Introduction to Zynq Hardware

Lab 8

Hardware Debugging Zynq Designs

November 2013
Version 03

Lab 8 Overview

Often times when hardware engineers create new IP they want to test it, not only through simulation but also in hardware. But with embedded designs like Zynq, it requires software to be written to test the IP. In Vivado 2013.3 Xilinx has introduced a new LogiCORE™ IP JTAG-AXI core, which we added in the last lab. This core is a customizable core that can generate AXI transactions and drive AXI signals internal to the AP SoC at run-time. We'll use this core to test our IP. To run the JTAG-AXI core, we'll utilize Vivado Logic Analyzer. Lastly, we'll add a prebuilt software application to validate our test. The software application will include an Interrupt Service Routine (ISR) that processes interrupts from our custom IP.

Lab 8 Objectives

When you have completed Lab 8, you will know how to do the following:

- Perform run-time interactions with IP cores
- Run software that handles PL-generated interrupts and utilizes an Interrupt Service Routine.

Experiment 1: Import Software Application

This experiment shows how to import a prebuilt SDK Workspace.

Experiment 1 General Instruction:

Launch Lab 8 SDK Workspace. Import existing software project.

Experiment 1 Step-by-Step Instructions:

1. <Optional> If you did not complete Lab 7 or wish to start with a clean copy, delete the `ZynqDesign, IP and SDK_Workspace` folders in the `ZynqHW/2013_3` folder. Then unzip **`Solutions\ZynqHW_Lab7_Solution.zip`** to the `2013_3` folder. If you have 7-zip installed, you can do this by right-clicking and dragging **`ZynqHW_Lab7_Solution.zip`** to the `2013_3` folder. Select **7-Zip → Extract Here**.

2. Open Vivado and export the design to **SDK**. Create a new SDK workspace called **SDK_PWM**. (Click **OK** to create new directory).

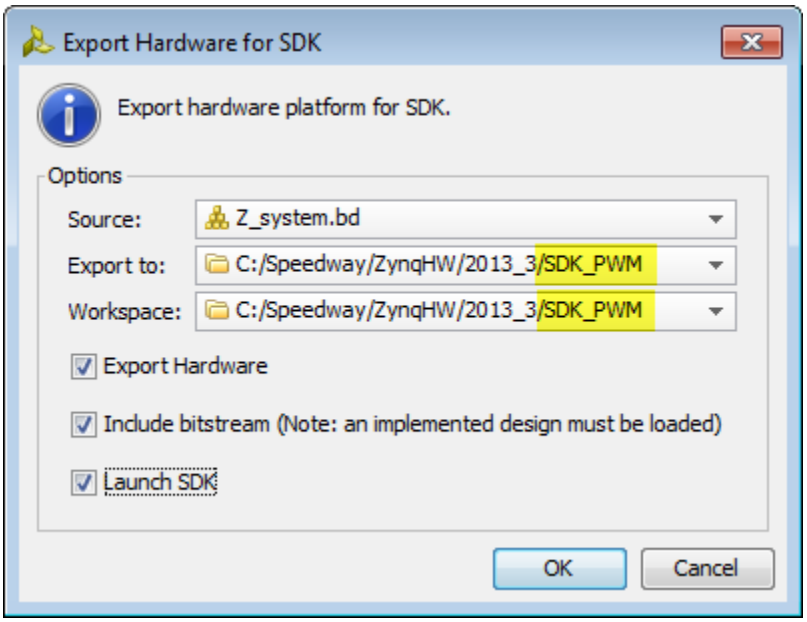

Figure 1 – Export to SDK

3. When SDK opens, create a new BSP. (**File → New → Board Support Package**). Call it **customIP_bsp_01**. Click **Finish**.

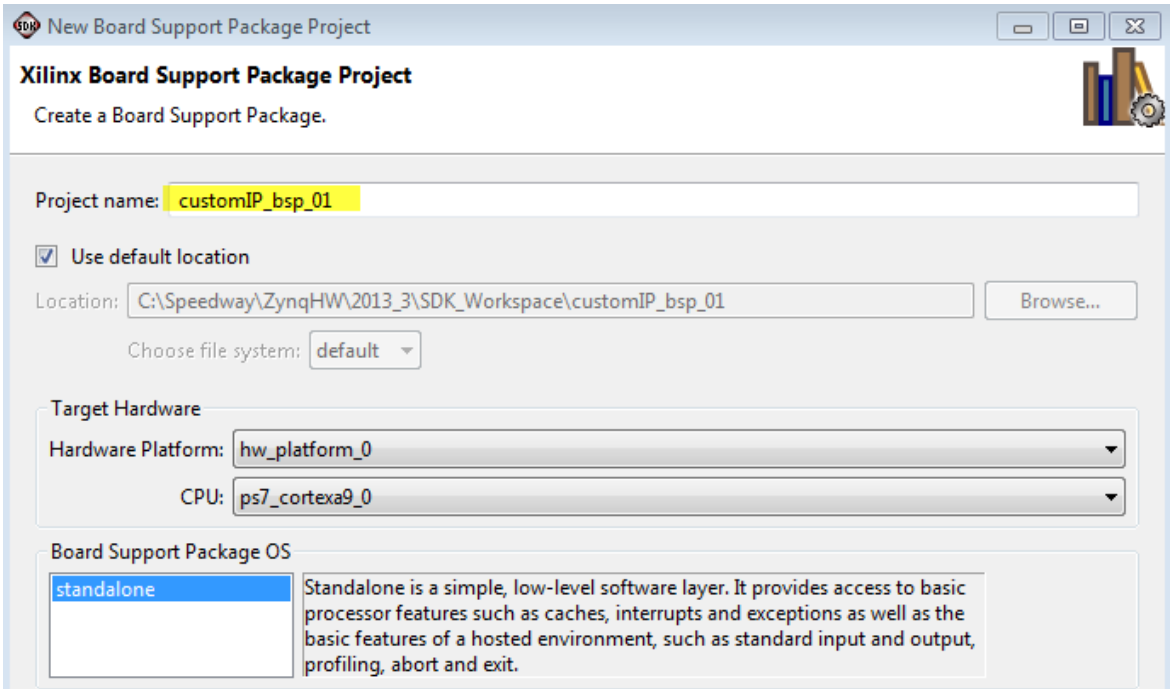

Figure 2 - Create New BSP

4. Click **OK** to accept BSP Settings.

5. When the BSP has finished being built. Select **File → Import**.

6. Select **General → Existing Projects into Workspace**.

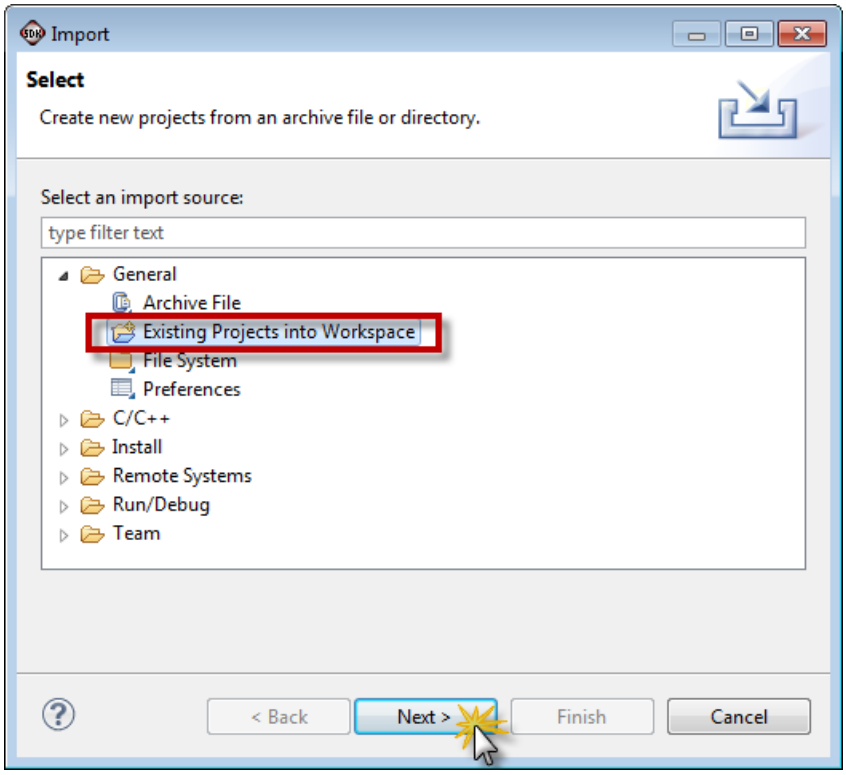

Figure 3 - Import Existing Project

7. Select the **Archive File** radio button and **Browse** to:

   C:\Speedway\ZynqHW\2013_3\Support_documents

   Select **ZynqHW_Lab8_Workspace.zip**. Make sure the project checkbox is checked for LED_Dimmer_Int.

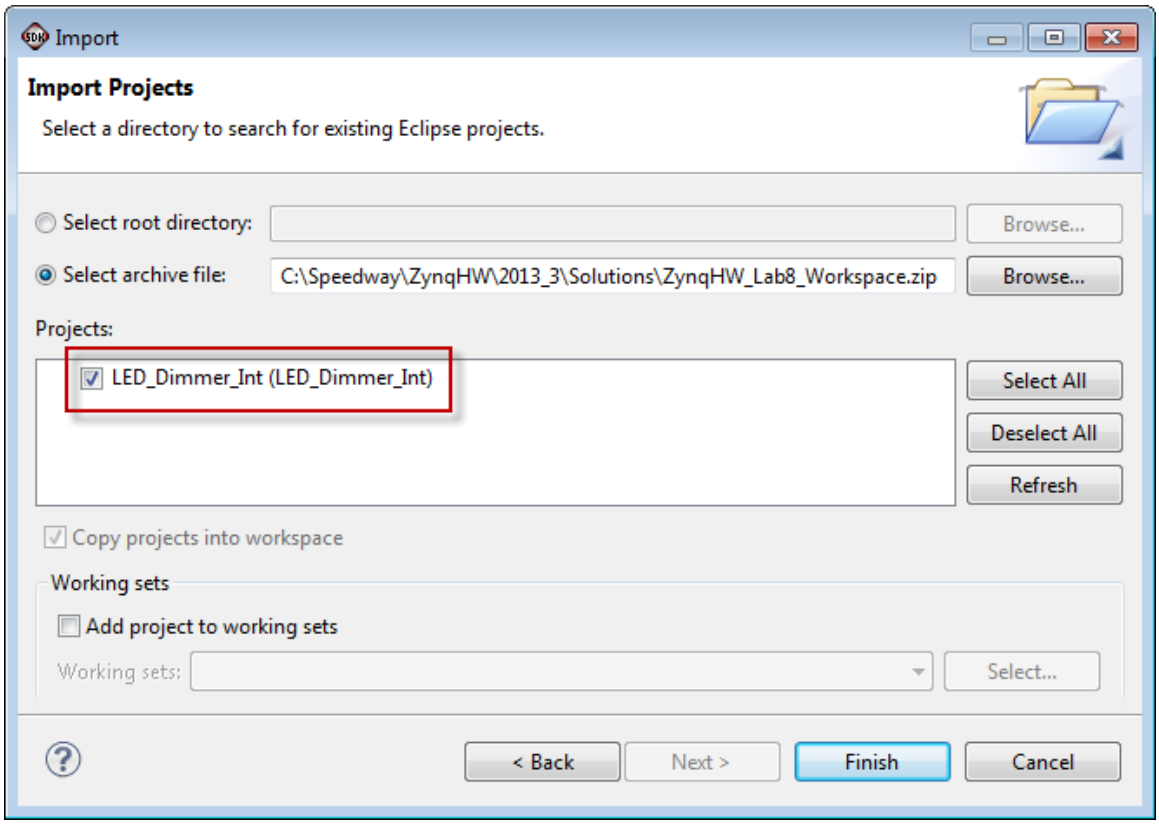

Figure 4 - Import SDK Workspace Projects

8. Click **Finish**.

The application will attempt to build but will stop due to an error. This is expected as we need to assign this application to our new BSP.

```
'Invoking: ARM gcc compiler'
arm-xilinx-eabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -I../../standalone_bsp_0/ps7_cortexa9_0/include
-MMD -MP -MF"src/main.d" -MT"src/main.d" -o "src/main.o" "../src/main.c"
../src/main.c:46:25: fatal error: xparameters.h: No such file or directory
compilation terminated.
make: *** [src/main.o] Error 1

15:34:37 Build Finished (took 450ms)
```

Figure 5 - Application Error - Missing BSP (xparameters.h)

In the SDK workspace you will see a BSP and the new pre-compiled software applications, LED_Dimmer_Int.

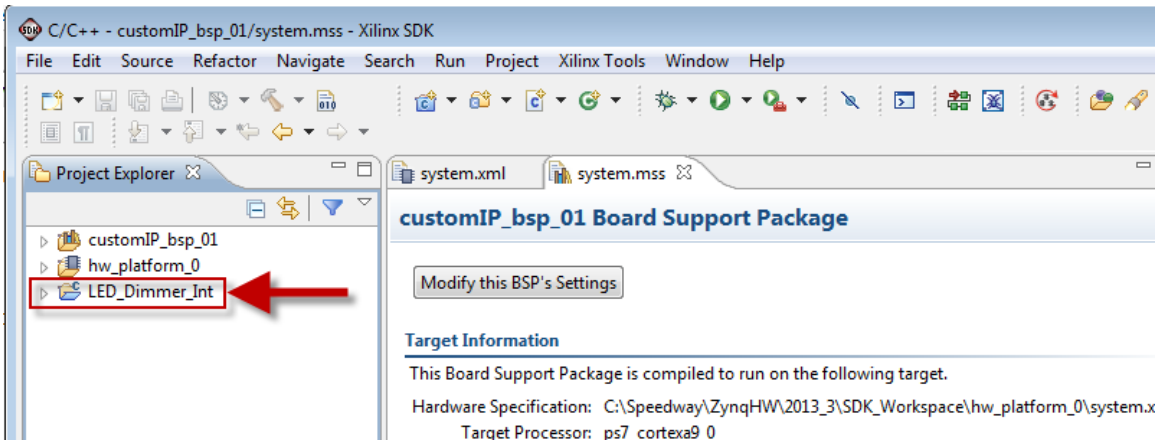

Figure 6 - SDK Workspace

9. Right-click on the imported application and select **Change Referenced BSP**.

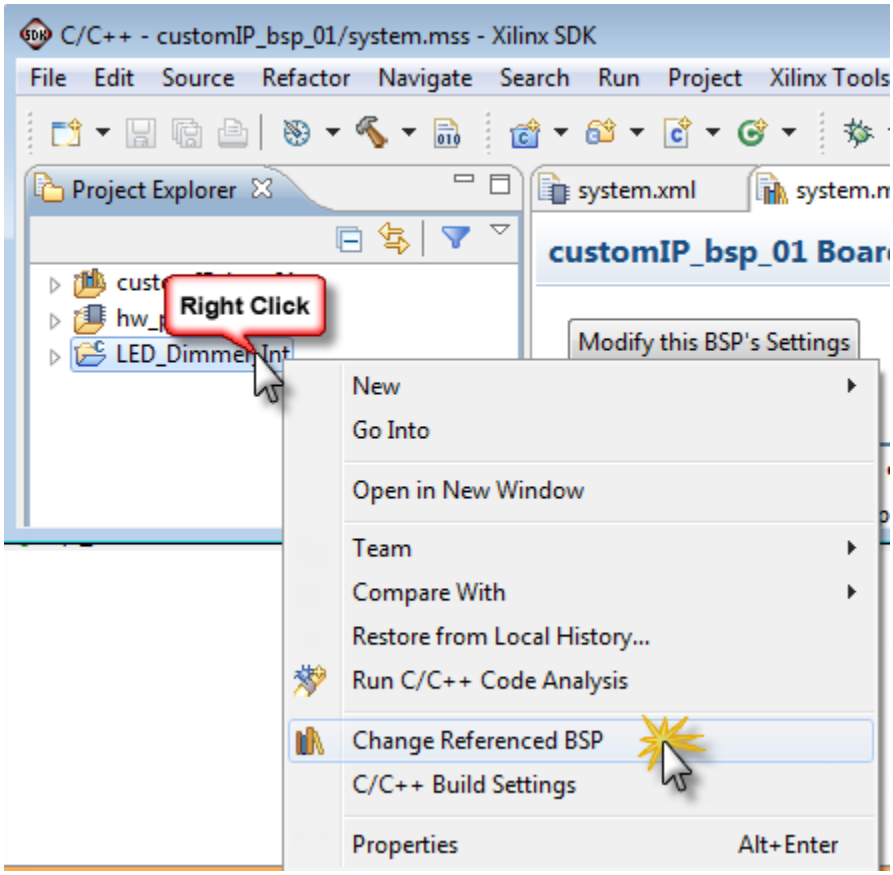

Figure 7 - Change Referenced BSP

10. Several BSP's may be shown, select the one we just created, **customIP_bsp_01**. Click **OK**.

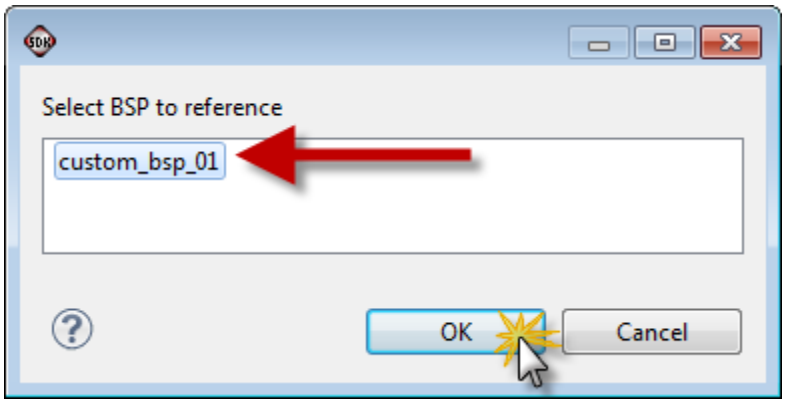

Figure 8 - Select BSP

11. The application will build, when done, open the source code, **main.c**, for LED_Dimmer_Int. Examine the code. Look at the ISR (PWMIsr) and Interrupt System (SetupInterruptSystem) functions.

Questions:

Answer the following questions:

- *What function is called when the Interrupt is detected?*

_____

- *What does the PWMIsr function do?*

_____

- *In the PWMIsr function, what does it do to the brightness value?*

_____

- *What is the INTC_PWM_INTERRUPT_ID? Where did this number come from?*

_____

_____

Experiment 2: Vivado Hardware Analyzer

This experiment shows how to open and setup the Vivado Hardware Analyzer. We'll capture hardware events and trigger on exceptions to validate the hardware design.

Experiment 2 General Instruction:

Program the FPGA. Download SW application to the board. Open Vivado Hardware Analyzer. Trigger events in the Analyzer while running software.

Experiment 2 Step-by-Step Instructions:

1. **Program** the FPGA.

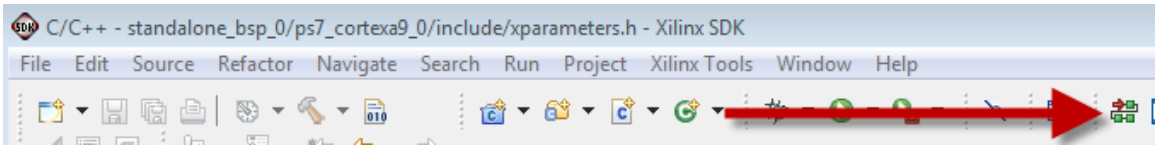

Figure 9 - Program FPGA

2. Select the default **bitstream,** click **Program**.
3. **Open a terminal** program and connect to the evaluation board's COM port. (If a terminal was left open from a previous lab, close it and reopen it.)
4. **Run As → Launch on Hardware** the **LED_Dimmer_Int** application.
5. In the terminal, enter **numbers 0-9** to see if the code is working. Then press any **letter** on the keyboard, this should create the exception.
6. In Vivado, select **Open Hardware Manager** from the Flow Navigator.

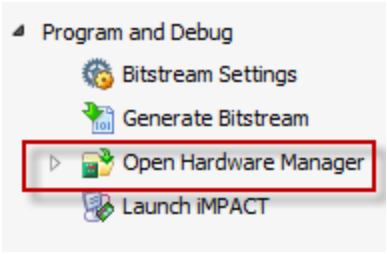

Figure 10 - Open Hardware Manager

7. At the top of the Hardware Manager Window, select **Open a new hardware target**.

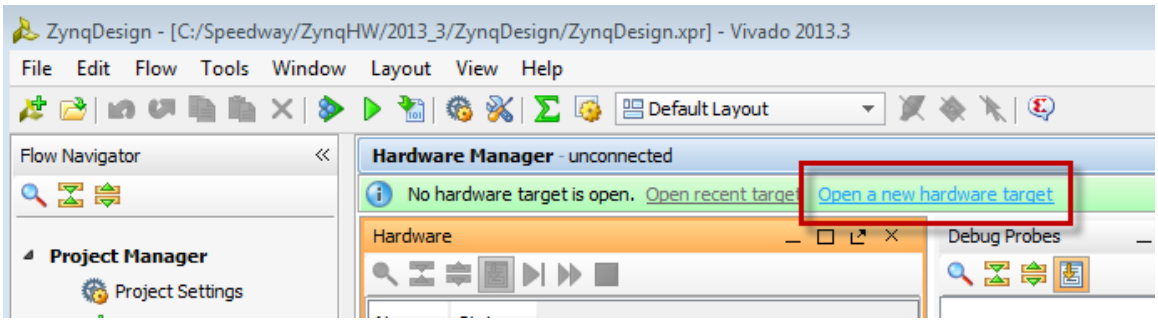

Figure 11 - Open new hardware target

8. When the Open Hardware Target window opens, click **Next >**.

9. Click **Next >** to accept the Server name.

10. Click **Next >** to Select the Hardware Target. This is only one choice.

11. Click **Next >** to accept the Hardware Target properties.

12. Click **Finish**.

13. The Hardware Manager is now ready. Select the **ILA core**.

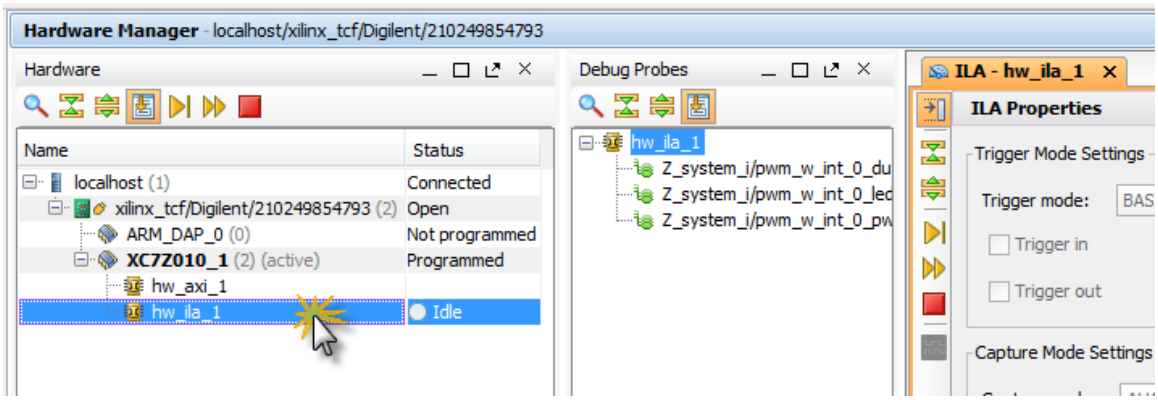

Figure 12 - Select ILA Core

14. Click the **Run Trigger Immedate for this ILA Core** button from the shortcut bar.

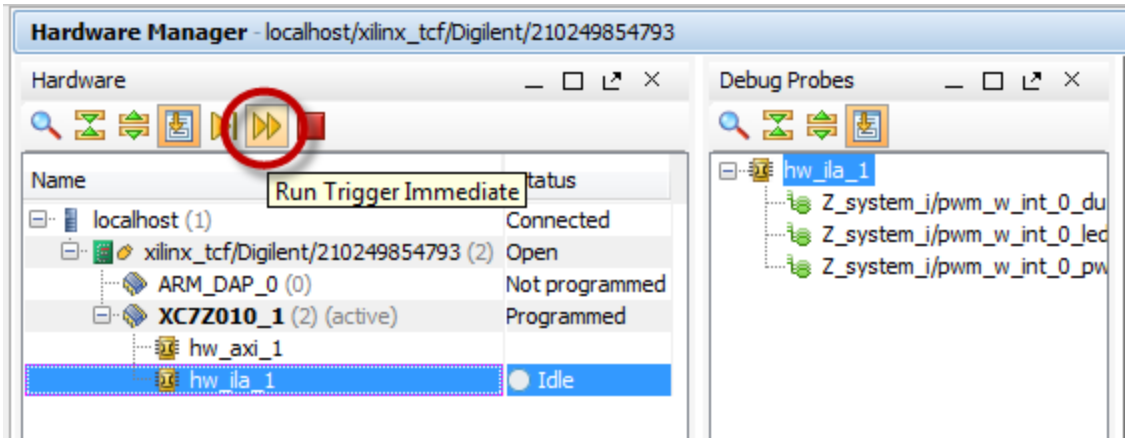

Figure 13 - Run Trigger Immediate

15. Depending on what value was entered last, you should see this in the waveform view. To help, change the **radix** of PWM_Counter and DutyCycle to Unsigned Decimal. (Hint: Zoom in to see the values as shown below.)

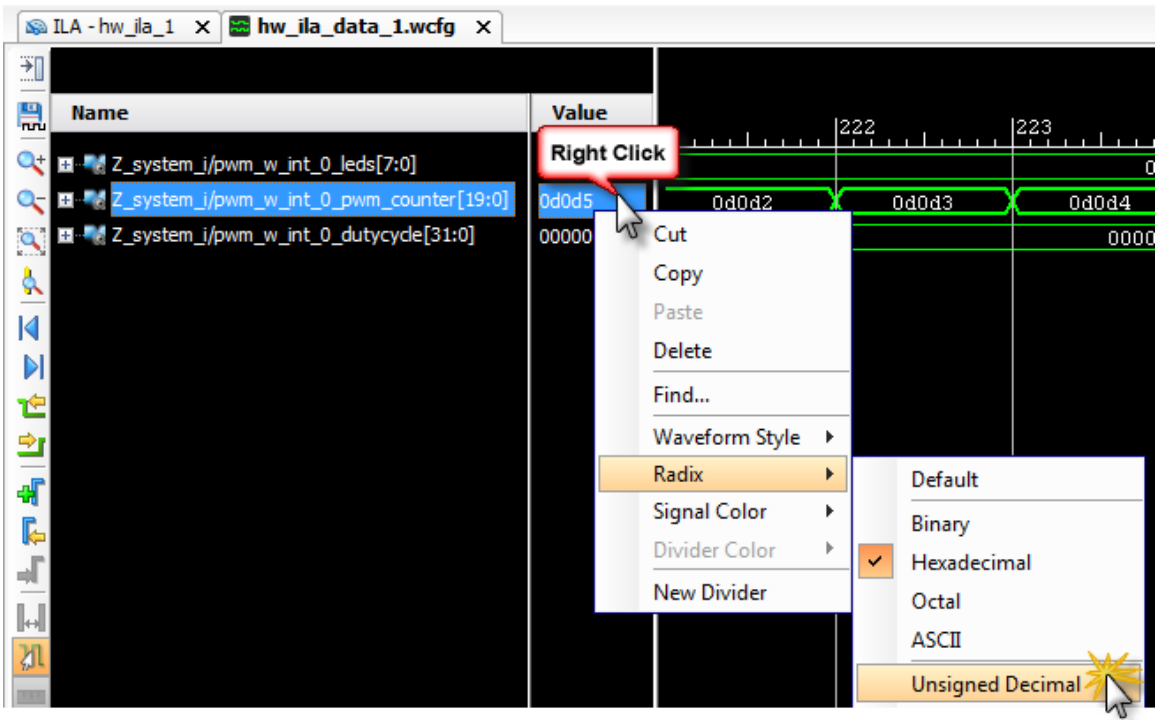

Figure 14 - Waveform View

16. Also, if the **Interrupt_out** signal is not in the waveform viewer, right-click it and select **Add Probes to Waveform**.

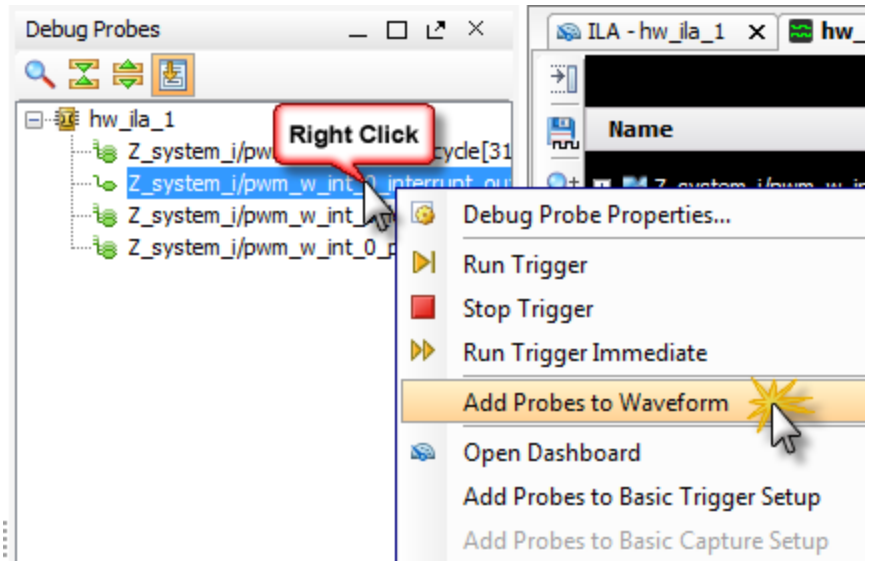

Figure 15 - Add Probes to Waveform

17. In the terminal enter a legal value such as **4**. **Trigger Immediate** again.

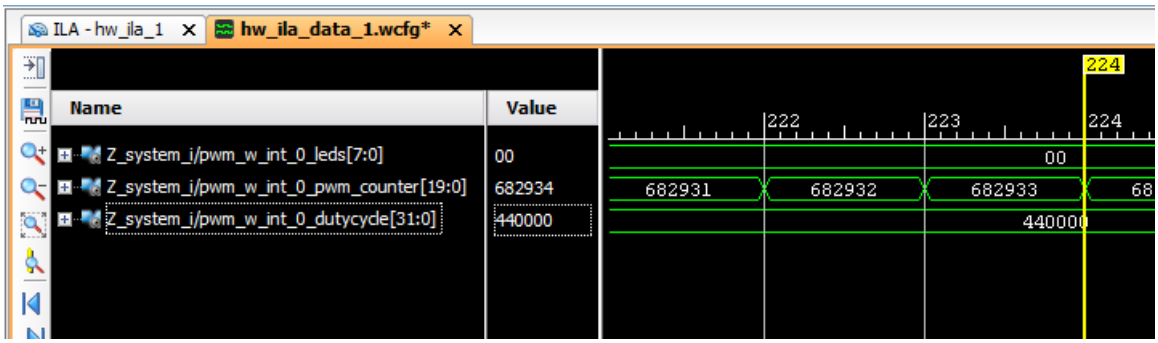

Figure 16 - Waveform View, Brightness level 4

Notice, the DutyCycle is set to 440,000. That is accurate as the software in the PS multiplies the number we enter by 110,000. Remember, the period parameter sets the counter depth of the PWM counter. The period default is 20, thus the counter counts to roughly 1 million. Thus any number over 1 million should create an exception and thus an interrupt. To capture an interrupt happening, we'll have to create a trigger in the ILA.

18. Select the ILA – hw_ila_1 tab at the top of the Hardware Manager window.

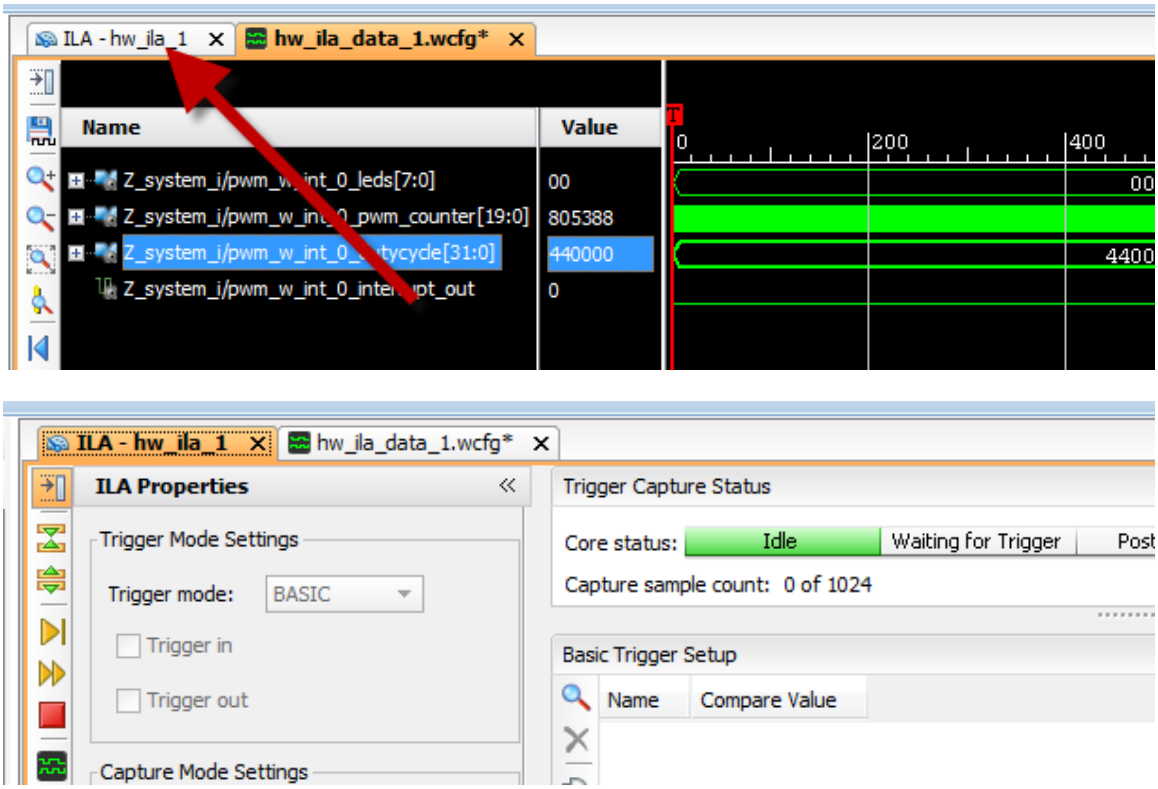

Figure 17 - Switch from Waveform View to ILA Properties

19. Right-click on the **Interrupt_out** signal and select **Add Probes to Basic Trigger Setup.**

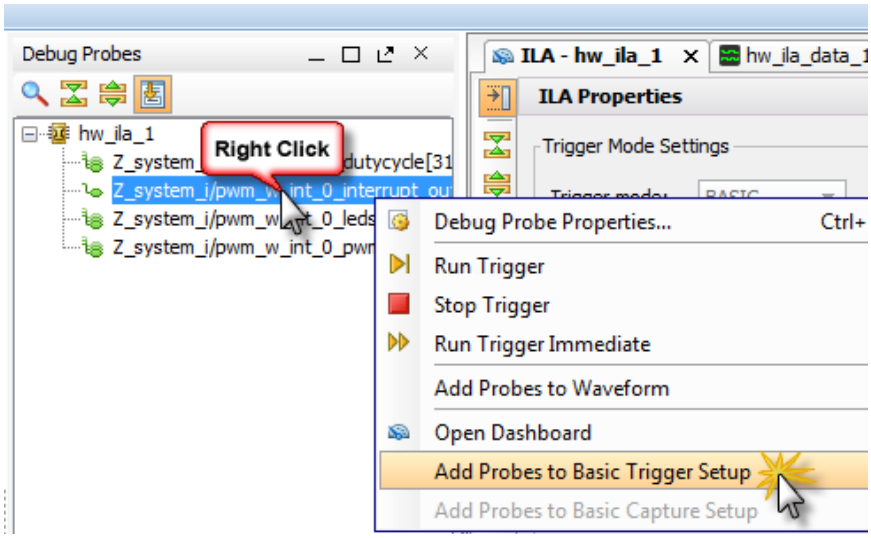

Figure 18 - Add Probes to Trigger

20. In the *Basic Trigger* Setup pane, left-click on the *Compare Value* for **Interrupt_out** and change the value to **R**, for rising edge. Then click **OK**.

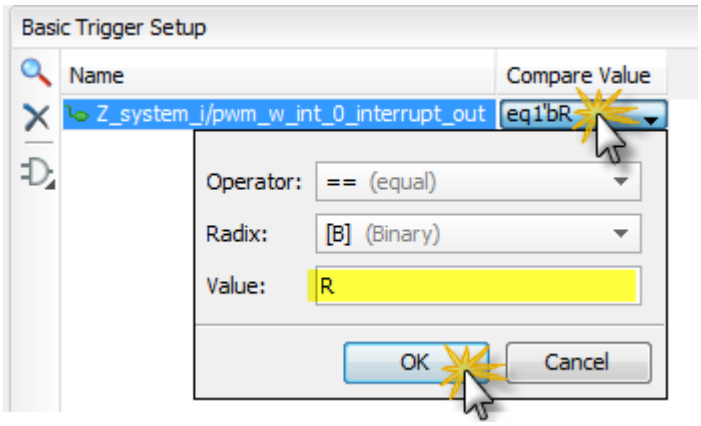

Figure 19 - Trigger Setup

21. Change the *Trigger Position* to **500**, then hit **ENTER**. This will set the trigger to the middle of our *capture depth*.

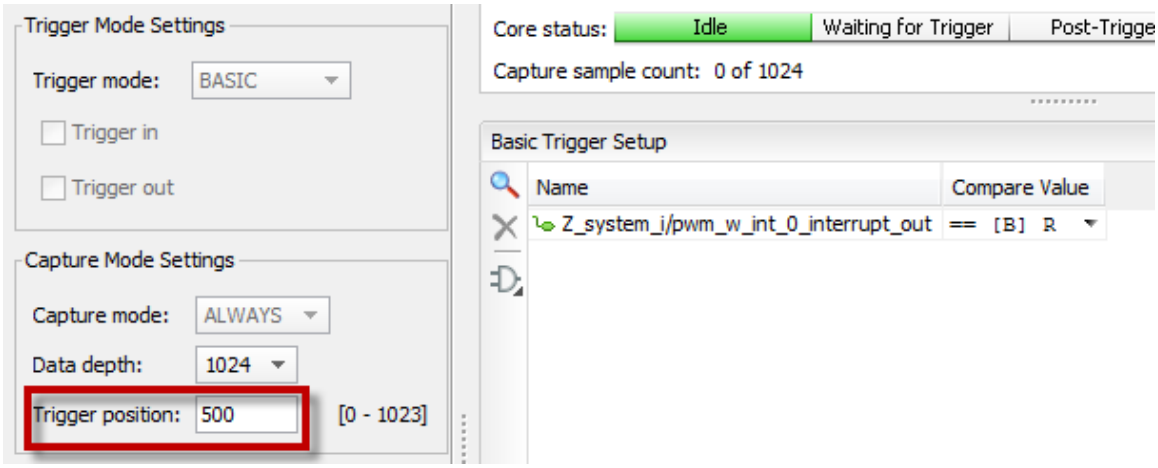

Figure 20 - Set Trigger Depth

22. Click the **Run trigger for this ILA core** button from the vertical shortcut bar to arm the core.

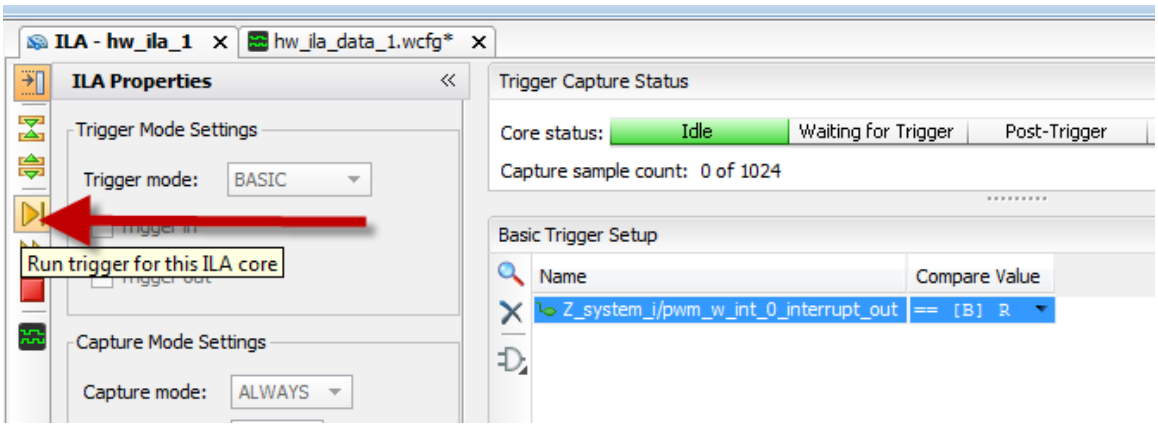

Figure 21 - Run trigger

23. In the Terminal, enter several **numbers** to see if the trigger occurs. If not, you should still see the Trigger Capture Status as Waiting for Trigger.

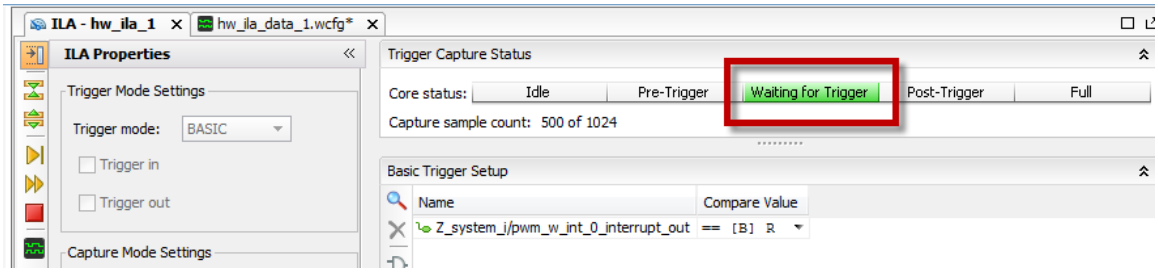

Figure 22 - Waiting for Trigger

24. Now enter a **letter** into the Terminal.

25. The trigger status should change from Full, then to idle. Switch back to the waveform view. The Trigger, and interrupt, should have occurred exactly when the DutyCycle exceeded our PWM counter maximum.

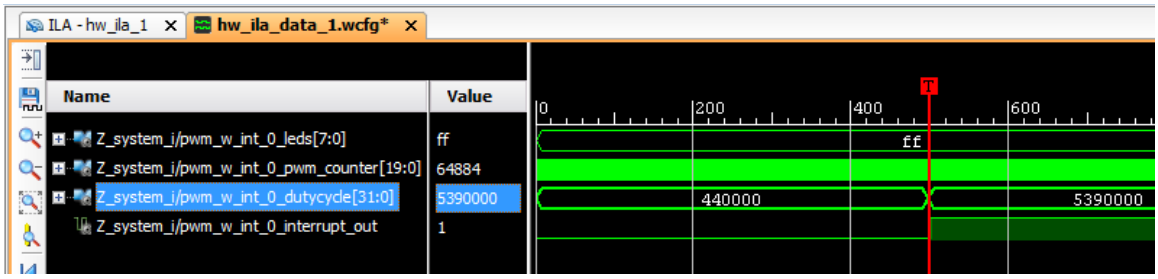

Figure 23 - Waveform showing Trigger event

Questions:

Answer the following questions:

- *Can you detect how long it takes the processor to handle the interrupt?*

_____

Experiment 3: Run-time Interactions with the AXI

Again, it's often very likely that hardware engineers will not have software code available when they are ready to test their IP. For this reason, Xilinx has created the new LogiCORE™ IP JTAG-AXI core. This experiment shows how to interact with the JTAG-AXI core. We'll perform AXI transactions through the AXI JTAG Master via TCL commands to perform run-time interactions to test the IP cores.

Currently the software is running from the PS and sending commands to our peripherals through the AXI Interconnect.

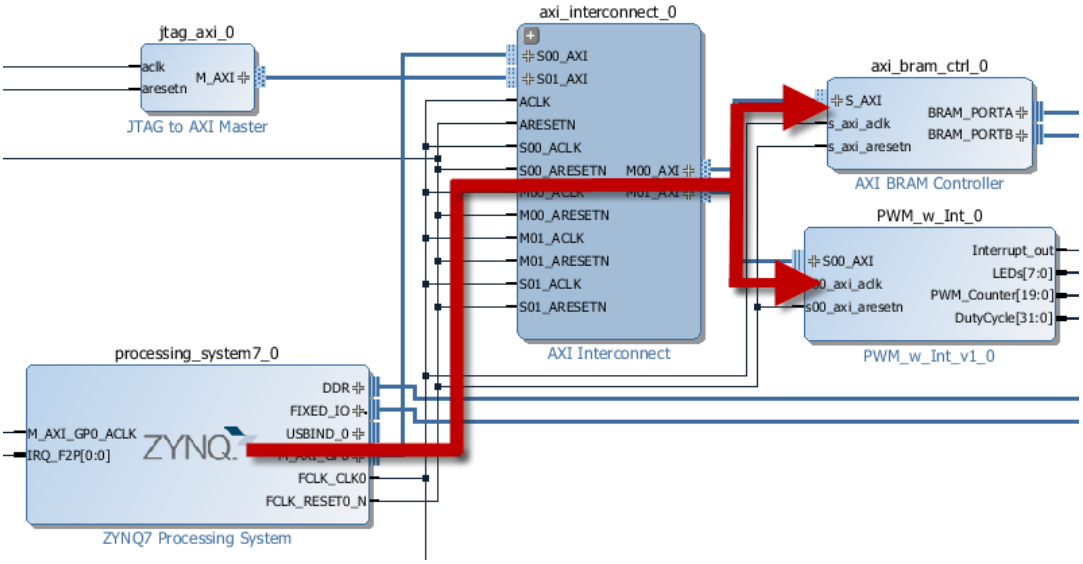

Figure 24 - PS Generated Transactions

With the AXI-JTAG Master core, AXI transaction instructions come from the JTAG chain via Vivado Logic Analyzer and TCL commands.

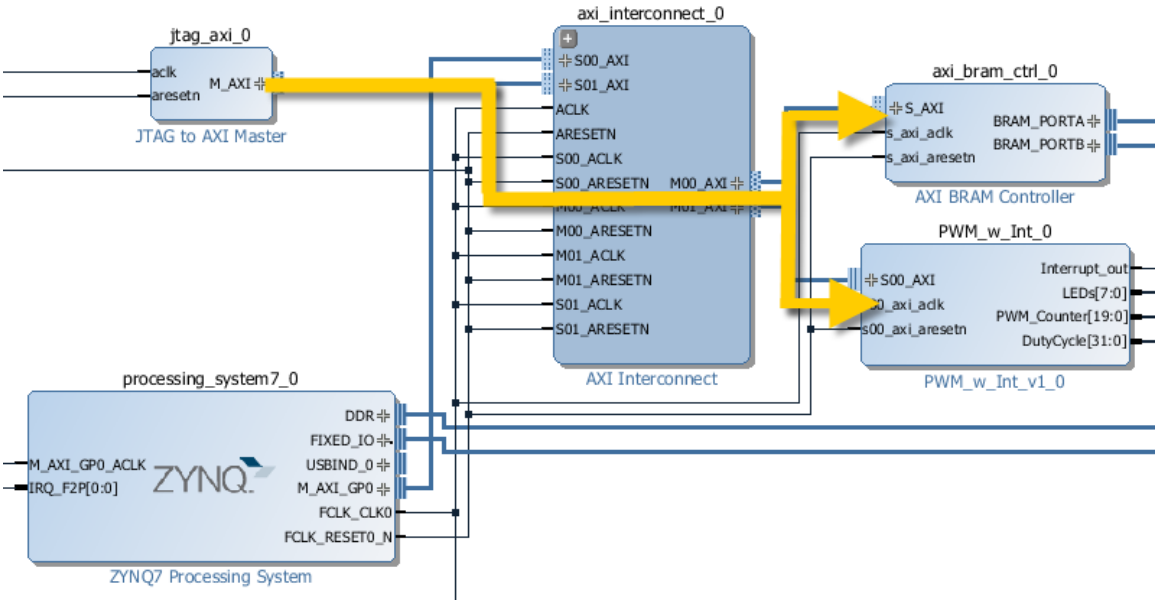

Figure 25 - AXI-JTAG Generated Transactions

Experiment 3 General Instruction:

Perform AXI transactions via TCL commands while monitoring the ILA core.

Experiment 3 Step-by-Step Instructions:

1. In Hardware Analyzer, find the TCL Console.

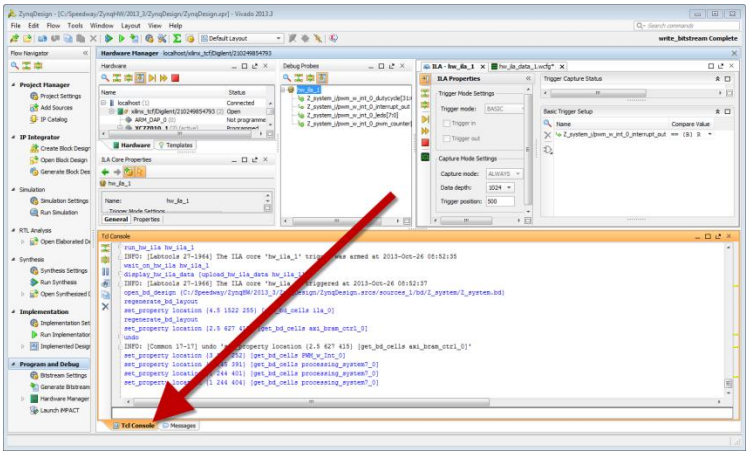

Figure 26 - TCL Console

2. Also find the **name** of our AXI-JTAG Master.

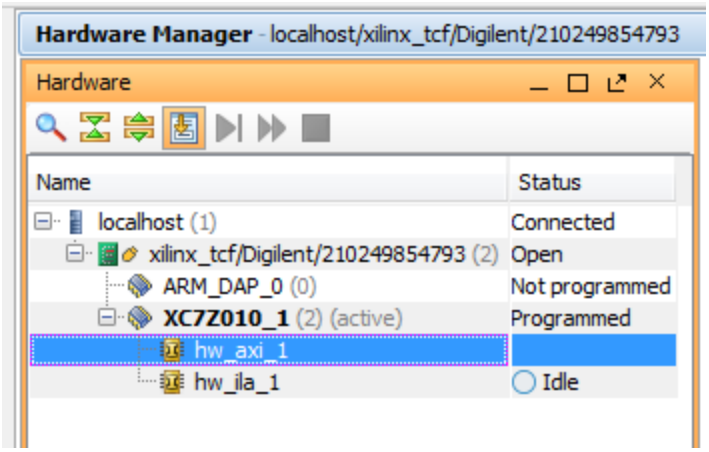

Figure 27 - hw_axi_1

3. The first step is to reset the AXI interface, enter the TCL command:

   reset_hw_axi [get_hw_axis hw_axi_1]

4. Before we can perform AXI transactions, we need the address of our peripherals. You may recall from our block design, the following addresses were assigned:

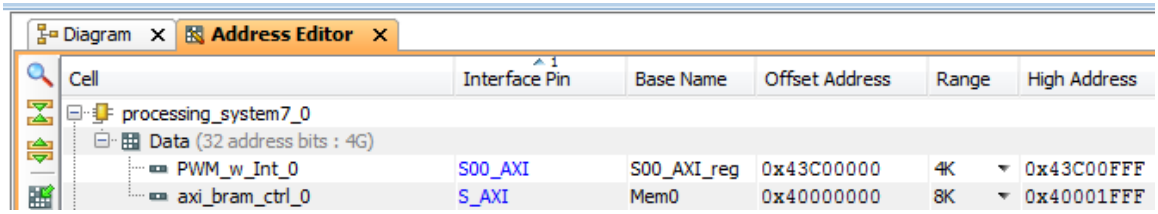

Figure 28 - Peripheral Addresses

5. The next step is to create an AXI transaction command. The command below, when run, will issue an AXI write at a specific address with specified data.

   create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type WRITE -address 43C00000 -len 1 -data 000A1220

   - **write_txn** is the name of the transaction
   - **[get_hw_axis hw_axi_1]** returns the hw_axi_1 object
   - **-address 43C00000** is the start address
   - **-len 1** sets the AXI burst length to 1 words
   - -**data 000A1220** is the data to send
     - This is brightness level 6 (decimal = 660000)

6. Once the command has been created we can run it with this TCL command. Write the brightness value to the PWM Controller peripheral.

   run_hw_axi [get_hw_axi_txns write_txn]

7. **Trigger Immediate** and open the waveform view. The brightness value should now be set to 660,000.

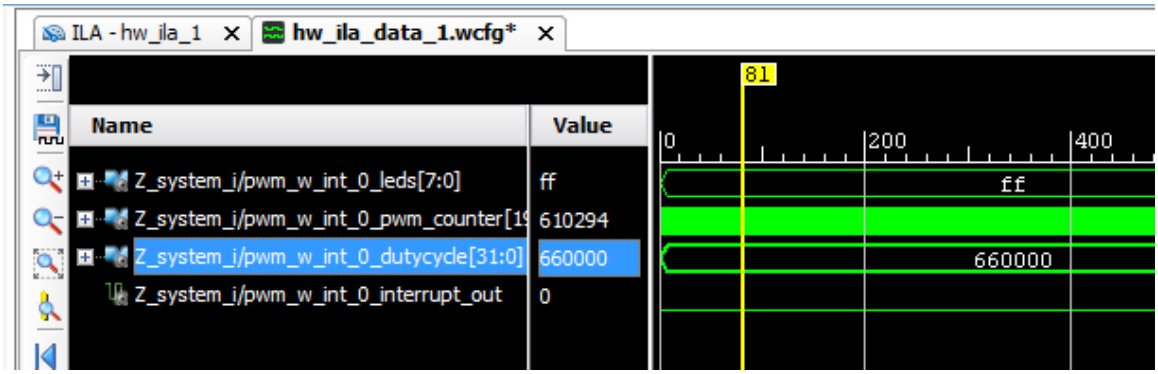

Figure 29 - AXI Transaction Complete

8. To change the write data of the write_txn transaction, simply edit the data field of the command.

   set_property DATA 000FFFFF [get_hw_axi_txns write_txn]

9. **Run Trigger**, not trigger immediate in the Analyzer.

10. Then running the run_hw_axi command:

    run_hw_axi [get_hw_axi_txns write_txn]

11. The waveform view should update showing the exception.

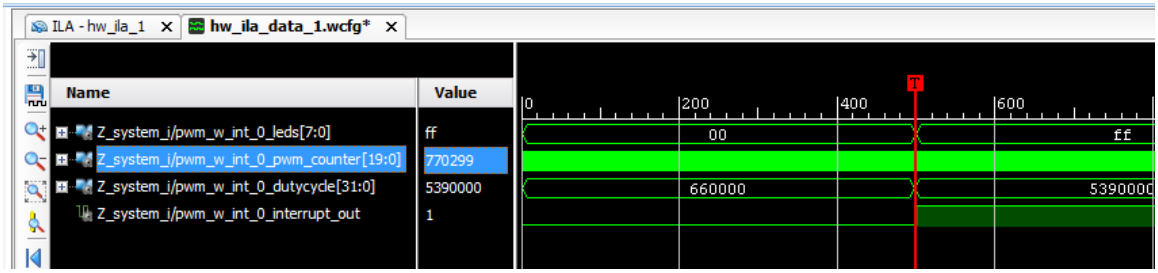

Figure 30 - Waveform view showing exception

12. Try reading data from the BRAM. Again, first create the command:

create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type READ -address 40000000 -len 4

13. Then run the command:

run_hw_axi [get_hw_axi_txns read_txn]

The result in the TCL console should look like this:

```
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type READ -address 40000000 -len 4
read_txn
run_hw_axi [get_hw_axi_txns read_txn]
INFO: [Labtools 27-147] vcse_server: READ DATA is : 00000000000000000000000000000000
```

Figure 31 - BRAM contents at base address

14. Create a new AXI Write transaction for the BRAM:

create_hw_axi_txn write_bram [get_hw_axis hw_axi_1] -type WRITE -address 40000000 -len 4 -data {44444444_33333333_22222222_11111111}

15. Then run the new BRAM Write Command:

run_hw_axi [get_hw_axi_txns write_bram]

16. Read the BRAM contents again to see if they have been updated:

run_hw_axi [get_hw_axi_txns read_txn]

```
create_hw_axi_txn write_bram [get_hw_axis hw_axi_1] -type WRITE -address 40000000 -len 4 -data {44444444_33333333_22222222_11111111}
write_bram
run_hw_axi [get_hw_axi_txns write_bram]
INFO: [Labtools 27-147] vcse_server: WRITE DATA is : 44444444333333332222222211111111
run_hw_axi [get_hw_axi_txns read_txn]
INFO: [Labtools 27-147] vcse_server: READ DATA is : 44444444333333332222222211111111
```

In summary, the JTAG to AXI Master core can easily be added to a design. And it's very quick and easy to use TCL commands to run read and write transactions to and from any AXI-based slave peripheral in a design.

Questions:

Answer the following questions:

- *How would you delete one of the AXI transaction commands?*

_____

Exploring Further

If you have more time and would like to investigate more…

- Explore more AXI TCL commands (type: help *axi*)
- Run the following TCL command to see the valid AXI transaction properties: report_property [get_hw_axi_txns write_txn]

This concludes Lab 8.

Revision History

Date	Version	Revision
6 Nov 13	02	Initial Draft
19 Nov 13	03	Pilot Updates

Resources

www.microzed.org

www.zedboard.org

www.xilinx.com/zynq

www.xilinx.com/sdk

www.xilinx.com/vivado

Answers

Experiment 1

- *What function is called when the Interrupt is detected?*

PWMIsr

```
/* Connect the interrupt handler that will be called when an
* interrupt occurs for the device. */
result = XScuGic_Connect(IntcInstancePtr, INTC_PWM_INTERRUPT_ID,
                         (Xil_ExceptionHandler) PWMIsr, 0);
```

- *What does the PWMIsr function do?*

Resets the Brightness value and writes it out.

- *In the PWMIsr function, what does it do to the brightness value?*

Zero

- *What is the INTC_PWM_INTERRUPT_ID? Where did this number come from?*

This is defined as: `XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR`
This is assigned to 91 in xparameters.h?

```
/* Definitions for Fabric interrupts connected to ps7_scugic_0 */
#define XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR 91
```

Experiment 2

- *Can you detect how long it takes the processor to handle the interrupt?*

Technically yes, but not with this design. Why? Because our capture depth is only 1024 clock cycles, and the trigger does not reset in that time. If the capture depth was increased then it could be captured.

Experiment 3

- *How would you delete one of the AXI transaction commands?*

delete_hw_axi_txn [get_hw_axi_txns write_txn]