# **Creating a custom IP block in Vivado Using ZedBoard**

Prepared by

**Binit Kumar Pandit**

# Introduction

In this lab we have created a custom AXI IP block of Multiplier in Vivado and modified its functionality by integrating custom VHDL code. We have used the Zync SoC and ZedBoard as a hardware platform. In this case, 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 our own code into an AXI IP block.

# Create a Vivado Project

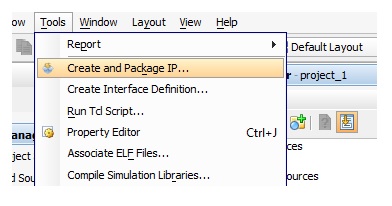
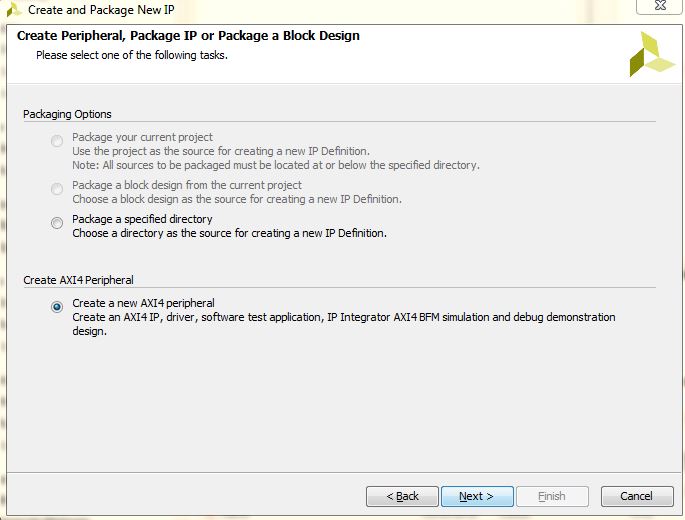
1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2015.4 > Vivado 2015.4.**
2. Click **Create New Project** to start the wizard. We will see the Create a New Vivado Project dialog box. Click **Next**.
3. Enter **Project name** and **Project location**. Click **Next**.
4. In the Project Type form select **RTL Project**, and click **Next**.
5. Click Boards. Search for **Zedboard Zynq Evaluation and Development Kit** and Select it. Click Next. Click **Finish.**
6. With the Vivado project opened, from the menu select **Tools > Create and package IP.**

Figure 1: Invoking New IP Generation

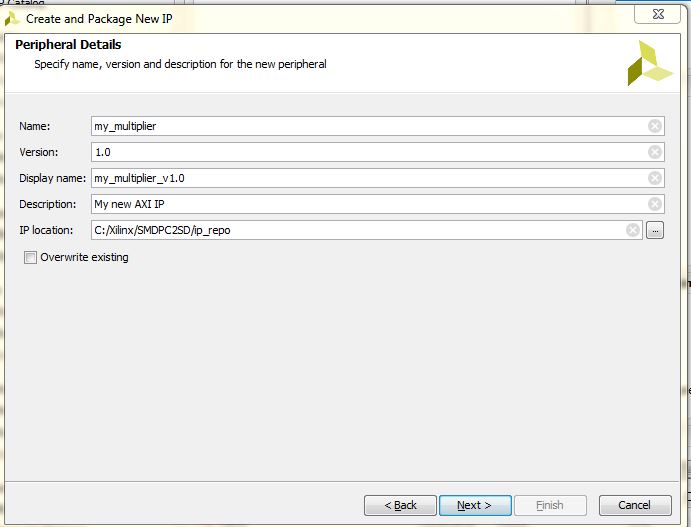
Click **Next.**

Figure 2: Create and Package IP



1. On the next page, select “**Create a new AXI4 peripheral**”. Click “**Next**”.

Figure 3: Create a new AXI4 Peripheral

1. Now we can give the peripheral an appropriate **name, description and location**. Click “**Next**”.

### 

### Figure 4: IP Name, description and location

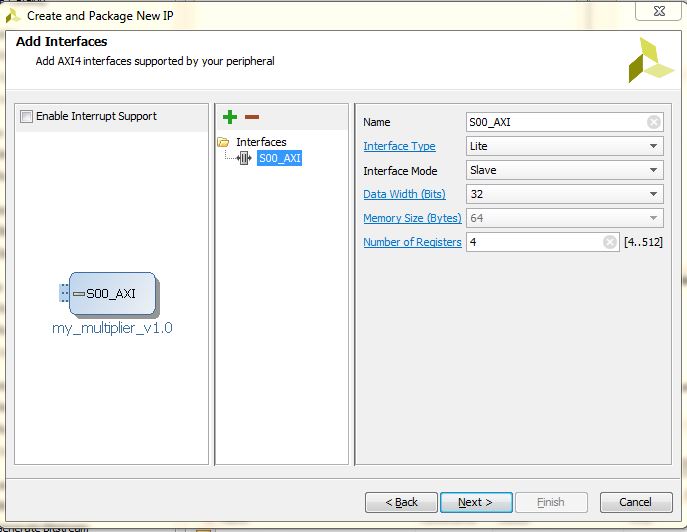
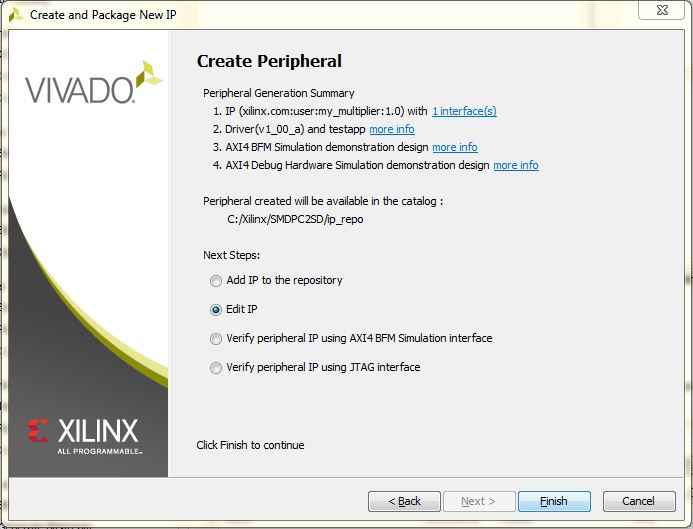
1. On the next page we can configure the AXI bus interface. For the multiplier we’ll use **AXI lite**, and it’ll be a **slave to the PS**, so we’ll stick with the **default** values.

Figure 5: Configuration of AXI Bus Interface

1. On the last page, select “**Edit IP**” and click “**Finish”.** Another Vivado window will open which will allow us to modify the peripheral that we just created.

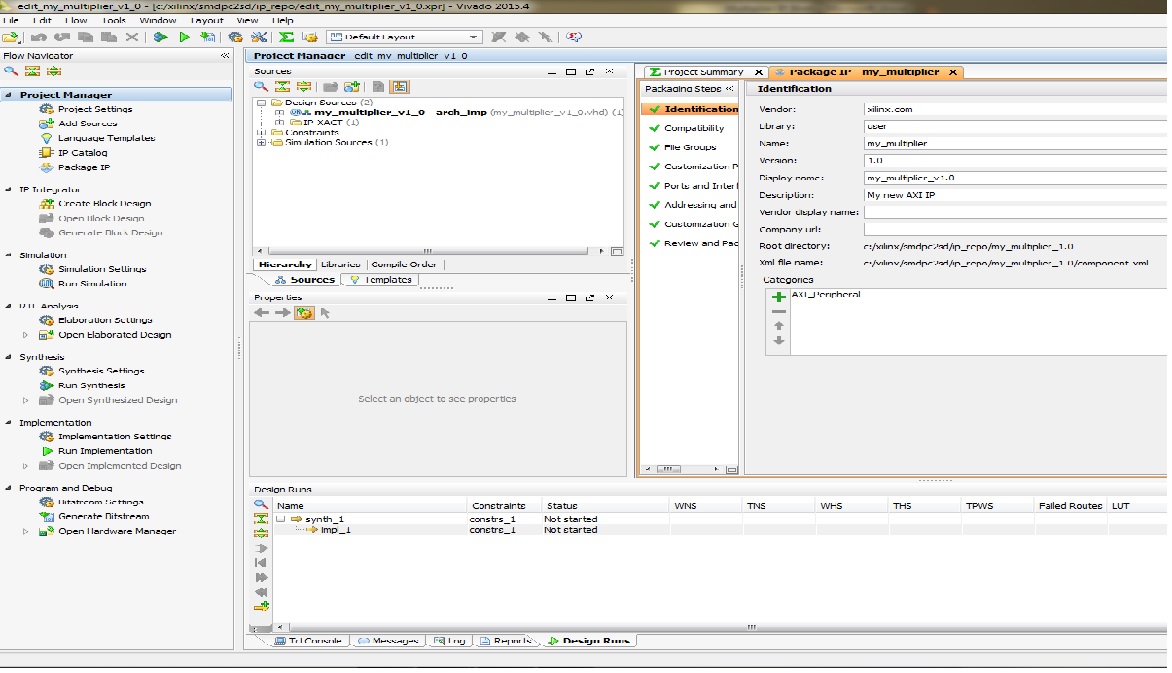
Figure 6: IP Creation

Figure 7: IP Window

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. We will 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 multiplier output.

# Add the multiplier code to the peripheral.

1. From the Flow Navigator, click “**Add Sources**”. In the window that appears, select “**Add or Create Design Sources**” and click “**Next**”.

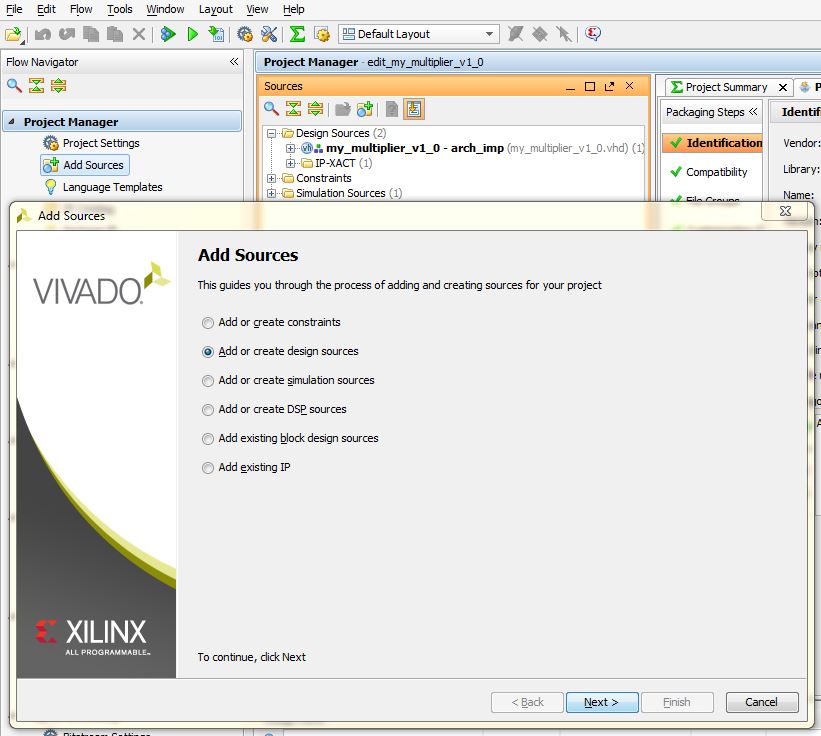


Figure 8: Adding Source file

1. On the next window, click “**Add Files**”.

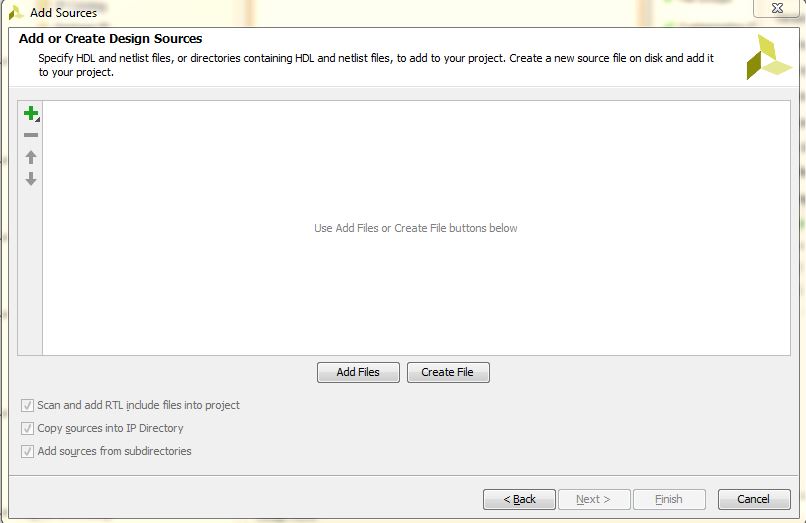


Figure 9: Browse Source file by Clicking + Sign.

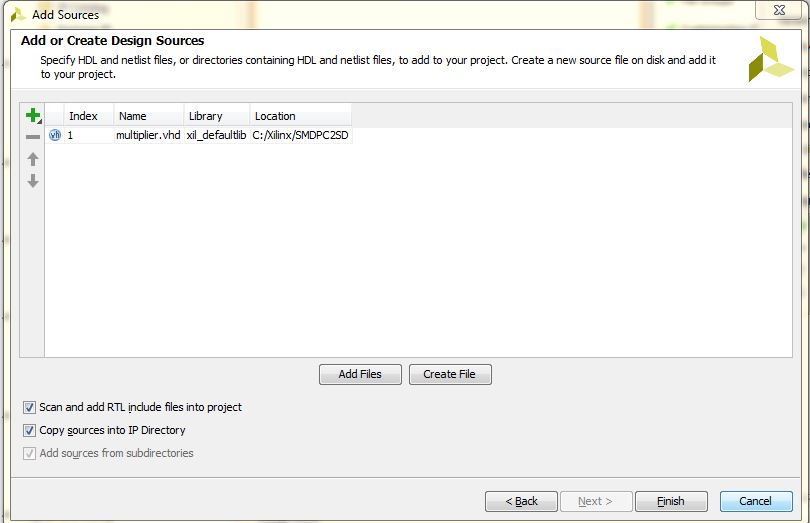
1. Browse to the “**multiplier.vhd**” file, select it and click “**OK**”.
2. Make sure we tick “**Copy sources into IP directory**” and then click “**Finish**”.

Figure 10: Selecting **multiplier.vhd**

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, our Project Manager Sources window should look like the following:

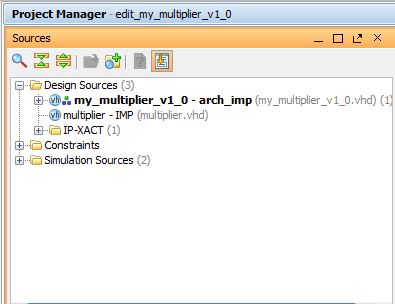


Figure 11: Project Manager

1. Open the branch “my\_multiplier\_v1\_0 – arch\_imp”.

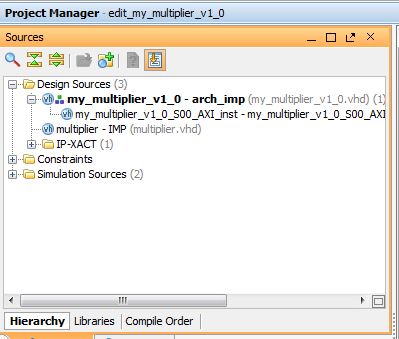


Figure 12: Selecting “**my\_multiplier\_v1\_0\_S00\_AXI\_inst**”

1. Double click on the “**my\_multiplier\_v1\_0\_S00\_AXI\_inst**” file to open it.
2. 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 the 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;**

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

**multiplier\_0 : multiplier**

**port map (**

**clk => S\_AXI\_ACLK,**

**a => slv\_reg0(31 downto 16),**

**b => slv\_reg0(15 downto 0),**

**p => multiplier\_out);**

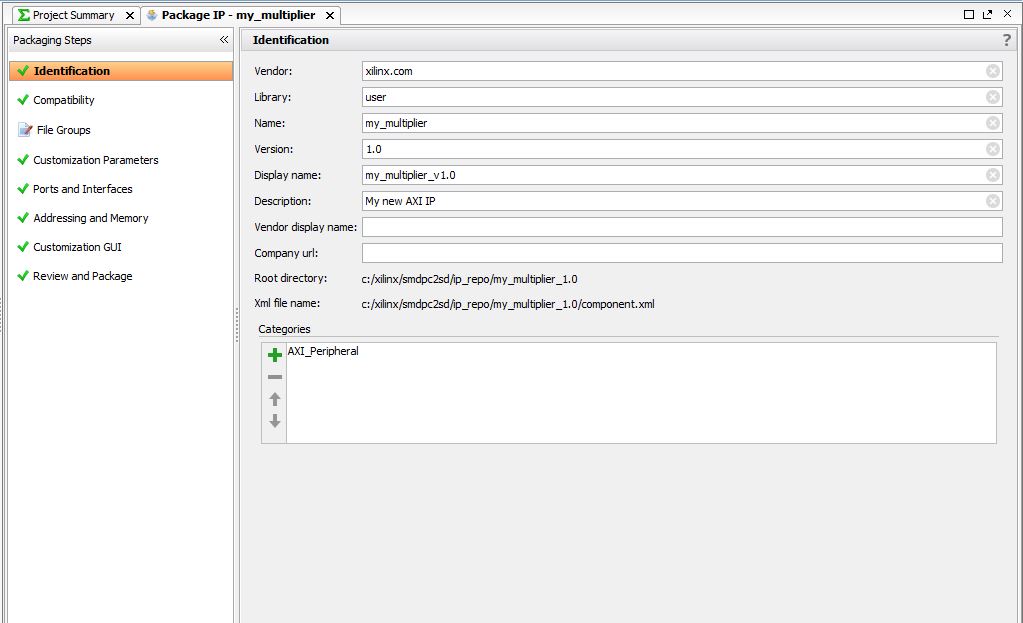
1. Find this line of code “**reg\_data\_out <= slv\_reg1;**” and replace it with “**reg\_data\_out <= multiplier\_out;**”.
2. In the process statement just a few lines above, replace “**slv\_reg1**” with “**multiplier\_out**”.
3. **Save** the file.
4. Click on “**IP File Groups**” in the Package IP tab of the Project Manager.

Figure 13: Selection of IP File Groups

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

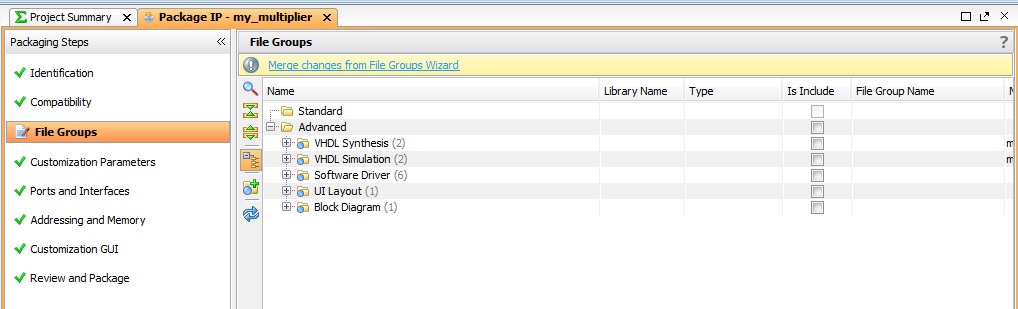


Figure 14: Merge changes from IP File Group

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

Figure 15: Green Tick indicates merging has been completed.

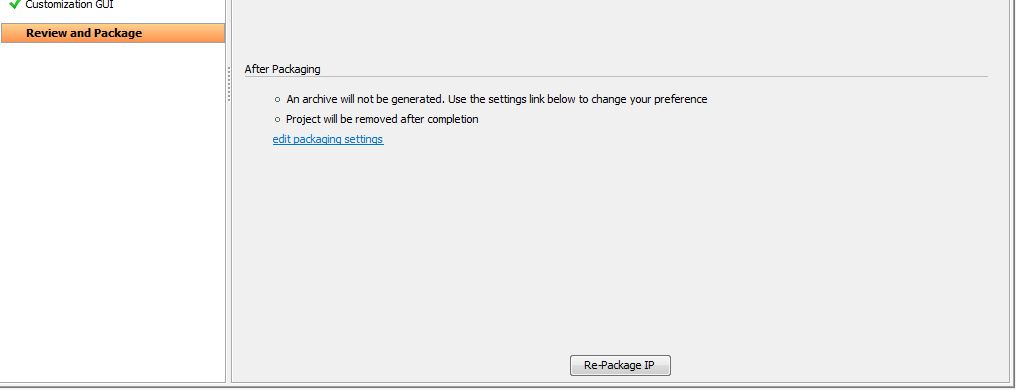
1. Now click “**Review and Package IP**”, and then click “**Re-package IP**”.

Figure 16: Re- packaging of IP.

1. If **Close Project** prompts the click **Yes**.

# Add the IP to the design

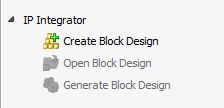
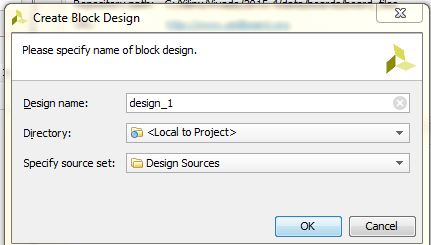
1. In the **IP Integrator**, Click on “**Create Block Design**”.

Figure 17: Creation of block design

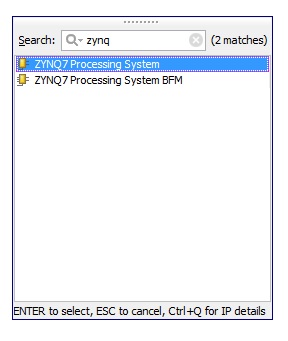
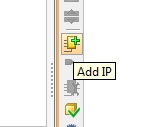
1. Click the “**Add IP**” icon. Find the “**zynq7 processing system**” IP and **double click** it.

Figure 18: Adding “**zynq7 processing system”** IP

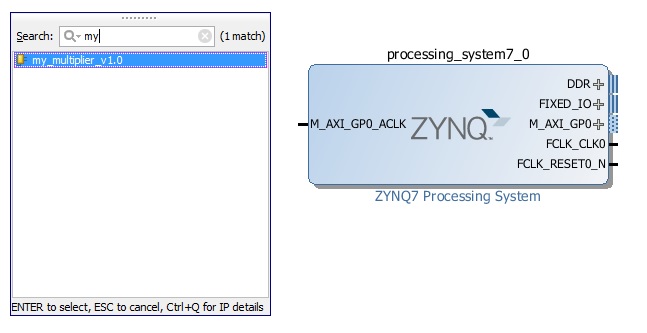
1. Find the “**my\_multiplier**” IP and **double click** it.

Figure 19: Addition of my\_multiplier IP

1. The block should appear in the block diagram and we should see the message “**Designer Assistance available**: **Run Block Automation** & **Run Connection Automation**”. Click the **Run Connection Automation**.

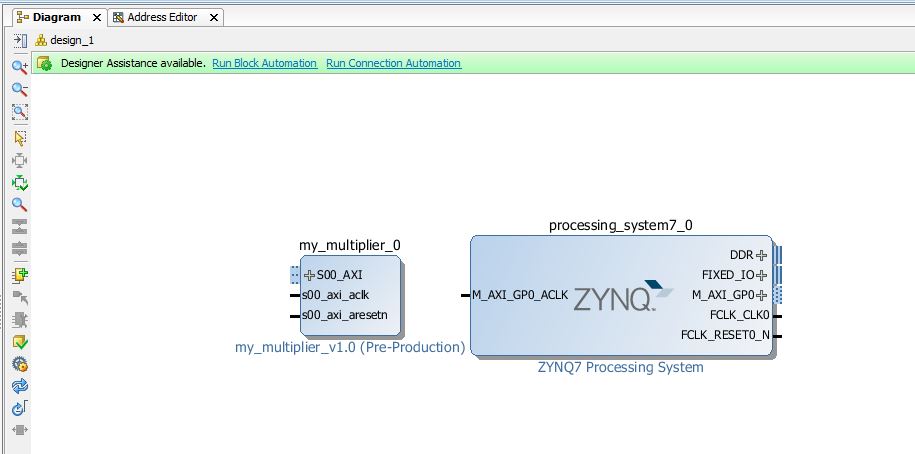


Figure 20: Addition of my\_multiplier and zynq7 processing system

1. First, Click on **Run Connection Automation** and then **OK**.

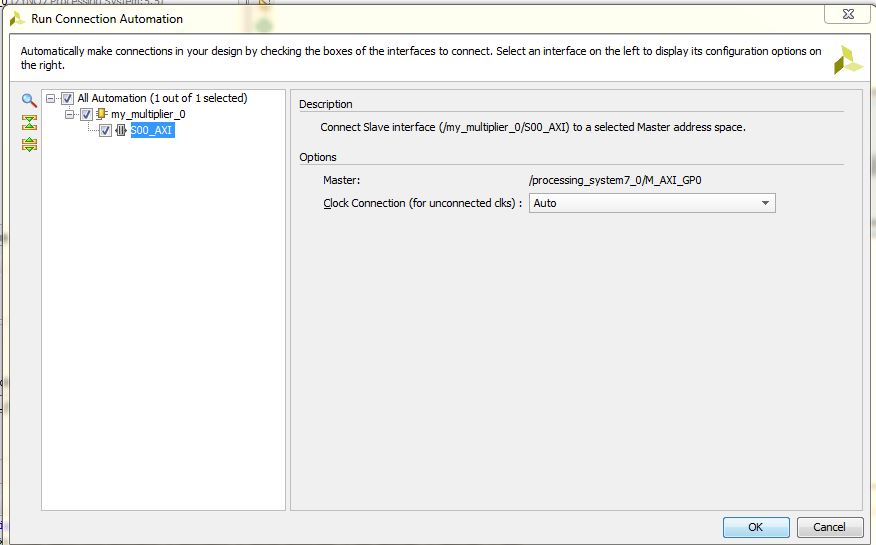


Figure 21: Connection Automation

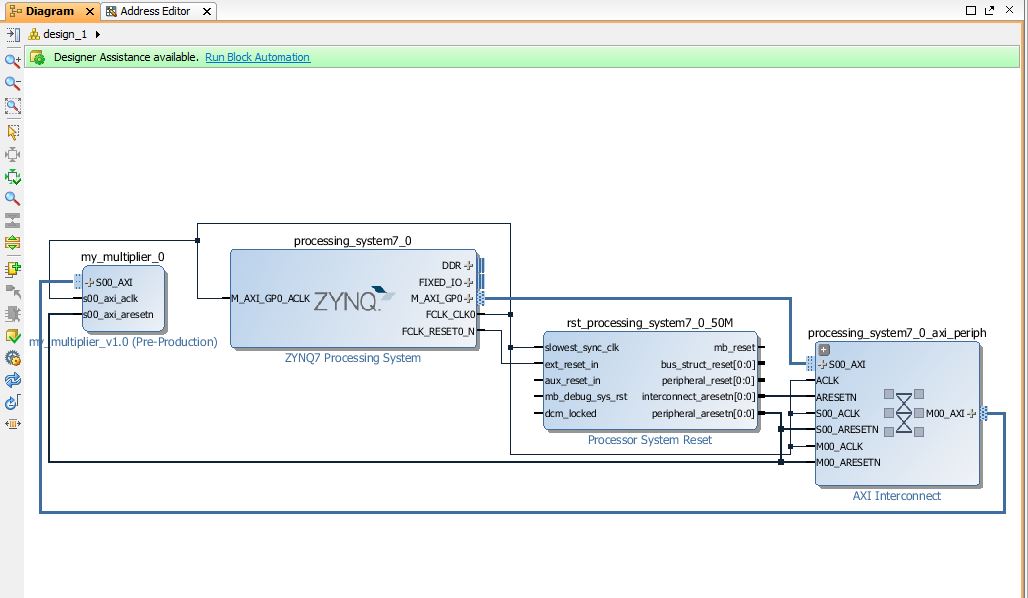
 We will be able to see this connection

Figure 22: Connection between the different IPs.

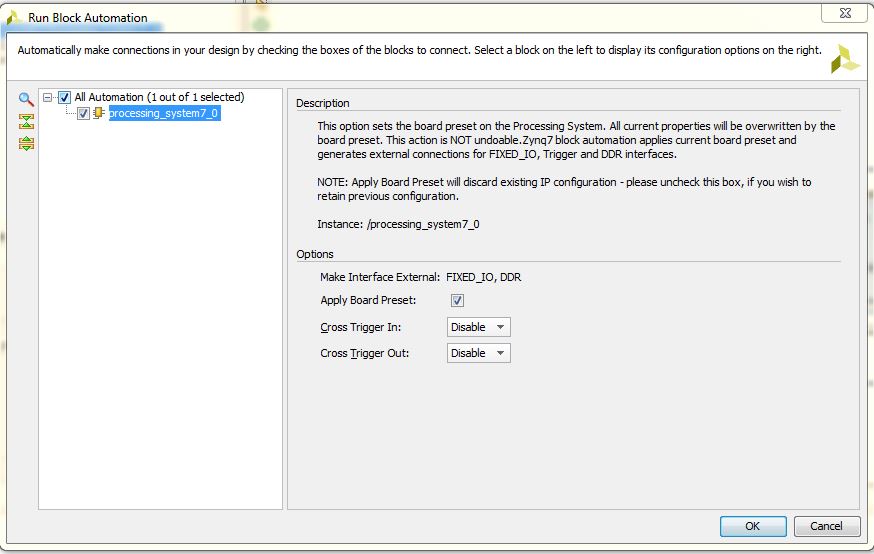
1. Then, **Run Block Automation** and click **OK**.

Figure 23: Block Automation

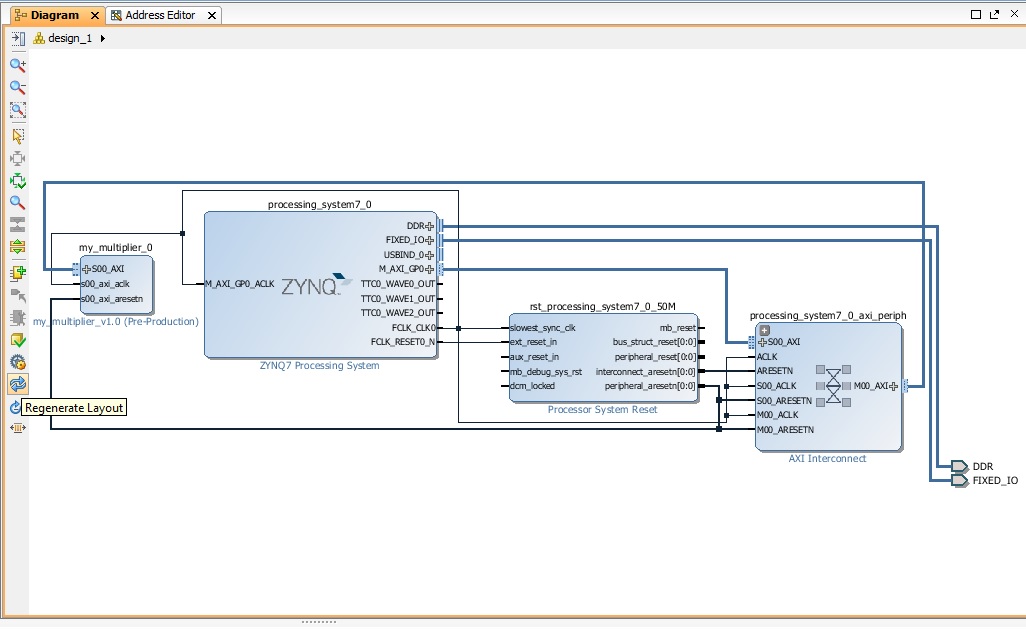


Figure 24: Overall Connection

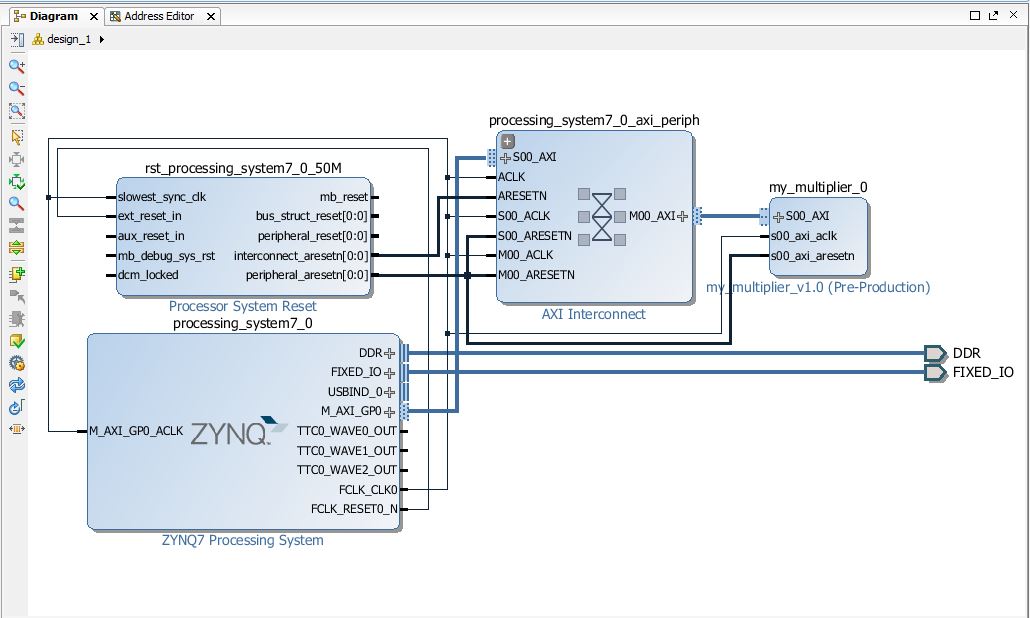
1. Regenerate Layout: Click the https://reference.digilentinc.com/_media/genesys2/regenerate.jpg **Regenerate Layout** button to rearrange your block design.

Figure 25: Regenerated layout

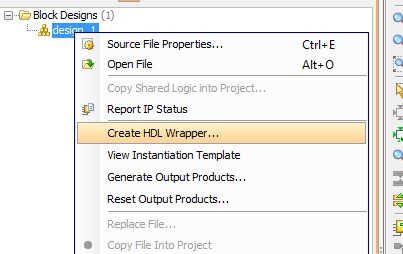
1. Select https://reference.digilentinc.com/_media/genesys2/validate.jpg **Validate Design**. This will check for design and connection errors.
2. Wrapper Creation: Right Click **design\_1.bd** , then select **Create HDL Wrapper** and click **OK.**

Figure 26 (a): Creation of HDL Wrapper

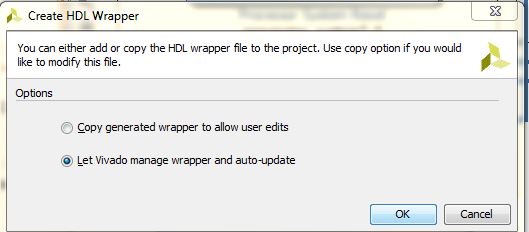


Figure 26 (b): Auto-updating of HDL Wrapper

1. Generate Bitstream: In the Program and Debug, select **Generate Bitstream.** And wait till the generation of bitstream.

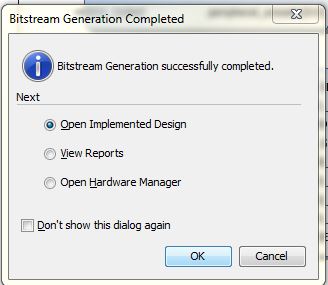
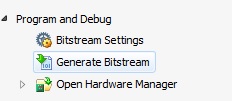
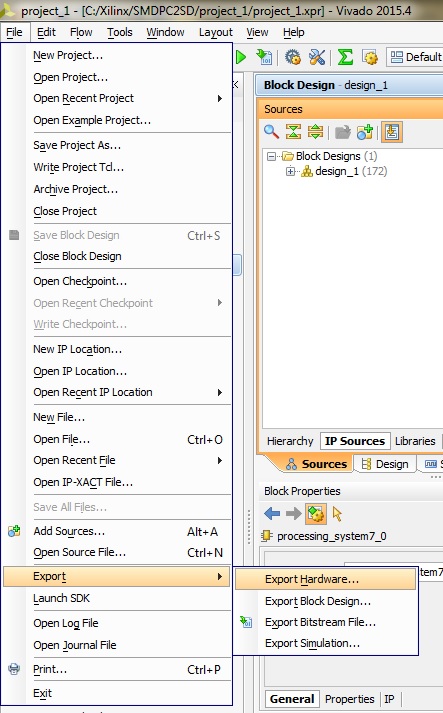


Figure 27: Bitstream Generation

# Export the hardware design to SDK

Once the bitstream has been generated, we can export our design to SDK where we can then write code for the PS. The PS is going to write data to our multiplier and read back the result.

1. In Vivado, from the File menu, select “**Export > Export hardware**”.

Figure 28: Export to Hardware

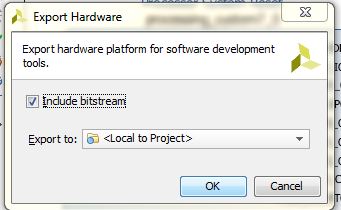
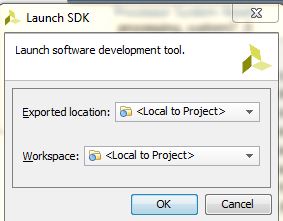
1. In the window that appears, tick “**Include bitstream**” and click “**OK**”.

Figure 29: Bitstream Inclusion

1. Again from the File menu, select “**Launch SDK**”. In the window that appears, use the following settings and click “**OK**”.



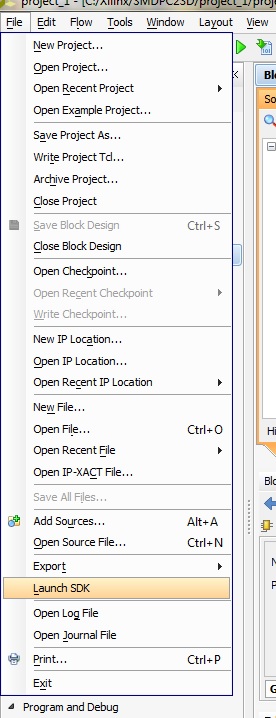
At this point, the SDK loads and a hardware platform specification will be created for our design. We should be able to see the hardware specification in the Project Explorer of SDK as shown in the image below.

Figure 30: Launch SDK

# Create a Software application

At this point, our SDK window should look somewhat like this:

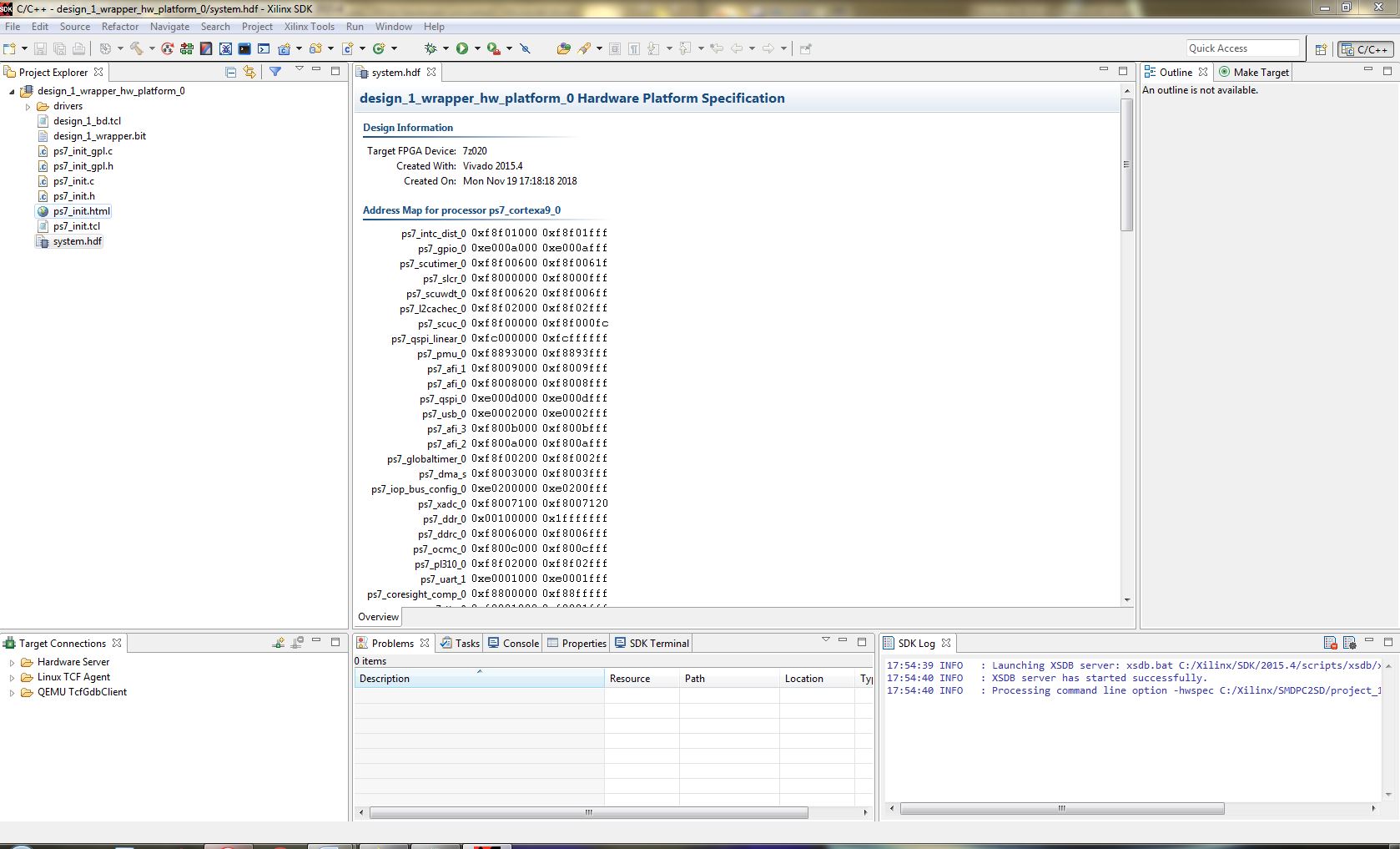


Figure 31: SDK Window

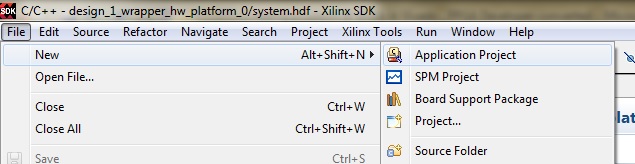
1. From the File menu, select **New** > **Application Project**.

Figure 32: Selection of Application Project

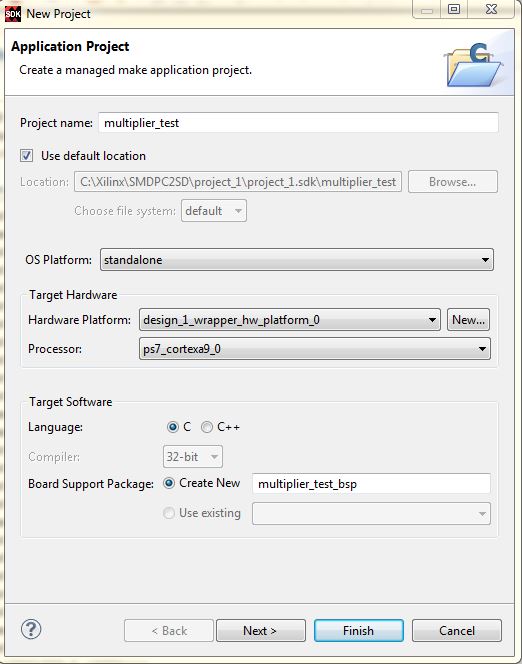
1. In the first page of the New Project wizard, choose a name for the application. We’ve chosen “**multiplier\_test**”. Click “**Next**”.

Figure 33: Creation of Application Project

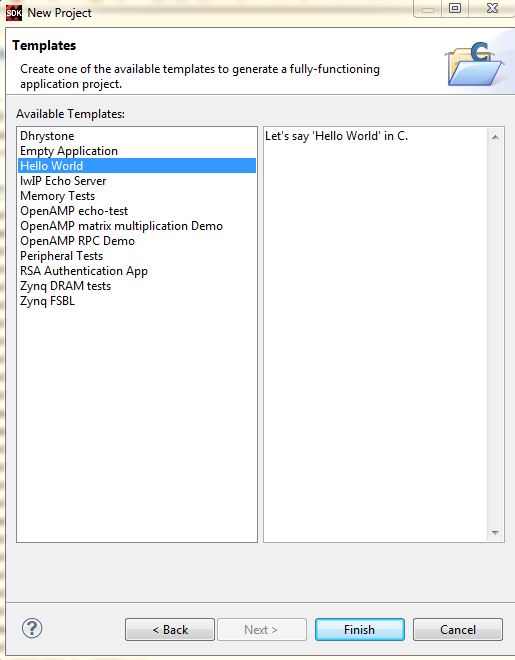
1. On the templates page, select the “**Hello World**” template and click “**Finish**”.

Figure 34: Selection of a template

1. The SDK will generate a new application which we should find in the Project Explorer as in the image below. The “**hello\_world”** folder contains the Hello World software application, which we will **modify** to test our multiplier.

Figure 35: Source folder of multiplier\_test

# Modify the Software Application

Now all we need to do is modify the software application to test our multiplier peripheral.

1. From the Project Explorer, open the “**hello\_world/src**” folder. Open the “**helloworld.c**” source file.
2. **Replace all the code** in this file with the following:

#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;

}

1. **Save** (Ctrl + S) and close the file.

# Test the design on the hardware

To test the design, we are using ZedBoard from Avnet.

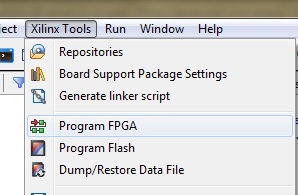
1. In the SDK, from the menu, select **Xilinx Tools** > **Program FPGA.**

Figure 36: Selecting Program FPGA

1. In the Program FPGA window, we select the hardware platform to program. We have only one hardware platform, so click “**Program**”.

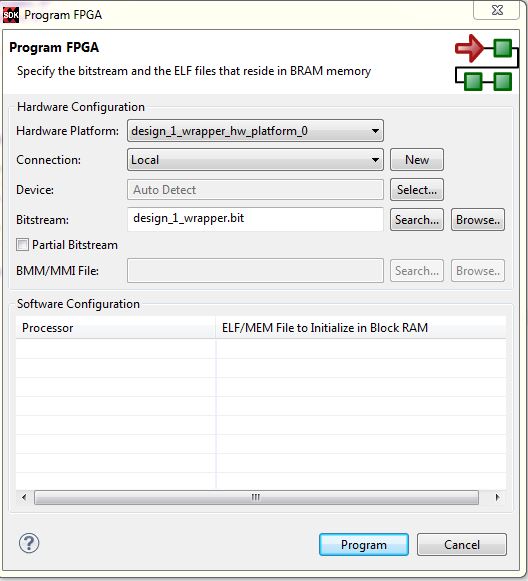
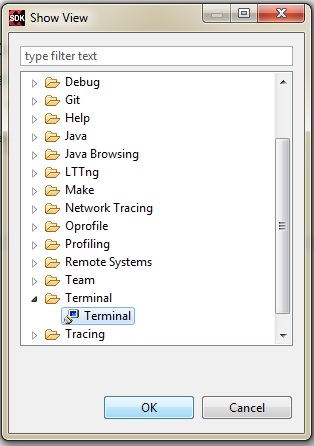
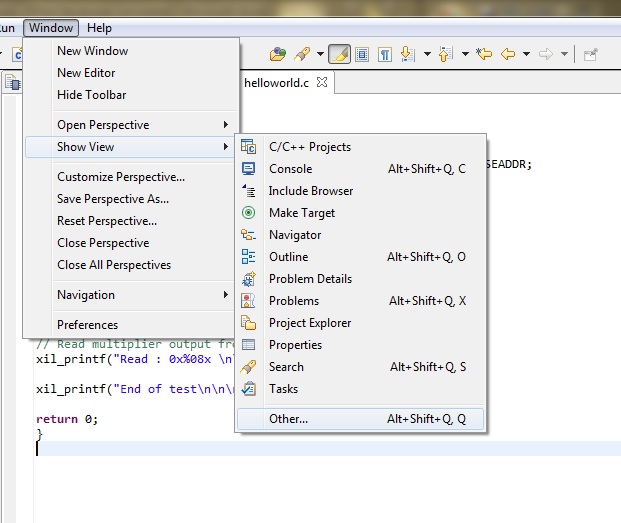
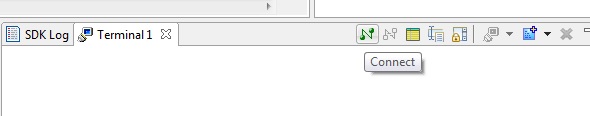


Figure 37: Programming of FPGA

1. Adding terminal: Add terminal to view the generated output. Select **Window > Show View > Other > Terminal > Terminal.**

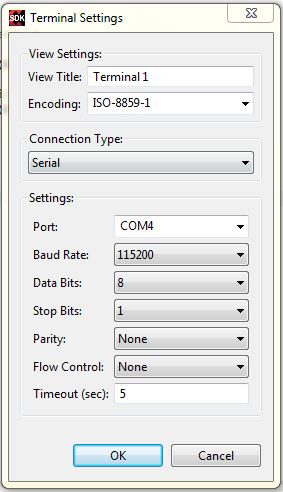
Figure 38: Addition of Terminal

1. Connecting Terminal

Figure 39: Connecting Terminal

Made the following changes in order to see the output in the terminal:

1. Connection Type: Serial
2. Port: COM4 (Nt.: It may be different with respect to PC, try selecting USB Driver).
3. Baud Rate: 11520.



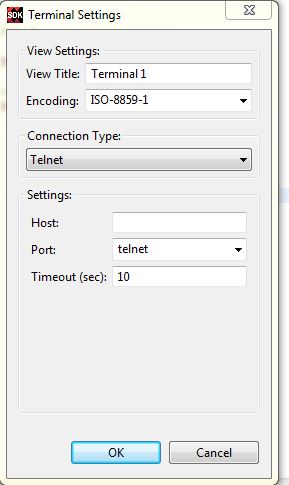


Figure 40: Configuration to Terminal

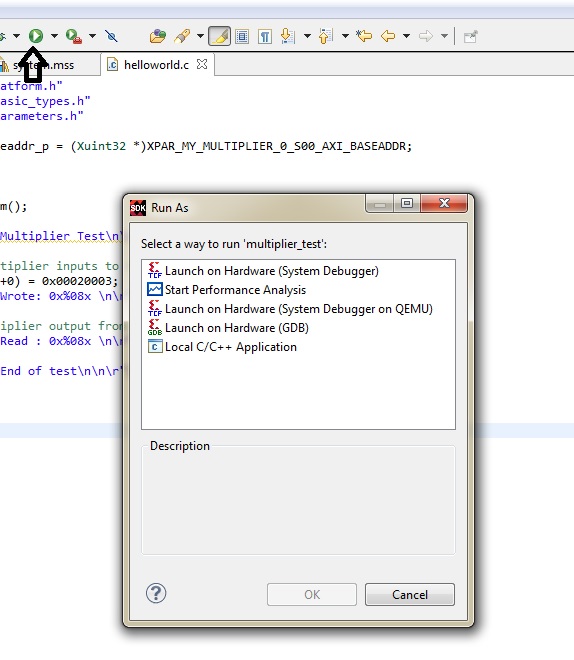
1. The bitstream will be loaded onto the Zynq and we are ready to load the software application. Select the “**multiplier\_test**” folder in the Project Explorer, and then from the menu, select **Run->Run**.
2. In the Run As window, select “**Launch on Hardware (System Debugger)**” and click “**OK**”.

Figure 41: Launching Hardware

1. Output: Then again click Run to get the output in the terminal.

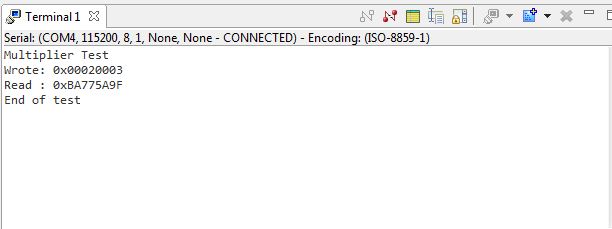


Figure 42: Output in the Terminal

1. We can also see the output in the Tera Term Software.
2. Open Tera Term.

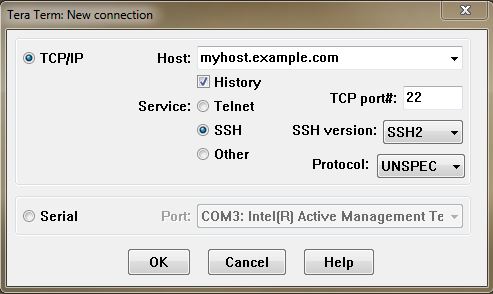


Figure 43: Tera Term

1. Select **Serial > COM4**

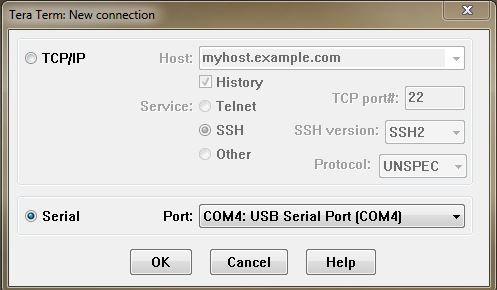


Figure 44: Selection of COM4: USB Serial Port

1. Click **Setup > Serial Port**. Change **Speed** to **115200** (Same as Baud Rate).

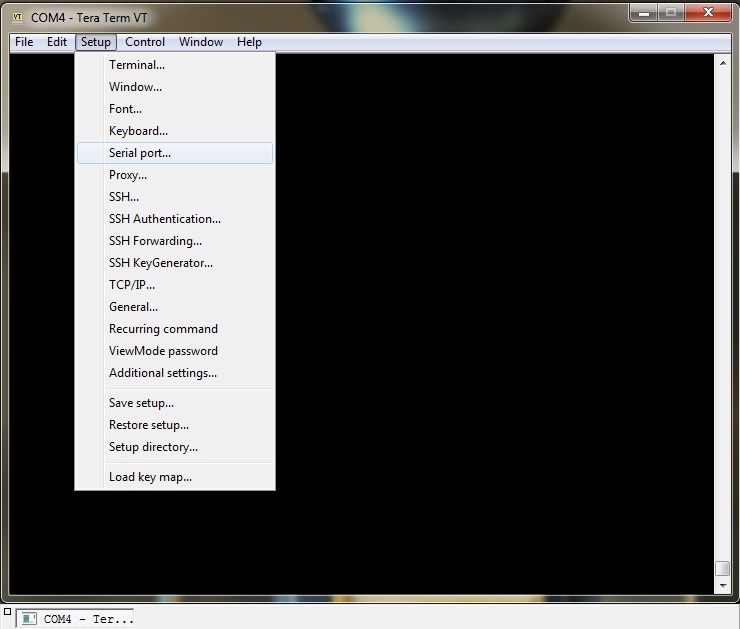
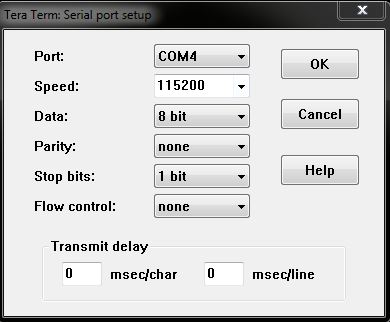


Figure 45: Setting up of Tera term

1. Output can be seen as the given figure

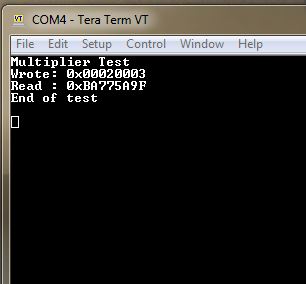
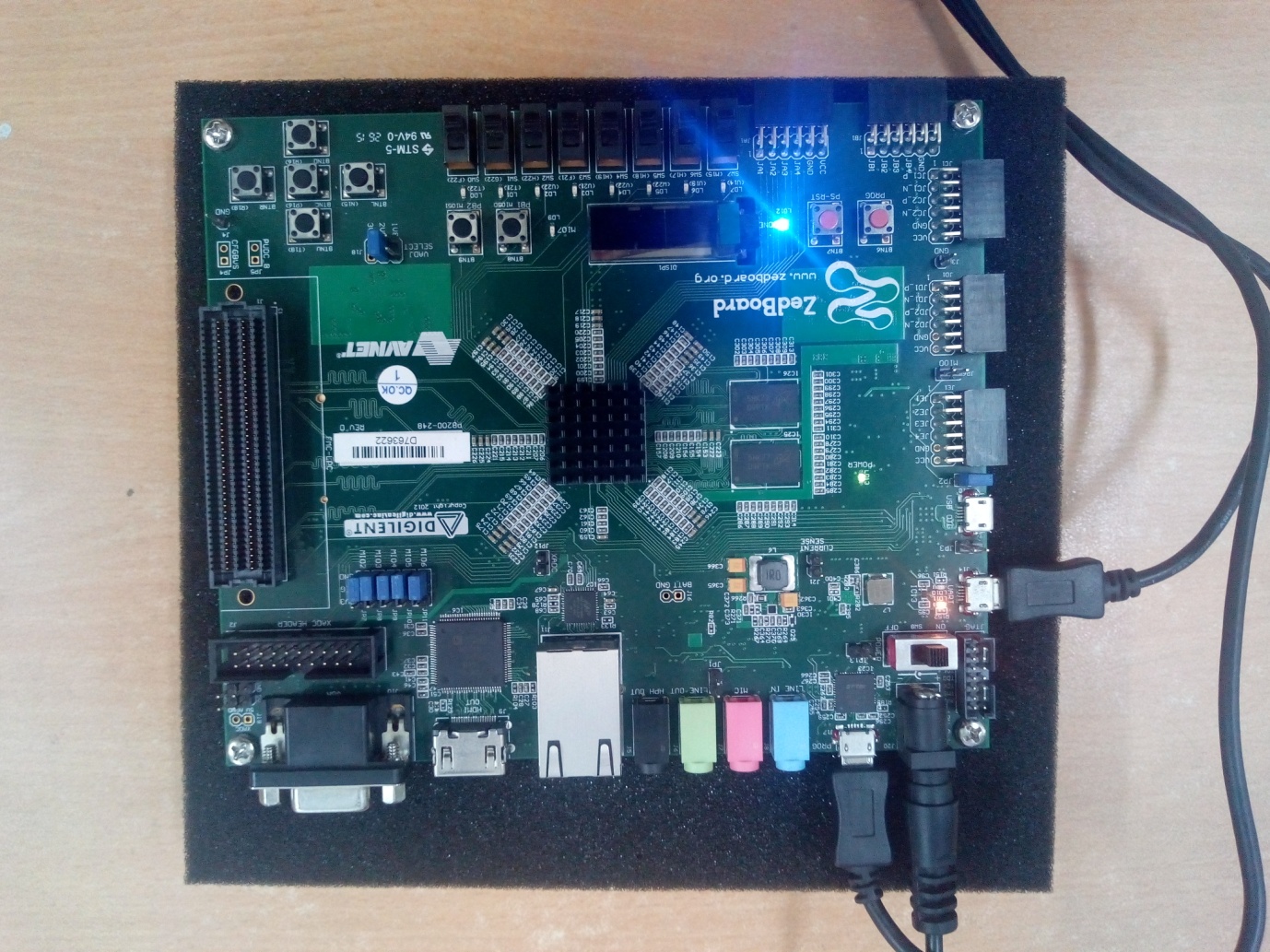


Figure 46: Output in the Tera Term

Figure 47: FPGA while Program is running