XAPP1201 (v1.1.1) September 8, 2015

Virtex-7 (XT and HT) and UltraScale FPGAs Gen3 Integrated Block for PCI Express to AXI4-Lite Bridge

Author: Luis Bielich

Summary

This application note describes the use of a bridge from the Completer Streaming Interfaces of the Gen3 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 Vivado® IP integrator, while utilizing less than 300 LUTs. The AXI4-Lite Master port connects to peripherals designed with an AXI4 slave interface.

Introduction

The Virtex®-7 (XT and HT) device family and UltraScale™ architecture contain Gen3 integrated hard blocks for PCI Express. Although these hard blocks are designed for high-performance systems, it is common for an endpoint to receive only one dword (DW) request from the host. The one DW request can set up a DMA engine or can be used to monitor and change peripheral registers in an AXI4-based system.

Because the Integrated Block for 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 Block for PCI Express. 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. 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 shows an entire system where the Completer interfaces are used for access to block RAM while the Requester interfaces remain open for bus mastering applications.

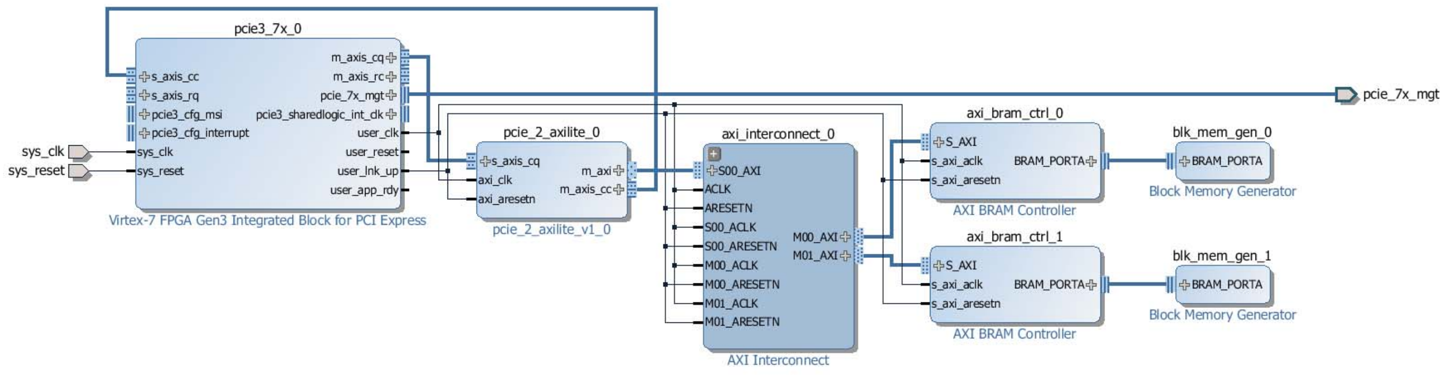


Figure 1: **IP Integrator Subsystem**

Figure 1 illustrates a system connecting local block RAM as memory mapped storage. There are many other AXI peripherals available from the Vivado IP catalog that you can also use to connect. The following list provides several examples of potential peripherals that connect similar to the block RAM example in Figure 1.

- AXI Quad SPI
- AXI UART Lite

- AXI Timer
- AXI IIC Bus Interface
- AXI Ethernet Lite MAC
- AXI GPIO
- AXI EMC

This application note provides the register-transfer language (RTL) to convert the CQ and the CC interfaces into an AXI-Lite interface. The RTL is packaged as an IP core so that it can be connected within IP integrator or instantiated as an IP module. The RTL is not encrypted which allows for the bridge to be customized, if desired.

Features

The bridge is packaged as a Vivado IP core supporting 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, or gen3) 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 hardblock for 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 the clock synchronous to the CC and CQ interfaces and serves as the clock driving 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. Figure 2 shows the CQ and the CC interface connected to the bridge along with the corresponding clock and reset within IP integrator.

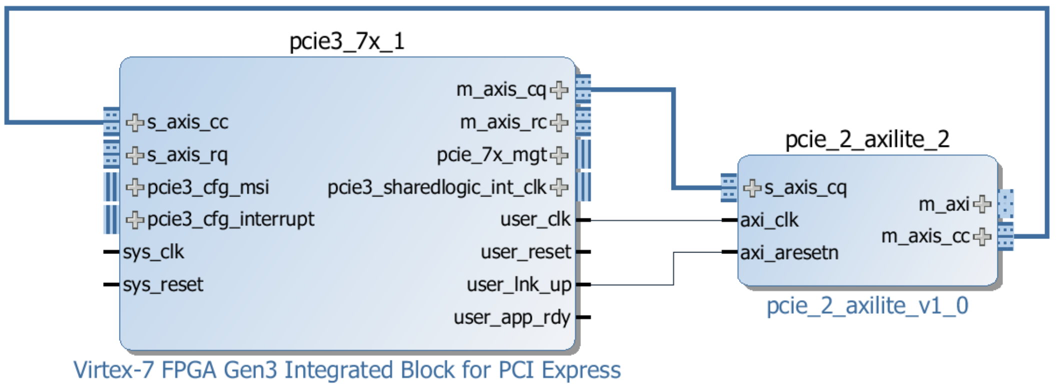


Figure 2: **Connecting the Completer Interfaces**

The CQ interface (`m_axis_cq`) provides memory read and memory write requests from the host. The bridge decodes memory read and memory write requests from the CQ interface, and then translates the requests to the master on an AXI4-Lite system. Memory write requests from the host translate 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.

For PCI Express, when a 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 a AXI4 Read Address Channel transaction. Then the AXI4 slave responds with the data on the AXI4 Read Data Channel. The bridge accepts the data on the AXI4 Read Data Channel and creates a completion TLP on the CC interface (`m_axis_cc`) with the payload from the AXI4 Read Data Channel. Figure 3 provides a conceptual visualization of the flow of the transactions.

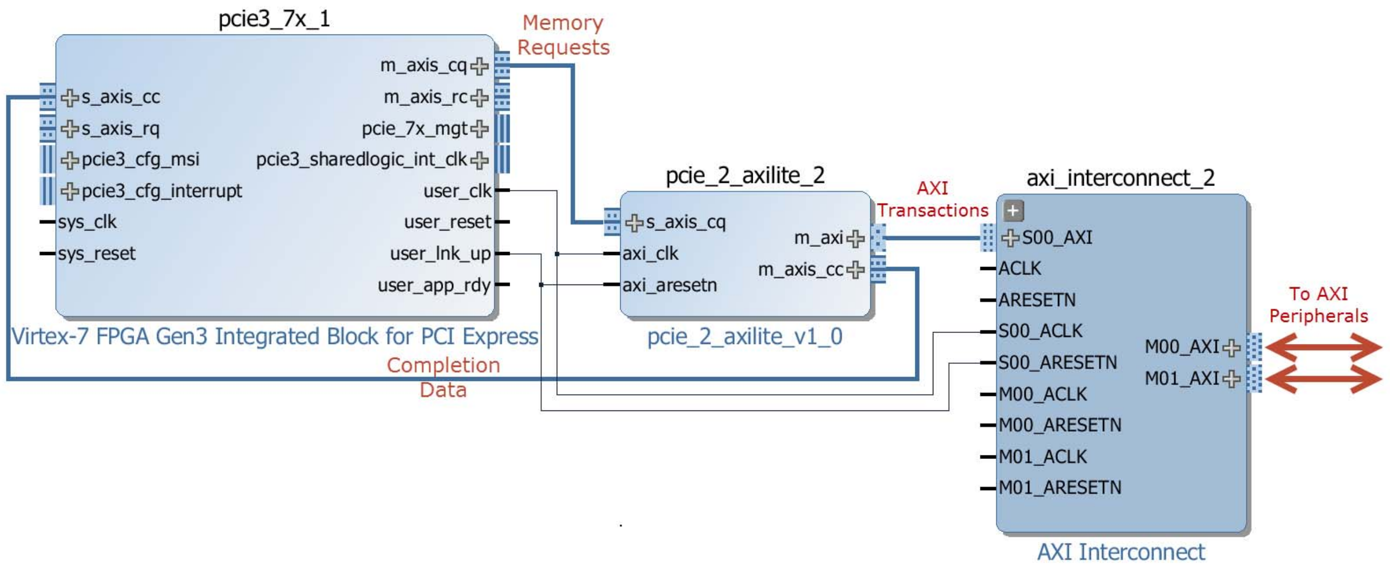


Figure 3: **Interface Connections with Corresponding Traffic Type**

Address Translation

The address from the TLP is provided in the CQ descriptor from the Integrated Block IP. The CQ descriptor is provided in Figure 4. Notice DW+0 and DW+1 provide the address from the TLP.

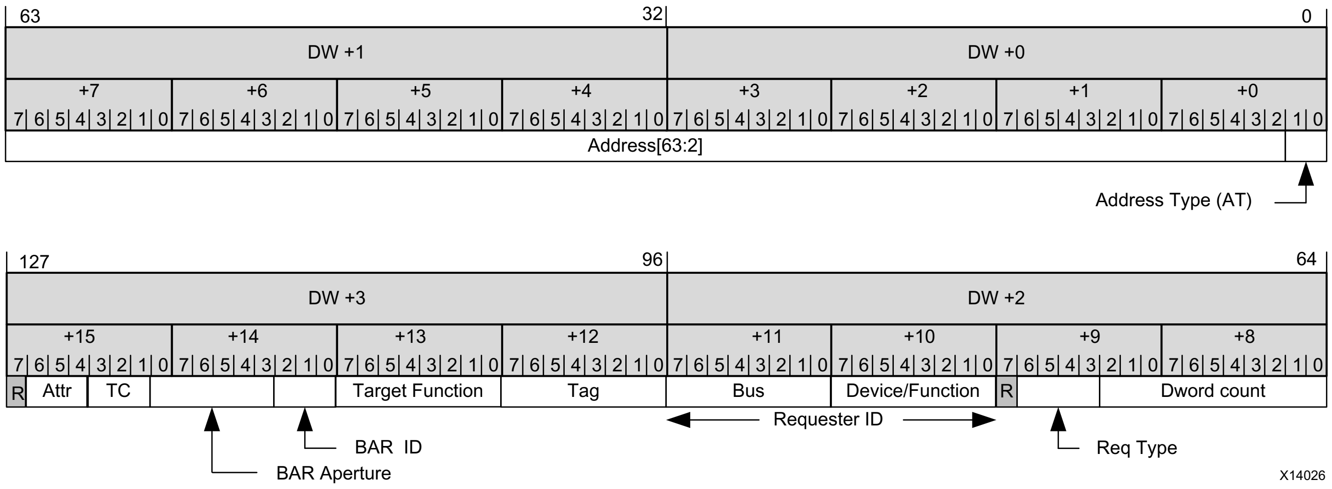
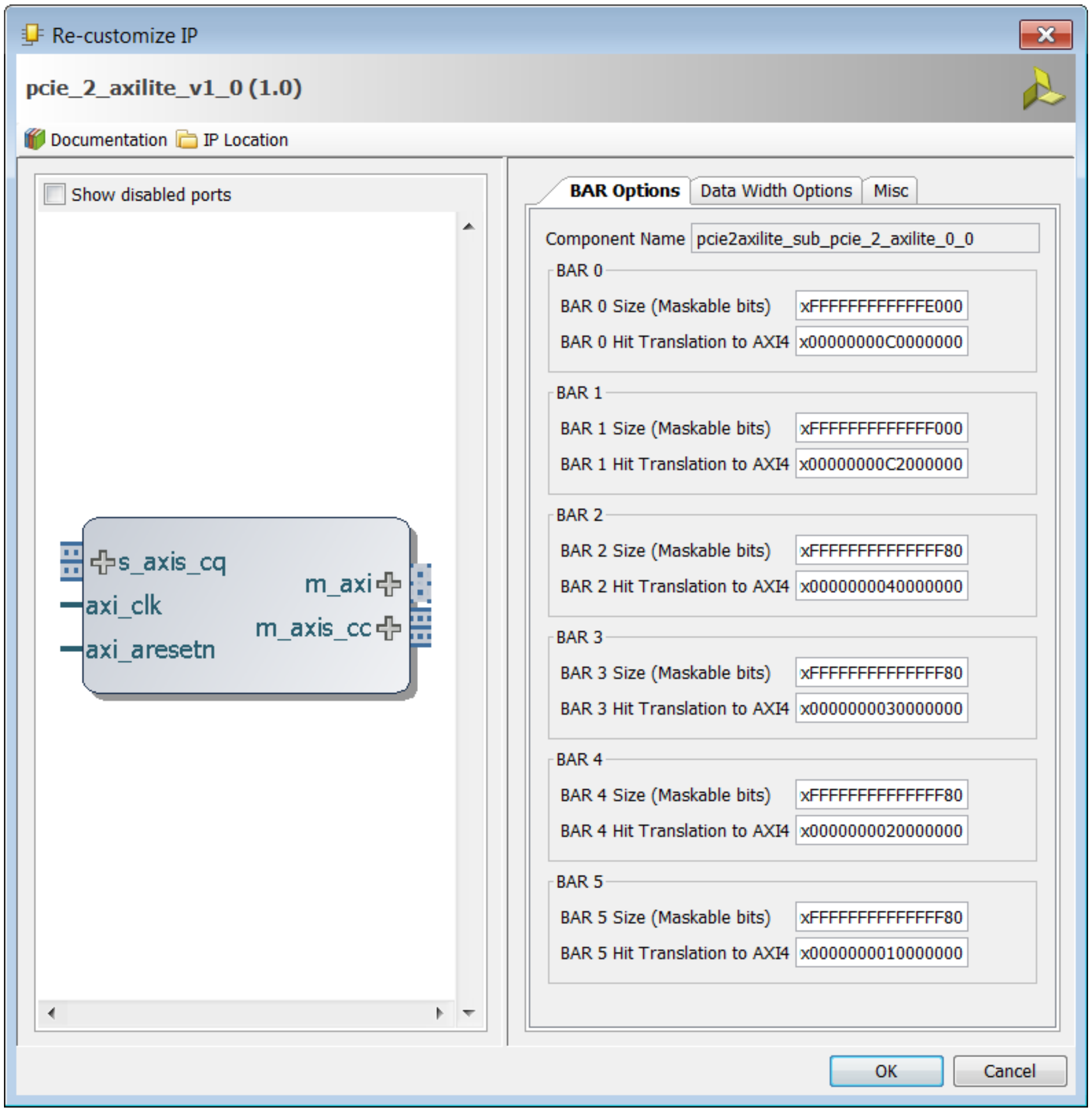


Figure 4: **CQ Descriptor**

The address provided from the descriptor comes directly from the PCI Express TLP. The TLP address is translated into the AXI4 system. Depending on the BAR hit, the address translates to different AXI4 addresses. The IP core is packaged with options for the translation to the AXI4 space within the BAR Options tab. Figure 5 shows the options available.



Component Name pcie2axilite_sub_pcie_2_axilite_0_0

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

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

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

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

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

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

Figure 5: **BAR Translation Options**

The BAR size changes depending on the amount of maskable bits assigned. The size of the BAR in the IP should match the BAR size that the endpoint is set to. Table 1 shows the valid values allowed in the BAR # Size field, along with their corresponding aperture size.

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

Valid Maskable Value	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 Translation to AXI4 value provides the Base Address of where a BAR hit will translate to in the AXI4 address space. The asserted bits on the Maskable Bits filter out the corresponding base address offset in the PCI Express space and only provide the offset from the BAR hit. For example, if a 1 KB BAR enumerates to 0x00000000C0000000, then the masked bits would filter out the upper 54 bits (because the mask is 0xFFFFFFFFFFFFFC00), and only provide the lower 10 bits to determine the offset from the Translation to AXI option. Because the lower 10 bits are being used for the offset, then the Translation to AXI field must not use the lower 10 bits as an offset value because this does not keep the aperture size address aligned. This must be taken into account when determining the Translation to AXI4 address.

Table 2 shows two examples for using BAR Size and Translation. Example #1 shows the bridge set up with 1 KB of addressable space assigned to a particular BAR. The 1 KB range comes from the Maskable Bits option of the bridge. 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, resulting from how the BAR is enumerated. After masking out the upper bits from the descriptor address x000000000C000004, the resulting address is simply the offset address of four. To translate this offset into the AXI domain, add the Translation to AXI4 option to the offset and the resulting address in the AXI domain is x0000000080000004. This is how a translation from the descriptor to the AXI domain is determined.

Example #2 of Table 2 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

Figure 6 provides a flow chart representation of Example #1 from Table 2.

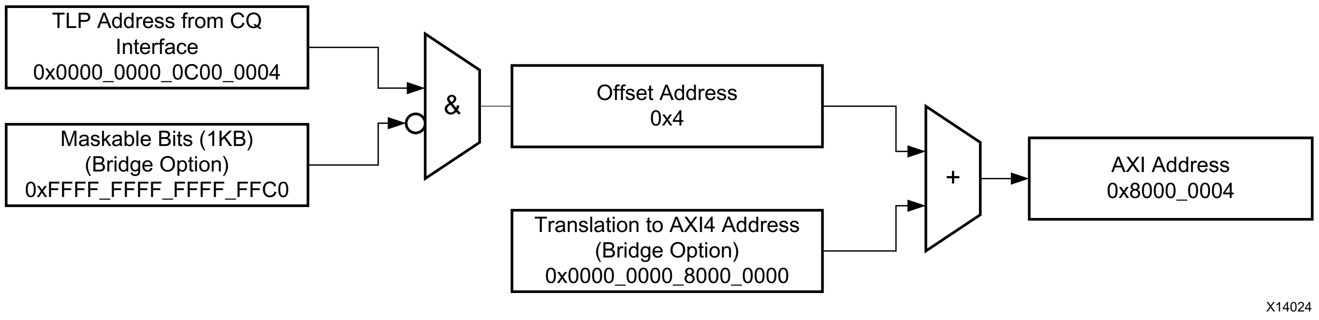


Figure 6: **Example #1 as a Flow Chart**

Figure 7 provides a flow chart representation of Example #2 from Table 2.

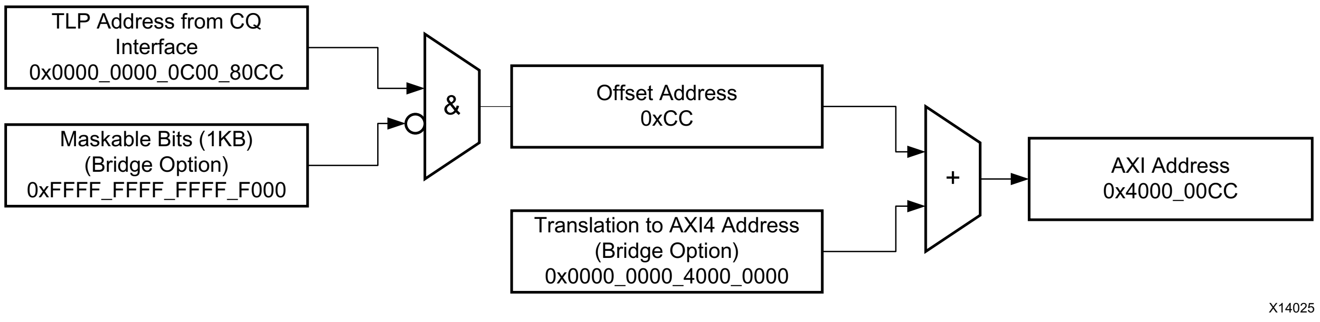
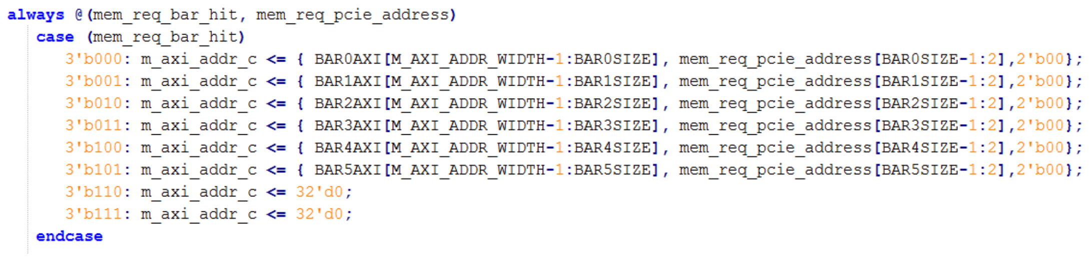

TLP Address from CQ Interface 0x0000_0000_0C00_80CC
Maskable Bits (1KB) (Bridge Option) 0xFFFF_FFFF_FFFF_F000
&
Offset Address 0xCC
Translation to AXI4 Address (Bridge Option) 0x0000_0000_4000_0000
+
AXI Address 0x4000_00CC
X14025

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

The RTL code performing the Translation to AXI4 is shown in Figure 8. The BAR#SIZE value is determined by the least significant high bit set in the Maskable Bits field. 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 Translation to AXI4 field.

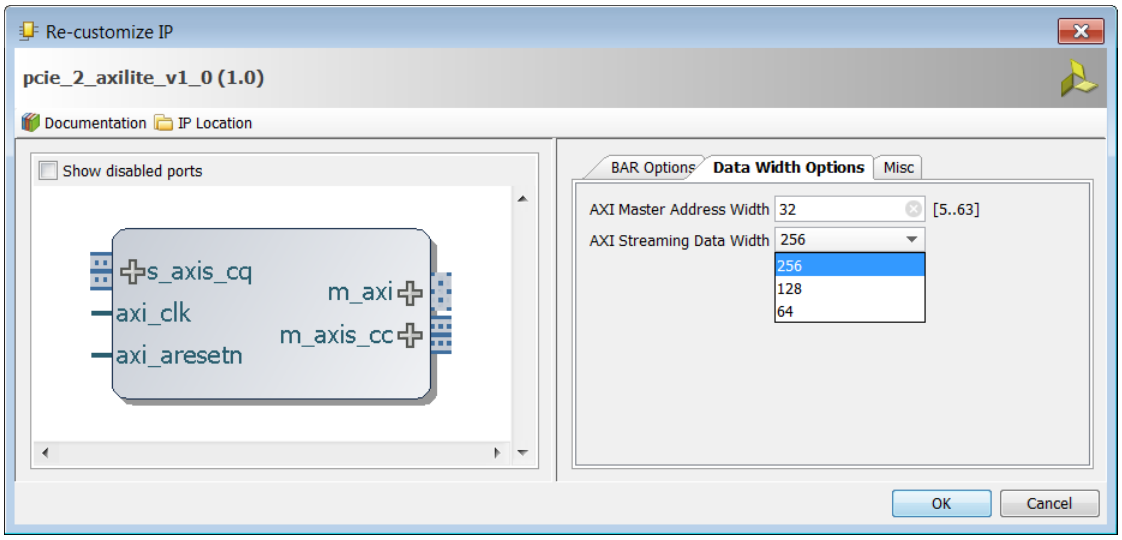


```
always @(mem_req_bar_hit, mem_req_pcie_address)
   case (mem_req_bar_hit)
      3'b000: m_axi_addr_c <= { BAR0AXI[M_AXI_ADDR_WIDTH-1:BAR0SIZE], mem_req_pcie_address[BAR0SIZE-1:2],2'b00};
      3'b001: m_axi_addr_c <= { BAR1AXI[M_AXI_ADDR_WIDTH-1:BAR1SIZE], mem_req_pcie_address[BAR1SIZE-1:2],2'b00};
      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
```

Figure 8: **BAR Translation RTL**

Data and Address Width

The data width of the AXI Master is fixed at 32-bit, whereas the data width of both Completer interfaces (CC and CQ) are modifiable in the bridge with the AXI Streaming Data Width option. The AXI Streaming Data option must match the width from the Integrated Block IP. The Master AXI Address Width determines the addressable space the master AXI interface can access. If this is 32, then the bridge can master onto 4 GB of address space. If it is set to 33, then it is able to master onto 8 GB of space. This is independent to the BAR sizes.

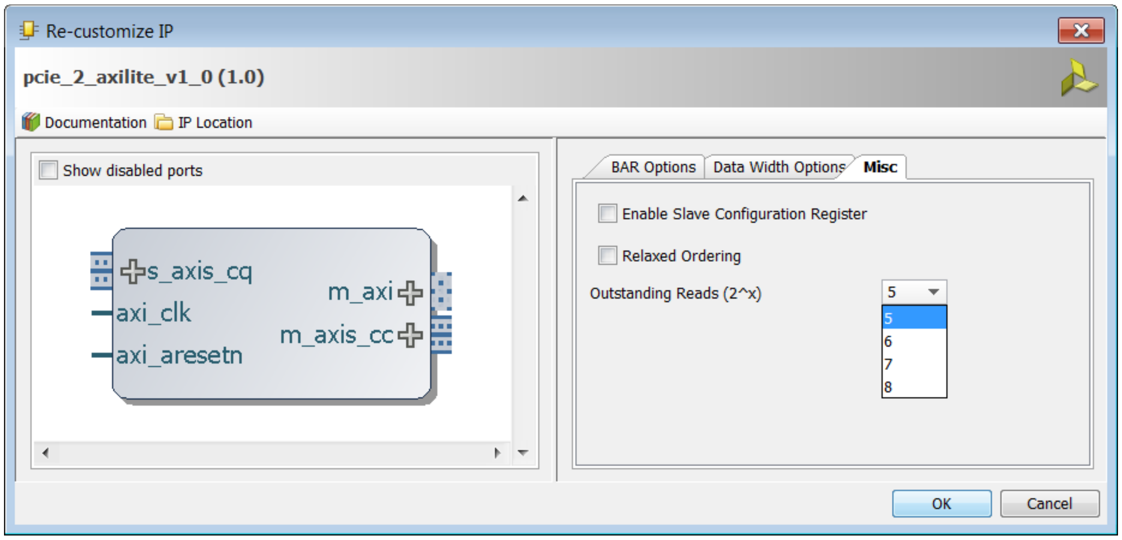


AXI Master Address Width 32 [5..63]
AXI Streaming Data Width 256

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

Additional Options

The bridge is able to accept multiple reads from the PCI Express block before the data is returned from the AXI interface. When a read is accepted by the bridge and the bridge has not returned the data, this condition is called an "outstanding read request." The bridge options also allow you to determine how many outstanding read requests need to be handled. Figure 10 shows the range of outstanding reads can vary from 32 (2^5) up to 256 (2^8).



Enable Slave Configuration Register
Relaxed Ordering
Outstanding Reads (2^x) 5 (options: 5, 6, 7, 8)

Figure 10: **Miscellaneous Options**

Two additional options are provided to allow for differentiation. The Enable Slave Configuration Register option allows a slave interface for some additional abilities. When this is enabled, you must make modifications to the RTL in the `pcie2axilite_bridge\rtl` directory to add customized options. The following bullets show some use cases for customizations with the Slave interface:

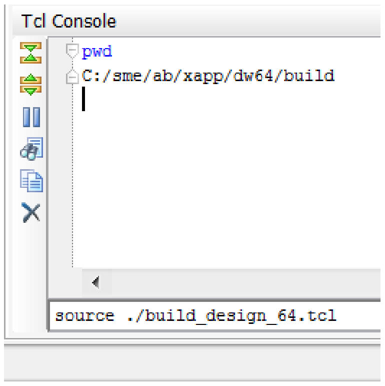
- Dynamic BAR translation
- Error conditions
- Debug register

The Relaxed Ordering option allows for TLPs to pass one another. This may violate the PCI Express specification so it is not recommended to use this option without a thorough analysis.

Implementing the Reference Design

Click here to download the design files associated with this application note.

The reference design provides three different examples of how the bridge is configured with three different user interface widths: 64-bit, 128-bit, and 256-bit. Within the Vivado IDE, source the `build_design_#.tcl` located in the `dw#/build` directory, where the # represents the data width.



```
pwd
C:/sme/ab/xapp/dw64/build
```

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

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

By sourcing the Tcl script, a Vivado project is generated and the project creates a bit file. The resulting bit file function with the MET driver and application from *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [Ref 2]. The software files are not provided in this application note to maintain one location for the MET driver. For the latest MET driver, refer to *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [Ref 2].

The reference design also comes with an Integrated Logic Analyzer (ILA) core on three different interfaces to see the flow of traffic when TLPs are coming in. The three interfaces with ILA cores are listed below:

- CQ interface
- CC interface
- AXI4-Lite Master

Triggering on a rising edge of "valid" shows the packets flowing through the bridge.

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, a batch script is provided in the `simulation` directory. The `run_sim.bat` script uses the Vivado Simulator to compile, elaborate, and run the simulation. The `pcie_2_axilite_tb.v` file contains parameter `C_DATA_WIDTH` to adjust for the 64-bit, 128-bit, or 256-bit interface data widths. To modify the transactions, the `cq_axis_stimulus.v` module provides the following four calls:

- pcie_write (address, data, enable bits, data width)
- pcie_read (address, data width)
- write_seq (write count, address, data, data width)
- read_seq (read count, data width)

Using the Prebuilt Images

Prebuilt bitstreams (.bit) and hardware analyzer files (.ltx) are provided for all three interface widths. These bitstreams functions with the MET driver from *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [Ref 2] on a VC709 board. To use the prebuilt bitstreams:

1. Open the Vivado IDE without a project and type the following into the Tcl console:

   ```
   >> open_hw
   ```

2. Establish a JTAG connection and select the bitstream and ltx file to target the Virtex-7 family on the VC709 board.
3. Program the FPGA and the notice all of the ILA cores in the design.
4. Set the trigger to detect a rising edge on any of the valid signals of any AXI interface.
5. Run the PIO application from *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* [Ref 2] and traffic appears on the ILA core.

Resource Utilization

Table 3 describes the resource utilization for Virtex-7 devices on the VC709 board.

Table 3: **Resource Utilization**

TDATA Width	LUTs	FFs	RAMs
x64	277	276	0
x128	289	297	0
x256	289	297	0

File Description

Table 4 describes the directory structure of the reference design.

Table 4: **Reference Design Files**

Directory and Files	Description
< data width: dw64, dw128, dw256 > <build> build_design_[data width].tcl	Tcl file to build a bitstream.
< data width: dw64, dw128, dw256 > <source> <constaints> top.xdc	Constraints file containing location constraints and Vivado Hardware Debug constraints.
< data width: dw64, dw128, dw256 > <source> <ipi> ipi_design_[data width].tcl	Tcl file to build IPI system.
< data width: dw64, dw128, dw256 > <source> <rtl> pcie2axilite_bridge.v	Top-level wrapper file for IPI design.
< pcie2axilite_bridge > <rtl> Verilog files	IP source files.
< pcie2axilite_bridge > <xgui> pcie_2_axilite_v1_0.tcl	Vivado packager Tcl Files for the GUI.
< pcie2axilite_bridge > component.xml	Vivado IP packaging file.

Table 4: **Reference Design Files *(Cont'd)***

Directory and Files	Description
< simulation > run_sim.bat/run_sim.sh run_time.tcl source.prj xsim_test.wcfg	Files to run simulation on Vivado Simulator.
< simulation > < verilog > *.v	Simulation test bench files.

Notes:
1. <> refers to a directory.

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.

References

This document contains the following references

1. *Virtex-7 FPGA Gen3 Integrated Block for PCI Express* (PG023)
2. *Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores* (XAPP1022)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
01/23/2014	1.0	Initial Xilinx release.
09/02/2014	1.1	Updated Title.
09/08/2015	1.1.1	Revised RTL acronym.

Notice of Disclaimer

Automotive Applications Disclaimer