ PCI Express Integrated Block (1.1)

Component Name: pcie4_uscale_plus_0

Mode: Basic

Device / Port Type: PCI Express Endpoint device

PCIe Block Location: X0Y1

Number of Lanes

Lane Width: X8

Maximum Link Speed

2.5 GT/s 5.0 GT/s [✓ 8.0 GT/s]

AXI-ST Interface Width

Axisten If Width: 256 bit

AXI-ST Interface Frequency (MHz)

AXI-ST Interface Frequency (MHz): 250

☐ Enable Client Tag

AXI-ST Alignment Mode

AXISTEN CQ/CC Alignment Mode: DWORD Aligned

AXISTEN RQ/RC Alignment Mode: DWORD Aligned

☑ Enable AXI-ST Frame Straddle

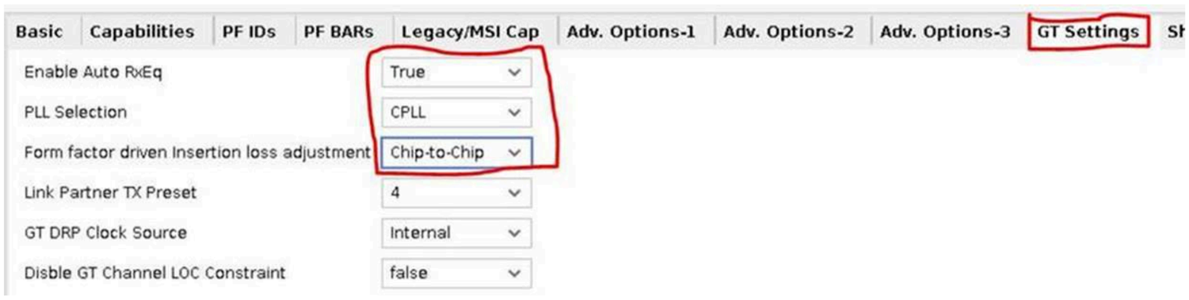
Reference Clock Frequency (MHz): 100 MHz

☐ Enable External PIPE Interface

☑ Additional Transceiver Control and Status Ports

System reset polarity: ACTIVE LOW

- **LTSSM is getting stuck in R.lock state.**
  - Try with the following settings:



Enable Auto RxEq	True
PLL Selection	CPLL
Form factor driven Insertion loss adjustment	Chip-to-Chip
Link Partner TX Preset	4
GT DRP Clock Source	Internal
Disble GT Channel LOC Constraint	false

9.5 Documents and Debug Collaterals

Description	URL
UltraScale+ Devices Integrated Block for PCI Express v1.3	https://www.xilinx.com/support/documentation/ip_documentation/pcie4_uscale_plus/v1_3/pg213-pcie4-ultrascale-plu
UltraScale+ PCI Express Integrated Block Release Notes and Known Issue	https://www.xilinx.com/support/answers/65751.html
Demystifying PIPE interface packets using the in-built descrambler module in UltraScale+ Devices Integrated Block for PCI Express Gen3	https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Demystifying-PIPE-interface-packets-using-the-in-built

https://www.xilinx.com/support/answers/72471.html	UltraScale+ FPGA Gen3 Integrated Block for PCI Express (Vivado 2019.1) - Integrated Debugging Features and Usage Guide
https://www.xilinx.com/support/answers/73361.html	Xilinx PCI Express Gen3 Link Training Debugging Guide for UltraScale and UltraScale+ Devices
https://www.xilinx.com/support/answers/72702.html	UltraScale/UltraScale+ PCI Express Integrated Block - Interrupt debug guide

9.6 Useful Links

Description	URL
UltraScale+ FPGAs Product Table and Product Selection Guide	https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

Chapter 10

DMA/Bridge Subsystem for PCI Express (XDMA IP/Driver)

10.1 General Debug Checklist

- **Check the status of top-level interface signals**
    - *Top-Level Interface Signals*
- **Check the DMA status ports**
    - *Channel 0-3 Status Ports*
- **Review 'Example H2C Flow'**
    - *Example H2C Flow*
- **Review 'Example C2H Flow'**
    - *Example C2H Flow*
- **Review the driver flow**
    - *Initial Setup For H2C and C2H Transfers*
    - *DMA H2C Transfer Summary*
    - *DMA C2H Transfer Summary*
- **For issues related to AXI-Stream designs, capture the following interfaces in Vivado ILA**
    - *H2C Channel 0-3 AXI4-Stream Interface Signals*
    - *C2H Channel 0-3 AXI4-Stream Interface Signals*
- **For issues related to AXI-Memory Mapped designs, capture the following interfaces in Vivado ILA**
    - *AXI4 Memory Mapped Read Address Interface Signals*
    - *AXI4 Memory Mapped Read Interface Signals*

  - *AXI4 Memory Mapped Write Address Interface Signals*
  - *AXI4 Memory Mapped Write Interface Signals*
  - *AXI4 Memory Mapped Write Response Interface Signals*

- **To track read/write to DMA internal registers, capture the following interfaces in Vivado ILA**
  - *Config AXI4-Lite Memory Mapped Write Master Interface Signals*
  - *Config AXI4-Lite Memory Mapped Read Master Interface Signals*
  - *Config AXI4-Lite Memory Mapped Write Slave Interface Signals*
  - *Config AXI4-Lite Memory Mapped Read Slave Interface Signals*

- **For issues related to DMA Bypass, capture the following interface in Vivado ILA**
  - *AXI4 Memory Mapped Master Bypass Read Address Interface Signals*
  - *AXI4 Memory Mapped Master Bypass Read Interface Signals*
  - *AXI4 Memory Mapped Master Bypass Write Address Interface Signals*
  - *AXI4 Memory Mapped Master Bypass Write Interface Signals*
  - *AXI4 Memory Mapped Master Bypass Write Response Interface Signals*

- **For issues related to Descriptor Bypass Port, capture the following interfaces in Vivado ILA**
  - *H2C 0-3 Descriptor Bypass Port*
  - *C2H 0-3 Descriptor Bypass Ports*

10.2 General FAQs

- **The channel register space is defined as 0x00 - H2C Channel Identifier 0x04 - H2C Channel Control etc. in PG195. How to find out address of these registers for different channels?**
  - **Refer to 'PCIe to DMA Address Format' table in PG195. For e.g. address for 'H2C Channel Control' register in Channel-1 will be: 0x00000104**
    - Bit 7:0 - Byte Offset
    - Bit 11:8 - Channel ID[3:0]
    - Bit 15:12 - Submodule within the DMA (0 for H2C channel and 1 for C2H Channel)
    - Bit 31:16 - Reserved

- **Is XDMA Driver supported on ARM Processor?**
  - No, it is not supported.

- **What is the difference between xdma0_bypass_h2c_0, xdma0_bypass_c2h_0 and xdma0_bypass?**
    - xdma0_bypass_h2c* and xdma0_bypass_c2h_* devices are for internal use only so should be ignored.
- **Is it possible to combine different devices such as xdma0_c2h_0, xdma0_c2h_1, xdma0_c2h_2, xdma0_c2h_3 as a single device and select the individual channel through a parameter?**
    - No, that is not possible. Each channel has its device and the driver is setup that way to be used.

10.3 XDMA Performance Debug

- How are you measuring the performance?
- Check the Link Status in lspci to ensure that your link is coming up to the full speed and width
- Have you checked Xilinx Video - "Getting the Best Performance with Xilinx's DMA for PCI Express" ?
- Have you checked XDMA Debug Guide – AR71435?
- Have you checked XDMA Performance Number answer record – AR68049?
- Are you using the Xilinx provided driver or Custom driver?
- If you are using Xilinx provided driver, did you download the driver from an answer record or from the GitHub? If you are using the driver from answer record 65444, can you try with the latest XDMA driver available in the GitHub: https://github.com/Xilinx/dma_ip_drivers/tree/master/XDMA/linux-kernel
- Which version of Vivado are you using? If it is not the latest Vivado version, can you try with the latest version?
- Which silicon device are you using? Is it production or ES?
- Have you tried by using prefetchable BAR memory? – possible to get performance improvement.
- **One of the main factors affecting data throughput is interrupt processing. Once data transfer is completed, the DMA sends an interrupt to the host and waits for ISR to process the status. However, this wait time is not predictable and so the overall total data transfer time is slow and unpredictable. There are a couple of options you can try to work around this.**
    - MSI-X interrupt: Users can try using MSI-X interrupt Instead of MSI or legacy interrupts. With MSI-X interrupt, the data rate is better than with an MSI or legacy interrupt-based design. – How? Can we explain this with some lower level details?
    - Poll mode (See AR:71435): Users can try using Poll mode which gives the best data rate. With Poll mode, there are no interrupts to process.
- Have you tried Descriptor Credit based transfer? The example design supports this only for C2H streaming. (See: AR71435) – How? Can we explain this with some lower level details?
- Have you checked MPS, MRRS values? Systems with better MPS will give a better performance; a typical system would have 128Bytes MPS. (See WP350 for more details). For e.g. the x58 supports up to 256-byte maximum payload size (MPS), and the x38 supports up to 128-byte MPS. Overall, the x58 is a high-end, efficient machine and the x38 is a value-based machine. The performance of the x38 will be lower in comparison to the x58.

- Have you checked if you have a stable link? Does LTSSM go to recovery intermittently or continuously? (See: AR71355)
- Do you have a link analyzer to see if there are any NAKs being issued? You could also check this by looking at the PIPE interface using Gen3 descrambler module. See Xilinx Blog for more details.
- What speed and lane width configuration are you using? Have you tried with Gen3x8?
- What is the transfer size of the DMA? Larger the transfer size (within the limit of the application), better is the performance.
- How many channels are you using in XDMA? Higher the number of channels, better the performance but the tradeoff is, it will consume more logic inside the device.
- Have you checked credit information? Is there enough credit from the link partner for the XDMA IP to initiate data transfer?
- Are you using BRAM or DDR for your endpoint memory?
- If you have configured your design for Gen3, can you try by configuring it as Gen2? Gen3 could be more error-prone if the signal integrity on the board is not robust enough. Configuring the IP for a lower speed will reduce the possibility of signal integrity issue kicking in. If this happens, you should see an increase in performance.
- What is the AXI side data width and frequency configured for? Have you tried higher values if the option is available?
- Do you have AXI Smart Connect in your design? If so, can you try by replacing it with AXI Interconnect IP?
- If you have AXI Interconnect, try by using it in synchronous mode.
- See if your AXI system is on the same data width. This could provide some performance boost if the performance is affected due to hardware.
- Try by disabling Narrow Burst, if it is enabled.
- Check in the XDMA log if there is a call for repeated ISR.

10.4 Debug Gotchas

- **The Host can enable one or more interrupt types from the specified list of supported interrupts during IP configuration.**
    - The IP only generates one interrupt type at a given time even when there are more than one enabled. MSI-X interrupt takes precedence over MSI interrupt, and MSI interrupt take precedence over Legacy interrupt. The Host software must not switch (either enable or disable) an interrupt type while there is an interrupt asserted or pending.
- **The user_irq_ack assertion indicates the requested interrupt has been sent to 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.

- **For host-to-card transfers, data is read from the host at the source address, but the destination address in the descriptor is unused. Packets can span multiple descriptors.**
    - The termination of a packet is indicated by the EOP control bit. A descriptor with an EOP bit asserts tlast on the AXI4-Stream user interface on the last beat of data.
- **The tkeep bits for transfers for all except the last data transfer of a packet must be all 1s.**
    - On the last transfer of a packet, when tlast is asserted, you can specify a tkeep that is not all 1s to specify a data cycle that is not the full datapath width. The asserted tkeep bits need to be packed to the lsb, indicating contiguous data.
- **The length of a C2H Stream descriptor (the size of the destination buffer)**
    - It must always be a multiple of 64 bytes.
- **The user IRQ example design can be generated by using the following Tcl command.**
    - set_property -dict [list CONFIG.usr_irq_exdes {true}] [get_ips <ip_name>]

Note The above debug gotchas are taken from QDMA Product Guide PG195.Please refer to the latest version of the document for new updates.

10.5 Issues/Debug Tips/Questions

- **DMA uses PCIe Base IP and GT similar to the base PCIe Integrated IP.**
    - If there are issues related to link up, enumeration, general PCIe boot-up, or a detection issue, please follow the PCIe debug strategy similar to the base PCIe Integrated IP.
- **Driver fails to load**
    - Enable debug in the driver. Check the output of the dmesg command to help narrow down where the issue is.Once it has been narrowed down to which function calls it fails, do a PIO transfer to read or write to the particular register that the driver is accessing to see what response you get. The primary section to look for is the probe function inside of the xdma-core.c file. This probe function is called when you insert the driver into the Kernel and will read various DMA status registers to indicate which features are available and set an initialization value to it.

- **XDMA driver hangs when doing C2H streaming transfer**
  - **Check by loading the driver into the kernel by enabling the descriptor “credit based”. Modify the Insmod command in the load_driver.sh file as follows before sourcing it.**
    - insmod ../driver/xdma.ko enable_credit_mp=1

10.6 Documents and Debug Collaterals

Description	URL
Download XDMA Driver	https://github.com/Xilinx/dma_ip_drivers
DMA/Bridge Subsystem for PCI Express v4.1	https://www.xilinx.com/support/documentation/ip_documentation/xdma/v4_1/pg195-pcie-dma.pdf
DMA Subsystem for PCI Express - Release Notes and Known Issues for Vivado 2015.3 and newer tool versions	https://www.xilinx.com/support/answers/65443.html
DMA Subsystem for PCI Express - Driver and IP Debug Guide	https://www.xilinx.com/support/answers/71435.html

Reading AXI PCIe Gen3/XDMA internal registers using JTAG to AXI Master IP	https://www.xilinx.com/support/answers/71322.html

10.7 Useful Links

Description	URL
QDMA vs XDMA	https://forums.xilinx.com/t5/PCIe-and-CPM/QDMA-vs-XDMA/td-p/944098

DMA/Bridge Subsystem for PCI Express (Bridge IP Endpoint)

11.1 Debug Gotchas

- **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.**
  -

- **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>]

- **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. s_axi_wstrb can be used to facilitate data alignment to an address boundary.**
  - s_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 s_axi_wstrb must be continuous from the first byte enable to the last byte enable.

- **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.

- **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.**

  -

- **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.**

  -

- **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.**

  -

- **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.**
    - 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.
- **Endpoint mode only (parameter: set_finite_credit)**
    - 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.

- **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.

Note The above debug gotchas are taken from QDMA Product Guide PG195.Please refer to the latest version of the document for new updates and more details.

11.2 General Debug Checklist

- **Confirm the clocking architecture is correct**
    - *Clocking Diagram (UltraScale+ Devices)*
- **Confirm the reset connection is correct**
    - **Endpoint:**
        - *Endpoint System Reset Connection (UltraScale+ Devices)*
    - **Rootport:**
        - *Root Port System Reset Connection (UltraScale+ Devices)*
- **Check the status of Phy Status Control Register**
    - *PHY Status_Control Register*
- **If the issue is related to incoming or outgoing packets from the user logic, check the following interface signals in Vivado ILA**
    - *AXI Slave Interface Signals*
    - *AXI Master Interface Signals*
- **If there is an issue with bridge register reads or write, check the following interface**
    - *AXI4-Lite Control Interface Signals*
- **For interrupt related issues, check the following signals and register:**
    - *AXI Bridge for PCIe Gen3 MSI Signals*
    - *AXI Bridge for PCIe Gen3 MSI-X Signals*
    - *DMA_Bridge Subsystem for PCIe in Bridge Mode Interrupt Signals*
- **For Root Port related issues, check:**
    - *Root Port Status_Control Register*
    - *Root Port Error FIFO Read Register*
    - *Interrupt Decode Register*

Note Please refer to the latest version of PG194 for new updates and more details on referenced signals and registers.

11.3 Issues/Debug Tips/Questions

- **Relation between PCIe link up and axi_aresetn**
    - When the PCIe IP is in D0 uninitialized state which means it physically linked up but there hasn't been any exchanges of INITFC and enumeration (Config TLPs) and thus no TLPs can get through. Only when axi_aresetn (which is essentially user_reset from PCIe IP) is released, that's when the Transaction Layer is up. AXI peripherals should therefore use the axi_aresetn indication from the AXI Bridge IP before sending over a packet because this indicates when the Bridge and PCIe link is usable].
- **Accessing the host testbench before the link is established results in the bridge IP to hang**
    - If the memory access is done after the link is established, it will return Slave Error or Decode error depending on whether it's non-existent PCIe or AXI memory addresses. These type of errors should always get a response. When the memory access is attempted when there is no link, the system would hang as there is no response on the AXI bus as ready is not asserted without a link. In AXI, there is no Timeout mechanism by default unlike PCIe. A possible workaround would be to use the firewall IP.
- **The AXI outstanding number is set to be 8 (C_S_AXI_NUM_READ = 8), but only one AXI read can be issued instead of 8.**
    - Check by disabling "AXI Slave narow burst"
- **The read access from the PCIe link reads the data equal to the size of the AXI bus width event though the read access is shorter than the AXI bus width**
    - This is a limitation in the IP. For PCIe® requests with lengths greater than 1 Dword, the size of the data burst on the MasterAXI 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.

11.4 Documents and Debug Collaterals

Description	URL
AXI Bridge for PCI Express Gen3 Subsystem v3.0	https://www.xilinx.com/support/documentation/ip_documentation/axi_pcie3/v3_0/pg194-axi-bridge-pcie-gen3.pd
Reading AXI PCIe Gen3/XDMA internal registers using JTAG to AXI Master IP	https://www.xilinx.com/support/answers/71322.html
AXI Bridge for PCI Express - Release Notes and Known Issues for Vivado 2013.1 and newer tool versions	https://www.xilinx.com/support/answers/54646.html

11.5 Useful Links

Description	URL
AXI PCI Express (PCIe) Gen 3 Subsystem	https://www.xilinx.com/products/intellectual-property/axi_pcie_gen3.html

11.6 Specific Issues

- **Data trasfer issue**
  - Probe the M_AXI and S_AXI interfaces (depending on the direction of the data flow you are debugging) and see if you can trigger / catch the request you have made.
  - If the packet shows up in here, check the address field (ARADDR for Read Request or AWADDR for Write Request) to ensure that the address is in the expected AXI Address (after translation) for this request.
  - AXI Translation vector is at C_PCIEBAR2AXIBAR_# parameter and C_PCIEBAR2AXIBAR_# parameter. AXI BAR settings are at: C_AXIBAR_# parameter
  - **If the M_AXI and S_AXI interfaces do not show anything but this is not the first AXI packet prior to the failure, then check previous AXI transactions and make sure that they are completed.**
    - For Writes you must see a corresponding BRESP, BVALID, and BREADY for each request.
    - For Reads you must see a corresponding RRESP, RVALID, and RREADY for each request.
    - If there is any packet that is not completed (check both the Writes and Reads interface), investigate those first before moving forward, because PCIe has a strict packet ordering rule and that pending request might have halted the data pipeline.
  - **If the M_AXI and S_AXI interfaces do not show anything and this is the first AXI packet prior to the failure (or no other pending transaction), then check the following:**
    - The Bridge Enable bit in the Bridge Control Register (Offset 0x148 on s_axil_* AXI Lite interface) must be set to 1 before any data can flow through
    - Check the AXIS_TX/RX interface and see if you can spot the packet there.
    - If it is visible in the AXIS* interface, then the problem is in the Bridge.
    - For AXI MM Gen2, internal state machines can be seen by following the AXIS_* signals datapath

Chapter 12

QDMA Subsystem for PCIExpress (IP/Driver)

12.1 QDMA Debug Flow

Get IP Configuration Details
Collect IP Configuration Parameters

Check Debug Checklist
Check *Debug Gotchas*
Check *QDMA Global Port Descriptions*
Check *General Debug Checklist*
Check *Issues/Debug Tips/Questions*

Verify Driver Initialization Data/Driver Flow			
H2C MM	**C2H MM**	**H2C ST**	**C2H ST**
Review H2C MM Driver Flow	*Review C2H MM Driver Flow*	*Review H2C ST Driver Flow*	*Review C2H ST Driver Flow*
Review dmesg log	*Review dmesg log*	*Review dmesg log*	*Review dmesg log*
Review Context Programming	*Review Context Programming*	*Review Context Programming*	*Review Context Programming*

Descriptor Update	
Check Global Error Registers	*Check Global Error Registers*
Check PIDX Update	*Check Software Context*
Check Traffic Manager Status Interface for new descriptor	*Check Hardware Context*
Check Queue Invalid bit in Hardware Context	*Check Credit Context*

Descriptor Fetch	
Check Descriptor Engine Error	*Check Software Context*
Review Descriptor Fetch Flow	*Check Hardware Context*
Check if Fetch Credit is enabled or not	*Check Credit Context*
Check if Descriptor Fetch Pending is enabled	*Check Prefetch Context*

QDMA Data Transfer - 1 (for all modes)			
H2C ST	**C2H ST**	**H2C MM**	**C2H MM**
Check Global Error Registers	*Check Global Error Registers*	*Check Global Error Registers*	*Check Global Error Registers*

H2C ST Engine Error	*C2H ST Engine Error*	*H2C MM Engine Error*	*C2H MM Engine Error*
Review H2C ST Driver Flow	*Review C2H Exception Handling*	*Reiview H2C MM Driver Flow*	*Review C2H MM Driver Flow*
AXI4-Stream H2C Port Descriptions	*Review C2H ST Driver Flow*	*AXI4 Memory Mapped DMA Read Interface Signals*	*AXI4 Memory Mapped DMA Write Interface Signals*
Review C2H/H2C Queue Flow	*AXI4-Stream C2H Port Descriptions*	*Review C2H/H2C Queue Flow*	*Review C2H/H2C Queue Flow*
QDMA Descriptor Credit Input Port Descriptions	*AXI4-Stream C2H Completion Port Descriptions*	*QDMA Descriptor Credit Input Port Descriptions*	*QDMA Descriptor Credit Input Port Descriptions*
QDMA TM Credit Output Port Descriptions	*Review Completion Queue Flow*	*QDMA TM Credit Output Port Descriptions*	*QDMA TM Credit Output Port Descriptions*
Queue Status Ports	*Review C2H/H2C Queue Flow*	*Queue Status Ports*	*Queue Status Ports*
Queue status data	*Check AXI4 Stream Status Ports*	*Queue status data*	*Queue status data*
	QDMA Descriptor Credit Input Port Descriptions		
	QDMA TM Credit Output Port Descriptions		
	Queue Status Ports		
	Queue status data		

QDMA Data Transfer- 2 (C2H Stream Only)		
C2H Stream - Internal Mode (Always Cache Mode)	**Cache Bypass Mode (Only for C2H Stream)**	**Simple Bypass Mode (Only for C2H Stream)**
Check Prefetch Context if Prefetch Enabled		*Check C2H Simple Bypass Mode Flow*
	Check Prefetch Context if Prefetch Enabled	*QDMA C2H Descriptor Bypass Output Port Descriptions*
	QDMA C2H-Streaming Cache Bypass Input Port Descriptions	
	Check C2H Cache Bypass Mode Flow	

QDMA Data Transfer - 3			
	H2C MM	**C2H MM**	**H2C ST**
Internal Mode	*Check H2C Internal Mode Flow*	*QDMA C2H-MM Descriptor Bypass Input Port Descriptions*	*Check H2C Internal Mode Flow*
Descriptor Bypass Mode	*QDMA H2C Descriptor Bypass Output Port Descriptions*	*QDMA C2H Descriptor Bypass Output Port Descriptions*	*Check H2C Bypass Mode Flow*
	Check H2C Bypass Mode Flow		*H2C ST Desc Bypass Output Ports*
	QDMA H2C-MM Descriptor Bypass Input Port Descriptions		*QDMA H2C-Streaming Bypass Input Port*

Status Update			
H2C MM	**C2H MM**	**H2C ST**	**C2H ST**
Check AXI MM H2C Writeback Status	*Check AXI MM C2H Writeback Status*	*Check AXI ST H2C Writeback Status Descriptor*	*Check AXI ST Completion Status*

12.2 Debug Gotchas

- **If a queue is associated with interrupt aggregation, Xilinx recommends that the status descriptor be turned off, and instead the DMA status be received from the interrupt aggregation ring.**
    -
- **To help a user-developed traffic manager prioritize the workload, the available descriptor to be fetched (incremental PIDX value) of the PIDX update is sent to the user logic on the interface. Using this interface it is possible to implement a design that can tm_dsc_sts prioritize and optimize the descriptor storage.**
    -
- **Pre-fetch mode can be enabled on a per queue basis, and when enabled, causes the descriptors to be opportunistically prefetched so that descriptors are available before the packet data is available. The status can be found in prefetch context. This significantly reduces the latency by allowing packet data to be transferred to the PCIe integrated block almost immediately, instead of having to wait for the relevant descriptor to be fetched.**
    -
- **Queue-based interrupts and user interrupts are allowed on PFs and VFs, but error interrupts are allowed only on PFs.**
    -
- **The driver can be in polling or interrupt mode. Either way, the driver identifies the new CMPT entry either by matching the color bit or by comparing the PIDX value in the status descriptor against its current software CIDX value.**
    -
- **Prior to enabling the queue, the hardware and credit context must first be cleared.**
    -

- The descriptor engine is informed of the availability of descriptors through an update to a queue's descriptor PIDX. This portion of the context is direct mapped to the QDMA_DMAP_SEL_H2C_DSC_PIDX and QDMA_DMAP_SEL_C2H_DSC_PIDX address space.

  -

- In normal operation, for an enabled queue, each time the irq_arm bit is asserted or PIDX of a queue is updated, the descriptor engine asserts tm_dsc_sts_valid.

  -

- The memory mapped DMA engines (H2C and C2H) are enabled by setting the run bit in the Memory Mapped Engine Control Register. When the run bit is deasserted, descriptors can be dropped. Any descriptors that have already started the source buffer fetch will continue to be processed. Reassertion of the run bit will result in resetting internal engine state and should only be done when the engine is quiesced.

  -

- There are two primary error categories for the DMA Memory Mapped Engine. The first is an error bit that is set with an incoming descriptor. In this case, the DMA operation of the descriptor is not processed but the descriptor will proceed through the engine to status update phase with an error indication. This should result in a writeback, interrupt, and/or marker response depending on context and configuration. It will also result in the queue being invalidated. The second category of errors for the DMA Memory Mapped Engine are errors encountered during the execution of the DMA itself. This can include PCIe read completions errors, and AXI bresp errors (H2C), or AXI errors and PCIe write errors due to bus master enable or functio bresp level reset (FLR), as well as RAM ECC errors. The first enabled error is logged in the DMA engine. Please refer to the Memory Mapped Engine error logs. If an error occurs on the read, the DMA write will be aborted if possible. If the error was detected when pulling write data from RAM, it is not possible to abort the request. Instead invalid data parity will be generated to ensure the destination is aware of the problem. After the descriptor which encountered the error has gone through the DMA engine, it will proceed to generate status updates with an error indication. As with descriptor errors, it will result in the queue being invalidated. See Descriptor Engine Errors.

  -

- The H2C stream interface is shared by all the queues, and has the potential for a head of line blocking issue if the user logic does not reserve the space to sink the packet. Quality of service can be severely affected if the packet sizes are large. The Stream engine is designed to saturate PCIe for packet sizes as low as 128B, so Xilinx recommends that you restrict the packet size to be host page size or maximum transfer unit as required by the user application

  -

- **A performance control provided in the H2C Stream Engine is the ability to stall requests from being issued to the PCIe RQ/RC if a certain amount of data is outstanding on the PCIe side as seen by the H2C Stream Engine. To use this feature, the SW must program a threshold value in the H2C_REQ_THROT (0xE24) register. After the H2C Stream Engine has more data outstanding to be delivered to the user logic than this threshold, it stops sending further read requests to the PCIe RQ/RC. This feature is disabled by default and can be enabled with the H2C_REQ_THROT (0xE24) register. This feature helps improve the C2H Stream performance, because the H2C Stream Engine can make requests at a much faster rate than the C2H Stream Engine. This can potentially use up the PCIe side resources for H2C traffic which results in C2H traffic suffering The H2C_REQ_THROT (0xE24) register also allows the SW to separately enable and program the threshold of the maximum number of read requests that can be outstanding in the H2C Stream engine. Thus, this register can be used to individually enable and program the thresholds for the outstanding requests and data in the H2C Stream engine.**
  -
- **The QDMA requires software to post full ring size so the C2H stream engine can fetch the needed number of descriptors for all received packets. If there are not enough descriptors in the descriptor ring, the QDMA will stall the packet transfer. For performance reasons, the software is required to post the PIDX as soon as possible to ensure there are always enough descriptors in the ring**
  -
- **In cache bypass or internal mode prefetch mode can be turned on which will prefetch descriptor and that this will reduce transfer latency significantly. When prefetch mode is enabled, user can not send credits as input in "QDMA Descriptor Credit input ports". Credits for all queues will be maintained by prefetch engine.**
  -
- **When C2H Streaming Completion is enabled, after the packet is transferred, CMPT entry and CMPT status are written to C2H Completion ring. PIDX in the Completion status can be used to indicate the currently available completion to be processed**
  -
- **If the PCIe link goes down during DMA operations, transactions may be lost and the DMA may not be able to complete. In such cases, the AXI4 interfaces will continue to operate. Outstanding read requests on the C2H Bridge AXI4 MM interface receive correct completions or completion with a slave error response. The DMA will log a link down error in the status register. It is the responsibility of the driver to have a timeout and handle recovery of a link down situation.**
  -

Note The above debug gotchas are taken from QDMA Product Guide PG302.Please refer to the latest version of the document for new updates and more details.

12.3 General FAQs

- **There are multiple QDMA IP versions for e.g. CPM QDMA and PL PCIe QDMA in Versal devices, different QDMA versions such as 3.0 and 4.0 etc. Do these need seperate driver?**
    - No, the driver provided in the link below should work for all QDMA versions.
    - https://github.com/Xilinx/dma_ip_drivers
- **Does the QDMA driver support metadata field?**
    - Metadata is intended to be used by the user logic; the drivers that we currently provide is for reference only. It doesn't provide an API to set the metadata.
- **Is there anything built in to the completion context or completion status that will let software know that the entry was for a marker request, and not an actual packet?**
    - Yes there is completion packet and status that is generated for marker request. User can send in a completion data that is recognized for Marker and can look for it.
- **Who generates tlast?**
    - PCIe block does not send rc_tlast; DMA block figures out tlast based on the number of bytes.
- **In simple bypass mode, what is the correct way of driving credit input ports for prefetch mode?**
    - When a user choose credit input mode (simple bypass mode) then there is no need to prefetch the credits. Because the user is giving the credits to fetch the descriptors so the DMA will go and fetch those credits and send it out. So in Simple bypass mode there is no prefetch mode
- **When is s_axis_c2h_mty signal asserted?**
    - It is asserted only when tlast is asserted. If there are any empty bytes then it needs to be passed when tlast is asserted. If mty is asserted when it is not the last packet, it is reported as an error in QDMA_C2H_ERR_STAT (0XAF0). If the erorr occurs, the log should show - "qdma_hw_error_handler(): Detected Fatal MTY mismatch error".
- **When is this error reported - "qdma_hw_error_handler(): Detected Packet length mismatch error" ?**
    - This error is reported when the total packet length do not match the signal from the s_axis_c2h_ctrl.len. bit-1 of QDMA_C2H_ERR_STAT (0XAF0) is set when this error occurs.

- **PG mentions h2c_byp_in_st_at port but it is not available in IP interface?**
  - To use this port, enable_at_ports {true} needs to be set in vivado tcl console. This is de-featured as it hasn't been tested thoroughly.

- **In PG302, it says "The driver sets up the mapping of the C2H descriptor queue to the CMPT queue". While going through CMPT context and DPDK driver, we don't find how this is done.**
  - The User should know which completion context (completion ring) is pointed to which QID. The user (software/driver) will program 0x844 register to program the completion context; at this time, user also needs to give QID which associates which completion points to which QID (and which C2H descriptor queue).

- **Can multiple C2H descriptor queues mapped to the same CMPT queue?**
  - No, this use case is not supported.

- **QDMA IP supports 4 PFs and 252 VFs, how are function numbers assigned?**
  - First for 4PFs, function number 0, 1, 2, and 3 will be programmed. Then all the VF's belonging to PF0, then VF's belonging to PF1.. etc.

- **Is Jumbo packet supported in QDMA? I have a customer wanting to use packet size of 10k.**
  - Yes Jumbo packet upto the size of 64K is supported. The provided driver breaks it into chunks of 4K to cater the requirement to meet 4K boundary in C2H direction. Jumbo packet is supported in both Internal Mode and Bypass Mode.

- **Is it the hard requirement that user logic should monitor descriptor credits first and then send C2H data to QDMA IP?**
  - Yes, the user is required to check for the descriptor credits before sending the packet and this is a hard requirement. If not, the DMA will drop the packet.

12.4 General Debug Checklist

- For " read file: Input/output error", check (Xilinx Answer 72813)
- If the simulation of the example design generated with "AXI Stream with Completion" is not working, make sure PF0 BAR2 is enabled. The example design implements registers which are accessed from the host through BAR2.
- Check if the base address of all ring buffer (H2C, C2H and CMPT) are aligned to the 4K address.
- Make sure the FMAP register is programmed. This is required to associate how many queues are allocated to each function.
- Check if fetch crediting is enabled or not (See Credit Descriptor Context Structure); the user logic is required to provide a credit for each descriptor that should be fetched.
- Make sure the hardware and credit context are cleared before enabling the queue.

- Check PIDX and CIDX values. Make sure there are enough descriptors.

12.4.1 QDMA Performance Debug Questions

- Is the driver being used a Linux Kernel Driver or a DPDK Driver?
- How many queues are required and enabled?
- What is the traffic pattern being applied?
- Achieved performance for desired traffic pattern?
- Required performance for desired traffic pattern?
- What is the configuration of the design being tested? Is it Gen3x16?
- Are you using the Xilinx provided driver or a Custom driver?
- If you are using Xilinx provided driver, did you download the driver from an answer record or form the GitHub?
- What are MPS and MRRS values? Typically, MRRS is 512. If line rate is not observed for packet size 4K on a system, PCIe MRRS could be tuned to 2048 depending on the system used.
- Are you using BRAM or DDR?
- Have you checked by settings BARs as prefetchable?
- What is the Packet size?

12.4.2 QDMA Performance Debug Checklist

- Make sure that "Relax ordering" is enabled and Host accept Relax ordering.
- Make sure "Extended tag" is enabled.
- Make sure all the Ring base address and data address are 4K aligned. (or to MPS boundary)
- **Review the following note in:** https://www.xilinx.com/support/answers/71453.html
    - Typically, networking applications optimize for small packet performance and so can use more queues to saturate the Ethernet interface, while compute or storage applications might optimize for 4KB performance and saturate with fewer queues. As the report suggests, more queues help achieve small packet performance, but the max number of queues cannot exceed the number of threads available for the application.
    - For the streaming mode this report suggests that 4 and more queues with prefetch enabled results in the high performance for different packet sizes.
    - For the memory mapped mode, the QDMA IP easily achieves the line rate with the typical 4K workload even with a single queue when using BRAM. If DDR is desired, more queues might be needed to obtain the best performance. This depends on the memory configuration and the access pattern. For example, concurrent read and write to the same memory bank would greatly reduce the efficiency and should be avoided if possible.
    - The bi-directional performance should be expected to be lower than uni-directional H2C and C2H, because the PCIe RQ interface is shared.
    - In a multi-socket machine where NUMA is enabled, the latency for DMA reads can be prohibitively high, causing lower performance. Caution must be taken in the driver to avoid using the memory far away from the CPU core.
    - Are you using a PCIe slot which hangs off directly from the CPU?

    - Recommended PCIe setting: Enable Extended Tag, Enable Relaxed Ordering
    - Recommended QDMA setting: TX Queue depth = 2048, RQ Queue depth = 2048, Packet Buffer Size = 4096, Burst Size = 4

- **DPDK Driver Specific Settings:**
    - Boot Setting default_hugepagesz=1GB hugepagesz=1G hugepages=20
    - If the system in use runs additional apps, it is always good practice to set aside CPU cores for dpdk workload.
    - The additional kernel command-line parameters below can be added to GRUB boot settings: isolcpus=1-<n> nohz_full=1- rcu_nocbs=1-<n>
    - isolcpus isolates cpus 1 to from kernel scheduling so that they are set aside for dedicated tasks like dpdk.
    - nohz_full will put cpus 1 to in adaptive-ticks mode so as not to interrupt them with scheduling-clock interrupts.
    - rcu_nocbs will fence off cpus 1 to from random interruptions of softirq RCU callbacks.
    - **Performance Setting:**
        - isolcpus=1-16
        - Disable iptables/ip6tables service
        - Disable irqbalance service
        - Disable cpuspeed service Point scaling-governor to performance

- **Disable CPU power savings from the kernel command-line**
    - **In some cases, disabling CPU power savings from the BIOS alone might not make the CPU(s) run on full horse-power. As an additional confirmation, disable power savings from the kernel command-line too:**
        - **processor.max_cstate=0 intel_idle.max_cstate=0 intel_pstate=disable**
            - This will disable power savings on CPUs and disable intel_idle as well as the intel_pstate cpu frequency scaling driver. In addition, always double-confirm from 'cat /proc/cpuinfo | grep -i Mhz' output that all of the intended CPUs are operating at max speed.

- **General Guidance**
    - The internal QDMA descriptor fetch engine is limited in its operation and is optimized for smaller numbers of queues (16 or less). This is common for compute applications.

- **To achieve high QDMA performance in cases where large numbers of queues are used (32+), small packet sizes are used (512-bytes or less), and worst-case traffic patterns are used (round-robin one transfer per queue); external descriptor management or Simple Bypass mode should be used. This is common for networking applications.**
    - **You may still be able to achieve sufficient performance for your application without using simple bypass if you can avoid one of the conditions described above.**
        - **Decrease the number of Queues by coalescing data streams into fewer queues.**
            - 16 or fewer Queues
        - **Increase packet size by combining transactions in the user-logic to create larger transfers.**
            - DPDK C2H: 1+ KBytes, DPDK H2C: 512+ Bytes
            - Linux Kernel Driver C2H: 1+Kbytes, Linux Kernel Drive H2C: 2Kbyte
        - **Modify the traffic pattern to avoid round-robin.**
            - **Send more transactions (burst) to a fewer numbers of queues (16 or less) before cycling in/out traffic from new queues.**
                - Min burst size = (packet size limit from (2) / (actual packet size)

- Make sure the driver is not reading the context when the queue is enabled; it can result in reduced performance.
- Excessive writebacks events can severely reduce the descriptor enginer performance and consume bandwidth to the host.
- H2C stream interface is shared by all the queues. Stream engine is designed to saturate PCIe for packets sizes as low as 128b. Recommendation: restrict the packet size to be host page size or maximum transfer unit as required by the user application.
- If there is both H2C and C2H traffic, use H2C_REQ_THROT, it will throttle H2C making way for more PCIe resource for C2H.
- The QDMA Subsystem for PCIe has a shallow completion input FIFO of depth 2. For bett erperformance, add FIFO for completi on input as shown in the diagram below. Depth and width of the FIFO depends on the use case. Width is dependent on the largest CMPT size for the applicati on, and depth is dependent on performance needs. For best performance for 64 Byte CMPT, a depth of 512 is recommended.
- If there is an issue with H2C performace, make sure h2c_byp_in_st_sdi port is asserted once in every 32 or 64 descriptors. If "h2c_byp_in_st_sdi" port is always high, for every packet transferred, there needs to be a status update going from QDMA to the Host. This

will impact performance because the DMA will have to share the bus with "status update" for every single packet along with the other requests going to the Host. Also, it slows down the descriptor engine as it needs to switch context every time there is a status update.

12.5 Issues/Debug Tips/Questions

- **QDMA Driver failed to load (qdma:qdma_device_online: qdma_init failed -16.)**
  - Make sure to connect st_rx_msg_rdy, tm_dsc_sts_rdy and soft_reset_n to 1'b1
- **tm_dsc_sts_error reports error in h2c direction. The issue seems to occur after the first PIDX update write to QDMA.**
  - Make sure the Ring Size configuration is done correctly.
- **CMPT Packet is not sent**
  - Review Software Context, Completion Context and Prefetch Context for each queue.
  - Review Software Context, Completion Context and Prefetch Context for each queue.
  - Which mode is used? Streaming mode or Memory Mapped mode?
  - How many queues are enabled?
  - How many packets are sent per queue?
  - What is the packet size?
  - When the issue occurs what happens when you do tear down? Does it timeout?
  - Is this custom driver or the driver downloaded from Xilinx github?
  - If it is Xilinx github, was there any modification done?
  - What commands were used to run the design? List all commands used. (This will help to reproduce the issue locally if feasible)
  - Is this Internal mode or Descriptor bypass mode?
  - Is C2H Prefetch enabled?
  - How many Queues is the QDMA configured for? Which queues are being used?
  - What are mask registers set to? Review all mask registers. (Note: '1' means the interrupt corresponding to that bit is enabled. Even though the mask bits are set to '0', the errors are still set in the corresponding registers. Only the interrupts are not propagated to the aggregator).
  - Is the issue deterministic? Does it happen always after sending certain number of packets?
  - what is the status of tm_dsc_sts_vld signal? [QDMA Traffic Manager Credit Output Ports Interface]
  - What is the Vivado version being used?
  - **Create a counter outside of the IP to count the packets sent to QDMA IP and compare them with the value reported by the QDMA registers:**
    - QDMA_C2H_STAT_S_AXIS_C2H_ACCEPTED (0xA88) (Number of C2H packet accepted from the user application)

    - QDMA_C2H_STAT_S_AXIS_CMPT_ACCEPTED (0xA8C) (Number of C2H completion packet accepted from the user application)
    - QDMA_C2H_STAT_DESC_RSP_PKT_ACCEPTED (0xA90) (Number of descriptor response packets accepted)

  - Check if s_axis_c2h_ctrl_has_cmpt and s_axis_c2h_cmpt_ctrl_cmpt_type are same or not

- **C2H Hang**
  - Is the C2H transfer hitting 4K boundery in host memory when the hang occurs?
  - What is the status of s_axis_c2h_tready and s_axis_c2h_cmpt_tready signals?
  - Do you have console log before the hang occurs?
  - In the test, is it only C2H traffic or there is H2C traffic as well?
  - How many queues are active when the hang occurs?

- **qdma_hw_error_handler( ): Detected Invalid PIDX update error**
  - **There could be two reasons for this issue:**
    - PIDX update if bigger than the Ring size
    - There are no descriptors in the ring and DMA is trying to fetch more descriptors. If there are descriptors or not, look at the context dump for PIDX and CIDX. If they are at same level, it tells the ring is empty, the driver needs to post more descriptors.

  - **Questions:**
    - Do you see packet drops on AXI side? Check the register dump. Check AXI4-Stream Status Ports (axis_c2h_status_error, axis_c2h_status_drop)
    - Dump register from 0xA80 to 0xBF4.
    - **Do you see if there are any error bit sets in completion entry?**
      - "Detected Completion queue gets full error" if the completion queue is full then there should be any error bit set completion entry.

    - If the packets are being dropped on AXI side, it might be due to lack of descriptors.

  - AXI_ST C2H block needs to be designed with 'QDMA Traffic Manager Credit Output Ports (tm_dsc_sts_*). Confirm this has been done or not. (Note: tm_dsc_sts_* signals must be used in both internal mode or bypass mode in order not to drop any packets.) These ports give how many credit are there for a given Q and so how many packets can the user send to QDMA. If a user sends more packets than the number of credits available then QDMA will drop those packets.
  - Check 0XB10 (QDMA_C2H_STAT_DESC_RSP_DROP_ACCEPTED) register. It gives number of packets that are dropped.
  - What is the Stream packet size (like 64 or 256 bytes, etc)?

  - **For smaller packet sizes like 64bytes all these 4 registers should have same value for a good transfer (0xA88, 0xA8C, 0XA90 and 0xA94)**
    - QDMA_C2H_STAT_S_AXIS_C2H_ACCEPTED (0XA88) - Number of C2H packet accepted from the userapplication.
    - QDMA_C2H_STAT_S_AXIS_CMPT_ACCEPTED (0xA8C) - Number of C2H completion packet accepted from the user application.
    - QDMA_C2H_STAT_DESC_RSP_PKT_ACCEPTED (0xA90) - Number of descriptor response packets accepted.
    - QDMA_C2H_STAT_AXIS_PKG_CMP (0xA94) - The number of C2H packets completed from the C2H DMA write engine.
  - Make sure the user logic is feeding the same QID to C2H and CMPT.

- **QDMA IP not detected in lspci**
  - **Check the following signals in ILA:**
    - user_link_up, axi_aresetn, soft_reset_n, phy_ready

- **axis_c2h_status_drop is asserted**
  - Check if desc_rsp_err is set or not in QDMa_C2H_ERR_STAT
  - **Check the following registers:**
    - QDMA_C2H_DROP_LEN_MISMATCH (0xBB4)
    - QDMA_C2H_DROP_DESC_RSP_LEN (0xBB8)
    - QDMA_C2H_DROP_QID_FIFO_LEN (0xBBC)
    - QDMA_C2H_DROP_PAYLOAD_CNT (0xBB0)
    - Interrupt Decode 2 Register (0xE38)
    - C2H_PACKETS_DROP (0x088)
  - Make sure the descriptor is still available; check tm_dsc_sts_error/tm_dsc_sts_avl
  - Make sure PIDX/CIDX is updated correctly
  - Check s_axis_c2h_tready

12.6 Documents and Debug Collaterals

Description	URL
Download QDMA Driver	https://github.com/Xilinx/dma_ip_drivers
QDMA Documentation	https://xilinx.github.io/dma_ip_drivers/

Queue DMA subsystem for PCI Express (PCIe) - Release Notes and Known Issues for Vivado 2018.1 and newer tool versions	https://www.xilinx.com/support/answers/70927.html
Queue DMA subsystem for PCI Express (PCIe) [Vivado 2020.1] - v3.0 to v4.0 Migration Guide	https://www.xilinx.com/support/answers/75234.html
Queue DMA subsystem for PCI Express (PCIe) - Performance Report	https://www.xilinx.com/support/answers/71453.html
QDMA Debug File System Support	https://xilinx.github.io/dma_ip_drivers/master/QDMA/linux-kernel/html/debugfs.html

12.7 Useful Links

Description	URL
Generating QDMA Subsystem for PCI Express v4.0 Example Design for U200 Board in Vivado 2020.1	https://www.youtube.com/watch?v=eSJc6TWGAFI

Chapter 13

Xilinx PCI Express (PS-PCIe/PL-PCIe) Drivers

13.1 General Debug Checklist

- **PS-PCIe Driver Debug Checklist**
    - The PCI Express Controller Programing Model section in UG1085 summarizes programming of the PCI Express controller for Endpoint and Root Port mode operations. Review that section to make sure programming of the PS-GT Transceiver Interface, IOU for Reset Pin, PCI Express Controller and Bridge initialization has been done correctly.
    - Try with the latest version of PetaLinux.
    - Make sure that the correct version of the BSP is being used for a particular board and silicon revision if the design being run is based on BSP and not on template flow.
    - For specific user design, it is recommended to configure and build a PetaLinux project from the HDF file generated from that design.
    - Confirm that the correct driver and DTS are being used
    - Test with an off the shelf card (for example, NIC as an endpoint)
    - **When debugging a user design, it is helpful to compare register values with the example design for a development board such as the ZCU102. The command below can be used to read a range of registers:**
        - xsct% mrd 0xFD480000 100
    - If the BARs are not assigned, check if the device is advertising a valid class code. If an incorrect class code is provided when configuring Root Port mode in PCW, BAR assignment fails during Linux enumeration.

- **PL-PCIe Driver Debug Checklist**
  - **When setting up a Zynq UltraScale+ MPSoC system for PetaLinux with XDMA PL-PCIe (Bridge Mode) in Root Port configuration, there are a number of settings and options that should be used to experience seamless interoperability of the System, IP, and the PetaLinux Driver (pcie-xdma-pl). (https://www.xilinx.com/support/answers/70854.html) provides details on the following topics.**
    - AXI BAR Translation Registers for correct DTS enumeration
    - Endpoint Devices with Non-Prefetch BARs
    - AXI-Lite Control Bus on AXI Bridge
    - Root Port PCI Express BAR Disable

- **Common Checklist**
  - Review: PS/PL PCIe Drivers - Release Notes (https://www.xilinx.com/support/answers/70702.html)
  - Review Xilinx PCI Express (PS-PCIe/PL-PCIe) Drivers Debug Guide (https://www.xilinx.com/support/answers/71210.html)
  - **Unlike x86 platforms where the BIOS enables Memory space (MemEn) in the device configuration space, embedded platforms do not enable MemEn bits during enumeration. An Endpoint driver with relevant PCIe API's is required for this. To check BAR access via devmem, you should set MemEn via the "setpci" command as shown below. setpci is a utility for querying and configuring PCI devices.**
    - setpci -s 01:00.1 COMMAND=0x7
  - If an endpoint is connected and if the link is established successfully, the log file reports 'Link is UP'. Confirm that the link is UP before proceeding with further investigation.
  - Endpoint functionality can be checked with endpoint drivers. The respective endpoint driver must be enabled in the kernel build.
  - **If the device is not deteced in lspci, try by rescanning as shown below.**
    - echo 1 > /sys/bus/pci/devices/0000:00:00.0/remove
    - echo 1 > /sys/bus/pci/rescan

13.2 Issues/Debug Tips/Questions

- **Mellanox PCIe NIC card is connected to the PCIe slot on ZCU102 board. During Linux boot up, the Mellanox card ("Connect4-Lx") is recognized and associated with the mlx5 driver, which starts its probe process. However, the probe encounters problem when allocating interrupts and fails. Below is an excerpt from "dmesg", showing the Mellanox card being recognized during boot by its mlx5 driver and probed:**
  - [4.940428] pci 0000:01:00.0: enabling device (0000 -> 0002)
  - [4.946132] pci 0000:01:00.0: mellanox_check_broken_intx_masking+0x0/0x1b8 took 11111 usecs
  - [4.954572] mlx5_core 0000:01:00.0: assign IRQ: got 51
  - [4.954852] mlx5_core 0000:01:00.0: enabling bus mastering
  - [4.954929] mlx5_core 0000:01:00.0: firmware version: 14.26.1040
  - [4.960966] mlx5_core 0000:01:00.0: 16.000 Gb/s available PCIe bandwidth, limited by 5 GT/s x4 link at 0000:00:00.0 (capable of 63.008 Gb/s with 8 GT/s x8 link)
  - [5.278293] mlx5_core 0000:01:00.0: mlx5_load:1068:(pid 26): Failed to alloc IRQs
  - [6.003727] mlx5_core 0000:01:00.0: init_one:1349:(pid 26): mlx5_load_one failed with error code -28
  - [6.012869] mlx5_core 0000:01:00.0: disabling bus mastering
  - [6.012981] mlx5_core: probe of 0000:01:00.0 failed with error -28
  - [6.019199] pci 0000:01:00.1: enabling device (0000 -> 0002)
  - [6.025095] mlx5_core 0000:01:00.1: assign IRQ: got 52
  - [6.025685] mlx5_core 0000:01:00.1: enabling bus mastering
  - [6.025765] mlx5_core 0000:01:00.1: firmware version: 14.26.1040
  - [6.031808] mlx5_core 0000:01:00.1: 16.000 Gb/s available PCIe bandwidth, limited by 5 GT/s x4 link at 0000:00:00.0 (capable of 63.008 Gb/s with 8 GT/s x8 link)
  - [6.349033] mlx5_core 0000:01:00.1: mlx5_load:1068:(pid 26): Failed to alloc IRQs
  - [7.072889] mlx5_core 0000:01:00.1: init_one:1349:(pid 26): mlx5_load_one failed with error code -28
  - [7.082038] mlx5_core 0000:01:00.1: disabling bus mastering
  - [7.082160] mlx5_core: probe of 0000:01:00.1 failed with error -28
  - **Answer:**
    - PS-PCIe in Root Port mode does not support receiving MSI-X interrupts.

- **Can PS-PCIe® on ZU+ in Endpoint mode meet 100ms requirement?**
  - When using PS-PCIe® on ZU+ in Endpoint mode, running FSBL is enough to set up the block for endpoint mode operation. FSBL should be able to program the PS/PS-PCIe® and GTR within 100 ms. However, this doesn't include PL-bitstream programming as including that would make this greater than 100 ms. (Ref: UG1137)

- **Link Down: Endpoint is not detected (PS-PCIe)**
  - Ensure the Endpoint card has fit in properly on the PCIe slot of the Rootport.
  - **Check the link status by doing 'devmem 0xfd480238' in the Linux prompt.**
    - Value should be 0x3 if the link is up https://www.xilinx.com/html_docs/registers/ug1087/ug1087-zynq-ultrascale-registers.html#pcie_attrib___attr_101.html
  - Read LTSSM state from 0xfd480228. 8 down to 3 gives the LTSSM coding. For LTSSM coding values, see (PG054).
  - **If the link is found to be up via reading PCIe Status register but the device is not seen in 'lspci':**
    - Check if the endpoint card was programmed before Rootport booted
    - **Do a rescan of the PCIe bus by using the following command on ZCU102 Linux prompt**
      - $ echo 1 > /sys/bus/pci/devices/0000:01:00.0/remove
      - $ echo 1 > /sys/bus/pci/rescan
  - **If the rescan did not help:**
    - Try with the latest FSBL.
    - Test on some other platform (x86) just to confirm the EP card has no issues.
  - **When EP is reset without resetting RP, the LTSSM on RP goes through either the Disabled or hot-reset states which are known to have some chances of link failures with errors.**
    - Issue Rootport Reset (0xFD1A0100 bit 17 pcie_ctrl_reset only, this is like PERST) https://www.xilinx.com/html_docs/registers/ug1087/ug1087-zynq-ultrascaleregisters.html#crf_apb___rst_fpd_top.html
- **nwl-pcie fd0e0000.pcie: Unsupported request Detected (PS-PCIe)**
  - **If the following error is reported when trying to initiate DMA from a third party GPU, make sure to enable the bus mastering bit in the root port as well as endpoint (EP).**
    - [3130.483591] nwl-pcie fd0e0000.pcie: Unsupported request Detected
    - [3130.490863] nwl-pcie fd0e0000.pcie: Non-Fatal Error Detected
    - **The Bus mastering bit can be set by using the following commands:**
      - setpci -s 00:00.0 COMMAND=0x4
      - setpci -s 01:00.0 COMMAND=0x7
      - setpci -s 01:00.1 COMMAND=0x7

- **Note:**
  - confirm the bus structure before executing the above command.
  - In the kernel EP driver flow, when EP driver invokes "pci_enable_device_mem" API, the bus mastering bit will be set. If there is no driver, it has to be done manually. On an x86 machine this bit is set by the BIOS, on ARM system this needs to be enabled using the above API.

13.3 Documents and Debug Collaterals

Description	URL
Xilinx PCIe Root Port Driver Landing Page	https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/85983409/Xilinx+PCIe+Root+Port
Zynq UltraScale+ MPSoC - PS/PL PCIe Drivers - Release Notes	https://www.xilinx.com/support/answers/70702.html
Xilinx PCI Express (PS-PCIe/PL-PCIe) Drivers Debug Guide	https://www.xilinx.com/support/answers/71210.html
PetaLinux Image Generation and System Example Design with ZCU102 PS-PCIe as Root Complex and ZC706 as Endpoint	https://www.xilinx.com/support/answers/71493.html

PetaLinux Image Generation and System Example Design with ZC706 as Root Complex and KC705 as Endpoint	https://www.xilinx.com/support/answers/71494.html

13.4 Useful Links

Description	URL
AXI PCI Express MIG Subsystem Built in IPI	https://www.xilinx.com/video/fpga/axi-pci-express-mig-subsystem-built-in-ipi.html