XAPP945 (v1.1) February 8, 2008

Reference System: PLB PCI Using the ML410 Embedded Development Platform

Author: Lester Sanders

Summary

This application note describes how to build a reference system for the Processor Local Bus (PLB) Peripheral Component Interconnect (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 six pcores.

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

Included Systems

This application note includes one reference system.

www.xilinx.com/support/documentation/application_notes/xapp945.zip

The project name used in xapp945.zip is ml410_ppc_plb_pci.

Required Hardware and Tools

Users must have the following tools, cables, peripherals, and licenses available and installed. A PLB PCI evaluation license is shipped with EDK.

- Xilinx ML410 Evaluation Platform Rev C
- Xilinx ML555 Evaluation Platform Rev A
- Xilinx EDK 9.2.02i
- Xilinx ISE™ 9.2.04i
- Xilinx Download Cable (Platform Cable USB or Parallel Cable IV)
- Monta Vista Linux v2.6 Development Kit
- Model Technology ModelSim v6.1d
- ChipScope Pro Analyzer 9.2.01
- PLB PCI License

Introduction

PCI transactions are done between an initiator and a target. This reference design is for the ML410 Embedded Development Platform. To be useful, a target board should be inserted into a PCI slot. In the examples provided in this application note, the ML555 Embedded Development Platform is inserted into PCI slot P3 of the Xilinx ML410 Evaluation Platform. This allows both configuration and memory transactions to be done on the PCI bus between an initiator and a target. The examples use the ML410 PLB PCI as the initiator and the ML555 PLBv46 PCI as the target. An Avnet Spartan-3 Evaluation board can be substituted for the ML555 Embedded Development Platform.

As of the EDK9.2.01 release in November 2007, Xilinx recommends using the PLBv46 PCI core rather than the PLB PCI core. A reference system for the PLBv46 PCI core is provided in XAPP1001.

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

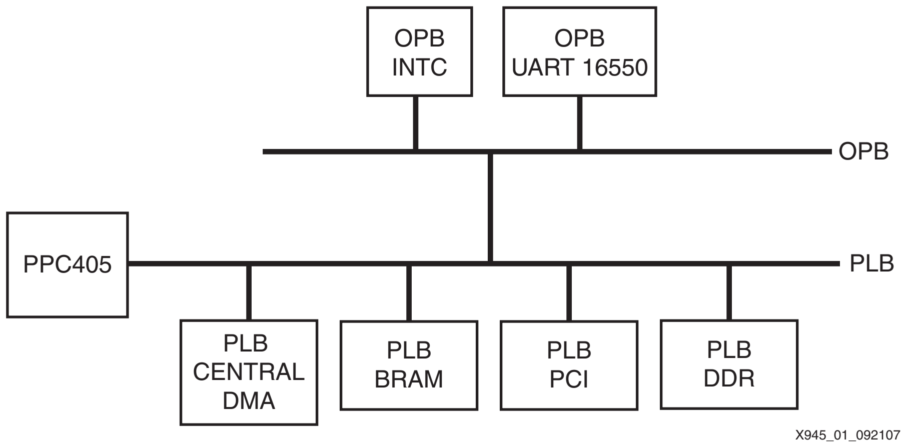


Figure 1: **ML410 PLB PCI Reference System Block Diagram (add spi_iic_gpio_sysace)**

The system uses the embedded PowerPC as the microprocessor and the PLB 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.

PCI configuration in this reference design uses the PLB PCI Bridge as a host bridge.

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 ML455 PCI/PCI-X Board is inserted into ML410 PCI slot P3.

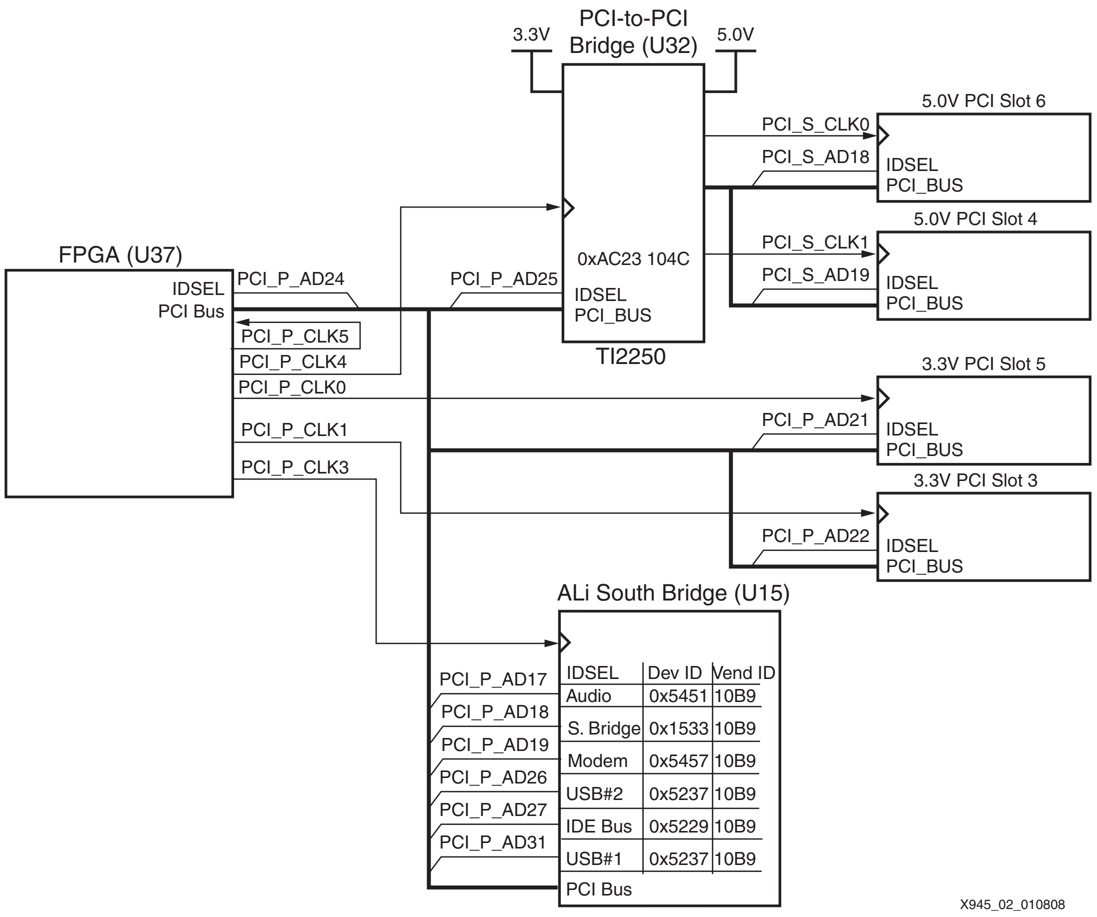
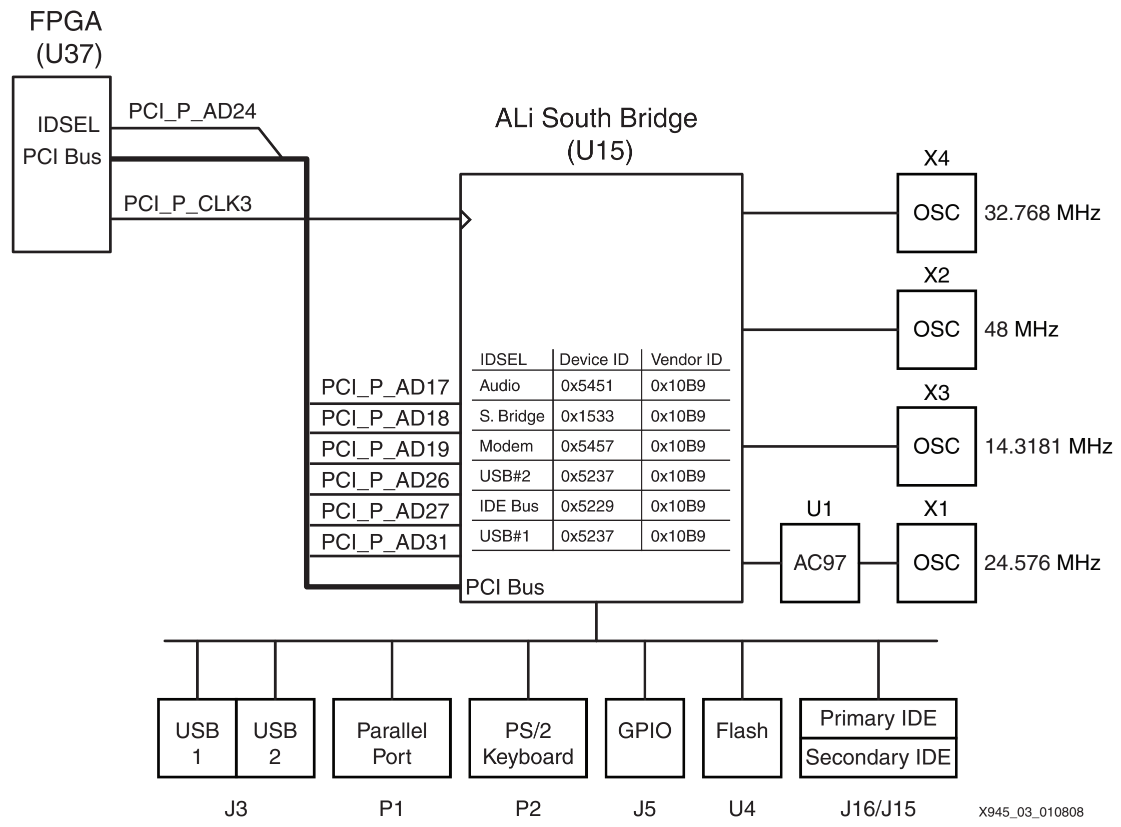


Figure 2: **ML410 PLB PCI Reference System Block Diagram**

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



IDSEL	Device ID	Vendor ID
Audio	0x5451	0x10B9
S. Bridge	0x1533	0x10B9
Modem	0x5457	0x10B9
USB#2	0x5237	0x10B9
IDE Bus	0x5229	0x10B9
USB#1	0x5237	0x10B9

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

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

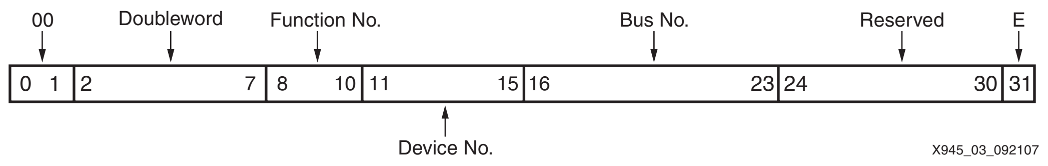


Figure 4: **Format of Configuration Address Port**

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

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 PowerPC 405 processor and PLB PCI, this system includes DDR, BRAM, 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	AD24
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	N/A	N/A			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 provides the address map of the ML410 XC4VFX60.

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

Peripheral	Instance	Base Address	High Address
PLB_DDR2	DDR_SDRAM_32Mx64	`0x00000000`	`0x03FFFFFF`
OPB UART16550	RS232_Uart_1	`0x40400000`	`0x4040FFFF`
OPB INTC	OPB_intc_0	`0x41200000`	`0x4120FFFF`
PLB_PCI	PCI32_Bridge	`0x42600000`	`0x4260FFFF`

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

Peripheral	Instance	Base Address	High Address
PLB Central DMA	plb_central_dma_0	`0x50000000`	`0x5000007F`
PLB BRAM	plb_bram_if_cntlr_0	`0xFFFF0000`	`0xFFFFFFFF`
OPB SPI	SPI_EEPROM	`0x41308000`	`0x413080FF`
OPB GPIO	LEDs_8Bit	`0x40000000`	`0x4000FFFF`
OPB SysAce	SyaACE_CompactFlash	`0x418000000`	`0x4180FFFF`
OPB IIC	IIC_Bus	`0x40800000`	`0x4080FFFF`

The reference design contains the settings for PLB PCI generics:

C_INCLUDE_PCI_CONFIG = 1

C_INCLUDE_BAROFFSET = 0

C_IPIFBAR_NUM = 2

C_PCIBAR_NUM = 2

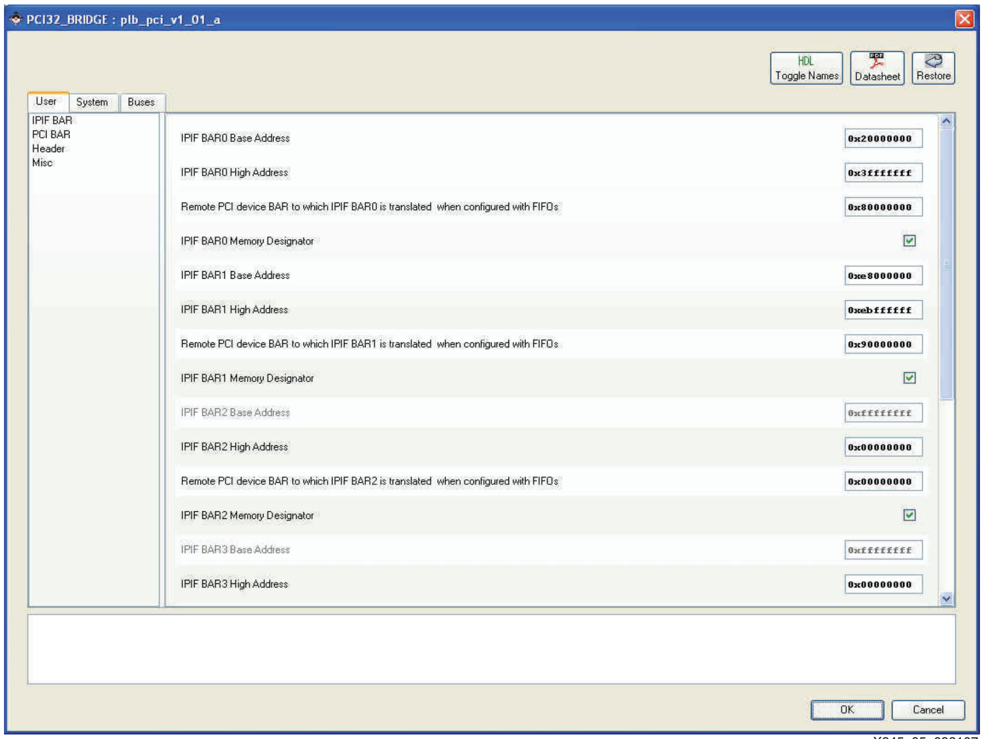
C_IPIFBAR_0 = `0x20000000`

C_IPIFBAR2PCIBAR_0 = `0x80000000`

C_IPIFBAR_1 = `0xE80000000`

C_IPIFBAR2PCIBAR_1 = `0x90000000`

Figure 5 shows how to specify the values of Base Address Register (BAR) generics in EDK.



IPIF BAR0 Base Address	0x20000000
IPIF BAR0 High Address	0x3fffffff
Remote PCI device BAR to which IPIF BAR0 is translated when configured with FIFOs	0x80000000
IPIF BAR0 Memory Designator	✓
IPIF BAR1 Base Address	0xe8000000
IPIF BAR1 High Address	0xebffffff
Remote PCI device BAR to which IPIF BAR1 is translated when configured with FIFOs	0x90000000
IPIF BAR1 Memory Designator	✓
IPIF BAR2 Base Address	0xffffffff
IPIF BAR2 High Address	0x00000000
Remote PCI device BAR to which IPIF BAR2 is translated when configured with FIFOs	0x00000000
IPIF BAR2 Memory Designator	✓
IPIF BAR3 Base Address	0xffffffff
IPIF BAR3 High Address	0x00000000

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

Setting C_INCLUDE_PCI_CONFIG = 1 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_IPIFBAR_NUM = 2 specifies that there are two address ranges for PLB to PCI transactions. Setting C_PCIBAR_NUM = 2 specifies that two address ranges are used for PCI to PLB transactions.

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

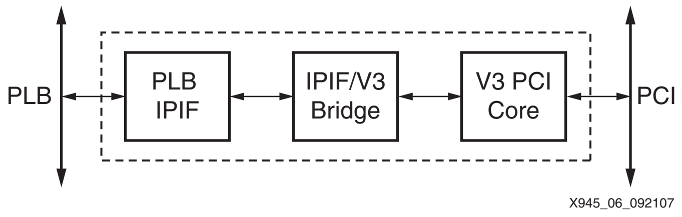


Figure 6: **PLB PCI Functional Diagram**

Implementation Results

The resource utilization in the reference design is shown in Table 4.

Table 4: **Design Resource Utilization**

Resources	Used	Available	Utilization (%)
Slice registers	8259	50560	16
Slice LUTs	8209	50560	18
DCM_ADV	2	12	16
Block RAM	40	232	17

Setting C_INCLUDE_PCI_CONFIG = 1 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_IPIFBAR_NUM = 2 specifies that there are two address ranges for PLB to PCI transactions. Setting C_PCIBAR_NUM = 2 specifies that two address ranges are used for PCI to PLB transactions.

ML555 PCI/PCI Express Evaluation Platform

In this reference design, the PLB PCI in the XC4VFX60 on the ML410 board interfaces to the PCI core in the Virtex-5 ML555 PCI/PCI Express Embedded Development Platform. The ML555 board uses the Xilinx XC5VLX50T device in the 1136 pin package.

Table 5 provides the address map for the XC5VLX50T.

Table 5: **ML555 Address Map**

Instance	Base Address	High Address
DLMB_CNTLR/ILMB_CNTLR	`0x00000000`	`0x00001FFF`
RS232_Uart_1	`0x84000000`	`0x8400FFFF`
pci_0	`0x42600000`	`0x4260FFFF`
DDR_SDRAM_64Mx32	`0x90000000`	`0x9FFFFFFF`
LEDs_8Bit	`0x81400000`	`0x8140FFFF`
debug_module	`0x84400000`	`0x8440FFFF`

Table 5: **ML555 Address Map**

Instance	Base Address	High Address
xps_intc_0	`0x81800000`	`0x8180FFFF`
xps_central_dma_0	`0x81810000`	`0x8181FFFF`
xps_bram_if_cntlr_1	`0x8A208000`	`0x8A20FFFF`

The ML555 includes a 64-bit PCI edge connector, 128 MB (or 256 or 512 MB) DDR2 SDRAM memory, RS232C port, LED displays, XCF32P-FSG48C Platform Flash configuration PROM, and a JTAG port. The MicroBlaze microprocessor is used.

Figure 7 shows the principle interface blocks when transferring data between the PLB PCI Bridge in the XC4VFX60 on the ML410 board and the PCI Bridge in the XC5VLX50T on the ML555 board.

The application note, XAPP999 Reference System: PLBv46 PCI in a ML555 Embedded Development Platform, provides a link to the ML555 system.

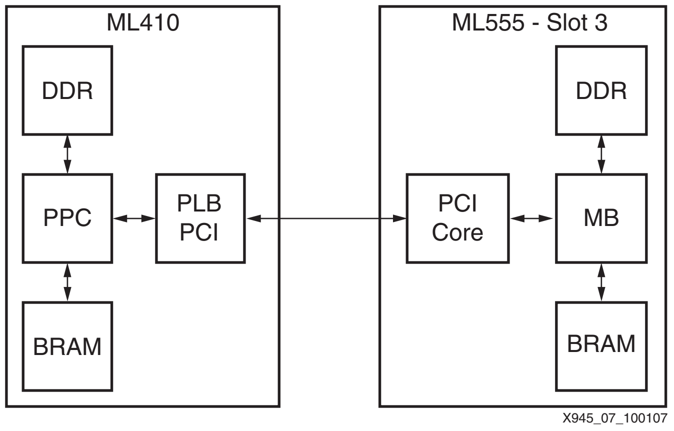


Figure 7: **Interfacing ML410 PLB PCI with ML555 PCI**

Configuration of PLB PCI on the ML410 Board

For the PLB PCI bridge to perform transactions on the PCI bus, the PCI LogiCORE IP v3.0 core must be configured using configuration transactions from either the PCI-side or from the PLB side. In this reference design the ML410 PLB PCI is configured as a host bridge from the PLB side. The IDSEL input of the v3.0 is connected to the address ports specified in Table 2, and the IDSEL port of the PLB PCI is unused.

The following steps are used to write the configuration header.

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, set the memory space bit. For I/O transactions, set the I/O space bit.

2. Configure the Latency Timer to a non-zero value, usually 0xFF.

3. Configure at least one BAR. Configure additional 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 PCI on the ML555 PCI/PCI-X Board

When the ML555 is inserted into the ML410 PCI slot P3, the PLB PCI Bridge in the XC4VFX60 FPGA interfaces to an PLB PCI Bridge in the XC5VLX50T FPGA. To configure the XC5VLX50T, connect the Xilinx Download (USB or Parallel IV) cable to the ML555 JTAG port, and use Impact to download the `download.bit` file. After downloading the XC5VLX50T FPGA bit file, the PCI functionality in the ML555 is configured using Configuration write transactions from the ML410 PLB PCI.

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

Use the steps below to run the system.

1. Change to the `ml410_ppc_plb_pci/ready_for_download` directory.
2. Use iMPACT to download the bitstream.
   `impact -batch xapp945.cmd`
3. Invoke XMD and connect to the MicroBlaze processor.
   `xmd -opt xapp945.opt`
4. Download the executable.
   `dow ready_for_download/pci_dma.elf`

or

`dow ready_for_download/hello_pci.elf`

Executing the Reference System from EDK

Use the steps below to execute the system using XPS.

1. Select **File** → **Open** system.xmp.
2. Use **Hardware** → **Generate Bitstream** to generate a bitstream.
3. Download the bitstream to the board using **Device Configuration** → **Download Bitstream**.
4. Run **Device Configuration** → **Update Bitstream** to generate bootloop.
5. Build `hello_pci` and `pci_dma` software applications.
6. Invoke XMD with **Debug** → **Launch XMD**.
7. Download the executable by the following command.
   `dow hello_pci/executable.elf`

or

`dow pci_dma/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.

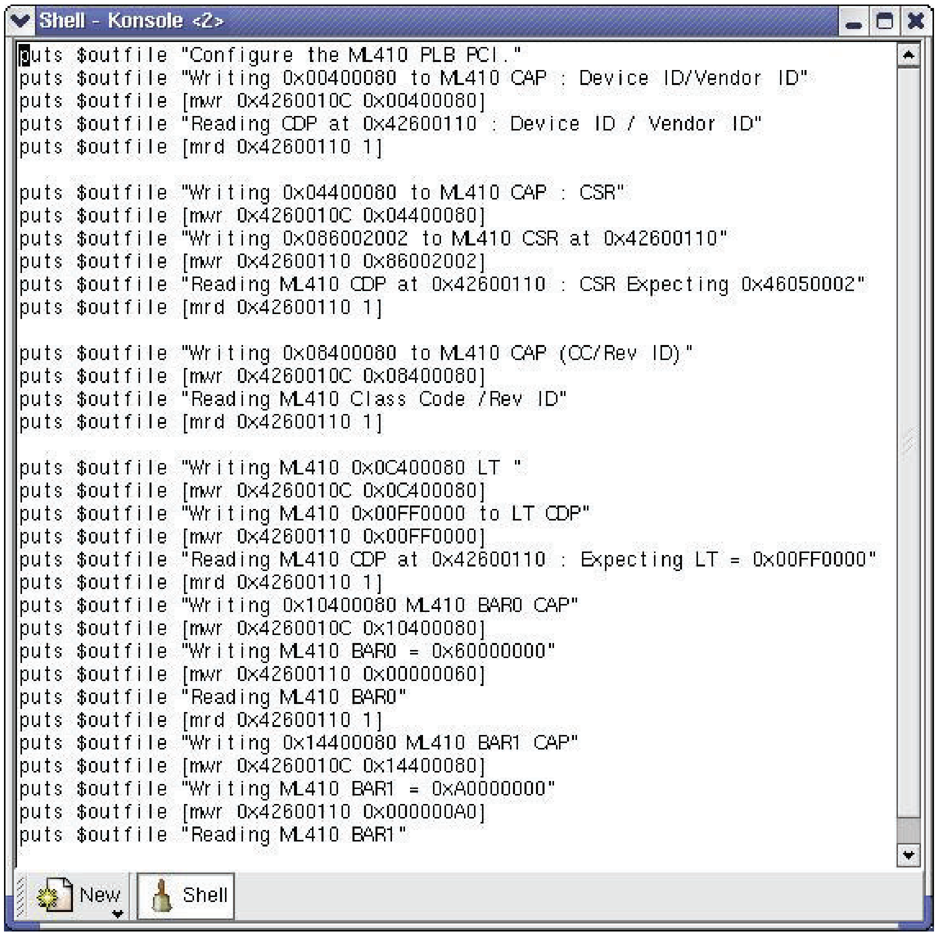
1. Configure the v3.0 Command Register, Latency Timer, and BAR(s).
2. Read the configuration header.
3. Configure the Command Register, Latency Timer, and BAR(s) of the other devices in the system.
4. Read the configuration headers of the other devices in the system.
5. Perform a memory read of one of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

6. Perform a memory write of one of the IPIF BARs.

Verification is done using either Xilinx Microprocessor Debugger (XMD) or the software projects discussed later. TCL scripts of XMD commands are provided in ml410_ppc_plb_pci/xmd_commands. The 410_555.tcl script configures and verifies the ML410 and ML555 PCI cores. To run this script from xmd_commands, enter

```
xmd -tcl 410_555.tcl
```

at the command prompt.The XMD commands in the `410_555.tcl` file, partially listed in Figure 8, 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 of the ML410 and ML555 PLBv46 PCI cores are written and read.



```
puts $outfile "Configure the ML410 PLB PCI."
puts $outfile "Writing 0x00400080 to ML410 CAP : Device ID/Vendor ID"
puts $outfile [mwr 0x4260010C 0x00400080]
puts $outfile "Reading CDP at 0x42600110 : Device ID / Vendor ID"
puts $outfile [mrd 0x42600110 1]

puts $outfile "Writing 0x04400080 to ML410 CAP : CSR"
puts $outfile [mwr 0x4260010C 0x04400080]
puts $outfile "Writing 0x086002002 to ML410 CSR at 0x42600110"
puts $outfile [mwr 0x42600110 0x86002002]
puts $outfile "Reading ML410 CDP at 0x42600110 : CSR Expecting 0x46050002"
puts $outfile [mrd 0x42600110 1]

puts $outfile "Writing 0x08400080 to ML410 CAP (CC/Rev ID)"
puts $outfile [mwr 0x4260010C 0x08400080]
puts $outfile "Reading ML410 Class Code /Rev ID"
puts $outfile [mrd 0x42600110 1]

puts $outfile "Writing ML410 0x0C400080 LT "
puts $outfile [mwr 0x4260010C 0x0C400080]
puts $outfile "Writing ML410 0x00FF0000 to LT CDP"
puts $outfile [mwr 0x42600110 0x00FF0000]
puts $outfile "Reading ML410 CDP at 0x42600110 : Expecting LT = 0x00FF0000"
puts $outfile [mrd 0x42600110 1]
puts $outfile "Writing 0x10400080 ML410 BAR0 CAP"
puts $outfile [mwr 0x4260010C 0x10400080]
puts $outfile "Writing ML410 BAR0 = 0x60000000"
puts $outfile [mwr 0x42600110 0x00000060]
puts $outfile "Reading ML410 BAR0"
puts $outfile [mrd 0x42600110 1]
puts $outfile "Writing 0x14400080 ML410 BAR1 CAP"
puts $outfile [mwr 0x4260010C 0x14400080]
puts $outfile "Writing ML410 BAR1 = 0xA0000000"
puts $outfile [mwr 0x42600110 0x000000A0]
puts $outfile "Reading ML410 BAR1"
```

X945_08_092107

Figure 8: **Excerpts from 410_555.tcl**

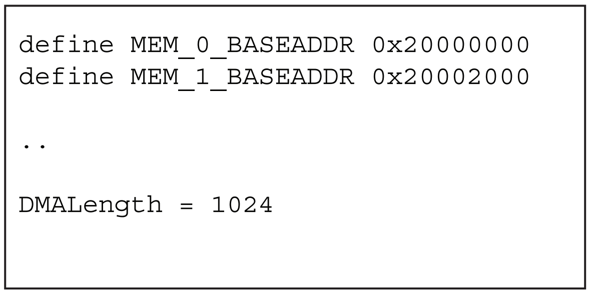
Software Projects

The reference system contains the following software projects.

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

pci_dma. This project runs DMA operations. The user sets the source address, destination address, and DMA length. This code is used for DMA operations between a variety of source and destination addresses.

Figure 9 shows the parameters in pci_dma.c which can be edited to run DMA transactions between different memory regions.



```
define MEM_0_BASEADDR 0x20000000
define MEM_1_BASEADDR 0x20002000

..

DMALength = 1024
```

X945_09_010808

Figure 9: **Defining Source and Destination Addresses, Length in pci_dma.c**

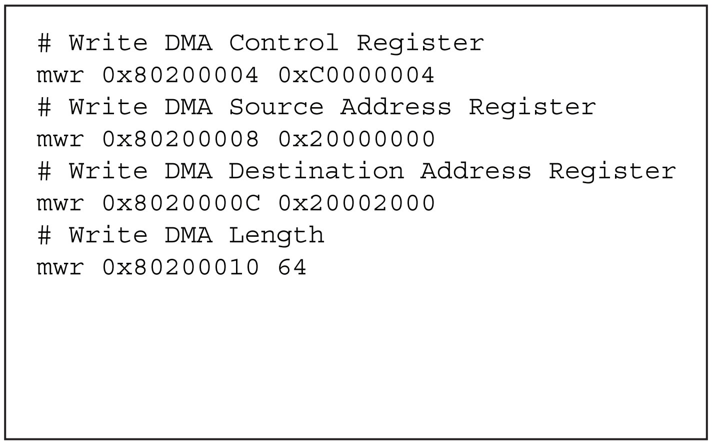
DMA Transactions

Many of the XMD scripts and C code examples generate Direct Memory Access (DMA) operations. DMA transactions are initiated by writing to the Control, Source Address, Destination Address, and Length registers of the DMA controller. Table 6 provides these register locations of the XPS Central DMA controller.

Table 6: **XPS Central DMA Registers**

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

An example of XMD code which generates DMA transactions is given in Figure 10.



```
# Write DMA Control Register
mwr 0x80200004 0xC0000004
# Write DMA Source Address Register
mwr 0x80200008 0x20000000
# Write DMA Destination Address Register
mwr 0x8020000C 0x20002000
# Write DMA Length
mwr 0x80200010 64
```

X945_10_010808

Figure 10: **Generating DMA Transactions**

The pci_dma.c code consists of the four functions in the functional diagram in Figure 11. The Barberpole 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.

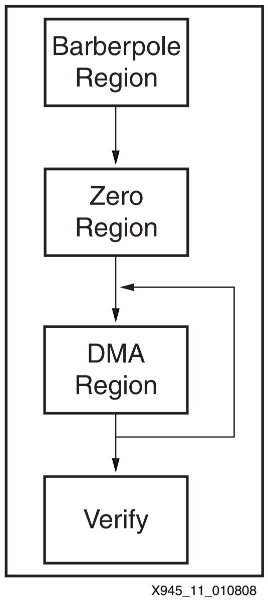
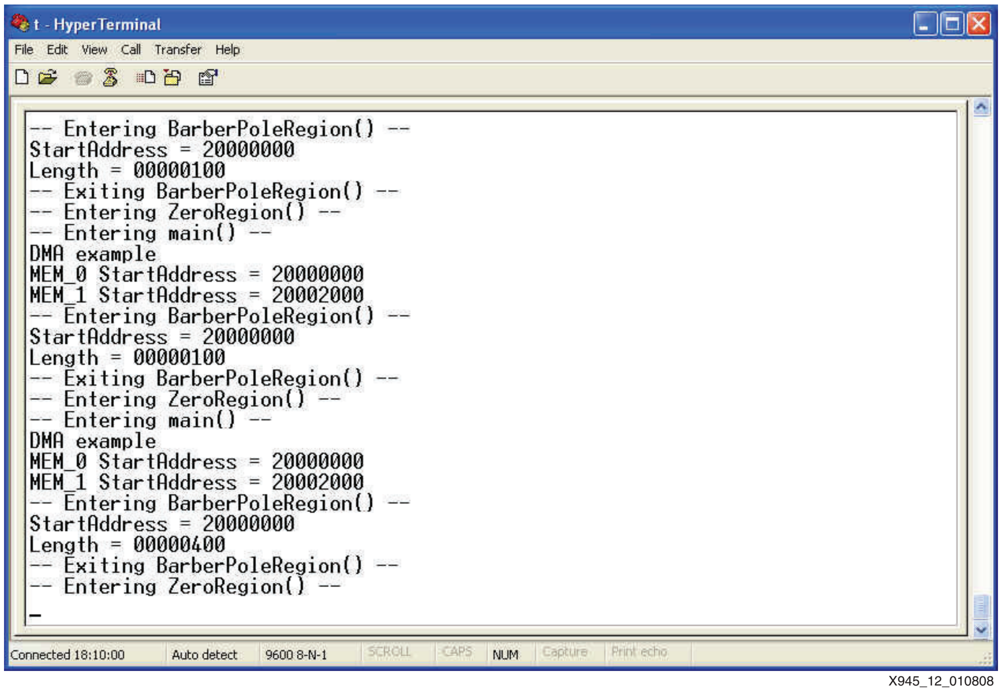


Figure 11: **Functional diagram of pci_dma.c**

Figure 12 show the Hyperterminal output when running the pci_dma/executable.elf. In the figure, the program is run twice, first with a length = 100, then with a length = 400.

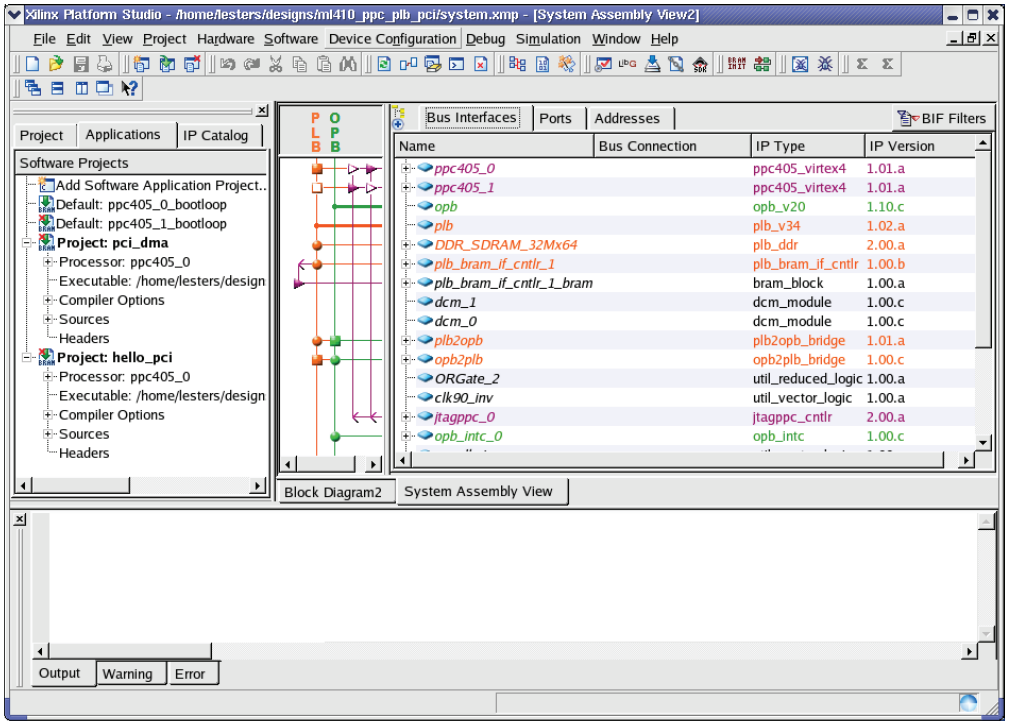


```
-- Entering BarberPoleRegion() --
StartAddress = 20000000
Length = 00000100
-- Exiting BarberPoleRegion() --
-- Entering ZeroRegion() --
-- Entering main() --
DMA example
MEM_0 StartAddress = 20000000
MEM_1 StartAddress = 20002000
-- Entering BarberPoleRegion() --
StartAddress = 20000000
Length = 00000100
-- Exiting BarberPoleRegion() --
-- Entering ZeroRegion() --
-- Entering main() --
DMA example
MEM_0 StartAddress = 20000000
MEM_1 StartAddress = 20002000
-- Entering BarberPoleRegion() --
StartAddress = 20000000
Length = 00000400
-- Exiting BarberPoleRegion() --
-- Entering ZeroRegion() --
_
```

Figure 12: **pci_dma.c output**

Running the Applications

The pci_dma and hello_pci projects are shown in Figure 13. Make the hello_pci project active and the remaining software projects inactive.


Xilinx Platform Studio - /home/lesters/designs/ml410_ppc_plb_pci/system.xmp - [System Assembly View2]

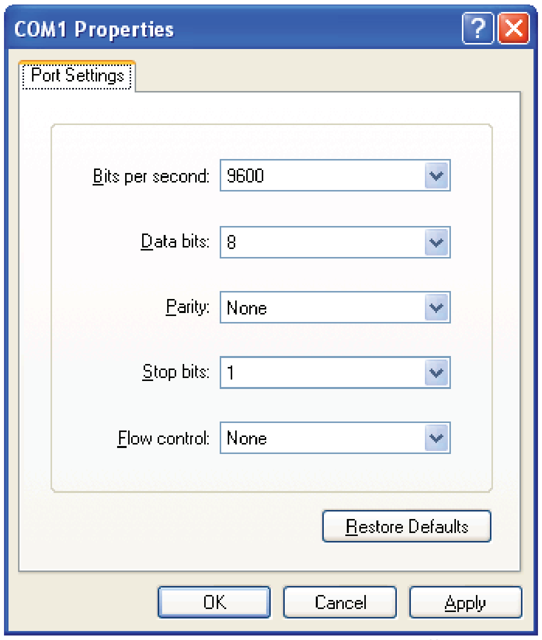
Software Projects
- Add Software Application Project..
- Default: ppc405_0_bootloop
- Default: ppc405_1_bootloop
- **Project: pci_dma**
  - Processor: ppc405_0
  - Executable: /home/lesters/design
  - Compiler Options
  - Sources
  - Headers
- **Project: hello_pci**
  - Processor: ppc405_0
  - Executable: /home/lesters/design
  - Compiler Options
  - Sources
  - Headers

Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	1.01.a
ppc405_1		ppc405_virtex4	1.01.a
opb		opb_v20	1.10.c
plb		plb_v34	1.02.a
DDR_SDRAM_32Mx64		plb_ddr	2.00.a
plb_bram_if_cntlr_1		plb_bram_if_cntlr	1.00.b
plb_bram_if_cntlr_1_bram		bram_block	1.00.a
dcm_1		dcm_module	1.00.c
dcm_0		dcm_module	1.00.c
plb2opb		plb2opb_bridge	1.01.a
opb2plb		opb2plb_bridge	1.00.c
ORGate_2		util_reduced_logic	1.00.a
clk90_inv		util_vector_logic	1.00.a
jtagppc_0		jtagppc_cntlr	2.00.a
opb_intc_0		opb_intc	1.00.c

X945_13_010808

Figure 13: **Software Projects**

With the hello_pci project selected, right click to build the project. Connect a serial cable to the RS232C port on the ML410 board. Start a HyperTerminal. Set the baud rate to **9600**, number of data bits to **8**, no parity, and no flow control, as shown in Figure 14.


COM1 Properties

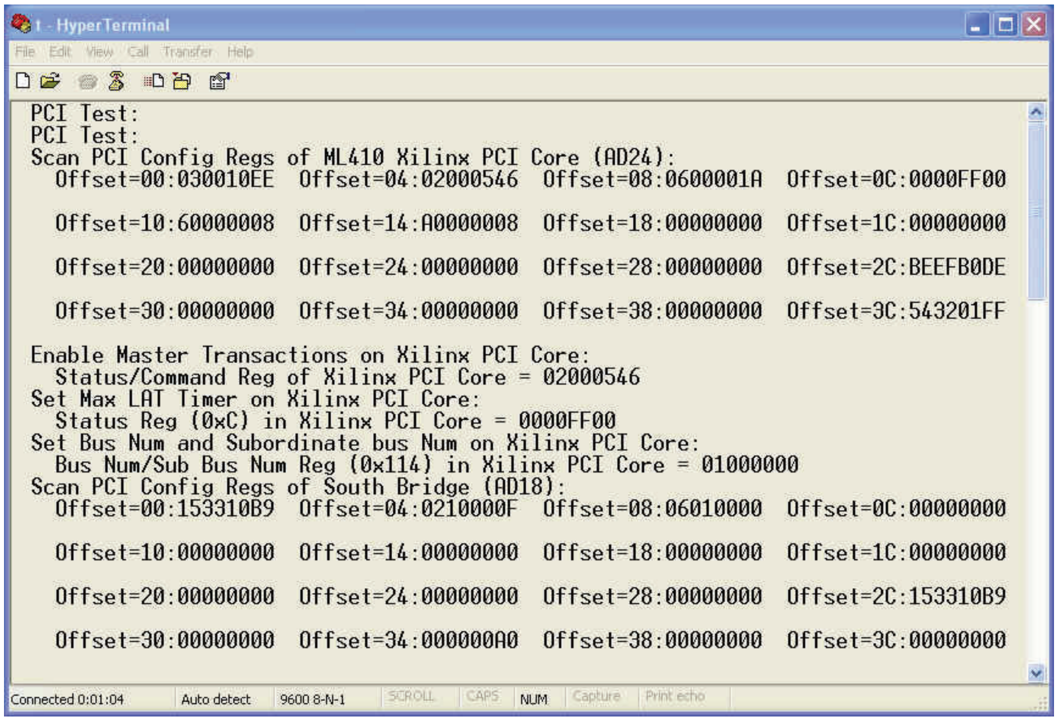
Port Settings

Bits per second:	9600
Data bits:	8
Parity:	None
Stop bits:	1
Flow control:	None

X945_14_010808

Figure 14: **HyperTerminal Parameters**

From XPS, start `XMD` and enter `connect ppc hw` and `rst` at the XMD prompt. Invoke GDB and select Run to start the application as shown in Figure 15. The hello_pci.c code, originally written for the OPB PCI on the ML310 Embedded Development Platform, runs without modifications on this reference system.



```
PCI Test:
PCI Test:
Scan PCI Config Regs of ML410 Xilinx PCI Core (AD24):
  Offset=00:030010EE  Offset=04:02000546  Offset=08:0600001A  Offset=0C:0000FF00

  Offset=10:60000008  Offset=14:A0000008  Offset=18:00000000  Offset=1C:00000000

  Offset=20:00000000  Offset=24:00000000  Offset=28:00000000  Offset=2C:BEEFB0DE

  Offset=30:00000000  Offset=34:00000000  Offset=38:00000000  Offset=3C:543201FF

Enable Master Transactions on Xilinx PCI Core:
  Status/Command Reg of Xilinx PCI Core = 02000546
Set Max LAT Timer on Xilinx PCI Core:
  Status Reg (0xC) in Xilinx PCI Core = 0000FF00
Set Bus Num and Subordinate bus Num on Xilinx PCI Core:
  Bus Num/Sub Bus Num Reg (0x114) in Xilinx PCI Core = 01000000
Scan PCI Config Regs of South Bridge (AD18):
  Offset=00:153310B9  Offset=04:0210000F  Offset=08:06010000  Offset=0C:00000000

  Offset=10:00000000  Offset=14:00000000  Offset=18:00000000  Offset=1C:00000000

  Offset=20:00000000  Offset=24:00000000  Offset=28:00000000  Offset=2C:153310B9

  Offset=30:00000000  Offset=34:000000A0  Offset=38:00000000  Offset=3C:00000000
```

Figure 15: **Running hello_pci**

Using the ChipScope Analyzer with PLB PCI

The Chipscope Analyzer is used to debug hardware problems. Debugging can be done at either the system or PLB PCI core level. To analyze PLB PCI internal signals, insert the Chipscope Analyzer cores into pci32_bridge_wrapper.ngc. To analyze signals involving multiple cores, insert the Chipscope Analyzer 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 Analyzer at the System Level

The following steps insert the Chipscope Analyzer cores into the system.

1. In XPS, select **Hardware** → **Generate Netlist.**
2. From the command prompt in the implementation directory, run

```
ngcbuild -i system.ngc system2.ngc
```

Invoke ChipScope Inserter. To specify the input file in the **Inserter Input Design Netlist** dialog box, browse to the system2.ngc file created in step 2. Define the Clock, Trigger, and Data signals in Inserter, and generate the ICON and ILA cores. The `chipscope/ml410_ppc_pllb_pci_scs.cdc` file provides an example of signals to monitor.

3. From ml410_ppc_plb_pci/implementation, copy the file displayed in the Inserter Output Design Netlist dialog box, usually system2.ngo, to system.ngc in the implementation directory.
4. In XPS, run **Hardware** → **Generate Bitstream**.

The `chipscope/ml410_ppc_plb_pci_scs.cpj` file is the Analyzer project.

Inserting Chipscope Analyzer at the PLB PCI Core Level

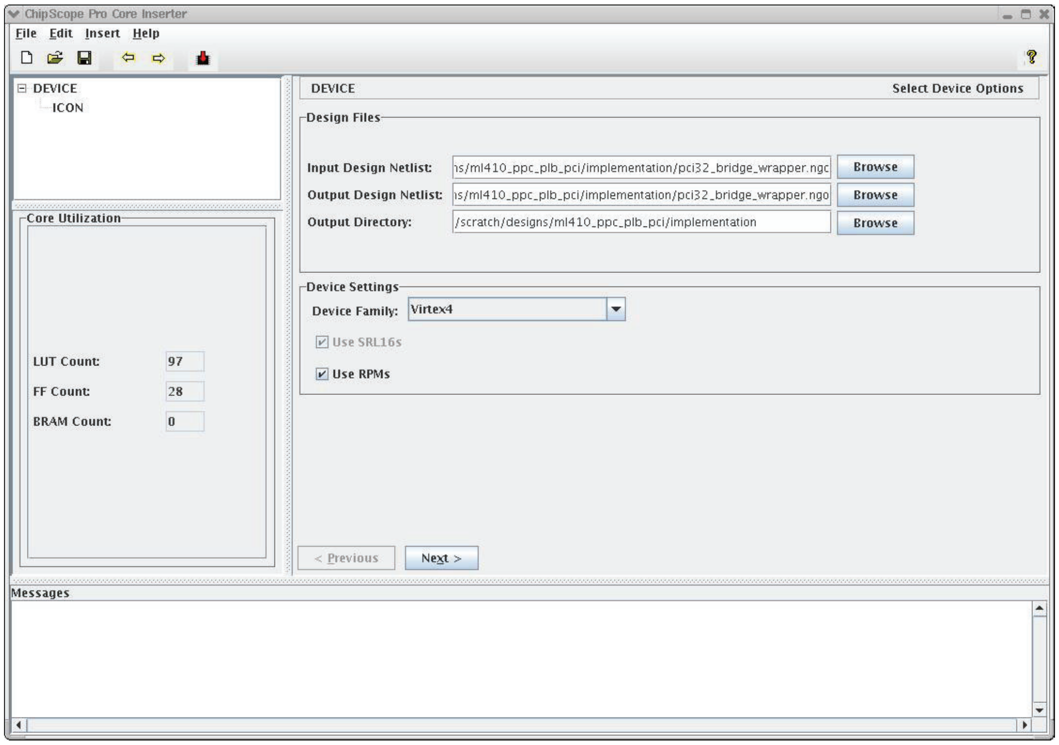
The `ml410_ppc_plb_pci/chipscope/plb_pci_ccs.cdc` file is used to insert an ILA core into the pci32_bridge_wrapper core. Do the following steps to insert a core and analyze PLB PCI problems with the Chipscope Analyzer.

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

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

3. From ChipScope Inserter, run **File Open** → **plb_pci_ccs.cdc**.

Figure 16 shows the ChipScope Inserter setup GUI after **File Open** → **plb_pci_ccs.cdc**.



Core Utilization

LUT Count: 97
FF Count: 28
BRAM Count: 0

Input Design Netlist: ıs/ml410_ppc_plb_pci/implementation/pci32_bridge_wrapper.ngc
Output Design Netlist: ıs/ml410_ppc_plb_pci/implementation/pci32_bridge_wrapper.ngo
Output Directory: /scratch/designs/ml410_ppc_plb_pci/implementation

Device Family: Virtex4
✓ Use SRL16s
✓ Use RPMs

Figure 16: **ChipScope Inserter Setup**

The PCI_Monitor signals are the PCI bus signals: AD, CBE, and the remaining PCI Bus signals. Table 7 defines the functionality of the PCI_Monitor signals. The Filter Pattern *PCI_Monitor* is used to locate the PCI 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

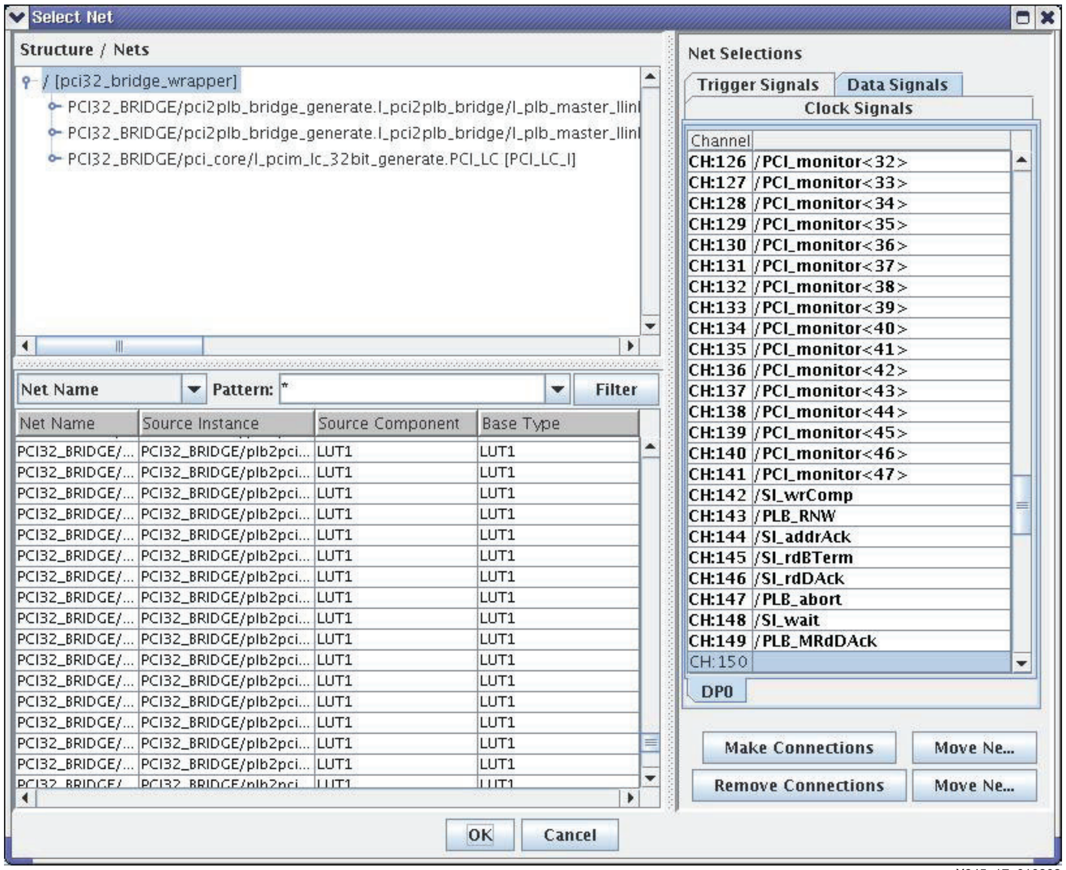
Table 7: **PCI Monitor Signals**

Bit Position	PCI Signal
8	SERR_N
9	Req_N_toArb
10	PAR
11	REQ_N
12:43	AD
44:47	CBE

5. The GUI for making net connections are shown in Figure 17. Click **Next** four times to move to the Modify Connections window. The nets from plb_pci_ccs.cdc are displayed.

In some analysis, additional nets are required. 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 as Sl_AddrAck. In the Net Selections area, select either Clock, Trigger, or Data Signals. Select the net and click **Make Connections**

Signal(s) with an invalid connection(s) are displayed in red. Correct Clock, Data, or Trigger signals displayed in red.



/ [pci32_bridge_wrapper]
- PCI32_BRIDGE/pci2plb_bridge_generate.I_pci2plb_bridge/I_plb_master_llin
- PCI32_BRIDGE/pci2plb_bridge_generate.I_pci2plb_bridge/I_plb_master_llin
- PCI32_BRIDGE/pci_core/I_pcim_lc_32bit_generate.PCI_LC [PCI_LC_I]

Pattern: *

Net Name	Source Instance	Source Component	Base Type
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1
PCI32_BRIDGE/...	PCI32_BRIDGE/plb2pci...	LUT1	LUT1

Channel	
CH:126	/PCI_monitor<32>
CH:127	/PCI_monitor<33>
CH:128	/PCI_monitor<34>
CH:129	/PCI_monitor<35>
CH:130	/PCI_monitor<36>
CH:131	/PCI_monitor<37>
CH:132	/PCI_monitor<38>
CH:133	/PCI_monitor<39>
CH:134	/PCI_monitor<40>
CH:135	/PCI_monitor<41>
CH:136	/PCI_monitor<42>
CH:137	/PCI_monitor<43>
CH:138	/PCI_monitor<44>
CH:139	/PCI_monitor<45>
CH:140	/PCI_monitor<46>
CH:141	/PCI_monitor<47>
CH:142	/Sl_wrComp
CH:143	/PLB_RNW
CH:144	/Sl_addrAck
CH:145	/Sl_rdBTerm
CH:146	/Sl_rdDAck
CH:147	/PLB_abort
CH:148	/Sl_wait
CH:149	/PLB_MRdDAck
CH:150	

Figure 17: **Making Net Connections in ChipScope Inserter**

6. Click **Insert** to insert the core into pci32_bridge_wrapper.ngo. In the `ml410_ppc_plb_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

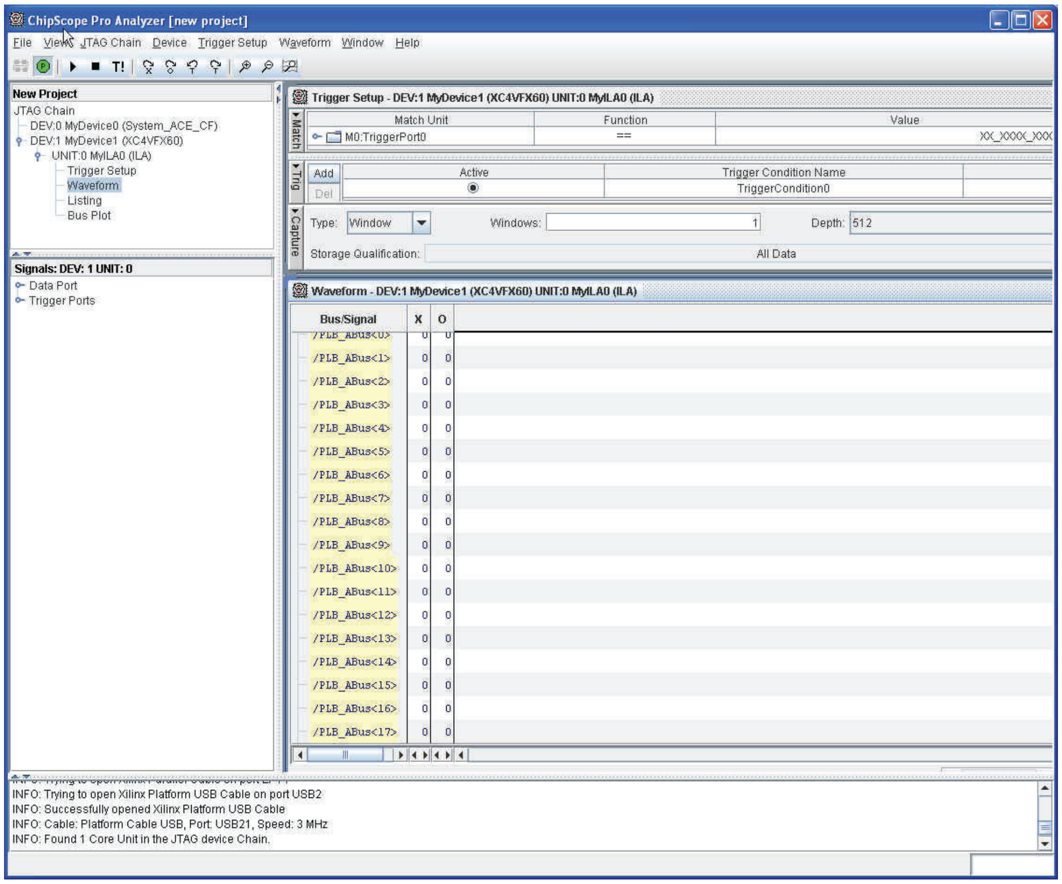
Start → **Programs** → **ChipScope Pro** → **ChipScope Pro Analyzer**

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 familiar signal names specified in ChipScope Inserter, select **File** → **Import** and enter **`ml410_ppc_plb_pci_ccs.cdc`** in the dialog box.

The Chipscope Analyzer waveform viewer is more readable when buses rather than discrete signals are displayed. Select the **32 PLB_ABus<*>** signals, click the right mouse button, and select **Add to Bus** → **New Bus**. With **PLB_ABus<0:31>** in the waveform viewer, select and delete the 32 discrete **PLB_ABus<*>** signals. Repeat this for the PLB data buses. Make PCI Bus signals by creating a new bus for PCI_Monitor(44:47). Rename PCI_Monitor(44:47) PCI_CBE. Create a bus for PCI_Monitor(12:43), renaming it PCI AD.The signals are displayed as buses as shown in Figure 18.

Note: The Reverse Bus Order operation option is useful for analyzing values of bus signals in Analyzer.


ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Trigger Setup Waveform Window Help
New Project
JTAG Chain
DEV:0 MyDevice0 (System_ACE_CF)
DEV:1 MyDevice1 (XC4VFX60)
UNIT:0 MyILA0 (ILA)
Trigger Setup
Waveform
Listing
Bus Plot
Signals: DEV: 1 UNIT: 0
Data Port
Trigger Ports
Trigger Setup - DEV:1 MyDevice1 (XC4VFX60) UNIT:0 MyILA0 (ILA)
Match
Trig
Capture
Add
Del
Waveform - DEV:1 MyDevice1 (XC4VFX60) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value
M0:TriggerPort0	==	XX_XXXX_XXX

Active	Trigger Condition Name
●	TriggerCondition0

Type: Window Windows: 1 Depth: 512
Storage Qualification: All Data

Bus/Signal	X	O
/PLB_ABus<0>	0	0
/PLB_ABus<1>	0	0
/PLB_ABus<2>	0	0
/PLB_ABus<3>	0	0
/PLB_ABus<4>	0	0
/PLB_ABus<5>	0	0
/PLB_ABus<6>	0	0
/PLB_ABus<7>	0	0
/PLB_ABus<8>	0	0
/PLB_ABus<9>	0	0
/PLB_ABus<10>	0	0
/PLB_ABus<11>	0	0
/PLB_ABus<12>	0	0
/PLB_ABus<13>	0	0
/PLB_ABus<14>	0	0
/PLB_ABus<15>	0	0
/PLB_ABus<16>	0	0
/PLB_ABus<17>	0	0

```
INFO: Trying to open Xilinx Platform USB Cable on port USB2
INFO: Successfully opened Xilinx Platform USB Cable
INFO: Cable: Platform Cable USB, Port: USB21, Speed: 3 MHz
INFO: Found 1 Core Unit in the JTAG device Chain.
```

X945_18_010808

Figure 18: **Creating Buses in ChipScope Analyzer**

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 ML410 PLB PCI, trigger on an PLB address of C_BASEADDR + 0x10C. If debugging a problem configuring from the PCI side, trigger on the PCI_CBE for a configuration write on CBE. Simpler triggers are PCI_FRAME_N and PA_Valid.

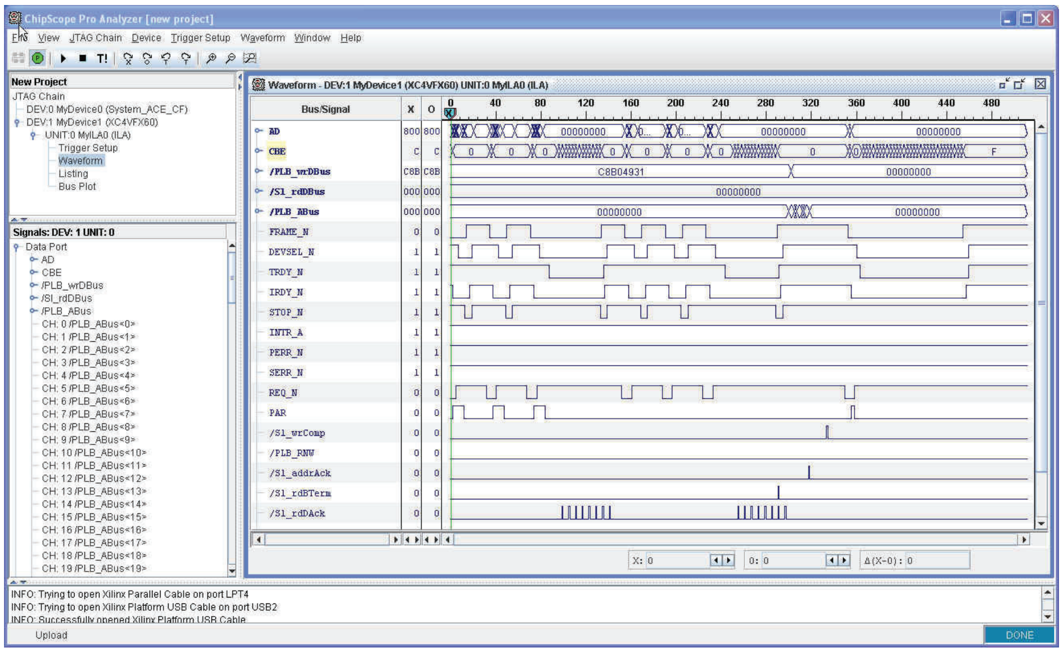
Arm the trigger by selecting **Trigger Setup** → **Arm**, or clicking on the **Arm** icon.

12. Run **XMD** or **GDB** to activate trigger patterns which cause the Chipscope Analyzer to display waveform output. For example, set the trigger to PA_Valid and run

```
xmd -tcl xmd_commands/410_555.tcl
```

at the command prompt.

13. The Chipscope Analyzer results are analyzed in the waveform window, as shown in Figure 19. This figure shows the bus signals generated in Step 10. 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 ModelSim waveform log file (wlf) file for viewing in the ModelSim waveform viewer. The vcd file can be opened in the Cadence Design System, Inc. Simvision design tool by selecting **File** → **Open Database**.



```
INFO: Trying to open Xilinx Parallel Cable on port LPT4
INFO: Trying to open Xilinx Platform USB Cable on port USB2
INFO: Successfully opened Xilinx Platform USB Cable
```

X945_19_010808

Figure 19: **ChipScope Analyzer Results**

After running the Chipscope Analyzer, it is sometimes necessary to revise the Trigger or Data nets used in a debug operation in Inserter. If the Inserter and Analyzer projects are saved, they can be re-opened, so that only edits need be made.

The `ml410_ppc_plb_pci_ccs.cpj` provides an Analyzer project.

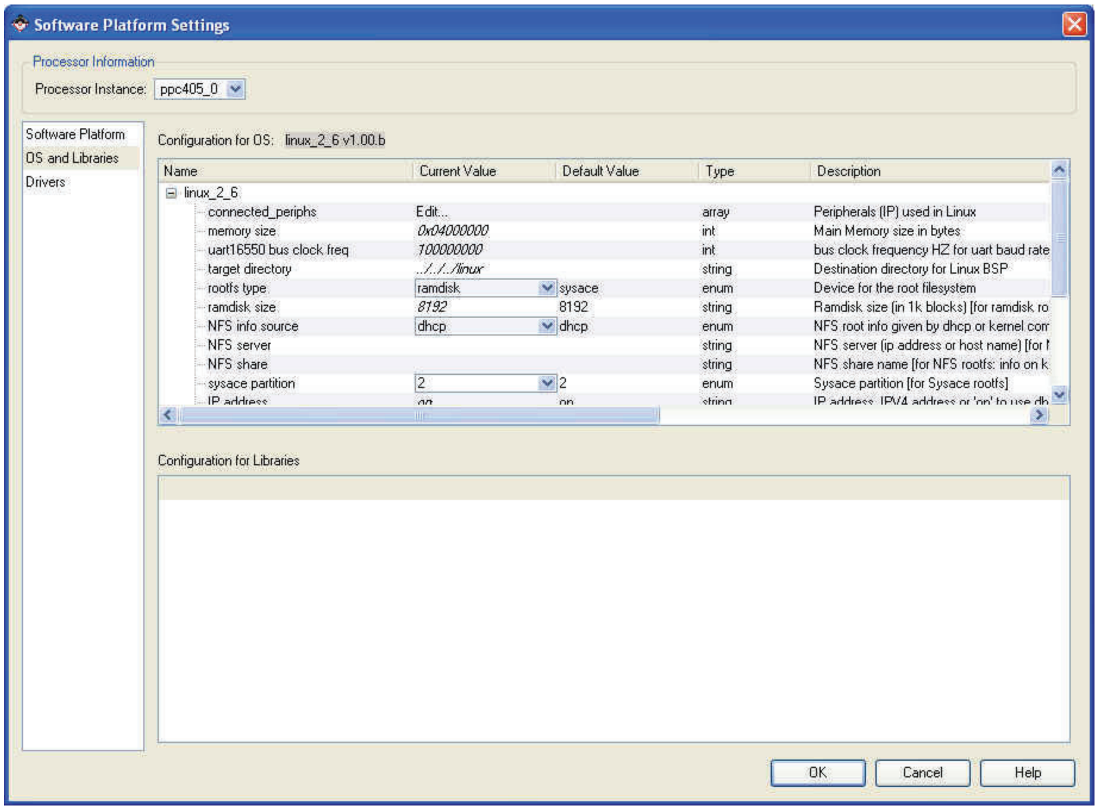
Linux Kernel

XAPP 765 provides an introduction for new users of Monta Vista Linux. 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. Create and change to the `ml410_ppc_plb_pci/linux` directory.
3. Run

```
tar cf - -C /opt3/montavista/pro/devkit/lsp/xilinx-ml40x-
ppc_405/linux-2.6.10_mvl401/ . | tar xf -
```

4. To generate the Linux LSP in XPS, enter **Software** → **Software Platform Settings.** Select **Kernel and Operating Systems**, then select **OS: linux_2_6 and Version: 1.00.b**.
5. Under **OS and Libraries**, set the entries as shown in Figure 20.



Processor Instance: ppc405_0

Configuration for OS: linux_2_6 v1.00.b

Name	Current Value	Default Value	Type	Description
linux_2_6				
connected_periphs	Edit...		array	Peripherals (IP) used in Linux
memory size	0x04000000		int	Main Memory size in bytes
uart16550 bus clock freq	100000000		int	bus clock frequency HZ for uart baud rate
target directory	../../../linux		string	Destination directory for Linux BSP
rootfs type	ramdisk	sysace	enum	Device for the root filesystem
ramdisk size	8192	8192	string	Ramdisk size (in 1k blocks) [for ramdisk ro
NFS info source	dhcp	dhcp	enum	NFS root info given by dhcp or kernel com
NFS server			string	NFS server (ip address or host name) [for
NFS share			string	NFS share name [for NFS rootfs: info on k
sysace partition	2	2	enum	Sysace partition [for Sysace rootfs]
IP address	on	on	string	IP address. IPV4 address or 'on' to use dh

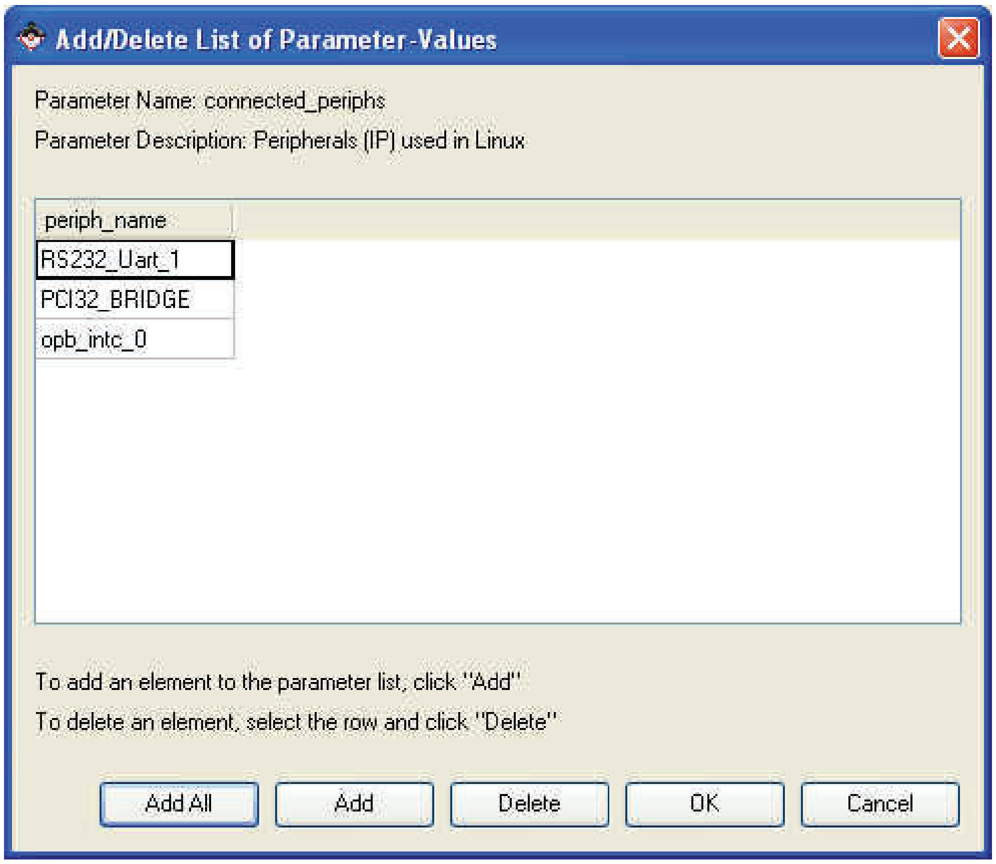
Configuration for Libraries

X945_20_010808

Figure 20: **Software Platform Setting Setup**

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

6. Click **Connect_Periphs** and add the OPB_INTC, PLB_PCI, and OPB16550 peripherals, using the instance names shown in Figure 21.



Parameter Name: connected_periphs

Parameter Description: Peripherals (IP) used in Linux

periph_name
RS232_Uart_1
PCI32_BRIDGE
opb_intc_0

To add an element to the parameter list, click "Add"

To delete an element, select the row and click "Delete"

X945_21_010808

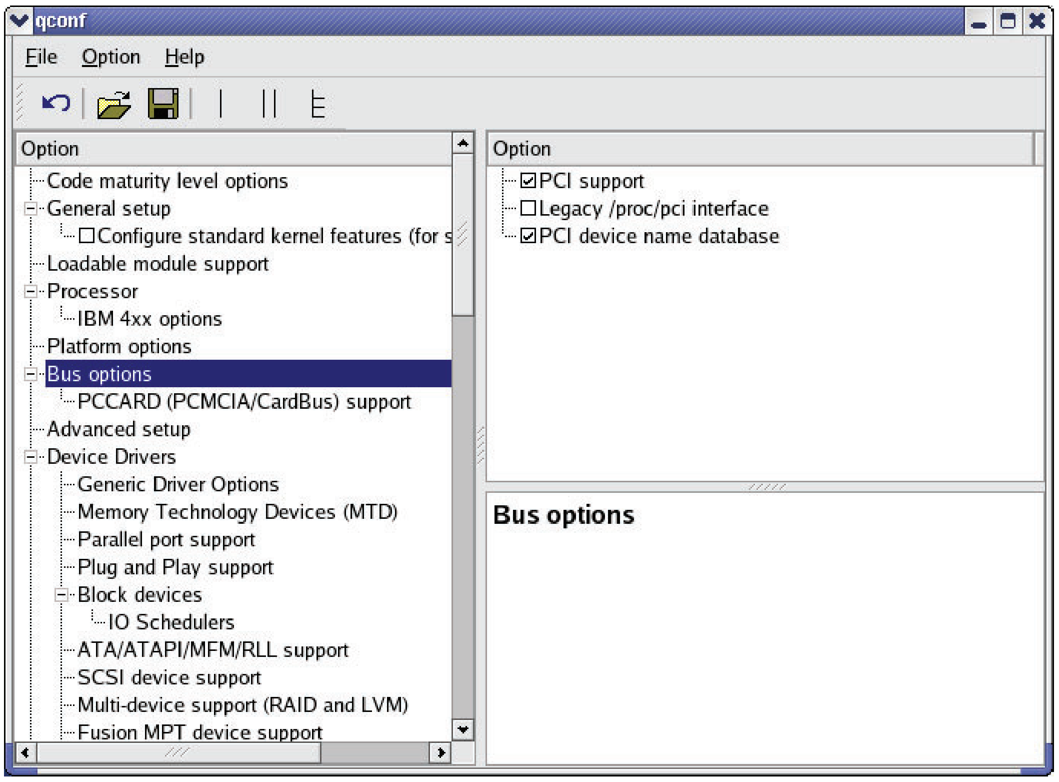
Figure 21: **Connected Peripherals**

Click **OK**.

7. Select **Software** → **Generate Libraries and BSPs** to generate the LSP in `ml410_ppc_plb_pci/linux`.

8. The `ml410_ppc_plb_pci/linux/.config` is used to define the contents of the Linux kernel. Run `make oldconfig`.

An alternative is to enter **`make xconfig`** as shown in Figure 22 and generate a new `.config` using the following options.



- Code maturity level options
- General setup
  - ☐ Configure standard kernel features (for s
- Loadable module support
- Processor
  - IBM 4xx options
- Platform options
- Bus options
  - PCCARD (PCMCIA/CardBus) support
- Advanced setup
- Device Drivers
  - Generic Driver Options
  - Memory Technology Devices (MTD)
  - Parallel port support
  - Plug and Play support
  - Block devices
    - IO Schedulers
  - ATA/ATAPI/MFM/RLL support
  - SCSI device support
  - Multi-device support (RAID and LVM)
  - Fusion MPT device support

- ☑ PCI support
- ☐ Legacy /proc/pci interface
- ☑ PCI device name database

X945_22_010808

Figure 22: **make xconfig**

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 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.

9. Run **make zImage.initrd**. Verify that the `zImage.initrd.elf` file is in the `ml410_ppc_plb_pci/linux/arch/ppc/boot/images` directory.
10. Use **Impact** to download `implementation/download.bit` to the XC4VFX60 device. Either select **Device Configuration** → **Download Bitstream** from XPS or run the following command from the command prompt:

```
impact -batch etc/download.cmd
```

11. Invoke XMD. From the `ml410_ppc_plb_pci/linux` directory, enter the following commands in the XMD window:

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

12. The HyperTerminal window displays the Linux boot process. Login as **root**. Enter **cd /** and **ls -l** to view the contents of the mounted Linux partition.
13. Enter **./lspci -vv** to view the PCI devices. For each line of output, the first two digits represent the PCI bus number, followed by the device number and function number.
14. 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_plb_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, and view Linux booting in the HyperTerminal window.

Reference Design Matrix

The reference design matrix is shown in Table 8.

Table 8: **Reference Design Matrix**

General	
Developer Name	Xilinx
Target devices (stepping level, ES, production, speed grades)	Virtex-5 LX50T
Source code provided	No
Source code format	VHDL
Design uses code/IP from an existing reference design/application note, 3rd party, or CORE Generator software	No
Simulation	
Functional simulation performed	No
Timing simulation performed	No

Table 8: **Reference Design Matrix**

Testbench used for functional simulations provided	No
Testbench format	N/A
Simulator software used/version (i.e., ISE software, Mentor, Cadence, other)	N/A
SPICE/IBIS simulations	No
Implementation	
Synthesis software	XST
Implementation software tools used/versions	ISE9.2i SP3
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ML410/ML555

References

1. DS207 *PCI 64/32 Interface v3.0 Data Sheet*
2. UG159 *LogiCORE IP Initiator/Target v3.1 for PCI*
3. UG262 *LogiCORE IP Initiator/Target v4.5 for PCI*
4. UG085 *ML410 Embedded Development Platform User Guide*
5. UG044 *ChipScope ILA Tools Tutorial*
6. UG201 *Virtex-5 ML555 Development Kit for PCI/PCI Express Designs User Guide*
7. UG241 *OPB PCI v1.02a User Manual*
8. XAPP765 *Getting Started with EDK and MontaVista Linux*
9. XAPP998 *PCI Bus Performance Measurements using the Vmetro Bus Analyzer*
10. XAPP999 *Reference System: PLBv46 PCI Using the ML555 Embedded Development Platform*
11. XAPP1001 *Reference System: PLBv46 PCI Using the Avnet Spartan-3 Evaluation Board*

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/03/07	1.0	Initial Xilinx release.
2/8/08	1.1	Updated with references to PLBv46.

Notice of Disclaimer