Abstract
This application note outlines the techniques for debugging PowerPC® 440 processor systems in hardware and simulation.

Included Systems
Included with this application note is one reference system, V5_PPC440_debug, built for the Xilinx ML507 Rev A board:


Introduction
After building a system, a user wants to ensure the system is functional. A system consists of different combined components, such as processor, bus, and peripherals. To ensure that components are working correctly, the user can simulate the system or test the system in hardware for desired behavior. For simulation, a simulation environment is necessary. With hardware, certain debug programs and cables are necessary.

This application note describes how to debug/monitor system behavior through simulation and hardware. A system simulation environment is set up and hardware debug tools are described. A software application provides stimulus to the system which the behavior is demonstrated in simulation and in hardware.

Hardware and Software Requirements
The hardware and software requirements are:

- Xilinx ML507 Rev A board
- Xilinx Platform USB Cable or Parallel IV Cable
- RS232 Cable
- Xilinx Platform Studio 10.1.02
- Xilinx Integrated Software Environment (ISE®) 10.1.02

Simulation Requirements
Simulation environment is verified with:

- Red Hat Enterprise Linux 3.0 or Windows XP
- ModelTech SE 6.3c with VHDL and Verilog co-simulation capability
- Xilinx Platform Studio 10.1.02
- Xilinx Integrated Software Environment (ISE) 10.1.02

Reference System Specifics
This system uses the PowerPC 440 processor block with a processor frequency of 400 MHz and the Memory Interface Block (MIB) frequency of 266 MHz. The processor block crossbar is set to 266 MHz. In addition, the frequency for the MPLB and SPLB0 is set to 133 MHz. For more information about valid clocking ratios and the PLL clocks controlling the processor block, see Virtex®-5 Embedded Block for PowerPC 440 Processor Designs.
The reference system includes a basic system with PPC440MC DDR2, XPS BRAM, XPS UART16550, XPS GPIO, and XPS INTC. The PPC440MC DDR2 is connected to the MIB on the processor block with a frequency of 266 MHz.

The description of the XPS MASTER EXAMPLE core and its addition to the system is outlined later in this application note. The slave connection of the XPS MASTER EXAMPLE core is connected to the MPLB. In addition, the XPS BRAM, XPS INTC, and XPS UART16550 cores are connected as slaves to the MPLB.

The master connection on the XPS MASTER EXAMPLE core is connected to the SPLB0 port on the processor block. The PowerPC 440 processor is set up for MPLB transactions to occur between the address ranges of 0x40000000 and 0xFFFFFFFF. All slave peripheral addresses in this system are in this range.

Adding ChipScope cores to the system for debugging is discussed in this application note. Separate PLB v4.6 IBA cores are added to monitor PLB v4.6 bus signals on SPLB0 and MPLB. In addition, an ILA core is added to monitor signals on the Memory Controller Interface (MCI).

**Block Diagram**

![Reference System Block Diagram](image)

**Address Map**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Peripheral</th>
<th>Base Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppc440mc_ddr2_0</td>
<td>ppc440_mc_ddr2</td>
<td>0x00000000</td>
<td>0xFFFFFFFF</td>
</tr>
<tr>
<td>xps_uart16550_0</td>
<td>xps_uart16550</td>
<td>0x40400000</td>
<td>0x4040FFFF</td>
</tr>
<tr>
<td>ddr2_init</td>
<td>xps_gpio</td>
<td>0x40040000</td>
<td>0x4004FFFF</td>
</tr>
<tr>
<td>xps_intc_0</td>
<td>xps_intc</td>
<td>0x41200000</td>
<td>0x4120FFFF</td>
</tr>
<tr>
<td>xps_master_example_0</td>
<td>xps_master_example</td>
<td>0xA0000000</td>
<td>0xA000FFFF</td>
</tr>
</tbody>
</table>

**Reference System Specifics**

This section describes setting addresses for the crossbar, describing and connecting the XPS MASTER EXAMPLE core, adding the XPS GPIO for the PPC440MC DDR2 ready to accept signal, and adding ChipScope cores to the system. These steps are already completed inside the reference system.
Setting Addresses for the Crossbar

Setting the PowerPC 440 processor address ranges correctly for masters connected to SPLB0/SPLB1 is important in the system.

In the Bus Interfaces tab, right click on `ppc440_0` and select **Configure IP...**.

The user is responsible in setting the correct address ranges for SPLB0 and SPLB1. The user has the option of allowing SPLB0/SPLB1 to execute master transactions to either the MPLB or MIB or both interfaces. In this reference system, only SPLB0 is configured for masters to execute on both the MPLB and MIB, but the same configuration could apply to SPLB1.

The master in this example should be able to execute transactions to both slave peripherals on the MPLB and the PPC440MC DDR2. Select the **Allow SPLB0 to Access MPLB Addr** check box. If the check box is not selected, the master cannot execute transactions to slave peripherals on the MPLB.

The user configures the address ranges for the MC and MPLB. In this example the memory controller is between `0x00000000` and `0x0FFFFFFF`.

For SPLB0, the MemCon Range Base Addr and High Addr is set to `0x00000000` and `0x0FFFFFFF` respectively. The address range of the slave peripherals in this reference system are between `0x40000000` and `0x7FFFFFFF`. To accommodate this address range, the Number of MPLB Addr Ranges is set to 2, and MPLB Range 0 Base Addr and High Addr are set to `0x40000000` and `0x7FFFFFFF` and MPLB Range 1 Base Addr and High Addr are set to `0x80000000` and `0xFFFFFFFF`, respectively. These settings are shown in Figure 2.

![PowerPC Processor Address Tab](image)

**Figure 2:** PPC440 Processor Address Tab
PLB v4.6 Master and Slave Core

The XPS MASTER EXAMPLE core provided in this application note is similar to a master and slave core created by Create IP Wizard. However, the slave registers have different locations and this core is only built and verified with a 32-bit master and slave native dwidth. Both this core and the Create IP Wizard core use a FIFO that stores the read data after a read and pops data off the FIFO during a write for the write data. This core allows for single and burst writes and reads.

The XPS MASTER EXAMPLE slave interface of the core is connected to the MPLB and the master interface is connected to SPLB0 PLB v4.6 instances.

The slave registers are shown in Table 2.

<table>
<thead>
<tr>
<th>Register Name</th>
<th>(offset from C_BASEADDR)</th>
<th>Access</th>
<th>Size in Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Register</td>
<td>0x0</td>
<td>Read/Write</td>
<td>8</td>
<td>Bit0 - Read Bit1 - Write Bit2 - Single Bit3 - Burst</td>
</tr>
<tr>
<td>Status Register</td>
<td>0x1</td>
<td>Read</td>
<td>8</td>
<td>Bit0 - Done Bit1 - Busy</td>
</tr>
<tr>
<td>IP2Bus Address Register</td>
<td>0x8</td>
<td>Read/Write</td>
<td>32</td>
<td>IP2Bus Address.</td>
</tr>
<tr>
<td>Burst Length Register</td>
<td>0xC</td>
<td>Read/Write</td>
<td>16</td>
<td>Burst length in bytes for burst only.</td>
</tr>
<tr>
<td>Byte Enable Register</td>
<td>0xE</td>
<td>Read/Write</td>
<td>8</td>
<td>IP2Bus Byte Enables for singles only.</td>
</tr>
<tr>
<td>Go Register</td>
<td>0xF</td>
<td>Write</td>
<td>8</td>
<td>Writing 0x0A starts transfer.</td>
</tr>
</tbody>
</table>

The XPS MASTER EXAMPLE core’s master interface is connected to plb_v46_1 and the slave interface is connected to plb_v46_0 as shown in Figure 3. The address for the slave registers are set between 0xA0000000 and 0xA000FFFF which are set in the All tab when right clicking the xps_master_example instance and selecting Configure IP ...

![Figure 3: XPS MASTER EXAMPLE Bus Interfaces](image-url)
Adding the XPS GPIO Core

The XPS GPIO core is beneficial for simulation purposes. For example, a software application is run from BRAM and the software application executes transactions to main memory. Problems can occur if the main memory is not ready to accept transactions. The XPS GPIO and a function that polls the GPIO registers before executing transactions to memory in the software application prevents this scenario. An example is given in the hello_memory software application in case the code is run out of BRAM and master transactions are occurring to DDR2. More information is provided in the “Software Application” section.

In the IP Catalog tab, expand the General Purpose IO tab. Right click on XPS General Purpose IO and select Add IP.

In the System Assembly view, right click on xps_gpio_0 and select Configure IP... .

In the User/Common tab, set GPIO Data Channel Width to 1.

In the User/Channel 1 tab, select Channel 1 is Input Only to TRUE and Channel 1 is Bidirectional to FALSE.

In the System/Addresses tab, set the Base Address to 0x40040000 and High Address to 0x4004FFFF and click OK.

In the System Assembly View/Port tab, expand the ppc440_0, ppc440_mc_ddr2_0, and the xps_gpio_0 tree nodes.

In the ppc440_mc_ddr2_0 tree node, for the mc_miaddrreadytoaccept port, select ppc440_mc_ddr2_0_mc_miaddrreadytoaccept.

In the xps_gpio_0 tree node, for the GPIO_in port, select ppc440_mc_ddr2_0_mc_miaddrreadytoaccept.

In the ppc440_0 tree node, for the MCMIADDRREADYTOACCEPT port, select ppc440_mc_ddr2_0_mc_miaddrreadytoaccept.

Adding ChipScope Cores to System

The first PLB v4.6 IBA is connected to plb_v46_0 where all the slaves in the system are connected. The IBA is set up for a 1-bit trigger in port and is used to monitor bus signals including the read and write data buses.

The second PLB v4.6 IBA is connected to plb_v46_1 where master interfaces are connected. The IBA is set up for a trigger out port and is set up to monitor bus signals including the read and write data buses. The trigger out port on the SPLB IBA allows the MPLB IBA and the MCI ILA to trigger on the trigger in port. This allows the user to see the correlation between a master and slave transaction (to MPLB or to the MIB). More details will be described later in this application note.

The ChipScope ILA is used for the top level ports in the MHS. In this application note, the MCI signals are monitored by ILA0. More details will be described later inside this application note.

Adding ChipScope PLB v4.6 IBAs and ChipScope ILAs To The System

To start adding ChipScope cores to the system, click on Debug→Debug Configuration... .

The Debug Configuration window allows the user to easily add PLB v4.6 IBA and ILAs cores to the system by clicking on Add ChipScope Peripheral... .
The Debug Configuration window is shown in Figure 4.

For the first PLB v4.6 IBA for MPLB, click on Add ChipScope Peripheral...

In the Add New ChipScope Peripheral window, select To monitor PLB v4.6 bus signals (adding PLB IBA), then click OK, as shown in Figure 5.
In the Monitor Bus Signals section, in the Please select the bus that you want to monitor section, select `plb_v46_0`. In the Please select the signals on the bus that you want to monitor section, select **Bus Control Signals**, **Bus Address Signals**, **Bus Read Data Signals**, and **Bus Write Data Signals** as shown in Figure 6.

**Figure 6: Chipscope_IBA_0 Basic Tab**
Select the Advanced Tab. In the User/Trigger In PLB Reset and PLB Error Status section, select the **Enable input trigger port**, set the Trigger input port width to 1 as shown in **Figure 7**, then click **OK**.

![Chipscope_IBA_0 Advanced Tab](image)

**Figure 7**: Chipscope_IBA_0 Advanced Tab
For the second ChipScope PLB v4.6 IBA instance for SPLB0, click on **Debug → Debug Configuration...**. In the EDK project and select **Add ChipScope Peripheral...**.

In the Add New ChipScope Peripheral window, select **To monitor PLB v4.6 bus signals (adding PLB IBA)**, then click **OK**, as shown in **Figure 8**.

![Add New ChipScope Peripheral Window](image)

**Figure 8:** Add New ChipScope Peripheral Window

In the Monitor Bus Signals section, in the **Please select the bus that you want to monitor** section, select **plb_v46_1**. In the **Please select the signals on the bus that you want to monitor** section, select **Bus Control Signals**, **Bus Address Signals**, **Bus Read Data Signals**, and **Bus Write Data Signals** as shown in **Figure 9**.

![Chipscope_IBA_1 Basic Tab](image)

**Figure 9:** Chipscope_IBA_1 Basic Tab
Select the Advanced Tab. In the User/Trigger Out and IBA Storage tab, select **Enable driving port iba_trig_out** as shown in Figure 10, then click **OK**.

![Chipscope_IBA_1 Advanced Tab](image)

**Figure 10:** Chipscope_IBA_1 Advanced Tab
For the MCI ILA instance, click on **Debug → Debug Configuration**... in the EDK project and select **Add ChipScope Peripheral**... .

In the Add New ChipScope Peripheral window, select **To monitor arbitrary system level signals (adding ILA)** and click on **OK**, as shown in **Figure 11**.

![Add New ChipScope Peripheral Window](image)

**Figure 11**: Add New ChipScope Peripheral Window

In the Basic Tab, **Select the clock to be used on monitoring**: to **mc_clk**. The ports for the triggers which are based upon the memory controller are added manually in the Ports tab in the EDK project which is discussed shortly.

Select the Advanced tab. In the Misc tab, deselect **Enable The Trigger Out Signal “trig_out”**.

In the User/Trigger Unit tab, set the following values as shown in **Table 3**. The following set up allows for the widths of the MCI signals and the output from the trigger out on the second PLB v4.6 IBA.

**Table 3**: Advanced User/Trigger Unit Settings

<table>
<thead>
<tr>
<th>N</th>
<th>Number of Match Unit Enabled for Trigger Unit N</th>
<th>Width of Trigger Input for Trigger Unit N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Click **OK**. If a Unconnected Core box appears, click **No** since the ports will be defined in the ports section in the EDK project.

In the System Assembly View/Ports tab, expand the tree nodes for chipscope_plbv46_iba_0, chipscope_plbv46_iba_1, and chipscope ila_0.

For chipscope_plbv46_iba_1, **iba_trig_out** port is connected to **New Connection**.

For chipscope_plbv46_iba_0, **iba_trig_in** port is connected to **chipscope_plbv46_iba_1_iba_trig_out**.

For chipscope ila_0, make the connections shown in **Table 4**.

**Note**: The mc_ prefix is replaced with the name of the bus interface connection between the memory controller and the PPC440. In this case the bus interface is called mc.

**Table 4: chipscope ila_0 Port Connections**

<table>
<thead>
<tr>
<th>PORT</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIG0</td>
<td>ppc440_mc_ddr2_0_mc_miaddrreadytoaccept</td>
</tr>
<tr>
<td>TRIG1</td>
<td>mc_MCMIREADDATAVALID</td>
</tr>
<tr>
<td>TRIG2</td>
<td>mc_MIMCADDRESS</td>
</tr>
<tr>
<td>TRIG3</td>
<td>mc_MCMINESSPRESSIONVALID</td>
</tr>
<tr>
<td>TRIG4</td>
<td>mc_MIMCBANKCONFLICT</td>
</tr>
<tr>
<td>TRIG5</td>
<td>mc_MIMCBYTEENABLE</td>
</tr>
<tr>
<td>TRIG6</td>
<td>mc_MIMCREADNOTWRITE</td>
</tr>
<tr>
<td>TRIG7</td>
<td>mc_MIMCROWCONFLICT</td>
</tr>
<tr>
<td>TRIG8</td>
<td>mc_MIMCWRITEVALID</td>
</tr>
<tr>
<td>TRIG9</td>
<td>chipscope_plbv46_iba_1_iba_trig_out</td>
</tr>
</tbody>
</table>

The connections shown in **Table 4** are shown in **Figure 12**.

**Figure 12: System Assembly View/Ports Tab**

Starting the ChipScope tool will be discussed later.
A generic CDC file for the ChipScope Analyzer project to import is generated in the `implementation/chipscope_plbv46_ila_0_wrapper/` directory after the system is implemented for the ChipScope ILA. Since the CDC file is generic, the modified CDC file for the MCI signals is stored in the CDC/ directory.

The CDC files for the ChipScope Analyzer project to import are generated in the `implementation/chipscope_plbv46_iba_0_wrapper/` and `implementation/chipscope_plbv46_iba_0_wrapper/` after the system is implemented. In this application note, the CDC files are stored in the CDC/ directory.

**General Debugging**

Dumping the executable from the software application is beneficial to monitor the system's behavior. This can be accomplished through simulation or GDB.

The `powerpc-eabi-objdump` command should be used to see the human-readable execution code of the program. For example:

```
powerpc-eabi-objdump -SD name_of_executable.elf > debug.dis.
```

**Note:** This technique is shown in the simulation example.

**Internal Registers**

PPC440 processor internal registers can only be accessed in simulation. Viewing internal registers can help in debugging the system dealing with the core pipeline stages.

For example, the PDCD0 and PDCD1 registers are part of the predecode stage. These registers are helpful in determining if the processor is fetching the correct data from the memory. An example of using these registers is given in this application note.

**PowerPC 440 Processor Core Registers**

The user has the ability to view the PowerPC 440 processor core registers in simulation or in hardware. Registers that can be seen in both environments are User Mode, Timer Control Registers, Supervisor Mode, SPR General Registers, MMU Control Registers, Exception Handling Registers, Cache Control, and Debug Registers.

**Hello_Memory:** Software application that demonstrates the ability of the master to do single and burst writes and reads by setting up the slave registers for the master transaction.

This software application allows master transactions to occur to either the PPC440MC DDR2 or XPS BRAM core depending on a `#define` in the code. By default, master transactions occur to the PPC440MC DDR2 core. This software application is beneficial to see transactions across the crossbar and system. In addition, the linker script is setup to run code out of the DDR2 core.

Included in this software application is a function called `wait_for_done()` which is called before the master starts any transaction. This function polls the GPIO for the PPC440MC DDR2 to finish calibration based upon the `mi_miaddrreadytoaccept` signal from the memory controller. This is useful for prevent master transactions from occurring before the memory controller is ready to accept transactions. To verify this feature, set the linker script to run code out of BRAM and the software application to run master operations to DDR2 which is discussed in the simulation section.

The software application is set up to do a single read and write and a burst read and write by writing to slave registers. For the single read operation, the processor writes a word to the memory location where the master reads. For the burst read operation, the processor writes four words to the memory location where the master starts the burst reads.
Executing the Reference System

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

To execute the system using files in the ready_for_download/ in the project root directory, follow these steps:

1. Change directories to the ready_for_download directory.
2. Use iMPACT to download the bitstream by using the following command:
   `impact -batch xapp1060.cmd`
3. Invoke XMD and connect to the processor by using the following command:
   `xmd -opt xapp1060.opt`
4. Download the executables by using the following command:
   `dow hello_memory.elf`

Note: The above method does not allow for debugging through SDK and ChipScope. This should be used to verify basic system operation.

Executing the Reference System from EDK

To execute the system using EDK, follow these steps:

1. Open system.xmp in EDK.
2. Use Hardware → Generate Bitstream to generate a bitstream for the system.
3. Download the bitstream to the board with Device Configuration → Download Bitstream.
4. Launch XMD with Debug → Launch XMD...
5. Download the executables by using the following command:
   `dow hello_memory.elf`

Note: The above method does not allow for debugging through SDK and ChipScope. This should be used to verify basic system operation.

Running Software Applications In Hardware

Executing Software Applications From SDK

1. Open system.xmp in EDK.
2. Click on Software → Launch Platform Studio SDK.
3. Once SDK starts, in the Application Wizard select Cancel. During this time the software projects and libraries recompile.
4. Select Project → Build Automatically. Before proceeding to the next step, ensure that all software applications are built. During this time the software projects and libraries are recompiled.
5. In the SDK project, click on Device Configuration → Program FPGA. This will program the board with the PPC440 Bootloop.

Executing Hello Memory

1. Expand the hello_memory tree node and expand the Binaries tree node and double click on hello_memory.c.
2. In the hello_memory.c window, set a breakpoint before the single read operation which is on line 102. Left click on the line two times on the certain line number as shown in Figure 13.

Follow the same process for the single write operation line 120, burst read operation line 141, and burst write operation line 157.

3. In the Binaries Tree node, right click on hello_memory.elf and select Debug As → 1 Debug on Hardware.

Starting ChipScope Analyzer

Before starting Analyzer, make sure the previous steps in Executing Software Applications From SDK section have been followed.

1. Open ChipScope Analyzer in ChipScope Pro → Analyzer.
2. Click on the Open Cable/Search JTAG Chain icon. Then click OK in the ChipScope Pro Analyzer window.
3. Select File → Open Project... and select No to Do you want to save the changes you made to project ‘new project’ ?. Then select the ppc440_debug.cpj file in the cpj/ directory. The ChipScope project imported cdc files from the cdc/ directory. Refer to the ChipScope documentation in importing CDC files.
4. Table 5 shows the relation of the ChipScope icons to the buses in the system.

| Unit:0 MyILA0 | MPLB |
| Unit:1 MyILA1 | SPLB0 |
| Unit:2 MyILA2 | MCI |

Table 5: ChipScope Icons
5. In the Project tab, select Trigger Setup for Unit:1 MyILA1 and Unit:2 MyILA2 and select **Trigger Setup → Run** in both cases.

6. In the ChipScope project, Unit:1 MyILA1 is setup to trigger on the PLB Address (M2:TRIG2) of 0x0000F000. Unit:0 MyILA0 and Unit:2 MyILA2 trigger when Unit:1 MyILA1 trigger condition is met. Single and burst reads occur at the address of 0x0000F000 and single and burst writes occurs at address 0x0000F100, The trigger address is changed in future steps to demonstrate writes.

---

**Figure 14:** ChipScope ILA1 Trigger
7. Select the Resume button in the SDK project as shown in Figure 15. The waveform in Figure 16 shows a single read. The Unit:2 MyILA2 waveform shows memory transactions and the single read during the master transaction.

![SDK Project Debug](image1)

**Figure 15:** SDK Project Debug

![ChipScope ILA1 Single Read Waveform](image2)

**Figure 16:** ChipScope ILA1 Single Read Waveform

8. In the ChipScope project, change the M2:TRIG2 to 0000_F100 for UNIT:1 MyILA1. In the Project tab, select Trigger Setup for Unit:1 MyILA1 and Unit:2 MyILA2 and select **Trigger Setup → Run** in both cases.

9. In the Debug SDK window, select the Resume button. The breakpoint is setup before the single write.
10. In the ChipScope project, the ChipScope core has triggered based upon the resume button (actions by the software). The waveform shows a single write as shown in Figure 17. The Unit:2 MyILA2 waveform shows memory transactions and the single write during the master transaction.

![Figure 17: ChipScope ILA1 Single Write Waveform](image)

11. In the ChipScope project, change the M2:TRIG2 to 0000_F000 for UNIT:1 MyILA1. In the Project tab, select Trigger Setup for Unit:1 MyILA1 and Unit:2 MyILA2 and select Trigger Setup→Run in both cases.

12. In the Debug SDK window, select the Resume button. The breakpoint is setup before the burst read.
13. In the ChipScope project, the ChipScope core has triggered based upon the resume button. The waveform shows a burst read as shown in Figure 18. The Unit:2 MyILA2 waveform shows memory transactions and the burst read during the master transaction.

![Waveform - XILINX MyILA1 Burst Read Waveform](image_url)

*Figure 18: ChipScope ILA1 Burst Read Waveform*

14. In the ChipScope project, change the M2:TRIG2 to 0000_F100 for UNIT:1 MyILA1. In the Project tab, select Trigger Setup for Unit:1 MyILA1 and Unit:2 MyILA2 and select Trigger Setup→Run in both cases.

15. In the Debug SDK window, select the Resume button. The breakpoint is setup before the burst write.
16. In the ChipScope project, the ChipScope core has triggered based upon the resume button. The waveform shows a burst write as shown in Figure 19. The Unit:2 MyILA2 waveform shows memory transactions and the burst write during the master transaction.

![Waveform](image)

**Figure 19:** ChipScope ILA1 Burst Write Waveform

17. Click on the **Stop** button to Terminate the debug session in the SDK project.

**Master Operations to XPS BRAM**

1. The above steps can be modified to monitor master operations to XPS BRAM. In the software application remove the `#define main_memory` line and rebuild and debug the software application in SDK.

*Note:* The software application will still be run from external memory.

2. In the Project tab, select Trigger Setup for Unit:0 MyILA0 and Unit:1 MyILA1 and select **Trigger Setup→Run** in both cases.

3. In the ChipScope project, for Unit:1 MyILA, set the trigger for PLB Address (M2:TRIG2) to FFFFF000. Single and burst reads occur at the address of 0xFFFFF000 and single and burst writes occur at address 0xFFFFF100.

4. Follow the same set of procedures as the previous section.

**Viewing DCR Registers for the PPC440 Processor Block**

For the setup of the MIB, the MI_CONTROL register is set to 0xF810008F. In addition, the MI_ROWCONFLICT_MASK and MI_BANKCONFLICT_MASK is set to 0x003FFE00 and 0x00C00000, respectively. Refer to DDR2 Memory Controller for PowerPC 440 Processor Product Specification in setting these parameters based upon the system.
These registers are DCR registers in the PPC440 processor block. In this system, the PPC440 wrapper is set up for these registers and are shown in Figure 20. The next steps verify the values of these registers in the system.

1. In SDK, select Xilinx Tools → Launch EDK Shell...
2. In the EDK Shell invoke XMD by entering in xmd.
3. In XMD, enter in connect ppc hw.

**Note:** By default DCR addresses are mapped to look like PLB addresses when connecting to the processor as shown in the following figure. In this case 0x78020000 is the base address and 0x78020fff is the high address.
4. Because the DCR registers are emulated PLB address ranges, the offset will be a multiple of 4. Table 6 shows the relation with the MIB DCR registers in this example.

**Table 6: DCR to PLB Address Mapping**

<table>
<thead>
<tr>
<th>Register</th>
<th>DCR Address</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI_Control</td>
<td>0x10</td>
<td>0x78020040</td>
</tr>
<tr>
<td>MI_ROWCONFLICT_MASK</td>
<td>0x11</td>
<td>0x78020044</td>
</tr>
<tr>
<td>MI_BANKCONFLICT_MASK</td>
<td>0x12</td>
<td>0x78020048</td>
</tr>
</tbody>
</table>

5. In XMD, enter in `mrd 0x78020040 3 w`. The following results should be seen which verify that the register values entered in the PPC440 wrapper in EDK.

![Figure 21: XMD Window](image)

![Figure 22: XMD DCR Reads](image)
Executing the Simulation

Setting the Simulation Environment

Compiling Libraries

1. Compile Unisim, Simprim, XilinxCoreLib, SecureIP, Smartmodel libraries and EDK libraries. Open XPS and select Cancel on the Create new or open existing project dialog box. Select Simulation → Compile Simulation Libraries...

Note: The hard blocks for this application note use SecureIP libraries and not Smartmodels libraries.

2. Follow the instructions to compile with the ModelSim simulator and select Both VHDL and Verilog. Both VHDL and Verilog simulation support are required for this reference system.

3. Set the options to compile the ISE libraries as shown in Figure 23.

![Figure 23: Compiling ISE Libraries](image_url)
4. Set the options to compile the EDK libraries as shown in Figure 24.

![Figure 24: Compiling EDK Libraries](image)

5. On the following dialog box, select **Do not compile deprecated library elements** and then select **Compile >** to start the compilation process.

   **Note:** This process might take a few hours depending on the machine being used.

6. Close XPS.

**Setting Environmental Variables and ModelSim in Windows**

1. Go to Start→Control Panel→System→Advanced and click on Environment Variables.
2. Set the following variables if necessary:
   a. MODELSIM <path to modelsim.ini>\modelsim.ini
3. Ensure that all of the EDK libraries and ISE libraries are mapped in the modelsim.ini file.
   a. If the libraries are not mapped correctly, copy the library mappings from the modelsim.ini file found in the compile EDK_Lib directory to the current modelsim.ini file above the [Vcom] section.

   An example modelsim.ini file is provided in the windows_setup directory.

**Setting Environmental Variables and ModelSim in Linux**

1. Set the following variables:
   a. setenv MODELSIM <path to modelsim.ini script>/modelsim.ini
2. Ensure that the EDK libraries and ISE libraries are mapped in the `modelsim.ini` file.
   a. If the libraries are not mapped correctly, copy the library mappings from the `modelsim.ini` file compile EDK_Lib directory of the to the current `modelsim.ini` file above the `[Vcom]` section.

An example `modelsim.ini` file is provided in the `linux_setup` directory.

**Simulation Directories and Files**

Two simulation directories are used for Linux and Windows. The directories are `sim_lin/` and `sim_win/`.

**Windows Simulation Files**

The `sim_win/` directory contains the following pertinent files and directories:

- `256Mb_ddr2/`: Directory that contains Micron DDR2 memory model.
- `mem/`: Directory that contains the perl scripts to create the data files for the DDR2 memory model.
- `system_tb_v.do`: ModelSim compile script for Verilog simulation.
- `system_tb.v`: Verilog testbench for system.
- `uart_rcvr.v`: For simulated RS232 terminal.
- `uart_rcvr_wrapper.v`: For simulated RS232 terminal.

**Note:** The simulated RS232 terminal will not be displayed in Windows. Output from the software application is monitored in the ModelSim terminal while the simulation is running.

**Linux Simulation Files**

The `sim_lin/` directory contains the following pertinent files and directories:

- `256Mb_ddr2/`: Directory that contains Micron DDR2 memory model.
- `mem/`: Directory that contains the perl scripts to create the data files for the DDR2 memory model.
- `system_tb_v.do`: ModelSim compile script for Verilog simulation.
- `system_tb.v`: Verilog testbench for system.
- `uart_rcvr.v`: For simulated RS232 terminal.
- `uart_rcvr_wrapper.v`: For simulated RS232 terminal.

---

**Executing the Simulation from EDK**

To generate and run the system simulation from EDK, follow these steps:

1. Open `system.xmp` in EDK.

   **Note:** For faster calibration of the memory controller, change the `PARAMETER C_SIM_ONLY = 0` under the PPC440MC DDR2 instance to `PARAMETER C_SIM_ONLY = 1`. For the following sections, the system is set to `PARAMETER C_SIM_ONLY = 1`.

2. Click on **Software** → **Launch Platform Studio SDK**.

3. Once SDK starts, in the Application Wizard select **Cancel**.

4. Select **Project** → **Build Automatically**. Before proceeding to the next step, ensure that all software applications are built. During this time the software projects and libraries are recompiled.

5. In the Applications/Software Projects tab, **Make Project Active** and **Mark to Initialize BRAMs** for the `hello_memory` software applications. This step is required for all applications regardless if they are run from either XPS BRAM or PPC440MC DDR2 so that
the ELF is converted to files that can be used for loading the external memory model. All other software applications projects are inactive.

**Note:** Only have ONE active software application in EDK.

6. Use **Simulation→Generate Simulation HDL Files**.

**Note:** Creating the simulation files will take a long time since ChipScope cores are in the system.

7. Launch ModelSim with **Simulation→Launch HDL Simulator**.

8. In ModelSim, execute the compile script for Verilog by using one of the following command depending on the operating system:
   
   ```
   do ../../sim_win/system_tb_v.do
   or
   do ../../sim_lin/system_tb_v.do
   ```

### Running Hello Memory

1. In ModelSim, execute the run command with the specified run time for the hello_memory software application.

   ```
   run 70 us
   ```

2. Launch an EDK shell. Change directories to the `<project>/SDK_projects/hello_memory/Debug` directory. Execute the following command:

   ```
   powerpc-eabi-objdump -SD hello_memory.elf > debug.dis.
   ```

Open the debug.dis with a text editor. This will be analyzed in the next section.

### Analysis of Simulation

1. In the Modelsim wave window, scroll down to the GPR0 signal under the PowerPC Processor Standard Registers divider.

2. At 56.165 us, notice how the registers are the same as in Figure 25. This time demonstrates the system before the hardware breakpoint.

![PowerPC Processor GPR Registers](X1060_25_06260)

---

**Figure 25:** PowerPC Processor GPR Registers
3. During the same time, scroll down to the PowerPC Processor Internal Registers. For the PDCD0ADDR and PDCD1ADDR which are at 0x00000280 and 0x00000284. Notice how the values for PDCD0DATA and PDCD1DATA are 0x3f60a000 and 0x93810010 which are fetched from memory. In the debug.dis file, scroll down to the 0x00000280 and 0x00000284 addresses. The following is displayed to verify the right data was fetched from the addresses in main memory.

\[
\begin{align*}
280: & \text{3f 60 a0 00 lis r27,-24576} \\
284: & \text{93 81 00 10 stw r28,16(r1)}
\end{align*}
\]

![PowerPC Processor Internal Registers](image1)

**Figure 26:** PowerPC Processor Internal Registers

4. Scroll down to the plb_v46_1 divider. The following waveforms are similar to ChipScope analyzer traces in the ChipScope section.

5. At 56.165 us, a single read occurs to the main memory as shown in **Figure 27**.

![Single Read to PPC440MC DDR2](image2)

**Figure 27:** Single Read to PPC440MC DDR2

6. At 58.658us, a single write occurs to the main memory as shown in **Figure 28**.

![Single Write to PPC440MC DDR2](image3)

**Figure 28:** Single Write to PPC440MC DDR2
7. At 62.060 us, a burst read occurs to the main memory as shown in Figure 29.

8. At 64.175 us, a burst write occurs to the main memory as shown in Figure 30.

Executing Master Operations to DDR2 While Running Code Out of BRAM

1. In the hello_memory.c file, ensure the following #define is uncommented:
   #define main_memory

2. In SDK, right click on hello_memory and select Generate Linker Script... . Set all code segments to xps_bram_1 and select Generate.

3. Follow the steps in the “Executing the Simulation from EDK” section to set up the simulation files and the simulation.

4. Run the simulation for 65 us.

5. In the waveform, scroll down to the ddr2_init divider.
6. Notice at time ~12.170 us that reads are occurring to the XPS GPIO. The software application is polling the GPIO registers to see if the memory controller is ready to accept transactions. Scroll up to the plb_v46_1 divider. Once the memory controller is ready to accept transactions which is at 60.615 us, master transactions are executed as in Figure 31.

![Figure 31: plb_v46_1 Master Transactions](image)

**References**

1. **UG200** *Embedded Processor Block in Virtex-5 FPGAs Reference Guide*
2. **DS567** *DDR2 Memory Controller for PowerPC 440 Processors Product Specification*
3. **DS621** *PPC440 Virtex-5 (Wrapper) Product Specification*

**Revision History**

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>9/11/08</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
<tr>
<td>9/26/08</td>
<td>1.1</td>
<td>Made minor content changes; updated Figure 1.</td>
</tr>
</tbody>
</table>

**Notice of Disclaimer**

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.**