Introduction to Zynq Hardware
Lab 8
Hardware Debugging Zynq Designs

June 2017
Version 11

AVNET
Reach Further™

Copyright © 2017 Avnet Inc. All rights reserved

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 2014.3 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` and `ip_repo` folders in the `ZynqHW/2017_1` folder. Then unzip **`Solutions\ZynqHW_Lab7_Solution.zip`** to the `2017_1` folder. If you have 7-zip installed, you can do this by right-clicking and dragging **`ZynqHW_Lab7_Solution.zip`** to the `2017_1` folder. Select **7-Zip → Extract Here**.

2. Open Vivado and make sure that the implemented Design is opened.

3. Export the hardware design (File → Export → Export Hardware) to a new folder named **ZynqDesign.lab8** which will also serve as the new SDK workspace. Make sure you export the bitstream as well as we have IP in the PL now. Click **OK**.

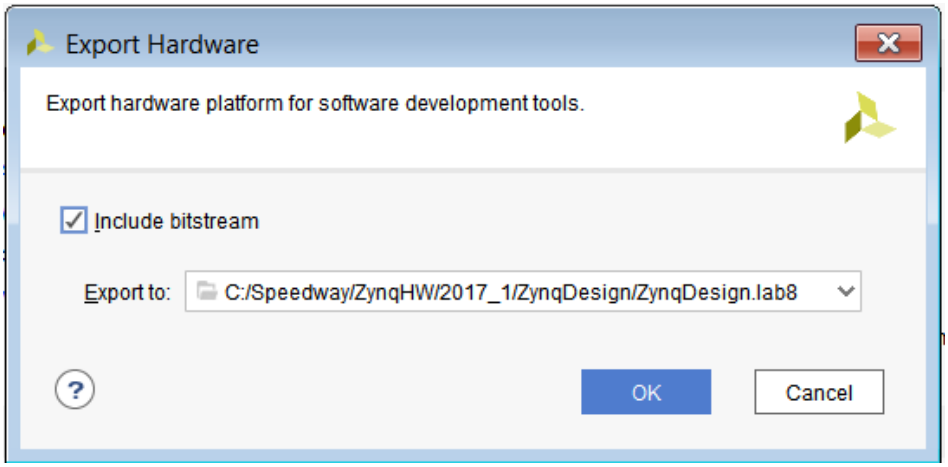

Figure 1 – Export hardware for SDK

4. Select **File → Launch SDK**. Change the *Exported location* and *Workspace* to the new **ZynqDesign.lab8** location. Click **OK**.

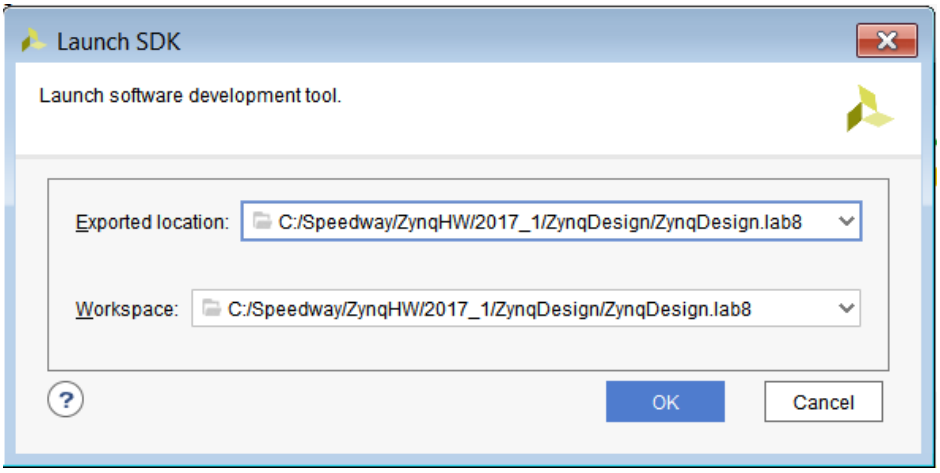

Figure 2 – Launch SDK

5. Create the standalone BSP using **File → New → Board Support Package → Finish → OK**.

6. Now import the previously created applications. Select **File → Import → General → Existing Projects into Workspace**.

7. Browse to the ZynqDesign.lab6 directory and select **OK**. Make sure the four applications are checked. Check the box to **Copy projects into workspace**. Then click **Finish**.

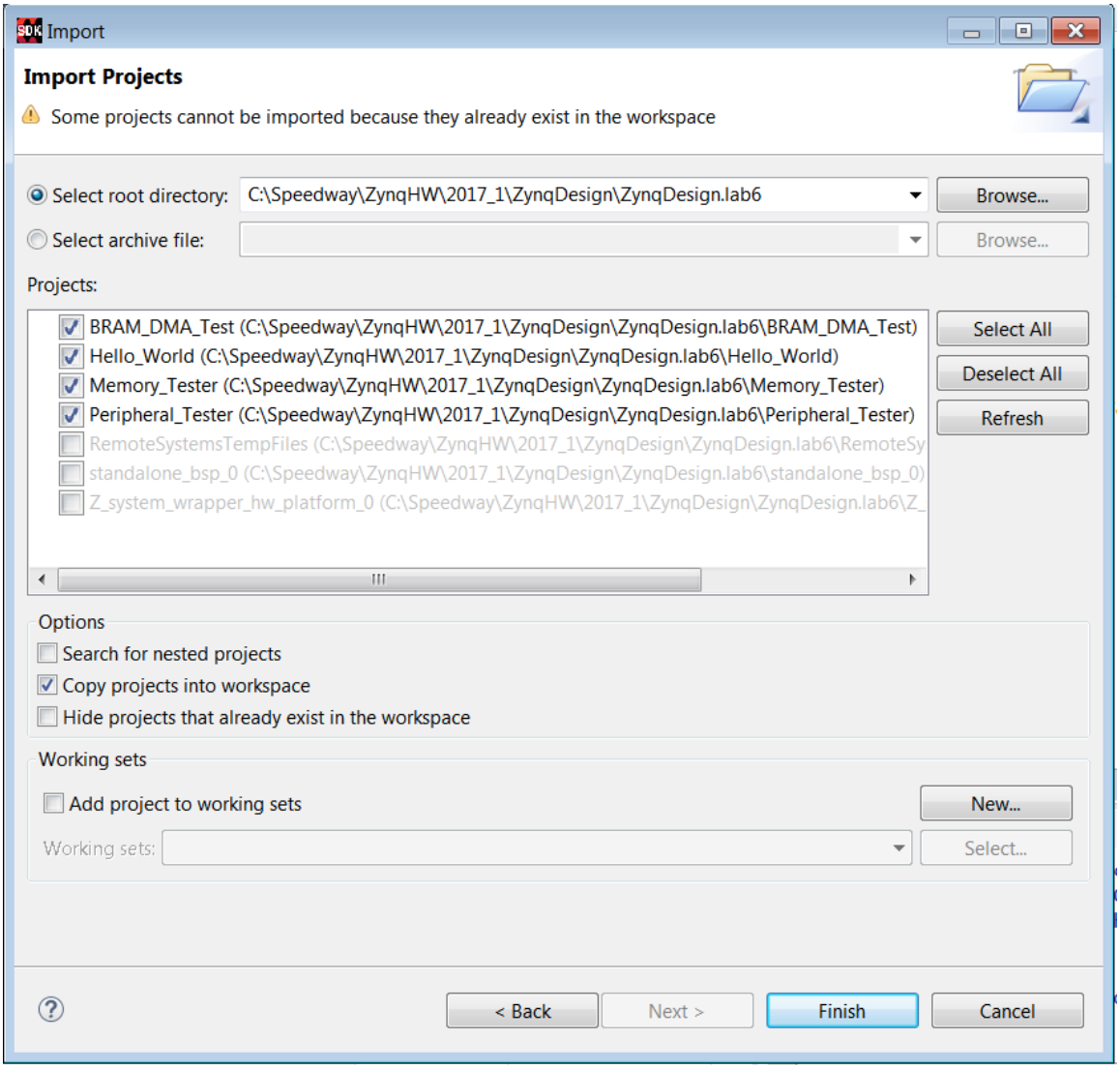

Figure 3 – Import Previous Applications

8. Select **File → New → Application Project**
9. Name it **LED_Dimmer_Int.**
10. Use existing standalone BSP.
11. Click **Next>**.

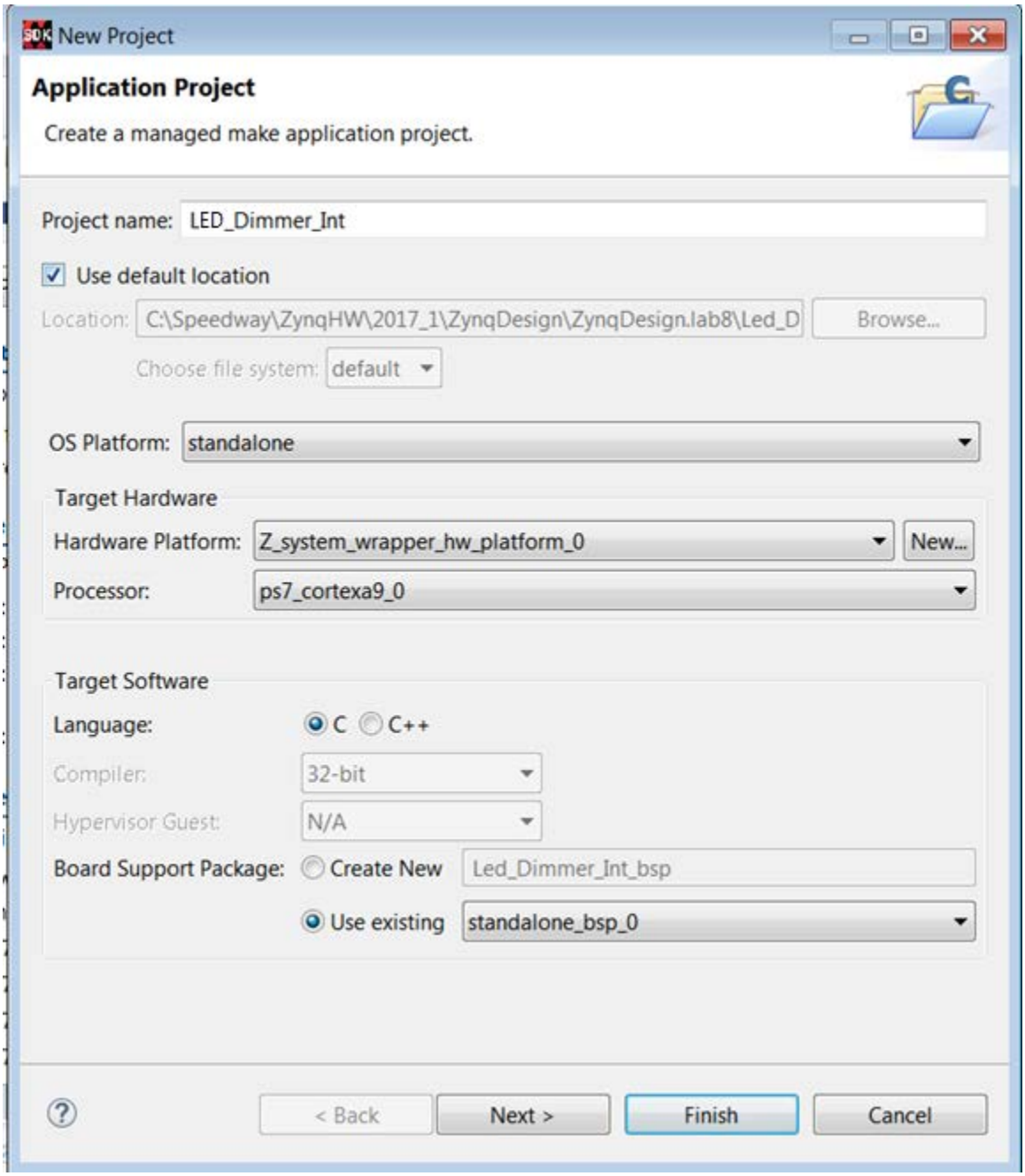

Figure 4 – New LED_Dimmer_Int Application

12. Select **Empty Application** then click **Finish**.

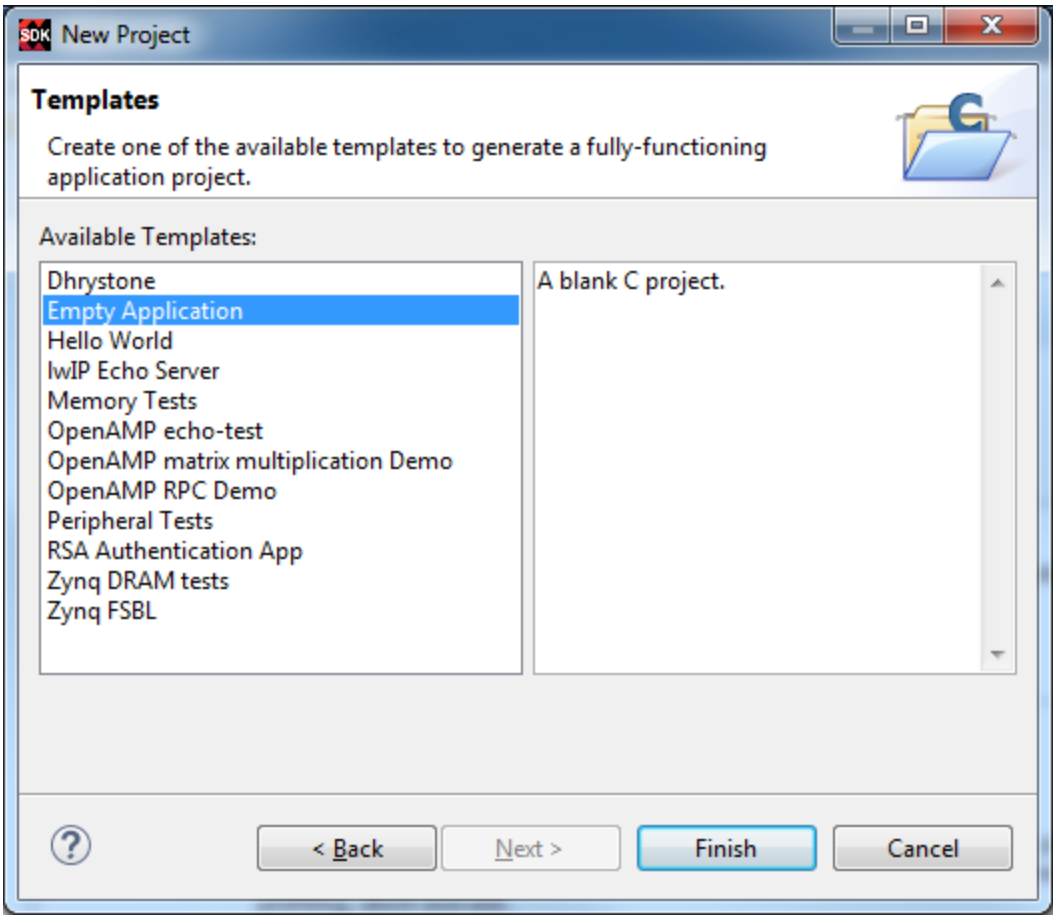

Figure 5 – Start with Empty Application

13. With the **LED_Dimmer_Int/SRC** folder highlighted right click and select **Import**. Next select **General→File System** then click **Next >**.

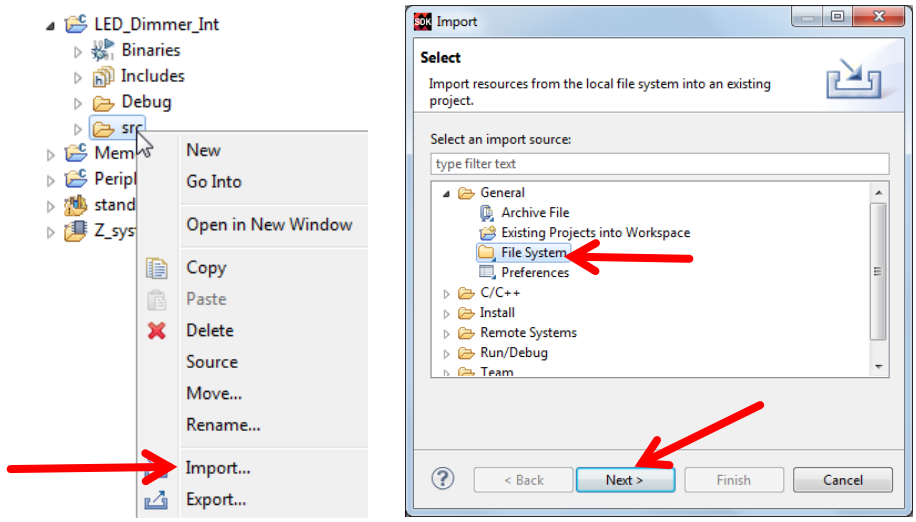

Figure 6 – Import from Support_Documents

14. Browse to C**:\Speedway\ZynqHW\2017_1\Support_documents.** Select **LED_Dimmer_Int.c**. Then click **Finish**.

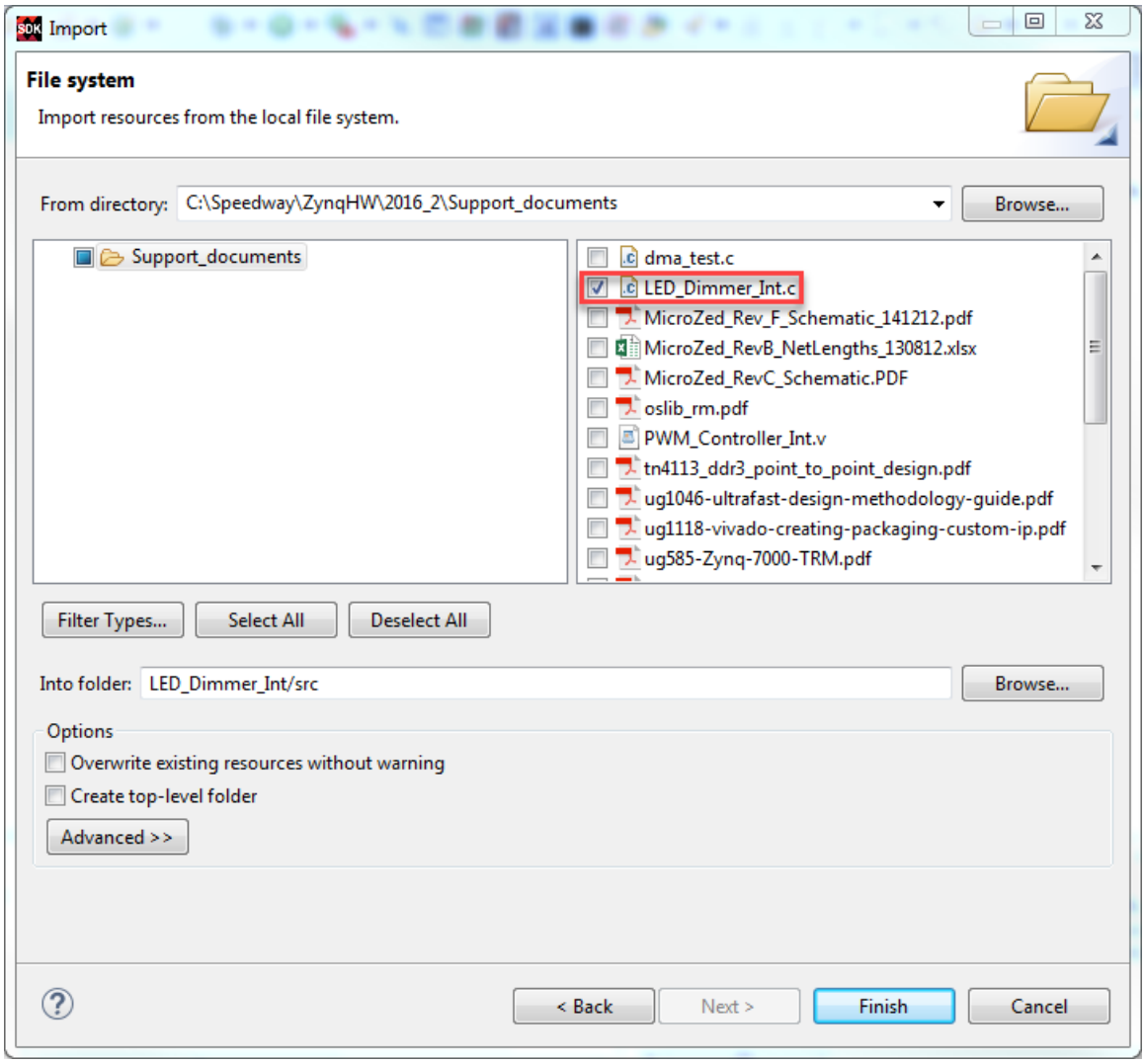

Figure 7 – Import LED_Dimmer_Int.c

15. The application will build, when done, open the source code, **LED_Dimmer_Int.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 the SW application to the board. Open Vivado Hardware Analyzer. Trigger events in the Analyzer while running software.

Experiment 2 Step-by-Step Instructions:

1. If the hardware was previously disconnected, make sure any Terminal windows are closed, then reconnect the USB_JTAG_UART J2 to turn it on.

2. **Program** the FPGA.

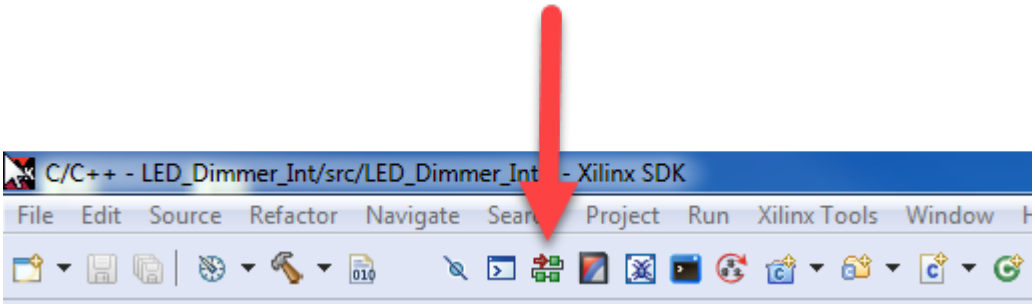

Figure 8 - Program FPGA

3. Select **Program FPGA,** click **Program**.

4. Open Terminal, such as **Tera Term**, and set the **COM port** to active COM setting for your board and set the **Baud Rate at 115,200**.

5. With **LED_Dimmer_Int** highlighted right click and select **Run As → Launch on Hardware(System Debugger)** the **LED_Dimmer_Int** application.

6. 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.

7. In Vivado, select **Open Hardware Manager** from the Flow Navigator.

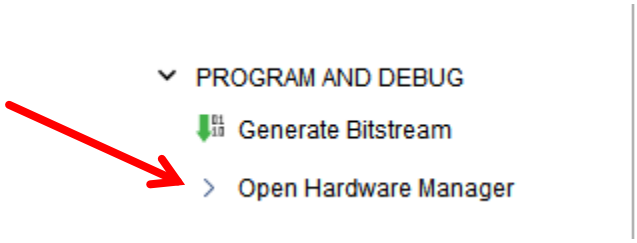

Figure 9 - Open Hardware Manager

8. At the top of the Hardware Manager Window, select **Open target** -> **Auto Connect**.

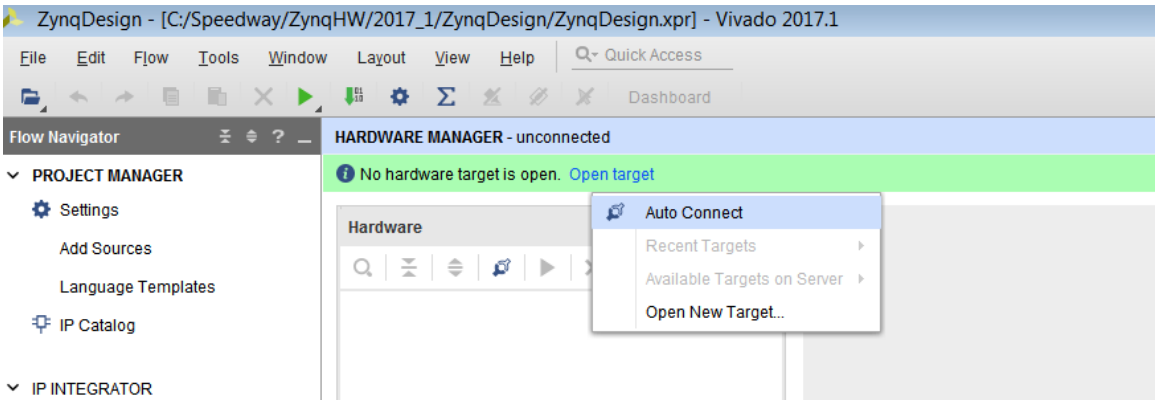

Figure 10 - Open new hardware target

9. The Hardware Manager is now ready. If not selected, select the **ILA core**.

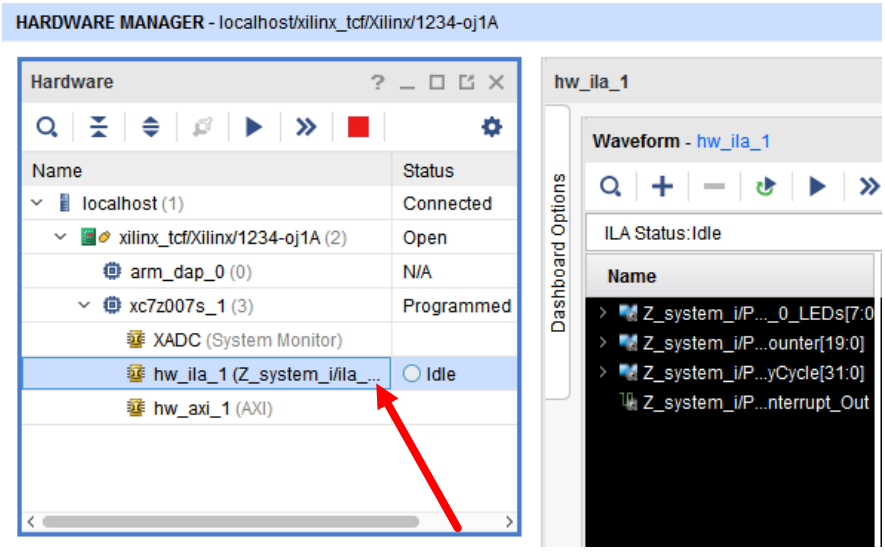

Figure 11 - Select ILA Core

10. Click the **Run Trigger Immediate for this ILA Core** button from the shortcut bar.

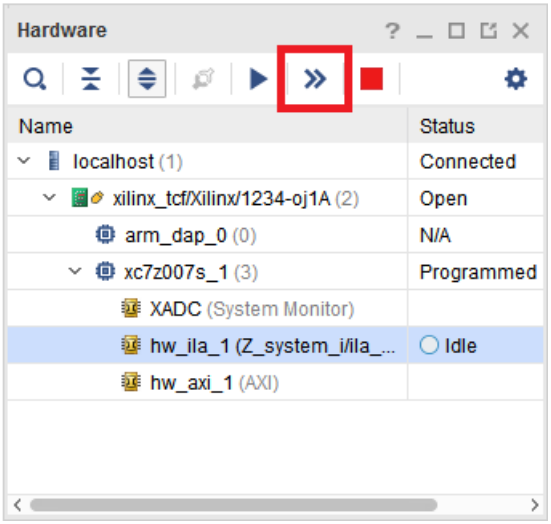

Figure 12 - Run Trigger Immediate

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

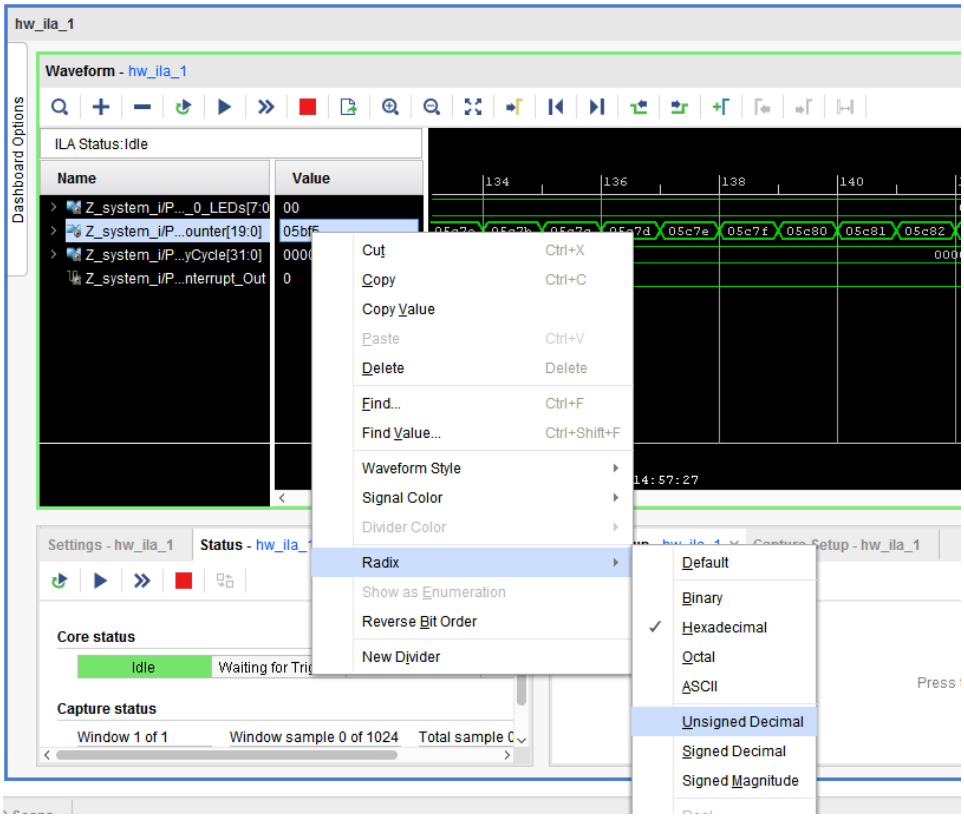

Figure 13 - Waveform View

12. To help us to interpret the values of the **DutyCycle** waveform accurately, also change the **radix** of that value to **Unsigned Decimal**.

13. In the terminal, type **4** and then press enter. **Run Trigger Immediate** again.

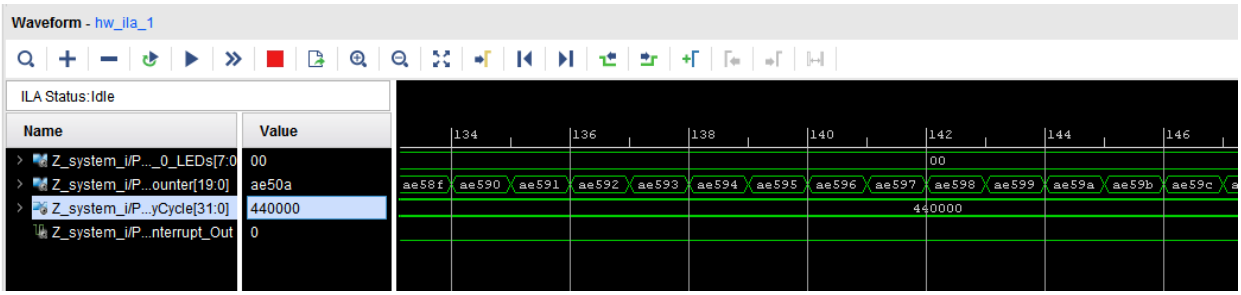

Figure 14 - Waveform View, Brightness level 4

Notice, the DutyCycle is set to 440,000 because of the value 4 that was entered into the terminal. 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, the counter counts to roughly 1 million. Any number over 1 million should create an exception causing an interrupt. To capture an interrupt happening, we'll have to create a trigger in the ILA.

14. Go to the Trigger Setup window of the hw_ila_1 tab

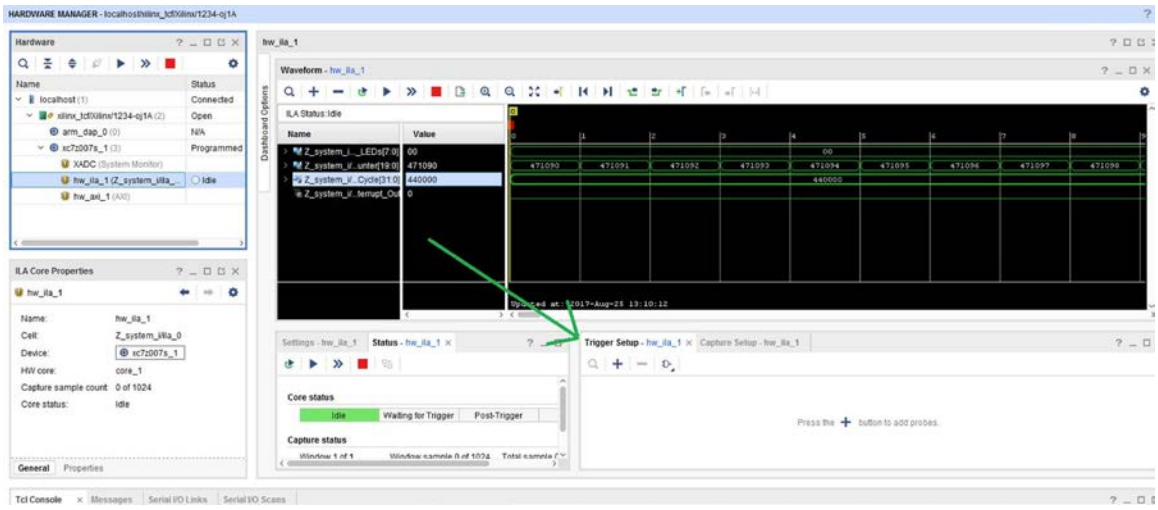

Figure 15 – ILA Settings

15. Press the + button, and select the **Interrupt_out** signal to add it as a probe. Click **OK**

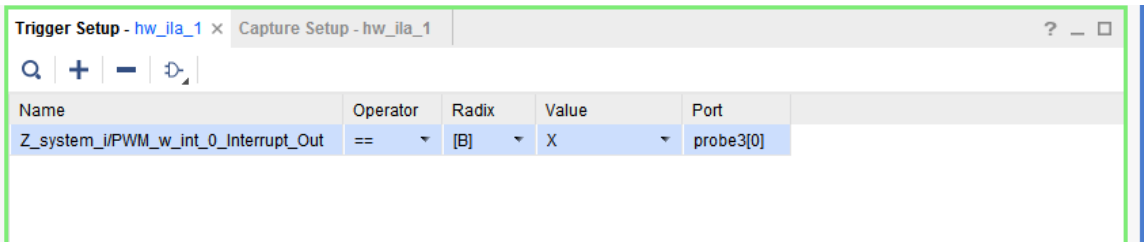

Figure 16 - Add Probes to Trigger

16. Left-click on the *Value* for **Interrupt_out** and change the value to **R**, for rising edge.

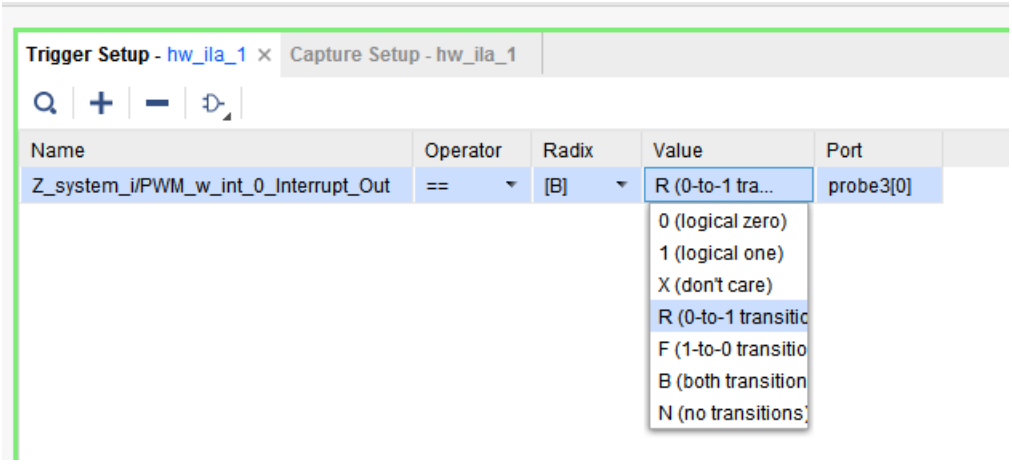

Figure 177 - Trigger Setup

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

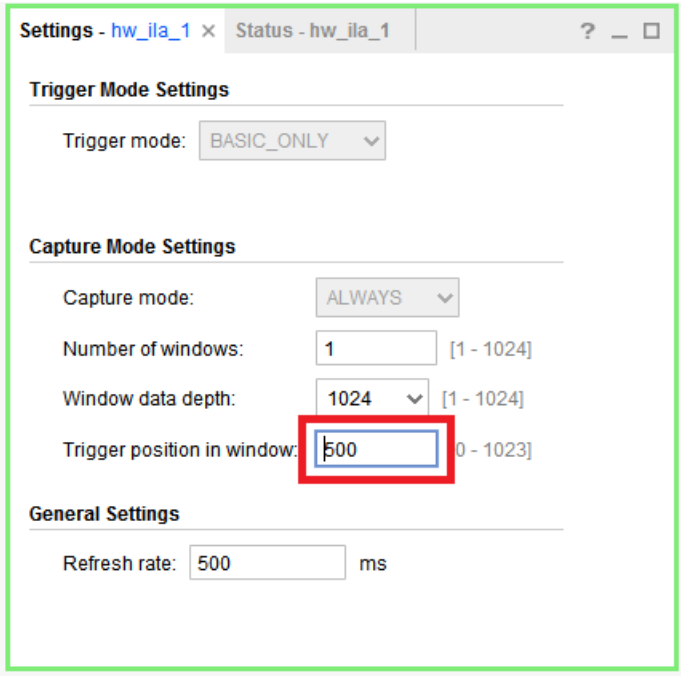

Figure 18 - Set Trigger Depth

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

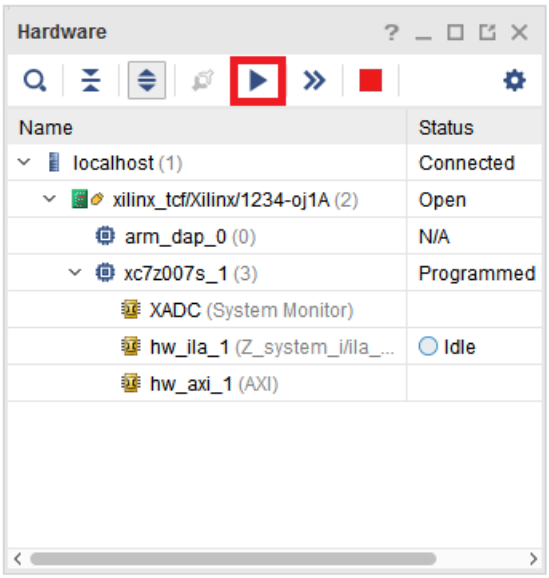

Figure 19 - Run trigger

19. In the Terminal, enter several **numbers** to see if the trigger occurs. The trigger should not occur. You should still see the Trigger Capture Status as Waiting for Trigger.

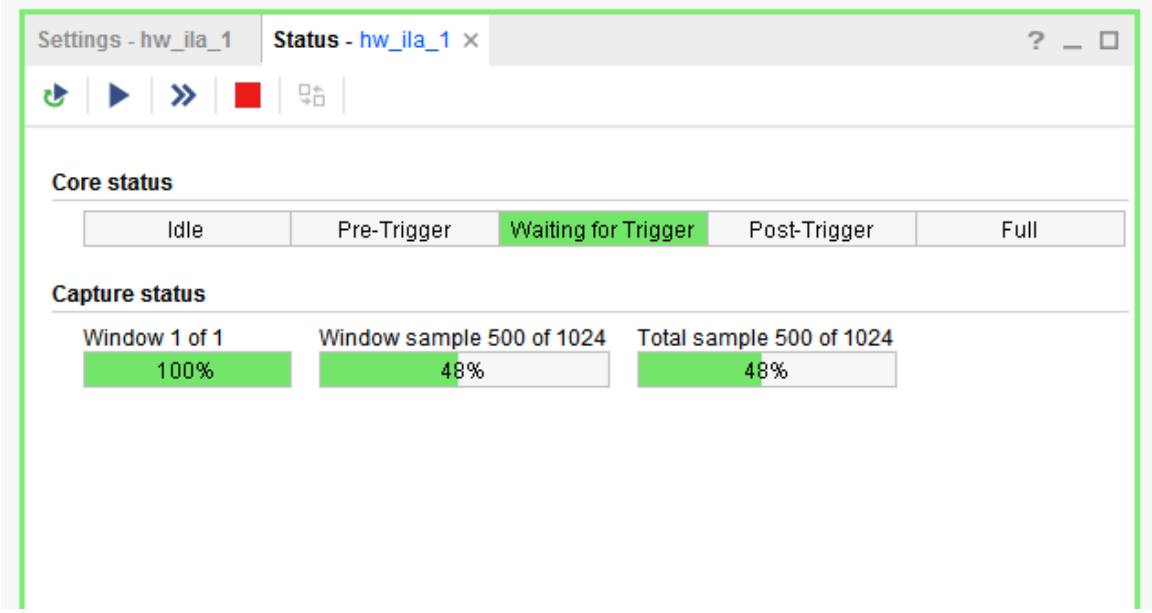

Figure 20 - Waiting for Trigger

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

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

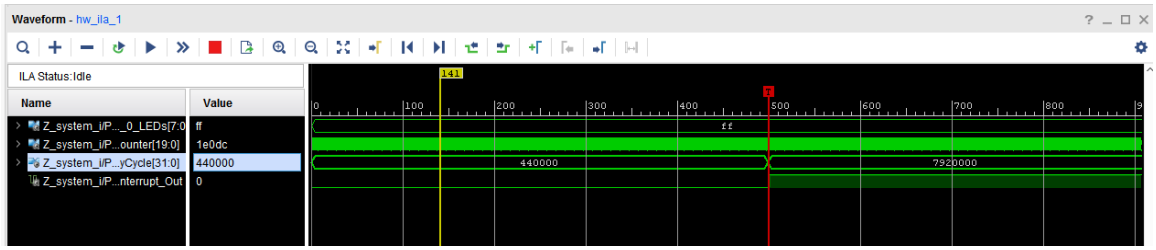

Figure 21 - Waveform showing Trigger event

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 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.

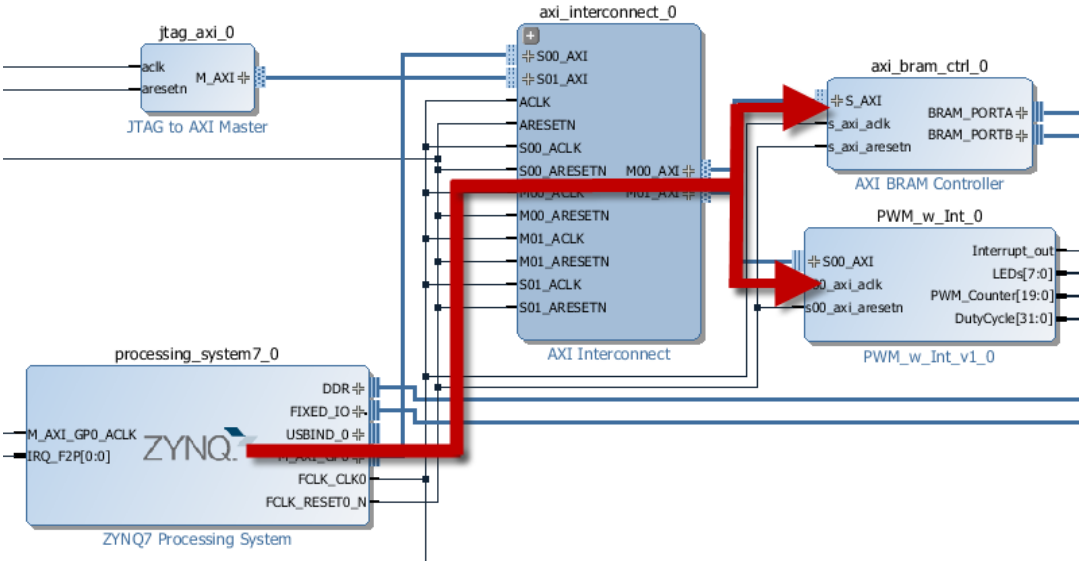

Figure 22 - PS Generated Transactions

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

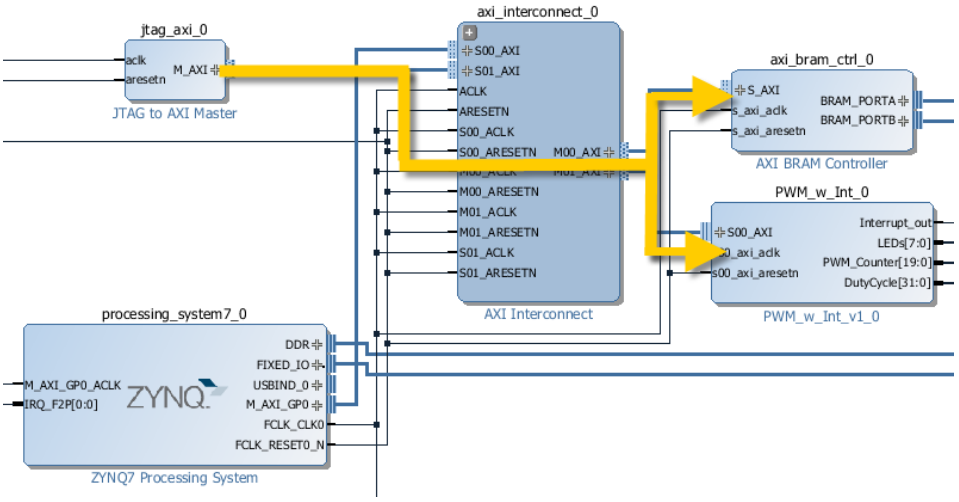

Figure 23 - AXI-JTAG Generated Transactions

Experiment 3 General Instruction:

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

Experiment 3 Step-by-Step Instructions:

1. In Hardware Manager, find the TCL Console.

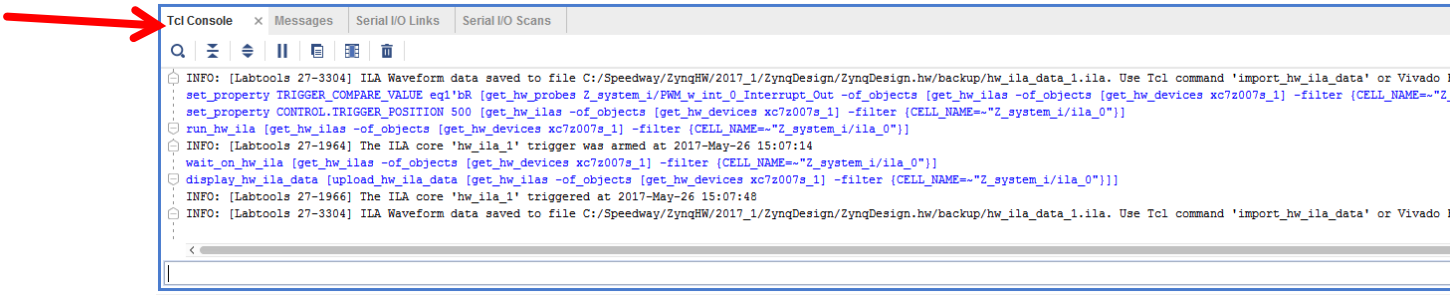

Figure 24 - TCL Console

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

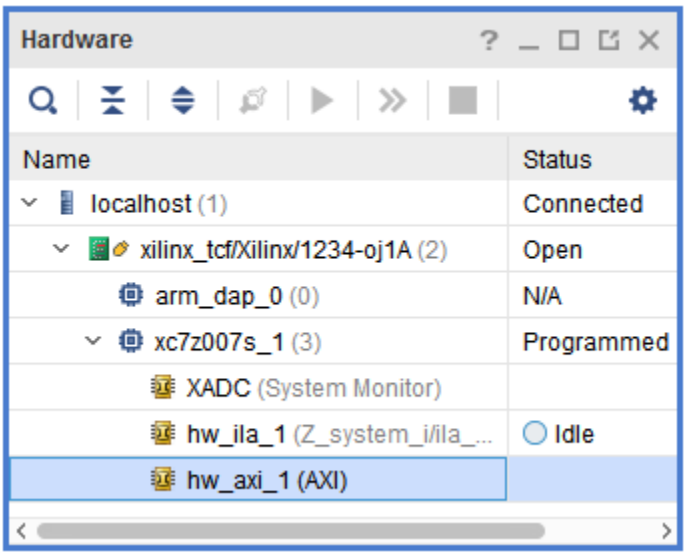

Figure 25 - hw_axi_1

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:

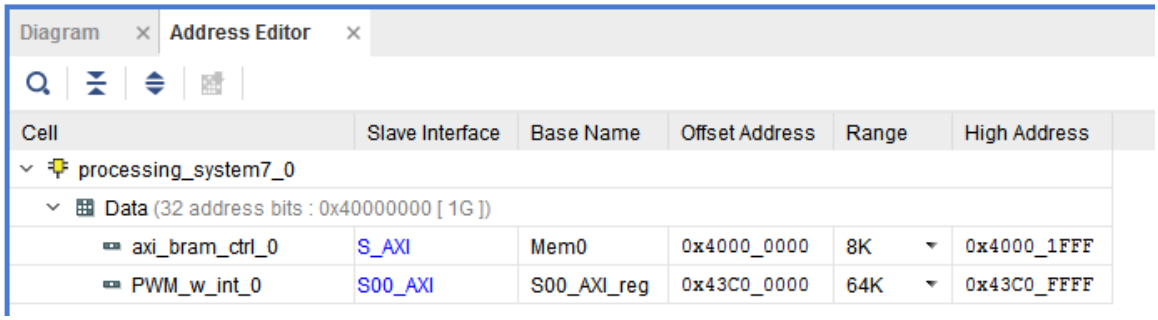

Figure 26 - Peripheral Addresses

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. **Run the TCL command.**

   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 27 - 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. Select **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.

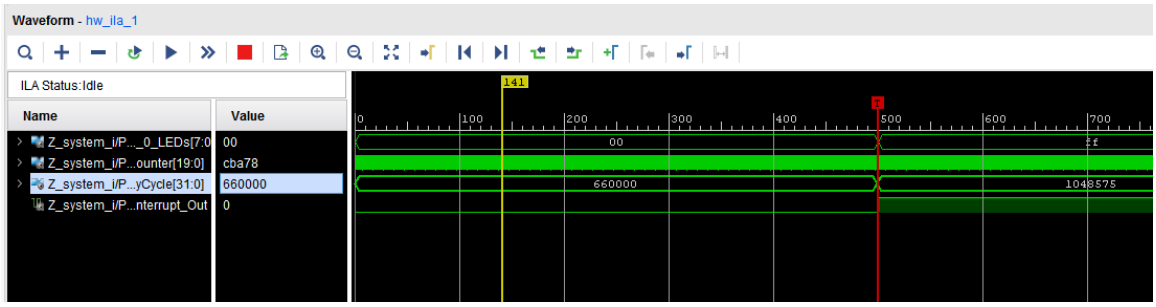

Figure 28 - Waveform view showing exception

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 29 - 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 by running the command:

    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: [Labtoolstcl 44-481] WRITE DATA is: 44444444_33333333_22222222_11111111
run_hw_axi [get_hw_axi_txns read_txn]
INFO: [Labtoolstcl 44-481] READ DATA is: 44444444333333332222222211111111
```

Figure 30 – Updated BRAM contents

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.

Experiment 4: Delivering Hardware to the Software Team

The previous experiment showed how to interact with the JTAG-AXI core to test the IP outside of a software environment. Now that the hardware platform development is complete and we have tested the hardware platform to validate that it functions as expected, we will perform some clean up step. Since the JTAG-AXI core is not something that software engineers would need in order to develop software, it will be removed during this experiment. One might say the same thing of the ILA core, however we will remove that in lab 9.

Experiment 4 General Instruction:

Remove the JTAG-AXI core from the design and export the hardware platform.

Experiment 4 Step-by-Step Instructions:

1. In the Flow Navigator pane, click **Open Block Design** to open our hardware design for modification.

2. With the block design open, right click on the **jtag_axi_0** instance of the **JTAG to AXI Master** IP block and select the **Delete** option from the drop-down menu.

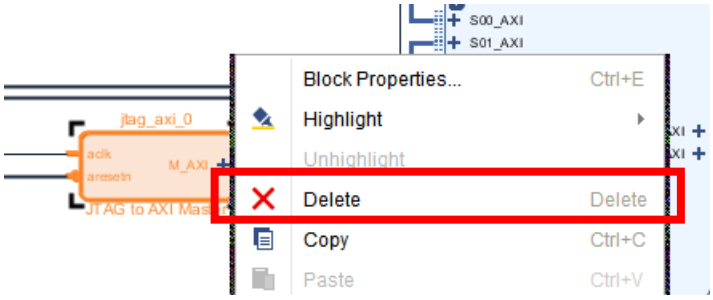

Figure 31 – Delete the JTAG to AXI Master IP Instance

3. Notice that the AXI interconnect block still retains the AXI Slave port that was connected to the JTAG-AXI block. That port should be disabled to prevent warnings from being reported during design validation. Double-click the **axi_mem_intercon** instance of **AXI Interconnect** block to customize the IP settings.

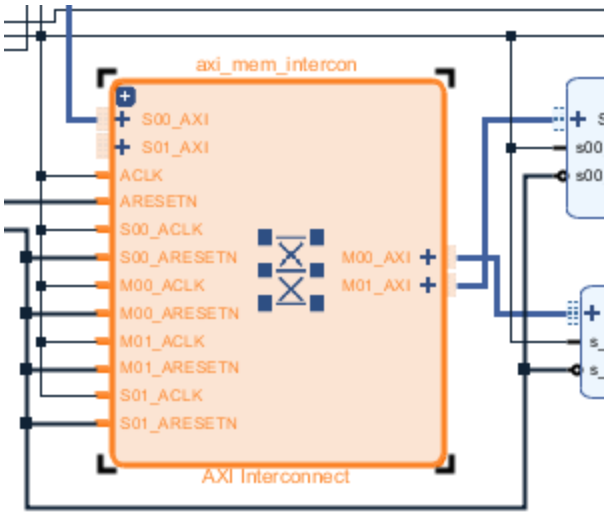

Figure 32 – Customize the AXI Interconnect IP Instance

4. Under the **Top Level Settings** tab for the AXI Interconnect IP settings customization window, set the **Number of Slave Interfaces** to **1**. Then click the **OK** button. This will disable the second AXI slave port and it will no longer be visible within the block design.

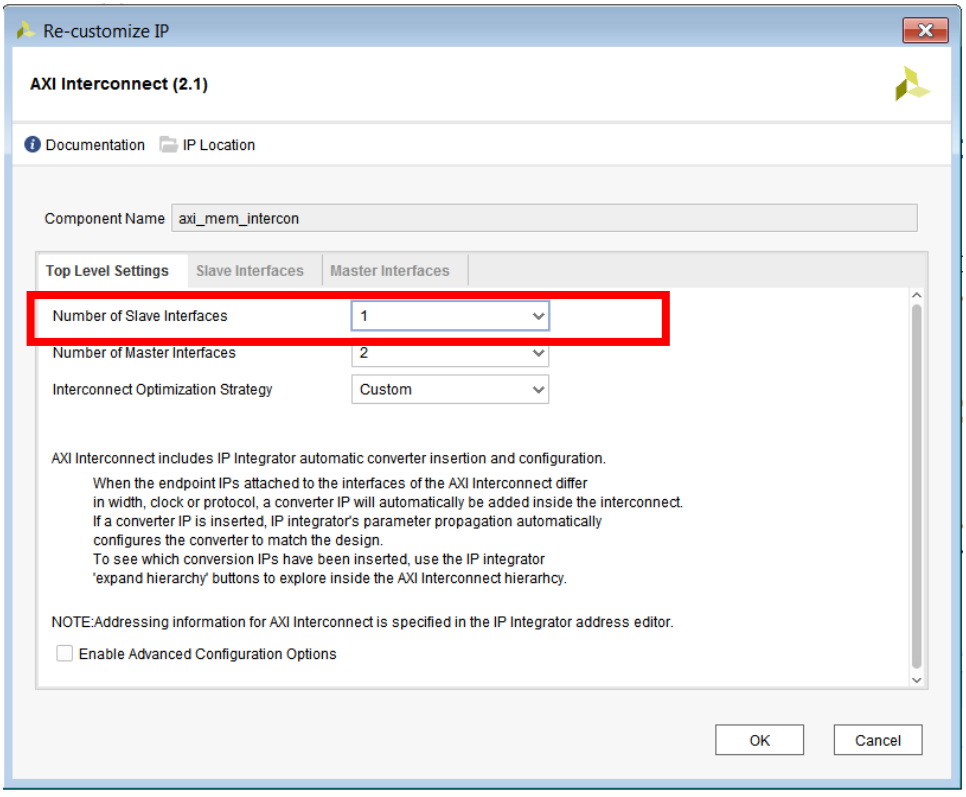

Figure 33 – Reduce the Number of Slave Instances to 1

5. **Regenerate Layout** and **validate** the block design.

6. If OK, **Save** the block design.

7. Right-click on Z_system_i in the Sources tab and **Reset Output Products** then **Generate Output Products**.

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*)
- Explore the hardware platform that was exported to SDK

This concludes Lab 8.

Revision History

Date	Version	Revision
6 Nov 13	02	Initial Draft
19 Nov 13	03	Pilot Updates
5 Nov 14	04	Updated for Vivado 2014.3
5 Jan 15	05	Updated for Vivado 2014.4
07 Mar 15	06	Finalize for Vivado 2014.4
17 Mar 15	07	Minor edits for release
Oct 15	08	Updated for Vivado 2015.2
Aug 16	09	Updated for Vivado 2016.2
May 17	10	Updated for Vivado 2017.1 and MiniZed
June 17	11	Updated to Vivado 2017.1 for MiniZed + Rebranding

Resources

www.minized.org

www.microzed.org

www.picozed.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 61 in xparameters.**h?**

```
/* Definitions for Fabric interrupts connected to ps7_scugic_0 */
#define XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR 61
```

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]