

Creating a custom IP block in Vivado Using ZedBoard: A Tutorial

Embedded Processor Hardware Design
February 24th 2015.

Introduction

This tutorial will guide you through the process of using Vivado and IP Integrator to create a custom AXI IP block in Vivado and modify its functionality by integrating custom VHDL code. We will be using the Zync SoC and ZedBoard as a hardware platform. For simplicity, our custom IP will be a multiplier which our processor will be able to access through register reads and writes over an AXI bus.

The multiplier takes in two 16-bit unsigned inputs and then it will output one 32 bit unsigned value. A single 32 bit writes to the IP will contain the two 16-bit inputs, separated by the lower and higher 16 bits. A single 32 bit read from the peripheral will contain the result from the multiplication of the two 16-bit inputs. The design is simple but it is a good example of integrating your own code into an AXI IP block.

Objectives

After completing this tutorial, you will be able to:

- Create an embedded system design using Vivado and SDK flow
- Configure the Processing System (PS)
- Add a custom IP in the Programmable Logic (PL) section
- Use SDK to build a software project and verify the functionality in hardware.

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the tutorial.

This tutorial comprises three stages (each consisting of steps): You will create a top-level project using Vivado, create the processor system using the IP Integrator, add two instances of the GPIO IP, validate the design, generate the bitstream, export to the SDK, create an application in the SDK, and, test the design in hardware. You will then be able to profile the application and produce statistics that will help you understand the main bottlenecks of your application.

Requirements

The following is needed in order to follow this tutorial:

- Vivado w/ Xilinx SDK (tested, **version 2014.4**)
- Zedboard (tested, **version D**)

Part 1: Building a Zynq-7000 Processor Hardware

Introduction

In this part of the tutorial you will create a Zynq-7000 processor based design and instantiate IP in the processing logic fabric (PL) to complete your design. Then you take the design through implementation, generate a bitstream, and export the hardware to SDK.

If you are not familiar with the Vivado Integrated Development Environment Vivado (IDE), see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893).

Step 1: Start the Vivado IDE and Create a Project

1. Start the Vivado IDE (**Figure 1**) by clicking the Vivado desktop icon or by typing **vivado** at a terminal command line.

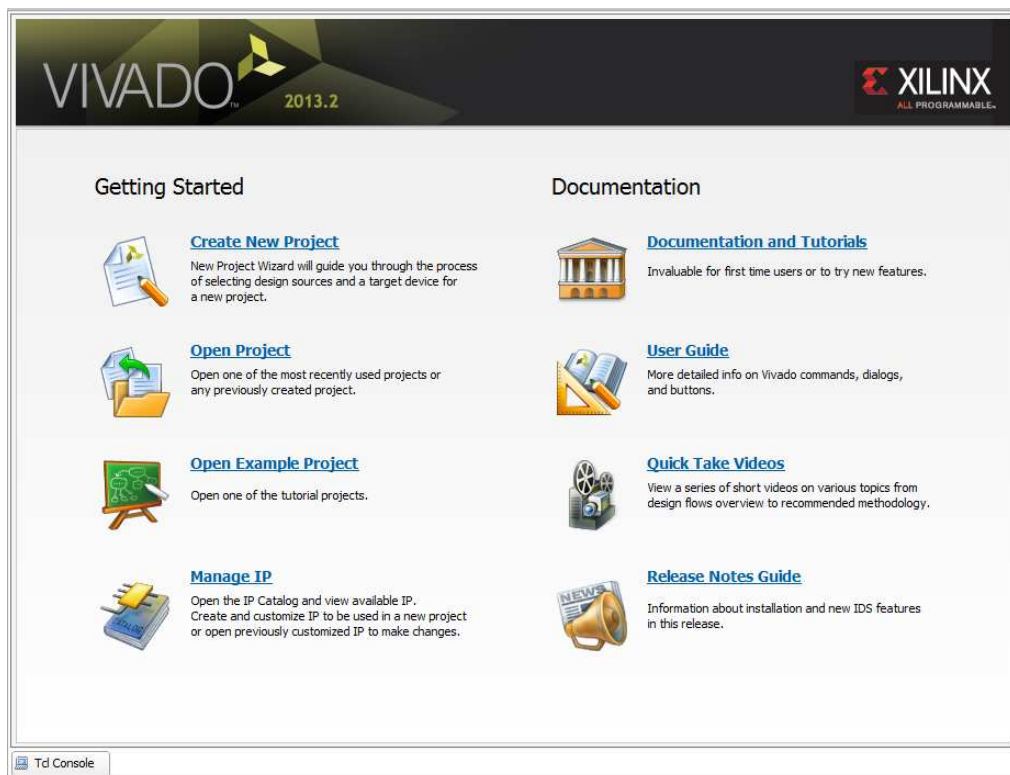


Figure 1: Getting Started Page

2. From the Getting Started page, select **Create New Project**. The New Project wizard opens (**Figure 2**).
3. Click **Next**

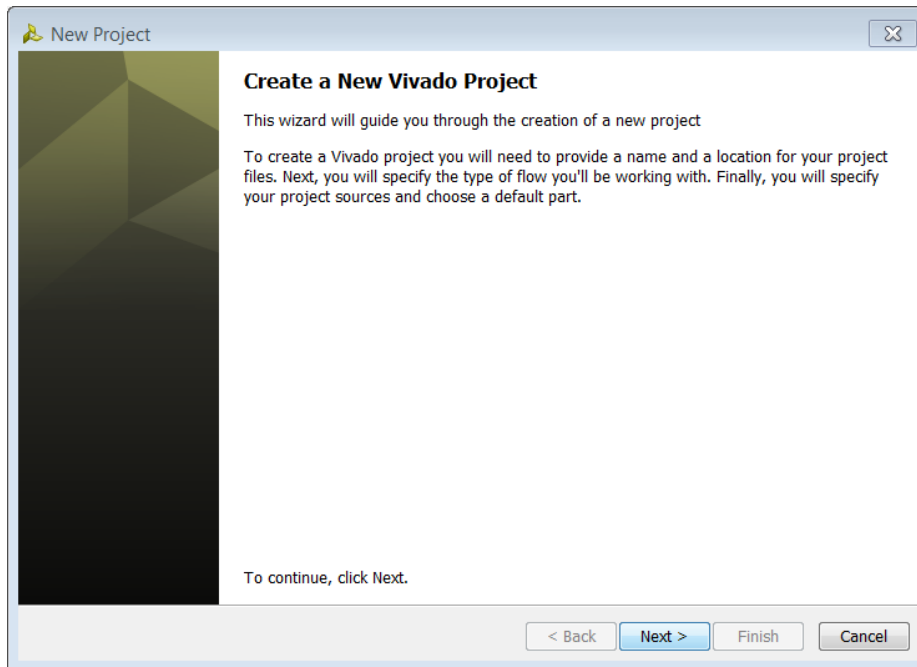


Figure 2: Create New Project Wizard

4. In the **Project Name** dialog box, type the project name and location. Ensure that **Create project subdirectory** is checked, and then click **Next**.
5. In the **Project Type** dialog box, select **RTL Project**, then click **Next**.
6. In the **Add Sources** dialog box, ensure that the **Target language** is set to **VHDL**, then click **Next**.
7. In the **Add Existing IP** dialog box, click **Next**.
8. In the **Add Constraints** dialog box, click **Next**.
9. In the **Default Part** dialog box select **Boards** and choose "ZedBoard Zynq Evaluation and Development Kit". Make sure that you have selected the proper Board Version to match your hardware because multiple versions of hardware are supported in the Vivado IDE. Click **Next**.
10. Review the project summary in the **New Project Summary** dialog box before clicking **Finish** to create the project.

Step 2: Create the Base Processing System

1. In the Flow Navigator, select **Create Block Design**.

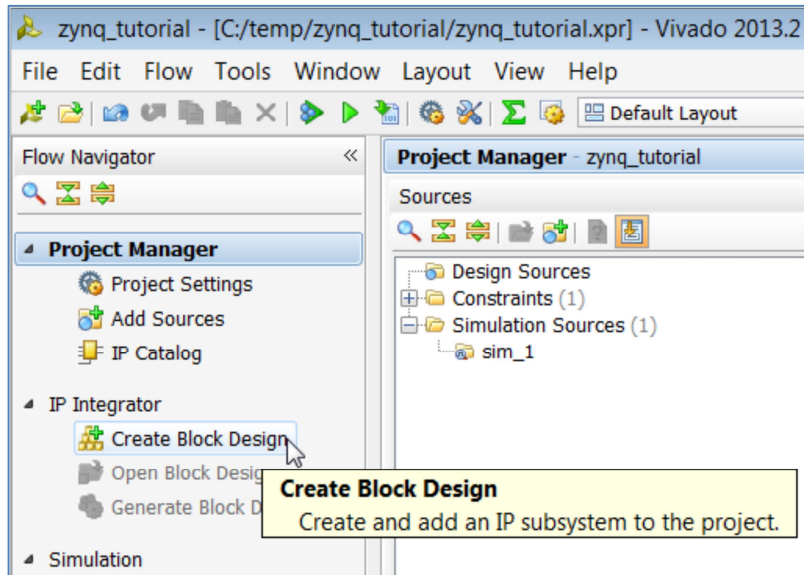


Figure 3: Create Block Design from Flow Navigator

2. In the **Create Block Design** popup menu, specify a name for your IP subsystem design as seen in **Figure 4**.

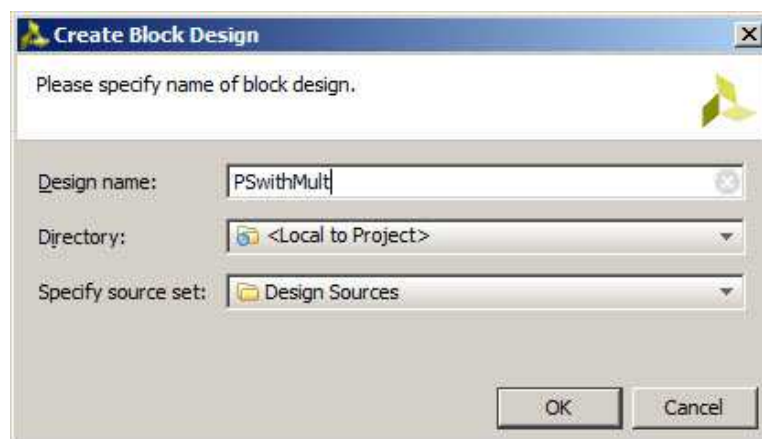


Figure 4: Create Block Design Dialog Box

3. Right-click in the Vivado IP integrator diagram window, and select **Add IP**.

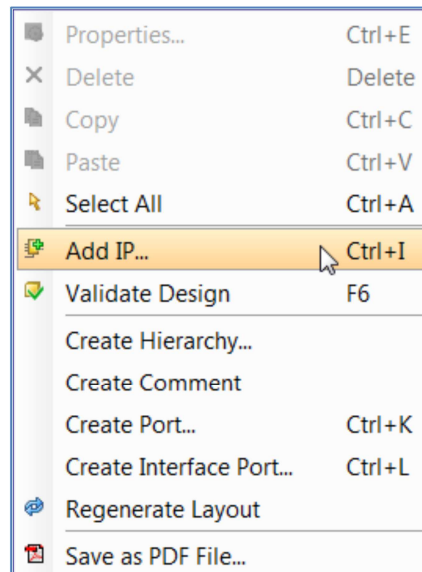


Figure 5: Add IP Option

4. Alternatively, you can click the **Add IP** link in the IP integrator diagram area.

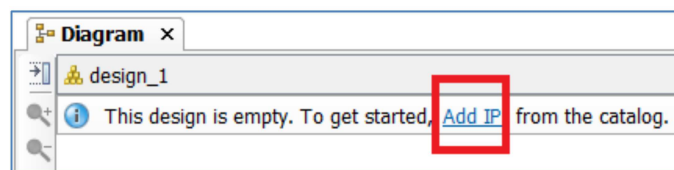


Figure 6: Add IP Link in IP Integrator Canvas

The IP Catalog opens.

5. In the search field, type **zynq** to find the ZYNQ7 Processing System IP (not the Zynq7 processing system BFM), and then press **Enter** on the keyboard.

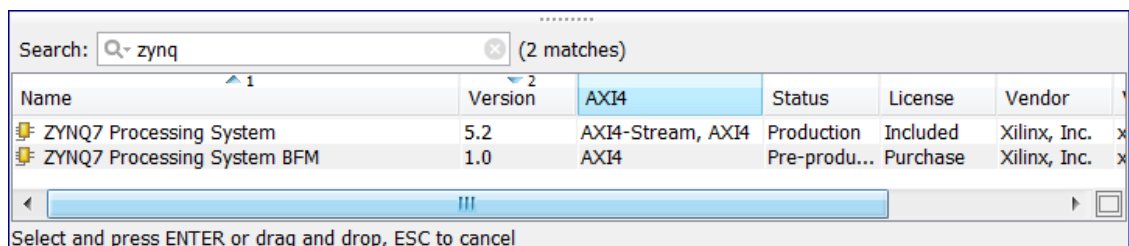


Figure 7: The IP Integrator IP Catalog

Because you selected the ZedBoard when you created the project, the Vivado IP integrator configures the design appropriately.

In the Tcl Console, you will see the following message:

```
“create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5  
processing_system7_0”
```

There is a corresponding Tcl command for all actions performed in the IP integrator block diagram. Those commands are not shown in this document. See the Tcl Console for information on those commands.

6. In the IP integrator diagram header, click **Run Block Automation**.

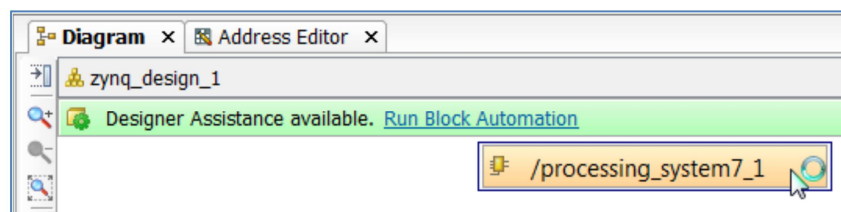


Figure 8: Run Block Automation on Zynq

The **Run Block Automation** dialog box opens, stating that the FIXED_IO and DDR interfaces will be created for the Zynq core.

7. Click **OK**.

After running block automation on the Zynq processor, the IP integrator diagram should look as shown in **Figure 9**.

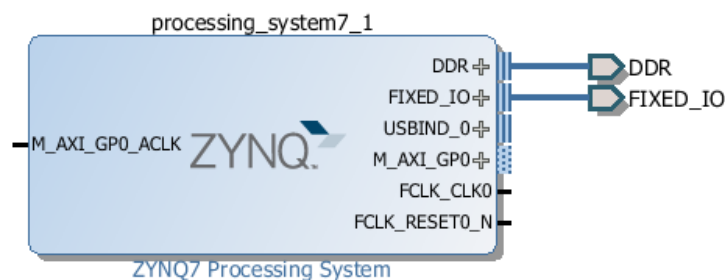


Figure 9: Zynq Processing System after Running Block Automation

8. We will now reconfigure the ZYNQ7 Processing System. **Double click** on the ZYNQ block diagram.
9. The Re-customize IP window will open as seen in **Figure 10**.

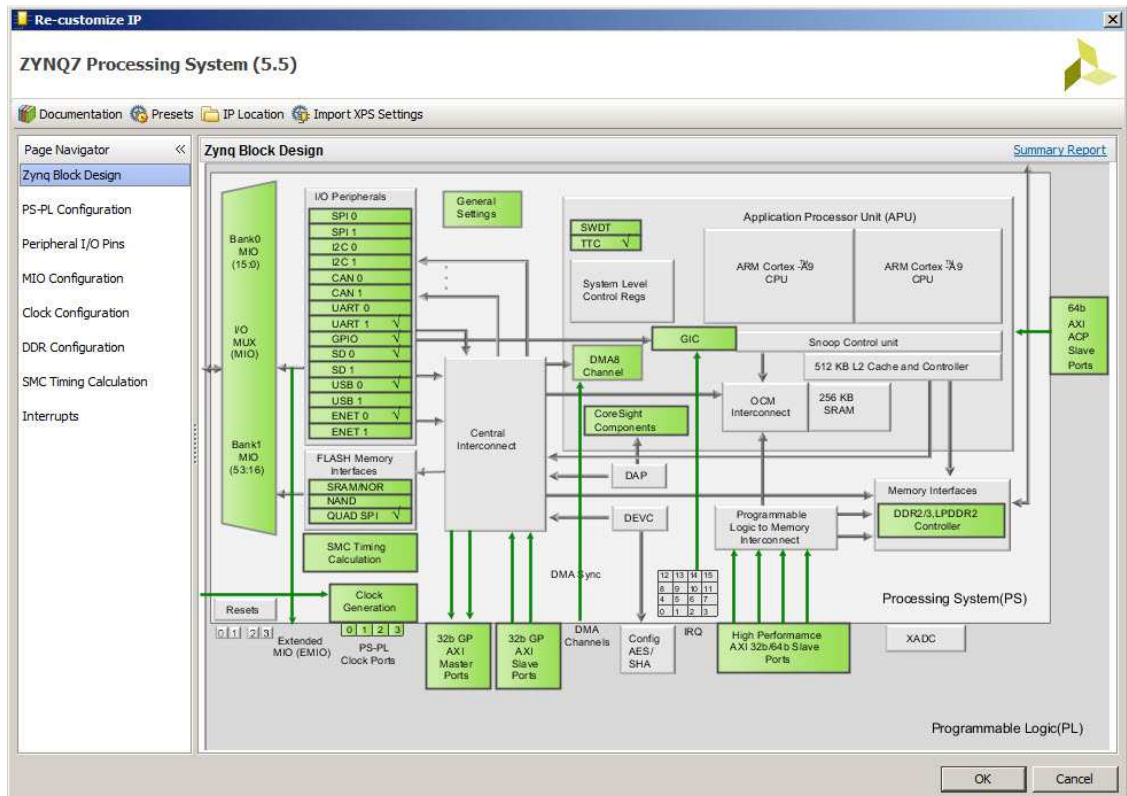


Figure 9: Re-customizing the ZYNQ Processing System

10. **Click** on the MIO Configuration panel to open its configuration form.
11. **Expand** the IO Peripherals on the right.
12. **Uncheck** ENET 0, USB 0, and SD 0, GPIO (GPIO MIO), leaving UART1 selected.
13. In the MIO Configuration panel, **expand** the Application Processing Unit and **uncheck** the Timer 0.
14. From the Page Navigator, **select** “Clock Configuration” and open the “PL Fabric Clocks” tree as seen in **Figure 11**.

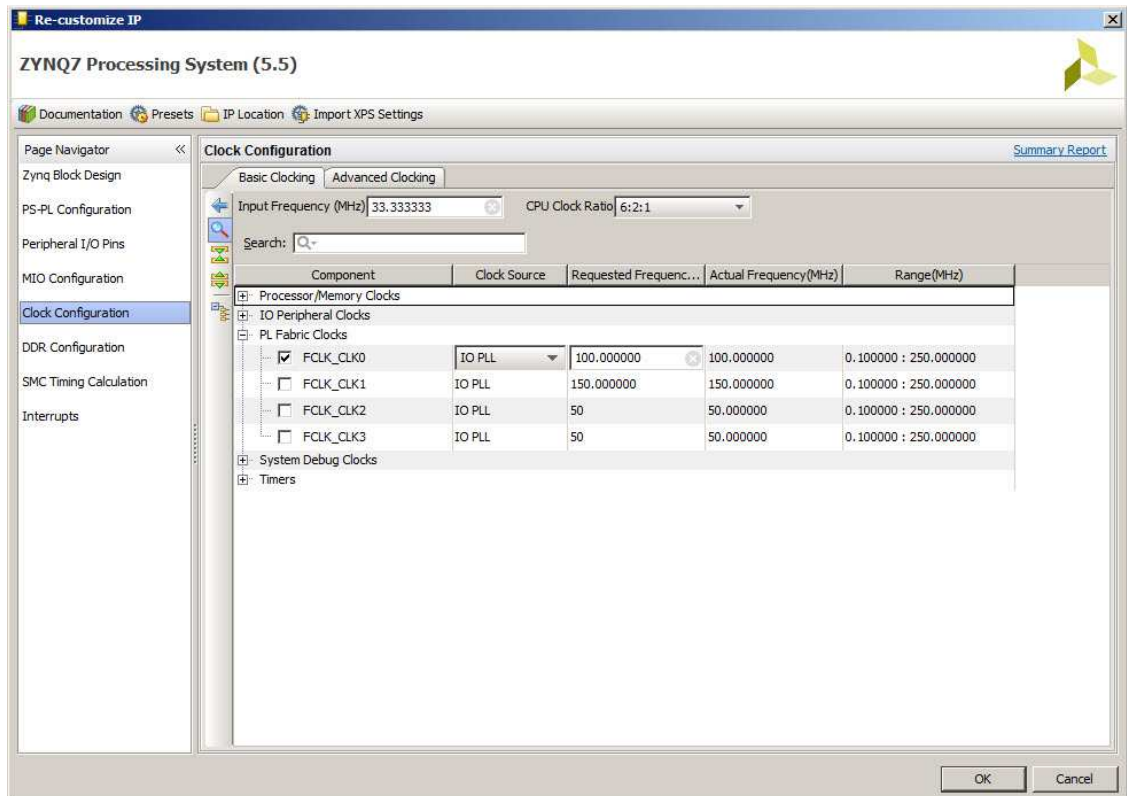


Figure 10: Clock Configuration

15. Make sure that the FCLK_CLK0 is enabled (ticked) and that it is set for a frequency of **100 MHZ**. This will be our AXI clock.
16. Now from the Page Navigator, **select** "PS-PL Configuration" and **open** the "GP Master AXI Interface" tree.
17. **Tick** the "M AXI GP0 interface" checkbox and enable it as seen in **Figure 12**.

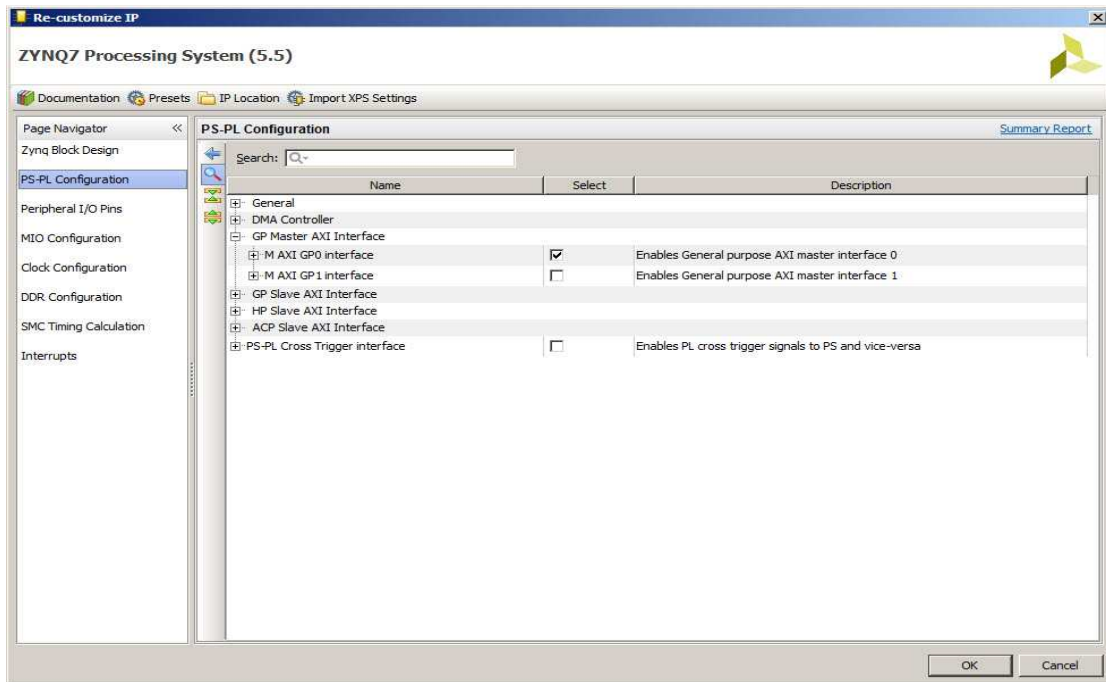


Figure 11: PS-PL Configuration

18. Now **click** “OK” to close the Re-customize IP window.
19. We must now **connect** the FCLK_CLK0 output to the AXI clock input. To do this, **click** on the FCLK_CLK0 output and then **click** on the M_AXI_GP0_ACLK input. This will trace a wire between the pins and make the connection as seen in **Figure 13**.



Figure 12: processing_system7_0 connection

Part 2: Create the Custom IP

Introduction

In this part of the tutorial you will create a custom IP by using the “Create and Package IP” facility in Vivado.

1. With the base Vivado project opened, from the menu select **Tools**→ **Create and package IP**. A new window will appear as seen in **Figure 14**.



Figure 13: Create and Package New IP

2. When the “Create and Package IP” wizard opens. **Click “Next”**.
3. On the next page (**Figure 15**), **select “Create a new AXI4 Peripheral**. **Click “Next”**.

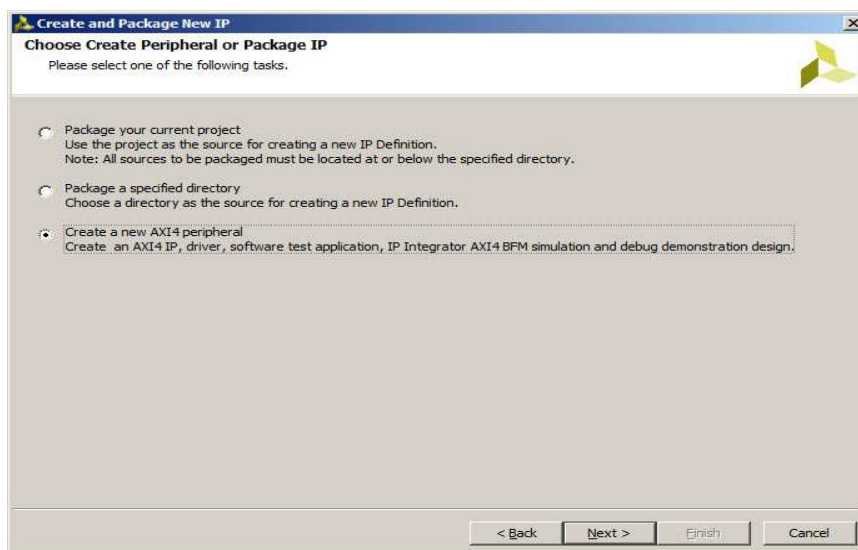


Figure 14: Create a new AXI4 Peripheral

- Now you can give the peripheral an appropriate name, description and location as seen in **Figure 16**. Click “Next”.

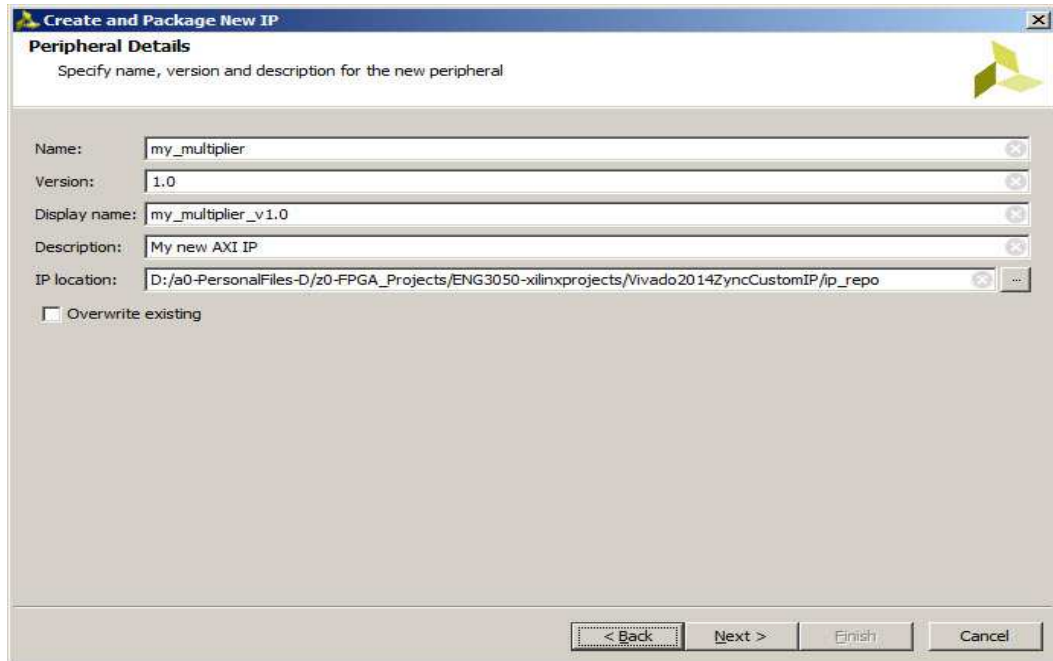


Figure 15: Peripheral Details

- On the next screen we can configure the AXI bus interface. For the multiplier we will use AXI lite, and it will be a slave to the PS, so we will stick to the default values shown on **Figure 17**.

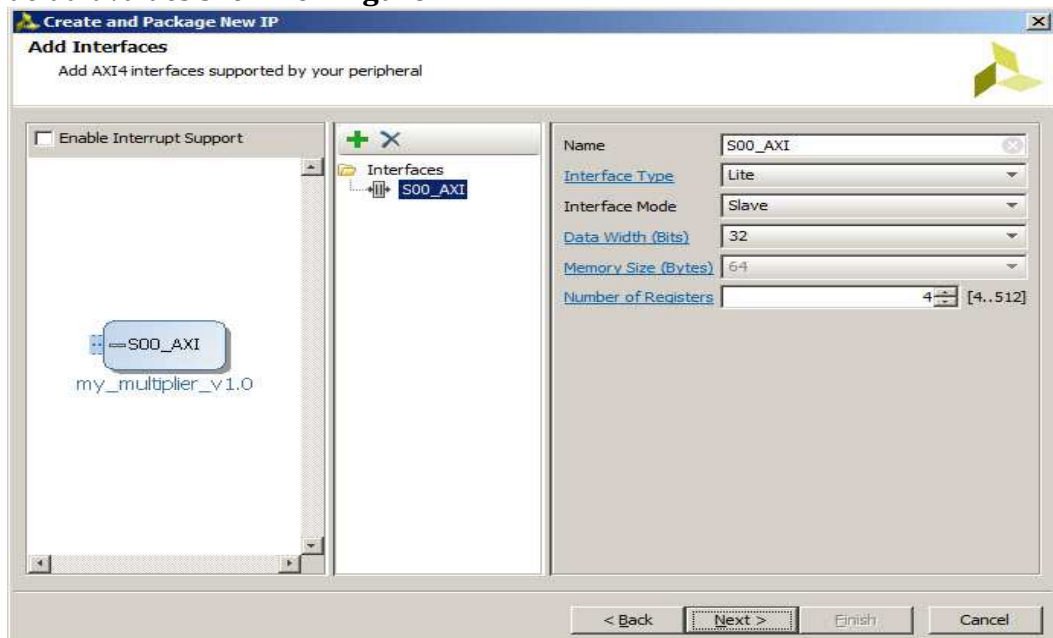


Figure 16: Add Interface

- On the last page, select “Edit IP” and click “Finish” as seen in **Figure 18**.

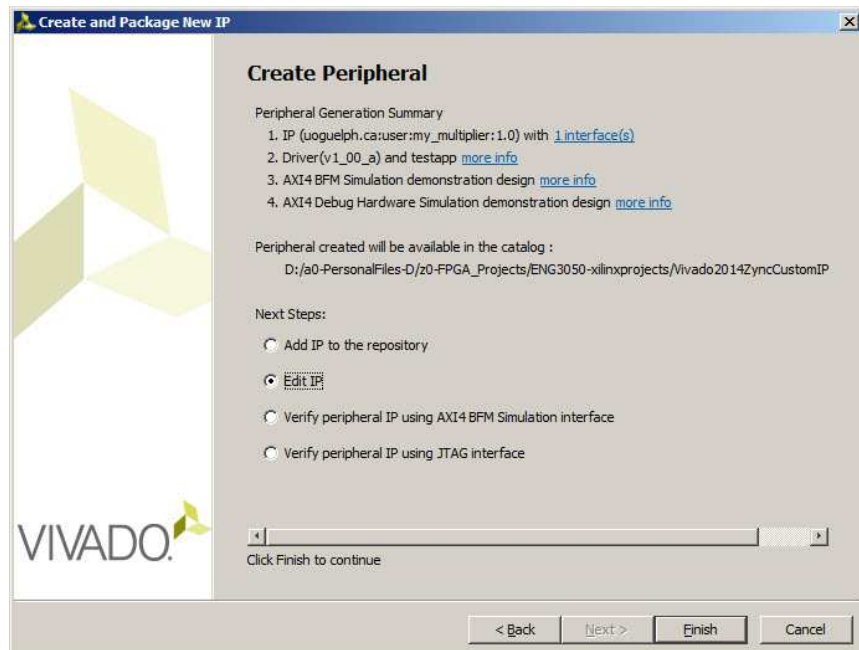


Figure 17: Create & Edit IP

- At this point, the peripheral that has been generated by Vivado is an AXI Lite Slave that contains 4 x 32 bit read/write registers (as seen in **Figure 19**). We want to add our multiplier code to the IP and modify it so that one of the registers connects to the multiplier inputs and another to the output.

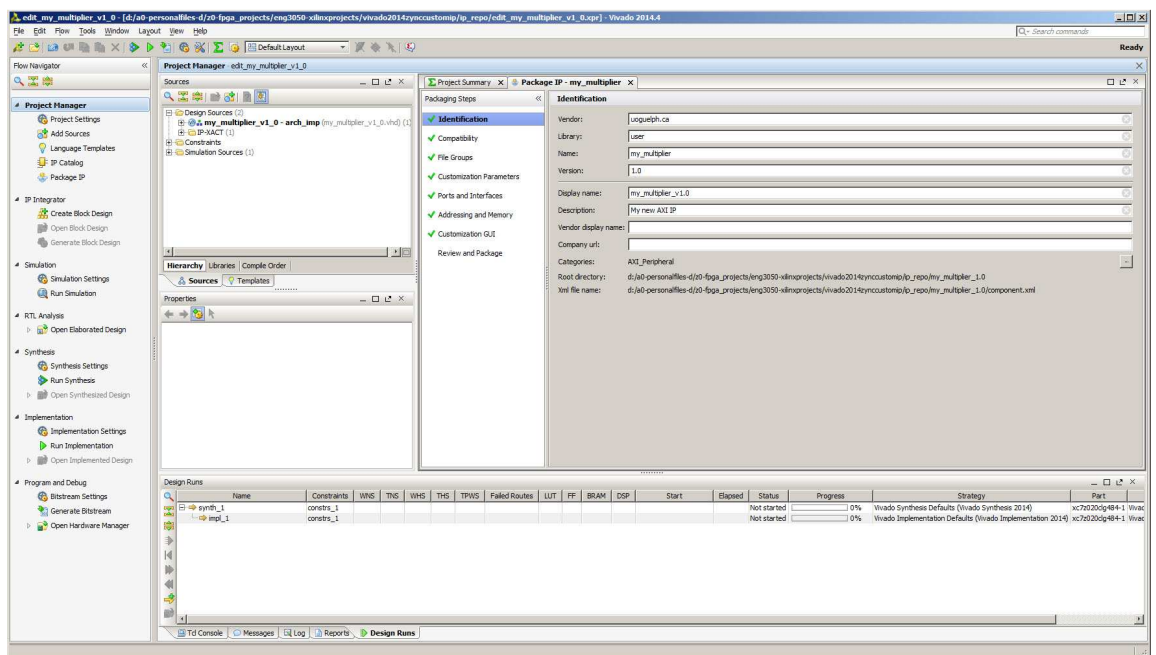


Figure 19: Summary of IP

Add the multiplier code to the peripheral

You can find the multiplier code on the web site of ENG3050. Download the “multiplier.vhd” file and save it somewhere, the location is not important for now. Note that these steps must be done in the Vivado window that contains the peripheral we just created (not the base project that contains the PS).

1. From the Flow navigator, click “Add Sources”. In the window that appears (**Figure 20**) select “Add or Create Design Sources” and click “Next”.



Figure 20: Add Sources

2. On the next window (**Figure 21**), click “Add Files”

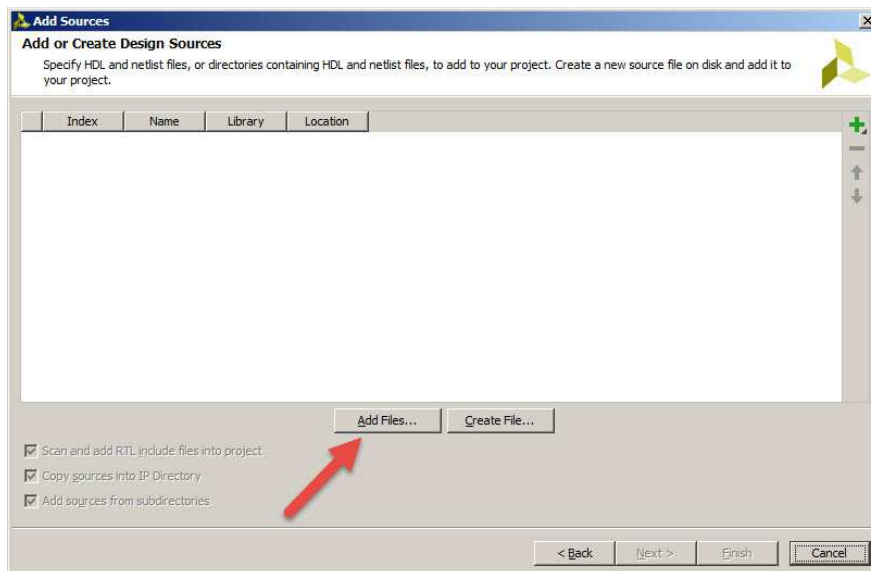


Figure 21: Add Files

3. **Browse to** the “multiplier.vhd” file, select it and **click** “OK”.
4. Make sure you **tick** “Copy sources into IP directory” and the **click** “Finish” as seen in **Figure 22**.

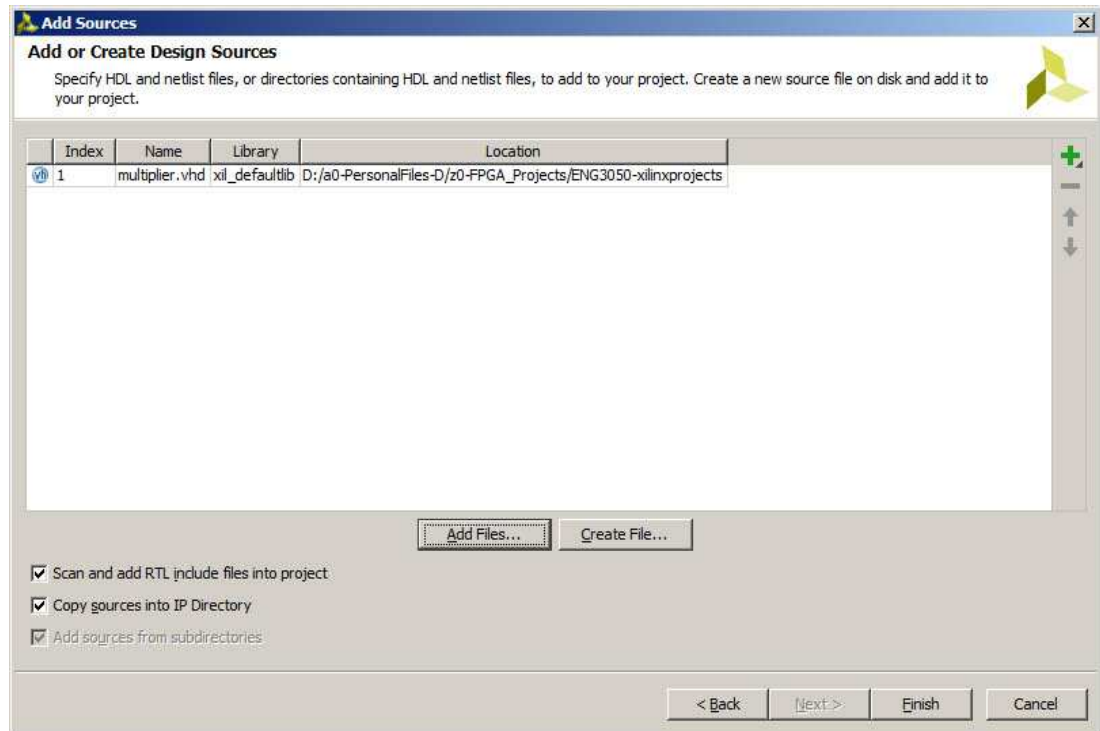


Figure 22: Choosing the VHDL Code

5. The multiplier code is now added to the peripheral; however **we still have to** instantiate it and connect it to the registers.

Modify the Peripheral

At this point, your Project Manager Sources window should look like the following **Figure 23**.

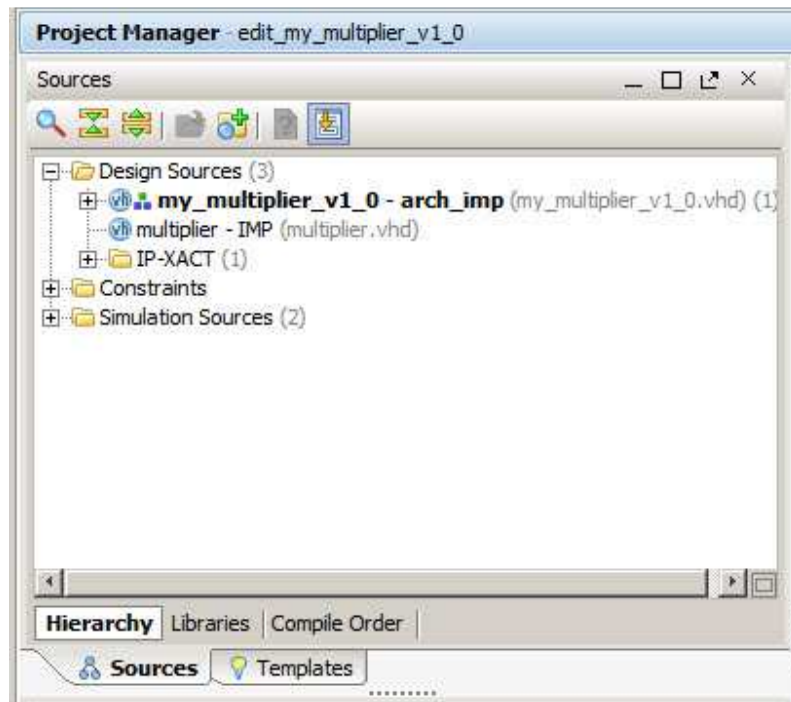


Figure 23: Project Manager

1. **Open** the branch “my_multiplier_v1_0-arch_imp”
2. **Double click** on the “my_multiplier_v1_0_S00_AXI_INST” file to open it.
3. The source file should be open in Vivado. **Find** the line with the “begin” keyword and add the following code just above it to declare the multiplier and output signal:

```
signal multiplier_out : std_logic_vector(31 downto 0);
```

```
component multiplier  
port (  
    clk: in std_logic;  
    a: in std_logic_VECTOR(15 downto 0);  
    b: in std_logic_VECTOR(15 downto 0);  
    p: out std_logic_VECTOR(31 downto 0));  
end component;
```


4. Now **find** the line that says “ – Add user logic here” and **add** the following code below it to instantiate the multiplier:

```
390    -- Add user logic here
391    multiplier_0 : multiplier
392    port map (
393        clk => S_AXI_ACLK,
394        a => slv_reg0(31 downto 16),
395        b => slv_reg0(15 downto 0),
396        p => multiplier_out);
397    -- User logic ends
```

5. **Find** this line of code “reg_data_out <= slv_reg1”; and **replace** it with “reg_data_out <= multiplier_out”.
6. In the process statement just a few lines above, **replace** “slv_reg1” with “multiplier_out”.
7. **Save the file**
8. You should notice that the multiplier.vhd” file has been integrated into the hierarchy (as seen in **Figure 24**) because we have instantiated it from within the peripheral.

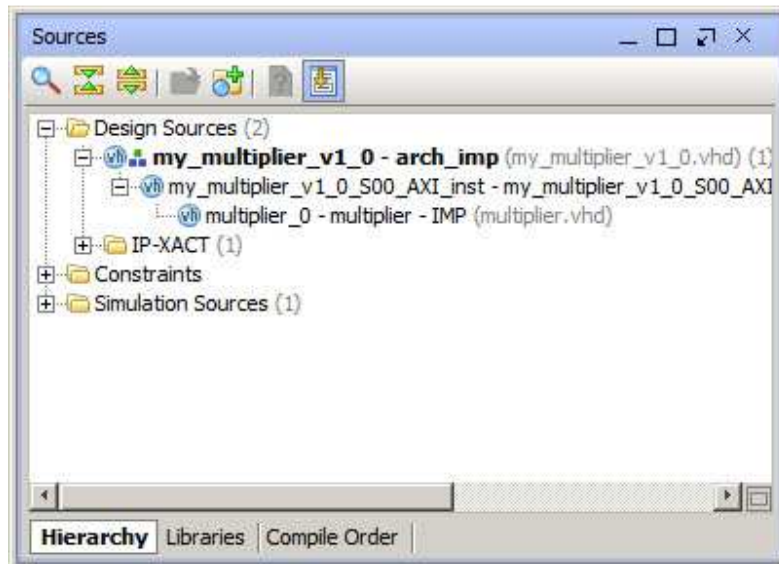


Figure 24: Sources Hierarchy

9. **Click** on “IP File Groups” in the Package IP tab of the Project Manager.

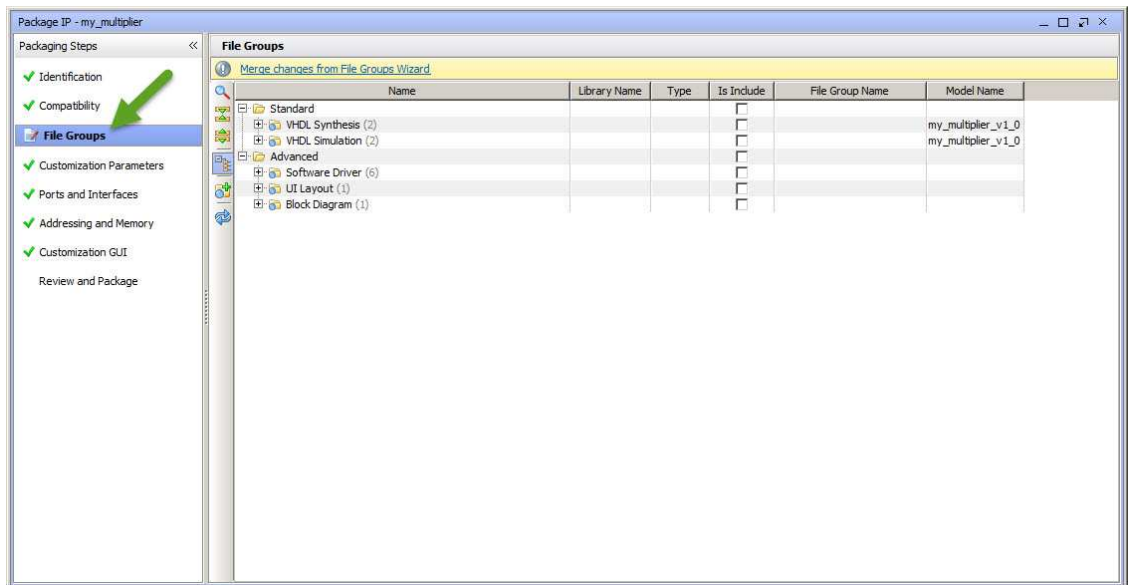


Figure 25: IP Groups

10. **Click** the “Merge changes from IP File Group Wizard” link.

11. The “IP File Groups” should now have a tick.

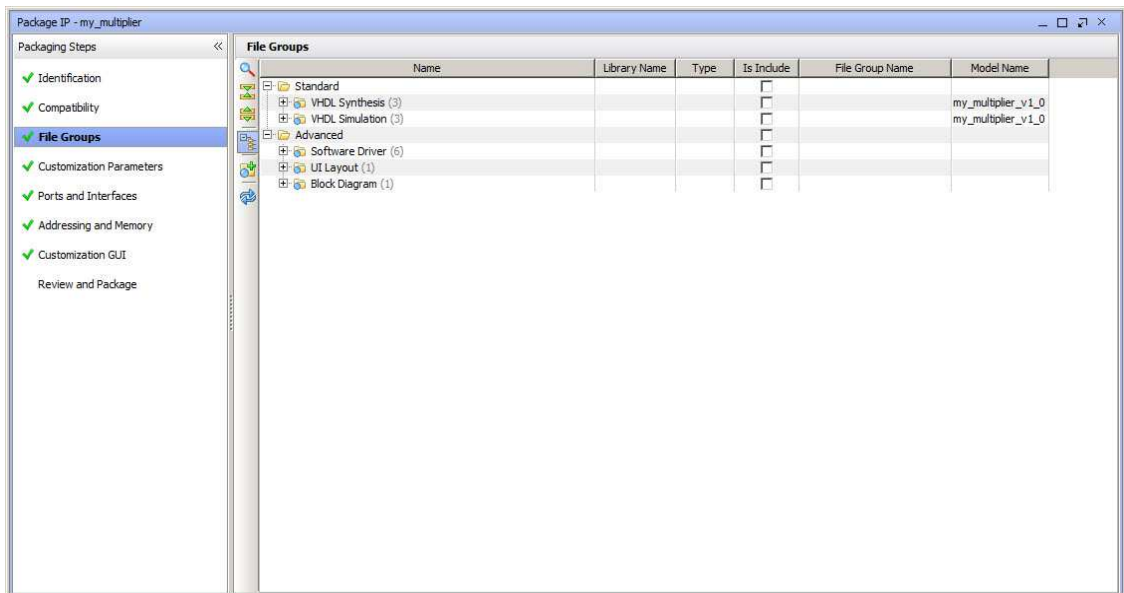


Figure 26: IP File Groups

12. Now **Click** “Review and Package IP” as seen in **Figure 27** and then **click** RePackage IP.

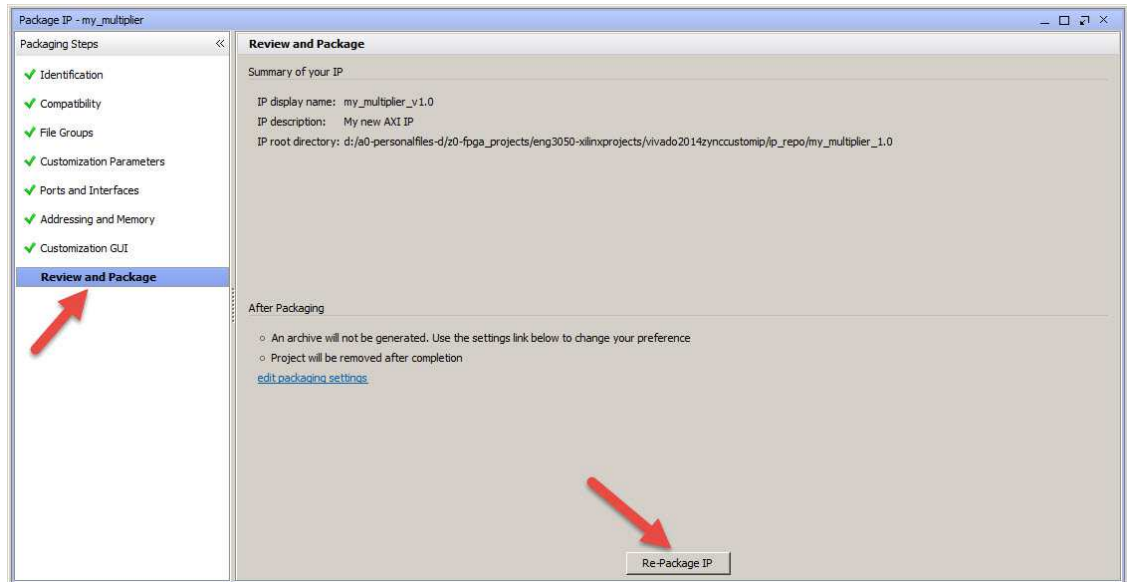


Figure 27: Review and Package IP

13. A final window will appear as seen in **Figure 28**. Press “OK”

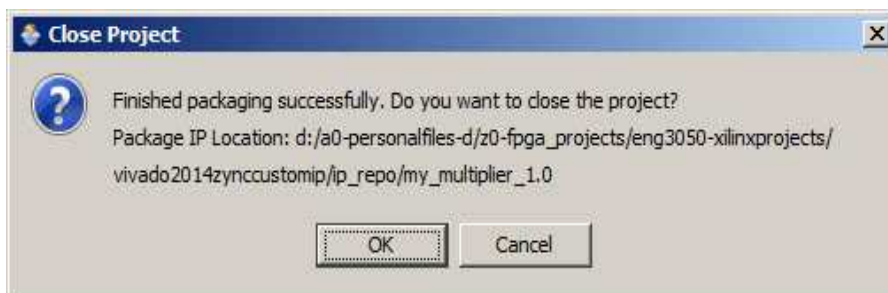


Figure 28: Close the Project

The peripheral will be packaged and the Vivado window for the peripheral should be automatically closed. We should now be able to find our IP in the IP catalog. Now the rest of this tutorial will be done from the original Vivado window.

Add the IP to the Design

1. Click the “Add IP” icon
2. Find the “my_multiplier” IP as seen in **Figure 29** and **double click** it.



Figure 29: Search for my_multiplier

3. The block should appear in the block diagram (**Figure 30**) and you should see the message “Designer Assistance available”. Click on “Run Connection Automation”.

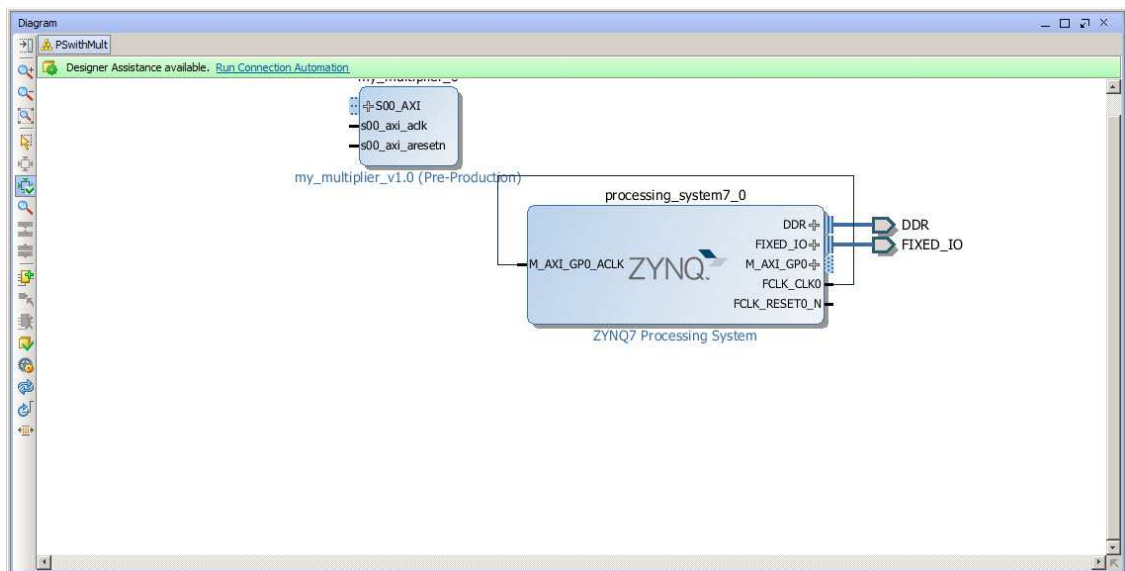


Figure 30: Run Connection Automation

- In the window that appears (**Figure 31**), set Clock connection to “Auto” and click “OK”.



Figure 31: Run Connection Automation (Auto)

- The new block diagram should like **Figure 32**.

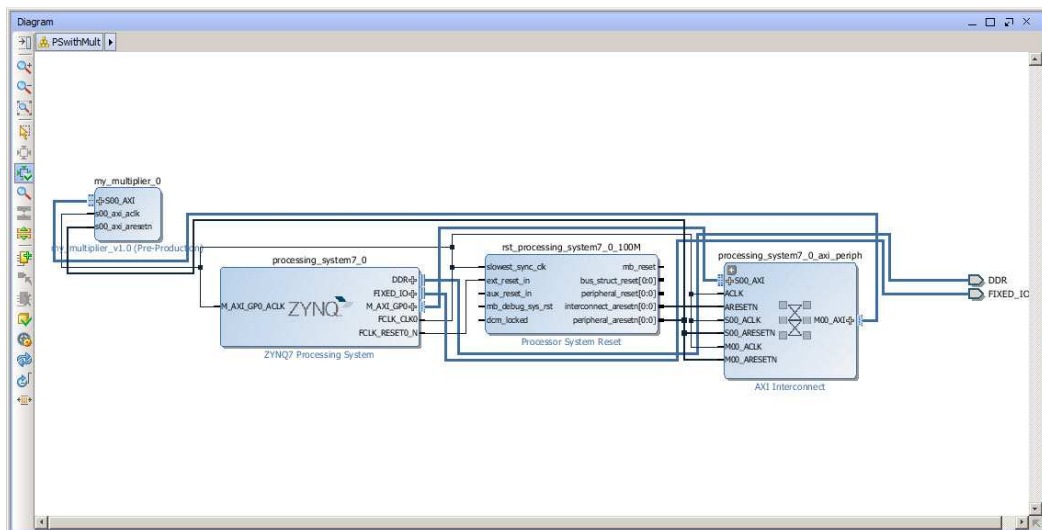


Figure 32: The Final Block Diagram

- Validate the design (**choose “Tools → Validate Design”**).
- If all goes well, your design will be validated and you will get a message that says “validation successful”.

Generate HDL Design Files

You now generate the HDL files for the design.

1. In the Source window, **right-click** the top-level subsystem design and **select** “Generate Output Products” (**Figure 33**). This generates the source files for the IP used in the block diagram and the relevant constraints file.

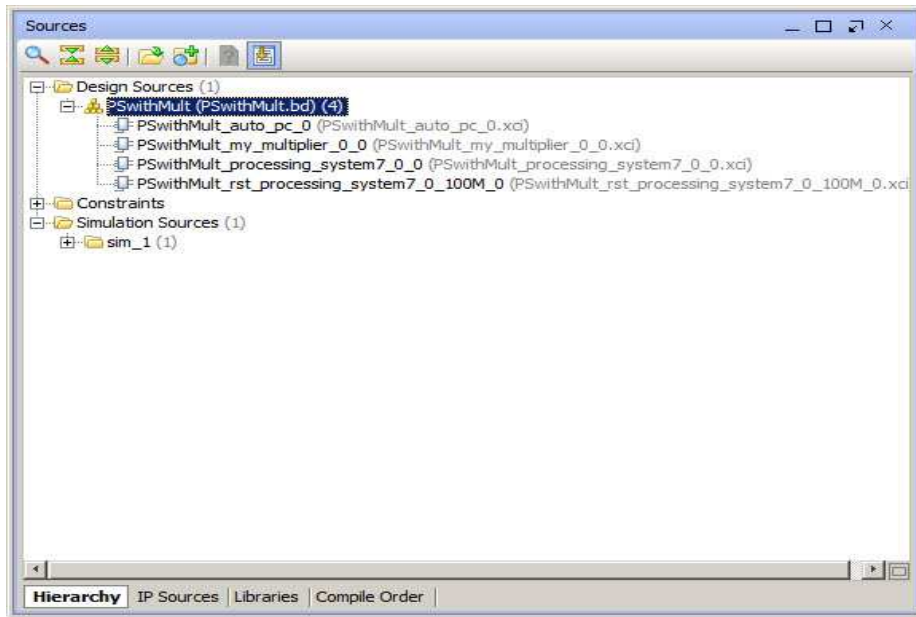


Figure 33: Sources Window

2. The **Generate Output Products** dialog box opens (Figure 34). Click “Generate”.

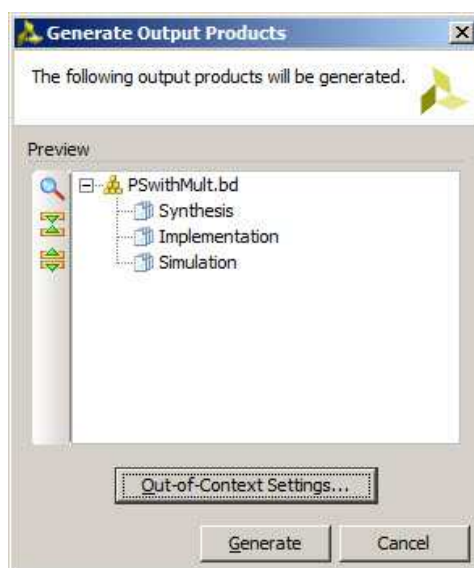


Figure 34: Generate Output Products Option

3. In the Sources window, select the top-level subsystem source, and select **Create HDL Wrapper** to create an example top-level HDL file (**Figure 35**).
4. Click **OK** when the **Create HDL Wrapper** dialog box opens.

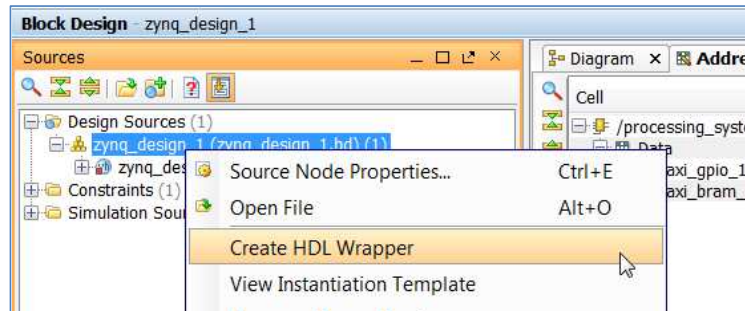


Figure 35: Create HDL Wrapper

Implement Design and Generate Bitstream

1. In Flow Navigator, click **"Generate Bitstream"** to implement the design and generate a BIT file.

Note: If the system requests to re-synthesize the design before implementing, click **No**. The previous step of saving the constraints caused the flow to mark synthesis out-of-date. Ordinarily, you might want to re-synthesize the design if you manually changed the constraints, but for this tutorial, it is safe to ignore this condition (**Figure 36**).

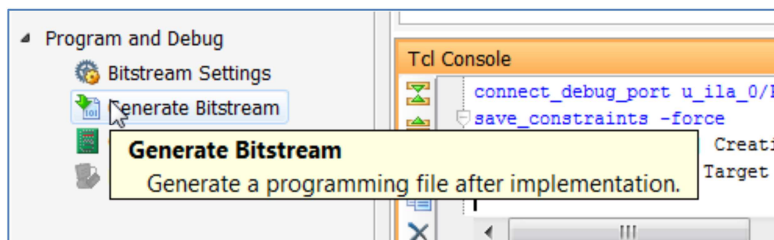


Figure 36: Generate Bitstream

You might see a dialog box stating no implementation results are available.

2. Click **"Yes"**.

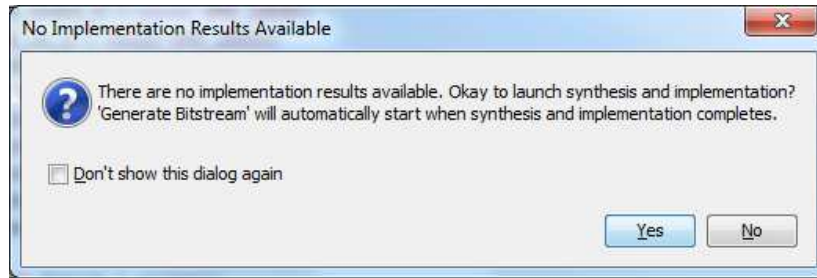


Figure 37: No Implementation Results Available Dialog Box

3. After the design implementation, **click “Open Implemented Design”, (Figure 38). Press “ok”**

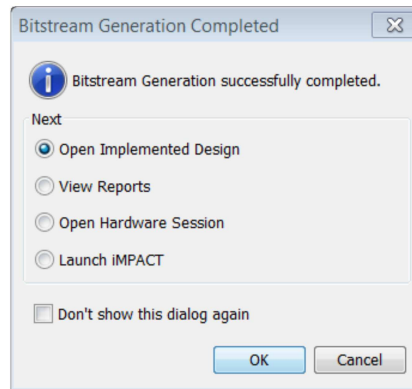


Figure 38: Bitstream Generation Completed

4. You might get a warning that the implementation is out of date. Click “Yes”.



Figure 39: Implementation Is Out-of-Date Dialog Box

Export Hardware to SDK

In this step, you export the hardware description to SDK. You use this in Part 2. The IP integrator block diagram, and the Implemented design, must be open to export the design to SDK.

Export to SDK

1. In the Flow Navigator, **click** “Open Block” to invoke the IP integrator design (Figure 40).



Figure 40: IP Integrator - Open Block Design

Now you are ready to export your design to SDK.

2. From the main Vivado File menu, **select** “Export → Export Hardware” (Figure 41).

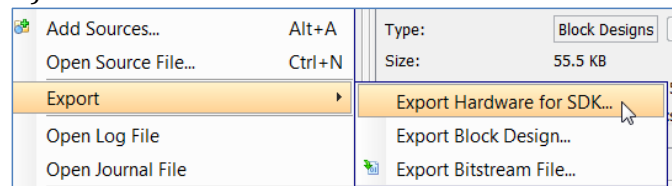


Figure 41: Export Hardware for SDK

The Export Hardware for SDK dialog box opens, **ensure** that Export Hardware, Include Bitstream, is checked (Figure 42).

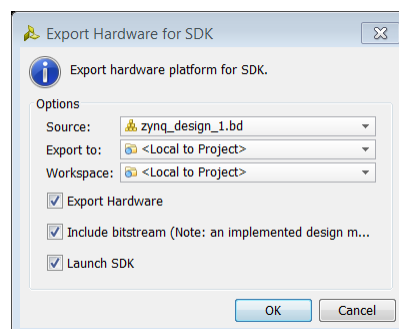


Figure 42: Export Hardware for SDK

3. From the main Vivado File Menu, **select** “Launch SDK”. The SDK will Launch in a new window.

Part 3: Build Zynq-7000 Processor Software

In this portion of the tutorial you will test the multiplier by printing results to the terminal via the UART (serial port).

Step 1: Start SDK and Create a Software Application

1. If you are doing this lab as a continuation of Part 2 then SDK should have launched in a separate window.
2. From the SDK window **Select “File > New > Application Project” (Figure 43).**

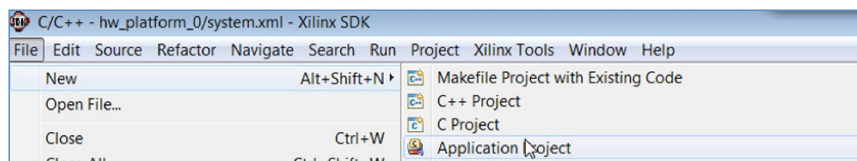


Figure 43: File->New->Application Project

A “New Project” dialog box opens. Here you will enter the name of the project

3. In the Project Name field, **type “TestMultiplier”, and click “Next” (Figure 44).**

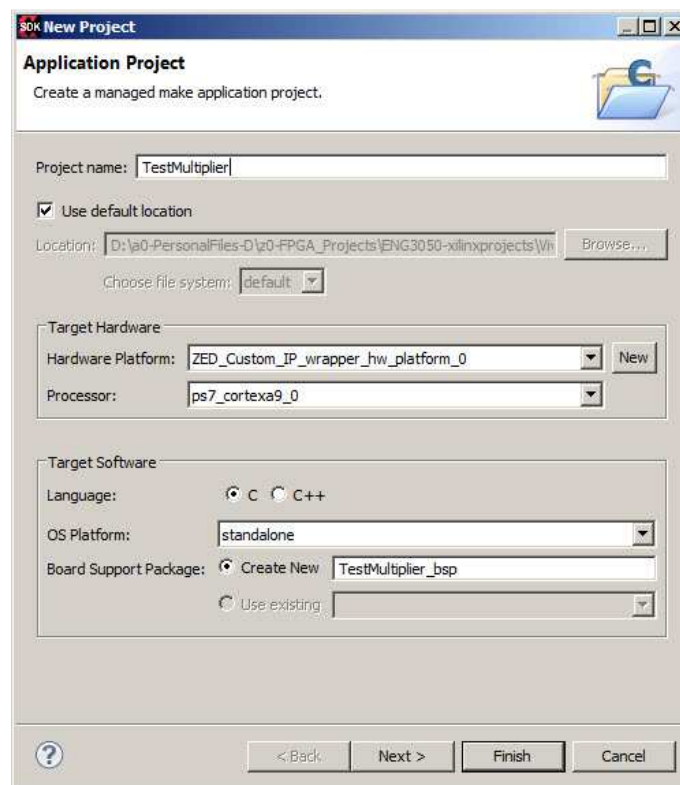


Figure 44: SDK Application Project

4. From the Available Templates, **select** “Hello World” as seen in (Figure 45) and **click** “Finish”.

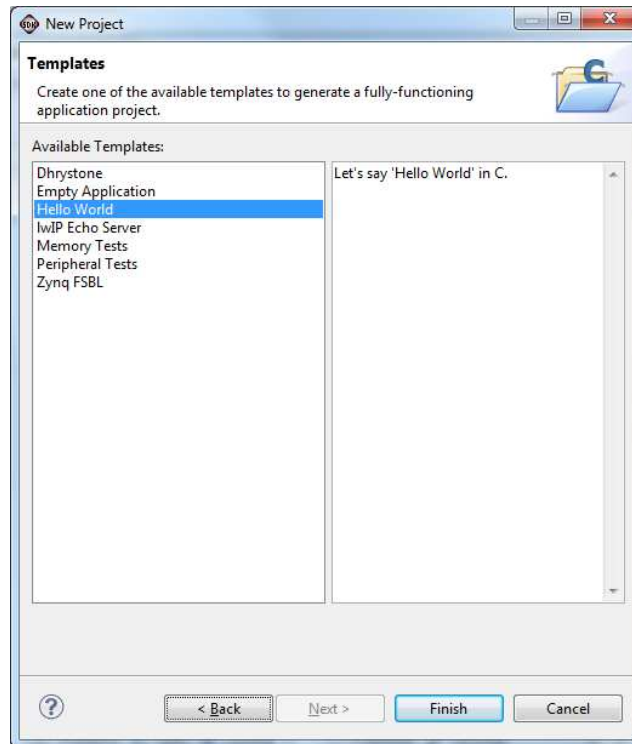


Figure 45: SDK New Project Template

When the program finishes compiling, you will see the following message on the console (Figure 46).

```
Overview | Source
Problems | Tasks | Console | Properties | Terminal
CDT Build Console [Zynq_Design]

Invoking: ARM Print Size
arm-xilinx-eabi-size Zynq_Design.elf |tee "Zynq_Design.elf.size"
  text  data   bss   dec   hex filename
 74740  2024   33700 110464 1af80 Zynq_Design.elf
Finished building: Zynq_Design.elf.size

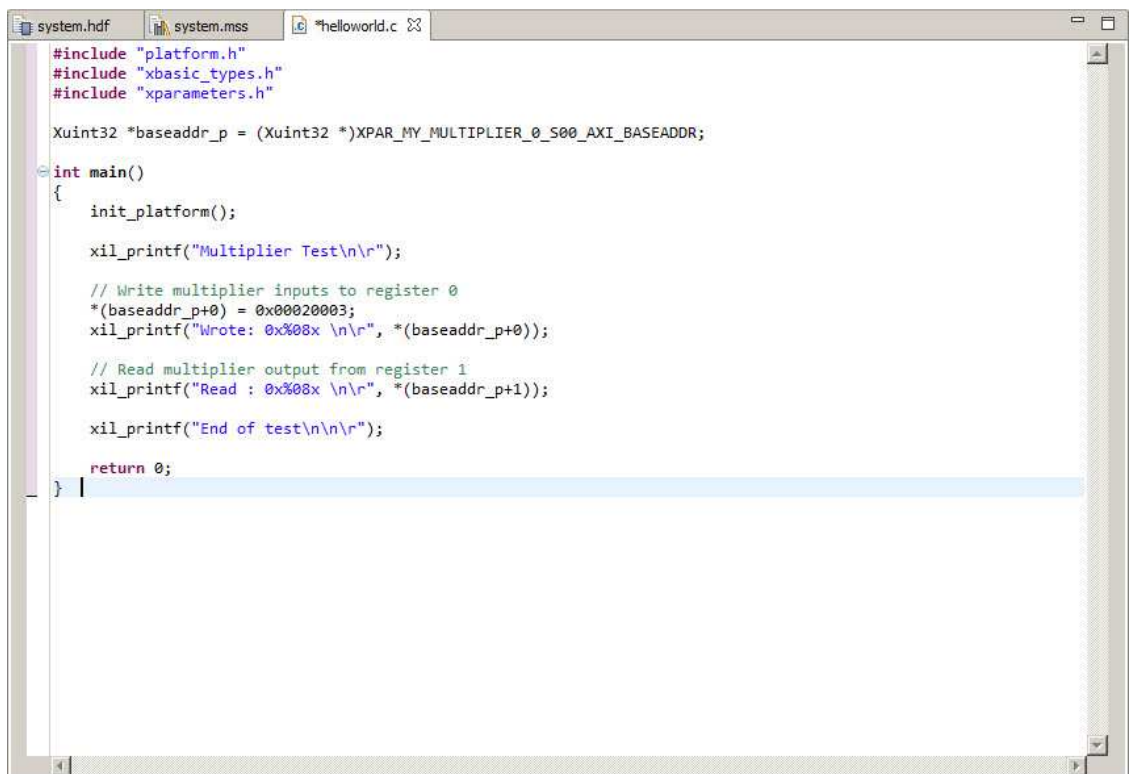
16:53:10 Build Finished (took 4s.976ms)
```

Figure 46: SDK Message

Step 2: Modify the Software Application

Now, you can either run the hello world application on the ZedBoard or test the multiplier!!. If you want to test the multiplier then you need to modify the software application.

1. From the Project Explorer, open the “TestMultiplier/src” folder. Open the “helloworld.c” source file.
2. Replace all the code in this file with the following code shown in Figure 47 (available on the webpage)



```
system.hdf | system.mss | *helloworld.c ✖
#include "platform.h"
#include "xbasic_types.h"
#include "xparameters.h"

Xuint32 *baseaddr_p = (Xuint32 *)XPAR_MY_MULTIPLIER_0_S00_AXI_BASEADDR;

int main()
{
    init_platform();

    xil_printf("Multiplier Test\n\r");

    // Write multiplier inputs to register 0
    *(baseaddr_p+0) = 0x00020003;
    xil_printf("Wrote: 0x%08x \n\r", *(baseaddr_p+0));

    // Read multiplier output from register 1
    xil_printf("Read : 0x%08x \n\r", *(baseaddr_p+1));

    xil_printf("End of test\n\n\r");

    return 0;
}
```

Figure 47: SDK Message

Step 3: Run the code on the FPGA

1. Download the bitstream into the FPGA by **selecting** “Xilinx Tools > Program FPGA” (FIGURE 48).

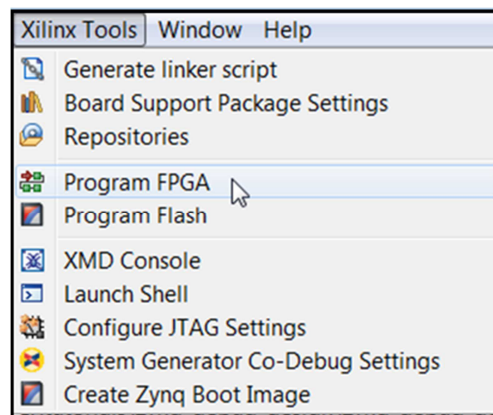


Figure 48: Program FPGA

This opens the Program FPGA dialog box.

2. A new window will appear (Figure 49)

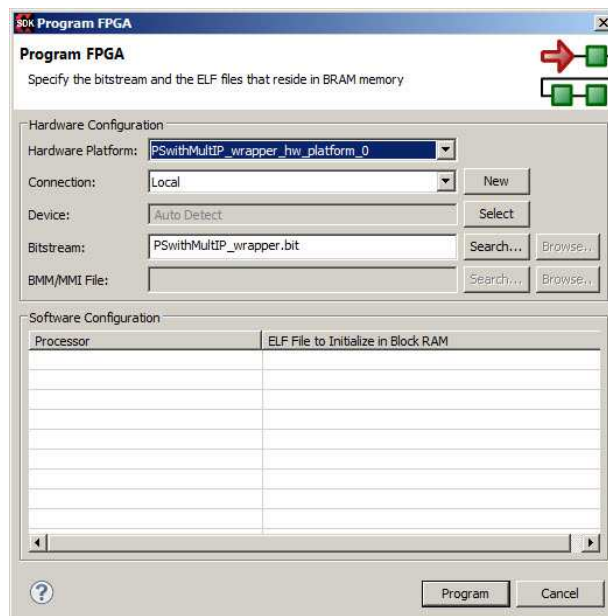


Figure 49: Program FPGA Window

3. Ensure that the path to the bitstream that you created is correct and then **click** “Program”.

Note: The DONE LED on the board turns blue if the programming is successful.

4. **Connect** a terminal to see the results (check previous tutorials of how to connect a terminal).

5. **Select** “Run → Run Configurations”. A new window will appear as seen in **Figure 50**.

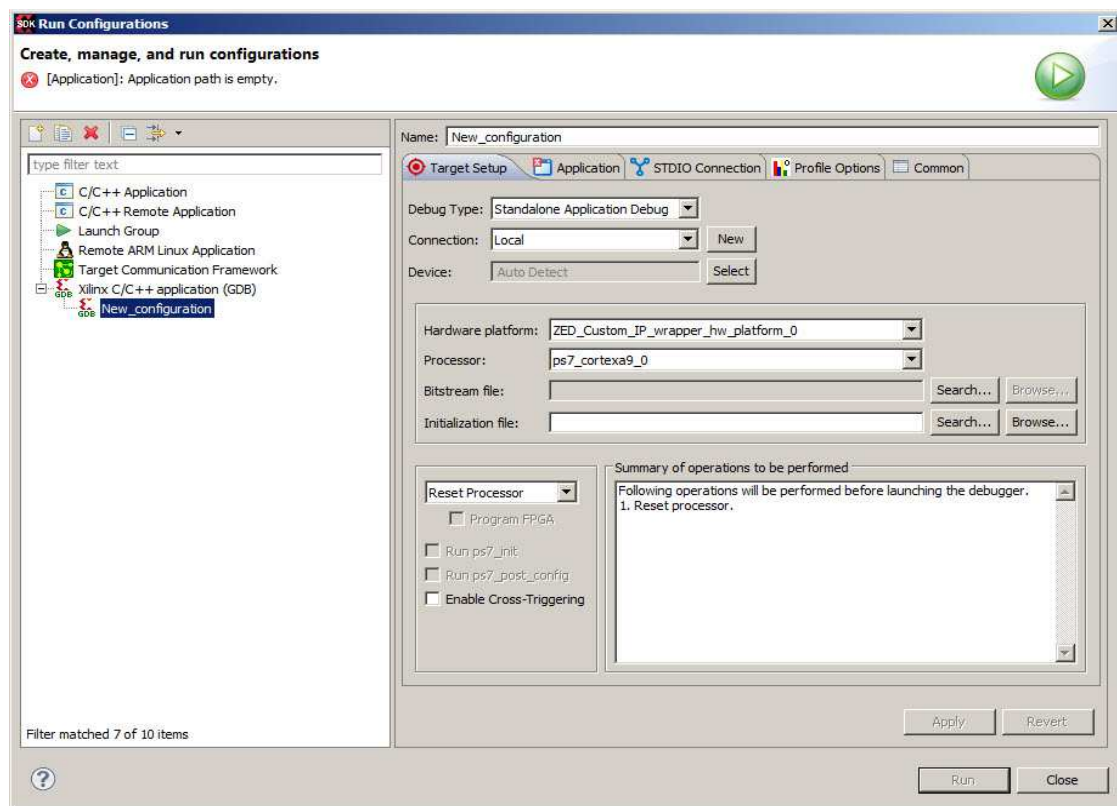


Figure 50: Run Configuration

6. **Double Click** on Xilinx C/C++ application (GDB) and a new configuration will be created “New Configuration” along with its setting menu as seen in **Figure 50**.

7. Make sure you have an application associated with your run configuration as seen in **Figure 51**.

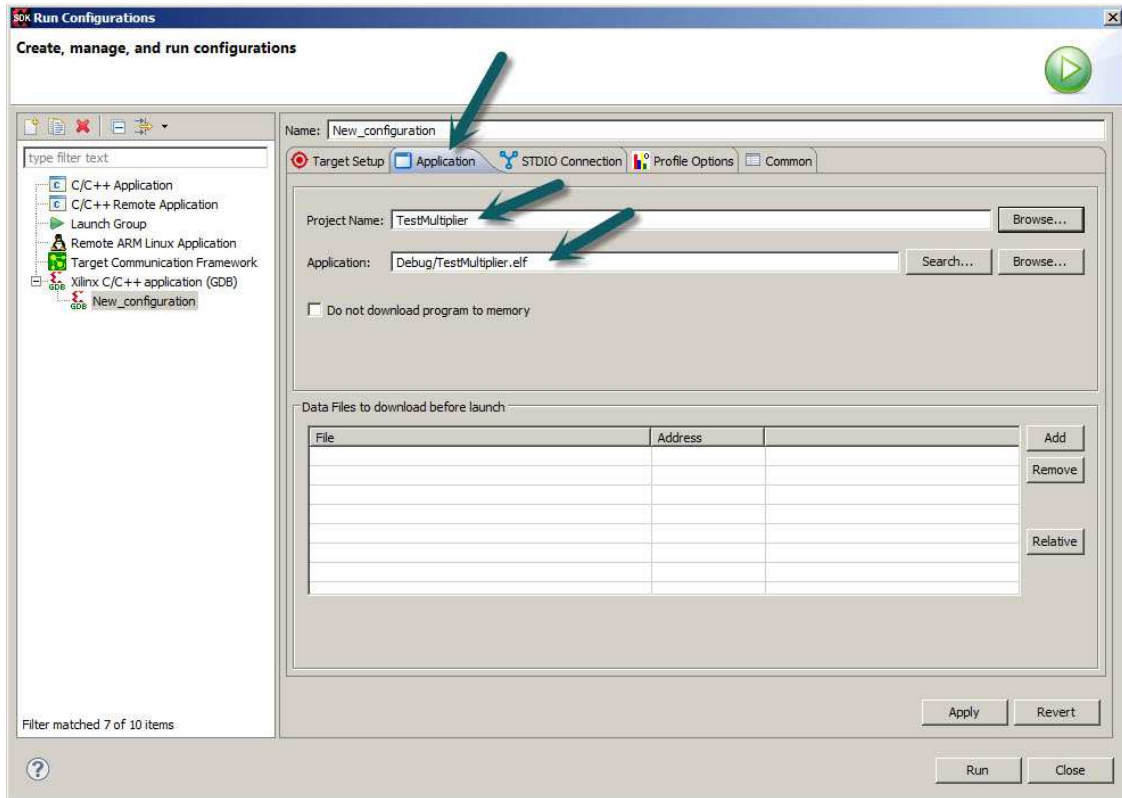


Figure 51: Application associated with your run configuration

8. Click on Run.

9. You will see the following results on the terminal (**Figure 52**)

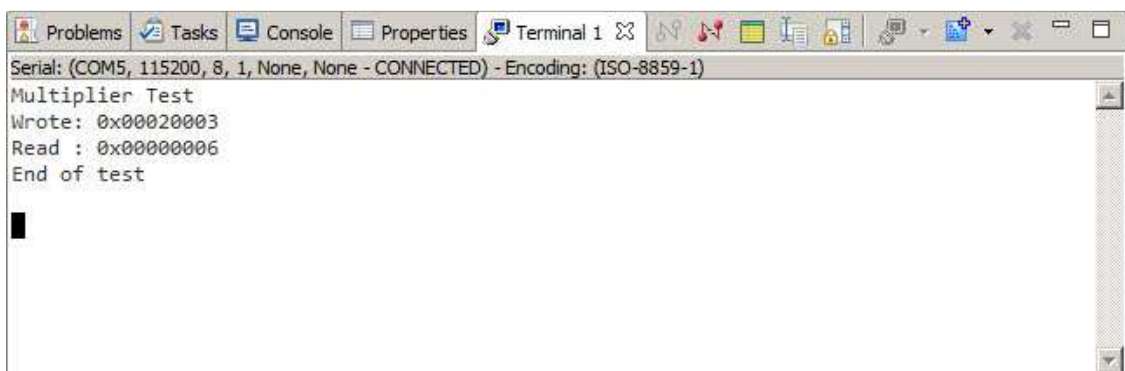


Figure 52: Results of Run Configuration