PCIE4 and PCIE4C AXI4-Lite Bridge IP Core

XAPP1201 (v1.2) June 7, 2024

Summary

This application note describes the use of a bridge from the Completer Streaming Interfaces of the Gen3/Gen4 Integrated Block for PCI Express® IP core to an AXI4-Lite master interface. The reference design provides a packaged IP core which connects to the Integrated Block for PCI Express IP core in AMD Vivado™ IP integrator, while using less than 300 LUTs. The AXI4-Lite Master port connects to peripherals designed with an AXI4 slave interface.

Download the reference design files for this application note from the AMD website. For detailed information about the design files, see Reference Design.

Introduction

The AMD Artix™ UltraScale+™ and AMD Kintex™ UltraScale+™ architectures contain the PCIE4 or PCIE4C integrated hard blocks for Gen3 and Gen4 PCI Express. Although these hard blocks are designed for high-performance systems, it is common for an endpoint to receive only a one dword (DW) request from the host. The one DW request can set up a DMA engine or be used to monitor and change peripheral registers in an AXI4-based system.

Because the integrated PCI Express IP core provides streaming interfaces, a bridge to AXI4 is commonly used to access the control-plane peripherals on a AXI4-Lite interconnect. Any incoming one DW request can operate with the Completer reQuest (CQ) and the Completer Completion (CC) interfaces of the integrated PCI Express IP. The bridge only uses the CC and CQ interfaces to bridge to an AXI4-Lite interface. For high performance applications, the endpoint becomes a master and makes multiple DW requests upstream. For the endpoint to master, the Requester reQuest (RQ) and the Requester Completion (RC) interfaces are used. As shown in the following figure, the bridge does not use either of the Requester interfaces allowing you to continue to use these high-performance ports for bus mastering applications.

Figure 1: **IP Integrator Subsystem**

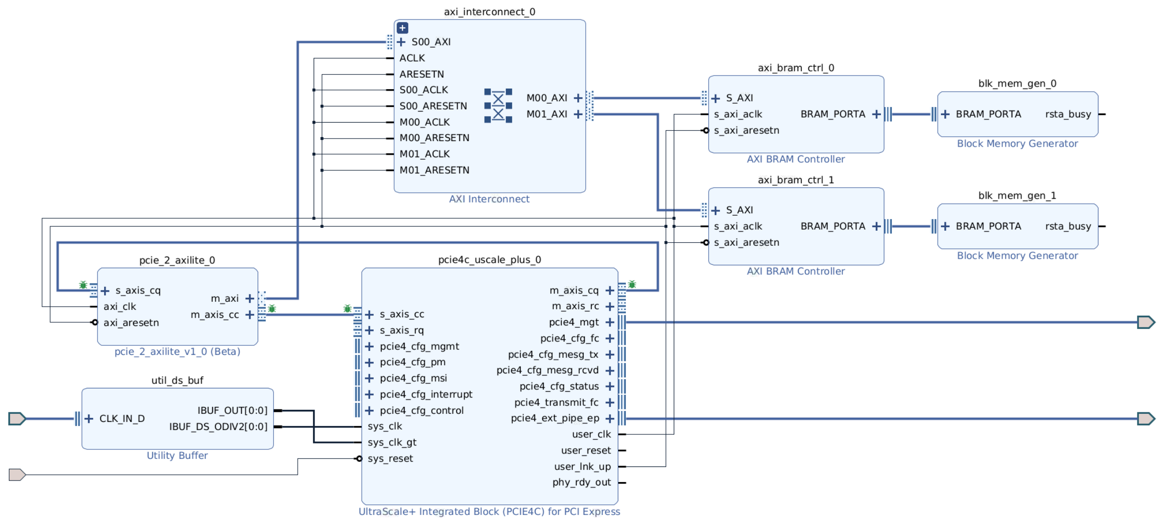


Figure 1 illustrates an example system connecting local block RAM as memory mapped storage using the pcie_2_axilite IP. There are many other AXI peripherals available from the Vivado IP catalog that could similarly be connected, such as:

- AXI Quad SPI
- AXI UART Lite
- AXI Timer
- AXI IIC Bus Interface
- AXI Ethernet Lite MAC
- AXI GPIO
- AXI EMC

Features

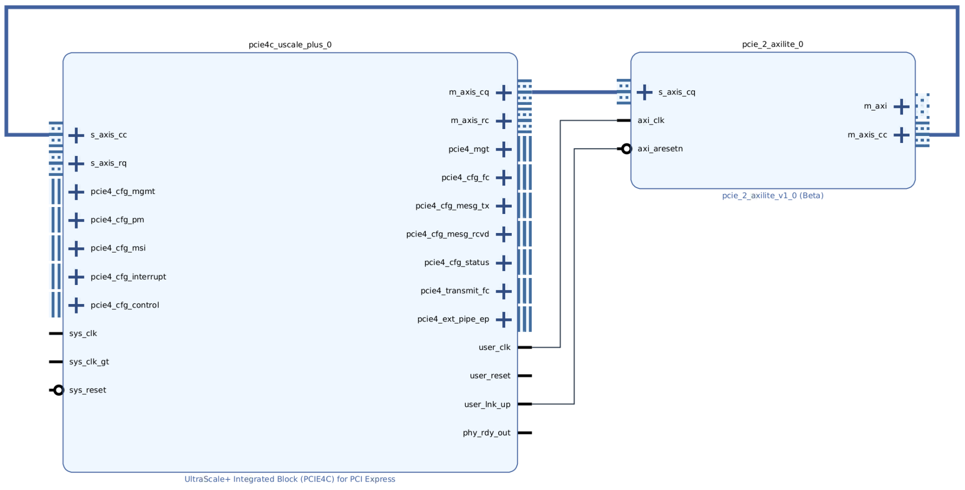
The pcie_2_axilite bridge supports the following features:

- One DW memory read and memory write requests
- Up to six base address registers (BARs)
- 32-bit and 64-bit BARs
- Address translation from transaction layer packet (TLP) header address to AXI4 addressing
- All data width configurations of the Completer interface, under any lane width and generation speed (Gen1, Gen2, Gen3, or Gen4) configuration
- Vivado IP integrator support
- Source provided (Verilog only)
- Data Aligned Mode Only (Not Address Aligned mode)

Hardware Description

The bridge only uses the CC and CQ interfaces of the integrated PCI Express IP core. The m_axis_cq interface of the integrated hardblock connects directly to the s_axis_cq interface of the bridge, while the s_axis_cc interface of the integrated hardblock connects the m_axis_cc interface of the bridge. The user_clk output of the integrated hardblock IP core is synchronous to the CC and CQ interfaces and serves as the source clock for the bridge. The axi_aresetn is an asynchronous active-Low reset of the bridge, and it holds the bridge in a reset state where packets cannot pass through the bridge. It is common to use the user_lnk_up output of the integrated hardblock as a reset to the bridge; however, it is possible to choose another signal as a reset. The following figure shows the CQ and the CC interface connected to the bridge along with the corresponding clock and reset.

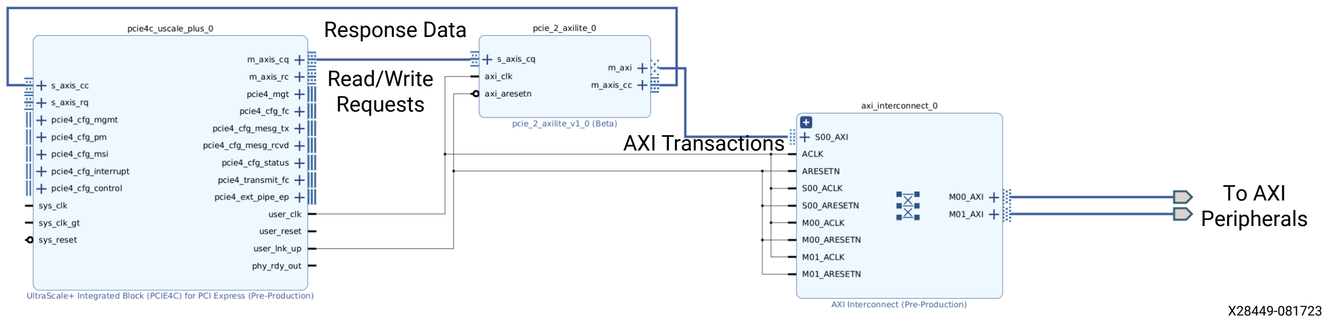
Figure 2: **Connecting the Completer Interfaces**



The CQ interface (m_axis_cq) provides memory read and write requests from the host. The bridge decodes the CQ interface requests and translates them to a master AXI4-Lite interface. Memory write requests from the host are translated to AXI4 Write Address Channel and AXI4 Write Data Channel transactions. The AXI4 Write Response Channel is connected, but not used with the bridge. The ready signal of the AXI4 Write Response Channel is asserted, but the data from the AXI4 Write Response Channel is ignored.

When the PCI Express host requests a memory read, a completion with data TLP is expected to return. When a memory read is requested through the CQ interface, the bridge first converts the read request to an AXI4 Read Address Channel transaction. Then the AXI4 slave responds with the data on the AXI4 Read Data Channel. The bridge accepts the data and creates a completion TLP on the CC interface (m_axis_cc) with the payload from the AXI4 Read Data Channel. The following figure provides a conceptual visualization of the flow of the transactions.

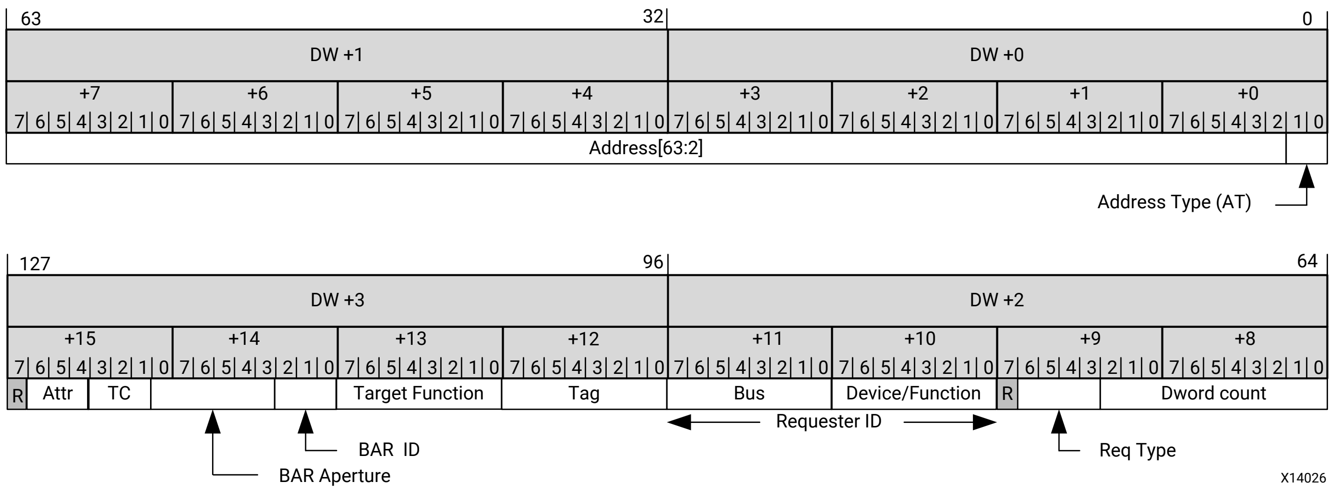
Figure 3: **Interface Connections with Corresponding Traffic Type**



Address Translation

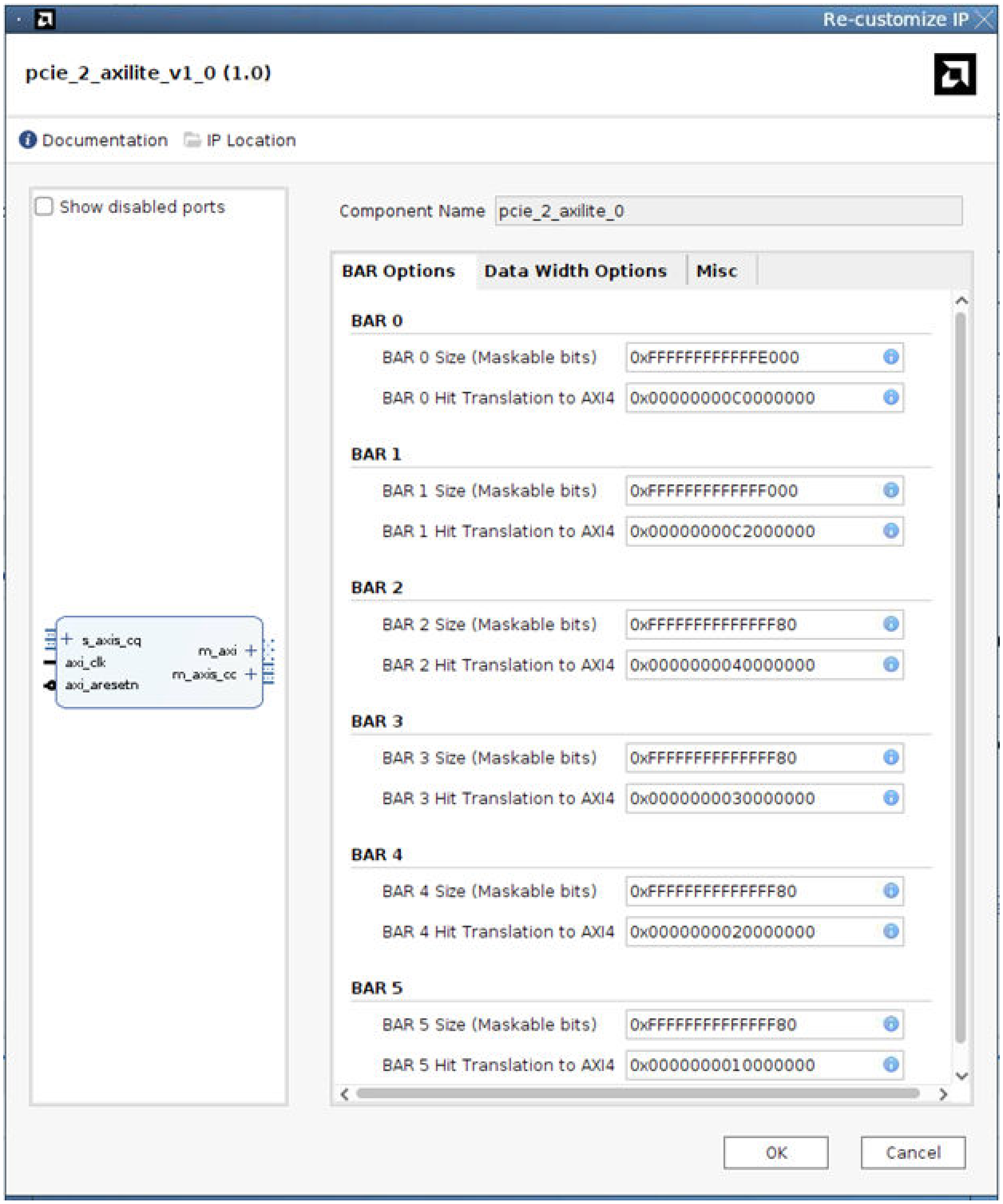
The address from the TLP is provided in the CQ descriptor from the integrated PCI Express IP. The CQ descriptor is provided in the following figure. DW+0 and DW+1 provide the address from the TLP.

Figure 4: **CQ Descriptor**



The address provided from the descriptor comes directly from the PCI Express TLP. Depending on the BAR hit, the address translates to different AXI4 addresses. The bridge IP core is packaged with options for the translation to the AXI4 space within the BAR Options tab as shown in the following figure.

Figure 5: **BAR Translation Options**



Component Name pcie_2_axilite_0

BAR 0

BAR 0 Size (Maskable bits)	0xFFFFFFFFFFFFE000
BAR 0 Hit Translation to AXI4	0x00000000C0000000

BAR 1

BAR 1 Size (Maskable bits)	0xFFFFFFFFFFFFF000
BAR 1 Hit Translation to AXI4	0x00000000C2000000

BAR 2

BAR 2 Size (Maskable bits)	0xFFFFFFFFFFFFFF80
BAR 2 Hit Translation to AXI4	0x0000000040000000

BAR 3

BAR 3 Size (Maskable bits)	0xFFFFFFFFFFFFFF80
BAR 3 Hit Translation to AXI4	0x0000000030000000

BAR 4

BAR 4 Size (Maskable bits)	0xFFFFFFFFFFFFFF80
BAR 4 Hit Translation to AXI4	0x0000000020000000

BAR 5

BAR 5 Size (Maskable bits)	0xFFFFFFFFFFFFFF80
BAR 5 Hit Translation to AXI4	0x0000000010000000

- **BAR # Size:** Defines the size of the address aperture. This field must be (manually) configured in the bridge IP to match the corresponding BAR size(s) configured in the integrated PCIe Block IP. In the bridge's address translation logic, it is used to mask the portion of the address field that is "ignored" from the PCIe TLP address when passing into the AXI4 address space. Table 1 shows the valid values allowed in BAR # Size and the resulting aperture size.

- **BAR # Hit Translation to AXI4:** Defines the AXI4 address offset that is added to the post-masked PCIe TLP address during address translation in the bridge IP. In other words, this field is the post-translation AXI4 base address that corresponds to the respective PCIe BAR # as configured in the integrated PCIe Block IP. It is required that the zeroed bits of BAR # Size are also zeroed in BAR # Hit Translation to AXI4.

Table 1: **Example of Maskable Bits and BAR Size**

BAR # Size (Maskable Bits)	Corresponding BAR Aperture Size
0xFFFFFFFFFFFFFF80	128 Bytes (minimum)
0xFFFFFFFFFFFFFF00	256 Bytes
0xFFFFFFFFFFFFFE00	512 Bytes
0xFFFFFFFFFFFFFC00	1 KB
...	...
0xFFFFFFF800000000	32 GB
0xFFFFFFF000000000	64 GB
0xFFFFFFE000000000	128 GB
0xFFFFFFC000000000	256 GB (maximum)

The following table shows two examples of how the BAR # Size and BAR # Hit Translation work. Example #1 shows the bridge set up with 1 KB of addressable space assigned to a particular BAR as configured from the BAR # Size maskable bits. The host has enumerated this BAR to address x000000000C000000. The host has requested from address offset four of the BAR, which comes out to be a TLP address of x000000000C000004. After masking the upper bits from the descriptor address, the resulting offset address is simply x4. To translate this offset into the AXI domain, add the BAR # Hit Translation value to the offset address and the resulting address in the AXI domain is x0000000080000004.

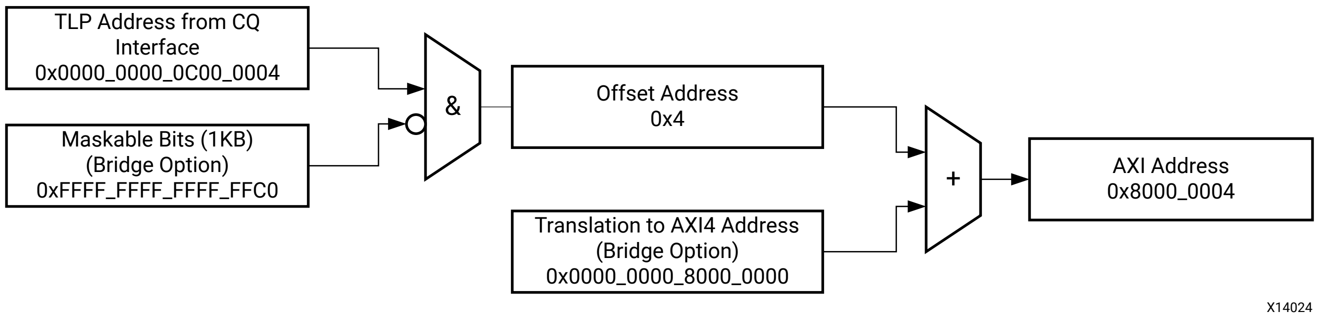
Example #2 of the following table shows another resulting AXI address from a descriptor address and how the translation is calculated.

Table 2: **Example of Address Translation**

	Example #1	Example #2
BAR Enumerated Address (Assigned from Host)	x000000000C000000	x0000000000008000
Address in Descriptor (Request from Host)	x000000000C000004	x00000000000080CC
Maskable Bits (Bridge Option)	xFFFFFFFFFFFFFC00	xFFFFFFFFFFFFF000
Size of BAR	1 KB	4 KB
Translation to AXI (Bridge option)	x0000000080000000	x0000000040000000
Resulting AXI Address	x0000000080000004	x00000000400000CC

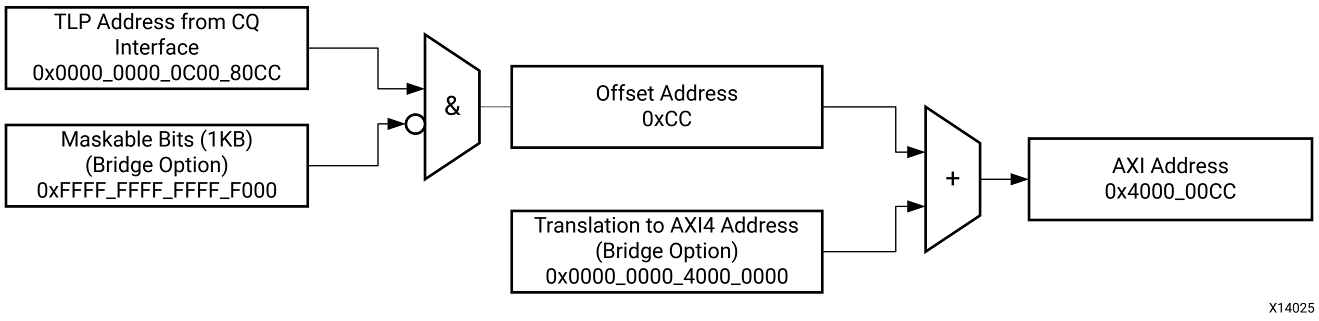
The following figure provides a flow chart representation of Example #1 from Table 2.

Figure 6: **Example #1 as a Flow Chart**



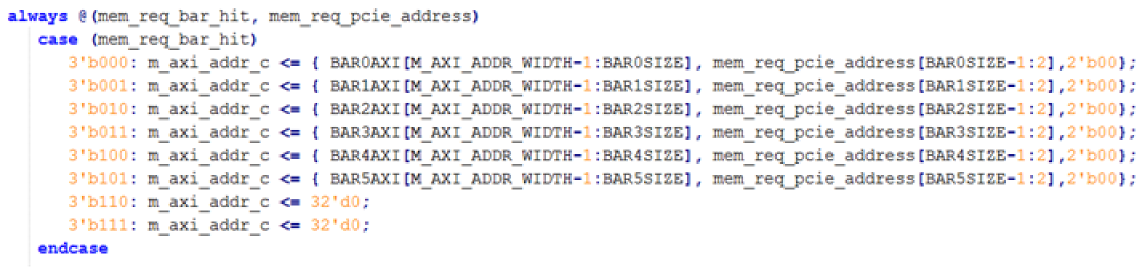
The following figure provides a flow chart representation of Example #2 from Table 2.

Figure 7: **Example #2 as a Flow Chart**



The RTL code performing the translation to AXI4 is shown in the following figure. The BAR#SIZE value is determined by the least significant High bit set in the BAR # Size option configured in the bridge IP. For example, xFFFFFFFFFFFFFC00 would yield a BAR#SIZE of 10 because the 11th bit is the least significant High bit. BAR#AXI is derived from the BAR # Hit Translation to AXI4 option configured in the bridge IP.

Figure 8: **BAR Translation RTL**



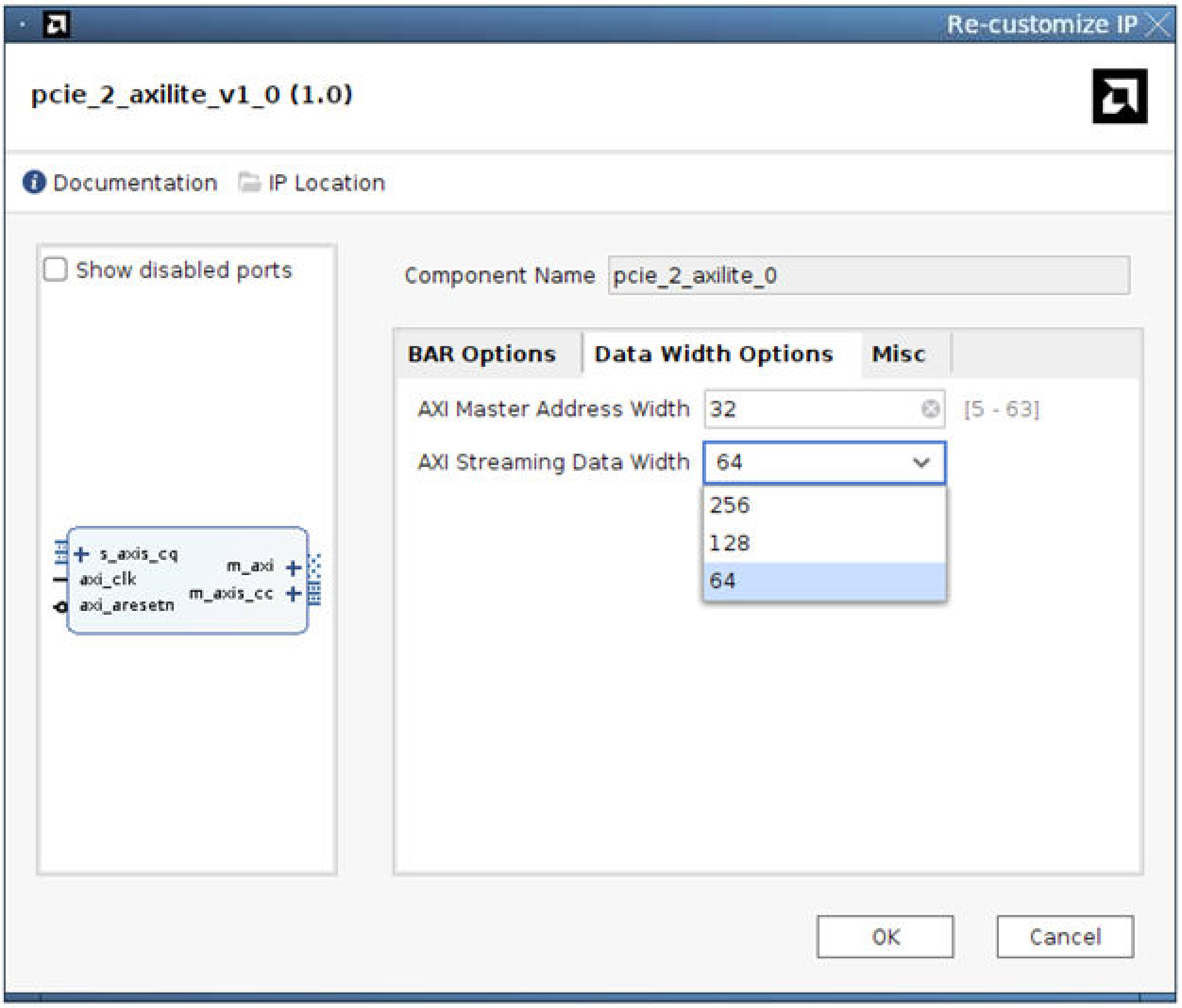
```
always @(mem_req_bar_hit, mem_req_pcie_address)
  case (mem_req_bar_hit)
    3'b000: m_axi_addr_c <= { BAR0AXI[M_AXI_ADDR_WIDTH-1:BAR0SIZE], mem_req_pcie_address[BAR0SIZE-1:2],2'b00};
    3'b001: m_axi_addr_c <= { BAR1AXI[M_AXI_ADDR_WIDTH-1:BAR1SIZE], mem_req_pcie_address[BAR1SIZE-1:2],2'b00};
    3'b010: m_axi_addr_c <= { BAR2AXI[M_AXI_ADDR_WIDTH-1:BAR2SIZE], mem_req_pcie_address[BAR2SIZE-1:2],2'b00};
    3'b011: m_axi_addr_c <= { BAR3AXI[M_AXI_ADDR_WIDTH-1:BAR3SIZE], mem_req_pcie_address[BAR3SIZE-1:2],2'b00};
    3'b100: m_axi_addr_c <= { BAR4AXI[M_AXI_ADDR_WIDTH-1:BAR4SIZE], mem_req_pcie_address[BAR4SIZE-1:2],2'b00};
    3'b101: m_axi_addr_c <= { BAR5AXI[M_AXI_ADDR_WIDTH-1:BAR5SIZE], mem_req_pcie_address[BAR5SIZE-1:2],2'b00};
    3'b110: m_axi_addr_c <= 32'd0;
    3'b111: m_axi_addr_c <= 32'd0;
  endcase
```

Data Width Options Configuration Tab

- **Master AXI Address Width:** Determines the total addressable space the master AXI interface can access. This is independent to the PCIe BAR sizes.
- **AXI Streaming Data Width:**

  Determines the data width of both completer interfaces (CC and CQ). This value must match the width selected in the Integrated PCIe Block IP.

Figure 9: **Data and Address Width Options**



Component Name: pcie_2_axilite_0

AXI Master Address Width: 32 [5 - 63]

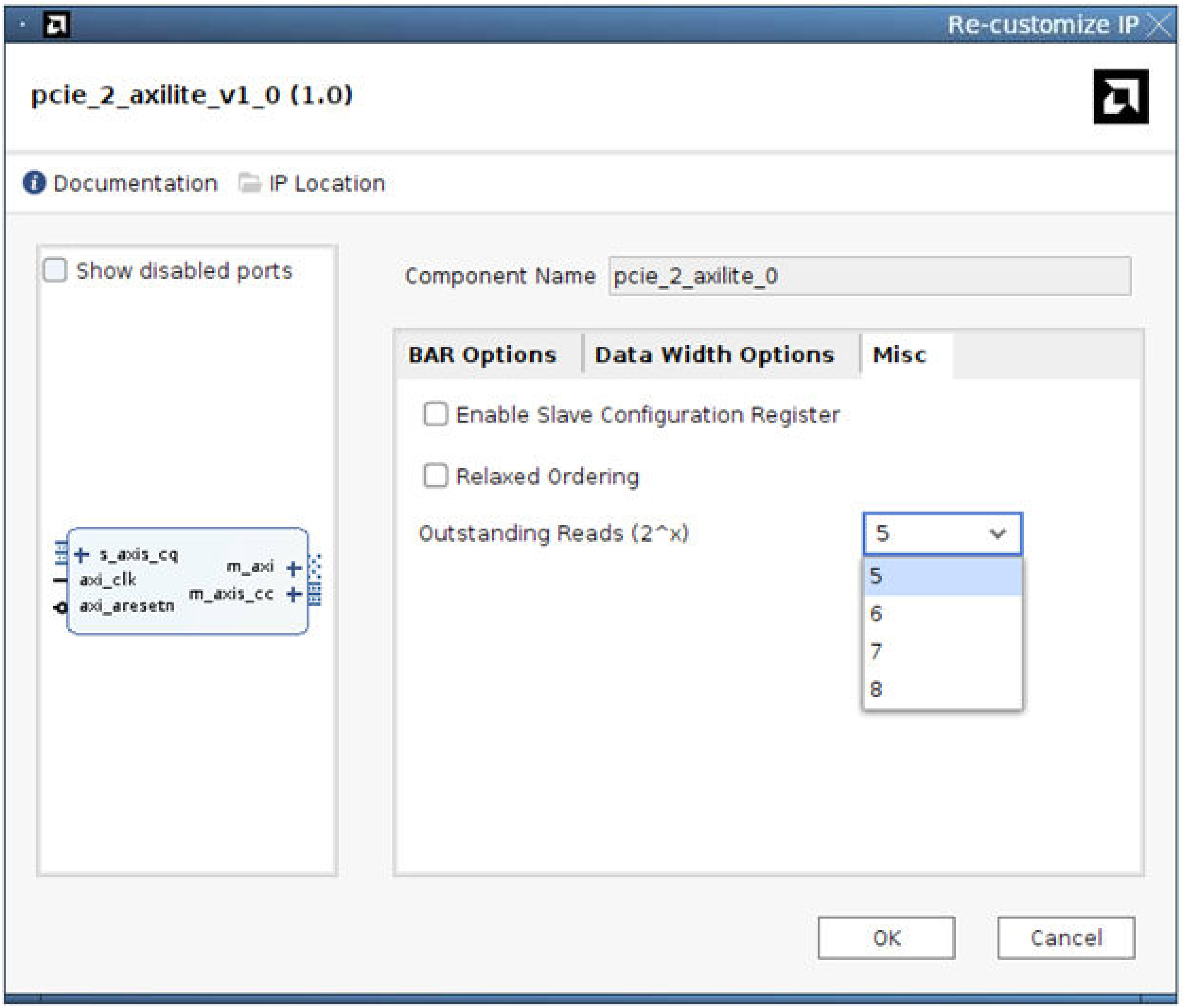
AXI Streaming Data Width: 64

Miscellaneous Options

- **Enable Slave Configuration Register:** When checked, enables an AXI-S interface into the bridge IP. It is intended as a convenient starting point to quickly enhance the bridge IP with custom features. Implementing such features requires editing the RTL in the pcie2axilite_bridge\rtl directory. For example, one might want to implement:
  - Dynamic BAR translation
  - Error conditions
  - Debug register
- **Relaxed Ordering:** Allows for read and write TLP requests to be processed in parallel and potentially out-of-order. This might violate the PCI Express specification. It is not recommended to use this option without a thorough analysis.

- **Outstanding Reads (2^x):** The bridge buffers multiple outstanding read request transactions from the PCI Express block before the data might get returned on the AXI interface. This option configures the maximum outstanding request to buffer from 32 (2^5) up to 256 (2^8) as highlighted in the following figure.

Figure 10: **Miscellaneous Options**



Component Name pcie_2_axilite_0

Enable Slave Configuration Register

Relaxed Ordering

Outstanding Reads (2^x) 5

Reference Design

Download the reference design files for this application note from the AMD website.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 3: **Reference Design Matrix**

Parameter	Description
General	
Developer name	AMD

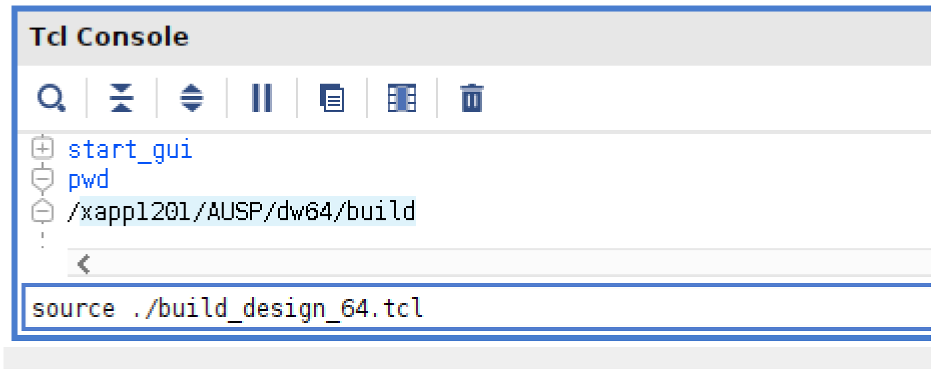
Table 3: **Reference Design Matrix** *(cont'd)*

Parameter	Description
Target devices	Artix UltraScale+ and Kintex UltraScale+ devices
Source code provided?	Y
Source code format (if provided)	Verilog
Design uses code or IP from existing reference design, application note, third party or Vivado software? If yes, list.	Yes. Vivado Catalog IP for PCIe and AXI interconnect.
Simulation	
Functional simulation performed	Y
Timing simulation performed?	N
Test bench provided for functional and timing simulation?	Y
Test bench format	Verilog
Simulator software and version	Vivado simulator 2022.2 to 2023.2
SPICE/IBIS simulations	N
Implementation	
Synthesis software tools/versions used	Vivado synthesis 2022.2 to 2023.2
Implementation software tool(s) and version	Vivado implementation
Static timing analysis performed?	Y
Hardware Verification	
Hardware verified?	N
Platform used for verification	N/A

Implementing the Reference Design

The reference design provides an example of how the bridge is configured for 64-bit usage. Within the Vivado IDE, source the `build_design_64.tcl` located in the `AUSP/dw64/build` directory.

Figure 11: **Running the Example Design with Vivado IDE**



```
start_gui
pwd
/xapp1201/AUSP/dw64/build
```

```
source ./build_design_64.tcl
```

By sourcing the Tcl script, a Vivado project is created and populated with the reference design and simulation sources. The reference design targets an Artix UltraScale+ device, but does not use a specific development board. Therefore, it is not intended to be implemented through bit-file generation without first tailoring the board-level constraints to the user's specific hardware definition.

Simulation

A reference simulation is provided to help show the translation from TLPs to AXI4 transactions. The example simulation test bench exercises the CQ interface with a memory write and memory read request. The resulting AXI transaction masters onto an AXI4 BRAM module to respond to the requests.

To run the simulation, after running the build Tcl script, click **Run Simulation** in the Flow Navigator. After about 200 µs of simulation time, the sample test completes with a "Test Completed Successfully" message in the simulation log.

Inside of `dw64/source/sim/sample_tests.vh`, the default test is `pio_writeReadBack_test0`. This routine exercises the example design's block RAM memories after configuring the BAR. This file can be used as a reference to further extend the test bench coverage for a specific application.

Resource Utilization

The following table describes the resource utilization for an Artix UltraScale+ AU10P device.

Table 4: **Resource Utilization**

TDATA Width	LUTs	Flip-Flops	RAMs
x64	170	277	0

File Description

The following table describes the directory structure of the reference design.

Table 5: **Reference Design Files**

Directory and Files	Description
`archive/xapp1201_virtex7.zip`	This ZIP contains the 2014 version of the example design that works with PCIE3 in Virtex 7 FPGAs and similar devices.
`AUSP/dw64`	Example design directory demonstrating 64-bit data bus widths
`AUSP/dw64/build/build_design_64.tcl`	Vivado tools script to create the example design project
`AUSP/dw64/source/constraints/top.xdc`	Example design constraints
`AUSP/dw64/source/ipi/ipi_design_64.tcl`	IP integrator example design build script
`AUSP/dw64/source/sim/*`	Verilog files for example design simulation and test bench
`pcie2axilite_bridge/`	IP repository files for the pcie2axilite IP
`pcie2axilite_bridge/component.xml`	Vivado IP packaging file
`pcie2axilite_bridge/xgui/pcie_2_axilite_v1_0.tcl`	Vivado packager Tcl Files for the GUI
`pcie2axilite_bridge/rtl/*`	Complete Verilog RTL source for the pcie2axilite IP used to build the IP repository

Conclusion

PCI Express endpoints generally receive only one DW request from a host. The request operates on the CQ and the CC interface of the Integrated Block for PCI Express. For high-performance applications, the endpoint becomes a master and makes the requests upstream. For the endpoint to master, the RQ and the RC are used. The bridge to AXI4-Lite does not use the Requester interfaces allowing you to continue to use these high-performance ports. It is recommended to use a bridge to AXI4-Lite with the completer interfaces because the host generally has one DW request. Leveraging the packaged IP core in this application note enables you to quickly accept incoming requests from a host and translate them to AXI4 transactions.

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/07/2024 Version 1.2	
General updates	• Added PCIe Gen4 support. • Added pcie_2_axilite IP core.
09/08/2015 Version 1.1.1	
General updates	Revised RTL acronym.
09/02/2014 Version 1.1	
General updates	Updated Title.
01/23/2014 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright