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 the Endpoint Block Plus Wrapper, Endpoint, and Endpoint PIPE for PCI Express® Xilinx solutions. 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 three cores for PCI Express to be used in typical 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

<table>
<thead>
<tr>
<th>Core Name</th>
<th>Device(s) Supported</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint Block Plus Wrapper for PCI Express</td>
<td>Virtex™-5</td>
<td>Utilizes the Virtex-5 Built-in Endpoint Block for PCI Express</td>
</tr>
<tr>
<td>Endpoint for PCI Express</td>
<td>Virtex-5, Virtex-4, and Virtex-II Pro</td>
<td>Soft-IP Implementation</td>
</tr>
<tr>
<td>Endpoint PIPE for PCI Express</td>
<td>Spartan™-3, Spartan-3A, and Spartan-3E</td>
<td>Soft-IP implementation utilizing an external physical layer from NXP Semiconductors</td>
</tr>
</tbody>
</table>

More information about the current versions of these cores is available in the endpoint data sheets located on the product lounge Web site. The specific product lounges are linked by the Core Name column in Table 1. Visit the Xilinx solutions for PCI Express for more information about PCI Express cores.

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 for the core. This document is delivered with the core, but can also be downloaded from the product lounge for the specific core.
Setting Up the PIO Example Design

By default, the endpoint core includes a working example design that can be downloaded to an add-in card and inserted into any PCI Express system. The Getting Started Guide contains 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. Ensure the latest Xilinx design tools are installed, along with applicable IP updates and patches for the core. The latest IP updates can be found at: http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp
2. Start the CORE Generator software and create a new project.
3. Target a part that supports either the Endpoint for PCI Express, Endpoint PIPE for PCI Express, or Endpoint Block Plus.
   See the CORE Generator User Guide for assistance with creating a new project.
4. In the taxonomy tree, select Standard Bus Interface > PCI Express.
5. Select Endpoint, Endpoint PIPE, or Endpoint Block Plus and click Customize.
6. Change the Sub-Class code to 80 to indicate an “Other Memory Controller” to the system (Figure 1).
7. Click Next to continue.

Figure 1: Class Code Settings for PIO Design
8. Change the BAR settings to implement a single Memory BAR. The recommended size is 8 Kilobytes, but is user configurable. An example is shown in Figure 2.

![Memory BAR Settings for PIO Design](image)

**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 search for the following code and make the change shown in red text.

Comment out the line of code that is used to differentiate the four memory regions in the 8 kB of memory, and replace it with the full 8 kB address from the incoming transaction in both the read and write states of the 32-bit addressable memory request.

```vhdl
PIO_64_RX_ENGINE.v or PIO_32_RX_ENGINE.v

`PIO_64_RX_MEM_RD32_DW1DW2 : begin
    if (!trn_rscc_rdy_n) begin
        trn_rdst_rdy_n <= #`TCQ 1'b1;
        //req_addr_o   <= #`TCQ {region_select[1:0],trn_rd[42:34], 2'b00};
        req_addr_o   <= #`TCQ {trn_rd[42:34], 2'b00};
        req_compl_o  <= #`TCQ 1'b1;
        state        <= #`TCQ `PIO_64_RX_WAIT_STATE;
    end else
        state        <= #`TCQ `PIO_64_RX_MEM_RD32_DW1DW2;
    end

`PIO_64_RX_MEM_WR32_DW1DW2 : begin
    if (!trn_rscc_rdy_n) begin
        wr_data_o   <= #`TCQ trn_rd[31:0];
        wr_en_o     <= #`TCQ 1'b1;
        trn_rdst_rdy_n <= #`TCQ 1'b1;
        //wr_addr_o      <= #`TCQ {region_select[1:0],trn_rd[42:34]};
        wr_addr_o    <= #`TCQ trn_rd[44:34];
        state        <= #`TCQ `PIO_64_RX_WAIT_STATE;
    end else
        state        <= #`TCQ `PIO_64_RX_MEM_WR32_DW1DW2;
    end
```
PIO 64 RX ENGINE.vhd or PIO 32 RX ENGINE.vhd

-- first and second dwords of mem32 read tlp
when RX_MEM_RD32_DW1DW2 =>
  if (trn_rsrc_rdy_n = '0') then
    trn_rdstart_rdy_n_int <= '1';
    --req_addr_o <= region_select(1 downto 0) & trn_rd(42 downto 34) & "00";
    req_addr_o <= trn_rd(44 downto 34) & "00";
    req_compl_o <= '1';
    state <= RX_WAIT_STATE;
  else
    state <= RX_MEM_RD32_DW1DW2;
  end if;

-- first and second dwords of mem32 write tlp
when RX_MEM_WR32_DW1DW2 =>
  if (trn_rsrc_rdy_n = '0') then
    wr_data_o <= trn_rd(31 downto 0);
    wr_en_o <= '1';
    trn_rdstart_rdy_n_int <= '1';
    --wr_addr_o <= region_select(1 downto 0) & trn_rd(42 downto 34);
    wr_addr_o <= trn_rd(44 downto 34);
    state <= RX_WAIT_STATE;
  else
    state <= RX_MEM_WR32_DW1DW2;
  end if;

See the Endpoint Block Plus Wrapper, Endpoint, or Endpoint PIPE 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 for Windows XP

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.

Note: The driver is available for download with this Application Note. See “MET Driver and Application” to download the files.

Installing the Driver

To install the driver, follow these steps:

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

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

   Figure 3: Welcome to the Found New Hardware Wizard

   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**.

![Found New Hardware Wizard](image)

**Figure 5:** Change Search and Installation Options

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

![Found New Hardware Wizard](image)

**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 “xlinx_PCIE_block.inf.” Select this file, click Open, and then click OK.

![Locate the Driver File](image1)

**Figure 7:** Locate the Driver File

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

![Install the Driver](image2)

**Figure 8:** Install the Driver
7. Progress of the installation is displayed.

![Wizard Installs the Software](image)

**Figure 9:** Wizard Installs the Software

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

![Hardware Update Complete](image)

**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 the MET Driver on Windows XP

### MET Arguments

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

**Table 2: MET Application Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>--log &lt;LogFile&gt;</td>
<td>[OPTIONAL] Logs all test results and non-prompt output to the given log file. For safety, it will not overwrite an existing log file.</td>
<td>MET --log log1.txt</td>
</tr>
<tr>
<td>--script &lt;ScriptFile&gt;</td>
<td>[OPTIONAL] Accepts and executes interactive &quot;monitor&quot; commands from the given file.</td>
<td>MET --script cmds.txt</td>
</tr>
<tr>
<td>&lt;TestSpec&gt;</td>
<td>[OPTIONAL] Executes the tests specified in the given file, in the &quot;.ini&quot; format documented below.</td>
<td>MET Memtest1.ini</td>
</tr>
</tbody>
</table>
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.

![Figure 12: Running Driver in Interactive Mode](image)

When first launched, the following prompt is displayed:

MEM:32:Hex:00000000>

The fields displayed in the prompt are defined in **Table 3**.

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

**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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access { C</td>
<td>I</td>
<td>M}</td>
</tr>
<tr>
<td>Width {B</td>
<td>W</td>
<td>D}</td>
</tr>
<tr>
<td>Radix {O</td>
<td>D</td>
<td>H}</td>
</tr>
<tr>
<td>Location &lt;Offset&gt;</td>
<td>Changes the current address offset to that given. By default, offset is parsed per current radix, but “C” notation is also accepted.</td>
<td>L 0x40 Change offset to 40 (hex) regardless of current radix</td>
</tr>
<tr>
<td>Dump &lt;Count&gt;</td>
<td>Dumps data, starting at the current address offset, for the given number of bytes and updates the current address offset.</td>
<td>D 40 Dump 40 (current radix) bytes</td>
</tr>
<tr>
<td>Next</td>
<td>Advances the dump; i.e., dumps the same number of bytes as given in the last dump command.</td>
<td>N Assuming above, dump 40 bytes</td>
</tr>
<tr>
<td>Set &lt;Datum&gt; […]&lt;Datum&gt;]</td>
<td>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.</td>
<td>S A5 88 F00F Assuming 16-bit and hex, write the 3 words at the current offset. SD AAAA5555 Write the 32-bit datum</td>
</tr>
<tr>
<td>Config</td>
<td>Performs an annotated dump of the PCI and PCI Express config space.</td>
<td>C Analyze config space</td>
</tr>
<tr>
<td>Test &lt;TestSpecFile&gt;</td>
<td>Runs the test suite from the given file (only once, vs. the continuous mode).</td>
<td>T memtest.ini Run the test suite in memtest.ini</td>
</tr>
<tr>
<td>Exit</td>
<td>Exits the program.</td>
<td>E</td>
</tr>
</tbody>
</table>
Interactive Mode User Examples

Example 1:
Dump 0x80 bytes (128 bytes decimal) of memory space at the current offset of 0x000000.

![Displaying Memory Space Contents](image1.png)

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).

![Accessing Memory Space](image2.png)

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
```

www.BDTIC.com/XILINX
The MET driver and application are available in the “xapp1022.zip”. Unzip this file to any directory and follow the instructions in the section “Installing the Driver”. 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 the file “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 the “xilinx_pcie_block.inf” file is modified, the driver must be re-installed.

**MET GUI**

Included in the “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 subdirectory of the ZIP file called MET_GUI. To launch the GUI, double-click the “launch_gui.bat” file.

**ML555 Bitstreams**

Included in the “xapp1022.zip” are bitstreams for the ML555 development board that will support the MET driver application. These bitstreams are located in a subdirectory of the ZIP file called ML555_bitstreams. Please refer to the “readme.txt” file in this directory for explanations of the included files.

**Conclusion**

The MET driver and application provide a simple interface to read and write to the PIO example design included with the Endpoint Block Plus Wrapper, Endpoint, and Endpoint PIPE Xilinx solutions. Instructions for installing and exercising the driver on a Windows XP operating system are provided in this document.
The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/19/07</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
</tbody>
</table>

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