XAPP964 (v1.1) January 9, 2007

Reference System: OPB PCI Using the ML410 Embedded Development Platform

Author: John Ayer, Jr., Kris Chaplin, Beth Farwell, Ed Meinelt, Matt Nielson, Lester Sanders

Summary

This application note describes how to build a reference system for the On Chip Peripheral Bus Peripheral Component Interconnect (OPB PCI) core using the IBM PowerPC™ 405 (PPC405) Processor-based embedded system in the ML410 Embedded Development Platform. The reference system is Base System Builder (BSB) based and uses eight pcores. This reference system is similar to the reference system in XAPP911, except with the principal exception that the ML410 and ML455 boards are used.

A set of files containing Xilinx Microprocessor Debugger (XMD) commands is provided for writing to the Configuration Space Header and for verifying that the OPB PCI core is operating correctly. Several software projects illustrate how to configure the OPB PCI core, set up interrupts, scan the configuration registers, and set up and use DMA operations. The procedure for using ChipScope™ to analyze OPB PCI functionality is provided. The steps used to build a Linux kernel using MontaVista are listed. Simulation output files for analyzing basic PCI transactions are provided.

Included Systems

This application note includes one reference system:

- www.xilinx.com/bvdocs/appnotes/xapp964.zip

ml410_ppc_opb_pci is the project name used in xapp964.zip.

Required Hardware and Tools

Users must have the following tools, cables, peripherals, and licenses available and installed:

- Xilinx EDK 8.2.02
- Xilinx ISE 8.2.02
- Xilinx Download Cable (Platform Cable USB or Parallel Cable IV)
- Monta Vista Linux v2.4 Development Kit
- Modeltech ModelSim v6.1d
- ChipScope v8.2

Introduction

This application note accompanies a reference system built on the ML410 development board. Figure 1 is a block diagram of the reference system.

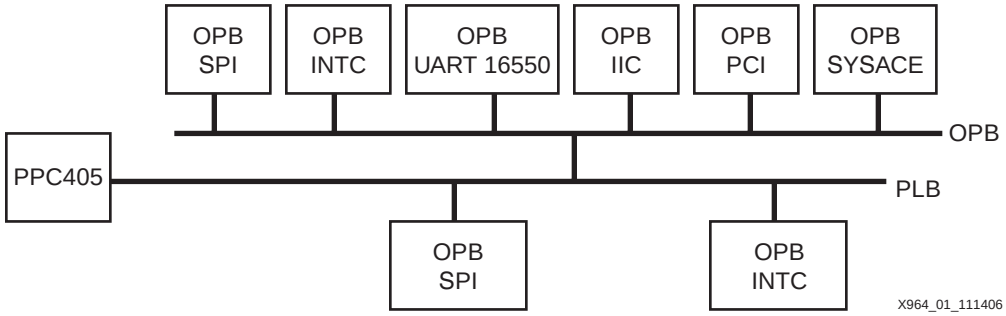


Figure 1: **OPB PCI Reference System Block Diagram**

The system uses the embedded PowerPC as the microprocessor and the OPB PCI core. On the ML410 board, the Virtex-4 XC4VFX60 accesses two 33 MHz 32-bit PCI buses: a primary 3.3V PCI bus and a secondary 5.0V PCI bus. The FPGA is directly connected to the primary 3.3V bus. The 5.0V PCI bus is connected to the Primary PCI bus with a PCI-to-PCI bridge, the TI2250. The PCI devices and four PCI add-in card slots on the ML410 are listed in Table 2. All PCI bus signals driven by the XC4VFX60 comply with the I/O requirements in the *PCI Local Bus Specification*, Revision 2.2.

Many of the ML410 functions are accessed over the 33 MHz 32-bit PCI bus. The Virtex-4 Platform FPGA contains PPC405 processors which access the primary PCI bus through the OPB PCI Bridge. PCI configuration in this reference design uses the OPB PCI Bridge as a host bridge.

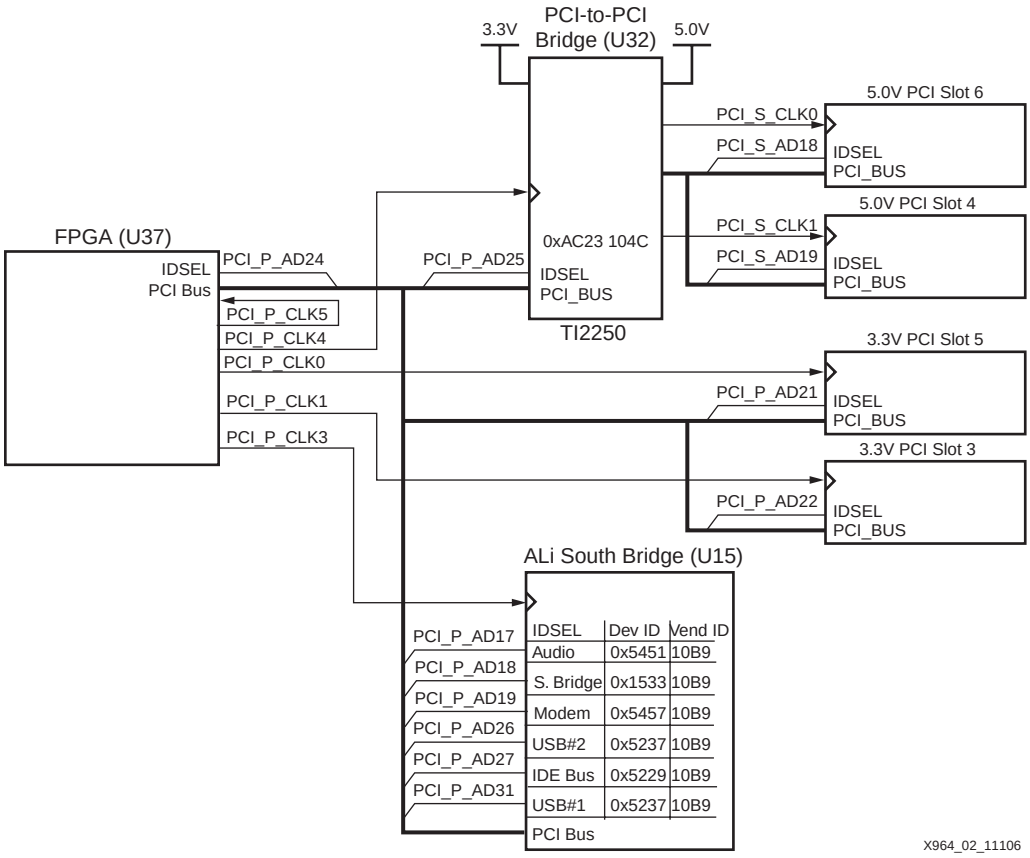


Figure 2: **PCI Bus Devices on the ML410**

Figure 2 shows PCI Bus Devices on the ML410. The TI2250 device is a PCI-to-PCI bridge to the two 5V PCI slots. The ALi M1535D+ South Bridge interfaces to the legacy devices, including the audio, modem, USB, and IDE ports. The Xilinx Virtex-4 ML455 PCI/PCI-X Board is inserted into slot 3.

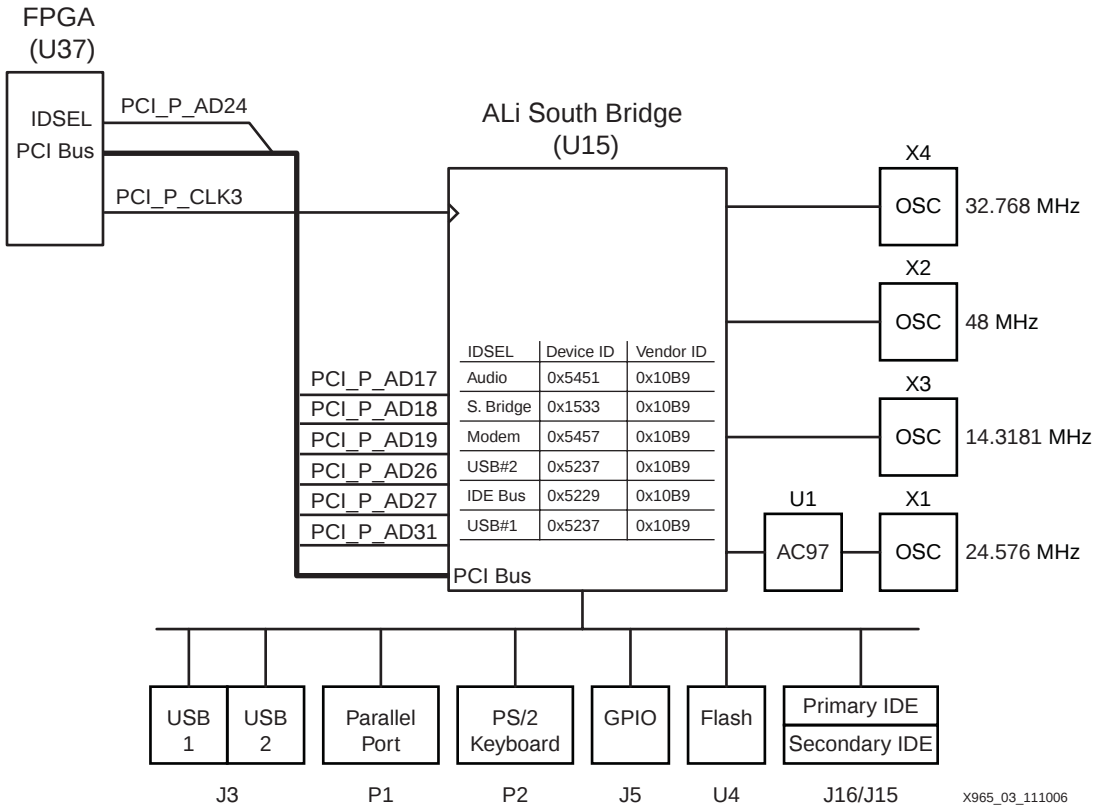


Figure 3: **ALI Bus - PCI to Legacy Devices**

Figure 3 shows the connections of the South Bridge to the legacy devices.

The functions, devices, and buses in the OPB PCI reference design defined in Figures 2 and 3 are addressed using the Configuration Address Port format shown in Figure 4.

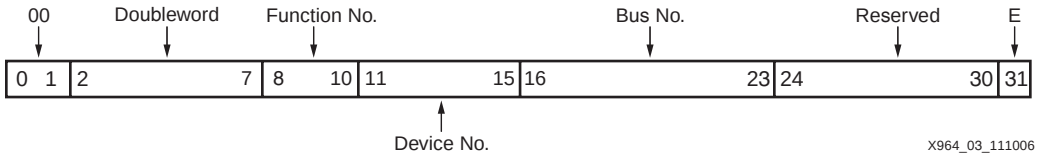


Figure 4: **Configuration Address Port Format**

The Configuration Address Port and Configuration Data Port registers in the Virtex-4 OPB PCI Bridge are used to configure multiple PCI bridges when host bridge configuration is enabled.The bit definitions of the Configuration Address Port in the big endian format used by the PLB is given in Table 1. The formats are different.

Table 1: **Configuration Address Port Register Definitions**

Bit	Definition
0-5	Target word address in configuration space
6-7	Hardwired to 0
8-12	Device
13-15	Function
16-23	Bus Number
24	Enable
25-31	Hardwired to 0

Reference System Specifics

In addition to the PowerPC405 processor and OPB_PCI, this system includes DDR and BRAM memory on the PLB, while on the OPB, it includes a UART, interrupt controller, SYSACE, IIC, and SPI. The relationship of the modules is shown in Figure 1. The PCI Arbiter core is included in the FPGA.

Table 2 provides the addresses of the IDSEL lines on the ML410 Board.

Table 2: **ML410 PCI Devices – IDSEL Lines**

Device	Dev ID	Vend ID	Bus	Dev	IDSEL Address
FPGA	0x0410	0x10EE	0	8	A
Ali M1535D+ South Bridge	0x1533	0x10B9	0	2	AD18
ALi Pwr Mgt	0x7101	0x10B9	0	12	AD28
ALi IDE	0x5529	0x10B9	0	11	AD27
ALi Audio	0x5451	0x10B9	0	11	AD17
ALi Modem	0x5457	0x10B9	0	3	AD19
ALi USB#1	0x5237	0x10B9	0	15	AD31
ALi USB#2	0x5237	0x10B9	0	10	AD26
TI Bridge (TI2250)	0AC23	0x104C	0	9	AD25
3.3V PCI Slot 3					AD22
3.3V PCI Slot 5	N/A	N/A	0	5	AD21
5.0V PCI Slot 4	N/A	N/A	1	3	AD19
5.0V PCI Slot 6	N/A	N/A	1	2	AD18

ML410 XC4VFX60 Address Map

Table 3: **ML410 XC4VFX60 System Address Map**

Peripheral	Instance	Base Address	High Address
PLB_DDR	DDR_SDRAM_32Mx64	0x00000000	0x03FFFFFF
OPB SPI	SPI_EEPROM	0x40A00000	0x40A0FFFF
OPB UART16550	RS232_Uart_1	0x40400000	0x4040FFFF
OPB INTC	opb_intc_0	0x41200000	0x4120FFFF
OPB_PCI	PCI32_Bridge	0x42600000	0x4260FFFF
OPB PCI DMA	PCI32_Bridge	0x42800000	0x4280FFFF
PLB BRAM	plb_bram_if_cntlr_0	0xFFFFC000	0xFFFFFFFF
OPB SYSACE	SysACE_CompactFlash	0x41800000	0x4180FFFF
OPB IIC	IIC_Bus	0x40800000	0x4080FFFF

Table 3 provides the address map of the ML410 XC4VFX60. The reference design contains the following settings for OPB PCI generics:

C_INCLUDE_PCI_CONFIG = 1

C_INCLUDE_BAROFFSET = 0

C_DMA_CHAN_TYPE = 0

C_IPIFBAR_NUM = 2

C_PCIBAR_NUM = 1

Figure 5 shows how to specify the values of generics in EDK.

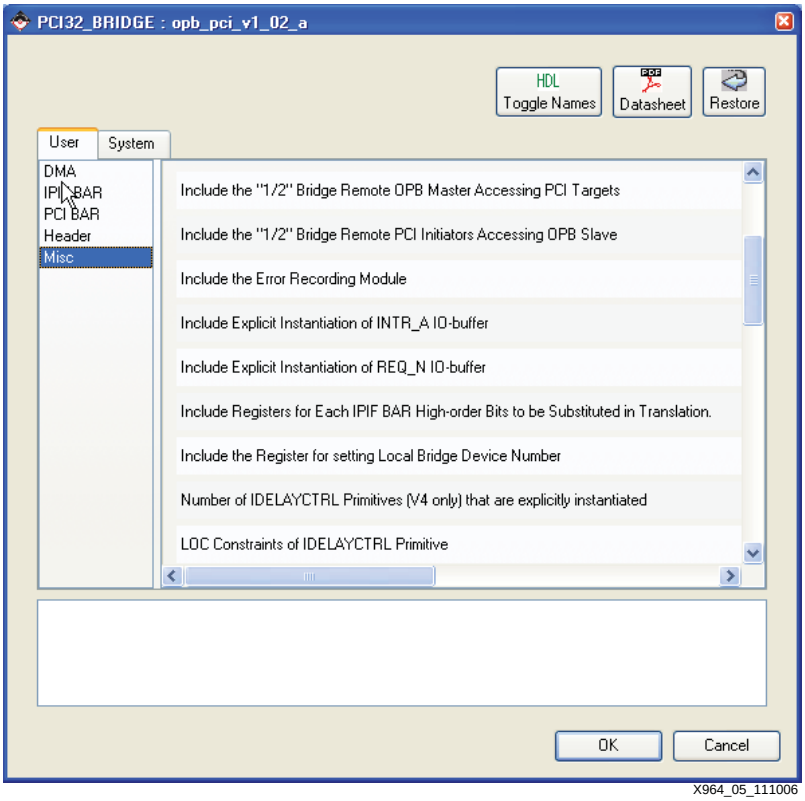


Figure 5: **Specifying the Values of Generics in EDK**

The C_INCLUDE_PCI_CONFIG generic configures the bridge as a host bridge. When C_INCLUDE_BAR_OFFSET = 0, the C_IPIFBAR2PCIBAR_* generic(s) are used in address translation instead of IPIFBAR2PCIBAR_* registers. Setting C_DMA_CHAN_TYPE = 0 specifies simple DMA. Setting C_IPIFBAR_NUM = 2 specifies that there are two address ranges for OPB to PCI transactions. Setting C_PCIBAR_NUM = 1 specifies that one address range is used for PCI to OPB transactions

Figure 6 provides a functional diagram of the OPB PCI Full Bridge core. The three functions of the core are the OPB IPIF, the v3.0 PCI Core, and the IPIF/v3.0 Bridge.

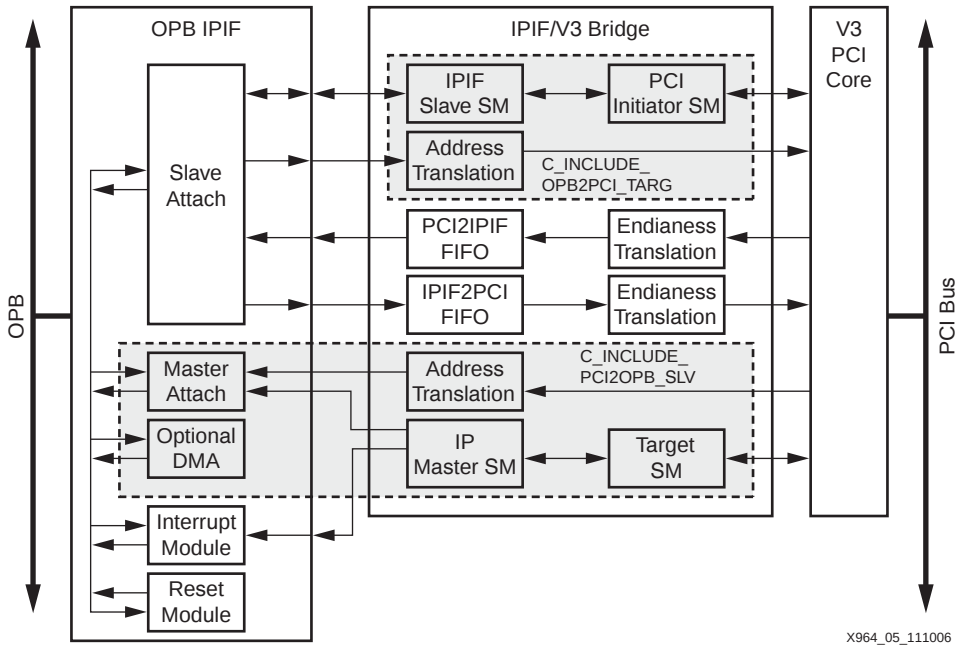


Figure 6: **Block Diagram of OPB PCI Bridge Core**

Virtex-4 ML455 PCI/PCI-X Development Board

In the reference design, the OPB PCI in the XC4VFX60 on the ML410 board interfaces to the OPB PCI in the Virtex-4 ML455 PCI/PCI-X Development board. The ML455 board uses the Xilinx XC4VLX25 device in the 668 pin package. The `ml410_ppc_opb_pci/455` directory contains the `system.mhs` and other project files for the ML455.

Table 4 provides the address map for the XC4VLX25.

Table 4: **XC4VLX25 Address Map**

Peripheral	Instance	Base Address	High Address
LMB_BRAM_IF_CNTLR	DLMB_CNTLR/ILMB_CNTLR	0x0000000	0x70003FFF
OPB_UART16550	RS232_Uart	0x40400000	0x4040FFFF
OPB PCI	PCI_Bridge	0x42600000	0x4260FFFF
OPB DDR	DDR_SDRAM_64Mx32	0x24000000	0x27FFFFFF
OPB GPIO	LEDs_4Bit	0x40000000	0x4000FFFF
OPB MDM	debug_module	0x41400000	0x4140FFFF
OPB INTC	opb_intc_0	0x41200000	0x4120FFFF

The ML455 includes three clock sources, a 64-bit PCI edge connector, 128 MB (16M x 64) DDR SDRAM memory, RS232C port, LED displays, XCF32P-FSG48C Platform Flash configuration PROM, and a JTAG port. The MicroBlaze microprocessor is used in this design.

Figure 7 shows the ML455 PCI/PCI-X Development board.

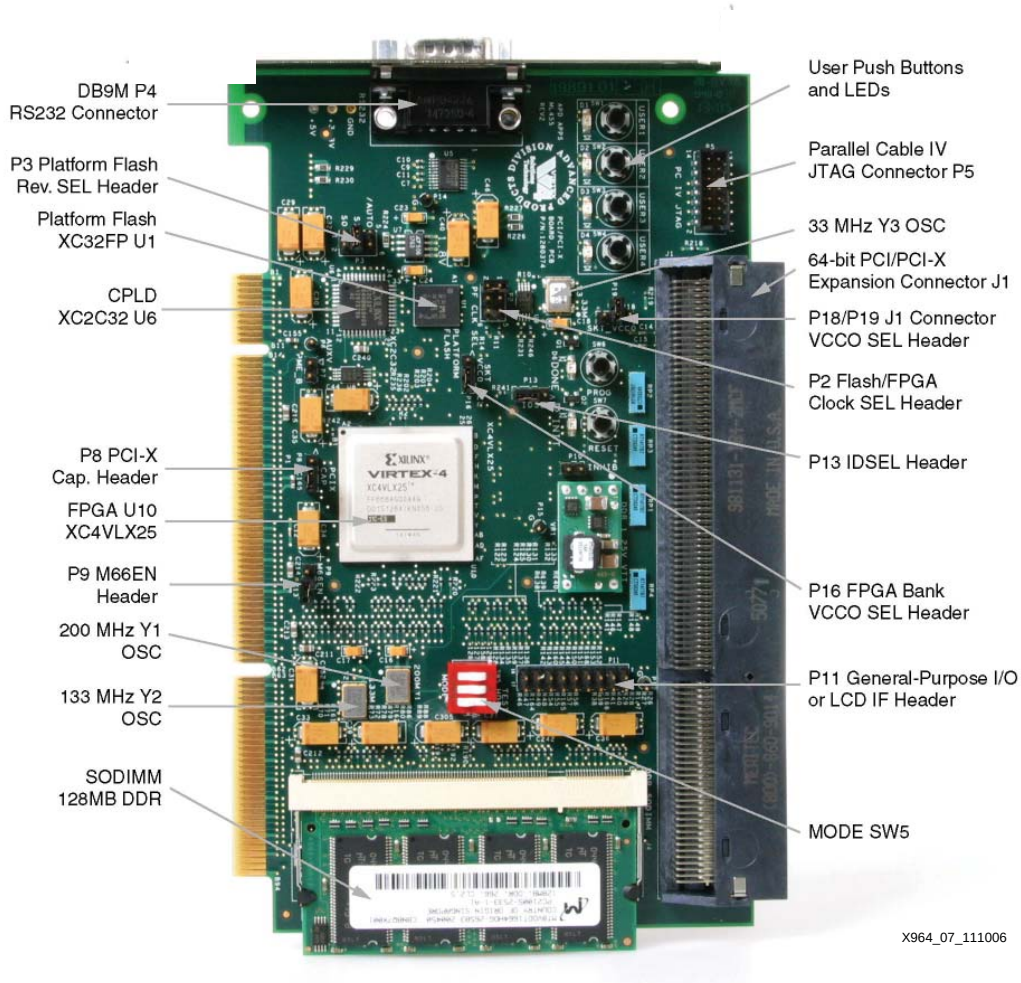


Figure 7: **ML455 PCI/PCI-X Development Board**

Interfacing to the OPB PCI on the ML455 PCI/PCI-X Board

Figure 8 shows the principle interface blocks when transferring data between the OPB PCI Bridge in the XC4VFX60 on the ML410 board and the OPB PCI Bridge in the XC4VLX25 on the ML455 board.

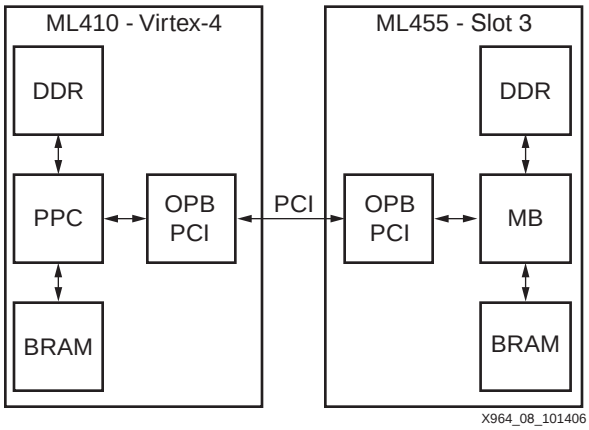


Figure 8: **Interfacing ML410 Board OPB PCI with the ML455 Board OPB PCI**

Configuration of OPB PCI on the ML410 Board

The OPB PCI bridge uses the 32-bit Xilinx LogiCore Version 3 IP (v3.0) core. For the OPB PCI bridge to perform transactions on the PCI bus, the v3.0 core must be configured using configuration transactions from either the PCI-side or from the OPB side. This reference design configures the bridge from the OPB side, therefore C_INCLUDE_PCI_CONFIG is set to 1. In this case, IDSEL input of the v3.0 is connected to the address ports specified in Table 2, while the IDSEL port of the bridge is unused.

To write to the configuration header, execute the following steps:

1. Configure the Command and Status Register. The minimum that must be set is the Bus Master Enable bit in the command register. For memory transactions, the memory space bit must be set. For I/O transactions, the I/O space bit must be set.

2. Configure the Latency Timer to a non-zero value.

3. Configure at least one BAR. Configure subsequent BARs as needed for other memory/IO address ranges.

The v3.0 core configures itself only after the Bus Master Enable bit is set and the latency timer is set to avoid time-outs. If the v3.0 core latency timer remains at the default 0 value, configuration writes to remote PCI devices do not complete, and configuration reads of remote PCI devices terminate due to the latency timer expiration. Configuration reads of remote PCI devices with the latency timer set to 0 return `0xFFFFFFFF`.

Configuration of OPB PCI on the ML455 PCI/PCI-X Board

When the ML455 is inserted into a ML410 PCI slot, the OPB PCI Bridge in the Xilinx XC4VFX60 FPGA interfaces to an OPB PCI Bridge in the XC4VLX25 FPGA on the ML455 PCI Board. To configure the XC4VLX25, connect the Xilinx Download (USB or Parallel IV) cable to the ML455 JTAG port, and use Impact to download the `ml410_pci/455/implementation/download.bit` file. After configuring the XC4VLX25 FPGA using the bit file, the PCI functionality OPB PCI in the XC4VLX25 is configured using Configuration write transactions from the OPB PCI in the XC4VFX60 of the ML410.

Executing the Reference System using the Pre-Built Bitstream and the Compiled Software Applications

To execute the system using files inside the ml410_ppc_opb_pci/ready_for_download directory, follow these steps:

1. Change to the `ml410_ppc_opb_pci/ready_for_download` directory.
2. Use iMPACT to download the bitstream by using the following:
   `impact -batch xapp964.cmd`
3. Invoke XMD and connect to the MicroBlaze processor by the following command:
   `xmd -opt xapp964.opt`
4. Download the executable by the following command
   `dow <path>/executable.elf`

Executing the Reference System from EDK

To execute the system using EDK, follow these steps:

1. Open system.xmp inside EDK.
2. **Use Hardware** → **Generate Bitstream** to generate a bitstream
3. Download the bitstream to the board using Device **Configuration** → **Download Bitstream**.
4. Invoke XMD with **Debug Launch XMD**.
5. Download the executable by the following command.
   `dow <path>/executable.elf`

Verifying the Reference Design with the Xilinx Microprocessor Debugger

After downloading the bitstream file and writing to the configuration header, execute the following steps to verify that the ML410 reference design is set up correctly.

1. Configure the v3.0 Command Register, Latency Timer, and BAR(s).
2. Read the configuration header.
3. Configure the Command Register, Latency Timer, and BAR(s) of the other devices in the system.
4. Read the configuration headers of the other devices in the system.
5. Perform a memory read of one of the IPIF BARs.
6. Perform a memory write of one of the IPIF BARs.

Verification is done using either Xilinx Microprocessor Debugger (XMD), the software projects, or both discussed later in this document. Text files of the XMD commands are provided in the xmd_command directory in the design files. The `configure*.xmd` files contain XMD commands which configure the bridge. The `test_interrupt_regs.xmd` file contains commands which set up and verify the interrupt registers. The `write_read_bram.xmd` and `write_read_ddr.xmd` files contain XMD commands which test PLB BRAM and PLB DDR respectively. Table 5 lists the files containing XMD commands which verify that the initial setup of the reference system is correct. In the nomenclature below, the board name (ML410, ML455) is used rather than the device name (4fx60, 4vlx25)

Table 5: **XMD Configuration Commands**

XMD command	Function
configure_ml410.xmd	Configures the ML410 CSR, Latency Timer, BARs.
configure_ml455.xmd	Configures the ML455 CSR, Latency Timer, BARs.
ml410_bram.xmd	Tests the PLB BRAM connected to the ML410.
ml410_plbddr.xmd	Tests PLB DDR connected to the ML410.
ml410_455bram.xmd	Tests ML410 writing to ML455 Board BRAM.
ml410_455ddr.xmd	Tests ML410 writing to ML455 DDR SDRAM.
ml455_bram.xmd	Tests the BRAM connected to the ML455.
ml455_410ddr.xmd	Tests the ML455 writing to ML410 PLB DDR.
dma_410_455ddr.xmd	Tests DMA from the ML410 to ML455 DDR
dma_410ddr_455ddr.xmd	DMA from ML410 PLB DDR to ML455DDR SDRAM.
dma_455ddr_410ddr.xmd	Tests DMA from M455 DDR to ML410 DDR.
dma_410bram_455bram.xmd	Tests DMA from PLB BRAM to PCI Bridge.

The following steps use the XMD commands in the files in the directory, `ml410_ppc_opb_pci/xmd_command`

1. Invoke **XMD**.
2. Open the XMD command file (e.g. `configure.xmd`) using a text editor such as WordPad.
3. Select and copy the XMD commands in the text editor.
4. Right click the mouse in the XMD window to paste the commands.

The XMD commands in the `configure_ml410.xmd` file, listed in Figure 9, write to the Configuration Address Port and to the Configuration Data Port to program the Configuration Space Header. The Command/Status Register, Latency Timer, and Base Address Registers are written and read.

```
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 8.2.02Build EDK_Im_Sp2.3
Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
XMD% mrd 0x4260010C 1
4260010C:   00000000
XMD%    mwr 0x4260010C 0x04400080
XMD%    mrd 0x4260010C 1
4260010C:   04400080
XMD%    mrd 0x42600110 1
42600110:   46050022
XMD%    mwr 0x42600110 0x86002002
XMD%    mrd 0x42600110 1
42600110:   46050002
XMD%    mwr 0x4260010C 0x08400080
XMD%    mrd 0x42600110 1
42600110:   1A452301
XMD%    mwr 0x4260010C 0x0C400080
XMD%    mwr 0x42600110 0x00FF0000
XMD%    mrd 0x42600110 1
42600110:   00FF0000
XMD%    mwr 0x4270010C 0x10400080
XMD%    mwr 0x42700110 0x00000000
XMD%    mrd 0x42700110 1
42600110:   08000000
XMD%    mwr 0x4260010C 0x14400080
```

X964_09_111406

Figure 9: **XMD Commands for Configuring OPB PCI**

Software Projects

The reference system contain the following software projects. In each software project directory, there are src sub-directories for the source code.

TestApp_Memory. This project tests the memory on the ML410 board.

TestApp_Peripheral. This project verifies that the OPB Sysace is used correctly.

hello_pci. This project enables master transactions, sets the latency timer, defines the bus number/subordinate bus number, and scans the ML410 configuration registers.

xpci_tapp_example. This project contains two major functions which use level 0 drivers: PciInitLevel_0() and ShowPCI(). PciInitLevel reads and writes the bus and subordinate bus number, initializes the bridge device number if present, and performs configuration writes to the PCI header to set up the bridge. The latency timer is set to the maximum. The BAR registers are set, and interrupts are enabled.

xpci_example_level_0. This project uses level 0 drivers to initialize the PCI bridge. It then initializes a remote device on the PCI bus, and prints messages describing the bridge and memory mappings.

xpci_example_level_1. This project initializes the OPB PCI bridge and a remote device on the PCI bus. It then initializes Direct Memory Access (DMA) and runs local to DMA and DMA to local transfers. The PciIsr() function handles interrupts. The OPB PCI Bridge supports simple but not scatter gather DMA. Simple DMA is enabled by setting C_DMA_CHAN_TYPE = 0. The DS416 Direct Memory Access and Scatter Gather product specification provides information on the use of the DMA function in the IPIF. The base address for DMA in the ML410 reference system is C_DMA_BASEADDR = `0x42800000`. The registers used in DMA setup are given below.

Table 6: **DMA Registers**

DMA Register	Address
Control Register	C_DMA_BASEADDR + 0x04
Source Address Register	C_DMA_BASEADDR + 0x08
Destination Address Register	C_DMA_BASEADDR + 0x0C
Length Address Register	C_DMA_BASEADDR + 0x10

The code which transfers DMA data is given in Figure 10.

```
/*****************************************************************************
* PciDmaTransfer - interrupt driven DMA to/from a device on the PCI bus
 *
 * Parameters:
 *   LocalAddress - address of a local memory device such as SRAM, DDR, etc.
 *   RemoteAddress - address local to the processor that maps to the PCI bus
 *   Bytes - bytes to transfer
 *   Direction - direction to transfer, 0 = local to remote, 1 = remote to local
 *****************************************************************************/
void PciDmaTransfer(Xuint32 LocalAddress, Xuint32 RemoteAddress, Xuint32 Bytes,
                    int Direction)
{
   Xuint32 DmaSrc;
   Xuint32 DmaDest;

   /* setup transfer direction */
   if (Direction == 0)
   {
      /* local to PCI */
      XDmaChannel_SetControl(&Bridge.Dma,
                        XDC_DMACR_SOURCE_INCR_MASK |
                        XDC_DMACR_DEST_INCR_MASK |
                        XDC_DMACR_DEST_LOCAL_MASK);
      DmaSrc = LocalAddress;
      DmaDest = RemoteAddress;
   }
   else if (Direction == 1)
   {
      /* PCI to local */
      XDmaChannel_SetControl(&Bridge.Dma,
                        XDC_DMACR_SOURCE_INCR_MASK |
                        XDC_DMACR_DEST_INCR_MASK |
                        XDC_DMACR_SOURCE_LOCAL_MASK);
      DmaSrc = RemoteAddress;
      DmaDest = LocalAddress;
   }
   else
   {
      printf("Invalid direction argument. Must be 0 or 1\n");
      return;
   }

   /* begin transfer */
   XDmaChannel_Transfer(&Bridge.Dma, (Xuint32*)DmaSrc,
                        (Xuint32*)DmaDest, Bytes);

   /* wait for completion */
   SemaphoreTake(&Bridge.DmaSemaphore);
   printf("Dma Transfer complete\n");
}
```

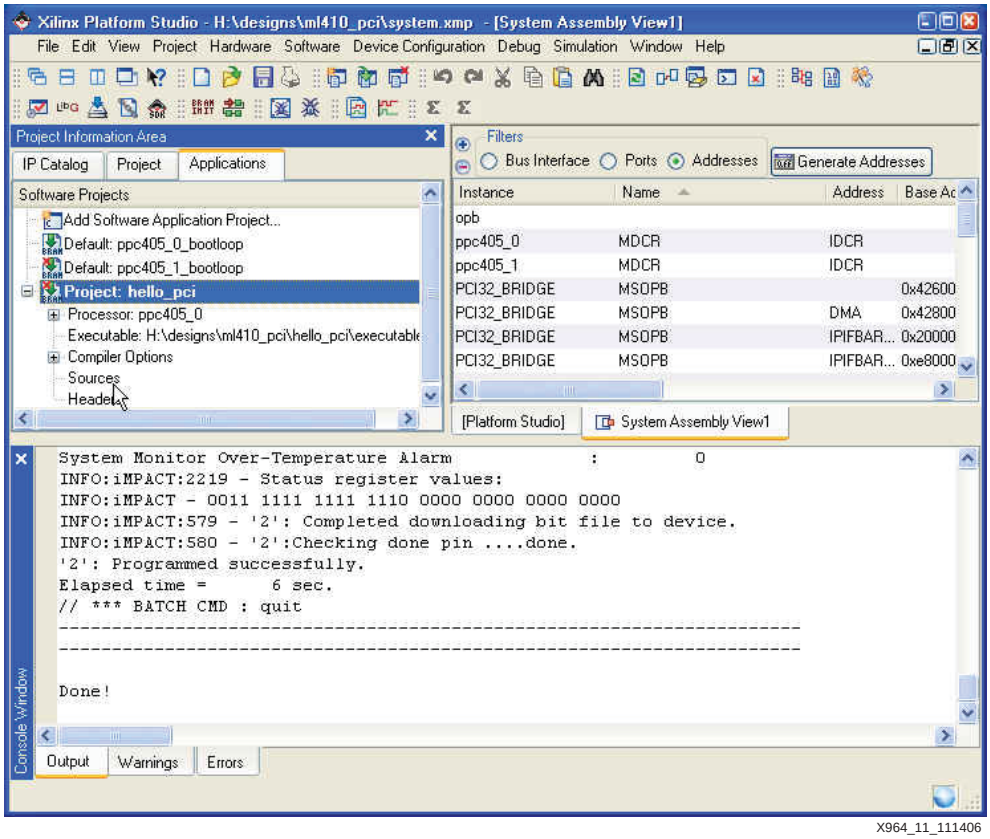
X964_10_111406

Figure 10: **C Code for DMA Operation**

Running the Applications

In XPS, select the **Applications** tab, **Software Projects**, and **Add SW Applications Projects**. Name the software project (e.g. hello_pci), and add the files from the software project src directory to **Sources** (see the cursor in the Project Information Area pane in Figure 11).

The structure of the hello_pci is shown in Figure 11. Make the hello_pci project active and the remaining software projects inactive.



X964_11_111406

Figure 11: **Selecting the hello_pci Software Project**

Select hello_pci and right click to build the project. If more than one software project is used, make the unused software projects inactive.

Connect a serial cable to the RS232C port on the ML410 board. Start up a HyperTerminal. Set the baud rate to **9600**, number of data bits to **8**, no parity, and no flow control, as shown in Figure 12.

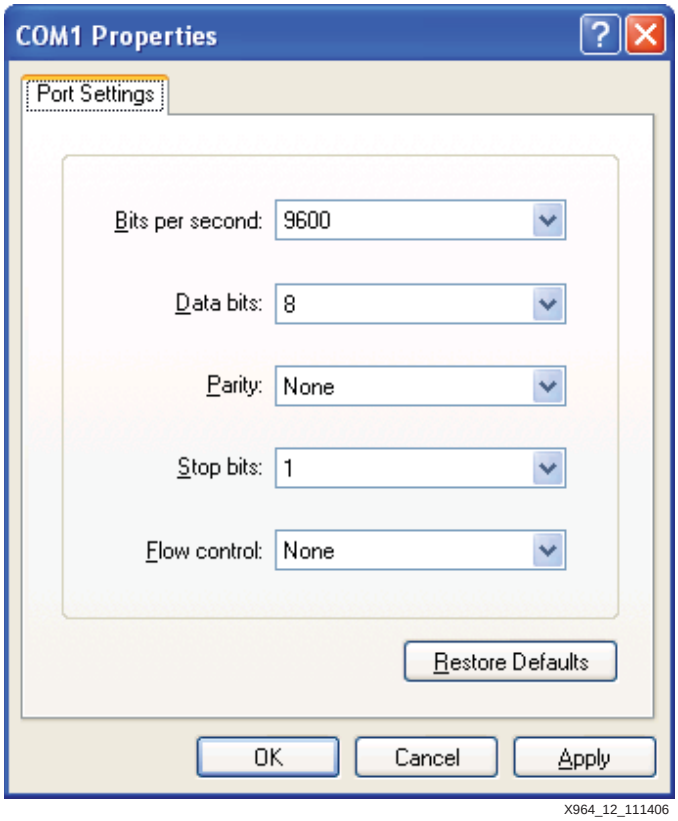


Figure 12: **HyperTerminal Parameters**

From XPS, start **XMD** and enter **rst**. Invoke GDB and select Run to start the application as shown in Figure 13. The hello_pci.c code written for the ML310 shown in the figure runs without any modifications on this reference system.

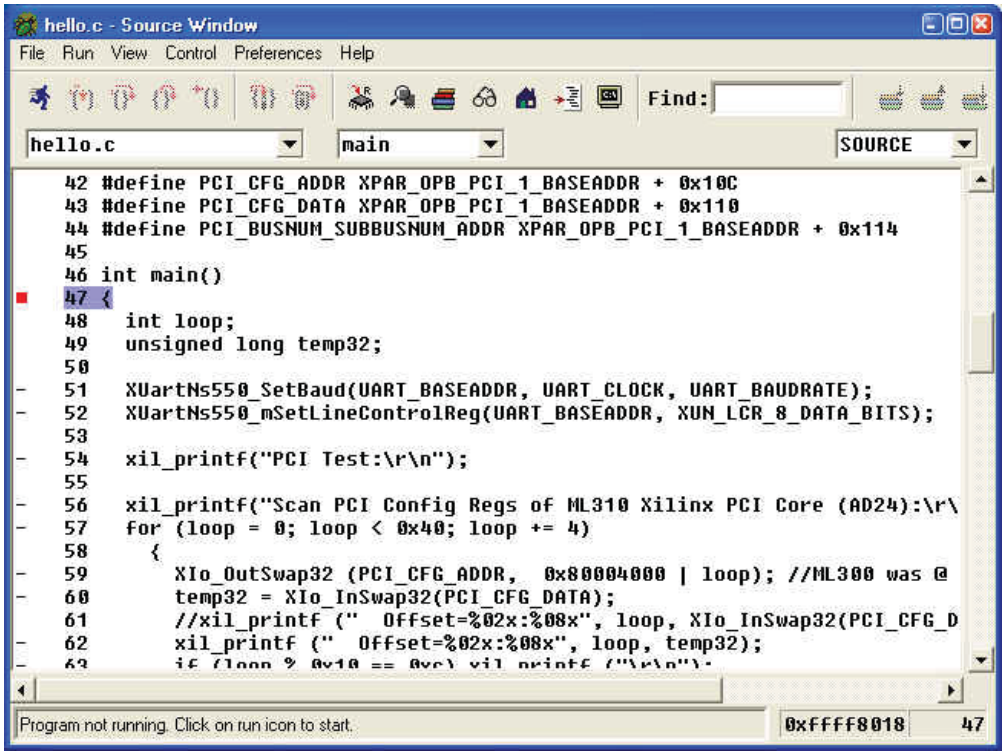


Figure 13: **Running hello_pci in gdb**

Using ChipScope with OPB PCI

To facilitate the use of ChipScope to analyze OPB PCI hardware, the `opb_pci.cdc` file is included in the `ml410_ppc_opb_pci /chipscope` directory. The opb_pci.cdc is used to insert a ChipScope ILA core into the pci32_bridge_wrapper core. To insert a core and analyze OPB PCI problems with ChipScope, execute the following steps:

1. Invoke XPS. **Run Hardware** → **Generate Netlist**.

2. In the `opb_pci.cdc` file, change the path `<design_directory>` name to the directory in which the design files are installed. Three paths need to be changed.

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

4. From ChipScope Inserter, run File → Open Project opb_pci.cdc.

Figure 14 shows the ChipScope Inserter setup GUI.

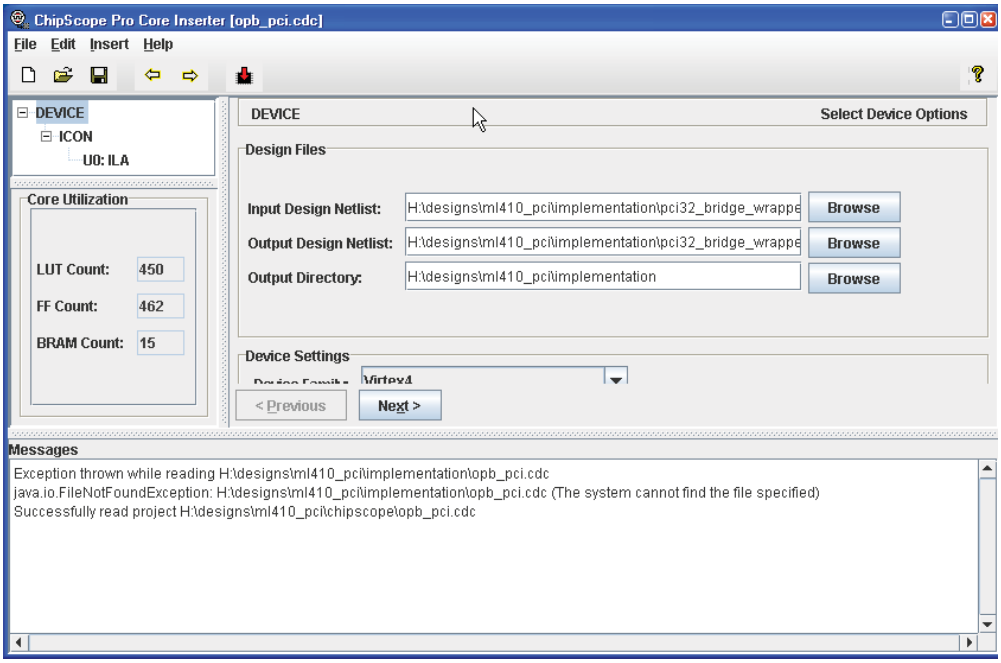


Figure 14: **ChipScope Inserter Setup**

The PCI_Monitor signals are the PCI bus signals. The PCI_Monitor signals are used in ChipScope to monitor signals AD, CBE, and the remaining PCI Bus signals. Table 7 defines the functionality of the PCI_Monitor signals which are generally useful to add to ChipScope. The Filter Pattern *PCI_Monitor* is used to locate the signals.

Table 7: **PCI Monitor Signals**

Bit Position	PCI Signal
0	FRAME_N
1	DEVSEL_N
2	TRDY_N
3	IRDY_N
4	STOP_N
5	IDSEL_int
6	INTA
7	PERR_N
8	SERR_N
9	Req_N_toArb
10	PAR
11:42	AD
43:47	CBE

5. Figure 15 shows the GUI for making net connections. Click **Next** to move to the Modify Connections window. If there are any red data or trigger signals, correct them. The Filter Pattern can be used to find net(s). As an example of using the Filter Pattern, enter **`*AD*`** in the dialog box to locate AD signals. In the Net Selections area, select either Clock, Trigger, or Data Signals. Select the net and click **Make Connections**.

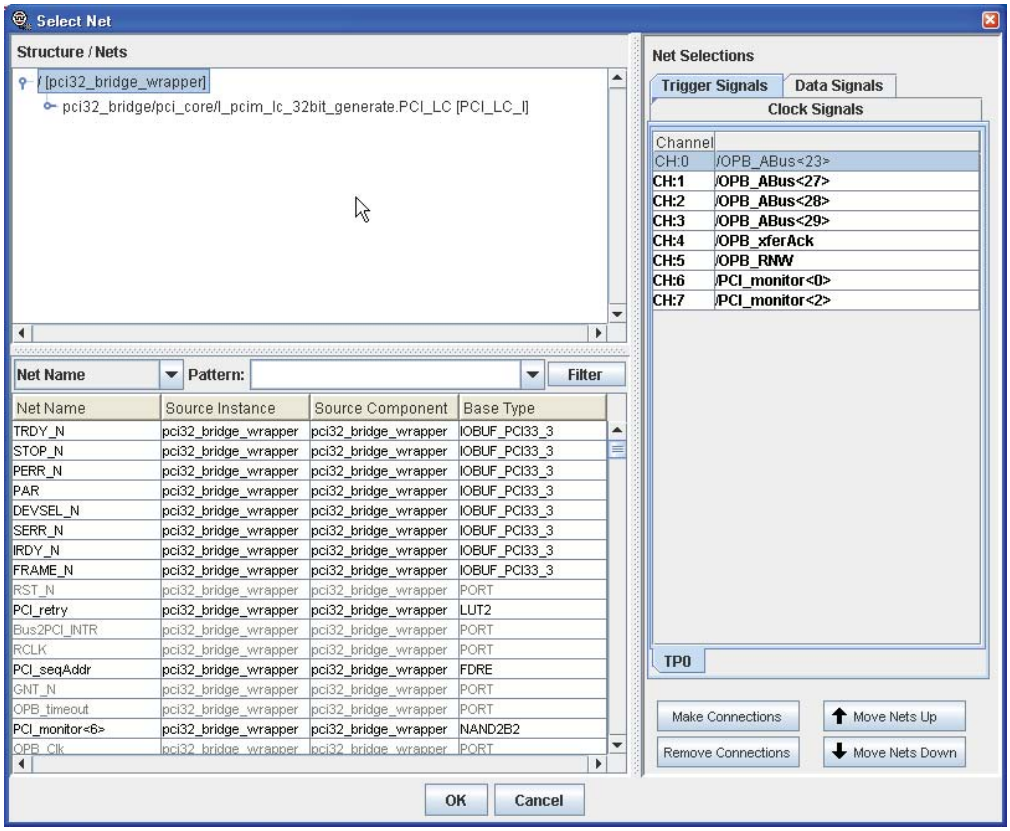


Figure 15: **Making net connections in ChipScope Inserter**

6. Click **Insert Core** to insert the core into pci32_bridge_wrapper.ngo. In the `ml410_pci/implementation` directory, copy `pci32_bridge_wrapper.ngo` to `pci32_bridge_wrapper.ngc`.

8. In XPS, run **Hardware** → **Generate Bitstream** and **Device Configuration** → **Download Bitstream**. Do not rerun **Hardware** → **Generate Netlist**, as this overwrites the `implementation/pci32_bridge_wrapper.ngc` produced by the step above. Verify that the file size of the pci32_bridge_wrapper.ngc with the inserted core is significantly larger than the original version.

9. Invoke ChipScope Pro Core Analyzer by selecting

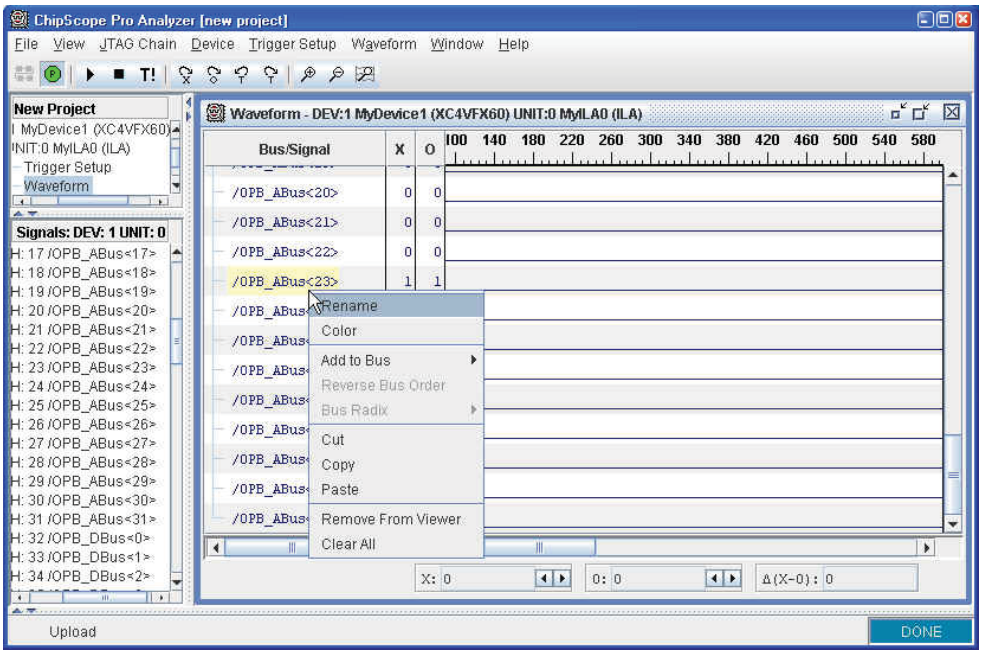
Start → **Programs** → **ChipScope Pro** → **ChipScope Pro Analyzer**

Click on the Chain icon located at the top left of Analyzer GUI. Verify that the message in the transcript window indicates that an ICON is found.

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

The waveform viewer is more readable when buses rather than discrete signals are displayed. As shown in Figure 16, select the **32 OPB_ABus<*>** signals, click the right mouse button, and select **Add to Bus** → **New Bus**. With **OPB_ABus<0:31>** in the waveform viewer, select and remove the **32 discrete OPB_ABus<*>** signals. Do this for the OPB_DBus. Make PCI Bus signals by creating a new bus for PCI_Monitor(43:47) and renaming it PCI_Monitor_CBE, and for PCI_Monitor(11:42) and renaming it PCI_Monitor_AD.

Note: In Figure Figure 16, the Reverse Bus Order operation option shown in below the Add Bus option is useful for analyzing ChipScope results.



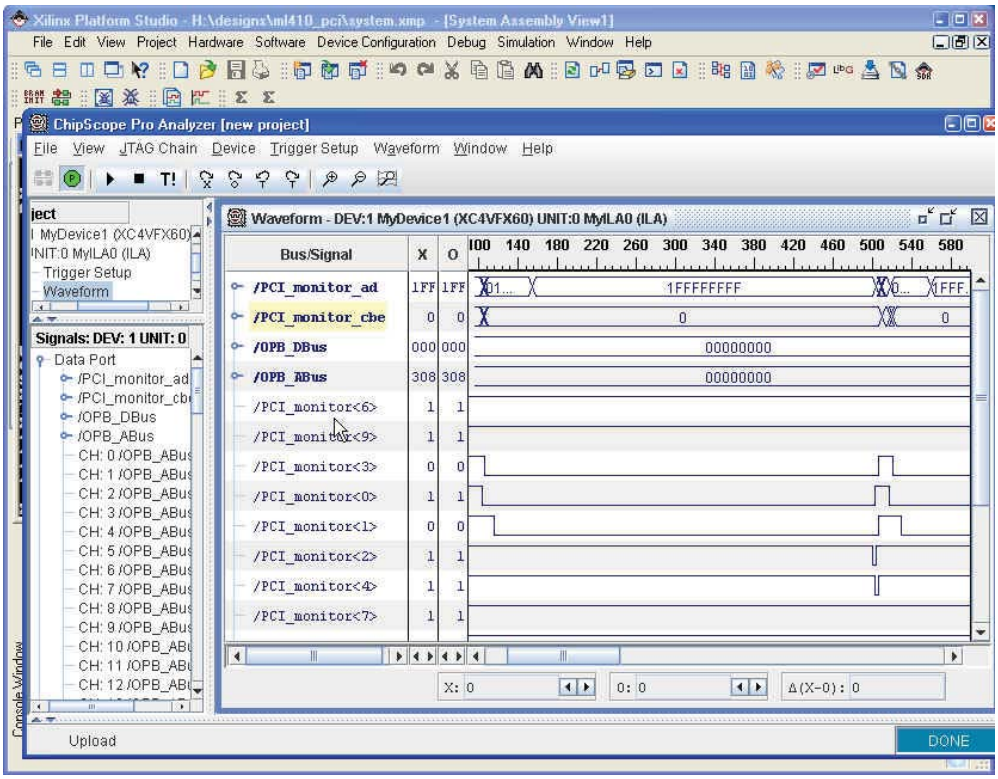
X964_16_111406

Figure 16: **Creating Buses from Discrete Signals**

11. Set the trigger in the Trigger Setup window. The trigger used depends on the problem being debugged. For example, if debugging a configuration transaction from the OPB, trigger on an OPB address of C_BASEADDR + 0x10C. If debugging a problem configuring from the PCI side, trigger on the PCI_Monitor(43:47) for a configuration write on CBE. Change the **Windows to N samples** to a setting of 500. Arm the trigger by selecting **Trigger Setup** → **Arm**, or clicking on the **Arm** icon.

12. Run **XMD** or **GDB** to activate the trigger patterns which cause ChipScope to display meaningful output. For example, invoke **`XMD`**, enter **`rst,`** and paste the contents of `configure*.xmd` into the XMD window.

13. ChipScope results are analyzed in the waveform window, as shown in Figure 17. This figure shows the original PCI_monitor<*> signals and the PCI_monitor_ad signal generated in Step 10. The waveforms may be easier to read if the discrete PCI_monitor<*> signals are removed after they are renamed. 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 Modeltech reads a vcd file and generates a wlf file for viewing in the Modeltech waveform viewer. The vcd file can be opened in the Cadence Design System, Inc. Simvision design tool by selecting **File** → **Open Database**.



X964_17_111506

Figure 17: **ChipScope Analyzer Results**

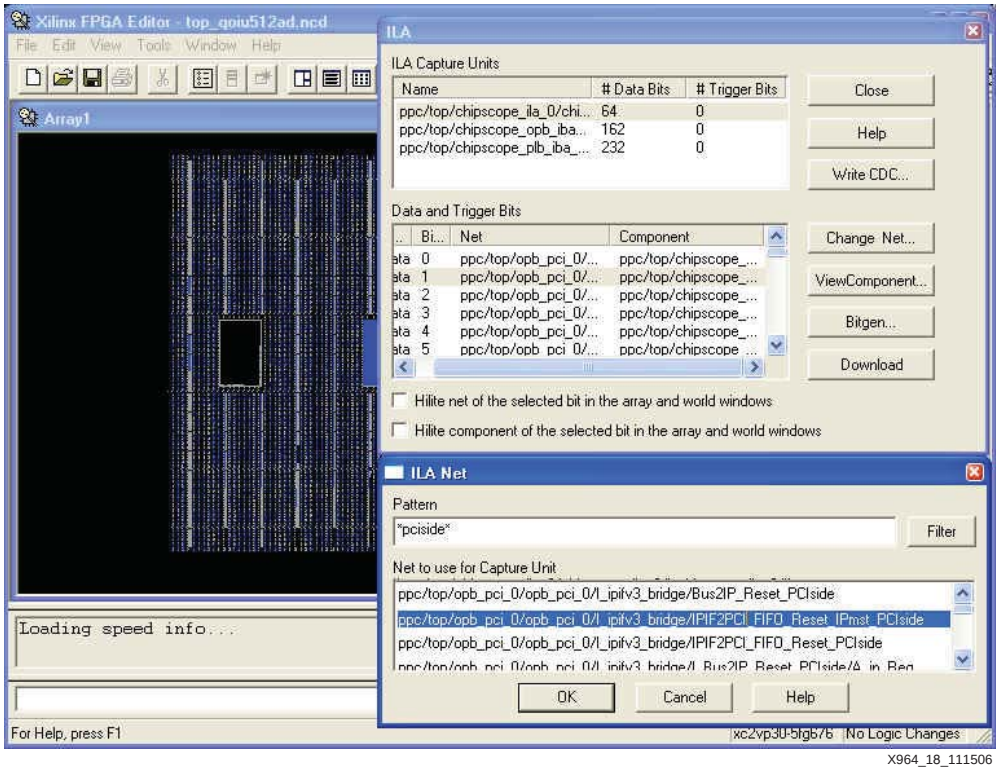
After running ChipScope, it is sometimes necessary to revise the trigger or data nets, or both, used in a debug operation. To revise nets used by ChipScope Analyzer enter the command:

```
fpga_editor <system>.ncd
```

From the FPGA Editor GUI, select **Tools** → **ILA.**

Select an existing net which is not needed in debugging, and click **Change Net**. The pattern filter box shown in Figure 18 facilitates the selection of a new net(s). Click **Write CDC** to generate an new `opb_pci.cdc` file. Click **Bitgen** to generate a new .bit file.

The FPGA Editor ILA flow is more efficient than regenerating the Chip Inserter flow listed above because the MAP and PAR implementation phases are not required.



X964_18_111506

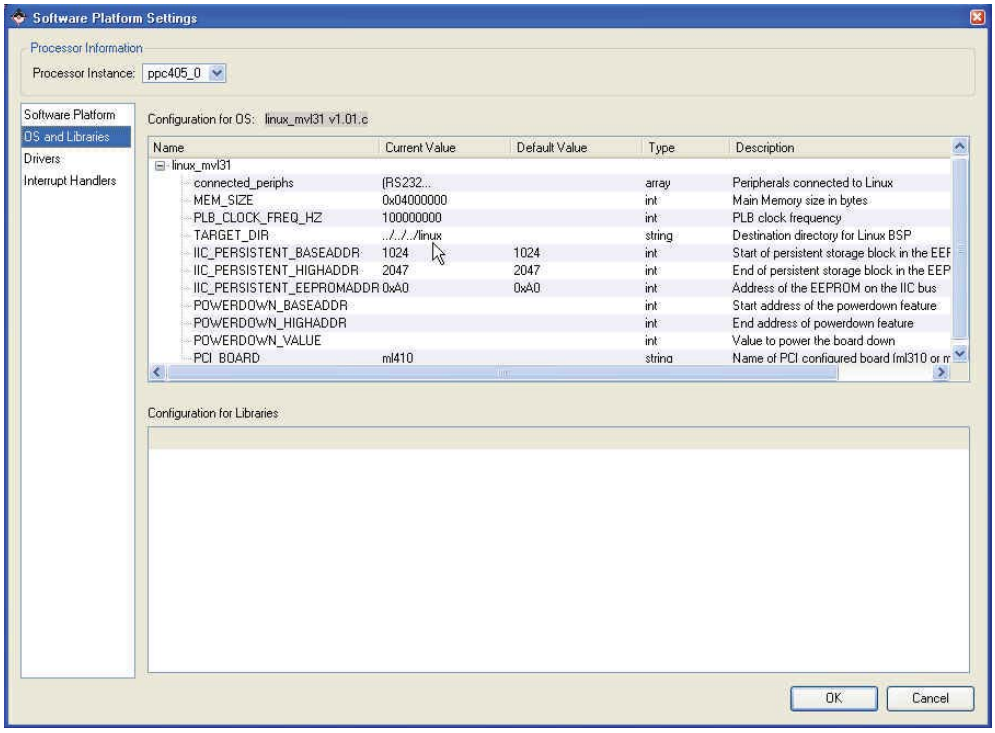
Figure 18: **Using FPGA Editor to Revise Nets used in ChipScope Analyzer**

Linux Kernel

New users of Monta Vista Linux should read XAPP 765. The steps to build and boot a Linux kernel are given below. Steps 1-3, 7, 8 are run on a Linux machine with MontaVista Professional Edition installed.

1. Add `/opt/montavista/pro/host/bin` and `/opt/montavista/pro/devkit/ppc/405/bin` to $PATH.
2. Change to the `ml410_pci/linux` directory.
3. Run

   `tar cf - -C /opt/montavista/pro/devkit/lsp/xilinx-ml300-ppc_405/linux-2.4.20_mvl31/ . tar xf -`
4. To generate the Linux LSP in XPS, enter **Software → Software Platform Settings. Select Kernel and Operating Systems**, then select **linux_mvl31 v1.00.c**.
5. Under **OS and Libraries**, set the entries as shown in Figure 19.

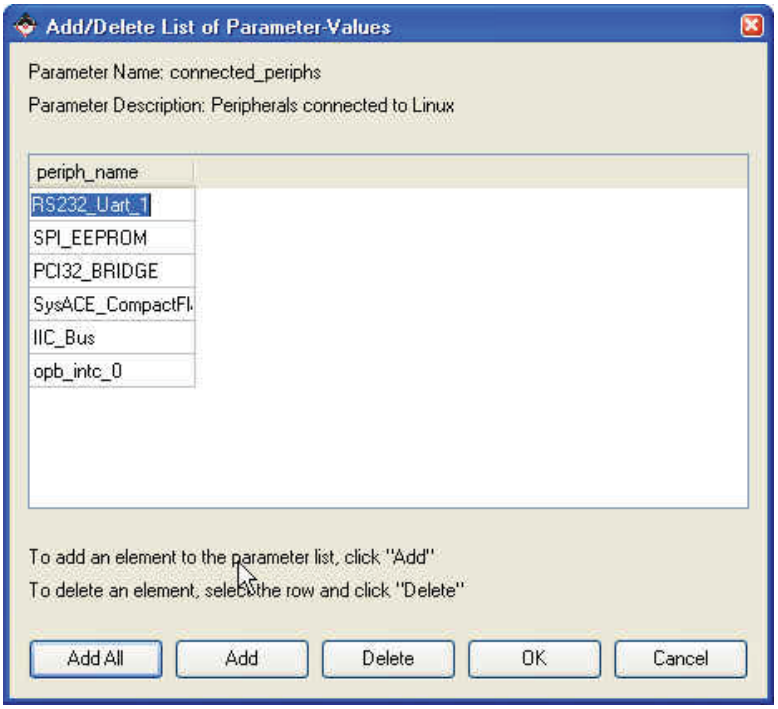


X964_19_111506

Figure 19: **BSP Settings**

Verify that the target directory is the same as the directory containing the Linux source.

6. Click **Connect_Periphs** and add the OPB_INTC, OPB_SYSACE, OPB_PCI, OPB_SPI, OPB_IIC, and OPB 16550 peripherals, using the instance names shown in Figure 20.



X964_20_111506

Figure 20: **Connected Peripherals**

Click **OK**.

7. Select **Software** → **Generate Libraries and BSPs** to generate the LSP in `ml410_ppc_opb_pci/linux`.
8. From ml410_ppc_opb_pci/linux, run `patch_nobspgen`.
9. The `ml410_ppc_opb_pci/linux/.config` is used to define the contents of the Linux kernel. Run `make oldconfig`.

An alternative is to enter **`make menuconfig`** and generate a new `.config` using the following options.

Select General Setup

Enable PCI. Disable PS/2 keyboard. Change to /dev/ram for booting from ramdisk.

Select ATA/IDE/MFM/RLL support.

Enable Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support.

Enable CMD640 chipset bugfix/support and CMD640 enhanced support.

Enable Include IDE/ATAPI CDROM support. Enable Generic PCI IDE chipset support.

Enable Include IDE/ATA-2 DISK support.

Enable ALI M15x3 chipset support.

Enable PROMISE PDC202 {46|62|65|68|69|70} support.

Enable SCSI support. Enable SCSI disk support.

Enable SCSI CD-ROM support.

Enable SCSI generic support.

Enable SCSI low-level drivers.

Enable Adaptec AHA152X/2825, Adaptec AHA1542, and Adaptec AHA1740 support.

Select Network Device Support → Ethernet (10 or 100), enable 3Com devices.

Enable Vortex if using the 3Com PCI card.

Enable EISA, VLB, PCI and on board controllers.

Enable DECchip Tulip (dc2lx4x) PCI, support, EtherExpressPro/100 support, National Semiconductor DB8381x..., and SMC EtherPowerII

Select Console Drivers. Disable Frame Buffer Support.

Select Input Core Support. Disable all.

Select Character Devices. Disable Virtual. Leave Serial enabled. Disable Xilinx GPIO and Touchscreen.

Enable USB support.

10. Run **`make clean dep zImage.initrd`**. Verify that the `zImage.initrd.elf` file is in `the ml410_ppc_opb_pci/linux/arch/ppc/boot/images` directory.
11. Invoke **Impact** and download `implementation/download.bit` to XC4VFX60. Either select **Device Configuration** → **Download Bitstream** from XPS or run the following command from the command prompt:

impact -batch etc/download.cmd

12. Invoke XMD. From the `ml410_pci/linux_pci` directory, enter the following commands in the XMD window:
    `rst`

```
dow arch/ppc/boot/images/zImage.initrd.elf
con
```

13. View the output in the HyperTerminal window. Login as `root`. Enter **`cd /`** and **`ls -l`** to view the contents of the mounted Linux partition.
14. Enter **`./lspci -vv`** to view the PCI devices. For each line of output, the first 2 digits represent the PCI bus number, followed by the device number and function number.
15. An alternative to downloading the Linux kernel executable is to load it into CompactFlash. The file used uses an ace file extension. To generate an ace file, run the command below from the `ml410_ppc_opb_pci` directory.

```
xmd -tcl ../genace.tcl -jprog -hw ../implementation/system.bit -ace
../implementation/ace_system_hw.ace -board ML410
```

Copy the ace file to a 64-512 MB CompactFlash (CF) card in a CompactFlash reader/writer. Remove the CF card from the CF reader/writer and insert it into the CompactFlash slot (J22) on the ML410 board. Power up the board.

Simulation

The bus transactions done by the OPB PCI Bridge are described the *UG241 OPB PCI User Guide*. The `ml410_ppc_opb_pci/simulation` directory contains waveform log files (files with .wlf extension) for the following types of transactions:

- Configuration from the OPB side
- Configuration from the PCI side
- OPB to PCI Write
- OPB to PCI Read
- PCI to OPB Write
- PCI to OPB Read

Load the *.wlf files into the Modeltech simulator by typing the **File** → **Open** command, then specifying the *.wlf file type.

See the UG241 *OPB PCI User Guide* for a detailed definition of each transaction.

References

DS437 OPB IPIF/LogiCore v3.0 PCI Core Bridge (v1.02a)

Xilinx LogiCore PCI Interface v3.0 Product Specification

Xilinx The Real PCI Design Guide v3.0

DS416 Direct Memory Access and Scatter Gather

Virtex-4 ML455 PCI/PCI-X Development Kit User Guide UG084 (v1.0) May 17, 2005

XAPP765 Getting Started with EDK and MontaVista Linux

ML41x Embedded Development Platform User Guide UG085 (v1.2) May 26, 2006

ChipScope ILA Tools Tutorial

UG241 OPB PCI User Manual

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/5/06	1.0	Initial Xilinx release.
1/9/07	1.1	Corrected spelling of author's name.