XAPP1286 (v1.0) June 23, 2016

7 Series FPGAs Gen2 Integrated Block for PCIe to AXI4-Lite Bridge

Authors: Jason Lawley, Sanjay Rai

Summary

Creating an AXI subsystem in a Xilinx FPGA allows designers to quickly and easily generate complex systems. When designers connect their system to PCI Express (PCIe) [Ref 2], they often use the *AXI Memory Mapped to PCIe Gen2 bridge*. This method may use valuable FPGA resources in a system which only requires simple read and write operations. This application note and accompanying source code shows designers how to create a very small PCIe to AXI bridge which supports 1 DWORD reads and writes from the host to the FPGA Endpoint, using a fraction of the resources of the fully featured *AXI Memory Mapped to PCIe Gen2 bridge*.

Reference Design

The PCIe to AXI4-Lite bridge design referenced in this application note has been packaged as a Vivado IP core to make it easy to integrate into an IPI design. The project and source code used to package the IP are also provided, as well as two projects which use the packaged IP targeting the Xilinx AC701 and ZC706 Xilinx reference boards.

Feature Support

- Endpoint accepts 1 DWORD PCIe reads and writes from the Host to the FPGA Endpoint
- Connects directly to the 7 Series Integrated Block for PCIe IP core
- Supports up to four 32-bit PCIe to AXI BAR translations with address masking
- Supports Endian swapping
- Supports 64-bit and 128-bit data widths (Gen1 x1 - Gen2 x8)

Hardware

Figure 1 shows the block diagram of the IPI block. The `s_axis_rx` and `m_axis_tx` ports connect to the 7 Series Integrated Block for PCIe. The `user_clk` port must come from the 7 Series Integrated Block for PCIe (as the PCIe to AXI4-Lite bridge must be synchronous to this clock). The `M_AXI` port connects to the AXI subsystem developed in IPI.

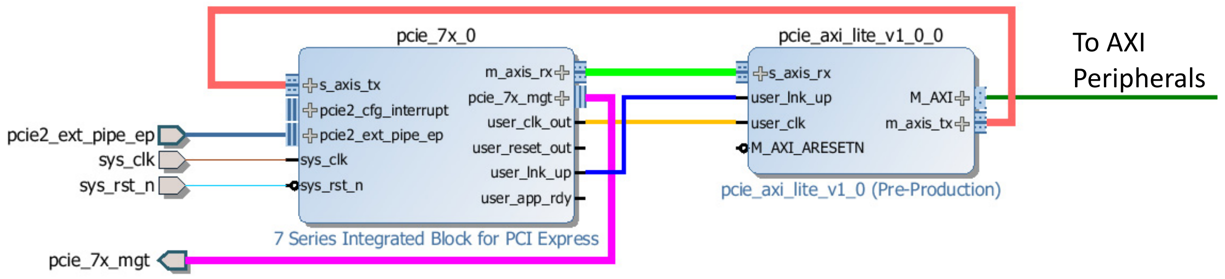
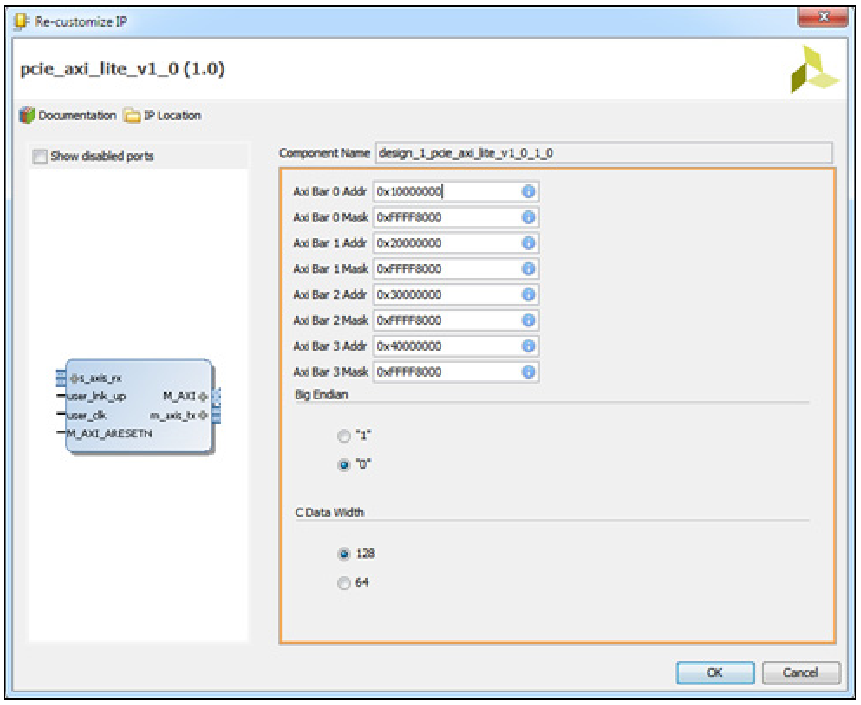


Figure 1: **IPI Block Diagram**

The `s_axis_rx` port receives PCIe reads and writes from the 7 Series Integrated Block for PCIe. This port processes one PCIe transaction at a time. The PCIe reads and writes have their address translated to the AXI transactions.

If the transaction is a PCIe read, an AXI read is generated on the `M_AXI` port. When the result comes back, a PCIe completion transaction is generated on the `m_axis_tx` port.

Customizing the PCIe to AXI4-Lite bridge is a simple process. The customization GUI is shown in Figure 2.



pcie_axi_lite_v1_0 (1.0)

Show disabled ports

Component Name	design_1_pcie_axi_lite_v1_0_1_0
Axi Bar 0 Addr	0x10000000
Axi Bar 0 Mask	0xFFFF8000
Axi Bar 1 Addr	0x20000000
Axi Bar 1 Mask	0xFFFF8000
Axi Bar 2 Addr	0x30000000
Axi Bar 2 Mask	0xFFFF8000
Axi Bar 3 Addr	0x40000000
Axi Bar 3 Mask	0xFFFF8000

Big Endian: "1" [✓ "0"]

C Data Width: [✓ 128] 64

Figure 2: **Customization GUI**

Component Name:

Enter the top level name for the core. An illegal name is highlighted in red until it is correct.

Address Translation and Masking:

You can configure address translation and masking for up to four 32-bit PCIe BARs. If a PCIe BAR is unused, the default value can be left "AS-IS".

Big Endian:

Endian selection allows the data to be either Big Endian or Little Endian. Select as required.

C Data Width:

The C data width must be set to match the data width of the 7 Series Integrated Block for PCIe IP.

The main benefit of the PCIe to AXI4-Lite bridge is its small size. Because it does not need to implement a full, high performance PCIe bridge, the amount of FPGA resources that it consumes is significantly reduced. Below is a table that shows the amount of resources each bridge uses based on the link width and speed of the PCIe link.

Table 1: **Bridge Resource Usage**

	Lite Bridge		**Full Bridge**[1]			
	LUTs	**FFs**	**LUTs**	**FFs**	**LUT Reduction**	**FF Reduction**
Gen 1 x1	881	1365	11372	8701	-92%	-84%
Gen 2 x4	1296	2344	14695	10888	-91%	-78%

1. AXI Memory Mapped to PCIe Gen2 bridge, provided by Xilinx.

Software Application

To show how to use the PCIe to AXI4-Lite bridge, two reference projects have been packaged with the IP. In both projects, all unneeded core interfaces have been disabled and PIPE mode enabled so the design can be quickly simulated.

The first reference project (Figure 3) targets an *AC701* Artix-7 reference board. A simple BRAM and the board LEDs are connected to the PCIe to AXI4-Lite bridge. BAR0 is connected to the BRAM, and BAR1 is connected to the LEDs.

The 7 Series Integrated Block for PCIe IP has been configured as Gen2 x4 with two PCIe BARs enabled.

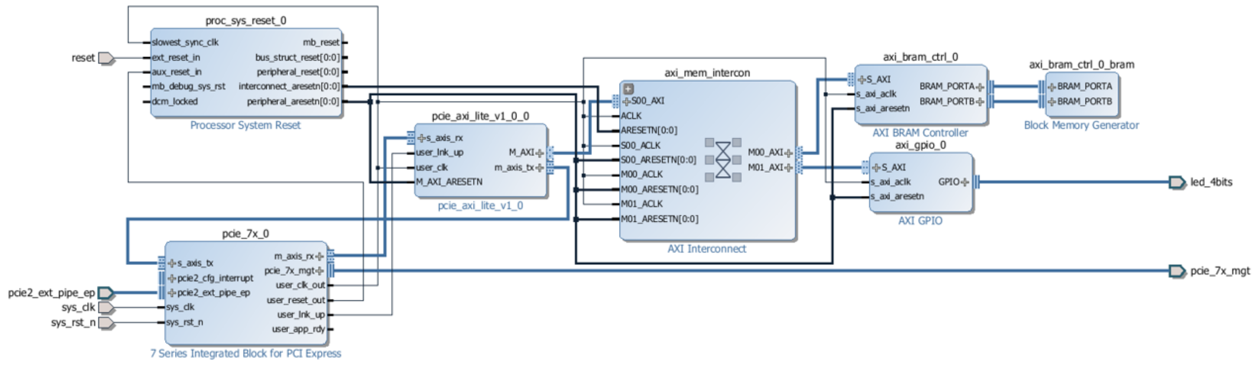


Figure 3: **AC701 Artix-7 Reference Project**

The second reference design (Figure 4) targets the *ZC706* Zynq-7000 reference board. This reference design again connects BRAM and the board LEDs. In addition, the Zynq-7 Processing System has been enabled to support ARM® processors.

The 7 Series Integrated Block for PCIe IP has been configured as Gen1 x1 with two 4 KB PCIe BARs enabled.

Note: AXI BFMs are required for the simulation to complete when targeting the ZC706 design. More information about the availability and cost of AXI BFMs can be found here.

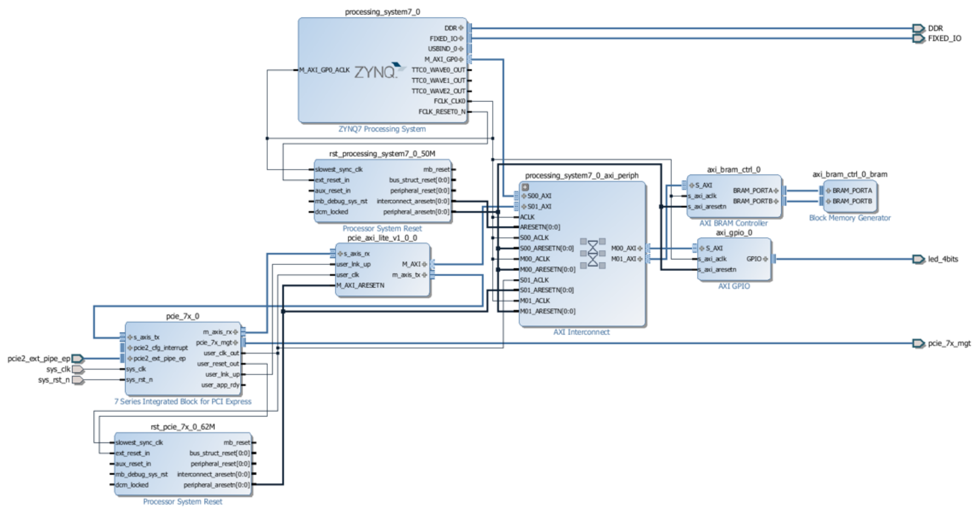


Figure 4: **ZC706 Zynq-7000 Reference Project**

Requirements

Both reference designs are built using Vivado 2015.4 and are upgradeable to any future version of Vivado. While the reference designs target specific boards, there is no specific board requirement for the PCIe to AXI4-Lite bridge. It can be used with any Xilinx 7 Series device with the exception of Virtex-7 devices with GTH transceivers (See XAPP1201).

Hardware

- Evaluation Kit:
    - Artix-7 FPGA AC701

      or
    - Zynq-7000 All Programmable SoC ZC706
- Power Supply for boards

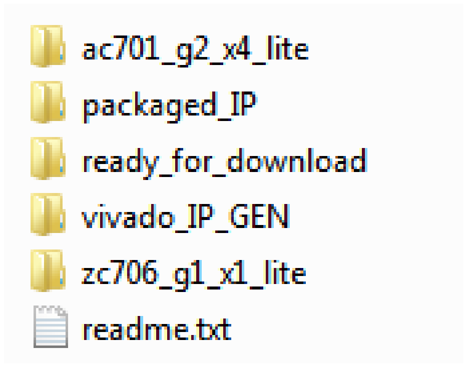
Software

Vivado 2015.4 was used to create the example designs. The designs can be upgraded to any newer version of Vivado using the automatic IP upgrade flow in Vivado.

Reference Design Files

You can download the Reference Design Files for this application note from the Xilinx website.

Figure 5 shows the reference design directory structure.



ac701_g2_x4_lite
packaged_IP
ready_for_download
vivado_IP_GEN
zc706_g1_x1_lite
readme.txt

Figure 5: **Reference Design Directory Structure**

Reference Design Steps

Follow these steps to setup the PCIe to AXI4-Lite bridge. Each step includes an explanation of the settings used in the **zc706_x1_g1_lite** reference design.

Note: **ac701_g2_x4** settings are similar except there is no processing subsystem in its design.

The setup steps are summarized below, detailed steps are given in Setup.

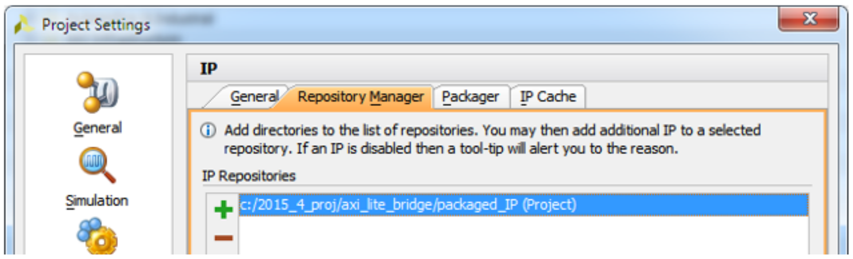
1. Download and extract the ZIP file.
2. In the Vivado project, add the 'packaged IP' directory to the Vivado IP Repository.
3. In a Vivado project in IPI, add the 7 Series Integrated Block for PCIe IP and configure it as required.
4. Add the PCIe to AXI4-Lite bridge to IPI and configure it as required.
5. Connect up the rest of the IPI design, and generate the top level file.
6. If required, simulate the design (Zynq 7000 requires AXI BFMs).
7. Implement the design.

Setup

This section provides detailed information to complement the above summary.

Note: The procedure assumes the use of the ZC706 board.

1. Download and extract the ZIP file.
2. Open Vivado.
3. Select **Open Project**.
4. Browse to the **zc706_x1_g1_lite** directory.
5. Add the PCIe to AXI4-Lite bridge to the IP catalog of the project:
   a. Go to **Tools** > **Project Settings**.
   b. In the Project Settings window, select the **IP** button on the right hand side.
   c. Click **'+'** to add the 'Packaged_IP' directory as shown in Figure 6.



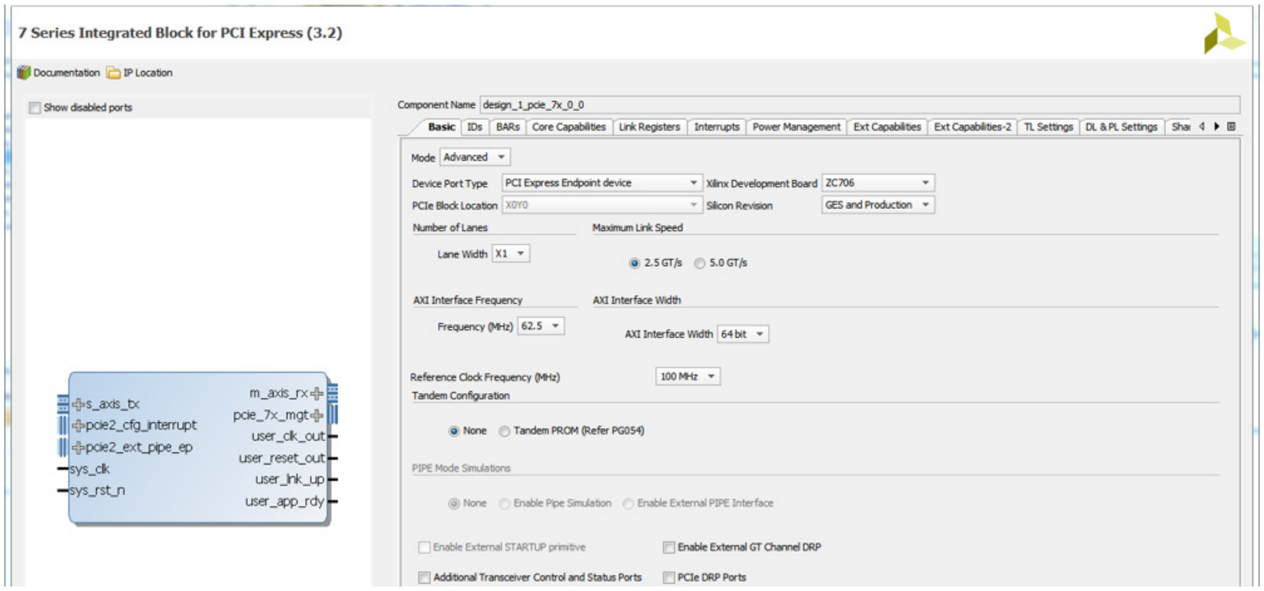
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

c:/2015_4_proj/axi_lite_bridge/packaged_IP (Project)

Figure 6: **Packaged_IP added to Repository Manager**

6. Create an IPI block diagram.

7. In the IPI block diagram, add the 7 Series Integrated Block for PCIe IP core.
8. Double click the core to display its settings (see Figure 7).



7 Series Integrated Block for PCI Express (3.2)

Documentation IP Location

Show disabled ports

Component Name: design_1_pcie_7x_0_0

Basic | IDs | BARs | Core Capabilities | Link Registers | Interrupts | Power Management | Ext Capabilities | Ext Capabilities-2 | TL Settings | DL & PL Settings | Sha

Mode: Advanced

Device Port Type: PCI Express Endpoint device   Xilinx Development Board: ZC706

PCIe Block Location: X0Y0   Silicon Revision: GES and Production

Number of Lanes
Lane Width: X1

Maximum Link Speed
[✓ 2.5 GT/s] 5.0 GT/s

AXI Interface Frequency
Frequency (MHz): 62.5

AXI Interface Width
AXI Interface Width: 64 bit

Reference Clock Frequency (MHz): 100 MHz

Tandem Configuration
[✓ None] Tandem PROM (Refer PG054)

PIPE Mode Simulations
[✓ None] Enable Pipe Simulation Enable External PIPE Interface

☐ Enable External STARTUP primitive   ☐ Enable External GT Channel DRP

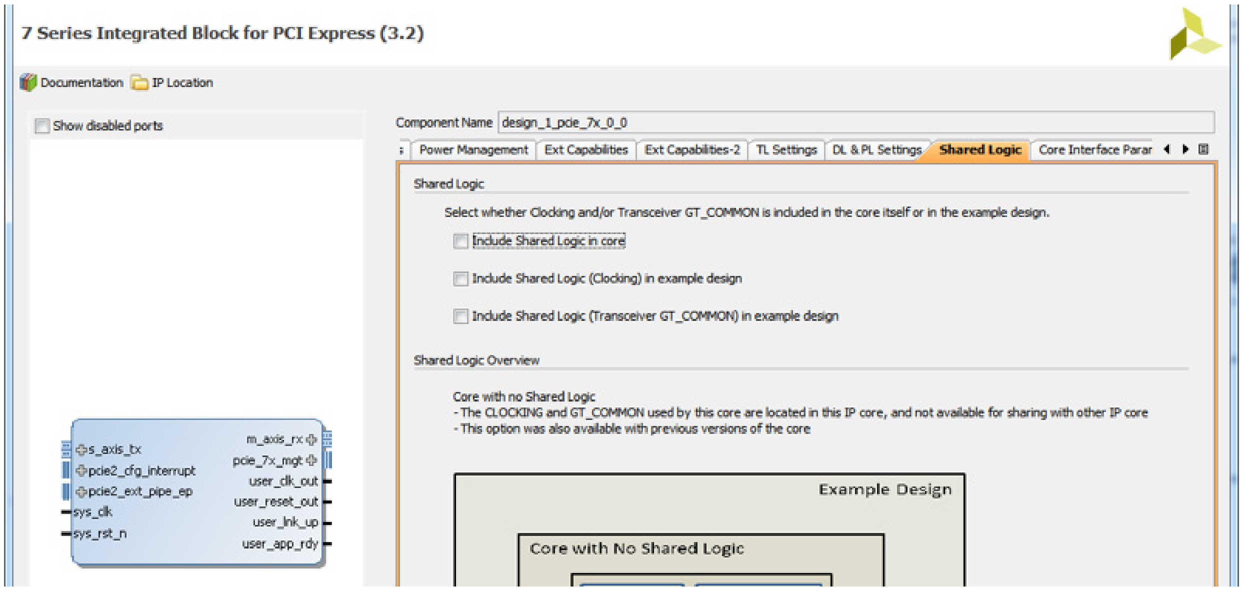
☐ Additional Transceiver Control and Status Ports   ☐ PCIe DRP Ports

Figure 7: **7 Series Integrated Block for PCIe - Basic Settings**

9. In the Basic tab, change the Lane Width and Maximum Link Speed to the required values.
10. In the Basic tab, change the Mode to 'Advanced'. Make any other changes necessary for the design including changing IDs, turning on PCIe BARs and setting the required sizes.

    Note: PCIe BAR sizes should not exceed 2 GB.

11. It is important to check the **Shared Logic** tab (see Figure 8) and ensure that none of the options are selected. If any of the options are selected, you must ensure that the shared logic gets added to the IPI design, and connected to the PCIe block.



7 Series Integrated Block for PCI Express (3.2)

Documentation IP Location

Show disabled ports

Component Name: design_1_pcie_7x_0_0

Power Management | Ext Capabilities | Ext Capabilities-2 | TL Settings | DL & PL Settings | Shared Logic | Core Interface Parar

Shared Logic

Select whether Clocking and/or Transceiver GT_COMMON is included in the core itself or in the example design.

☐ Include Shared Logic in core

☐ Include Shared Logic (Clocking) in example design

☐ Include Shared Logic (Transceiver GT_COMMON) in example design

Shared Logic Overview

Core with no Shared Logic
- The CLOCKING and GT_COMMON used by this core are located in this IP core, and not available for sharing with other IP core
- This option was also available with previous versions of the core

Figure 8: **7 Series Integrated Block for PCIe - Shared Logic Settings**

The PCIe to AXI4-Lite bridge offers several customization options as shown in Figure 2.

PCIe to AXI Address Translation and Masking

The PCIe to AXI Bar Address and PCIe to AXI Bar Mask settings allow for the incoming PCIe packets that 'hit' different PCIe BARs to be translated to different address spaces in the AXI memory map.

The size of the PCIe BAR specified in the 7 Series Integrated Block should be set to the same size as that of the Address Bar Mask in the PCIe Lite customization window. Table 2 shows the valid values allowed in the PCIe BAR # Size field, along with their corresponding aperture size.

Table 2: **Example of maskable bits and BAR size**

Valid Maskable Value	Corresponding BAR Aperture Size
`0xFFFF_FF80`	128 Bytes (minimum)
`0xFFFF_FF00`	256 Bytes
`0xFFFF_FE00`	512 Bytes
`0xFFFF_FC00`	1 KB
...	...
`0xF000_0000`	256 MB
`0xE000_0000`	512 MB
`0xC000_0000`	1 GB (maximum)
`0x8000_0000`	2 GB (maximum)

The Translation to the AXI4 value provides the Base Address of where a PCIe BAR hit will translate to in the AXI4 address space. The asserted Maskable Bits filter out the corresponding base address offset in the PCIe space, and provide the offset from the PCIe BAR hit. For example, if a 1 KB PCIe BAR enumerates to 0xC000_0000, then the masked bits would filter out the upper 22 bits (because the mask is 0xFFFF_FC00), 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, 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 an AXI4 address.

Table 3 shows two examples for using PCIe BAR Size and Translation.

Example #1 shows the bridge set up with 1 KB of addressable space assigned to a particular PCIe BAR. The 1 KB range comes from the Maskable Bits option of the bridge. The host has enumerated this PCIe BAR to address 0x0C00_0000. The host has requested from address offset four of the PCIe BAR, which comes out to be a TLP address of 0x0C00_0004, resulting from how the PCIe BAR is enumerated. After masking out the upper bits from the PCIe descriptor address 0x0C00_0004, 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 0x8000_0004. This is how a translation from the PCIe descriptor to the AXI domain is determined.

Figure 9 shows the address calculation for example 1 represented as a flowchart.

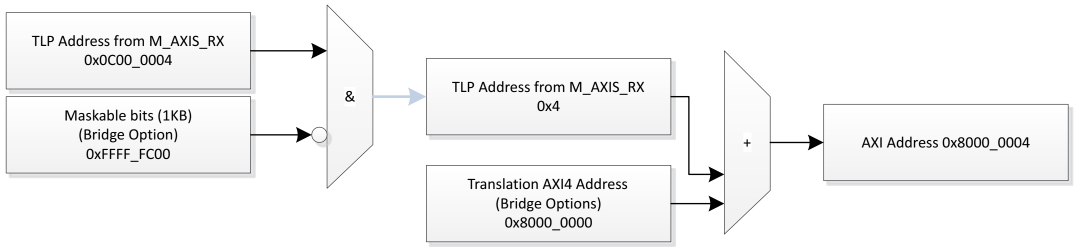


Figure 9: **Example 1 - Flowchart**

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

Figure 10 shows the address calculation for example 2 represented as a flowchart.

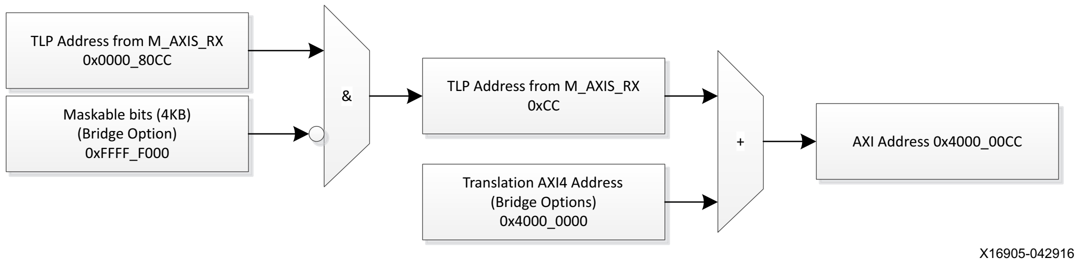


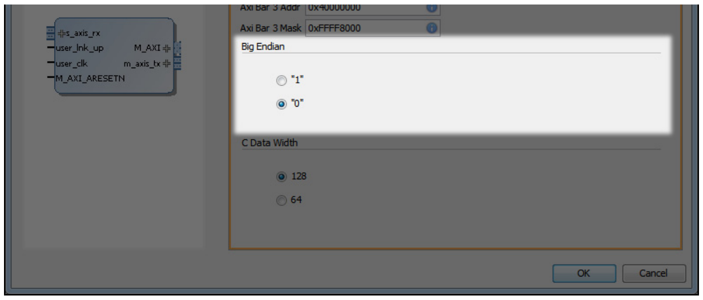
Figure 10: **Example 2 - Flowchart**

Table 3: **Example of Address Translation**

	Example 1	**Example 2**
BAR Enumerated Address (Assigned from host)	0x0C00_0000	0x0000_8000
Address in PCIe Descriptor (Request from host)	0x0C00_0004	0x0000_80CC
AXI Bar x Mask (Maskable bits)	0xFFFF_FC00	0xFFFF_F000
BAR Size (PCIe Core Option)	1 KB	4 KB
AXI Bar x Addr (Bridge Option)	0x8000_0000	0x4000_0000
Resulting AXI Address	0x8000_0004	0x4000_00CC

Endian Selection

The endianness of the data can be controlled from the **Big Endian** setting, as shown in Figure 11.


Axi Bar 3 Addr 0x40000000
Axi Bar 3 Mask 0xFFFF8000

Big Endian
"1" [✓ "0"]

C Data Width
[✓ 128] 64

Figure 11: **Endian Selection**

This selection determines how bytes from the PCIe packet are mapped into the AXI memory space. There is no right or wrong selection. Selecting “1” means that the data will be Big Endian and selecting “0” means that the data will be Little Endian.

As shown Figure 12, the Endian selection will determine the order of the bytes within the Dword.

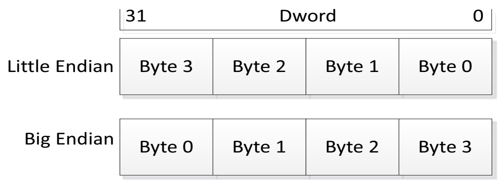


Figure 12: **Endian Selection (Dword bytes)**

Data Width

The data width (64 or 128 bits) should be changed (if required) to match the data width as selected when customizing the 7 Series Integrated Block for PCIe IP core. This width changes based on the PCIe link width and speed selected. In some cases, you have the option to choose the data width.

Note: When changing data width, it is important to remember that the lower of the data widths will need to operate at a higher frequency, so timing closure may be more difficult.

Running the Reference Design

You can run the AC701 and ZU706 designs from the Vivado Flow Navigator by double clicking **Run Simulation** under the Simulation heading. The provided reference projects have been setup to run PIPE mode simulation which is not the default. Enabling PIPE mode simulations can drastically decrease simulation times.

To run **Synthesis**, **Implementation** or to **Generate a bitstream**, double click the appropriate option.

When a bitstream has been generated, the design can be loaded onto the board and 1 DWORD memory reads and writes can be generated. You can implement this flow using WinDriver from Jungo Systems, available here.

See here for a video demonstration of AXI PCIe with MIG on a KCU105 using WinDriver from Jungo Connectivity.

In some operating systems, the memory enable bit may be turned off by default. The user driver or application should enable this bit for memory read/writes to function properly.

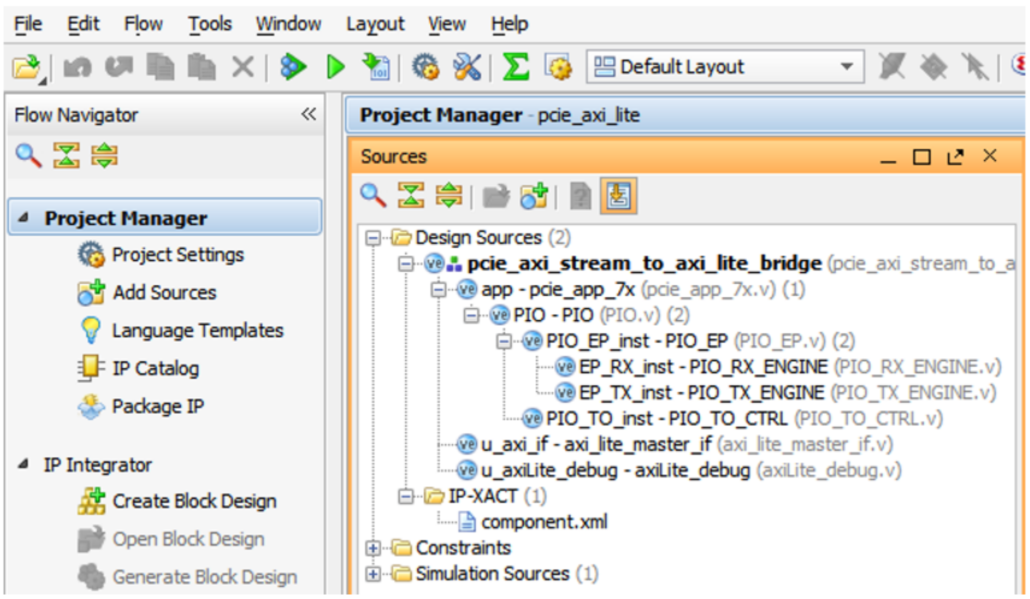
Modifying the IP

You can modify the IP to enable additional functions, for example:

- Add additional PCIe BAR support
- Add an AXI Slave to PCIe 1 DWORD bridge

Modifying and repackaging the core is performed in Vivado in the directory:

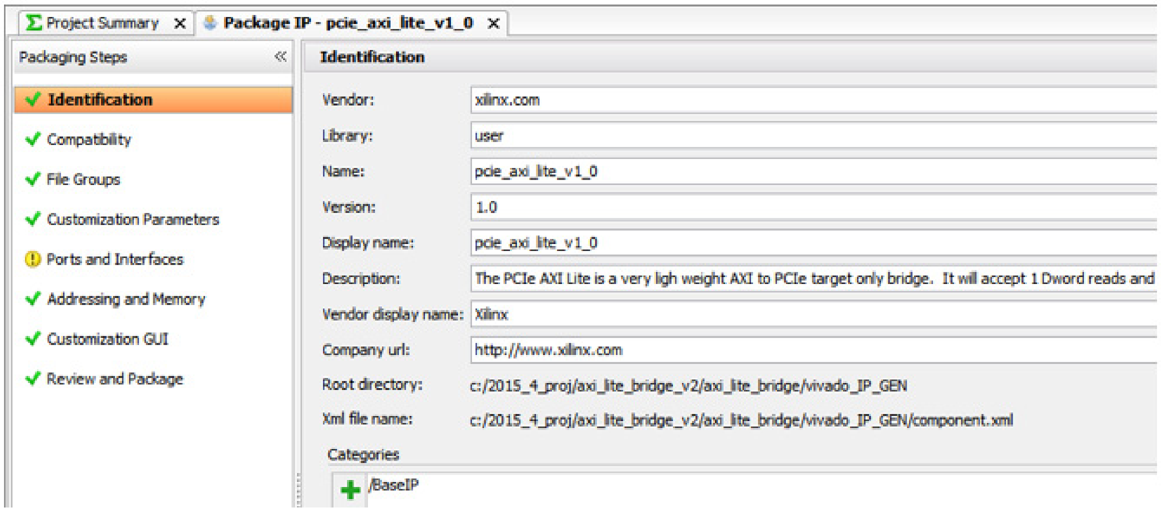
`axi_lite_bridge/Vivado_IP_GEN/pcie_axi_lite/pcie_axi_lite.xpr` (see Figure 13).



```
Design Sources (2)
  pcie_axi_stream_to_axi_lite_bridge (pcie_axi_stream_to_a
    app - pcie_app_7x (pcie_app_7x.v) (1)
      PIO - PIO (PIO.v) (2)
        PIO_EP_inst - PIO_EP (PIO_EP.v) (2)
          EP_RX_inst - PIO_RX_ENGINE (PIO_RX_ENGINE.v)
          EP_TX_inst - PIO_TX_ENGINE (PIO_TX_ENGINE.v)
        PIO_TO_inst - PIO_TO_CTRL (PIO_TO_CTRL.v)
    u_axi_if - axi_lite_master_if (axi_lite_master_if.v)
    u_axiLite_debug - axiLite_debug (axiLite_debug.v)
  IP-XACT (1)
    component.xml
Constraints
Simulation Sources (1)
```

Figure 13: **Vivado Project Manager**

After making the required modifications, double click on the **component.xml** file in the **Sources** window. This will open the IP packaging options. Here, updates can be made before the IP is packaged including updating the description, version number etc. (see Figure 14).



Identification

Vendor: xilinx.com
Library: user
Name: pcie_axi_lite_v1_0
Version: 1.0
Display name: pcie_axi_lite_v1_0
Description: The PCIe AXI Lite is a very ligh weight AXI to PCIe target only bridge. It will accept 1 Dword reads and
Vendor display name: Xilinx
Company url: http://www.xilinx.com
Root directory: c:/2015_4_proj/axi_lite_bridge_v2/axi_lite_bridge/vivado_IP_GEN
Xml file name: c:/2015_4_proj/axi_lite_bridge_v2/axi_lite_bridge/vivado_IP_GEN/component.xml

Figure 14: **Package IP Window**

Once the IP is packaged, import it into the desired Vivado project. The new IP design can now be used and shared.

References

1. *7 Series FPGAs Integrated Block for PCI Express Product Guide* (PG054)
2. *AXI Memory Mapped to PCI Express (PCIe) Gen2 Product Guide* (PG055)
3. *AXI Interconnect v2.1 Product Guide* (PG059)
4. *Virtex-7 (XT and HT) and UltraScale FPGAs Gen3 Integrated Block for PCI Express to AXI4-Lite Bridge* (XAPP1201)

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
06/23/2016	1.0	Initial Xilinx release.

Please Read: Important Legal Notices