XAPP998 (v1.0) February 7, 2008

PCI Bus Performance Measurements using the Vmetro Bus Analyzer

Author: Lester Sanders

Abstract

This application note illustrates how to measure performance using the Vmetro Vanguard PCI Bus analyzer. These measurements use a system in which the ML410 Evaluation Platform, the ML555 Evaluation Platform, and the Vmetro Vanguard-PCI (VG-PCI) boards communicate over the PCI bus. The test method is provided so that similar tests can be done using different PCI bus transactions, different boards, and/or other Xilinx PCI cores.

This application note outlines how to set up the three boards, and how to use the ChipScope™ and Vmetro logic analyzers to analyze PCI bus transactions. The bus transactions used in performance tests are defined. The logic analysis is used to verify that the data transferred in the transactions are correct. Three methods of generating the stimuli for the performance tests are given. The use of the Vmetro BusView software to generate performance statistics is described and the results are provided.

Included Systems

Two reference systems, ml410_ppc_plbv46_pci_cs and ml555_mb_plbv46_pci_cs, built for the Xilinx ML410 Rev E and ML555 Rev A boards, are provided in the link below.

www.xilinx.com/support/documentation/application_notes/xapp998.zip

The ml410_ppc_plbv46_pci_cs reference system is a Base System Builder design with `ready_for_download`, `chipscope`, `waveforms`, `xmd_commands` directories added. The `ready_for_download` directory contains the `download.bit` and `xapp998.cmd` files for quickly downloading the bit file. The `chipscope` directory contains the `ml410_ppc_plbv46_pci.cdc` for use with ChipScope. The `waveforms` directory contains vcd and wlf files used in latency analysis. The `xmd_commands` directory provides tcl scripts used for DMA transactions.

Introduction

Xilinx offers OPB PCI, PLB PCI, and PLBv46 PCI cores, in several FPGA families. PCI boards manufactured by Xilinx, Avnet, and Enterpoint can use the Xilinx PCI cores. This application note provides a test method for measuring PCI performance using the Vmetro Vanguard PCI (VG-PCI) Bus Analyzer and BusView software.

A section is provided on board setup for the performance tests. The ML410 and ML555 Evaluation Platform designs are provided, so the specifics of the reference systems are not included in this application note. If further information is needed, the "References" section at the end of this application note lists reference systems for several boards.

The "Using Logic Analyzers for Verification and Latency Measurements" section discusses measurements using Xilinx ChipScope and Vmetro BusView software. The Generating Stimuli section discusses the generation of stimuli using C code and xmd commands. When the Vmetro VG-PCI is used as the initiator, the BusView Exerciser is used to generate stimuli. An introduction to using BusView Exerciser is provided.

The BusView Statistics Function generates statistics on transfer rate, bus utilization, and other predefined statistical measurements. The setup and use of the Statistics Function is described.

The Results section provides bus transfer rate and latency results for the OPB PCI, PLB PCI, and PLBv46 PCI cores.

Hardware and Software Requirements

The hardware and software requirements for this application note are listed below.

- Xilinx ML555 Rev A board
- Xilinx ML410 Rev E board
- Two Xilinx Platform USB or Parallel IV programming cables
- RS232 null modem serial cable and serial communication utility (HyperTerminal, TeraTerm, Minicom)
- Xilinx Platform Studio EDK9.2.02i
- Xilinx Integrated Software Environment (ISE™) 9.2.04i
- Vmetro Vanguard PCI Board
- Vmetro BusView 5.12 Software
- Xilinx ChipScope 9.2 Software

System Specifics

This system uses the ML410 Evaluation Platform, ML555 Evaluation Platform, and Vanguard PCI (VG-PCI) boards communicating over a 33 MHz PCI bus. The ML410 uses a PPC405 running at 300 MHz and includes a PLBv46 PCI configured to operate in a Virtex™-4 FPGA. The ML555 uses a MicroBlaze™ processor running at 75MHz and a PLBv46 PCI configured to operate in a Virtex-5 FPGA.

Figure 1 is a photograph of the Xilinx ML410 Evaluation Platform showing the Vmetro Bus Analyzer in PCI slot P5 and the Xilinx ML555 Evaluation Platform in PCI slot P3.

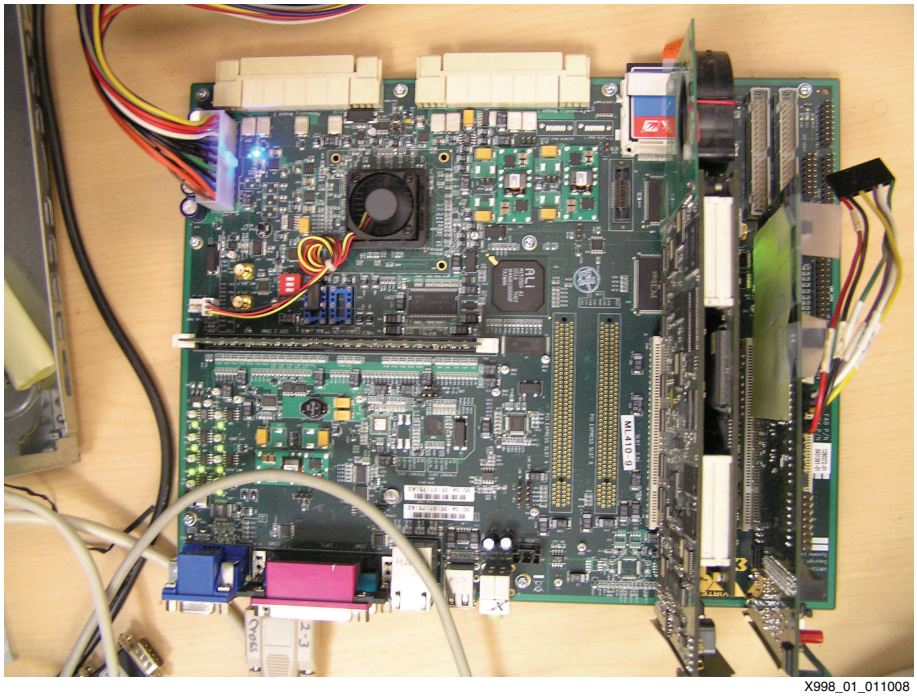


Figure 1: **ML410 Evaluation Platform**

The Vmetro Vanguard PCI board is inserted into ML410 PCI slot P5. It connects to the PC using a USB cable, interfacing to the Vmetro BusView software. The PC connects to the ML410 using the Xilinx Platform USB programming cable for communicating with Impact, XMD, and ChipScope. The serial communication cable between the PC and ML410 allows print statements in the C code to be displayed on a communication terminal such as HyperTerminal, Teraterm, or Minicom.

In this setup, a second PC is used to configure and monitor the ML555 FPGA. An alternative to using a second PC is to disconnect the Platform USB programming cable from the ML410 and connect it to the ML555. A second reason a second PC is used is to interface to the ChipScope inserted core in the ML555 FPGA.

The second PC is not necessary for analyzing PCI transactions. If ChipScope is not used to debug the ML555 FPGA, the second PC is not needed. The ChipScope core inserted in the ML410 FPGA provides access to the PCI bus which is driven by the ML555. It does not provide access to internal ML555 FPGA internal signals, which can only be monitored using a ChipScope core inserted into the ML555 FPGA.

The PC connection to the ML555 FPGA also allows XMD verification of ML555 memory to be run without changing the Xilinx Platform USB cable connection.

Figure 2 is a system diagram of the board setup used for the test.

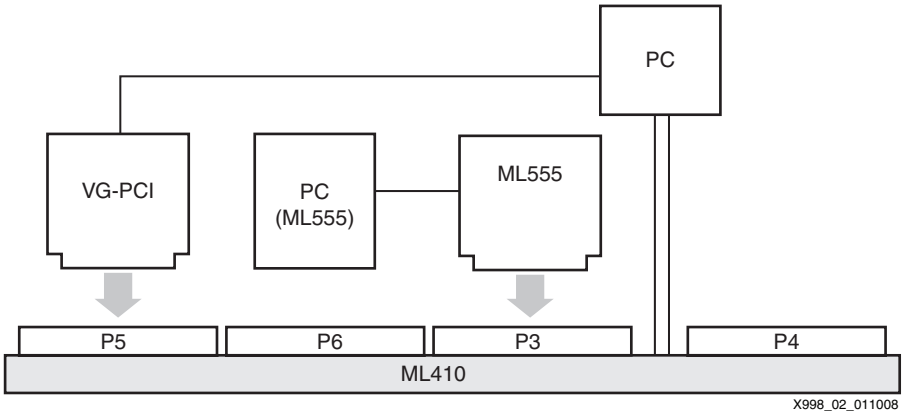


Figure 2: **System Setup**

ML410 and ML555 Address Maps and Generics

The ML410 Evaluation Platform and the ML555 Evaluation Platform each contain BRAM and DDR memory. The PCI transactions measured in this application note are across the PCI bus, so the transaction is from one board, which is the initiator, to another board, the target. An example is a DMA transaction from ML410 BRAM to ML555 DDR2. Generics are parameters that are used to define the PLBv46 PCIe Bridge functionality. The 32-bit/33 MHz PCI bus transactions can occur within a single memory, using different address ranges, or between different memory.

Figure 3 provides the signal flow for PCI transactions between the ML410 and ML555 memory.

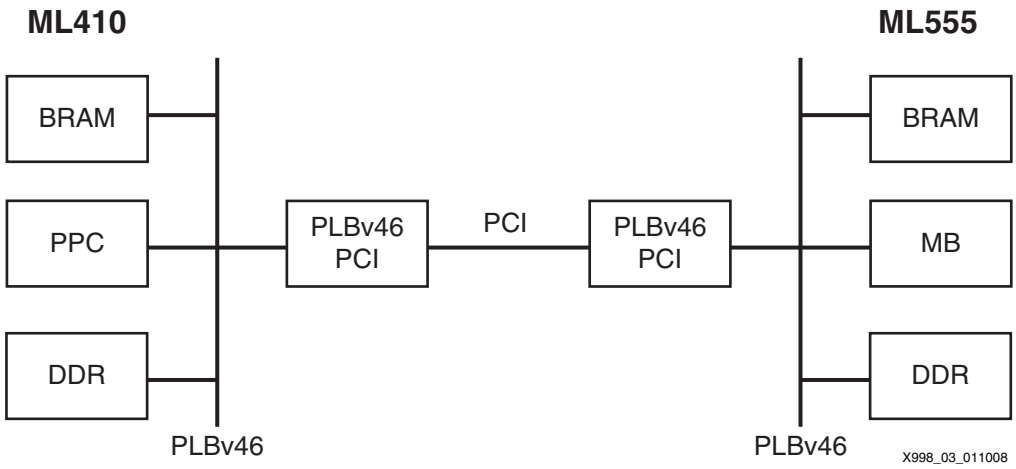


Figure 3: **PCI Transactions Signal Flow**

Table 1 provides the address map and generics for the ML410 FPGA.

Table 1: **ML410 Addresses and Generics**

Parameter	Value
BRAM Address	0xFFFF0000
DDR Address	0x00000000
C_IPIFBAR0	0x20000000
C_IPIFBAR2PCIBAR0	0x80000000
C_IPIFBAR1	0xE8000000
C_IPIFBAR2PCIBAR1	0x90000000
C_PCIBAR2IPIFBAR0	0xFFFF0000
C_PCIBAR2IPIFBAR1	0x00000000

Table 2 provides the address map and generics for the ML555 FPGA.

Table 2: **ML555 Addresses and Generics**

Parameter	Base Address
BRAM Address	0x8AE08000
DDR Address	0x90000000
C_IPIFBAR0	0x20000000
C_IPIFBAR2PCIBAR0	0x60000000
C_IPIFBAR1	0x30000000
C_IPIFBAR2PCIBAR1	0xA0000000
C_PCIBAR2IPIFBAR0	0x8AE08000
C_PCIBAR2IPIFBAR1	0x90000000

- If the PLBv46 PCI in the ML410 is the initiator, the PLBv46 PCI in the ML555 and VG-PCI are targets.
- If the VG-PCI is the initiator, the PLBv46 PCI cores in the ML410 and ML555 are targets.
- If the PLBv46 PCI in the ML555 is the initiator, the VG-PCI and ML410 PLBv46 PCI are targets.

Transactions may be initiated by the microprocessor or a DMA Controller, or by the Vmetro BusView Exerciser. Transactions initiated by a microprocessor or DMA Controller are done using XMD commands or C code.

With address translations, the bus transactions used in performance tests involve different address buses - ML410 PLBv46, PCI, and ML555 PLBv46 address buses. The ChipScope logic analyzer is used to validate bus transactions and to determine latency. To use ChipScope effectively, an understanding of the address translations for transactions using the ML410 PLBv46 PCI and ML555 PLBv46 is needed.

Address Translation in PCI Transactions

In PCI bus transactions, the Base Address Registers in the Configuration Space Headers (CSHs) are used in accessing a memory on the board, BRAM, or DDR. The address translation for a PCI transaction from the ML410 to the ML555 is described below for the address range defined by the ML410 C_IPIFBAR_0. Referencing Table 1 and Table 2, the ML410 processor writes an address to the C_IPIFBAR_0 range on the PLBv46 address bus, 0x20000000. This is translated to a PCI address by the PLBv46 PCI core defined by the C_IPIFBAR2PCIBAR_0

generic, 0x80000000. For the ML555 PLBv46 PCI to decode this PCI bus transaction, the ML555 PLBv46 PCI CSH BAR_0 value is written as 0x8000000. The CSH is written using XMD commands in a tcl script as discussed later. The ML555 generic C_PCIBAR2IPIFBAR_0 address is 0x8AE08000. The ML555 system.mhs specifies a C_BASEADDR for XPS BRAM of 0x8AE08000.

Writing the Configuration Space Headers

After configuring the FPGAs in the ML410 and ML555 using Impact, the configuration space headers (CSHs) in the PCI cores and VG-PCI are written by the initiator. The CSH registers written are the Command Status Register, the Latency Timer, and the PCIBAR0 and PCIBAR1. If the ML410 PLBv46 PCI is the initiator, CSHs are written with the xmd command

```
xmd -tcl wr_csh_410_555_vm.tcl
```

The wr_csh_410_555_vm.tcl script, provided in the ml410_ppc_plbv46_pci_cs/ready_for_download directory, writes the registers in the M410, ML555, and VG-PCI Configuration Space Headers (CSH). The wr_csh_410_555_vm.tcl script outputs csh.txt which is used to verify that the CSHs are set up correctly.

If the PLBv46 PCI in the ML555 is used as the initiator, wr_csh_410_555_vm.tcl can be revised to write the CSHs of the ML555, ML410, and VG-PCI. Additional information on the CSHs is described in *XAPP964 Reference System: OPB PCI in a ML410 Evaluation Platform* and *XAPP999 Reference System: PLBv46 PCI in a ML555 Evaluation Platform*.

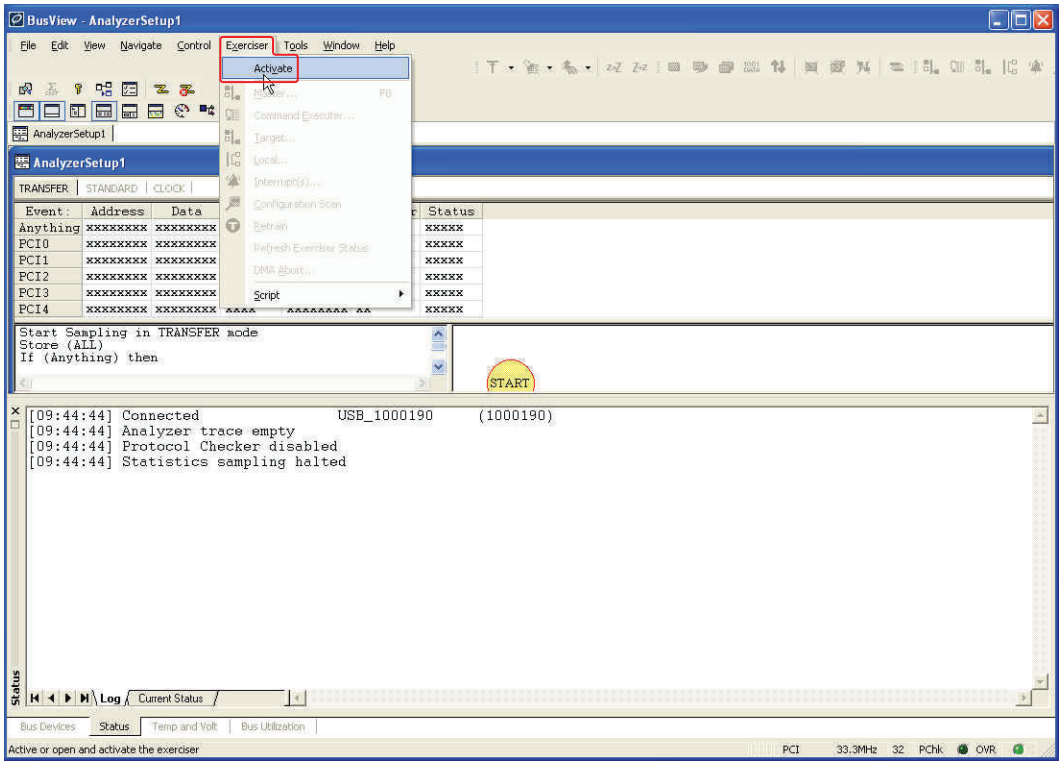
When the VG-PCI is used as a master, the ML410 and ML555 CSH headers can be configured using the xmd script, but because the standard approach is to use the initiator for this function, the following sections illustrate the BusView Exerciser scan and Configuration Write functionality.

Verifying the Configuration Space Headers

The csh.txt provides values of some of the more important registers in the CSHs. BusView provides a graphical interface for verifying that the ML410 PLBv46 PCI, the ML555 PLBv46 PCI, and the VG-PCI Configuration Space Headers are set up correctly. The BusView Exerciser configuration scan command is used.

The scan is used to verify that PCIBAR registers at offsets 10, 14, 18 in the CSH are correct.

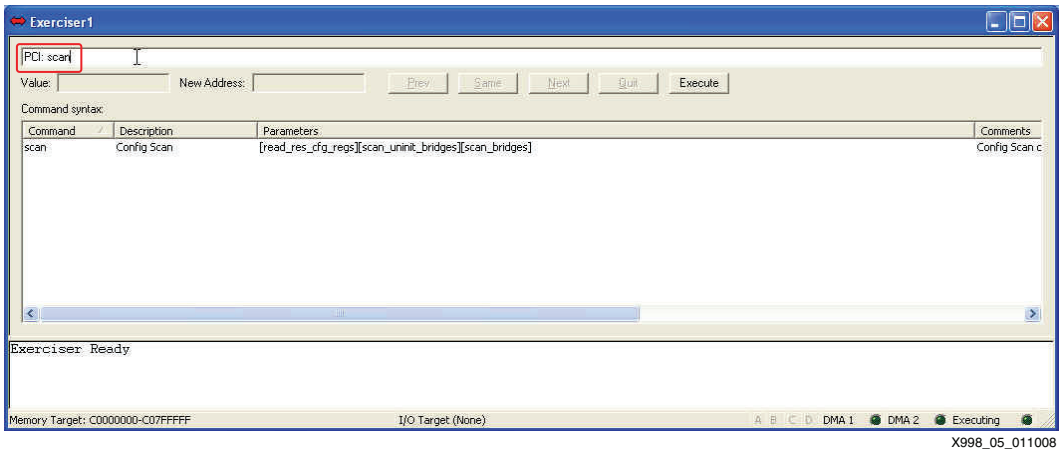
Figure 4 shows the setup of a BusView scan. Invoke **`BusView`**, then in the BusView window menu, select **Exerciser** → **Activate**.



X998_04_011008

Figure 4: **CSH Scan Setup**

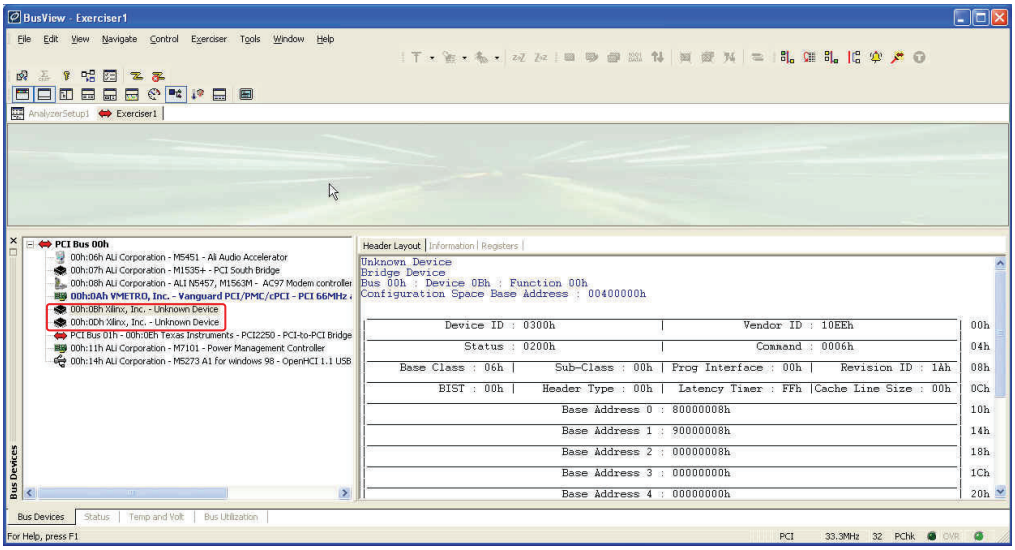
Figure 5 shows the execution of the BusView Master scan command. In the BusView Exerciser1 command window, enter **`scan`** in the PCI field.



X998_05_011008

Figure 5: **Executing BusView Scan Command**

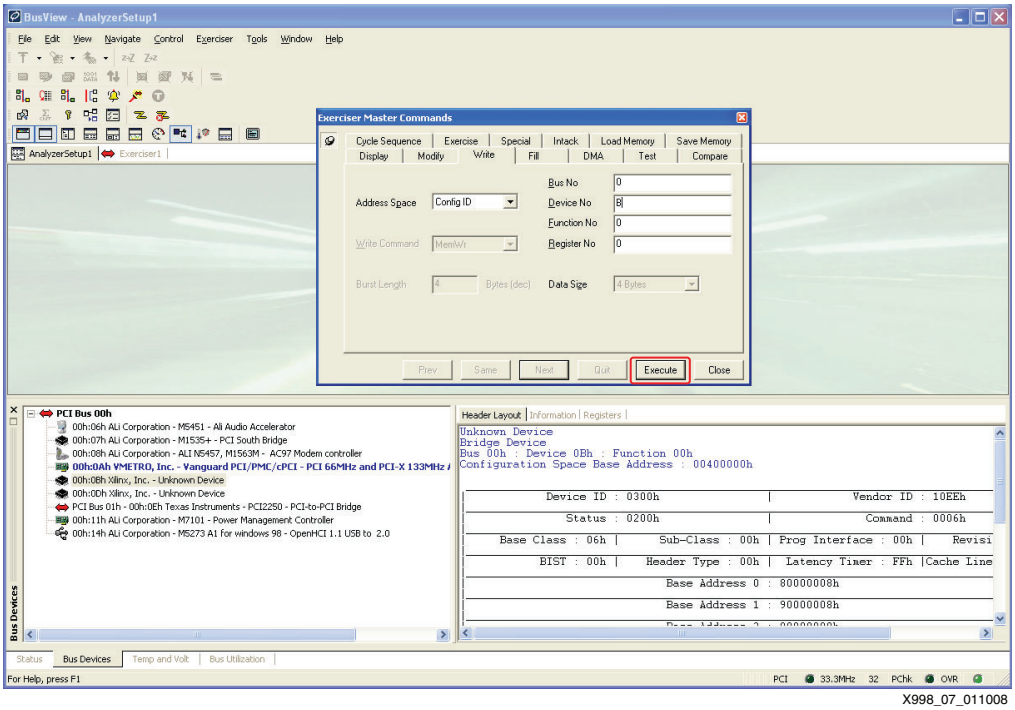
Figure 6 shows the output of a scan of the Configuration Space Headers. The left window shows the PCI devices recognized in the PCI scan. When a Bus:Device:Function (BDF) entry in the left window is selected, the values of CSH registers of the selected BDF are displayed in the right window. In the figure, the CSH register values for ML555 PLBv46 PCI at BDF 0 B 0are displayed. The Device ID and Vendor ID are determined by PLBv46 PCI generics, and the Command Status Register, Latency Timer, and BARs are configured by the `wr_csh_410_555_vm.tcl` script.



X998_06_011008

Figure 6: **Scanned CSH**

Figure 7 shows how BusView is used to write the Configuration Space Header. This window defines the ML555 PLBv46 PCI BDF as 0 B 0. Within the ML555 PLBv46 PCI CSH, Base Address Register (offset 10H) is written. After clicking on Execute, BusView provides a prompt for the value.



X998_07_011008

Figure 7: **Writing the CSH from BusView**

Using Logic Analyzers for Verification and Latency Measurements

Prior to running a performance test, XMD, ChipScope, or BusView is used to verify that the data transmitted is the same as the data received. This section discusses verification using XMD commands, and the ChipScope and BusView Analyzer software. The use of ChipScope for latency measurements is also discussed.

Verification with XMD Commands

The easiest method of verification is to use XMD to verify that the source and target memory contain the same data. XMD commands are used to verify the functionality of the DMA transactions between the different memory regions. The `mem.tcl` is used to set up source memory with a known pattern, such as `0x12345678`, and also set the destination memory to `0x00000000s`. After the DMA transaction is run, the XMD **`mrd <destination>`** command is used to verify that the transaction is correct.

Verification with ChipScope

While BusView Analyzer has a larger buffer than ChipScope, only PCI signals are displayed. To analyze both PLBv46 and PCI bus signals, ChipScope is used.

ChipScope ICON and ILA cores are inserted into the `ml410_ppc_plbv46_pci_cs` and `ml555_mb_plbv46_pci` projects. The `ml410_ppc_plbv46_pci.cdc` and `ml555_plb_plbv46_pci.cdc` files imported into the ChipScope Analyzer provide recognizable names in the waveform viewer of the Chipscope Analyzer.

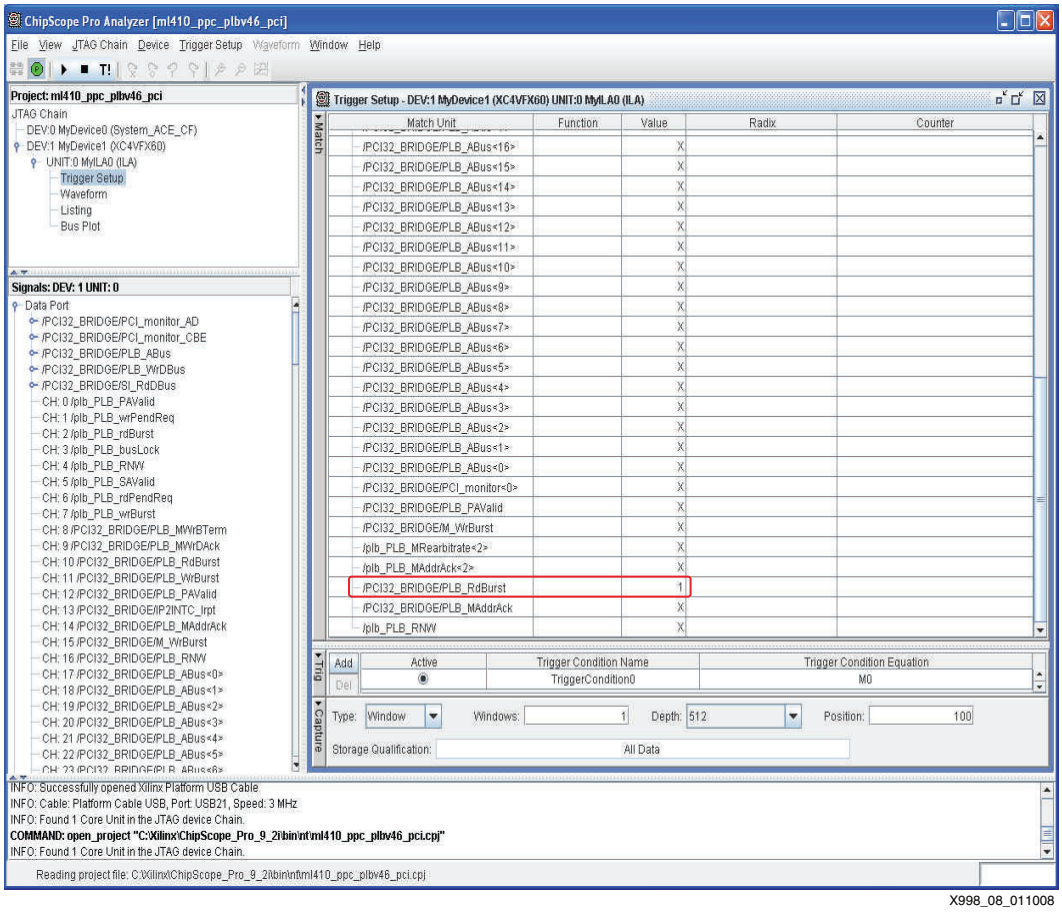
Follow the steps below to insert the ILA into the top level netlist, `system.ngc`, so that the signals from different cores and different buses are viewed simultaneously.

1. Invoke EDK. Execute **Hardware** → **Generate Netlist**.
2. At the command prompt, run **`ngcbuild -i system.ngc system2.ngc`**.
3. Invoke ChipScope Inserter. If using Linux, at the command prompt, run **`inserter.sh`**. From the Inserter GUI, select **File Open** → **ml410_ppc_plbv46_pci.cdc**. This cdc file sets the input file as `system2.ngc`, the output file as `system.ngo`, and the clock, trigger, and data signals. Insert the core.
4. Copy `implementation/system.ngo` to `implementation/system.ngc`.
5. From EDK, run **Hardware** → **Generate Bitstream**.

This procedure inserts an ILA into the `system.ngc`, thereby allowing the PLBv46, PCI, xps_central_dma, and PLBv46 PCI signals to be viewed concurrently. *XAPP964 Reference System: OPB PCI in a ML410 Evaluation Platform* and *XAPP999 Reference System: PLBv46 PCI in a ML555 Evaluation Platform* provide the steps for inserting the ILA into the OPB PCI, the PLBv46 PCI, `pci32_bridge_wrapper.ngc`, for debugging the PCI core.

Note: If it is necessary to debug ML555 FPGA signals, insert an ILA into the ML555 FPGA.

Figure 8 shows the setup for ChipScope Analyzer triggering on Rd_Burst. The Capture position is set to **100**.



X998_08_011008

Figure 8: **Setting ChipScope to Trigger on Rd_Burst**

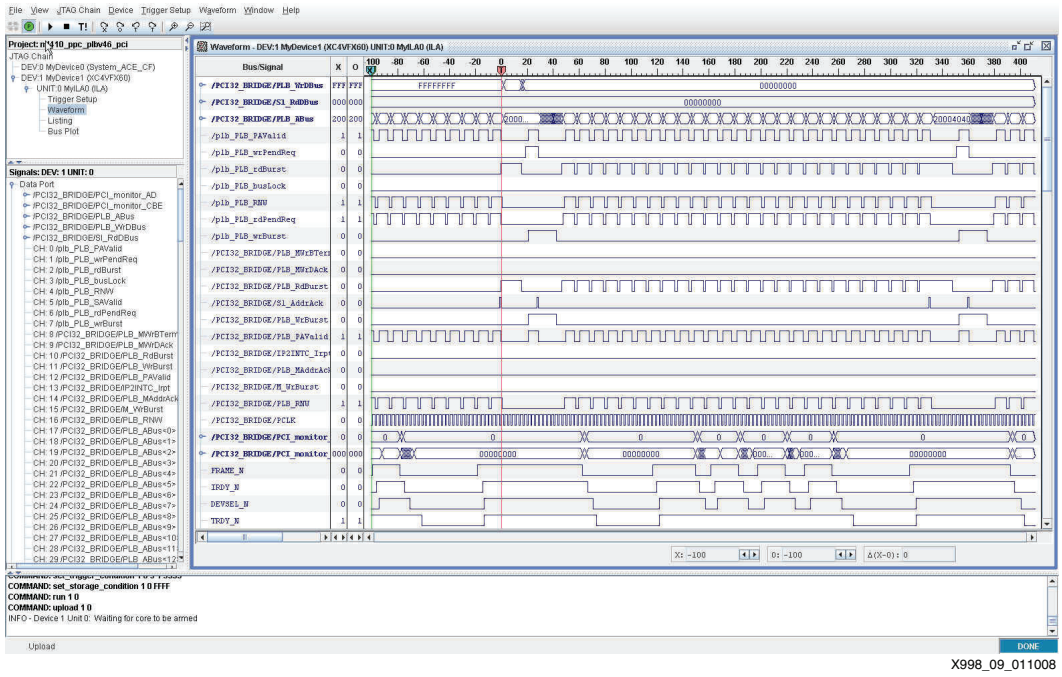
Note: For the tests which run in loops, re-triggering may be necessary to view multiple bus transactions.

Figure 9 shows the ChipScope trace after triggering on Rd_Burst. The PCI bursts are shown with FRAME_N Low for ~ 16 PCI clocks. This trace is triggered by running the command

```
xmd -tcl dma.tcl
```

The `dma.tcl` file, located in `ml410_ppc_plbv46_pci_cs/xmd_commands`, generates a DMA transaction from the source address to the destination address.

The resolution in the figures in this application note makes them difficult to read. The value change dump (vcd) and waveform log file (wlf) waveforms provided in the `ml410_ppc_plbv46_pci_cs/waveforms` directory are more readable than those in the figure. ModelSim SE 6.1e is used to view the waveforms.

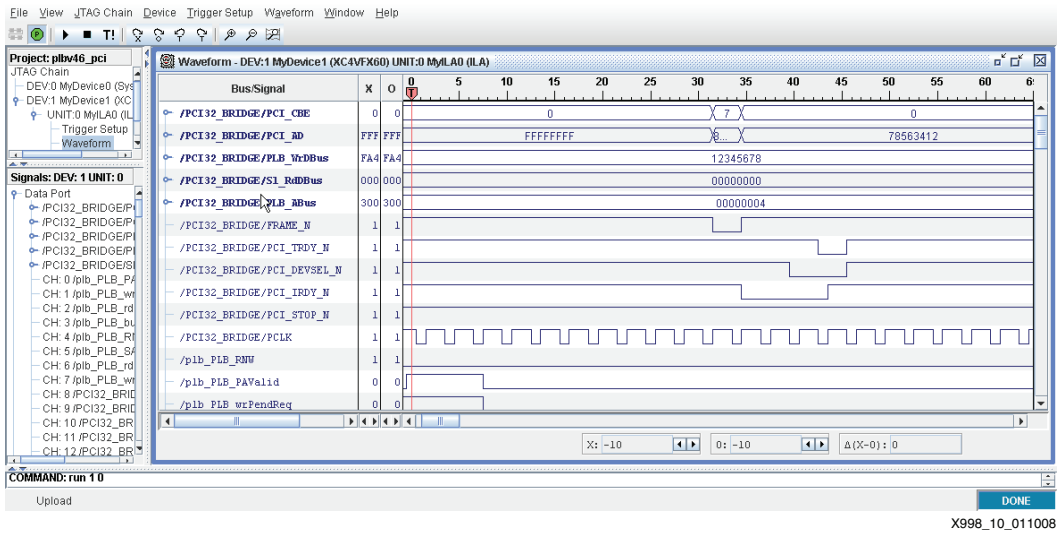


X998_09_011008

Figure 9: **ChipScope Trace**

In addition to verification, ChipScope is used to measure latency. Latency is the number of PCI clocks from the start to the end of the transaction. The wlf files in the `ml410_ppc_plbv46_pci_cs/waveform` directory can be used for latency measurement.

The write operation done by the XMD command, `mwr 0x2000000 0x12345678`, generates activity on the ML410 PLBv46 bus, which is followed by activity on the PCI bus, which is then followed by ML555 PLBv46 activity. PA_Valid is used to identify the start of the operation, and SL_WrDack is used to identify the completion of the data transfer. Figure 10 shows the latency of a memory write command.

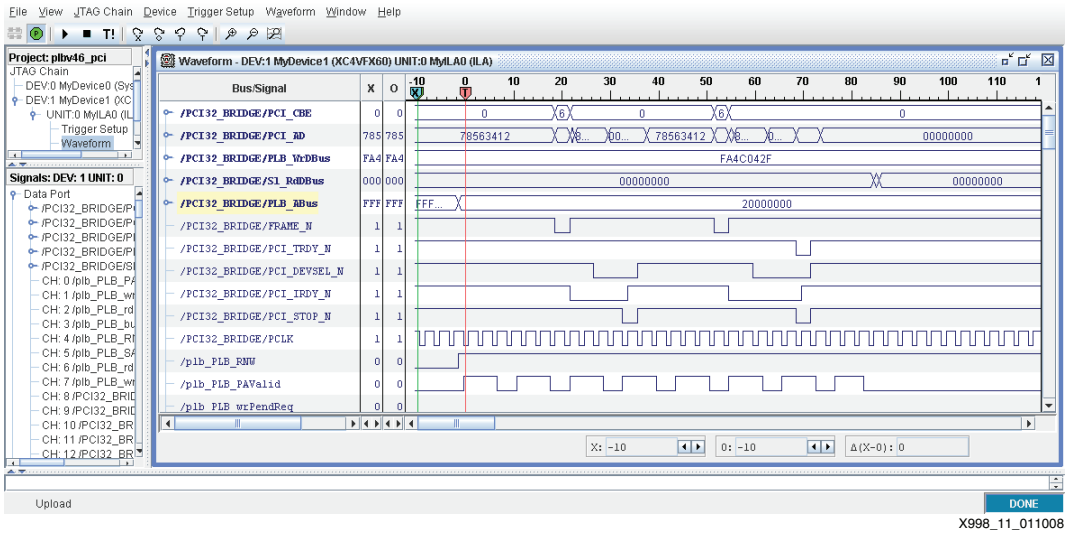


X998_10_011008

Figure 10: **Latency of Memory Write Operation**

The XMD read operation, `mrd 0x20000000 1`, generates activity on the ML410 PLBv46 bus, followed by activity on the PCI bus, which is then followed by ML555 PLBv46 bus activity. In the read operation, there a delay at the target for the read data on the PCI AD bus.

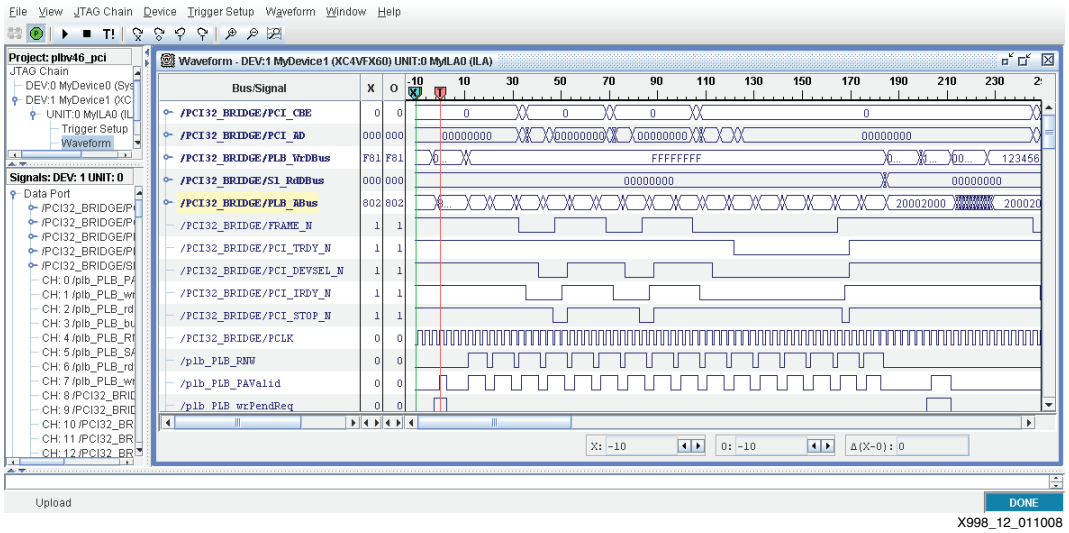
Figure 11 shows the latency of a memory read.



X998_11_011008

Figure 11: **Latency measurement of Memory Read**

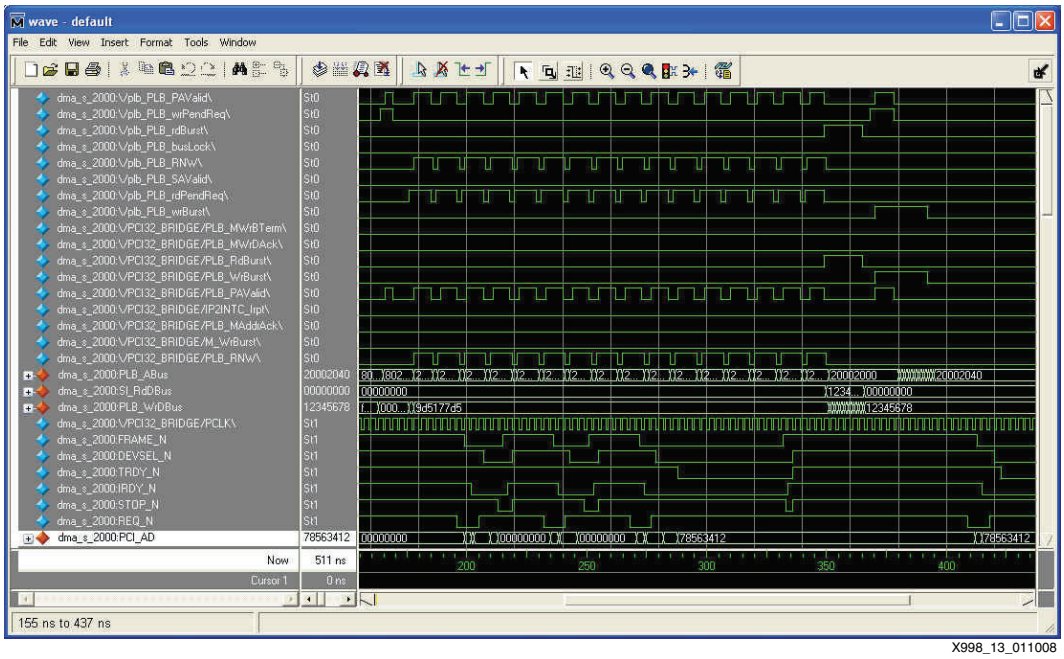
Figure 12 shows the latency of a DMA transaction.



X998_12_011008

Figure 12: **DMA Latency test**

Figure 13 shows the ModelSim waveform. When displayed in the ModelSim waveform viewer, the signals are easier to read than in the ChipScope waveform viewer.



X998_13_011008

Figure 13: **Latency Measurement of DMA Transaction**

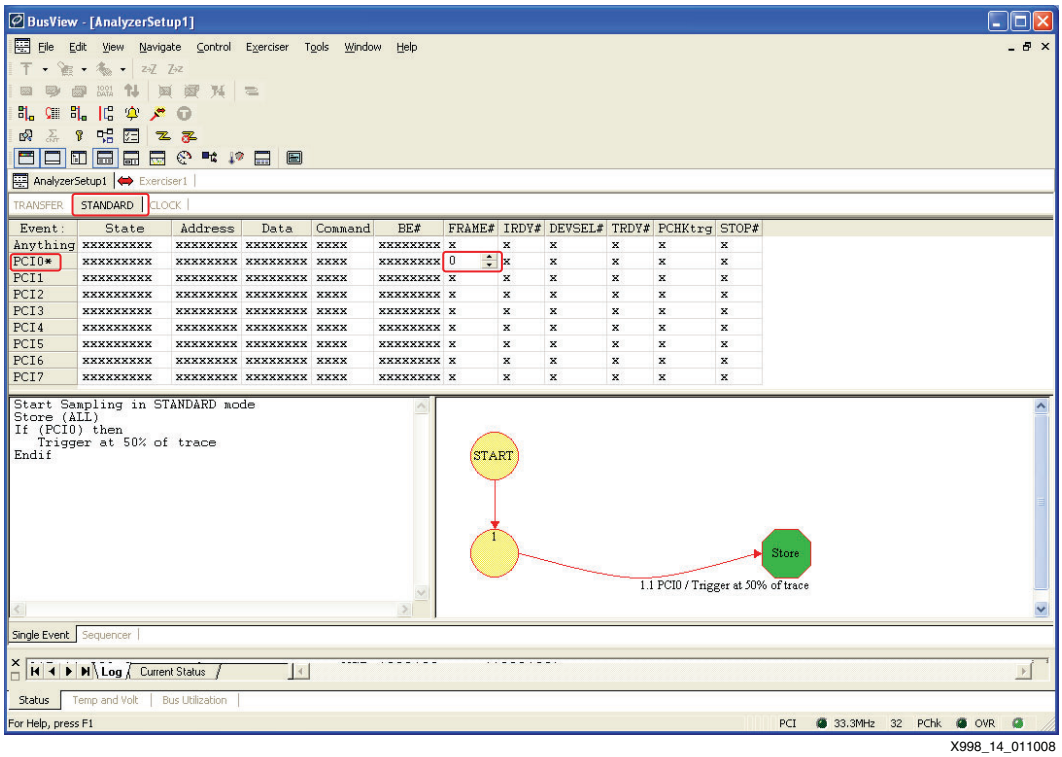
Verification with BusView Analyzer

The BusView Analyzer is easier and quicker to use than ChipScope, and it allows detailed analysis of PCI signals. It can be used for verification of many bus transactions.

The flow for BusView analysis is:

1. Set up BusView Analyzer
2. Run BusView Analyzer
3. Generate Stimuli

Figure 14 shows the setup of the BusView Analyzer. The three modes for the BusView Analyzer are Transfer, Standard, and Clock. Select the **Standard** mode. Specify the trigger in **PCI0** (second row) by scrolling through the FRAME# column and selecting **0**.



X998_14_011008

Figure 14: **BusView Analyzer Setup**

The Control pull-down in the BusView file menu controls the starting and stopping of the BusView analyzer and statistics functions.

Figure 15 shows running BusView Analyzer. Select **Control** → **Run** → **Analyzer** or the F9 shortcut. In this figure, the trigger has been changed to the MRdMul entry under the Command column.

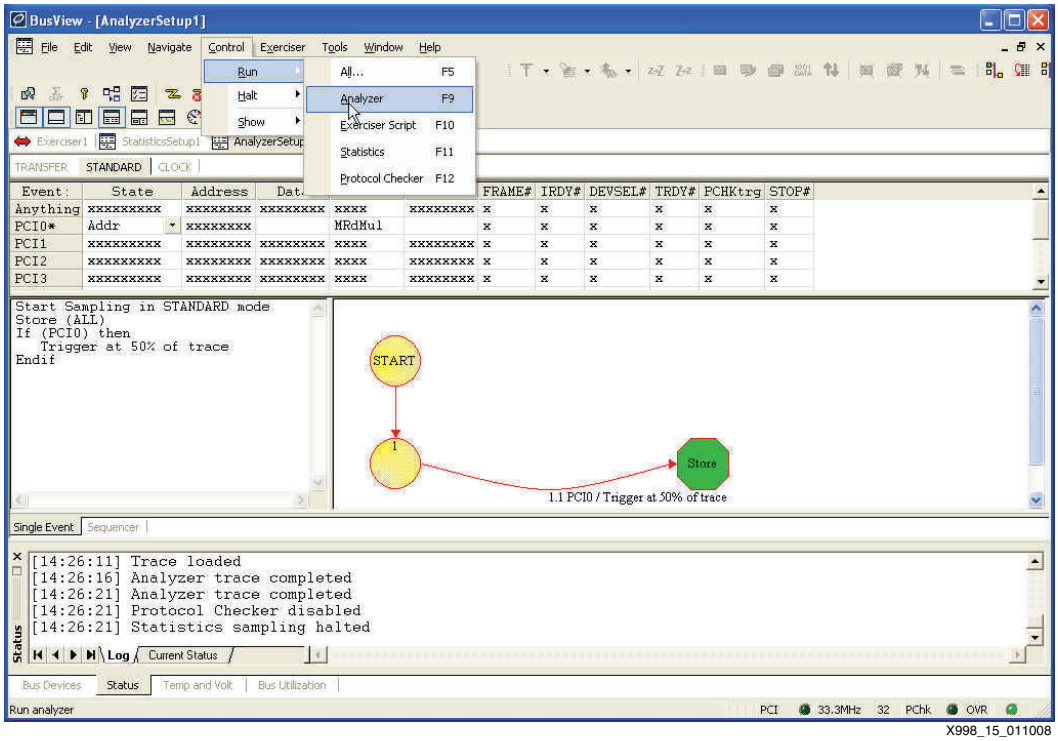
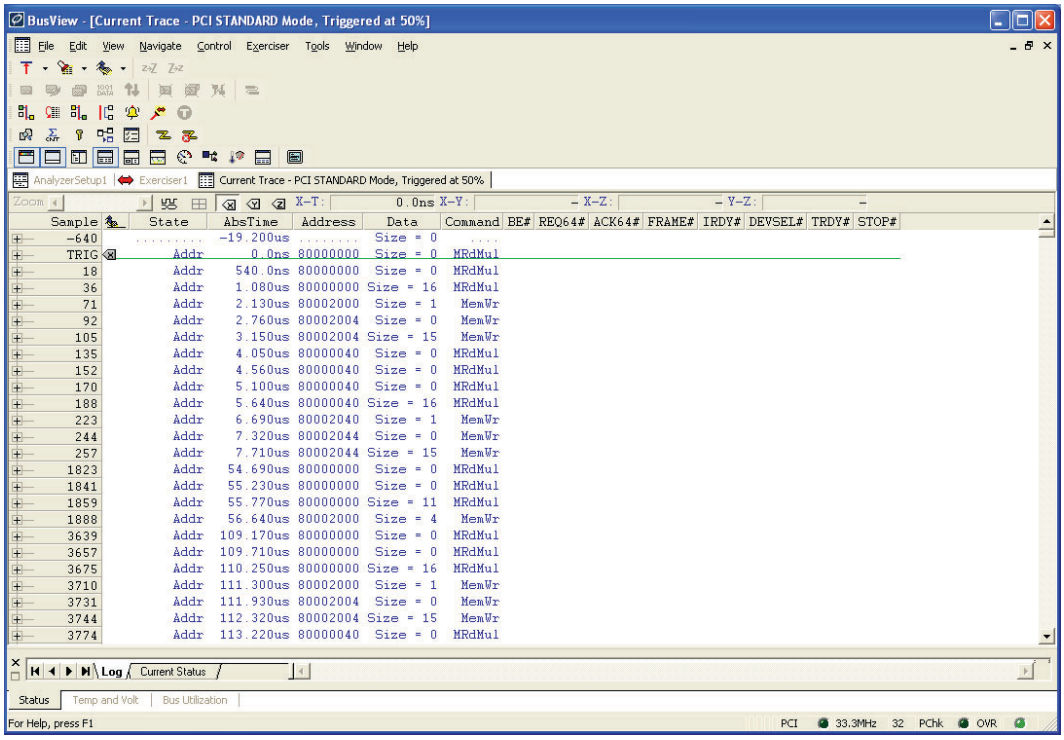


Figure 15: **Running BusView Analyzer**

After starting Analyzer, prior to stimuli generating a trigger, BusView displays **Analyzer sampling** in the status window.

The next step is to generate stimuli to cause Analyzer to trigger. Methods to generate stimuli are discussed in the "Generating Stimuli" section.

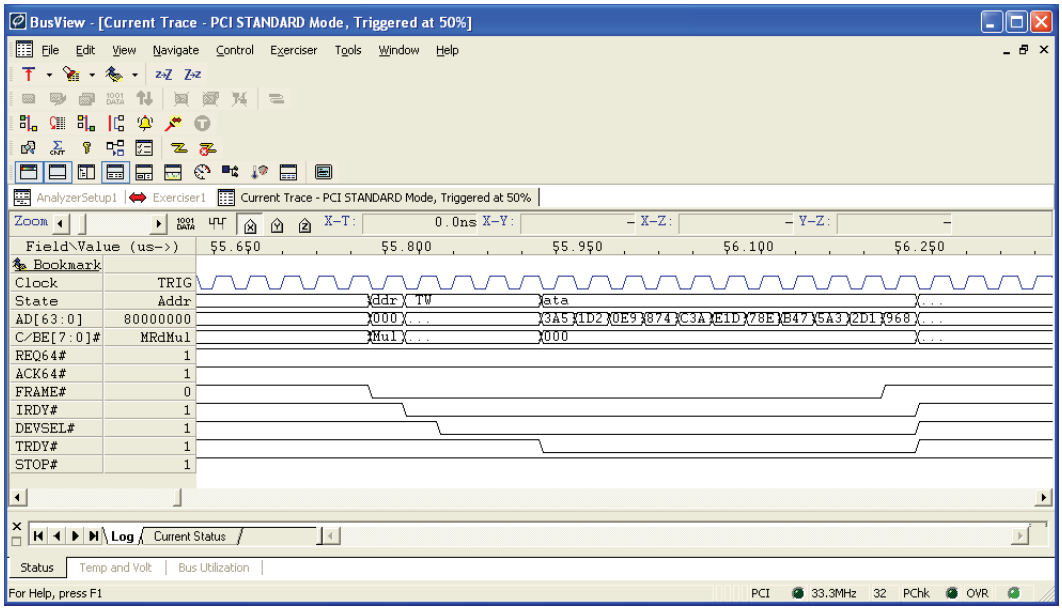
Figure 16 shows BusView Analyzer text output of the trace. The trace shows time, PCI address, data, and PCI command in a tabular format. This log shows three consecutive MemReadMultiple operations retried (size = 0) in several cases.



X998_16_011008

Figure 16: **BusView Analyzer Trace Output**

To convert the Trace text output to a waveform, click on the BusView waveform icon above the **State** column heading. Figure 17 shows BusView Analyzer Trace waveform output.



X998_17_011008

Figure 17: **BusView Analyzer Trace Waveform Output**

Generating Stimuli

The stimuli for the PCI performance tests run on the initiator. If the ML410 or ML555 is the initiator, XMD commands or C code is used. If the VG-PCI is the initiator, the BusView Exerciser Master transactions are used.

XMD Commands

XMD commands generate stimuli for both latency tests and the DMA transfers used in performance tests.

DMA transactions are used as stimuli for measuring bus transfer rate. DMA transactions are initially done using XMD commands to verify that the tests are functionally correct. In the performance test, continuously looping C code is used.

For DMA transactions initiated on the PLBv46 side, the xps_central_dma is used. The location of the registers in XPS Central DMA for generating a DMA transaction are given in Table 3. This table provides a reference for tcl scripts such as `dma.tcl`. To execute a DMA operation with XMD commands, the XPS Central DMA control, source address, destination address, and length registers are written. Sample tcl commands are under the xmd_commands directory.

Table 3: **ML410 FPGA XPS Central DMA Registers**

Register	Address
Reset	0x80200000
DMA Control Register	0x80200004
DMA Source Address	0x80200008
DMA Destination Address	0x8020000C
DMA Length	0x80200010

The `dma.tcl` script in Figure 18 writes to the four xps_central_dma registers defined in Table 3 to create DMA transactions.

The `dma.tcl` script also includes commented commands which illustrate how to run the loop continuously using a `while` statement and how to run the loop a specified number of times. Running continuously or looping is needed when XMD commands are used to generate stimuli for running statistical functions.

```
set outfile [open "dma.txt" "w"]
connect ppc hw
rst
#while {1} {
#for {set x  {$x < 256} {incr x} }

puts $outfile [mwr 0x80200004 0xC0000004]
puts $outfile [mwr 0x80200008 0x20000000]
puts $outfile [mwr 0x8020000C 0x20002000]
puts $outfile [mwr 0x80200010 1024]
#}
```

X998_18_011008

Figure 18: **XMD Commands for Generating Stimuli**

C Code

The performance tests are run using the C code in `pci_dma.c`. A functional diagram of the `pci_dma.c` code is given in Figure 19. The Barperpole Region function provides a rotating data pattern on 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 performs a DMA transaction of data located at the source address to the memory at the destination address. The Verify function verifies that data at the source address and destination address are equal.

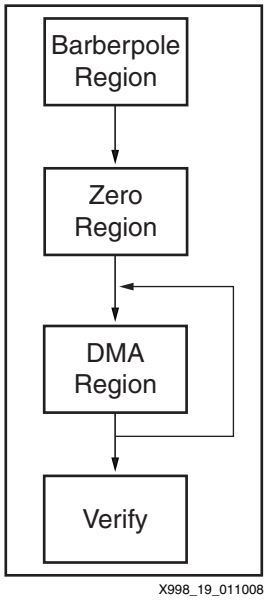


Figure 19: **Functional Diagram of pci_dma.c**

In the initial verification of the bus transactions, the pci_dma.c code is run with all four functions active as shown in Figure 19, without the loopback on DMA Region. When the performance test is run with the BusView statistics functions enabled, the code is edited to continuously loop on only the DMA Region. The Verify function is not included. Print statements are commented. Running DMA operations continuously is needed because the sampling interval for the BusView Statistical Function is set to 1 second (1 s), and running a single transaction doesn't last 1s. Running without verification or print statements is necessary to accurately measure performance.

The MEM0_StartAddress, MEM1_StartAddress, and DMA_Length parameters in pci_dma.c define the memory transfer and length for the DMA transaction. After the values are set as in Figure 20, EDK is used to build pci_dma/executable.elf. The values in the figure define a memory transfer from one location in ML555 BRAM to another location in the same BRAM.

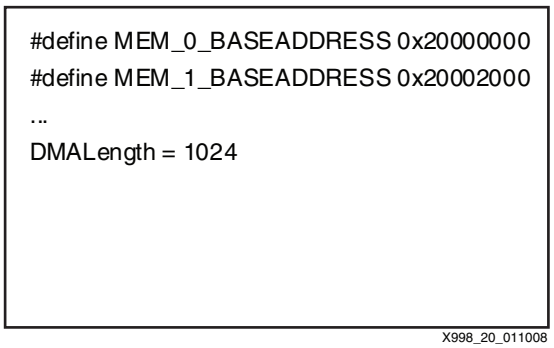


Figure 20: **Defining Source and Destination Addresses, Length in pci_dma.c**

Analogous to the xmd commands used to generate a DMA transaction discussed earlier, the C code writes the values to the xps_central_dma Source Address, Destination Address, and Length registers. To transfer data between different memory or to vary the DMA length, edit the MEM_*_BASEADDRESS and DMALength values, or both, referencing Table 1 and Table 2.

Figure 21 shows the Hyperterminal output when running `pci_dma/executable.elf`.

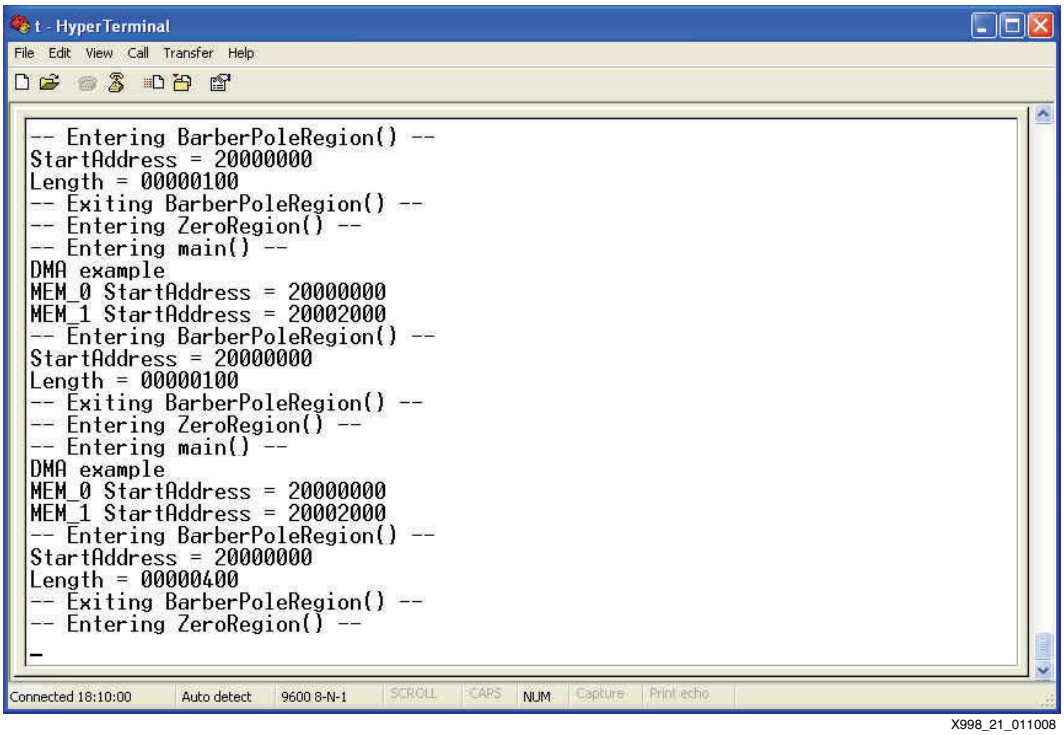


Figure 21: **pci_dma.c Output**

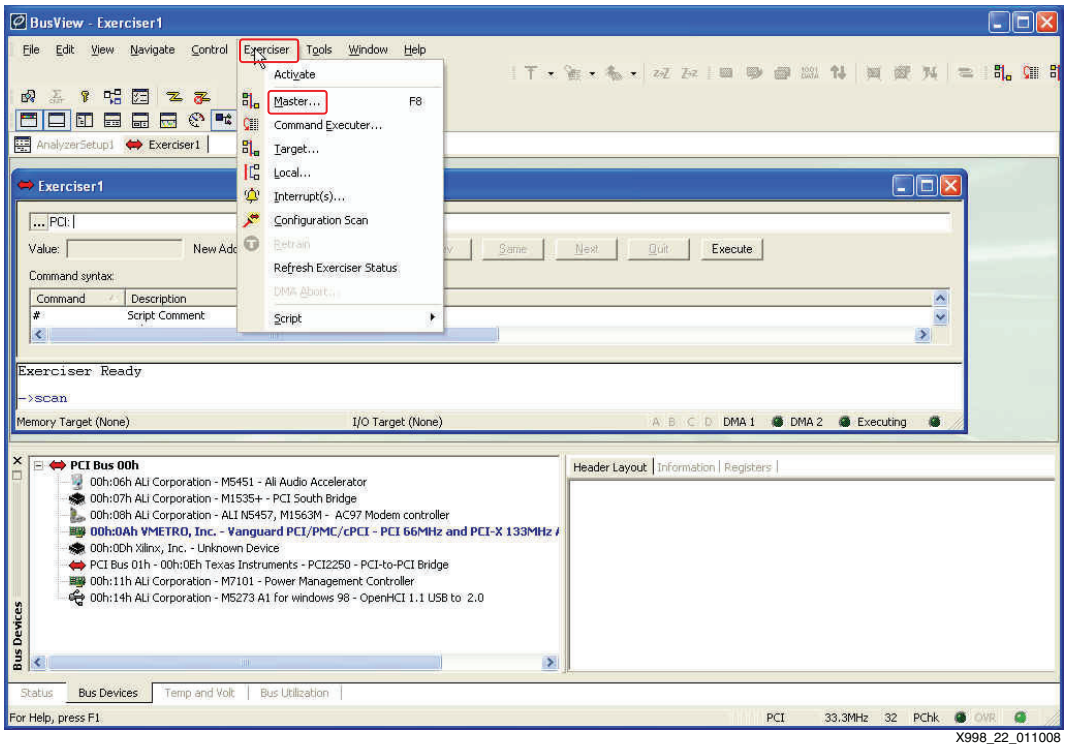
Using BusView Exerciser to Generate Stimuli

The BusView Exerciser generates three types of transactions.

- Local to PCI
- PCI to Local
- PCI to PCI

Figure 22 shows the BusView Exerciser menu items available after running the scan command.

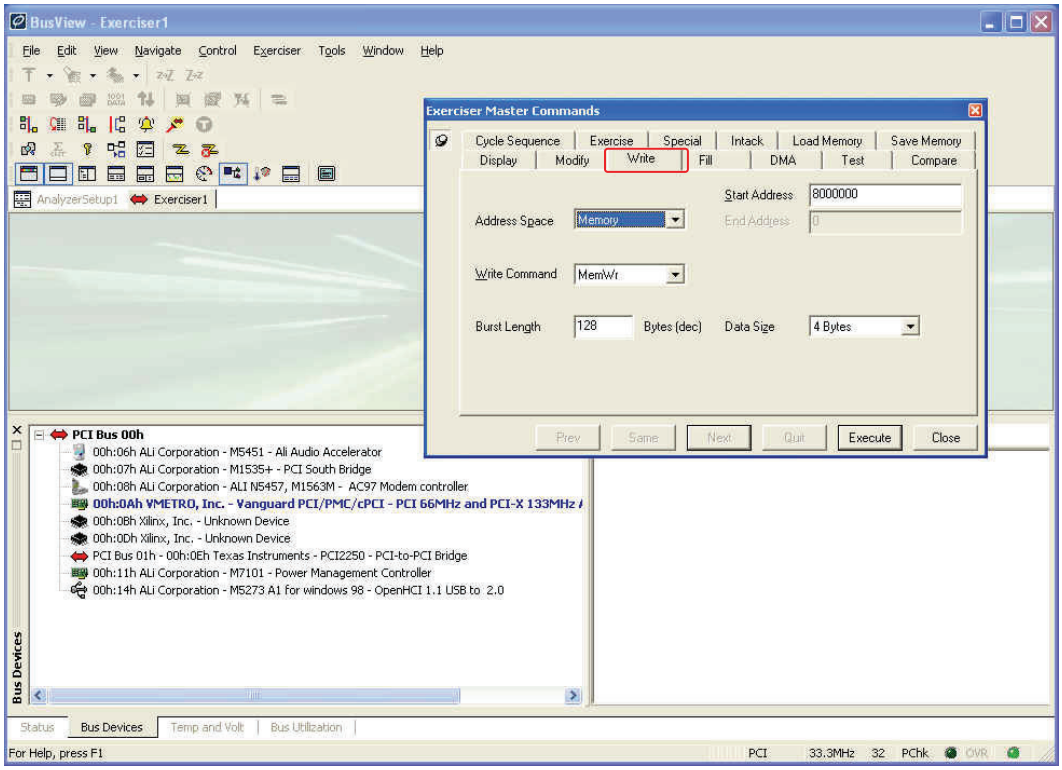
Select **Exerciser** → **Master**.



X998_22_011008

Figure 22: **BusView Exerciser Menu Items**

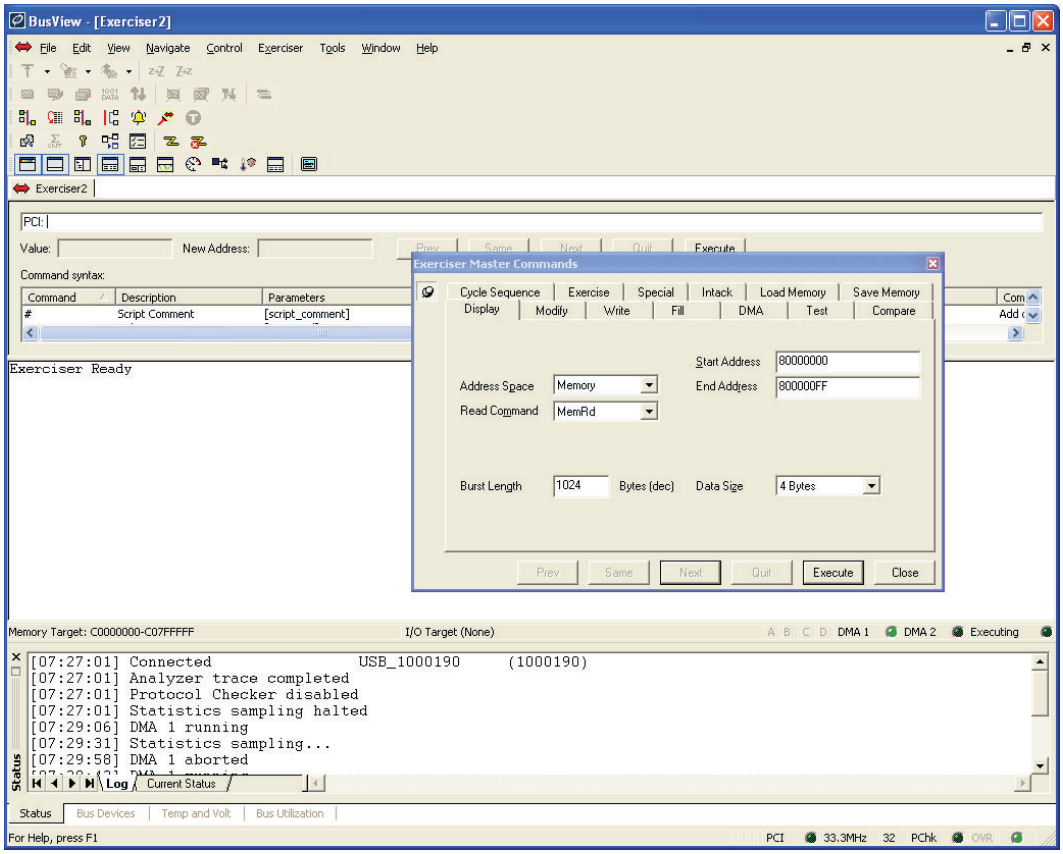
Figure 23 shows the Memory Write command from BusView Exerciser. The Write tab is enabled in the Exerciser Master window. This writes the memory decoded at PCI address `0x80000000`, which in this system is defined by the ML555 PLBv46 CSH BAR0. This addresses the ML555 BRAM located at `0x8AE08000`. The data written is defined as walking ones. BusView Exerciser can write data as a fixed pattern, walking ones, walking zeroes, address as data, and random.



X998_23_011008

Figure 23: **BusView Exerciser Memory Write**

The BusView read operation is done using the BusView Exerciser Master Display command. Figure 24 shows BusView Exerciser Display command. After **Execute** is clicked, a window displays the contents of the specified memory region.



X998_24_011008

Figure 24: **BusView Exerciser Display**

Figure 25 shows the BusView Exerciser command with the DMA tab enabled. This defines a DMA transaction from ML555 BRAM to a different region in ML555 BRAM. The source address is **0x80000000**. The destination address is **0x80002000**. The transfer size is **2000**. The burst length is **128** bytes.

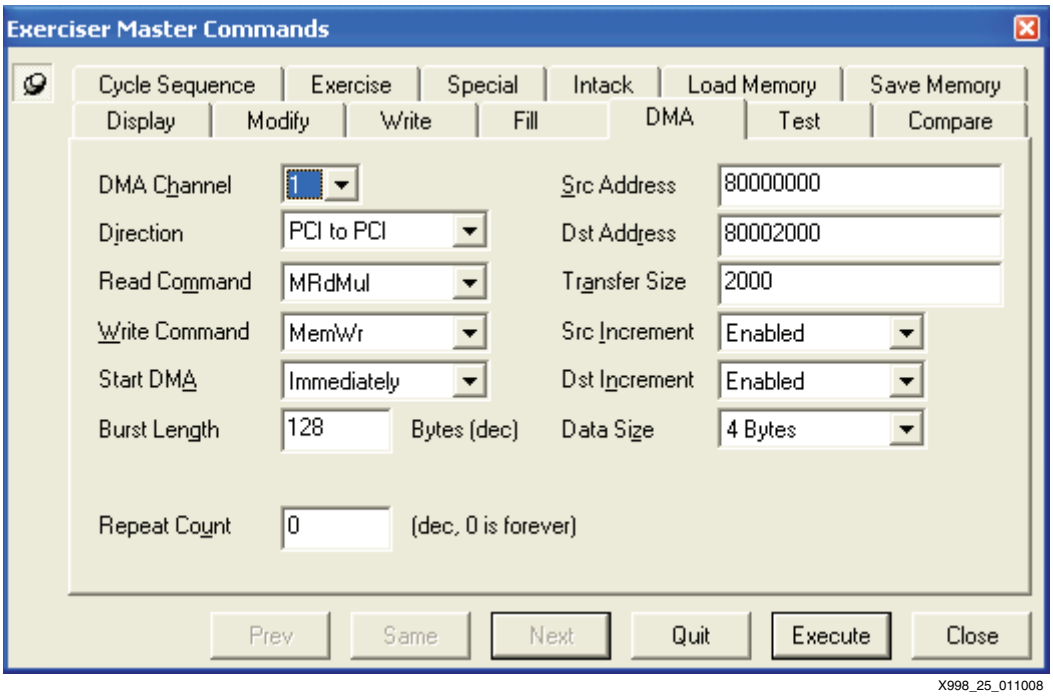


Figure 25: **BusView Exerciser DMA Command**

BusView Statistics Functions

The BusView Statistics Functions reports the seven predefined bus statistics shown in Figure 26 under Charts to View. BusView User defined statistics, supported with counters, are not discussed in this application note.

Figure 26 shows the setup of the BusView statistics functions. This statistics setup file enables the Transfer Rate, Command Distribution, and Burst Length Distribution functions.

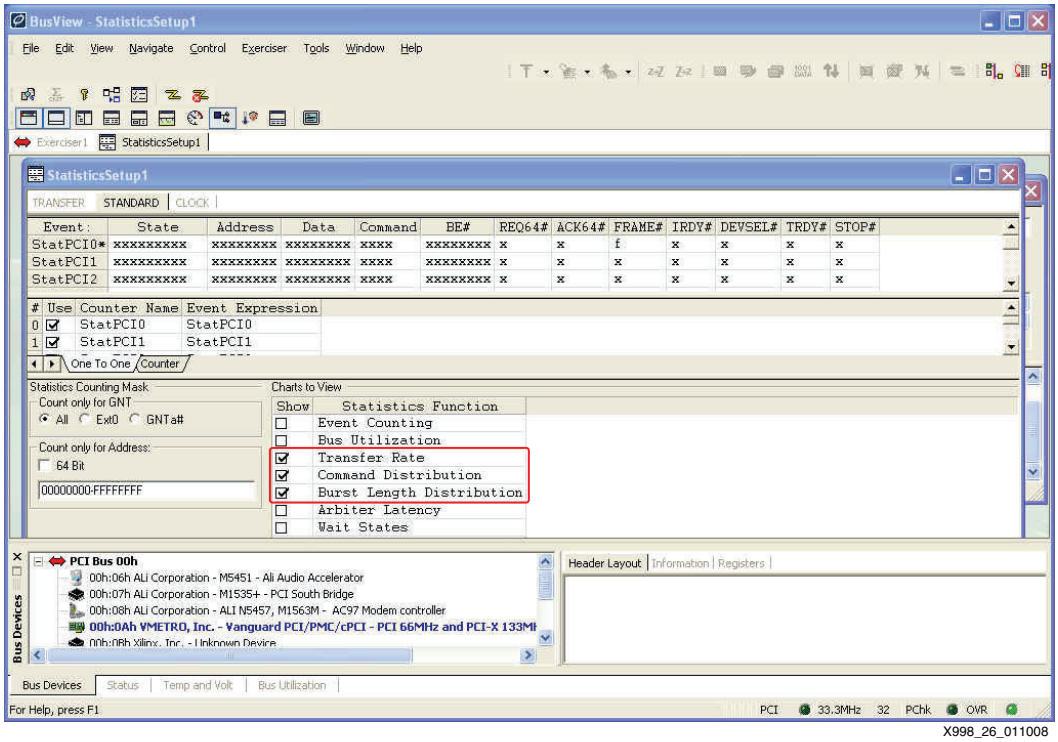
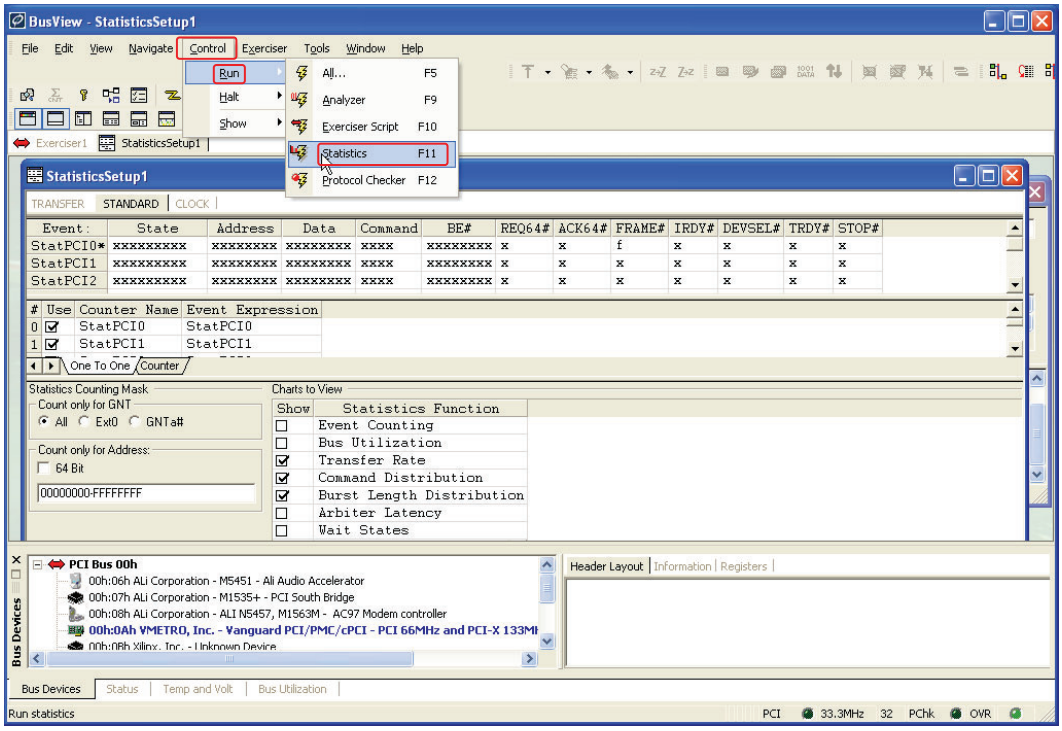


Figure 26: **BusView Statistics Setup**

Figure 27 shows how to run BusView statistics.

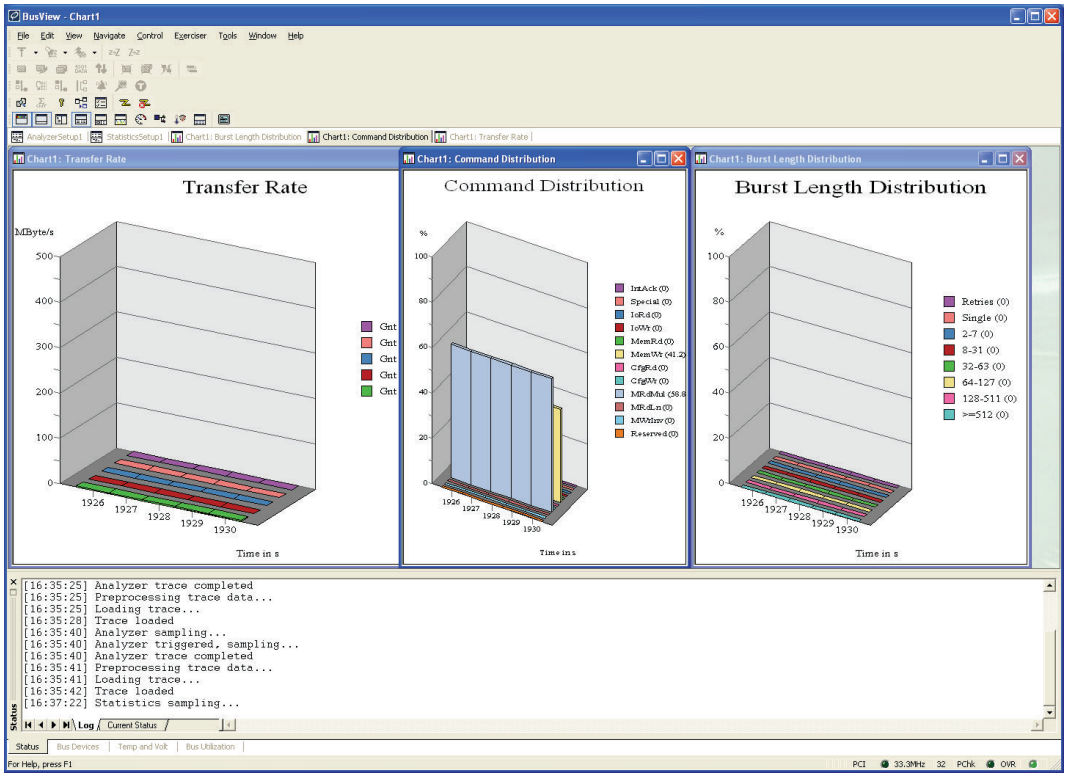
Control → **Run** → **Statistics** or the F11 shortcut.



X998_27_011008

Figure 27: **Starting BusView Statistics Sampling**

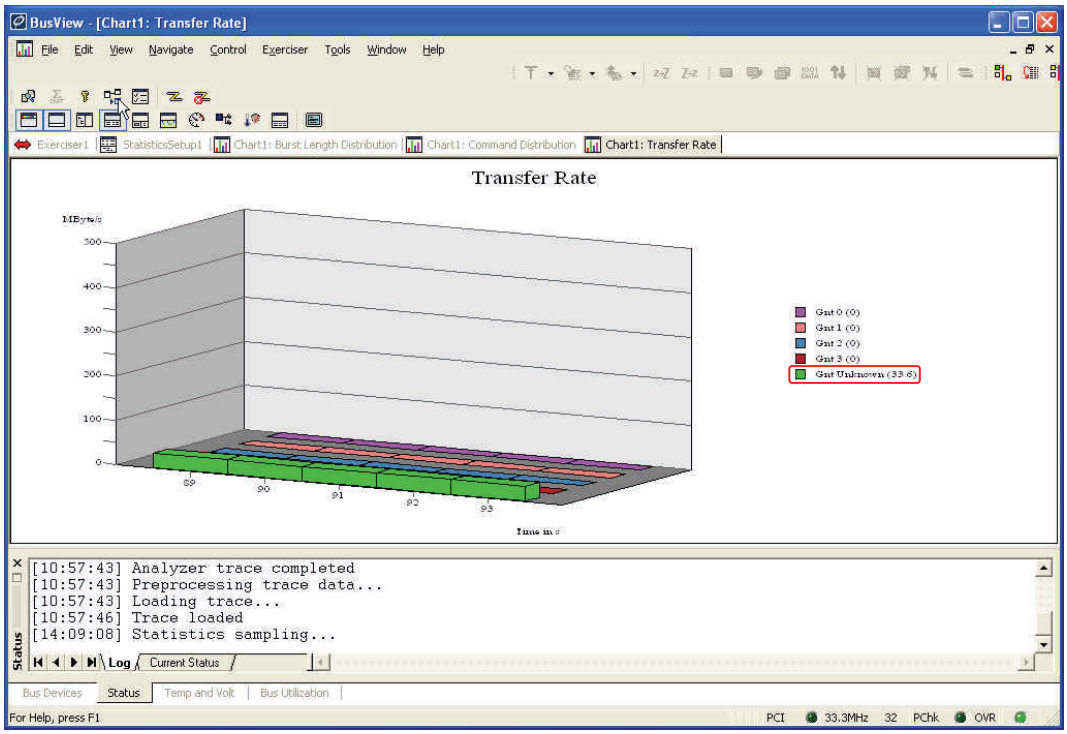
Figure 28 shows the statistical charts in the initial sampling mode. After traffic is generated by downloading the `pci_dma/executable.elf` file to the ML410 FPGA, a non-zero value is reported for the transfer rate.



X998_28_011008

Figure 28: **BusView Statistics Charts**

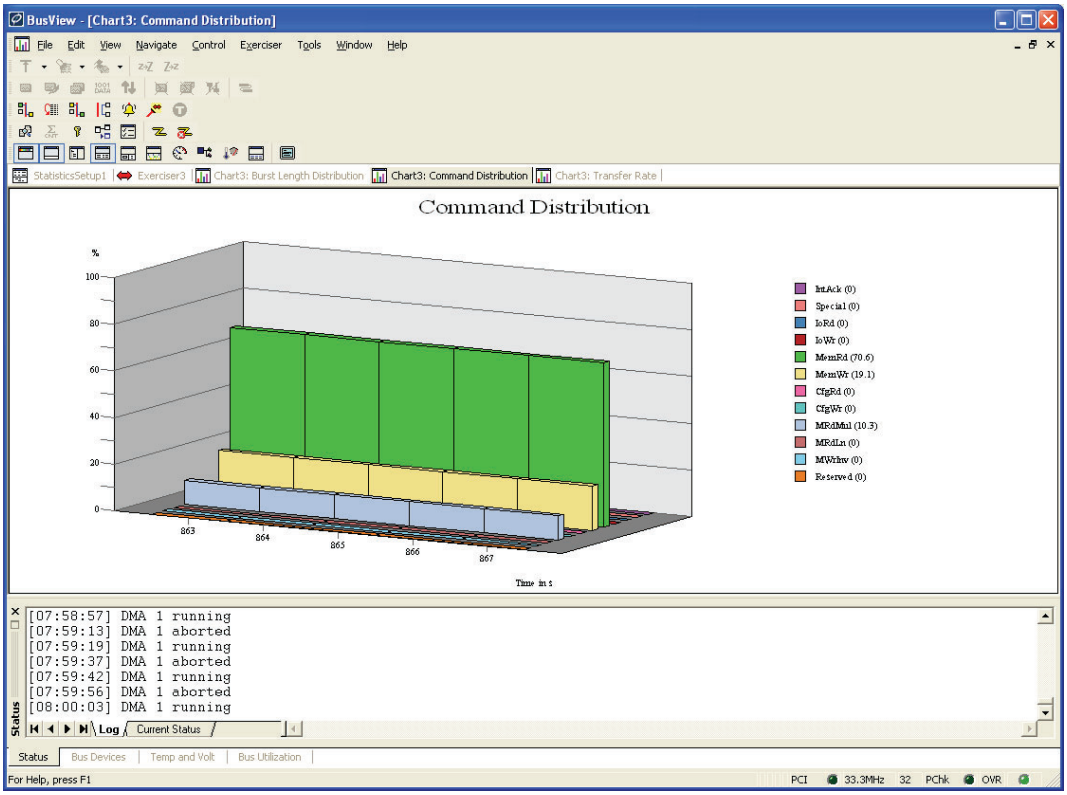
Figure 29 shows the BusView statistics for the transfer rate as 33.6 MB/s.



X998_29_011008

Figure 29: **Bus Transfer Rate Statistical Charts**

Figure 30 shows the command distribution as 70.6% MemRd, 10.3% MRdMul, and 19.1% MemWr/.



X998_30_011008

Figure 30: **Command Distribution**

Test Flow

The following steps outline the procedure for running performance tests.

1. Insert the ML555 in the ML410 slot P3 and the VG-PCI in slot P5.
2. Connect Platform USB from a PC to the ML410 and run **`impact -batch download.cmd`** from the ml410_ppc_plbv46_pci project.
3. Connect Platform USB from a 2nd PC to the ML555 and run **`impact -batch download.cmd`** from the ml555_mb_plbv46_pci project.
4. From the ml410_ppc_plbv46_pci project, run **`xmd -tcl wr_csh_410_555_vm.tcl`**
5. Invoke ChipScope Analyzer connected to the ML410 and import `ml410_plbv46_pci.cdc`.
6. In ChipScope Analyzer, run Setup Trigger to trigger on an active FRAME_N.
7. Invoke BusView. Run scan and verify the CSHs of VG-PCI, ML410 PCI, and ML555 PCI.
8. Optionally, invoke ChipScope Analyzer on the second PC and import `ml555_plbv46_pci.cdc`.
9. Edit the MEM_*_BASEADDRESS and DMALength parameters in `pci_dma.c` to specify the memory regions for data transfer. Use EDK to build the project and generate the elf file.
10. Set up the BusView statistics by running **File Open** → **statistics_setup**.
11. Using ML410 XMD, run **`dow pci_dma/executable.elf`**.
12. View the statistics reported by BusView.

Results

This section provides results for latency tests and PCI transfer rate for bus transactions between memory on the ML410 and ML555 Evaluation boards.

Latency Results

The stimuli for the latency tests is provided using XMD commands. ChipScope is used to determine the number of PCI clocks required for a transaction to complete.

Table 4 provides latency results in terms of the number of PCI clock cycles. The transactions are single reads and single writes from the ML410 to the ML555 BRAM (`0x20000000`) and DDR (`0xE8000000`).

Table 4: **PCI Latency tests**

xmd command	Latency (PCI Clocks)	Operation
`mwr 0x20000000 0x12345678`	16	Write to ML555 BRAM
`mrd 0x20000000 1`	24	Read of ML555 BRAM
`mwr 0xE8000000 0x12345678`	16	Write to ML555 DDR2
`mrd 0xE8000000 1`	35	Read of ML555 DDR2

Table 5 shows the latency of DMA operations. The 62PCI clock cycles is the latency from PA_Valid to the PCI transaction to complete.

Table 5: **Latency of DMA Operations**

Source Address	Destination Address	Latency (PCI Clocks)
`0x00000000`	`0x20000000 (ML555 BRAM)`	62
`0x20000000`	`0xE8000000 (ML555 DDR2)`	62

BusView Statistics Results

A comparison of the bus transfer rates of the PLBv46 PCI, the OPB PCI, and the PLB PCI cores is made. The test setup for Table 6 is a ML410 PLBv46 PCI initiator driving a ML555 PLBv46 PCI core inserted into the ML410 PCI slot P3. The stimuli is generated using `pci_dma.c`.

Table 6 shows the bus transfer rate for the OPB PCI, the PLB PCI, and the PLBv46 PCI cores from one region of ML555 BRAM to another region of ML555 BRAM for DMA lengths ranging from 32 to 1024.

Table 6: **PCI Transfer Rate - ML555 BRAM - MB/s**

DMA Length	32	64	128	256	512	1024
OPB PCI	2.67	4.87	9.27	16	22.7	28.7
PLB PCI	2.57	4.52	7.83	12.9	18.2	20.5
PLBv46 PCI	3.13	5.82	10.8	19.4	24.8	28.7

The test setup for Table 7, Table 8, and Table 9 is a ML410 PLBv46 PCI with ML555 and VG-PCI boards inserted into the ML410 PCI slots P3 and P5, respectively, and the VG-PCI acting as initiator.

Table 7 shows the bus transfer rate for the OPB PCI and the PLBv46 PCI cores from one region of the ML555 BRAM to another region of the ML555 BRAM for burst lengths ranging from 16 to 1024.

Table 7: **PCI Transfer Rate - Vmetro Initiator - ML555 BRAM to ML555 BRAM - MB/s**

Burst Size	16	32	64	128	256	512	1024
OPB PCI	6.21	16.1	27.4	34.4	38.3	40.7	42.6
PLBv46 PCI	3.84	14.1	26.7	33.5	37.5	40.0	41.4

Table 8 shows the bus transfer rate for the OPB PCI and PLBv46 PCI cores from ML555 BRAM to ML410 BRAM for burst lengths ranging from 16 to 1024.

Table 8: **PCI Transfer Rate - Vmetro Initiator - ML555 BRAM to ML410 BRAM - MB/s**

Burst Size	16	32	64	128	256	512	1024
OPB PCI	5.92	9.85	12.8	14.4	15.3	16.3	17.9
PLBv46 PCI	3.83	14.2	26.7	33.5	37.5	38.0	40.5

Table 9 shows the bus transfer rate for the OPB PCI and PLBv46 PCI cores from ML555 BRAM to ML555 DDR2 for burst lengths ranging from 16 to 1024.

Table 9: **PCI Transfer Rate - Vmetro Initiator - ML555 BRAM to ML555 DDR2 - MB/s**

Burst Size	16	32	64	128	256	512	1024
OPB PCI	6.21	16.1	27.4	34.4	38.3	40.0	40.0
PLBv46 PCI	3.83	14.2	26.7	33.5	37.5	38.0	40.5

Conclusion

Using the methodology provided, performance measurements are run for different PCI board configurations and transactions. The latency and transfer rate for transactions can be measured for different board configurations, DMA transactions, and burst lengths.

Using this approach, performance can be measured for different board configurations, such as the Enterpoint Raggestone 1 PCI board which uses the Spartan™-3 FPGA. Because the BusView Exerciser can generate multiple, simultaneous DMA transactions, bus performance can be measured involving different types of PCI traffic.

References

1. XAPP964 *Reference System: OPB PCI Using the ML410 Embedded Development Platform*
2. XAPP945 *Reference System: PLB PCI Using the ML410 Embedded Development Platform*
3. UG241 *OPB PCI v1.02.a User Guide*
4. *Vanguard PCI Bus Analyzer and Exerciser for PCI/PCI-X*

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
2/7/08	1.0	Initial Xilinx release.

Notice of Disclaimer