Abstract
This reference system demonstrates the functionality of the PLBv46 Endpoint Bridge for PCI Express® used in the Xilinx ML507 Embedded Development Platform. The PLBv46 Endpoint Bridge uses the Xilinx Endpoint core for PCI Express in the Virtex®-5 XC5VFX70T FPGA. The PLBv46 Bus is an IBM CoreConnect bus used for connecting IBM PowerPC® 405 and PowerPC 440 and the MicroBlaze™ microprocessors to Xilinx IP cores.

A variety of tests generate and analyze PCIe® traffic for hardware validation of the PLBv46 Endpoint Bridge. PCIe transactions are generated and analyzed by Catalyst and LeCroy test equipment. For endpoint to root complex transactions, the pcie_dma software application generates DMA transactions which move data over the PCIe link. For root complex to endpoint transactions, Catalyst and LeCroy scripts generate PCIe traffic. A Catalyst script which configures the PLBv46 Endpoint Bridge and performs memory write/read transactions is discussed. The steps to use the Catalyst test equipment to measure PCIe performance are given, and performance results are provided. The principal function of the section on performance measurements is to show how performance measurements are made.

Two stand-alone tools, PciTree and Memory Endpoint Test, are used to write and read PLBv46 Endpoint Bridge configuration space and memory in a PC environment. The PC provides the least expensive and easiest to use PCIe hardware test environment.

The use of the ChipScope™ tool in debugging PLBv46 Endpoint Bridge issues is described.

Included System
The reference system for the PLBv46 Endpoint Bridge in the ML507 Embedded Development Platform is available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=113624

The reference system in the xapp1040.zip file is described on page 2.

Introduction
The PLBv46 Endpoint Bridge is a PCIe endpoint instantiated in a Xilinx FPGA which communicates with a root complex. The reference systems are tested using commercial test equipment from LeCroy and Catalyst. LeCroy and Catalysts are two Analyzers/Exercisers used to verify PCIe systems. The Catalyst and LeCroy testers allow generation, analysis, capture, and triggering of Translation Layer, Data Link Layer, and Physical Layer packets. The reference system is also tested in an inexpensive PC based test environment.

The PLBv46 Endpoint Bridge is tested using the LeCroy and Catalyst testers as root complex. The ML507 Evaluation Board is inserted into the LeCroy or Catalyst PCIe slots for testing. Sample Catalyst scripts are provided in the ml507_ppc440_plbv46_pcie/catalyst directory. Sample Lecroy scripts are provided in the ml507_ppc440_plbv46_pcie/lecroy directory.

The tests for the PLBv46 Endpoint Bridge which do not require LeCroy or Catalyst test equipment are the PCIE Configuration Verification (PCIE CV), PciTree, and the Memory EndPoint Test (MET) tests. These are run using the ml507_ppc440_plbv46_pcie project. These
tests are quick to setup and costs nothing other than a PC with PCIe slots. For these tests, the ML507 Embedded Development Platform is inserted into the x1 PCIe slot of a PC (Dell 390). The PC based PciTree and/or MET software are installed. The PciTree Bus Viewer (www.pcitree.de) and the Xilinx MET tests allow the user to write and read ML507 memory with any pattern, with different lengths. PciTree and the MET do not provide the capability to analyze PCIe traffic.

The hardware and software requirements for this reference system are:

- Xilinx ML507 Rev A board
- Xilinx Platform USB or Parallel IV programming cable
- Serial communication cable and serial communication utility (Tera Term, HyperTerminal)
- Xilinx Platform Studio 10.1.03
- Xilinx Integrated Software Environment (ISE®) 10.1.03
- Xilinx ChipScope Pro 10.1.03
- Catalyst SPX Series PCI Express Bus Protocol Analyzer/Exerciser
- LeCroy PETracer Analyzer / PETrainer Exerciser

This reference system includes the Power PC 440 Processor, PPC DDR2 Memory Controller, XPS BRAM, XPS INTC, XPS UART Lite, XPS Central DMA, and PLBv46 Endpoint Bridge. The Power PC 440 processor runs at 400 MHz and the bus run at a frequency of 100 MHz. The PowerPC440 processor uses the instruction cache (I-cache) and data cache (D-cache). The PowerPC DDR2 Memory Controller runs at a frequency of 200 MHz.

Figure 1 is the block diagram of the reference system.

**Table 1** provides the address map of the system.

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Instance</th>
<th>Base Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPS INTC</td>
<td>xps_intc_0</td>
<td>0x81800000</td>
<td>0x8180FFFF</td>
</tr>
<tr>
<td>XPS BRAM CNTLR</td>
<td>xps_bram_if_cntlr_1</td>
<td>0xFFFF0000</td>
<td>0xFFFFFFFF</td>
</tr>
<tr>
<td>XPS Central DMA</td>
<td>xps_cdma_0</td>
<td>0x80200000</td>
<td>0x8020FFFF</td>
</tr>
</tbody>
</table>

**Figure 1:** Reference System
Note: This reference system is developed using Base System Builder. In the xapp1040.zip design file provided, a second MHS file, system2.mhs, provides a reference system which uses two PLBv46 Buses. In this MHS, the master(s) of the PLBv46 Bridge for PCI Express and the XPS Central DMA controller connect to the SPLB0 port of the Power PC 440 to communicate to the DDR2. This system is simulated in XAPP1111, scheduled for release in December 2008.

In XPS, double click on PCIe_Bridge in the System Assembly View to invoke the PLBv46_PCIe generics editor. The generics shown in Figure 2 are used to configure the PLBv46 Endpoint Bridge. The Xilinx Device ID = 0x0505 and Vendor ID = 0x10EE are displayed in many of the PCIe tests done in this application note. The PLBv46 Bridge for PCI Express v3.00.a supports one PCI Base Address Register (PCIBAR) and one IPIF Base Address Register (IPIFBAR). New releases of the core will support additional BARs. In Figure 2, unsupported BARs are dimmed.

Table 1: Reference System Address Map

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Instance</th>
<th>Base Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLBv46 Endpoint Bridge</td>
<td>plbv46_pcie_0</td>
<td>0x85C00000</td>
<td>0x85C0FFFFF</td>
</tr>
<tr>
<td>XPS Uartlite</td>
<td>RS232</td>
<td>0x84000000</td>
<td>0x8400FFFFF</td>
</tr>
<tr>
<td>ppc440mc_ddr2</td>
<td>DDR2_SDRAM</td>
<td>0x00000000</td>
<td>0x00FFFFFFF</td>
</tr>
</tbody>
</table>

The resource utilization in the reference design is shown in Table 2.
Table 2: Design Resource Utilization

<table>
<thead>
<tr>
<th>Resources</th>
<th>Used</th>
<th>Available</th>
<th>Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>10946</td>
<td>44800</td>
<td>41</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>12188</td>
<td>44800</td>
<td>42</td>
</tr>
<tr>
<td>DCM_ADV</td>
<td>2</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Block RAM</td>
<td>36</td>
<td>60</td>
<td>93</td>
</tr>
</tbody>
</table>
Figure 3 shows the ML505 Embedded Development Platform, which has the same form factor as the ML507. The ML505 uses the xc5vlx50t and the ML507 uses the xc5vfx70t. The xc5vfx uses the hard PowerPC 440 processor. The xc5vlx uses the MicroBlaze processor. The ML507 has a x1 PCIe connector on one edge of the printed circuit board.

Executing the Reference System

The sequence of steps to test the PLBv46 Endpoint Bridge reference system differs depending on whether endpoint to root complex transactions or root complex to endpoint transactions are run. For endpoint to root complex transactions, the steps must be run in the order below. For root complex to endpoint transactions, the steps are the same, but there is no ELF to download.

1. Use iMPACT to download the bitstream.
2. `impact -batch xapp1040.cmd`
3. Use the Catalyst to write the PLBv46 Endpoint Bridge Configuration Space Header.
Testing the PLBv46 Endpoint Bridge

File -> Open catalyst/cfg_x1.sdc
Run
4. Invoke XMD and connect to the MicroBlaze processor.
   xmd
   connect ppc hw
   debug -reset_onrun system disable
5. Download the executable.
   dow pcie_dma.elf
6. From the XMD prompt, run
   con

Testing the PLBv46 Endpoint Bridge

The system, including the interface to the LeCroy and Catalyst test equipment, is shown in Figure 4. The root complex is the Catalyst or LeCroy test equipment, and the endpoint is the PLBv46 Endpoint Bridge in the ML507 reference system.

Figure 4: PLBv46 Endpoint Bridge System Identifying Root Complex/Endpoint

Endpoint to Root Complex Transactions

Endpoint to root complex transactions are tested using XMD commands and C code. Two software projects, pcie_dma and pcie_mch_dma, generate Direct Memory Access (DMA) transactions which create PCIe traffic. This code provides an interface to the user which allows the selection of the number of loops to run and the seed. The code generates and verifies pseudo random traffic patterns on the PCIe link.

The pcie_dma.c code uses one DMA channel. The pcie_mch_dma.c code allows the user to specify 1-3 DMA channels.
The PLBv46 Endpoint Bridge Configuration Space Header (CSH) must be written for the code to run correctly. The Catalyst and LeCroy scripts, cfg_x1.sdc and cfg_x1.peg, set up the configuration space header of the PLBv46 Endpoint Bridge.

The Catalyst PCI Express Bus Protocol Exerciser/Analyzer has memory located at address 0x00000000. In the reference systems, the PLBv46 Endpoint Bridge generic C_IPIFBAR2PCIBAR_0 is set to 0x00000000. This is different from the Base System Builder (BSB) generated value for C_IPIFBAR2PCIBAR_0.

Figure 5 shows the selection of the pcie_dma software project.
endpoint_example. The endpoint_example project demonstrates basic Endpoint operations of the PLBv46 PCIe.

pcie_dma. The pcie_dma project runs Direct Memory Access (DMA) operations. The user sets the source address, destination address, and DMA length. The pcie_dma code is used for DMA operations between user defined source and destination addresses. Figure 6 shows the parameters in pcie_dma.c which are edited to test PCI transactions between different memory regions. The ELF for pcie_dma.c runs on the PowerPC440 processor in the xc5vfx70t FPGA on the ML507 Evaluation Platform.

pcie_mch_dma. The pcie_mch_dma project runs multi-channel Direct Memory Access (DMA) operations. The user sets the source address, destination address, and DMA length for each channel. The pcie_mch_dma code is used for DMA operations between user defined source and destination addresses. As with the pcie_dma code, the parameters in pcie_mch_dma.c which can be edited to test PCI transactions between different memory regions are DMAChannel[*].BAR. The ELF for pcie_mch_dma.c, provided in ready_for_download, as pcie_mch_dma.elf, runs on the PowerPC440 processor in the xc5vfx70t FPGA on the ML507 Evaluation Platform.

DMA Transactions
As examples of specifying the source and destination addresses in DMA transactions, the source address may be an address in the ML507 XPS BRAM and the destination address a Catalyst memory across the PCIe link. Another option is source address in Catalyst memory to a second location in Catalyst memory.

```c
#define MEM_0_BASEADDR 0xFFFF0000
#define MEM_1_BASEADDR 0xC0000000

DMALength = 1024
```

Figure 6: Defining Source and Destination Addresses, Length in pcie_dma.c

The XMD scripts and C code generate DMA operations to transfer data between different ML507 and Catalyst memory regions. DMA transactions are generated by writing to the Control, Source Address, Destination Address, and Length registers of the DMA controller. Table 3 provides the register locations for the XPS Central DMA. In the reference design, C_BASEADDR is set to 0x80200000.

Table 3: XPS Central DMA Registers

<table>
<thead>
<tr>
<th>DMA Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Register</td>
<td>C_BASEADDR + 0x04</td>
</tr>
<tr>
<td>Source Address Register</td>
<td>C_BASEADDR + 0x08</td>
</tr>
<tr>
<td>Destination Address Register</td>
<td>C_BASEADDR + 0x0C</td>
</tr>
<tr>
<td>Length Register</td>
<td>C_BASEADDR + 0x10</td>
</tr>
</tbody>
</table>
The `pcie_dma.c` code consists of the four functions in the functional diagram in Figure 7. The Barberpole Region function provides a rotating data pattern in the memory located at the source address. The Zero Region function sets the memory located at the destination address to all zeroes. The DMA Region function generates a DMA transaction of data located at the source address to the memory at the destination address. Following the DMA transfer, the Verify function verifies that data at the source and destination address are equal.

![Functional Diagram of pcie_dma.c](image_url)
Figure 8 show the communication terminal output when running the pcie_dma/executable.elf.

Figure 8: pcie_dma.c Output

Catalyst Testing

This section discusses testing using Catalyst Enterprises SPX Series PCI Express Analyzer/Exerciser system. The SPX is a serial bus Analyzer/Exerciser used to analyze and/or exercise PCI Express data transactions. The SPX4 Analyzer consists of the SPX4 card and Analyzer software. The Analyzer allows capture and trigger on Transaction and Data Link Layer Packets, Physical Layer Ordered Sets, and all bus conditions. The Exerciser generates bus traffic while operating as either a root complex or endpoint device.

Figure 9 shows a functional diagram of the Catalyst test setup.

Figure 9: Catalyst Test Setup
Figure 10 is a photograph of the Catalyst setup. The ML507 Evaluation Platform is inserted into the PCIe slot. The Platform Cable USB cable is connected to the ML507 to use Impact, XMD, and GDB. A USB cable connects the PC based Catalyst software to the SPX4 Analyzer.

Figure 10: Photo of Catalyst PCI Express Test Equipment

In addition to using the Catalyst Bus Protocol Analyzer/Exerciser software as discussed in this application note, the Catalyst SpekChek PCI Express Compliance Suite has been run with this reference design to verify that the PLBv46 Endpoint Bridge meets PCI SIG compliance tests. The SpekChek tests are defined in the SpekChek User Manual Version 6.5.
After downloading the bit file into the ML507 FPGA using Impact, the PLBv46 Endpoint Bridge Control Register (BCR) is written as shown in Figure 11. The BCR enables the PCIe Bus Master and the Base Address Registers (BARs). This step is not necessary if using the pcie_dma or pcie_mch_dma software applications because these applications write to the BCR.

Figure 11: Writing the Bridge Control Register
Five tabs are used to set up the Catalyst PCIe Bus Protocol Analyzer/Exerciser. Figure 12 shows Catalyst **Capture** settings. The option selected is to **Capture Everything except Idles**. In the **Trigger On** tab, select **Pattern** and **Trigger on TLP (Any Type)**. Select **Any Direction**. In the **Settings** tab, specify the name of the output ssf file.

*Figure 12: Capture Settings*
Figure 13 shows the setup of the Catalyst **Link Settings**. Select the Platform mode (hidden behind the Link Status pane). Click on the Link Status button to invoke the Link Status pane displayed. The figure shows a Link Width = 1, indicating that the link is up and trained as x1.

**Figure 13: Catalyst Link Settings**
Figure 14 is a graphical view of the stimuli for configuring the PLBv46 Endpoint Bridge, including BAR 0. The ml507_ppc440_plbv46_pcie/catalyst directory contains the cfg_x1.sdc stimuli file. The cfg_x1.sdc project is loaded using the File -> Open pull down menu. The *.sdc files are readable text files which contain the transactions used as stimuli.

In cfg_x1.sdc, the Device ID/Vendor ID is read. The Command Status register is written and read. The Revision ID and Class Code register is read.

In the figure, the Name column provides the type of transaction and the Reg Num column specifies the register in the Configuration Space Header.

BAR0 is written and read. BAR0 is a 64-bit BAR with the lower 32 bits defined at Configuration Space Header (CSH) Register Number 4 and the higher 32 bits defined at CSH Register Number 5.

Packets 10 and 11 are Configuration Writes and packets 12 and 13 are Configuration Reads.

In the Data field in packet 10, the endianess of the data written is swapped.
Figure 15 shows the Analyzer output after running cfg_x1. The results are contained in the cfg_x1.ssf file. Registers in the Configuration Space Header are displayed in packet 0 using Vendor ID and Device ID symbolic names, with Xilinx 0x10EE and 0x0505 values. The Command Status Register is read. The SC in the status field indicates successful completion of the transaction. In the figure, the Revision ID and Class Code Register field is expanded to provide a readable table of the values in the Data field.

Figure 15: Results from Catalyst Configuration
Figure 16 shows an excerpt of the Exerciser cfg_x1.sdc file. The file contains the stimuli TLPs. While it is generally easier to read and edit the TLPs using the Catalyst Display Viewer, the text file is readable and editable, and more details are provided than can be efficiently presented in a Display Viewer. The figure shows the content of a single Configuration Read TLP.

```
Packet_Type = "Config Read TO"
Framing_Symbol1 = "FB"
Reserved_1 = "0"
Sequence_Number = "000"
Reserved_2 = "0"
Format = "0"
Type = "04"
Reserved_3 = "0"
TC = "0"
Reserved_4 = "0"
TD = "0"
EP = "0"
Attribute = "0"
Reserved_5 = "0"
Length = "001"
Requester_ID = "0000"
Tag = "00"
Last_DW_BE = "0"
First_DW_BE = "F"
Bus_Number = "00"
Device_Number = "00"
Function_Number = "0"
Reserved_6 = "0"
Register_Address = "000"
Reserved_7 = "0"
TLP_Digest = ""
LCRC = "2AC19647"
Framing_Symbol2 = "FD"
Loop_Type = "No Loop"
Loop_Count = ""
Iterate_After_Trigger = "No"
Delay_Count = "0"
Trigger_Source = "Immediate_Execution"
Disparity_Error = "No"
ZData = "10000000000000000001"
Symbol_View = "Collapse"
Trigger_Output = "No"
Trigger_Output_Type = "Pulse"
Global_Loop
```

Figure 16:  sdc_example
As Root Complex, the Catalyst Exerciser performs memory writes and memory reads to the ML507 Evaluation Platform memory. The ML507 reference design contains an XPS BRAM controller and a Power PC Memory Controller interface to DDR2. Figure 17 shows the memory addressing for Root Complex (Catalyst) to Endpoint (ML507) transactions. The memory addressed is controlled by the BAR value written and by the C_PCI BAR2IPIF BAR_* generic(s). In the reference design, BAR0 is written as 0x0000000060000000. C_PCIE BAR2IPIF BAR_0 addresses XPS BRAM at 0xFFFF0000, and C_PCIE BAR2IPIF BAR_1 addresses DDR2 at location 0x00000000.

Figure 17: Catalyst Root Complex
Figure 18 shows the TLPs which write then read in the wr_rd_x1.sdc file. In the figure, Packet 0 is a MWr64 to address 0x0000000060000000 of 128 bytes. The Data Field allows the user to specify data as Upcount, Walking Bit, Random pattern, or a user defined pattern such as 0x12345678 can be entered. As exercises in learning to use the PLBv46 Endpoint Bridge, the data can be varied, and the memory written/read can be changed from XPS BRAM to DDR2.

The Length field is 020H which is 32 doublewords (DWs) or 128 bytes.

Packet 1 is a MRd64 of address 0x0000000060000000, used to verify the written data. The MRd64 TLP address endianess differs from the CfgWr address endianess used when the BAR was written with a CfgWr in Figure 14. Bit Order and Endianess can be specified by right clicking a field to invoke a pop up menu.
Figure 19 shows the results after running a version of wr_rd_x1.sdc in which a random pattern of 0x0AADC5B9F1B0DC89 is transmitted.

Figure 19: Catalyst wr_rd_x1 Results
Figure 20 shows the use of XMD to read ML507 DDR2 memory to provide a second verification that the wr_rd_x1.sdc script functioned as intended. The data read in XMD should be the same as the data in the Analyzer waveform display.

Figure 20: Verifying Root Complex to Endpoint Transactions with XMD
Using Catalyst to test PCIe Performance

In this section, the Catalyst test equipment provides performance tests for Root Complex to Endpoint transactions, first for read transactions and then for write transactions. The test setup is defined and then performance results are given for various lengths for 32 and 64 bit transactions.

Figure 21 shows the physical link setup for the performance test. For the ml507_ppc440_plbv46_pcie project, change the Physical Layer Settings Link Width to x1.

![Catalyst Enterprise, Inc. SPN Analyzer/Exerciser Software](image)

**Figure 21:** Performance Test Physical Settings

**Root Complex to Endpoint Performance Tests**

To set up the performance test, insert the ML507 Evaluation Platform into the Catalyst test equipment. The bitstream is downloaded into the FPGA. Use XMD to write 0x003F0107 to the PLBv46 Endpoint Bridge Bridge Control Register to enable the Bus Master and BARs.
Root Complex to Endpoint Read Operations

Figure 22 shows the opening of the rc2ep_rd64 performance project. Performance projects use the `spf` extension.

![Opening a Catalyst Performance Test](image)

Figure 22: Opening a Catalyst Performance Test

The four tabs used in performance projects are the Exercise Program, Performance Items, Link Settings, and Settings. In Performance Items, the type of performance tests run are defined. The PCIe traffic used in the performance measurement is defined in the Exercise Program.
Figure 23 shows a single TLP used in the performance measurements of Rd64 transactions of length = 003. Click the TLP button below Performance Items to add the TLP to the Exercise Program. Using the pop up menu, select Memory -> Read Request 64 bits. Fill out the address and Len fields. Select the Continuous radio button so that the TLP is continuously transmitted.

![Catalyst Enterprise, Inc. SPX Analyzer/Exerciser Software - [nc2ep_rd64.cfg](image)](image)

**Figure 23:** Defining MRd64 Performance Stimuli

The next figures show the performance results of MRd64 transactions, varying the length of the TLP. The single continuously transmitted TLP stimuli just defined is shown in the pane at the bottom of the figure. The left pane is a Link Chart which provides the average payload size. The right pane is a Link Chart which provides the data throughput and the payload throughput.

In the Performance Items tab, Link Usage, Number of Packets, and Latency are unchecked. Under Report Directions, Aggregate is checked. The Aggregate data throughput is the throughput in both directions.

In the following tests, Data Throughput is the overall bus traffic of all non-idle packets divided by the update interval. Payload Throughput is the payload data of TLPs divided by the update interval. The update interval, defined in the Settings tab, for the performance measurements in this document is 1 second.

The MRd performance is the round trip time including the MRd and Completion with Data packets.
Figure 24 shows the performance results of a MRd64 TLP of length 10. The data and payload throughput are 334.8 MB/s and 151.7 MB/s.
Figure 25 shows the performance results of a MRd64 TLP of length 40. The data and payload throughput are 282.3 MB/s and 194.2 MB/s.
Figure 26 shows the performance of MRd32 transactions of length = 3. The data and payload throughput are 298.5 MB/s and 42.2 MB/s.
Root Complex to Endpoint Write Transactions

Figure 27 shows a write transaction. The length field is set to 020H or 128 bytes. The data written is an Upcount pattern. The Continuous radio button is selected. The data throughput is 247.1 MB/s and the payload throughput is 185.3 MB/s.

Figure 27: MWr64 Performance Results, Length = 20
Figure 28 shows the performance results from running a continuous MWr32 transaction. The data and payload throughput are 241.4 MB/s and 185.3 MB/s.

This section measures the performance of Endpoint to Root Complex transactions. The stimuli for these transactions are generated using the Xilinx XPS Central DMA Controller (DMAC) in the system.mhs. The functionality of the DMA controller is discussed earlier in this application note. The DMA transaction is from the address specified in the DMAC Source Address register to the address specified in the DMAC Destination Address register. The length of the DMA transaction is specified by the value in the DMAC Length register.
Prior to generating the stimuli, the performance test is set up. Figure 29 shows the importing of the performance test setup file `catalyst/pcie_dma.spf`. The throughput measurements are aggregate.

**Figure 29:** Importing Performance Test Setup
To generate stimuli, either C code or an XMD script is used to write the DMAC registers. **Figure 30** shows an XMD script to generate stimuli. Using XMD scripts and commands allows the quick verification that the operation is functioning correctly. After running a DMA transaction, a `mrd` command can verify that the data at the source and destination addresses are equivalent. XMD commands may be too slow to give maximum performance results.

The DMA Status Register is monitored to determine if the DMAC is Busy. When the DMAC is not busy, a DMA transaction is initiated by a write to the DMAC Length register.

```tcl
set outfile [open "dma.txt" "w"]
connect ppc hw
rst
puts $outfile [mwr 0x85C001E0 0x003F0107]
puts $outfile [mwr 0x80200000 0x0000000A]
puts $outfile [mwr 0x80200030 0x00000003]
puts $outfile [mwr 0x80200004 0xC0000004]
puts $outfile [mwr 0x80200008 0x20000000]
puts $outfile [mwr 0x8020000C 0x20002000]
puts $outfile [mwr 0x20000000 0x12345678 100]
puts $outfile [mwr 0x20002000 0x00000000]
puts $outfile [mrd 0x80200014 1]
set DMASR_BUSY 0x40000000
puts $outfile "DMA Status Register = $DMASR"
set DMASR [mrd 0x80200014 1]
set DMASR_BUSY 0x40000000
puts $outfile [mwr 0x80200010 64]
puts $outfile [mrd 0x20000000 100]
puts $outfile [mrd 0x20002000 100]
close $outfile
exit
```

**Figure 30:** dma.tcl
Figure 31 shows the Catalyst SPX4 Analyzer/Exerciser output after running the per_ep2rc_x1.spf performance analyzer project. The payload throughput depends on various factors such as the size of the transfer, if print statements are included in the source code, and if the verification is included in the source code. For this run, all print statements are removed, there is no verification, and length is set to 20. The transfer is from XPS BRAM to Catalyst memory across the PCIe link. The data throughput is 19.0 MB/s and the payload throughput is 8.3 MB/s.

Figure 31: EP to RC Performance Test Using XMD
Figure 32 shows the performance of an Endpoint to Root Complex transaction using C code (pcie_dma_0.c) to generate stimuli with the length = 200. The data throughput is 61.8 MB/s and the payload throughput is 36.8 MB/s. In this test, the Source Address is XPS BRAM, which is 0xFFFF0000, and the Destination Address is 0x20000000, which translates to Catalyst memory, across the PCIe link.

Figure 32: EP to RC Performance Test Using C Code
LeCroy Testing

The LeCroy - ML507 test setup shown in Figure 33 is used to verify the PLBv46 Endpoint Bridge. The LeCroy tester is used as root complex, for both configuration and data transactions. The ML507 is inserted into the host emulator.

The ml507_ppc440_plbv46_pcie/lecroy directory contains the stimuli files which use peg as the filename extension.

This section discusses the procedures used in setting up the LeCroy, including defining the Recording and Generation Options. Root Complex to Endpoint transactions are discussed, followed by a section on Endpoint to Root Complex transactions.

Figure 33: LeCroy Test Setup
Figure 34 is a photograph of the LeCroy test setup. The ML507 Evaluation Platform is inserted into the LeCroy Host Emulator. The Platform Cable USB Programming cable is connected to the ML507 JTAG connector.
Figure 35 shows the menu for setting Generation Options after selecting Setup -> Generation Options.

The LeCroy ML test equipment is selected. Link Width is specified as x1. Select Host as the Interposer.

Figure 35: Setting Generation Options
Figure 36 shows the menu for setting Recording Options after selecting Setup -> Recording Options. The Simple Mode is used. An Event Trigger is selected.

The Buffer Size is specified as 32 MB and the Trigger Position is set at 90% post triggering. The x1 Lane Width is selected.
Figure 37 shows using **File -> Open** to open a LeCroy stimuli (cfg_x1.peg) file.

The LeCroy PETracer software provides the interface to the PETracer (Analyzer) and PETrainer (Exerciser). To run an analysis, click on the Record icon (the Sun) in the menu bar. Click the Traffic Light icon at the bottom left of the GUI. After the status bar indicates Traffic Finished, click the Stop icon (black filled square next to the Sun). This causes results to be shown in the Display area. Results files have a pex extension. Like peg files, pex files can be opened using File -> Open.
Figure 38 shows the use of XMD to enable the Bridge Control Register. The BCR enables the Bus Master and the Base Address Registers (BARs).

![Command Prompt - xmd]

After generation and recording options are specified and the BCR is written, the link must be trained. The Link State is displayed at the bottom of the PETracer GUI. Prior to training, the Link State is displayed as Detect.Quiet as shown at the bottom of Figure 37. After training, the Link State is displayed as L0. To initiate training, click on the Connect icon. To disable a trained link, click on the Disconnect icon.
Figure 39 shows LeCroy - ML507 PLBv46 Endpoint Bridge link is trained with the LTFSM in L0. If the clocking and resets are correct, link training occurs in less than one second. If link training is unsuccessful, the LTFSM cycles through training states.

**Figure 39: LeCroy After Link Trained**
**Root Complex to Endpoint Transactions**

As Root Complex, the LeCroy Trainer generates memory writes and memory reads to the ML507 Evaluation Platform memory. The ML507 reference design contains an XPS BRAM controller and an PowerPC Memory Controller interface to DDR2. *Figure 40* shows the memory addressing for Root Complex (Catalyst) to Endpoint (ML507) transactions. The memory addressed is controlled by the BAR value written and by the `C_PCIBAR2IPIFBRAR_*` generics. In the reference design, PCI BAR0 is written as `0x0000000600000000`. `C_PCIBAR2IPIFBRAR_0` addresses XPS BRAM at `0xFFFF0000`, and `C_PCIBAR2IPIFBRAR_1` addresses DDR2 at location `0x00000000`.

*Figure 40: LeCroy - ML507 Memory Addressing*
The display area shows the TLPS defined in the peg file. Figure 41 shows an excerpt from the rc2ep_wr_rd.peg file. The rc2ep_wr_rd.peg shown writes FFFFFFFFs to the six BAR registers in the Configuration Space Header (CSH). This is done using the Repeat construct. The first register written is BAR0, located at offset 0x10. After writing and reading the CSH, packets 32 - 34 are MRd64, MW64 0x12345678, and MRd64.

Figure 41: RC to EP Write/Read Test

The next figures show BAR0 configuration packets, followed by write, then read, operations on BAR0.
Figure 42 shows the configuration of BAR0 and the read, write, and read transactions. The address of BAR0 is 0x0000000060000000. Packet 0 is a CfgWr of the lower order address and packet 2 is a CfgWr of the higher order address. Packets 4 and 5 use CfgRd TLPs to verify the configuration writes. Packets 6, 7, and 8 are MRd32, MWr32, and MRd32 TLPs used to read and write BAR0 memory.

Double click on the Data field in packet 7 to display the 1234678 value.

The endianess of the address in the CfgWr0 TLP differs from the endianess of the address in the MWr32 and MRd32 TLPs.

Figure 42: Configuring and Testing BAR0
Figure 43 shows the results after running \texttt{rc2ep_cfg_wr_rd_bar0.peg}. Packet 9 is a MWr32 of 0x12345678 to address 0x0000000060000000. The address is translated using the generic \texttt{C_PCBAR2IPIFBAR\_0} to XPS BRAM at 0xFFFF0000. In packet 12, the data value 0x12345678 is returned in the CplD packet.

The status fields indicate Successful Completion (SC).
Figure 44 shows the verification of the Endpoint to Root Complex PCIe transactions using XMD. In the `system.mhs`, the PLBv46 Endpoint Bridge generic C_PCIBAR2PIFBAR0 is 0xFFFF0000, the location of XPS BRAM. This shows that the 0x12346578 written by the LeCroy Root Complex MWr64 TLP is resident in XPS BRAM.

Figure 44: XMD Verification of BAR0 Tests
Figure 45 shows an excerpt of a peg file.

The peg file used as stimuli in LeCroy transactions is readable and editable. In the figure, templates are defined for Configuration Write and Configuration Read TLPs. The Configuration Write template is called in the repeat loop to write \texttt{0xFFFFFFFF}s to the six Configuration Space Header BARs.

The peg files in the \texttt{ml507_ppc440_plbv46_pcie/lecroy} directory can be used to test the PLBv46 Endpoint Bridge on the ML507 Evaluation Platform.

```plaintext
template = TLP{
    Name = "MyCfgWrite"
    ; Template name
    TlpType = CfgWr0
    ; Write device Configuration Space
    FirstDwBe = 0xF;
    First DW Byte Enables
    Length = 1
    ; 1 DWORD
    Payload = ( 0xFFFFFFFF )
    template = TLP
    {
        Name = "MyCfgRead"
        ; Template name
        TlpType = CfgRd0
        ; Read device Configuration Space
        FirstDwBe = 0xF; First DW Byte Enables
        Length = 1
        ; 1 DWORD
    }
    ; Enumerate all 6 Base Address registers
    repeat = Begin { Count = 6 Counter = i }
    ; Write 0xFFFFFFFF into Base Address register
    packet = "MyCfgWrite"
    { Register = ( 0x10 + i * 4 )
        ; Wait for completion received
        wait = TLP {
            TLPType = Cpl
        }
        ; Read Base Address register
        packet = "MyCfgRead"
        { Register = ( 0x10 + i * 4 )
        }
        ; Wait for completion received
        wait = TLP {
            TLPType = CplD
        }
    }
    repeat = End
```

\textit{Figure 45: peg Example}
Endpoint to Root Complex Transactions

In Endpoint to Root Complex transactions, the read and write operations originate from the ML507 Evaluation Platform and target the LeCroy. The LeCroy model used in this application note, the PCI Express Multi-Lane (ML) Exerciser/Analyzer, does not have target memory. For read operations, the peg files are written to respond with read data.

Invoke PETracer and run File → Open lecroy/ep2rc_mrd32_1dw.

Endpoint (EP) to Root Complex (RC) transactions are generated with XMD commands or C code. Since the MWr and MRd TLPs originate from the ML507, the LeCroy peg files cause the LeCroy to wait for the TLP(s) from the ML507. Figure 46 shows the peg for the EP to RC MRd32. The LeCroy waits for the MRd32 packet from the ML507. When the MRd32 packet is received, the LeCroy returns a Completion with Data (CplD) packet with a \(0x12345678\) payload.

![Figure 46: ep2rc_mrd32](image)

Figure 47 defines the functionality of the LeCroy Root Complex when receiving a MRd32 transaction from the PLBv46 Endpoint Bridge endpoint on the ML507.
Figure 47: EP to RC MRd32 Test Stimuli (1 DW)
Figure 48 shows results from running the Endpoint to Root Complex memory read. The peg file is loaded. Start recording by clicking on the Sun icon in the menu bar. Click the Traffic Light icon. Generate a 1 read using XMD.

```
mrd 0x20000000 1
```

Click the Black Square to stop recording and view the results.

---

**Figure 48:** EP to RC MRd32 Test Results (1 DW)
Figure 49 shows the ep2rc_mrd32_4dw.peg for a four doubleword Endpoint to Root Complex MRd32.
Figure 50 shows results from running the XMD command below.

```
mrd 0x20000000 4
```

**Figure 50: EP to RC MRd32 Test Results (4 DW)**

**Endpoint to Root Complex Write Transactions**

Figure 51 shows the peg file for the EP to RC MWr32. As with EP to RC memory reads, start recording by clicking on the Sun icon, and then click on the traffic light.

```
wait = TLP {
  TLPPType = MWr32
}
```

**Figure 51: ep2rc_wait_mwr32.peg**
Figure 52 shows LeCroy Root Complex setup for analyzing an Endpoint to Root Complex MWR32 operation.

The XMD command below generates the stimuli for the PLBv46 Endpoint Bridge to transmit the TLP.

```
mwr 0x20000000 0x12345678
```

Figure 53 shows the Analyzer output for an EP to Root Complex Memory Write of 0x12345678.

Figure 52:  **EP to RC - Write Operation**

The XMD command below generates the stimuli for the PLBv46 Endpoint Bridge to transmit the TLP.

```
mwr 0x20000000 0x12345678
```

Figure 53:  **EP to RC Write Results**
The write operation is easily varied using XMD. The XMD command below writes eight locations.

\texttt{mwr 0x20000000 0x12345678 8}

Figure 54 shows the results from running the eight doubleword Endpoint to Root Complex write transaction.

\textbf{Figure 54: EP to RC Write Results - 8 DW}
Testing with a PC

Using a Personal Computer (PC) as Root Complex is an inexpensive method of verifying PLBv46 Endpoint Bridge functionality. PciTree and the Memory Endpoint Test run on PCs. Figure 55 shows the ML507 in a Dell 390 PC. The PC runs Windows XP and has the ISE, EDK, and PciTree software installed. The PC PCIe integrated circuits act as root complex. The Dell 390 has a x1 connector for PCIe slot 1 and a x8 connector for PCIe slot 4. In the Dell 390, only 4 of the 8 lanes of the x8 connector are active. The ML507 is powered from the ML507 power supply. The ML507 is inserted in PCIe slot 1.

The USB Platform Cable is connected to the ML507 JTAG port for Impact, XMD, and ChipScope operations. A Serial Communication Cable is connected to communicate to a communication terminal.

The power up sequence of the PC affects the PCIe scan. In order for BIOS to recognize the drivers and PCIe BARs at power up, the FPGA bit file should be loaded prior to PC power up. It is possible to configure the FPGA after PC power up using JTAG mode, but a warm restart is usually required to get a PCI scan to work. A warm restart is a PC Shutdown with Restart. Xilinx recommends writing the ML507 XCF32P PROM with the contents of the MCS file so that configuration occurs at power up.

Xilinx recommends the use of the Master SelectMap configuration mode. Since it is faster than Master Serial mode, the ML507 is more likely to be configured at the time of the PCIe scan.

Configuring the ML507 vfx70t when used in a PC PCIe Slot

In the ml507_ppc440_plbv46_pcie/ready_for_download directory, the ml507_ppc440_plbv46_pcie.mcs file is the configuration file for this reference design. Because configuring from PROM improves the accuracy of a PCIe scan, the next figures provide the steps for creating a mcs for the ML507.
Figure 56 shows the ML507 Boundary Scan chain. The first XCF32P is used to configure the FPGA. Right click the XCF32P to invoke the Prepare PROM GUI.
Provide the PROM file name as shown in Figure 57.

Figure 57: Defining the PROM File
Specify the XCF32P PROM as shown in Figure 58. Click **Add** and then **Next**.
Select the bit file (download.bit) as shown in Figure 59.
Figure 60 shows the generated mcs file.

Users generating the PROM file for the first time should reference pages 101-107 of UG201 (v1.4) Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs.

Use Impact to download the mcs file into the ML507 XCF32 PROM. Select the XCF32P, left click to invoke a menu, and select Program. Under the Programming Properties menu, check Parallel Mode under PROM Specific Properties.

The recommended configuration mode is Master SelectMap, which is specified when the configuration Mode Switch (SW3) is set to M0-0 (ON), M1-0, M2-1.

Insert the ML507 into the PCIe slot and apply power to the PC. Verify that the DONE LED is lit.

PciTree Testing

PciTree is shareware available from http://www.pcitree.de. It runs on Windows XP. PciTree can be used for either PCI or PCIe tests. In the tests described in this section, the ML507 Embedded Development Platform is inserted into a Dell 390 x1 slot for the mlt507_ppc440_plbv46_pcie project.
Invoke XMD and enable the master and BARs by writing to the Bridge Control Register by using the command:

\texttt{mwr 0x85C001E0 0x003F0107}

\textbf{Figure 61} shows the XMD output when the PLBv46 Endpoint Bridge configuration space header registers are read. At power-up, the Device ID is 0x0505 and the Vendor ID is 0x10EE. BAR0 is 0x0000000C. The values are displayed in Big Endian format (EE100505). The address value at offset 10 contains the addressing size and prefetchability fields.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{X1040_61_09}
\caption{XMD Read of PLBv46 Endpoint Bridge Registers}
\end{figure}
After invoking PciTree and running a scan, Figure 62 shows the ML507 PLBv46 Endpoint Bridge detected as Other; Memory Controller, with Bus Number 3, Device Number 0, Function Number 0, or BDF = 3.0.0. The Xilinx Vendor ID and Device ID are displayed. In its Configuration Space Header, BAR0 has a value of 0x000000C. The address, 0xE000000, in this example, varies for different BIOS setups.

Figure 62: PciTree Scan

To edit the registers in the Configuration Space Header (CSH), highlight the register in the Configuration Space Dump pane (lower right area of GUI) to edit and provide a value in the Edit Config Reg dialog box. As an example, select the Command Status Register, write xFFFFFFFFF in the Edit Config Reg dialog box, click Write ConfReg, and then click Refresh Dump to see the new value of the Command Status Register (CSR) displayed. The CSR value is not 0xFFFFFFFF because some bits in the CSR are reserved.
Click on BAR0 and use the edit ConfReg dialog box to change the value of BAR0 to xE000000C as shown in Figure 63. Click Write ConfReg and then Refresh Dump. The new value of BAR0 is displayed.
Figure 64 is XMD output which shows that BAR0 has been written as 0xE000000C. The XMD `mrd` command also shows that the data in the initial 8 addresses in XPS BRAM is 0x00000000.

As noted earlier, the XMD displays data in Big Endian format, while the x86 displays data in Little Endian format.

Figure 64: XMD showing the Configuration Space Header, XPS BRAM
Figure 65 shows the memory test for PciTree. To run the memory test, click on **mem test** at the lower left of the BAR Space GUI. Check **Auto Read Memory** at the top of the BAR Space GUI to display memory values in the left side of the display. To edit a memory location, highlight the location to be edited, and enter the value in the **edit memory** dialog box. Click on **Write Memory**. To view the results, click on the **refr. view** icon.

![BAR space GUI](image)

**Figure 65**: Running PciTree Memory Test
Figure 66 shows the results of running the memory test. The leftmost column shows the count pattern used for data. The count increments for even addresses and decrements on odd addresses. With the PciTree read of BAR0, the data is the count value specified in the PciTree memory test. The results No Errors!! are provided.

Figure 66: PciTree Memory Test Results

The ML507 memory written/read is the BRAM and/or DDR2 defined in the system.mhs and addressed with the PLBv46 Endpoint Bridge C_PCIBAR2IPIFBAR_* generics. In this reference system, two PLBv46 Endpoint Bridge BARs are active. The C_PCIBAR2IPIFBAR_0 generic points to the ML507 BRAM located at 0xFFFF0000.

After writing the ML507 BRAM using PCI tree Edit Memory, XMD can be used to verify BRAM (or DDR2 if the BAR is enabled) from the PLBv46 side.
Figure 67 shows XMD verification that the XPS BRAM contains the data written by PciTree using XMD commands.

![Command Prompt - xmd]

Figure 67: XMD Verification of PciTree Write Operation

In the next two figures, XMD is used to write XPS BRAM, which is then read by PciTree.
Figure 68 shows the writing and reading of 0x12345678 to the first four locations in XPS BRAM.

![Figure 68: Writing XPS BRAM Using XMD](image)
Figure 69 shows a PciTree read of XPS BRAM. The first four locations are read as 0x12345678. The other memory locations retain the values shown in Figure 66.

**Memory Endpoint Test**

The Memory Endpoint Test (MET) is run on a PC with the ML507 Evaluation Platform inserted into a PCIe slot. MET provides a simple method of writing and reading memory. Like PciTree, the ML507 memory written/read is the BRAM and/or DDR2 defined in the system.mhs, and addressed with the PLBv46 Endpoint Bridge C_PCIBAR2IPIFBAR_* generics.

The MET requires the installation of the Xilinx Virtex-5 Endpoint Driver for PCI Express. The Xilinx application note XAPP1022 Using the Memory Endpoint Test (MET) Driver with the Programmed Input/Output (PIO) Example Design for PCI Express Endpoint Cores provides instructions on setting up and running the MET. XAPP1022 uses the PCIe Endpoint Block Plus core driven by the PIO interface. This section uses MET to write and read ML507 memory using the PLBv46 Endpoint Bridge.

XAPP1022 provide instructions for installing the Xilinx Virtex-5 PCIe Endpoint Driver.
Figure 70 shows the invocation of the Memory Endpoint Test. The values for the Device Number, Vendor Number and the address indicate that the PLBv46 Endpoint Bridge on the ML507 Evaluation Platform is detected.

```
H:\met>met
MET v2.4
Status request:
OK-1
Target report:
OK=0x1
Vendor=0xe10ce
Device=0x505
Bus#=0x0
DevH=0x0
MemAddress=0xe0000000
MemSize=0x10000
IOAddress=0x0
IOSize=0x0
IsPc1Express=TRUE
Interactive mode
MEM:32:Hex:00000000>
```

Figure 70: Invoking the Memory Endpoint Test

XAPP1022 provide detailed instructions on using the MET to test transfers to PLBv46 Endpoint Bridge memory.
Figure 71 shows basic read and write operations using the MET. In the figure, the Display (d), Location (l), and Set (s) instructions illustrate basic memory read and write transactions.

The command
\[ d \ 40 \]
causes the values of 40 current memory locations to be displayed. The values displayed (00000000 FFFFFFFF 00000002 FFFFFFFD ...) are the same as the values displayed by PciTree in Figure 66 because this test was run shortly after the PciTree tests.

The location command
\[ l \ 0 \]
moves the address to location 0x00000000. All addresses are offset addresses from the BAR start address.

The set command
\[ s \ 12345678 \]
is a memory write to the current address. In the figure, after the write of 0x12345678, the address pointer is move back to location 0x00000000 (l 0), and the contents of the memory is re-displayed using d 40. The 0x12345678 value just written at location 0x00000000 is displayed.
ChipScope is used to debug hardware problems. Debugging is done at either the system or PLBv46 Bridge for PCI Express level. To analyze PLBv46 Endpoint Bridge internal signals, insert the ChipScope cores into implementation/pcie_bridge_wrapper.ngc. To analyze signals involving multiple cores, insert the ChipScope cores into system.ngc. The flow for using the two debugging methods differs. Below, an outline of the steps for debugging at the system level is provided. This is followed by a detailed list of steps for debugging at the core level.

**Inserting ChipScope Cores at the System Level**

The following steps insert the ChipScope cores into the system.

1. In XPS, select **Hardware → Generate Netlist**.
2. From the command prompt in the implementation directory, run
   ```bash
   ngcbuild -i system.ngc system2.ngc
   ```
3. Invoke ChipScope Inserter. From the `ml507_ppc440_plbv46_pcie/chipscope` directory, open `ml507_ppc440_plbv46_pcie.cdc`. Figure 72 shows the input file `system2.ngc` in the **Input Design Netlist** window. Define the Clock, Trigger, and Data signals in Inserter, and generate the ICON and ILA cores.
4. From `ml507_ppc440_plbv46_pcie/implementation`, copy the file displayed in the Inserter Output Design Netlist window, usually `implementation/system2.ngo`, to `implementation/system.ngc`.

![Figure 72: Inserting an ILA into a System](image-url)
5. In XPS, run Hardware → Generate Bitstream.

Inserting ChipScope in the PLBv46 Endpoint Bridge

The ml507_ppc440_plbv46_pcie/chipscope/plbv46_pcie.cdc file is used to insert a ChipScope ILA core into the pcie_bridge_wrapper core. Use the following steps to insert an ILA core and analyze PLBv46 Endpoint Bridge signals with the ChipScope tool.

1. Invoke XPS. Run Hardware → Generate Netlist.

3. Run Start → Programs → ChipScope Pro → ChipScope Inserter

4. From ChipScope Inserter, run File Open → plbv46_pcie.cdc.
Figure 73 shows the ChipScope Inserter setup GUI after running
File Open → plbv46_pcie.cdc.

![ChipScope Inserter Setup GUI](image)

**Figure 73:** Opening plbv46_pcie.cdc
5. The signals defined in `plbv46_pcie.cdc` provide a good starting point for analyzing designs. In most analyses, additional nets are needed. Figure 74 shows the GUI for making net connections. Click **Next** four times to move to the Modify Connections window. Select Modify Connections. The Filter Pattern is used to find net(s). As an example of using the Filter Pattern, enter `*ack*` in the dialog box to locate acknowledge signals such as `Sl_AddrAck`. In the Net Selections area, select either Clock, Trigger, or Data Signals. Select the net and click **Make Connections**.

Correct Clock, Trigger, and/or Data signals displayed in red in the Net Selections pane.

![Figure 74: Inserter Data Signals](image)

6. Click **Insert** to insert the ILA into `pcie_bridge_wrapper.ngo`. From the `ml507_ppc440_plbv46_pcie/implementation` directory, copy `pcie_bridge_wrapper.ngo` to `pcie_bridge_wrapper.ngc`.

7. Remove the `ml507_ppc440_plbv46_pcie/implementation/cache/pcie_bridge_wrapper.ngc` file.

8. In XPS, run **Hardware** → **Generate Bitstream** and **Device Configuration** → **Download Bitstream**. Do not rerun **Hardware** → **Generate Netlist**, as this overwrites the `implementation/pcie_bridge_wrapper.ngc` produced in the step above. Verify that the file size of the `pcie_bridge_wrapper.ngc` with the inserted core is significantly larger than the `pcie_bridge_wrapper.ngc` file which did not have the ILA inserted.

9. Invoke ChipScope Pro Analyzer by selecting **Start** → **Programs** → **ChipScope Pro** → **ChipScope Pro Analyzer**.
Click on the Chain icon located at the top left of the Analyzer GUI. Verify that the INFO message in the transcript window indicates that a core unit is found.

10. The ChipScope Analyzer waveform viewer displays signals named DATA*. To replace the DATA* signal names with the familiar signal names specified in ChipScope Inserter, select File → Import and browse to plbv46_pcie.cdc in the dialog box.

The Analyzer waveform viewer is more readable when buses rather than discrete signals are displayed. Select the SI_rdDBus<> signals, click the right mouse button, and select Add to Bus → New Bus. With SI_rdDBus in the waveform viewer, select and delete the discrete SI_rdDBus<> signals. The signals are displayed as buses in Figure 75.

**Note:** The Reverse Bus Order operation is useful for analyzing buses in the Analyzer Waveform Viewer.

---

**Figure 75:** ChipScope Pro Analyzer Waveform
11. Set the trigger in the Trigger Setup window as shown in Figure 76. The trigger used depends on the problem being debugged. Simple triggers are PA_Valid, Sl_AddrAck, Sl_wrComp. The trigger defined below is a 1 on Sl_addrAck.

Figure 76: ChipScope Analyzer Trigger Setup
12. Arm the trigger by selecting **Trigger Setup → Arm**, or clicking on the **Arm** icon as shown in **Figure 77**.

![Figure 77: ChipScope with Trigger Armed](image)

13. Run **XMD** or **GDB** to trigger patterns which cause ChipScope to display waveform output. For example, set the trigger to **Sl_addrAck**, arm the trigger, and run

```
xml -tcl xmd_commands/dma.tcl
```

at the command prompt. This produces signal activity in the Analyzer waveform viewer.
14. ChipScope results are analyzed in the waveform window, as shown in Figure 78. This figure shows the bus signals generated in step 10.

Figure 78: ChipScope Pro Analyzer Triggered

To share the results with remote colleagues, save the results in the waveform window as a Value Change Dump (vcd) file. The vcd files can be translated and viewed in most simulators. The vcd2wlf translator in ModelSim reads a vcd file and generates a waveform log file (wlf) file for viewing in the ModelSim waveform viewer. Select File → Open Database to open the vcd file in the Cadence Design System, Inc. Simvision design tool.

After running ChipScope, it is sometimes necessary to revise the Trigger or Data nets, or both, used in a debug operation. Saving Inserter and Analyzer projects simplifies this procedure. The saved project can be re-opened in Inserter, and edits can be made.
Figure 79 shows the waveform output of a ChipScope inserted into the reference system when running the endpoint to root complex performance tests.

Memory, XPS Central DMA, and PLBv46 Bridge for PCI Express transactions are monitored simultaneously. The trigger is PCIe_bridge/comp_slave_bridge/sig_request_complete. The ml507_ppc440_plbv46_pcie_scs.cdc is included in the chipscope directory.

![System Debugging Using ChipScope](image)
The reference design matrix is shown in Table 4.

**Table 4: Reference Design Matrix**

<table>
<thead>
<tr>
<th>General</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer Name</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Target devices (stepping level, ES, production, speed grades)</td>
<td>Virtex-5 xc5vfx70t (Production Silicon)</td>
</tr>
<tr>
<td>Source code provided</td>
<td>No</td>
</tr>
<tr>
<td>Source code format</td>
<td>VHDL/Verilog</td>
</tr>
<tr>
<td>Design uses code/IP from an existing reference design/application note, 3rd party, or CORE Generator software</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional simulation performed</td>
<td>No</td>
</tr>
<tr>
<td>Timing simulation performed</td>
<td>No</td>
</tr>
<tr>
<td>Testbench used for functional simulations provided</td>
<td>No</td>
</tr>
<tr>
<td>Testbench format</td>
<td>N/A</td>
</tr>
<tr>
<td>Simulator software used/version (i.e., ISE software, Mentor, Cadence, other)</td>
<td>N/A</td>
</tr>
<tr>
<td>SPICE/IBIS simulations</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis software</td>
<td>XST</td>
</tr>
<tr>
<td>EDK Software</td>
<td>EDK10.1.03</td>
</tr>
<tr>
<td>Implementation software tools used/versions</td>
<td>ISE10.1.03</td>
</tr>
<tr>
<td>Static timing analysis performed</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware Verification</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware verified</td>
<td>Yes</td>
</tr>
<tr>
<td>Hardware platform used for verification</td>
<td>ML507</td>
</tr>
</tbody>
</table>

**References**

1. [UG197](#) Virtex-5 FPGA Integrated Endpoint Block for PCI Express Designs User Guide
2. [UG341](#) LogiCORE Endpoint Block Plus for PCI Express User Guide
3. [DS540](#) PLBv46 Bridge for PCI Express (v3.00.a)
5. [XAPP1022](#) Using the Memory Endpoint Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores
6. [XAPP859](#) Virtex-5 FPGA Integrated Endpoint Block for PCI Express Designs: DDR2 SDRAM DMA Initiator Demonstration Platform
8. LeCroy PCI Express Multi-Lane Exerciser User Manual Version 5.0
Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/05/09</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

Notice of Disclaimer

Xilinx is disclosing this Application Note to you “AS-IS” with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.