



XAPP1022 (v2.0) November 20, 2009

Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores

Author: John Ayer Jr.

Summary

This application note discusses using the provided Memory Endpoint Test (MET) demonstration driver to exercise the Programmed Input/Output (PIO) design that is delivered with all Xilinx solutions for PCI Express®. Instructions for installing this driver on a typical Windows XP operating system are provided, along with how to access the I/O and memory space of the design.

Important Notice

The MET driver is provided *as is* with no implied warranty or support. This driver is not guaranteed to work on all systems. While there are no known issues with using the driver application, no technical support will be provided for problems that might arise. Source code for the MET driver is not available.

Overview

Xilinx offers cores for PCI Express to be used in endpoint applications. The appropriate core choice is based on the target device and design requirements. [Table 1](#) shows a summary of the cores and supported devices.

Table 1: Cores for PCI Express

Core Name	Device(s) Supported	Comments
Integrated Block for PCI Express	Virtex®-6	Utilizes the Virtex-6 built in block for PCI Express.
Endpoint Block Plus Wrapper for PCI Express	Virtex-5	Utilizes the Virtex-5 Built-in Endpoint Block for PCI Express
Endpoint for PCI Express	Virtex-5 and Virtex-4	Soft-IP Implementation
Integrated Block for PCI Express	Spartan®-6	Utilizes the Spartan-6 built in block for PCI Express.
Endpoint PIPE for PCI Express	Spartan-3, Spartan-3A, and Spartan-3E	Soft-IP implementation utilizing an external physical layer from NXP Semiconductors

More information about the current versions of these cores is available in the product data sheets located in the Xilinx online [Documentation Center](#). The specific product pages are linked by the Core Name column in [Table 1](#). Visit the [Xilinx solutions for PCI Express page](#) for more information about the cores for PCI Express.

The cores for PCI Express are delivered by the Xilinx CORE Generator™ software. This software allows users to customize various parameters of the core such as Device and Vendor ID, BAR requirements, and power management settings. Detailed instructions for generating the core using the CORE Generator software can be found in the Getting Started Guide or User Guide for the core. This document is delivered with the core, but can also be downloaded from the Xilinx online [Documentation Center](#).

Setting Up the PIO Example Design

By default, the endpoint core includes a working example called the PIO design that can be downloaded to an add-in card and inserted into any PCI Express system. The Getting Started Guide and User Guide contain detailed information about generating a core. This application note has less detailed instructions to allow the user to generate a core and download it to a board.

To use the PIO design with the MET driver, changes need to be made to some of the CORE Generator software customization parameters. See steps 6 and 7 under “[Generating the Core](#)”.

Generating the Core

1. Install the latest version of the ISE Design Suite tools. Updates can be found at: <http://www.xilinx.com/tools/designtools.htm>
2. Start the CORE Generator software and create a new project.
3. Select the appropriate part.
4. In the taxonomy tree, select **Standard Bus Interface > PCI Express**.
5. Select the appropriate core for PCI Express and click **Customize**.
6. Change the BAR settings to implement a single Memory BAR. The recommended size is 8 Kilobytes, but this value is user configurable. An example is shown in [Figure 1](#).
7. Click **Next** to continue.

Note: If the Virtex-5 Block Plus Wrapper for PCI Express is selected, the panel to select the class code will be first followed by the panel to customize the BARs.

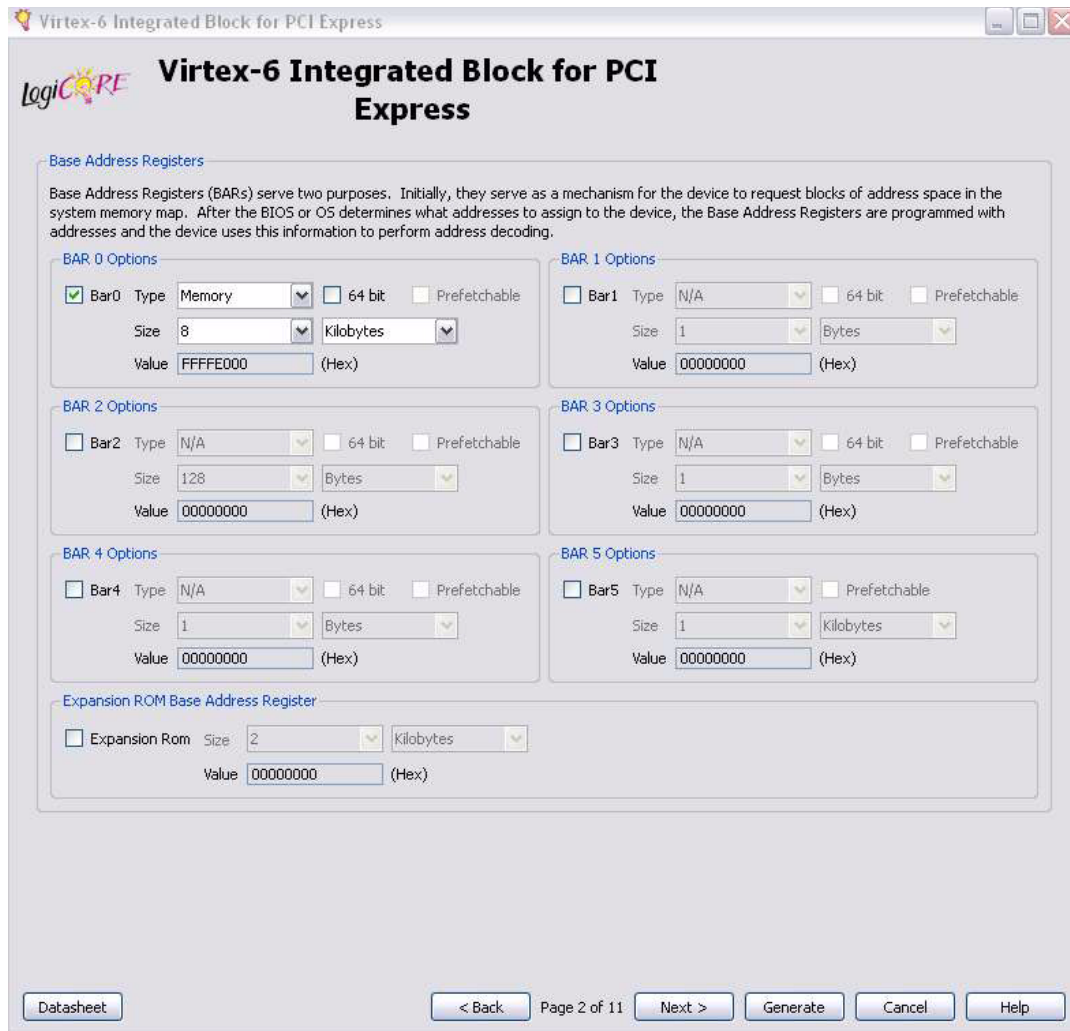
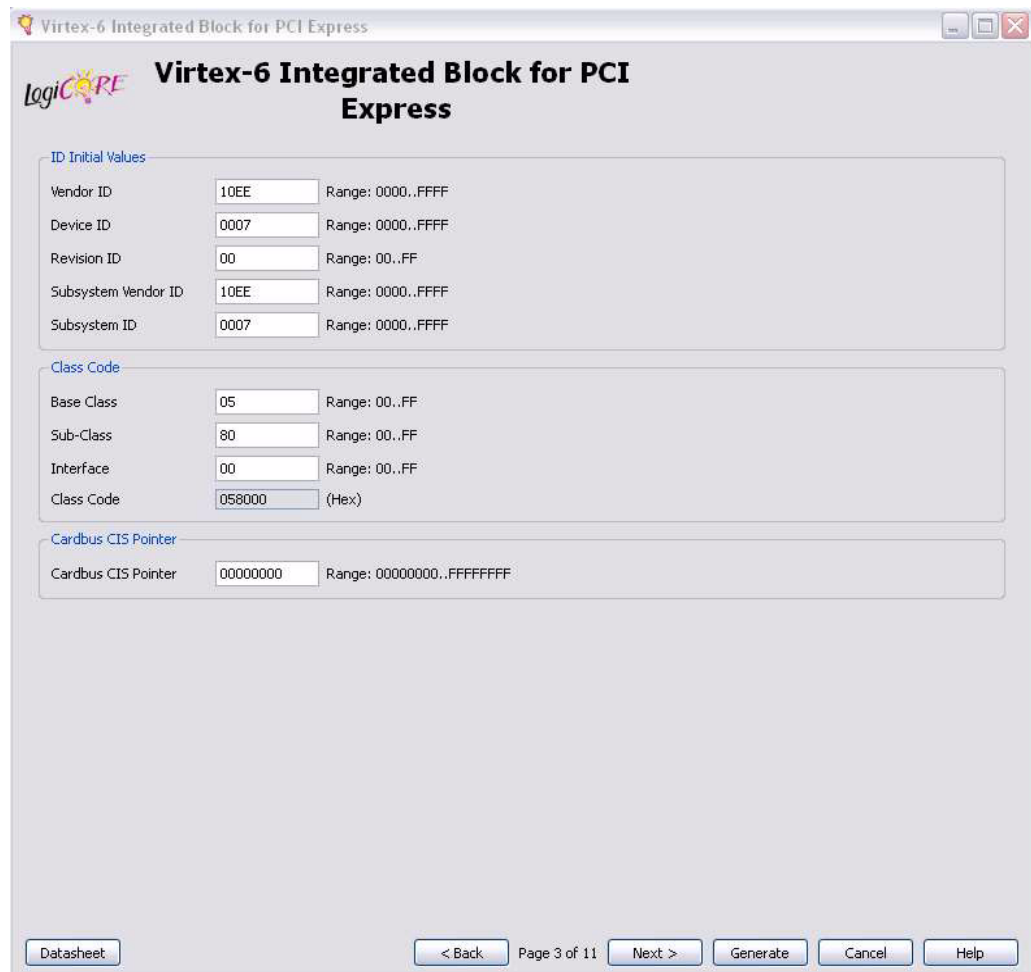


Figure 1: Class Code Settings for PIO Design

8. Change the Sub-Class code to 80 to indicate an “Other Memory Controller” to the system. An example is shown in [Figure 2](#).



Virtex-6 Integrated Block for PCI Express

ID Initial Values

Vendor ID	10EE	Range: 0000..FFFF
Device ID	0007	Range: 0000..FFFF
Revision ID	00	Range: 00..FF
Subsystem Vendor ID	10EE	Range: 0000..FFFF
Subsystem ID	0007	Range: 0000..FFFF

Class Code

Base Class	05	Range: 00..FF
Sub-Class	80	Range: 00..FF
Interface	00	Range: 00..FF
Class Code	058000	(Hex)

Cardbus CIS Pointer

Cardbus CIS Pointer	00000000	Range: 00000000..FFFFFFFF
---------------------	----------	---------------------------

Buttons: Datasheet, < Back, Page 3 of 11, Next >, Generate, Cancel, Help

Figure 2: Memory BAR Settings for PIO Design

9. Accept the default settings for all other fields. Click **Finish**.

Implementing the Core

1. Navigate to the output directory and browse to the implement folder.
2. Double-click or source the implementation script provided.

The PIO example design is synthesized and implemented. A results directory is created containing a routed.bit file. This file is to be downloaded to the board. Note that the implementation script might need to be modified to point to the correct UCF file for the board used.

Programming the Board

For a system to recognize a PCI Express add-in card, the card must be present during bus enumeration. Enumeration is performed by the BIOS during the boot process. For this reason, the FPGA must be programmed in one of two ways:

- Through an on-board PROM, so that when the system is powered on, the FPGA is programmed in time to be enumerated by the BIOS.
- Through the JTAG interface after the OS has started. However, for the card to be recognized, a “warm reset” must be performed. In Windows, this equates to performing a

Restart. Note that sometimes re-programming the FPGA after the OS has started can cause the system to hang.

Exploring the PIO Design

The PIO design implements an 8192 byte target space in FPGA block RAM, behind the PCI Express Endpoint core. This 32-bit target space is accessible through single DWORD I/O Read, I/O Write, Memory Read 64, Memory Write 64, Memory Read 32, and Memory Write 32 TLPs.

The PIO design generates a completion with 1 DWORD of payload in response to a valid Memory Read 32 TLP, Memory Read 64 TLP, or IO Read TLP request presented to it by the PCI Express Endpoint Core. In addition, the PIO design returns a completion without data with successful status for I/O Write TLP request.

The PIO design processes a Memory or I/O Write TLP with 1 DWORD payload by updating the payload into the target address in the FPGA block RAM space.

By default, the PIO design supports four discrete target spaces, each consisting of a 2 kB block of memory represented by a separate Base Address Register (BAR). Using the default parameters produces a core configured to work with the PIO design defined in this section, and consists of the following:

- One IO Space BAR
- One 64-bit Addressable Memory Space BAR
- One 32-bit Addressable Memory Space BAR
- One Expansion ROM BAR

The MET driver application described in this document is mainly used to interface with memory space. It is recommended that the user modify the PIO design to use the entire 8 kB of block RAM space for 32-bit addressable memory accesses. This modification can be done in many ways, but the simplest method is to modify the `PIO_128_RX_ENGINE.v[hd]`, `PIO_64_RX_ENGINE.v[hd]`, or `PIO_32_RX_ENGINE.v[hd]` file by searching and modifying assignments to the "req_addr_o" and "wr_addr_o" buses.

Depending on the file, there will be assignments made to these buses in various states of the RX Engine state machine. Each time an assignment is made, it will be a concatenation of the following terms: `region_select[1:0]`, `trn_rd[##:##]`, and "00" if its "req_addr_o" bus. In each case, remove the `region_select[1:0]` and increase the upper index of the `trn_rd` bus by 2. For example:

- Change:

```
req_addr_o <= #'TCQ {region_select[1:0],trn_rd[42:34], 2'b00};
```


To:

```
req_addr_o <= #'TCQ {trn_rd[44:34], 2'b00};
```
- Change:

```
wr_addr_o <= #'TCQ {region_select[1:0], trn_rd[42:34]};
```


To:

```
wr_addr_o <= #'TCQ {trn_rd[44:34]};
```

Carry out this modification each time an assignment to `req_addr_o` and `wr_addr_o` is found. Note that the MET driver will still work without this modification, but it will only be able to access 2 KB of space in the device.

See the applicable core User Guide for more information about the PIO design.

Users can write and read the address space using any available software that will recognize the PCI Express design and access its BAR space. One such tool is the PCITree shareware tool available from <http://www.pcitree.de> for Windows XP operating systems.

Using the MET Driver with Windows XP

MET Driver and Application

The MET driver and application are available in `xapp1022.zip`. Unzip this file to any directory and follow the instructions in “[Installing the Driver](#),” page 6. Currently, the driver is set to recognize cards with a Vendor ID of 10EEh and a Device ID of 0007h. To change this setting, open `xilinx_pcie_block.inf` and modify the line:

```
Xilinx Endpoint for PCI Express = XILINXPCIe,PCI\VEN_10EE&DEV_0007
```

For example, to use a Vendor ID of 1234 and a Device ID of 0101, change this line to read:

```
Xilinx Endpoint for PCI Express = XILINXPCIe,PCI\VEN_1234&DEV_0101
```

Also, more than one Device and Vendor ID can be recognized by adding multiple lines. For example:

```
Xilinx Endpoint for PCI Express = XILINXPCIe,PCI\VEN_10EE&DEV_0007
Xilinx Endpoint for PCI Express = XILINXPCIe,PCI\VEN_1234&DEV_0101
```

Note that if `xilinx_pcie_block.inf` is modified, the driver must be re-installed.

Installing the Driver

When the card with the PIO design for PCI Express is first installed, Windows attempts to locate a device driver for the card. Using the PIO example design with the CORE Generator defaults, the Vendor ID is “0x10EE” and the device ID is “0x0007”. Windows searches the driver database for the card found. Initially, a driver will not be found, and the Found New Hardware Wizard is launched.

Installing the Driver

To install the driver, follow these steps:

1. Select **No, not at this time**, and click **Next**.



Figure 3: Welcome to the Found New Hardware Wizard

2. Select **Install from a list or specific location (Advanced)** and click **Next**.

This is because the driver is being provided as a ZIP file instead of on a CD.

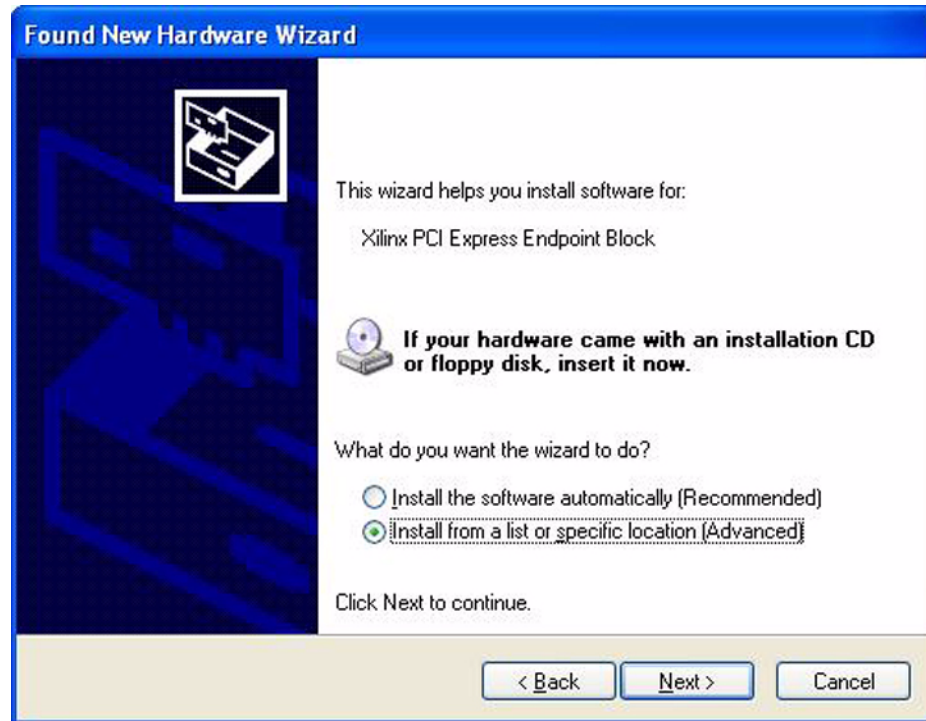


Figure 4: Install from a List or Specific Location (Advanced)

3. Select **Don't search, I will choose the driver to install** and click **Next**.

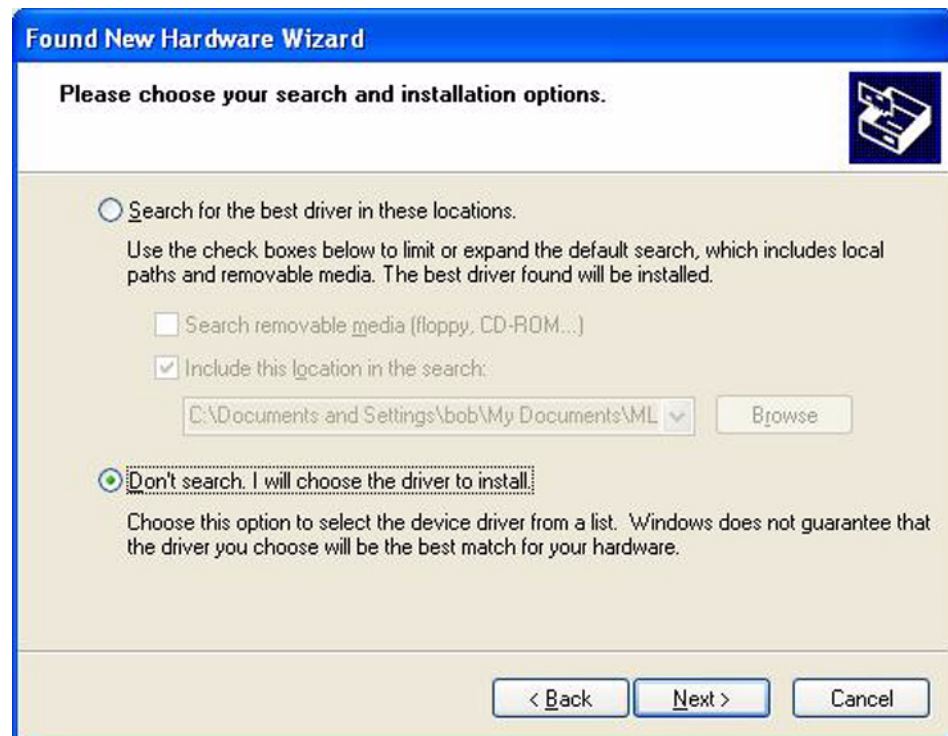


Figure 5: Change Search and Installation Options

4. Click **Have Disk**, and then click **Next**.



Figure 6: Select the Device Driver

5. Browse to the location of the driver (Figure 7). The driver is provided as a ZIP file with this document. Unzip it to any location on the machine and browse to the filename "xilinx_pcie_block.inf." Select this file, click **Open**, and then click **OK**.

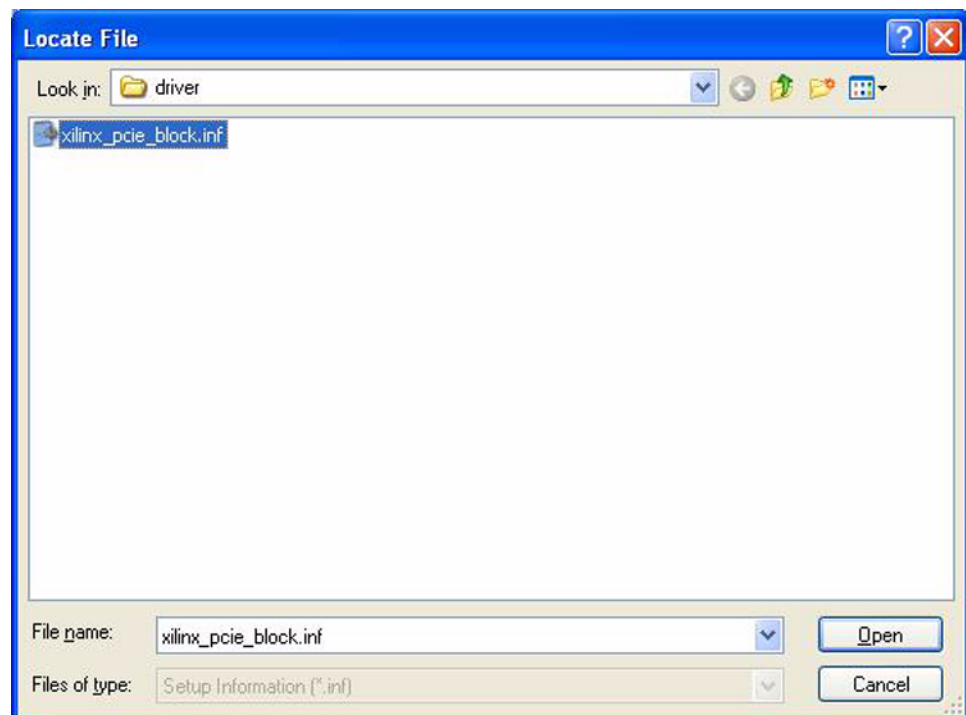


Figure 7: Locate the Driver File

6. After clicking OK to choose the “.inf” file, click **Next** to install the driver (Figure 8).

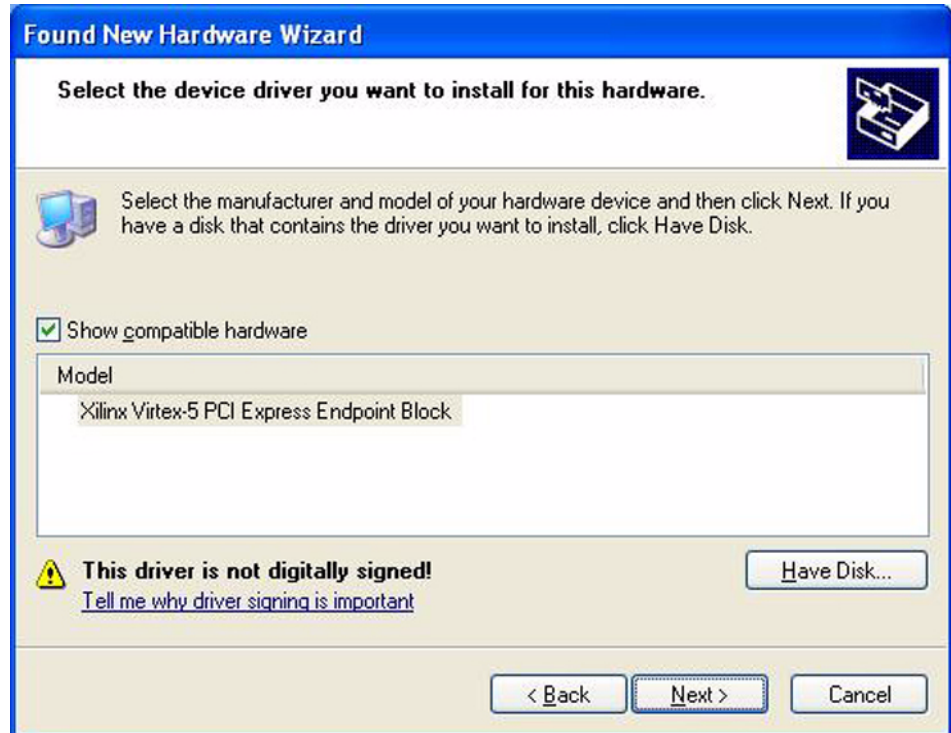


Figure 8: Install the Driver

7. Progress of the installation is displayed.

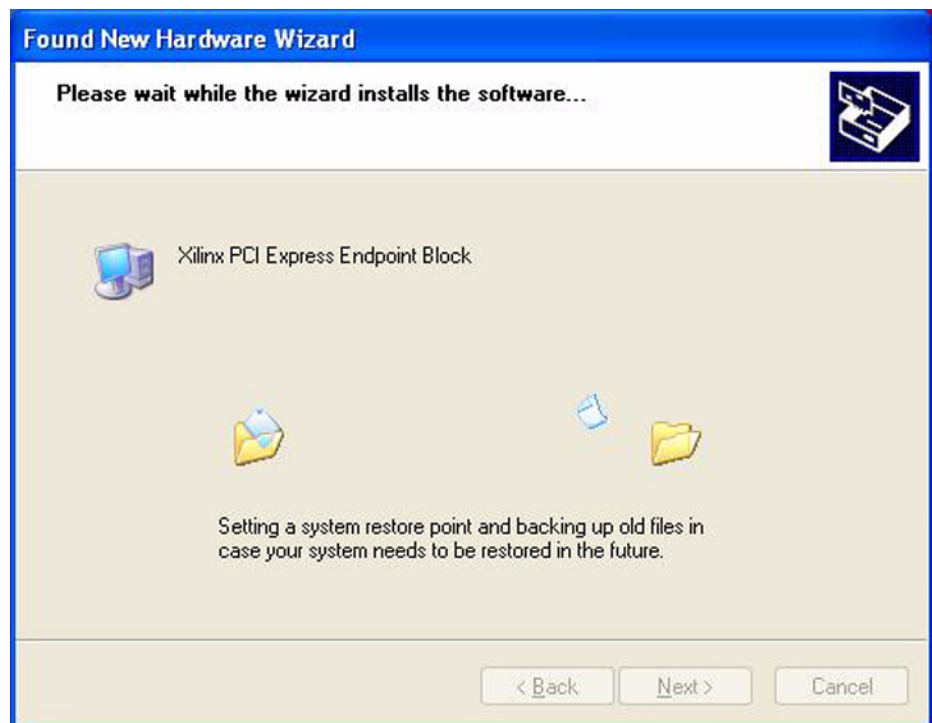


Figure 9: Wizard Installs the Software

8. After successful installation, you are instructed to reboot. Click **Finish** to exit the Wizard.

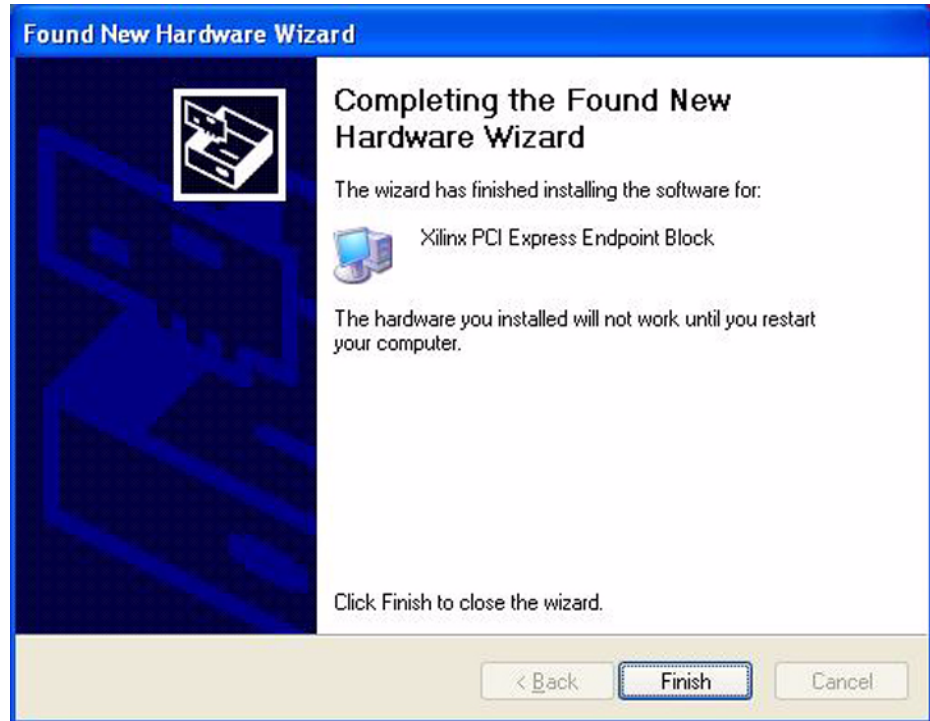


Figure 10: Hardware Update Complete

9. After the reboot, the device will appear in the Device Manager under “System Devices” (Figure 11).

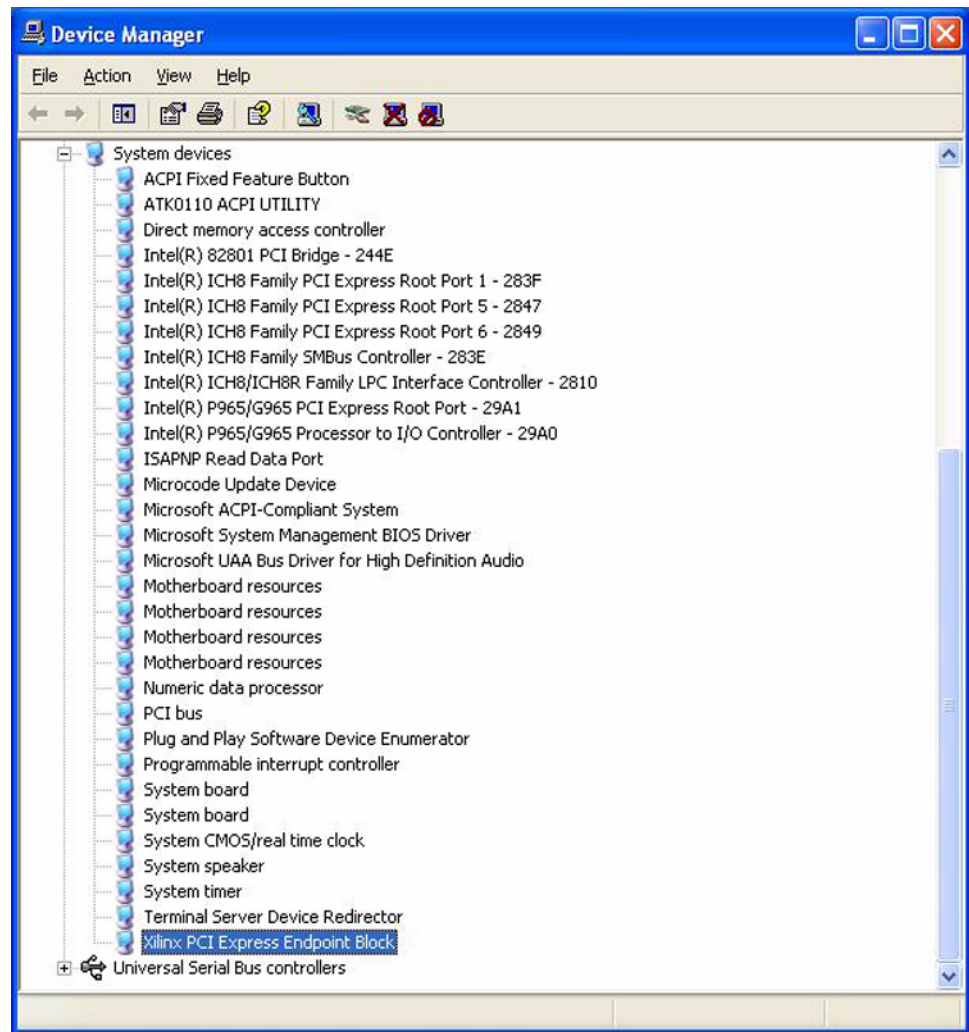


Figure 11: Device Manager—System Devices

Using MET in Command Line Mode

The MET application can be executed with the arguments shown in Table 2.

Table 2: MET Application Arguments

Argument	Definition	Example
--log <LogFile>	[OPTIONAL] Logs all test results and non-prompt output to the given log file. For safety, it will not overwrite an existing log file.	MET --log log1.txt
--script <ScriptFile>	[OPTIONAL] Accepts and executes interactive “monitor” commands from the given file.	MET --script cmds.txt
<TestSpec>	[OPTIONAL] Executes the tests specified in the given file, in the “.ini” format documented below.	MET Memtest1.ini

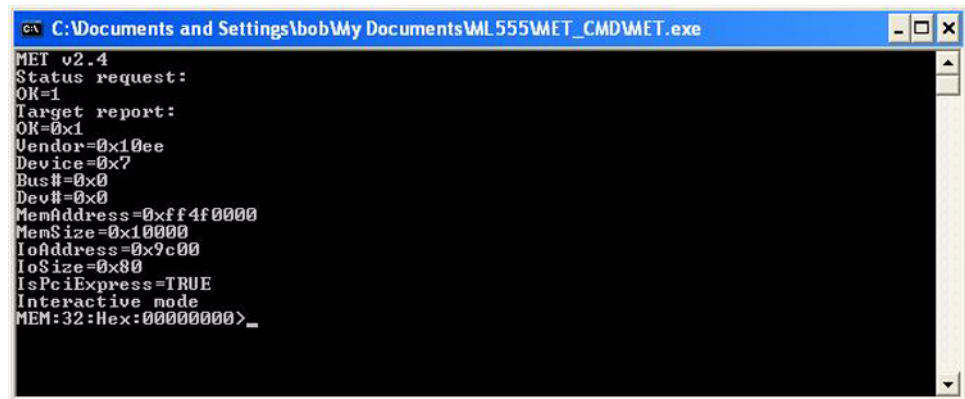
The ScriptFile and TestSpec file options are mutually exclusive, meaning only one can be used at a time. However, a test specification can be combined with the scripting method by calling it through the interactive test specification command. Following are usage examples:

- C:\>MET
Runs the MET application in interactive mode.
- C:\>MET --log log1.txt
Runs the MET application in interactive mode and logs the results.
- C:\>MET --log log1.txt --script my_script.txt
Runs the MET application with script file inputs and logs the results.
- C:\>MET --log log1.txt memtest.ini
Runs the MET with the test file specified and logs the results.

Running in Interactive Mode

The driver is run by either double-clicking the “MET.exe” file or typing **MET.exe** at the command prompt.

When launched, the driver displays configuration space information, as shown in [Figure 12](#).



```

C:\Documents and Settings\Bob\My Documents\ML555\MET_CMD\MET.exe
MET v2.4
Status request:
OK=1
Target report:
OK=0x1
Vendor=0x10ee
Device=0x?
Bus# =0x0
Dev# =0x0
MemAddress=0xff4f0000
MemSize=0x10000
IoAddress=0x9c00
IoSize=0x80
IsPciExpress=TRUE
Interactive mode
MEM: 32 : Hex: 00000000>_

```

Figure 12: Running Driver in Interactive Mode

When first launched, the following prompt is displayed:

```
MEM: 32 : Hex: 00000000>
```

The fields displayed in the prompt are defined in [Table 3](#).

Table 3: MET Application Prompt Definition

Field	Definition
1 - MEM	The current PCI™ space; one of: MEM (memory space), I/O (I/O or “port” space), CNF (config space).
2 - 32	The default bit-width for reads/writes and input data (currently 8, 16, and 32 are supported).
3 - HEX	The default radix for display and input; one of: Oct (base 8), Dec (base 10), or Hex (base 16).
4 - 00000000	The current address offset in the current space, displayed in the current radix.

Table 4 shows the supported commands that can be used to access the memory space of the PIO design. Commands are not case-sensitive. Only the first 255 characters are parsed.

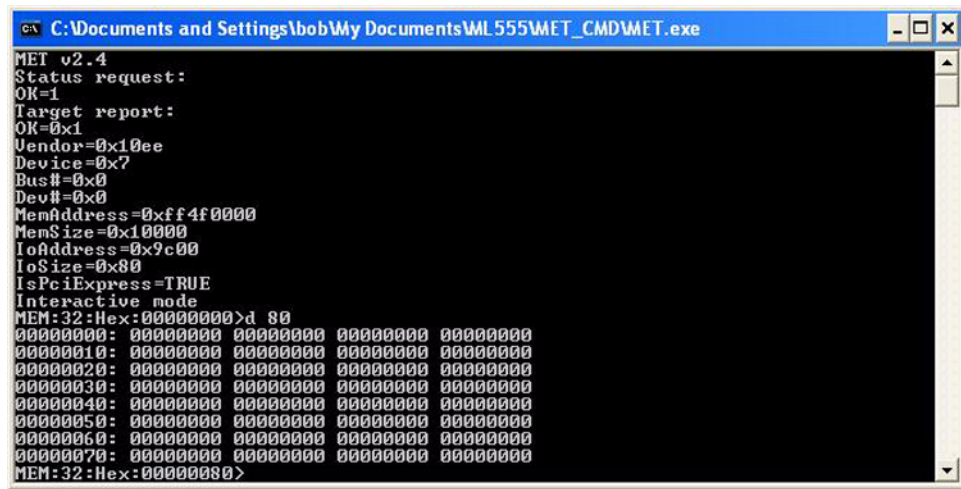
Table 4: MET Application Commands

Syntax	Description	Example
Access { C I M }	Changes the current space to Config, I/O, or Memory, respectively. Only memory space accesses are currently supported.	A M Change to Memory Space
Width { B W D }	Changes width to byte, word (16-bit), or doubleword (32-bit), respectively.	W D Change width to 32-bit
Radix { O D H }	Changes radix to octal, decimal, or hex, respectively.	R D change radix to decimal
Location <Offset>	Changes the current address offset to that given. By default, offset is parsed per current radix, but “C” notation is also accepted.	L 0x40 Change offset to 40 (hex) regardless of current radix
Dump <Count>	Dumps data, starting at the current address offset, for the given number of bytes and updates the current address offset.	D 40 Dump 40 (current radix) bytes
Next	Advances the dump; i.e., dumps the same number of bytes as given in the last dump command.	N Assuming above, dump 40 bytes
Set <Datum> [... <Datum>]	Writes the given data starting at the current address offset. Data is parsed assuming the current radix and bit-width. Shortcuts are provided, which override the current bit-width: sb for byte, sw for word, sd for doubleword.	S A5 88 F00F Assuming 16-bit and hex, write the 3 words at the current offset. SD AAAA5555 Write the 32-bit datum
Config	Performs an annotated dump of the PCI and PCI Express config space.	C Analyze config space
Test <TestSpecFile>	Runs the test suite from the given file (only once, vs. the continuous mode).	T memtest.ini Run the test suite in memtest.ini
Exit	Exits the program.	E

Interactive Mode User Examples

Example 1:

Dump 0x80 bytes (128 bytes decimal) of memory space at the current offset of 0x000000.



```

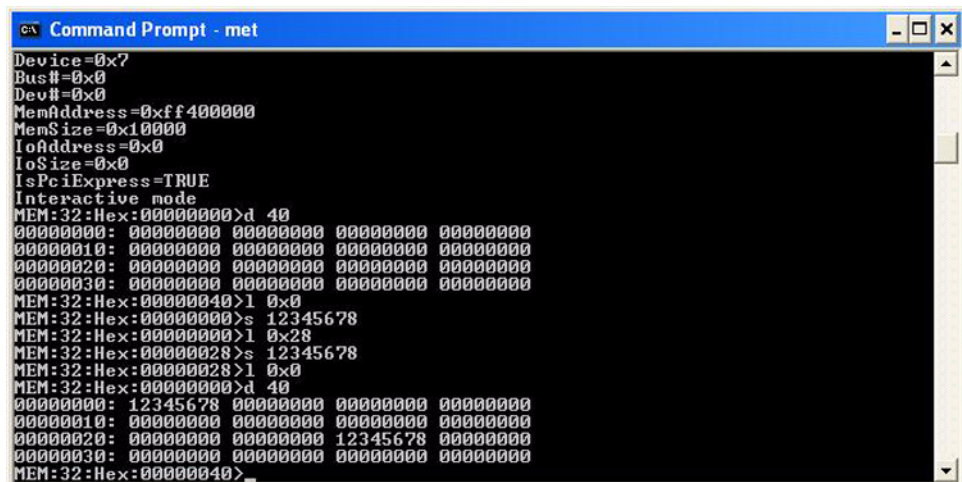
C:\Documents and Settings\lbob\My Documents\WL555\MET_CMD\MET.exe
MET v2.4
Status request:
OK=1
Target report:
OK=0x1
Vendor=0x10ee
Device=0x7
Bus#=0x0
Dev#=0x0
MemAddress=0xff4f0000
MemSize=0x10000
IoAddress=0x9c00
IoSize=0x80
IsPciExpress=TRUE
Interactive mode
MEM:32:Hex:00000000>d 80
00000000: 00000000 00000000 00000000 00000000
00000010: 00000000 00000000 00000000 00000000
00000020: 00000000 00000000 00000000 00000000
00000030: 00000000 00000000 00000000 00000000
00000040: 00000000 00000000 00000000 00000000
00000050: 00000000 00000000 00000000 00000000
00000060: 00000000 00000000 00000000 00000000
00000070: 00000000 00000000 00000000 00000000
MEM:32:Hex:00000080>

```

Figure 13: Displaying Memory Space Contents

Example 2:

1. Dump 0x40 bytes of memory space (d 40).
2. Change the address offset back to the beginning (l 0x0).
3. Write one DWord at address offset 0x00000000 (s 12345678).
4. Change the address offset to 0x28 (l 0x28).
5. Write one DWord at address offset 0x00000028 (s 12345678).
6. Change the address offset back to the beginning (l 0x0).
7. Display what was written (d 40).



```

Command Prompt - met
Device=0x7
Bus#=0x0
Dev#=0x0
MemAddress=0xff400000
MemSize=0x10000
IoAddress=0x0
IoSize=0x0
IsPciExpress=TRUE
Interactive mode
MEM:32:Hex:00000000>d 40
00000000: 00000000 00000000 00000000 00000000
00000010: 00000000 00000000 00000000 00000000
00000020: 00000000 00000000 00000000 00000000
00000030: 00000000 00000000 00000000 00000000
MEM:32:Hex:00000040>l 0x0
MEM:32:Hex:00000000>s 12345678
MEM:32:Hex:00000000>l 0x28
MEM:32:Hex:00000028>s 12345678
MEM:32:Hex:00000028>l 0x0
MEM:32:Hex:00000000>d 40
00000000: 12345678 00000000 00000000 00000000
00000010: 00000000 00000000 00000000 00000000
00000020: 00000000 00000000 12345678 00000000
00000030: 00000000 00000000 00000000 00000000
MEM:32:Hex:00000040>

```

Figure 14: Accessing Memory Space

Running a Test Suite from an “ini” File

The MET driver allows testing from a specified file using the format outlined in this section. The test file can be executed as an argument, or it can be run in the interactive mode using the “T” command.

- C:\>MET memtest.ini
- MEM:32:Hex:00000000>T memtest.ini

Contents of the Test Specification File

```
[Suite]

TestCount=4; how many tests are specified in the sections below

TestGap=2 ; how many seconds to delay between tests
; (i.e. between Test1 and Test2, etc.)

TestMode=List; optional, use to run entire suite in kernel mode,
; uninterrupted by app requests
[Test1]

Count=200 ; how many times to repeat this test before moving to next one
; the starting data pattern is rotated at each
; iteration, so a different set
; of data is used each iteration
Delay=10;optional microsecond delay between iterations of this test
;(iteration means the 200 iterations here since
; Count=200)

Space=M ; space (M=memory, I=I/O, C=config)

Length=0x10000 ; how many bytes to transfer (fixed) or the most to transfer
(random)

RandomLength=TRUE ; TRUE for random length; FALSE or omitted for fixed-
length

Offset=0x28000 ; offset within device's resource

Test=W ; R=read "Count" times (just a loop of reads for a
; "scope-loop" test)
; W=write "Count" times and read/verify at the end
; WI=write w/ immediate readback (i.e. to force bus
; bridge to perform
; small writes)

WriteWidth=2 ; optional; write using word (16-bit) accesses (default is
byte)
; this also controls the data pattern's wrapping

ReadWidth=4 ; optional
; read using double-word (32-bit) accesses
;(default=byte)

Pattern=W1 ; W0=walking 0s (all 1 bits except one 0, pattern
; rotates at every write)
; W1=walking 1s (inverse of W0)
; A0=all 0s
; A1=all 1s
; L=alternating 1/0 pattern
; R=random
```

```

; in absence of a pre-defined pattern, parses as a
; numeric value
; to use throughout
[Test3]
...
[Test4]
...

```

MET GUI

Included in `xapp1022.zip` is a GUI application that will write and read packets to the endpoint application and note any errors that occurred. This application is located in a sub-directory of the ZIP file called `MET_GUI`. To launch the GUI shown in [Figure 15](#), double-click the `MET_GUI.bat` file.



Figure 15: MET GUI for Windows XP

Using the MET Driver for Linux

All Linux software was built and tested on Fedora core 10. It is likely to work on different variations of Linux, but none have been tested. The driver source and application are provided as is with no implied support or warranty. Xilinx appreciates any feedback regarding problems and solutions found with different versions of Linux and will try to incorporate these in future updates. To provide feedback, open a webcase and include details about:

- Linux distribution
- Version
- Description of the problem
- Work around, if found

Linux Driver Contents

The Linux driver is included in the `xapp1022.zip` in the directory called `Linux_Driver`. The contents of this folder needed to install and run the driver are:

- `README` : Describes loading the driver.
- `xpcie.c` : XPCIE device driver.
- `met.cpp` : Example memory endpoint test. Modify this file to run other test.
- `make_device` : Script that creates the xpcie device on the system. Must be run before loading the device driver.
- `Makefile` : Makefile.
 - ◆ Compiles `xpcie.c` into a loadable `xpcie.ko`.
 - ◆ Compiles `met.cpp` into MET executable.

Compiling and Installing on Fedora Linux

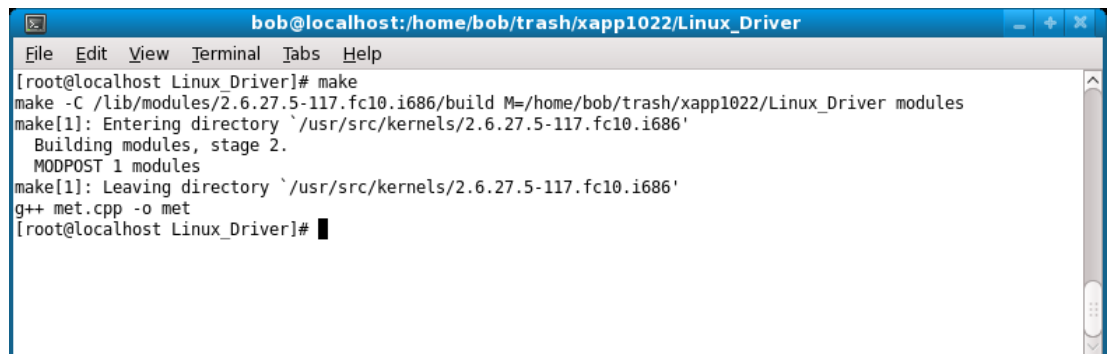
The Vendor ID of the driver is set to 10EEh, and the Device ID is set to 0007h by default. If this is different from the generated core, edit the `xpcie.c` file and change these two lines to match the Vendor and Device ID in use:

```
#define PCI_VENDOR_ID_XILINX      0x10ee
#define PCI_DEVICE_ID_XILINX_PCIE 0x0007
```

Installation of the driver requires root privileges. To install the driver and run the test, go to the directory containing the driver files and type the following commands:

1. `make`

The expected output is shown in [Figure 16](#).

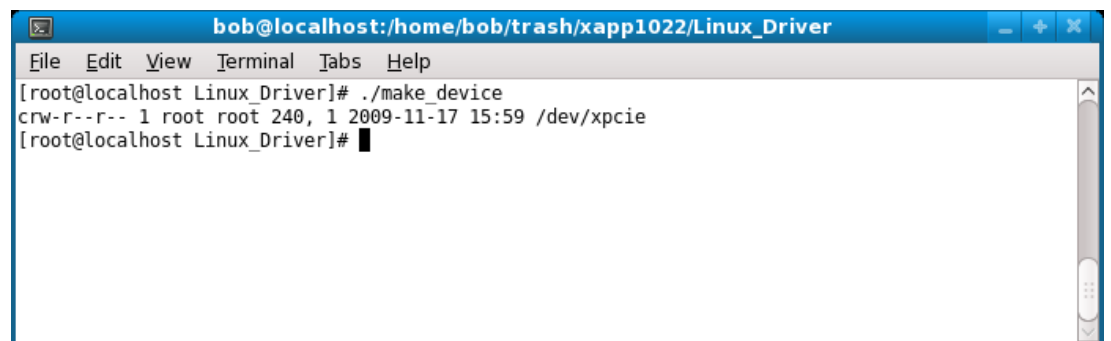


```
bob@localhost:/home/bob/trash/xapp1022/Linux_Driver
File Edit View Terminal Tabs Help
[root@localhost Linux_Driver]# make
make -C /lib/modules/2.6.27.5-117.fc10.i686/build M=/home/bob/trash/xapp1022/Linux_Driver modules
make[1]: Entering directory `/usr/src/kernels/2.6.27.5-117.fc10.i686'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/kernels/2.6.27.5-117.fc10.i686'
g++ met.cpp -o met
[root@localhost Linux_Driver]#
```

Figure 16: Make Command Output

2. `./make_device`

The expected output is highlighted in [Figure 17](#).



```
bob@localhost:/home/bob/trash/xapp1022/Linux_Driver
File Edit View Terminal Tabs Help
[root@localhost Linux_Driver]# ./make_device
crw-r--r-- 1 root root 240, 1 2009-11-17 15:59 /dev/xcpcie
[root@localhost Linux_Driver]#
```

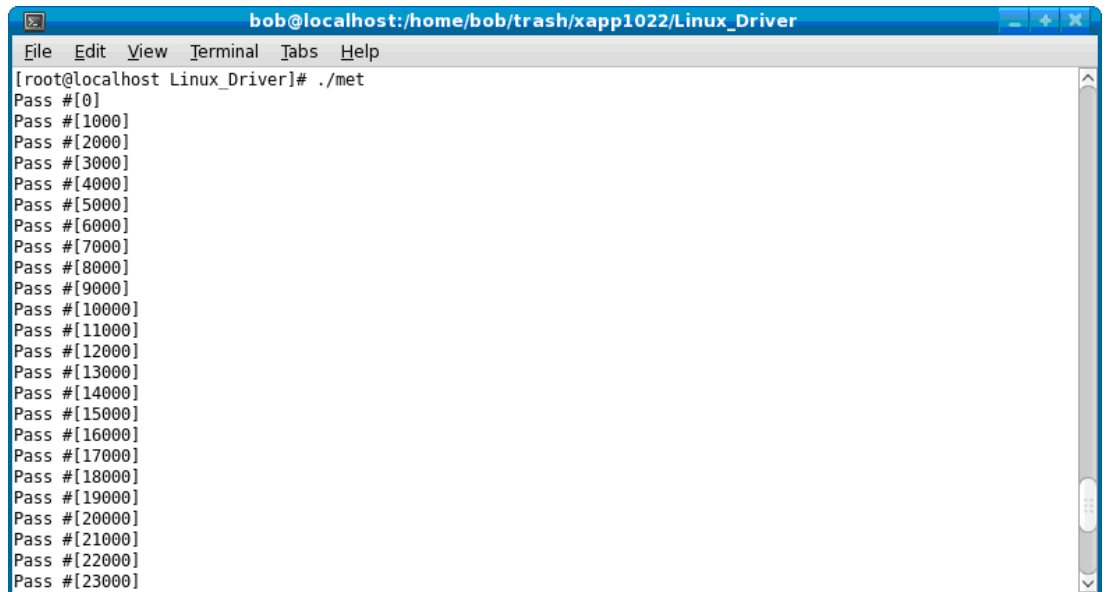
Figure 17: Make_device Command Output

3. `insmod xpcie.ko`

There is no noticeable output after running `insmod`. Note that without root privileges `insmod` will fail to load the driver.

4. `./met`

This runs the application and expected output is shown in [Figure 18](#).



```

bob@localhost:/home/bob/trash/xapp1022/Linux_Driver
File Edit View Terminal Tabs Help
[root@localhost Linux_Driver]# ./met
Pass #[0]
Pass #[1000]
Pass #[2000]
Pass #[3000]
Pass #[4000]
Pass #[5000]
Pass #[6000]
Pass #[7000]
Pass #[8000]
Pass #[9000]
Pass #[10000]
Pass #[11000]
Pass #[12000]
Pass #[13000]
Pass #[14000]
Pass #[15000]
Pass #[16000]
Pass #[17000]
Pass #[18000]
Pass #[19000]
Pass #[20000]
Pass #[21000]
Pass #[22000]
Pass #[23000]

```

Figure 18: MET Command Output

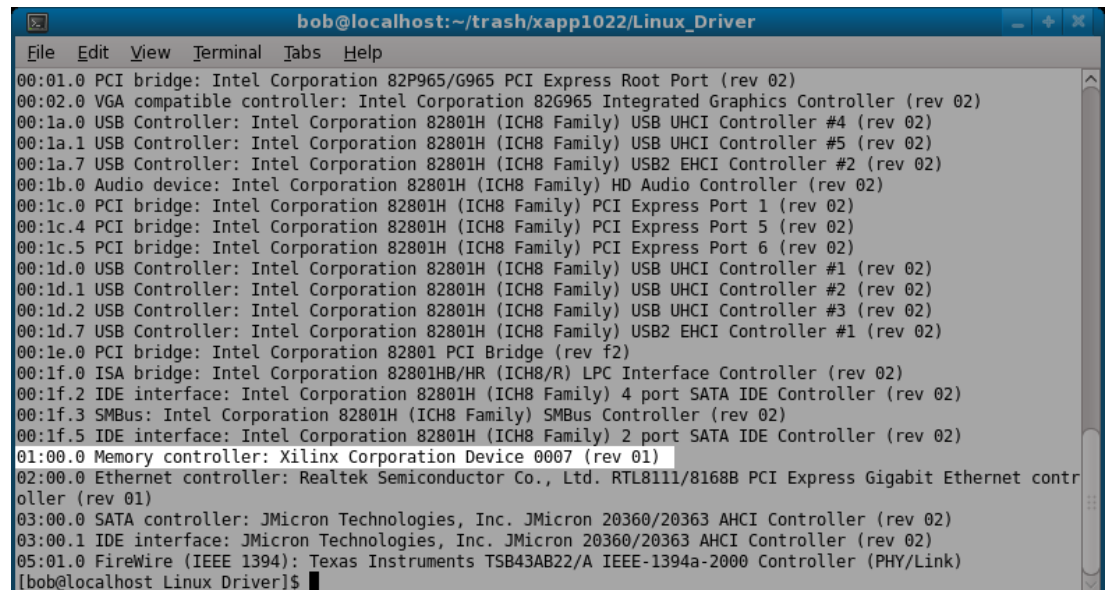
Debugging Problems

If problems are encountered, use the following commands to determine what is happening.

1. Verify the card is recognized by the system by using `lspci`. At the prompt type:

```
lspci
```

Look for the output highlighted in [Figure 19](#).



```

bob@localhost:~/trash/xapp1022/Linux_Driver
File Edit View Terminal Tabs Help
00:01.0 PCI bridge: Intel Corporation 82P965/G965 PCI Express Root Port (rev 02)
00:02.0 VGA compatible controller: Intel Corporation 82G965 Integrated Graphics Controller (rev 02)
00:1a.0 USB Controller: Intel Corporation 82801H (ICH8 Family) USB UHCI Controller #4 (rev 02)
00:1a.1 USB Controller: Intel Corporation 82801H (ICH8 Family) USB UHCI Controller #5 (rev 02)
00:1a.7 USB Controller: Intel Corporation 82801H (ICH8 Family) USB2 EHCI Controller #2 (rev 02)
00:1b.0 Audio device: Intel Corporation 82801H (ICH8 Family) HD Audio Controller (rev 02)
00:1c.0 PCI bridge: Intel Corporation 82801H (ICH8 Family) PCI Express Port 1 (rev 02)
00:1c.4 PCI bridge: Intel Corporation 82801H (ICH8 Family) PCI Express Port 5 (rev 02)
00:1c.5 PCI bridge: Intel Corporation 82801H (ICH8 Family) PCI Express Port 6 (rev 02)
00:1d.0 USB Controller: Intel Corporation 82801H (ICH8 Family) USB UHCI Controller #1 (rev 02)
00:1d.1 USB Controller: Intel Corporation 82801H (ICH8 Family) USB UHCI Controller #2 (rev 02)
00:1d.2 USB Controller: Intel Corporation 82801H (ICH8 Family) USB UHCI Controller #3 (rev 02)
00:1d.7 USB Controller: Intel Corporation 82801H (ICH8 Family) USB2 EHCI Controller #1 (rev 02)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev f2)
00:1f.0 ISA bridge: Intel Corporation 82801HB/HR (ICH8/R) LPC Interface Controller (rev 02)
00:1f.2 IDE interface: Intel Corporation 82801H (ICH8 Family) 4 port SATA IDE Controller (rev 02)
00:1f.3 SMBus: Intel Corporation 82801H (ICH8 Family) SMBus Controller (rev 02)
00:1f.5 IDE interface: Intel Corporation 82801H (ICH8 Family) 2 port SATA IDE Controller (rev 02)
01:00.0 Memory controller: Xilinx Corporation Device 0007 (rev 01)
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168B PCI Express Gigabit Ethernet controller (rev 01)
03:00.0 SATA controller: JMicron Technologies, Inc. JMicron 20360/20363 AHCI Controller (rev 02)
03:00.1 IDE interface: JMicron Technologies, Inc. JMicron 20360/20363 AHCI Controller (rev 02)
05:01.0 FireWire (IEEE 1394): Texas Instruments TSB43AB22/A IEEE-1394a-2000 Controller (PHY/Link)
[bob@localhost Linux_Driver]$

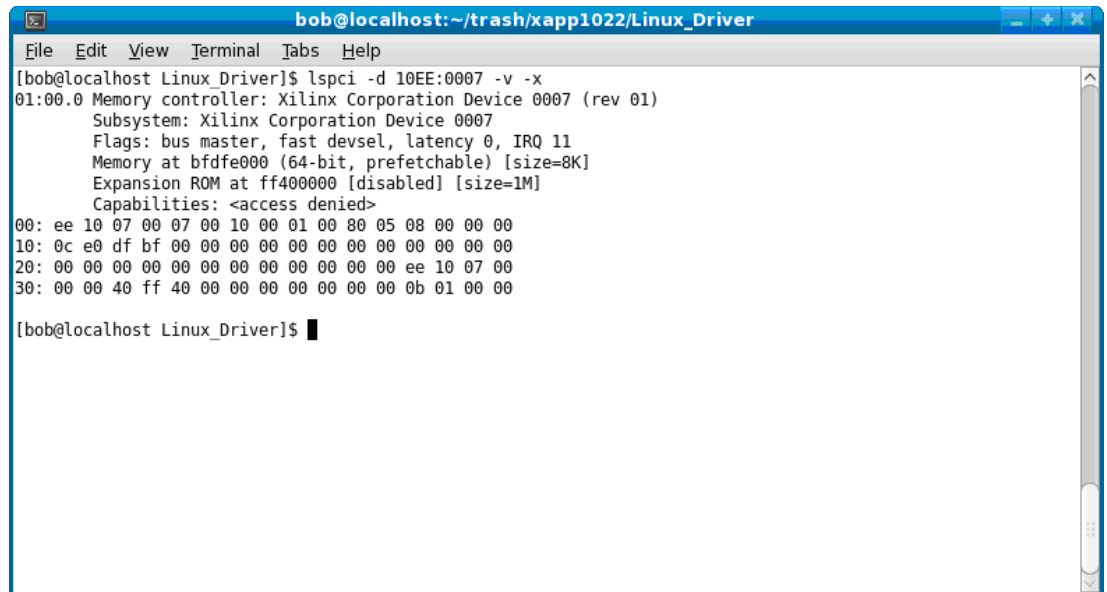
```

Figure 19: LSPCI Command Output

To see more details of the device and verify the BAR is set type:

```
lspci -d 10EE:0007 -v -x
```

This will give the output shown in [Figure 20](#).



```

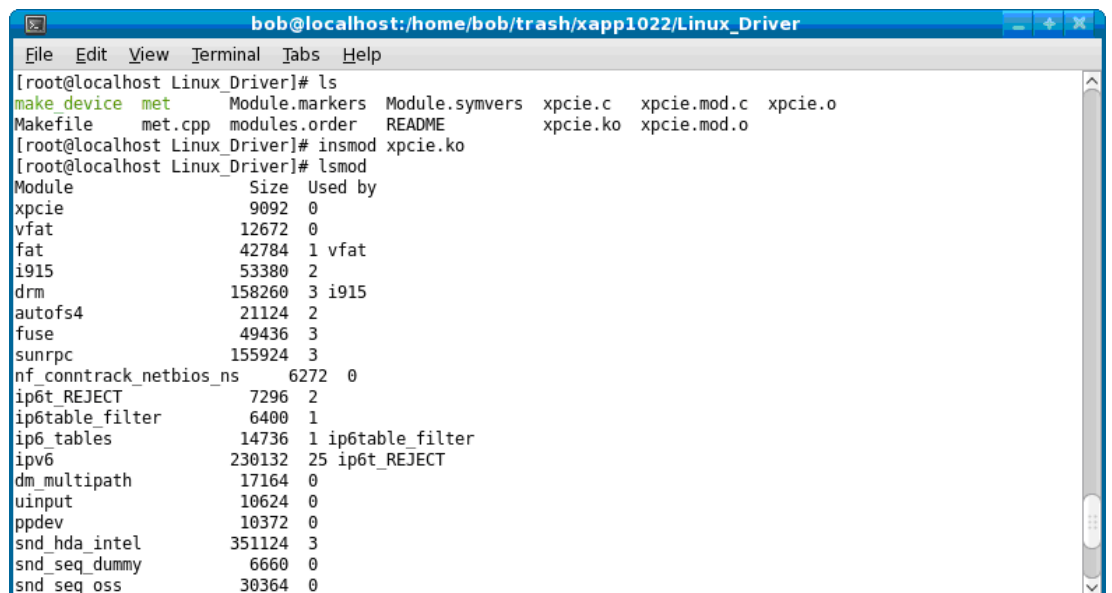
bob@localhost:~/trash/xapp1022/Linux_Driver
File Edit View Terminal Tabs Help
[bob@localhost Linux_Driver]$ lspci -d 10EE:0007 -v -x
01:00.0 Memory controller: Xilinx Corporation Device 0007 (rev 01)
Subsystem: Xilinx Corporation Device 0007
Flags: bus master, fast devsel, latency 0, IRQ 11
Memory at bfdfe000 (64-bit, prefetchable) [size=8K]
Expansion ROM at ff400000 [disabled] [size=1M]
Capabilities: <access denied>
00: ee 10 07 00 07 00 10 00 01 00 80 05 08 00 00 00
10: 0c e0 df bf 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 ee 10 07 00
30: 00 00 40 ff 40 00 00 00 00 00 00 00 0b 01 00 00

[bob@localhost Linux_Driver]$

```

Figure 20: Detailed LSPCI Output

2. Type **dmesg** to see the kernel message output. If a problem occurs while trying to load the driver, this command may provide helpful messages.
3. Use the **lsmod** command to verify the device is loaded on the system. You should see an output similar to [Figure 21](#).



```

bob@localhost:/home/bob/trash/xapp1022/Linux_Driver
File Edit View Terminal Tabs Help
[root@localhost Linux_Driver]# ls
make_device met Module.markers Module.symvers xpcie.c xpcie.mod.c xpcie.o
Makefile met.cpp modules.order README xpcie.ko xpcie.mod.o
[root@localhost Linux_Driver]# insmod xpcie.ko
[root@localhost Linux_Driver]# lsmod
Module Size Used by
xpcie 9092 0
vfat 12672 0
fat 42784 1 vfat
i915 53380 2
drm 158260 3 i915
autofs4 21124 2
fuse 49436 3
sunrpc 155924 3
nf_conntrack_netbios_ns 6272 0
ip6t_REJECT 7296 2
ip6table_filter 6400 1
ip6_tables 14736 1 ip6table_filter
ip6v6 230132 25 ip6t_REJECT
dm_multipath 17164 0
uinput 10624 0
ppdev 10372 0
snd_hda_intel 351124 3
snd_seq_dummy 6660 0
snd_seq_oss 30364 0

```

Figure 21: LSMOD Command Output

Bitstreams

Included in `xapp1022.zip` are bitstreams for the Virtex-6 ML605, Spartan-6 SP605, and Virtex-5 ML555 development boards that will support the MET driver application. These bitstreams are located in a subdirectory of the ZIP file called `bitstreams`.

Conclusion

The MET driver and application provide a simple interface to read and write to the PIO example design included with Xilinx Endpoint solutions for PCI Express. Instructions for installing and exercising the driver on a Windows XP operating system are provided in this document.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/19/07	1.0	Initial Xilinx release.
11/20/09	2.0	Added support for Virtex-6 and Spartan-6 FPGA Endpoint cores.

Notice of Disclaimer

Xilinx is providing this product documentation, hereinafter "Information," to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.