fpgadeveloper.com

Creating a custom IP block in Vivado

Jeff Johnson

9-12 minutes

Update 2017-11-01: Here's a newer tutorial on creating a custom IP with AXI-Streaming interfaces

Tutorial Overview

In this tutorial we'll create a custom AXI IP block in Vivado and modify its functionality by integrating custom VHDL code. We'll be using the Zynq SoC and the MicroZed 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 outputs one 32 bit unsigned output. A single 32 bit write 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 doesn't serve much purpose, but it is a good example of integrating your own code into an AXI IP block.

Requirements

Before following this tutorial, you will need to do the following:

- Vivado 2014.2
- MicroZed
- Platform Cable USB II (or equivalent JTAG programmer)

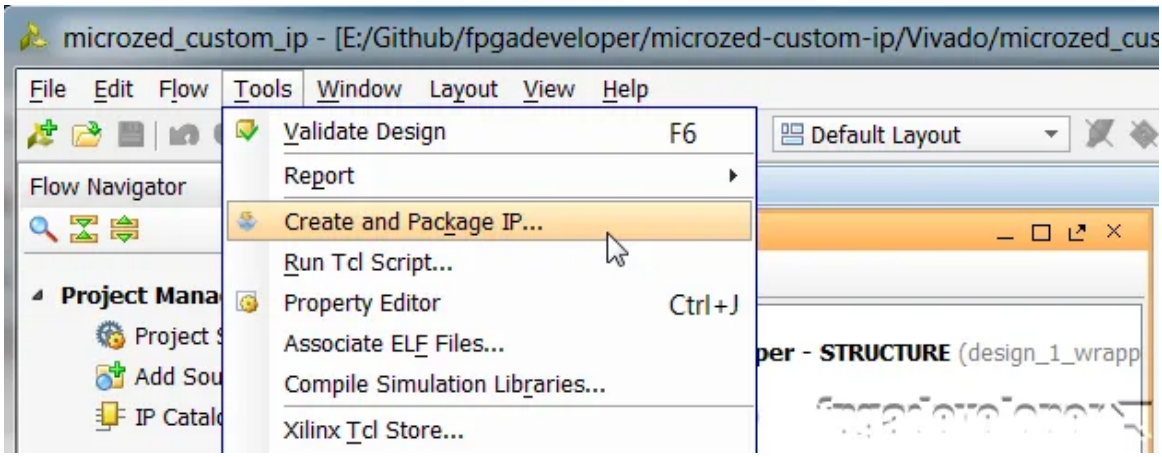
Start from a base project

You can do this tutorial with any existing Vivado project, but I'll start with the base system project for the MicroZed that you can access here:

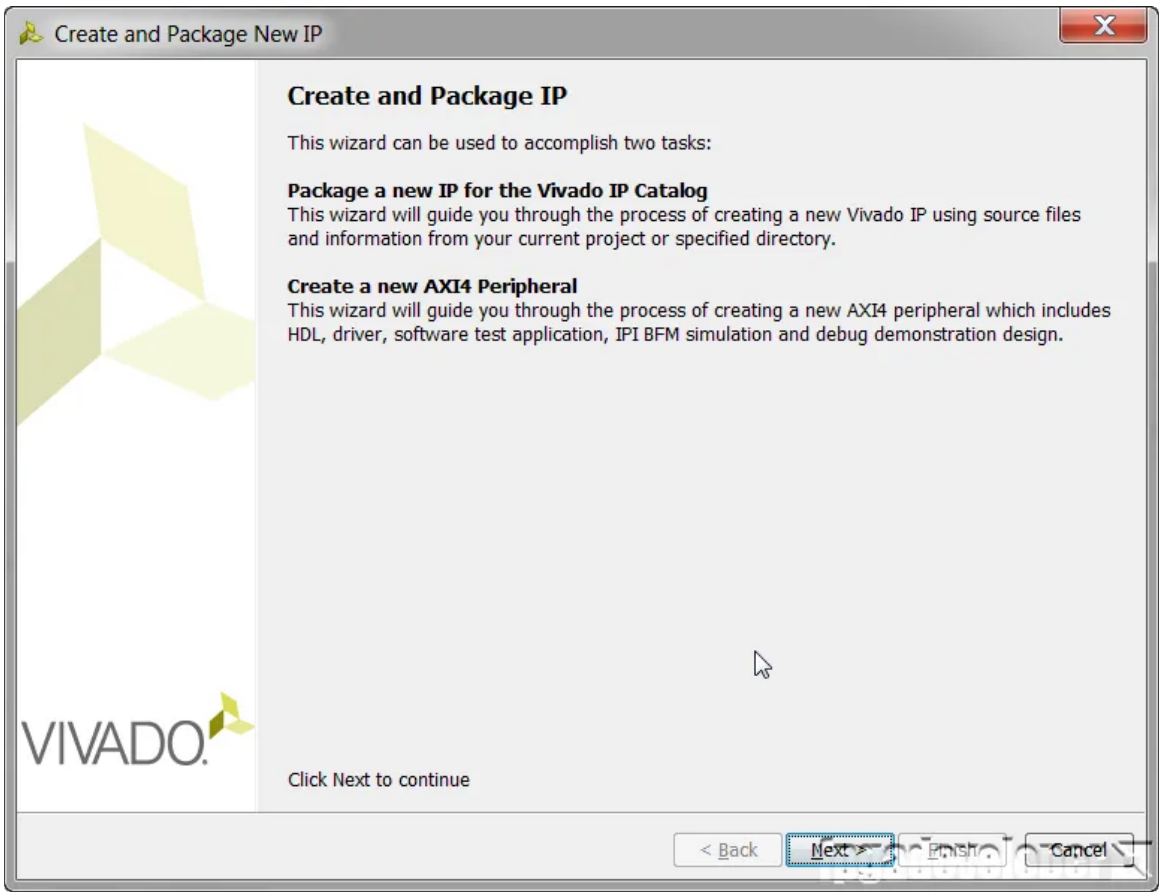
Base system project for the MicroZed

Create the Custom IP

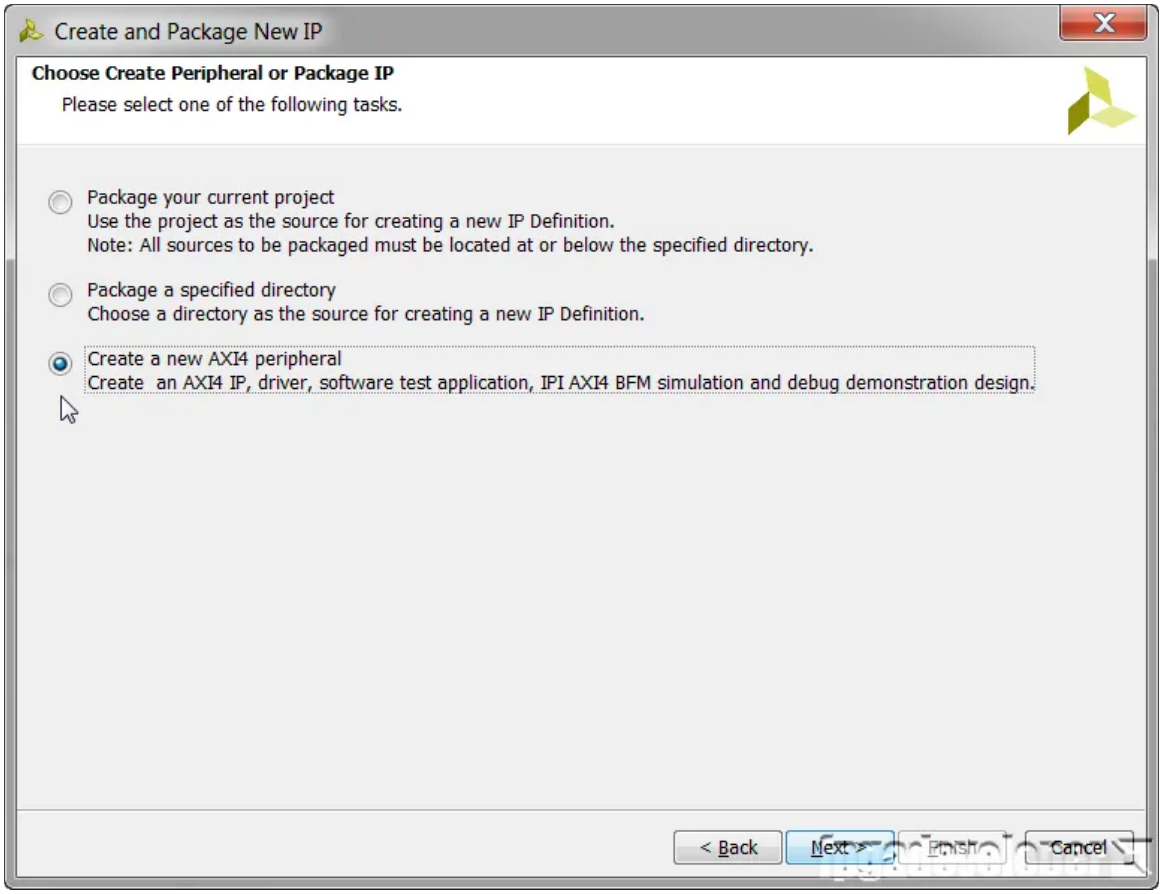
1. With the base Vivado project opened, from the menu select Tools->Create and package IP.



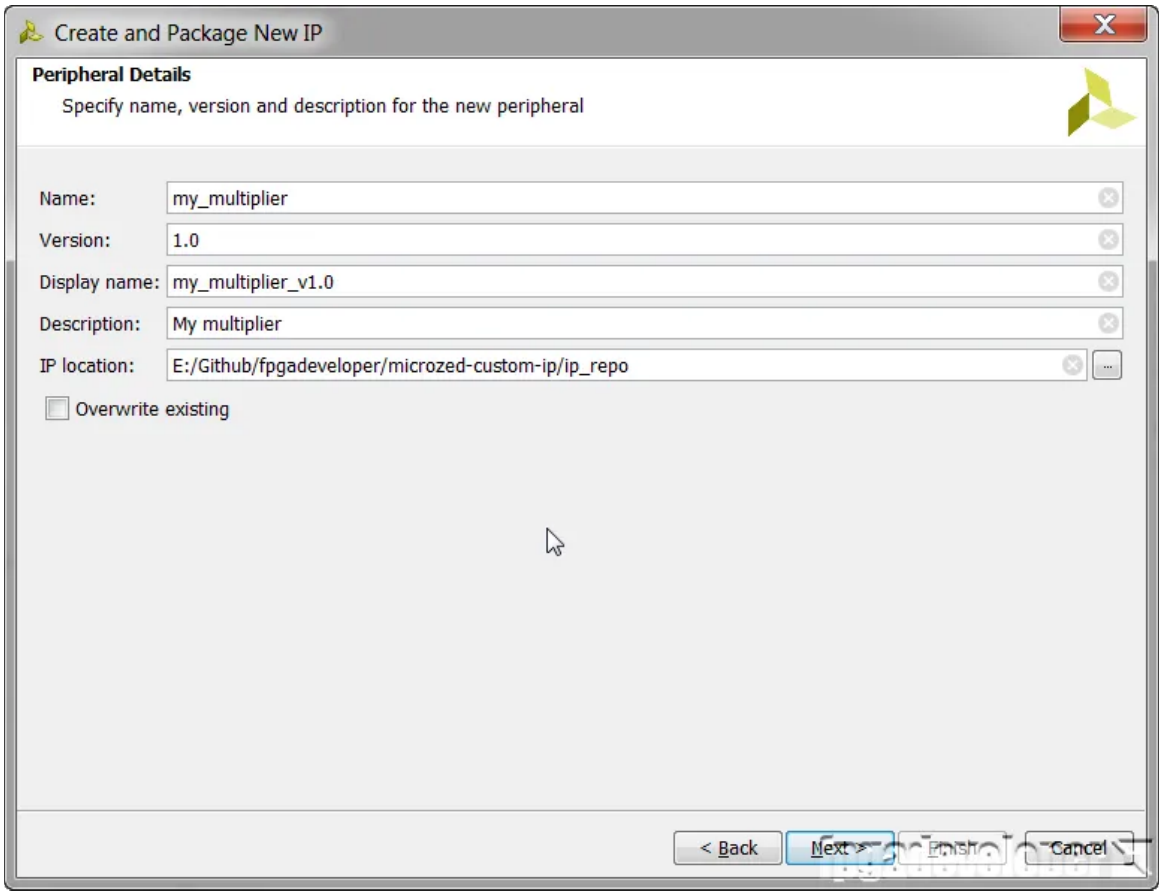
2. The Create and Package IP wizard opens. If you are used to the ISE/EDK tools you can think of this as being similar to the Create/Import Peripheral wizard. Click “Next”.



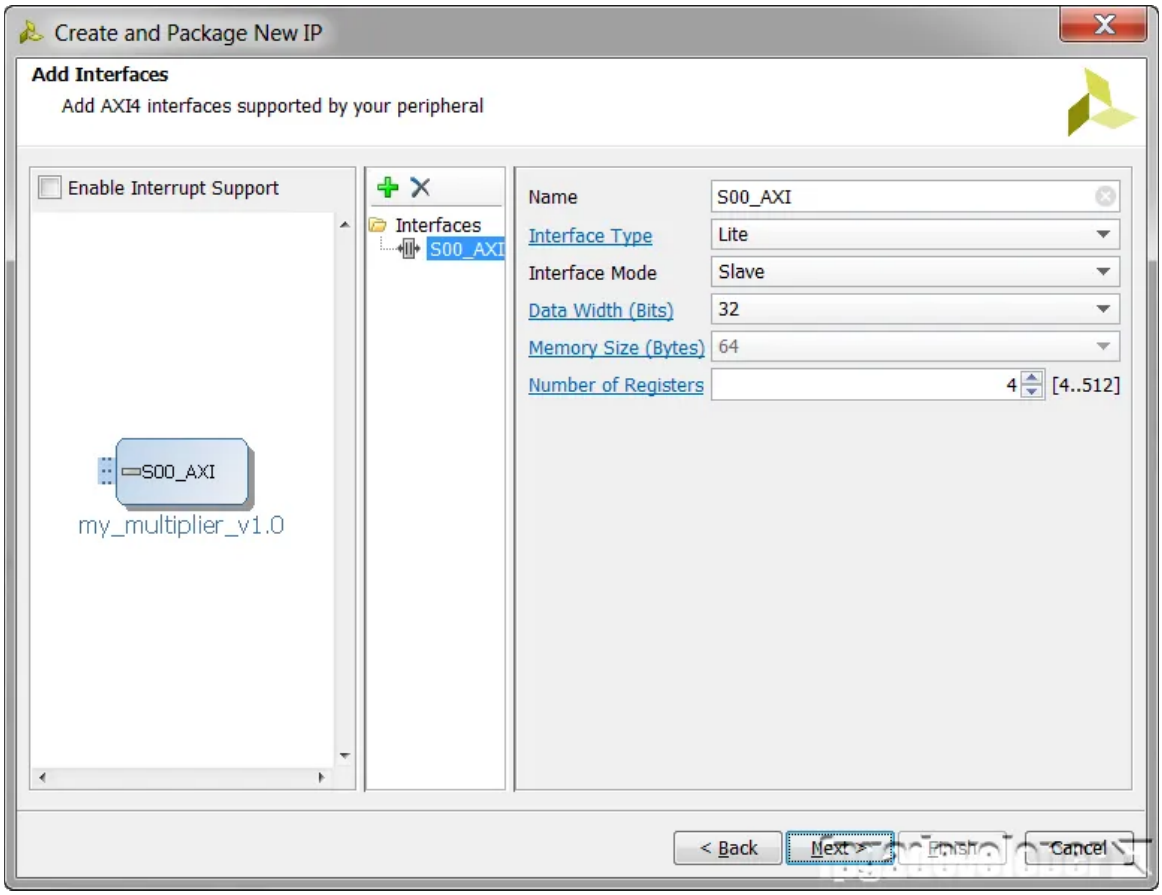
3. On the next page, select “Create a new AXI4 peripheral”. Click “Next”.



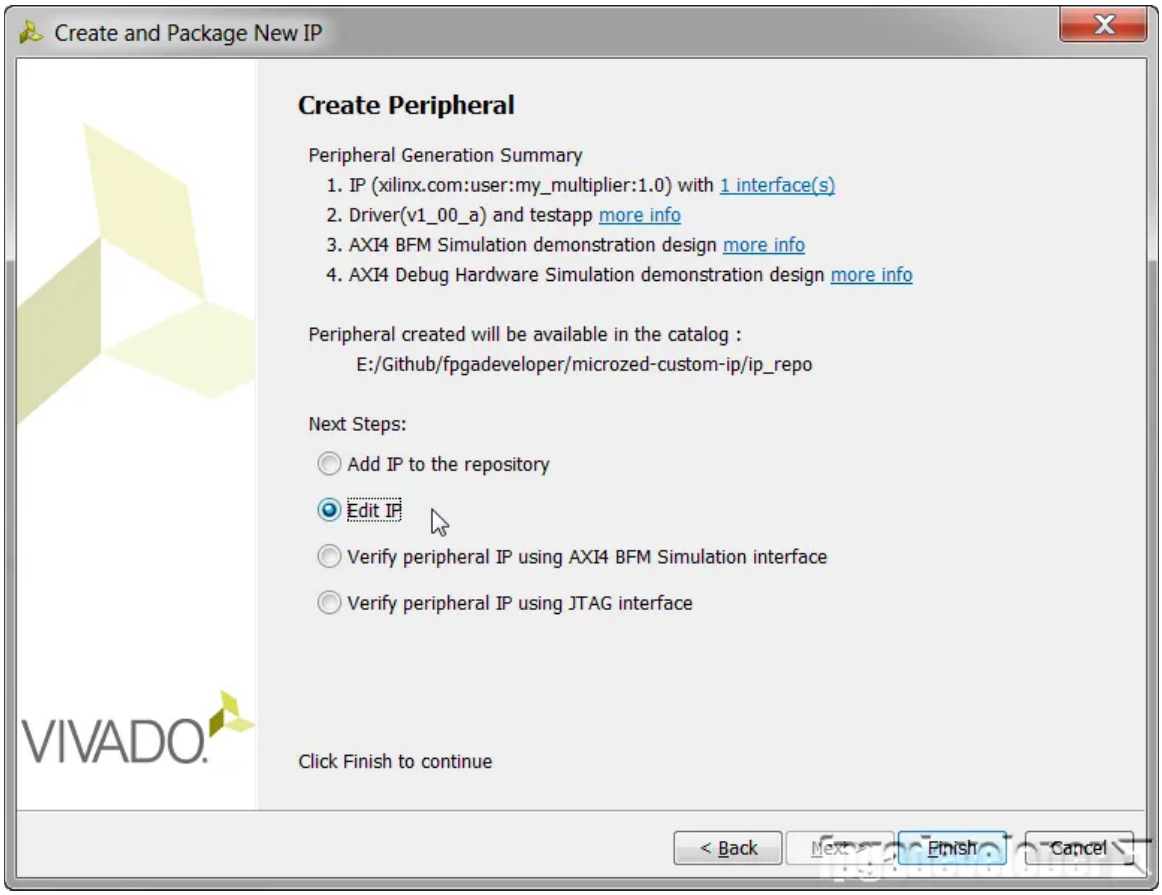
4. Now you can give the peripheral an appropriate name, description and location. Click “Next”.

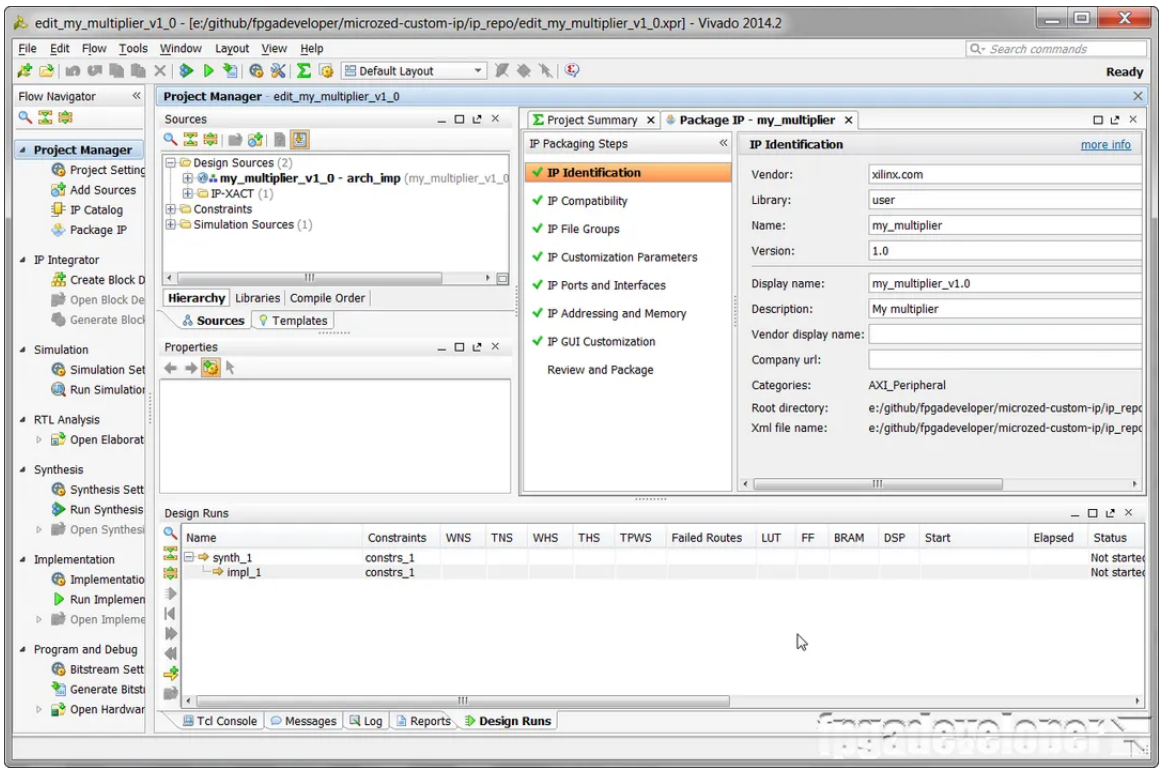


5. On the next page we can configure the AXI bus interface. For the multiplier we'll use AXI lite, and it'll be a slave to the PS, so we'll stick with the default values.



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





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 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 multiplier output.

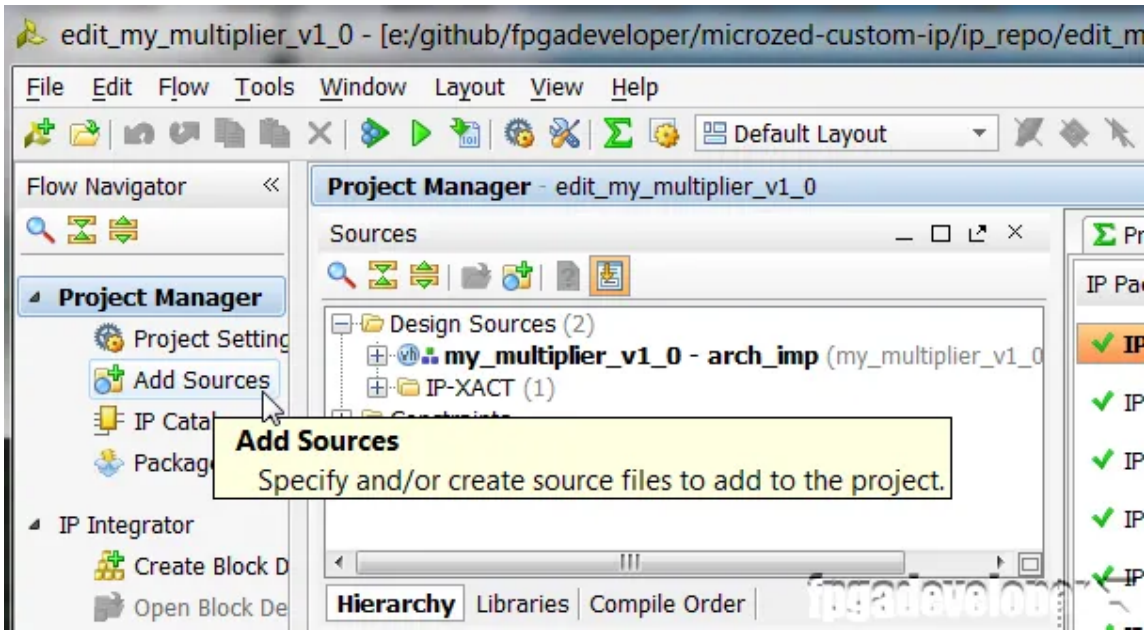
Add the multiplier code to the peripheral

You can find the multiplier code on Github at the link below. Download the “multiplier.vhd” file and save it somewhere, the location is not important for now.

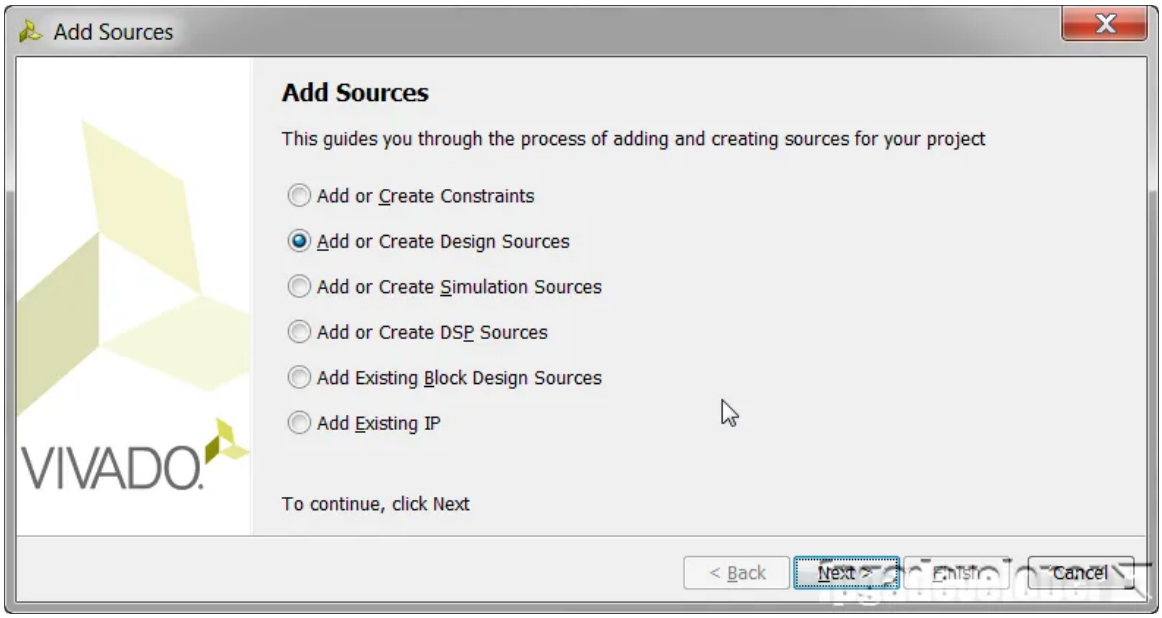
https://github.com/fpgadeveloper/microzed-custom-ip/blob/master/Vivado/ip_repo/my_multiplier_1.0/src/multiplier.vhd

Note that these steps must be done in the Vivado window that contains the peripheral we just created (not the base project).

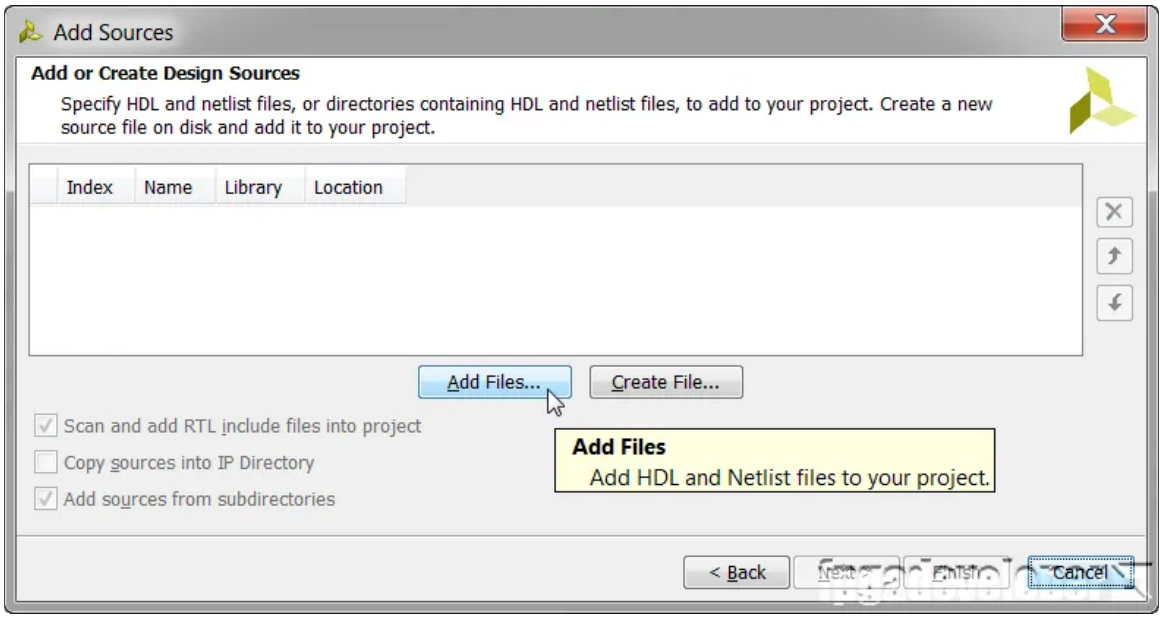
1. From the Flow Navigator, click “Add Sources”.



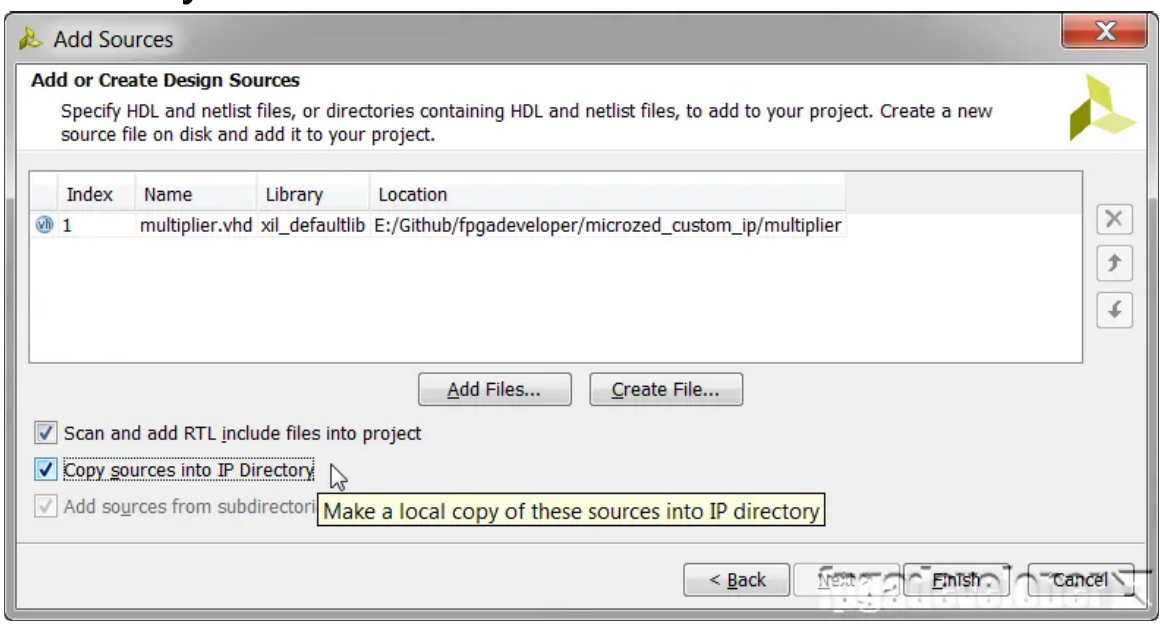
2. In the window that appears, select “Add or Create Design Sources” and click “Next”.



3. On the next window, click “Add Files”.



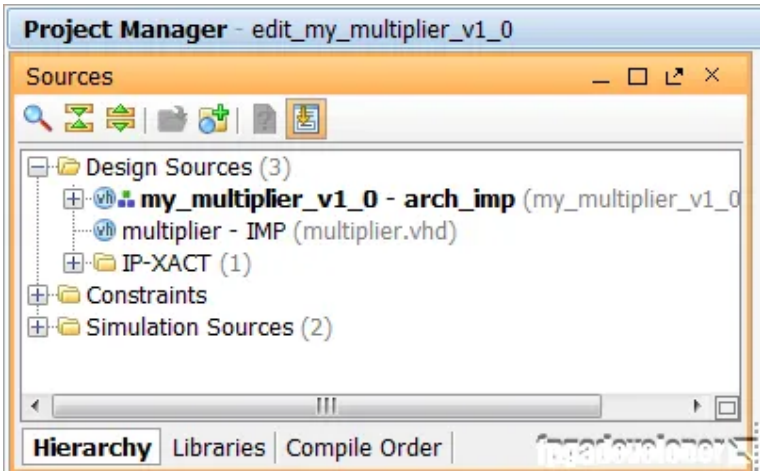
4. Browse to the “multiplier.vhd” file, select it and click “OK”.
5. Make sure you tick “Copy sources into IP directory” and then click “Finish”.



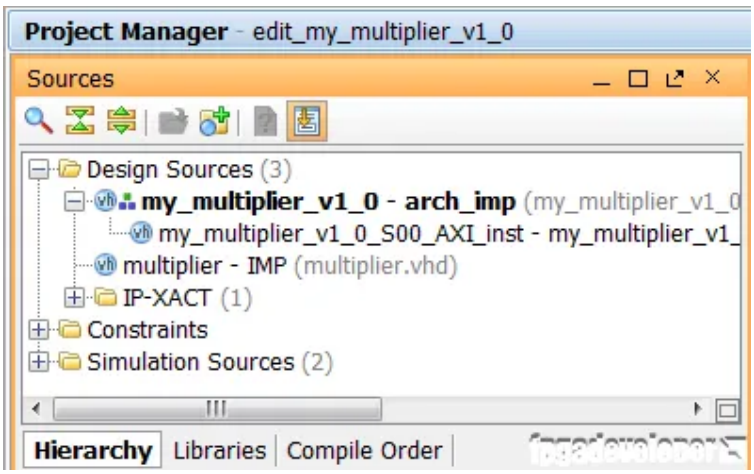
The multiplier code is now added to the peripheral, however we still have to instantiate it and connect it to the registers.

Modify the Peripheral

At this point, your Project Manager Sources window should look like the following:



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 the output signal:

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

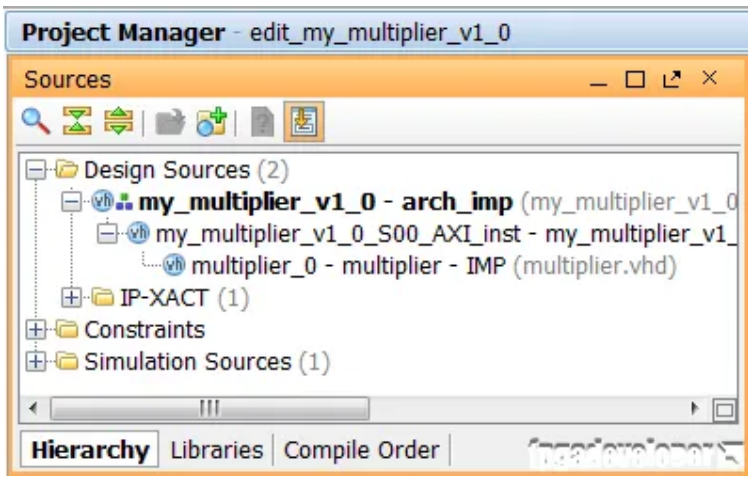
```
component multiplier
port (
  clk: in std_logic;
  a: in std_logic_VECTOR(15 downto
0);
  b: in std_logic_VECTOR(15 downto
0);
  p: out std_logic_VECTOR(31 downto
0));
end component;
```

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

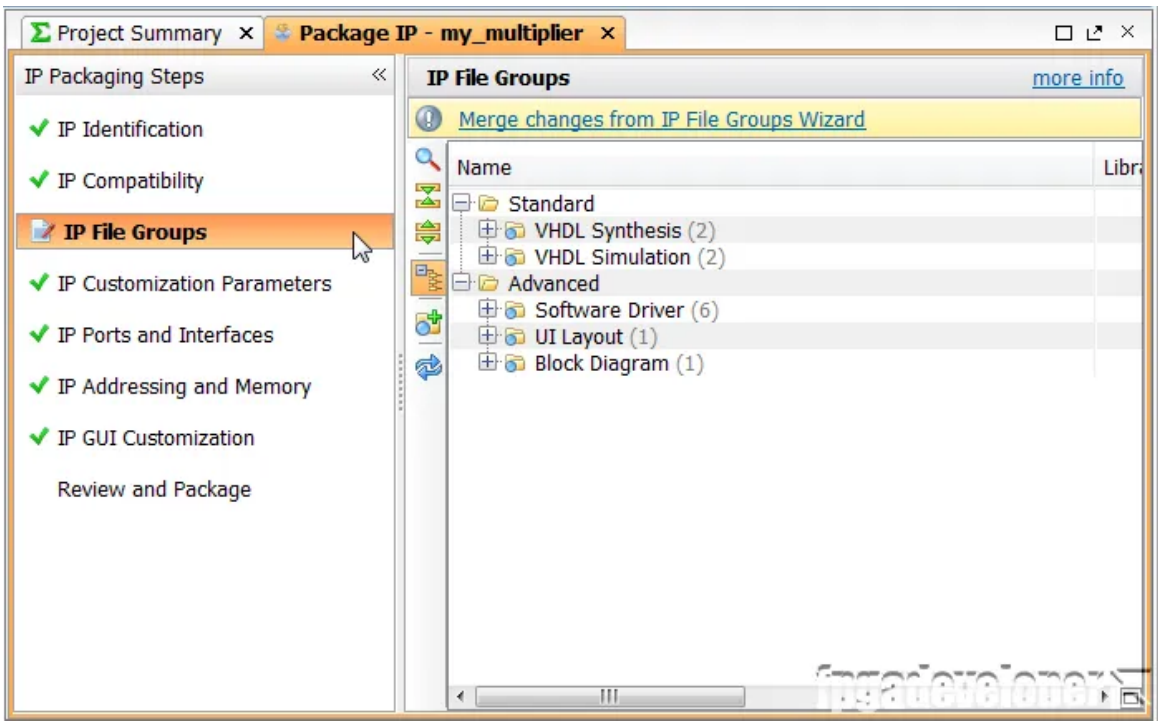
```
multiplier_0 : multiplier
port map (
  clk => S_AXI_ACLK,
  a => slv_reg0(31 downto 16),
  b => slv_reg0(15 downto 0),
  p => multiplier_out);
```

5. Find this line of code “reg_data_out <= slv_reg1;” and replace it with “reg_data_out <= multiplier_out;”.

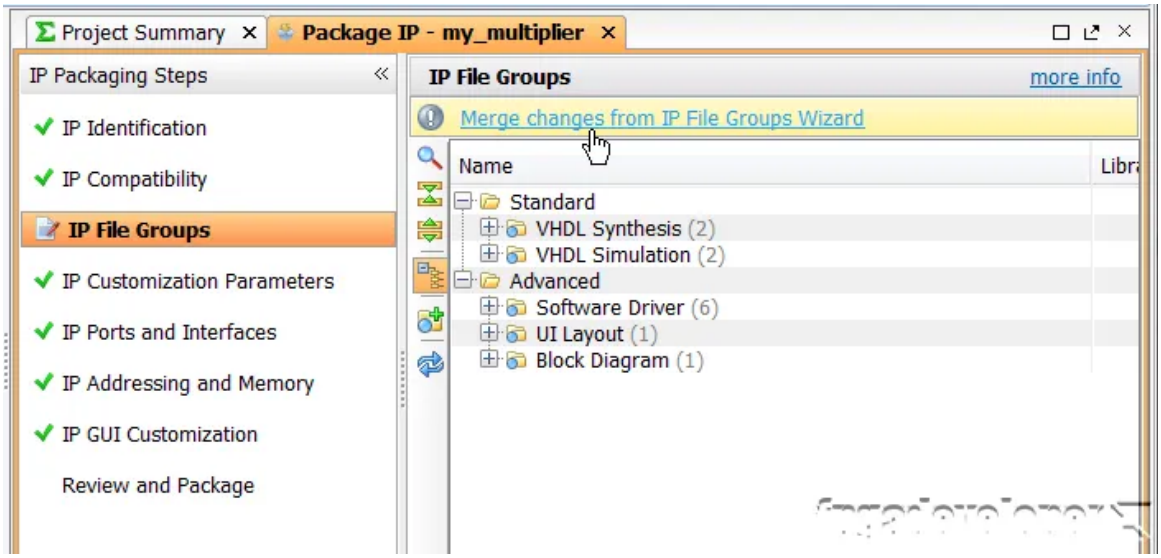
6. In the process statement just a few lines above, replace “slv_reg1” with “multiplier_out”.
7. Save the file.
8. You should notice that the “multiplier.vhd” file has been integrated into the hierarchy because we have instantiated it from within the peripheral.



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

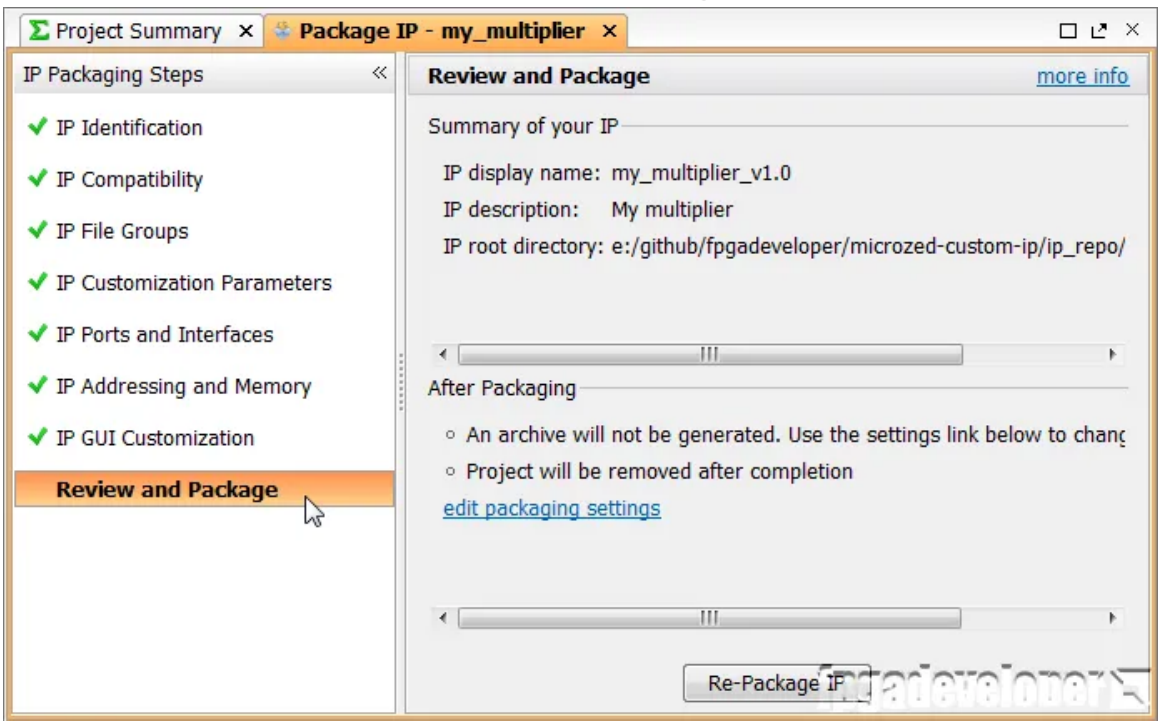


10. Click the “Merge changes from IP File Group Wizard” link.

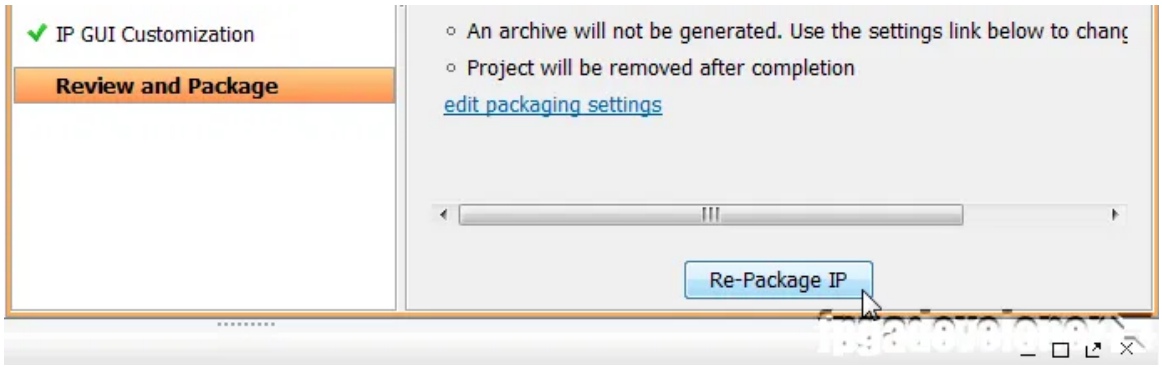


11. The “IP File Groups” should now have a tick.

12. Now click “Review and Package IP”.



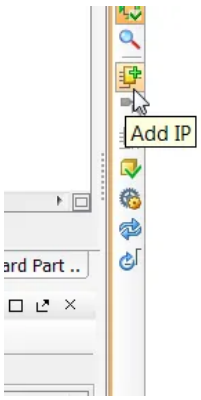
13. Now click “Re-package IP”.

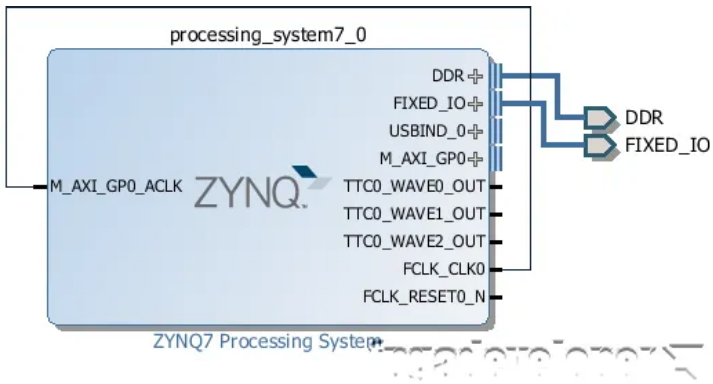


The peripheral will be packaged and the Vivado window for the peripheral should be automatically closed. We should now be able to find our IP in the IP catalog. Now the rest of this tutorial will be done from the original Vivado window.

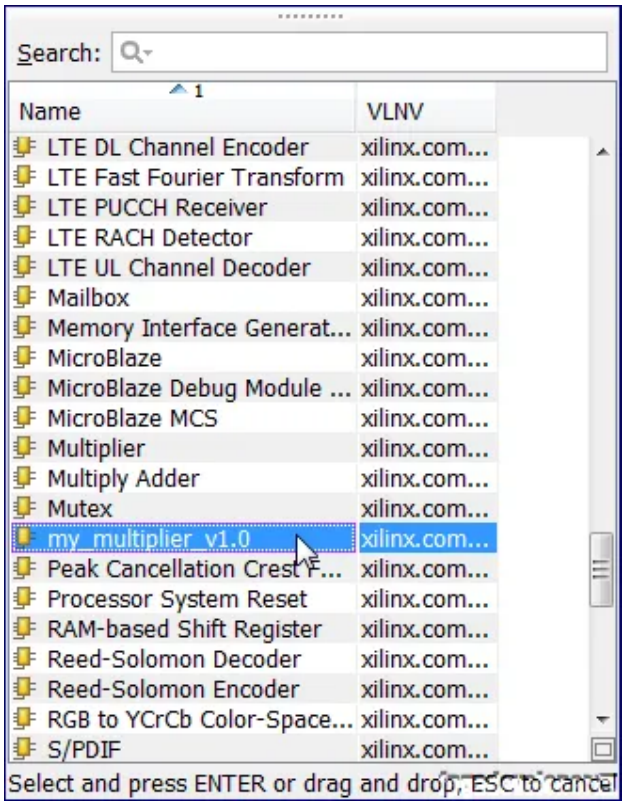
Add the IP to the design

1. Click the “Add IP” icon.

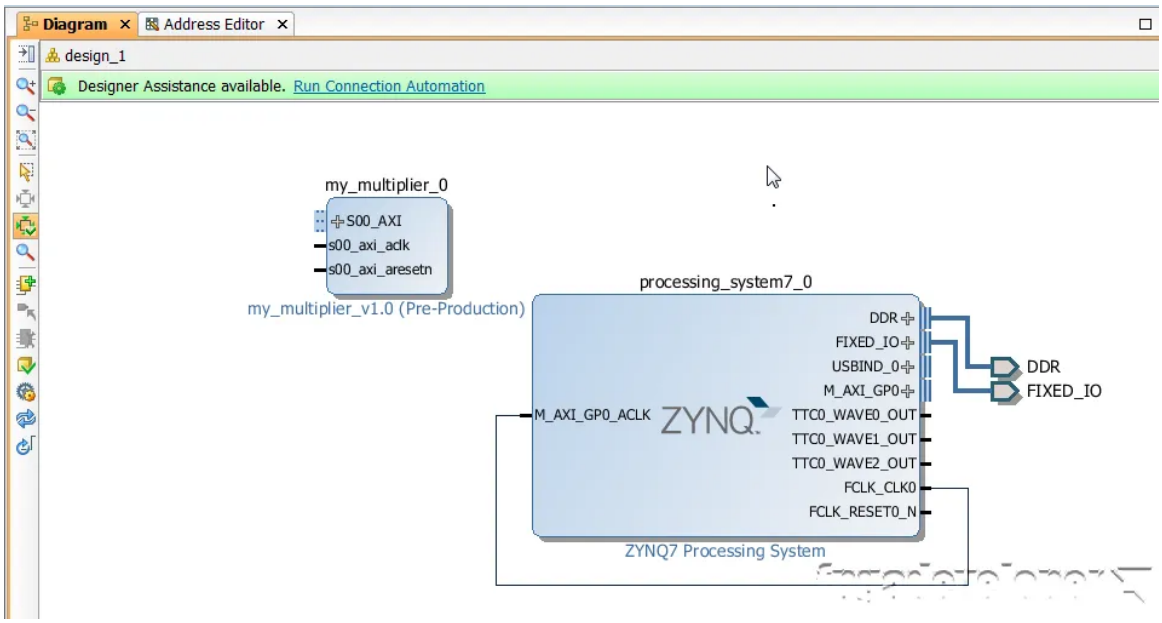




2. Find the “my_multiplier” IP and double click it.

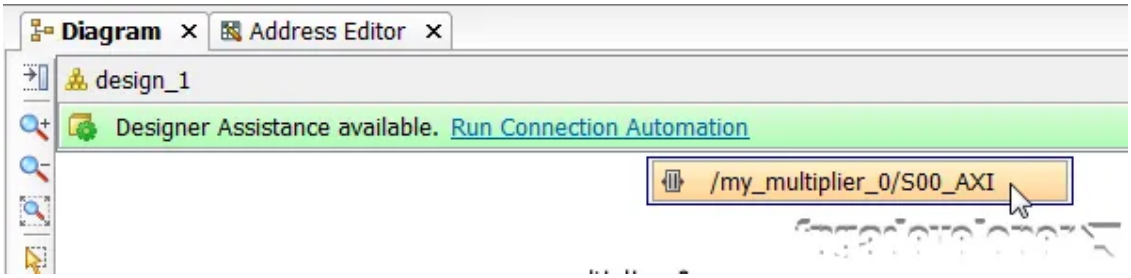


3. The block should appear in the block diagram and you should see the message “Designer Assistance available. Run Connection Automation”. Click the connection automation link.

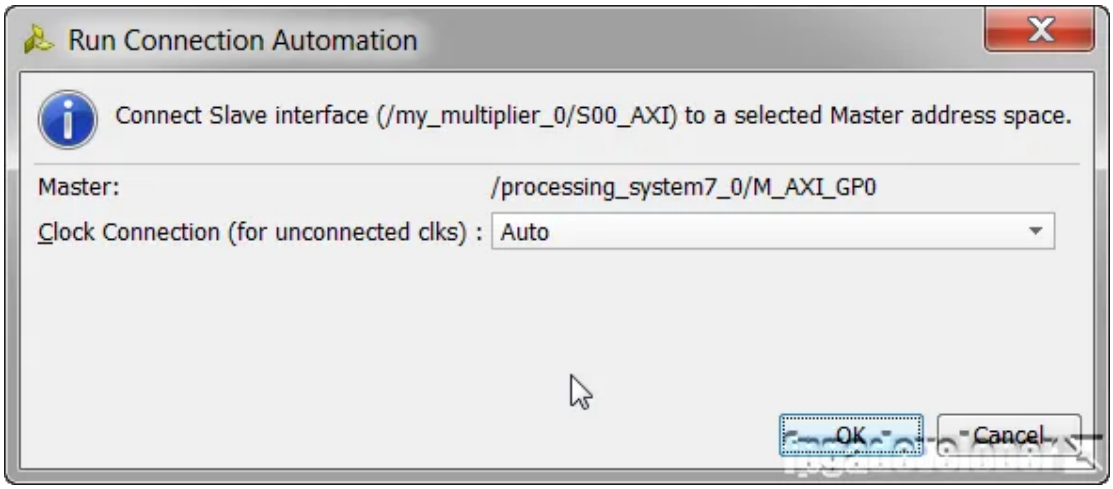


4. Click the “my_multiplier_0” peripheral from the

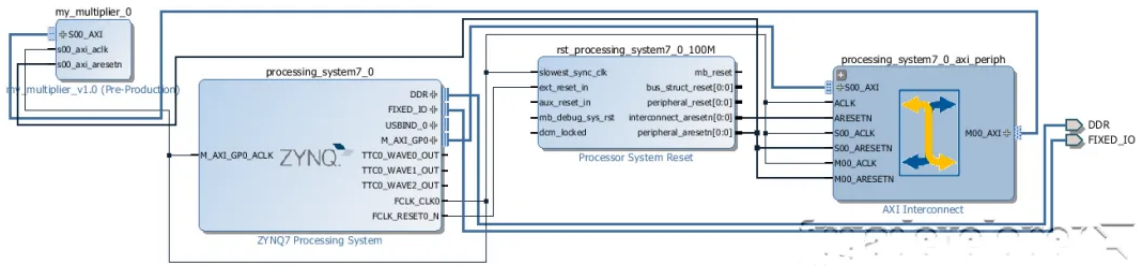
drop-down menu.



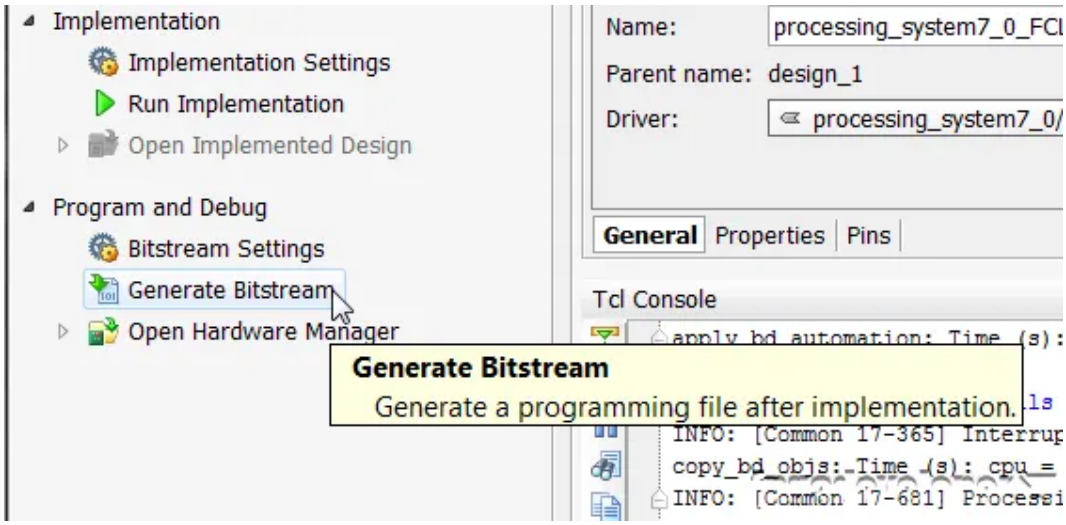
5. In the window that appears, set Clock connection to “Auto” and click “OK”.



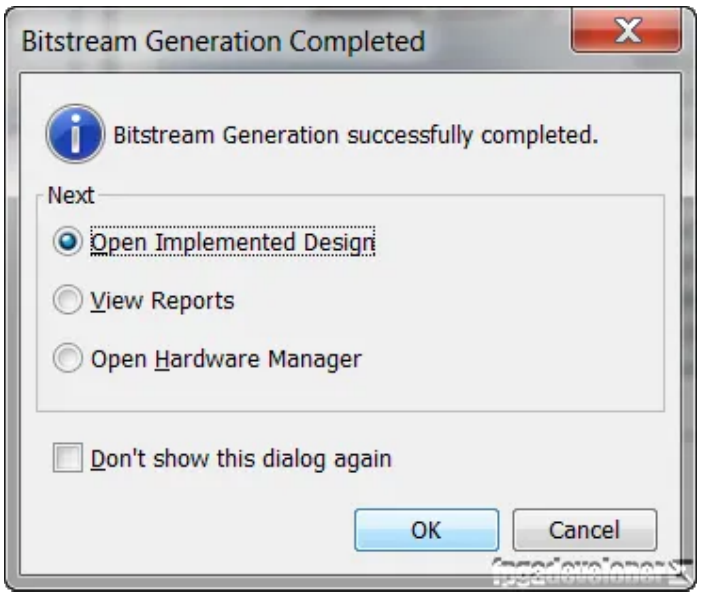
6. The new block diagram should look like this:



7. Generate the bitstream.



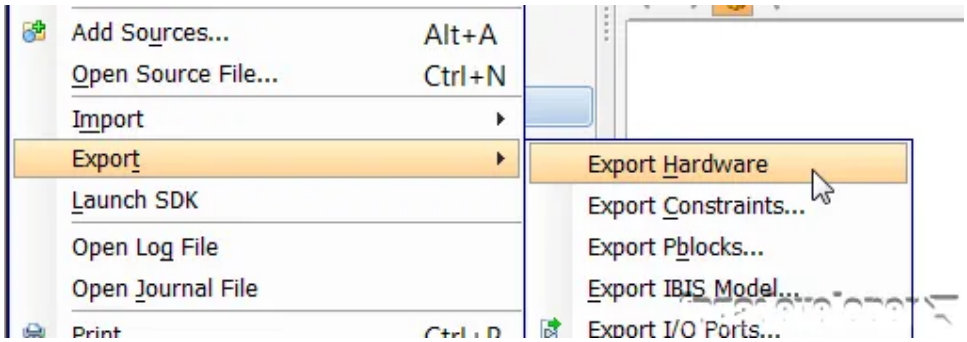
8. When the bitstream is generated, select “Open the implemented design” and click “OK”.



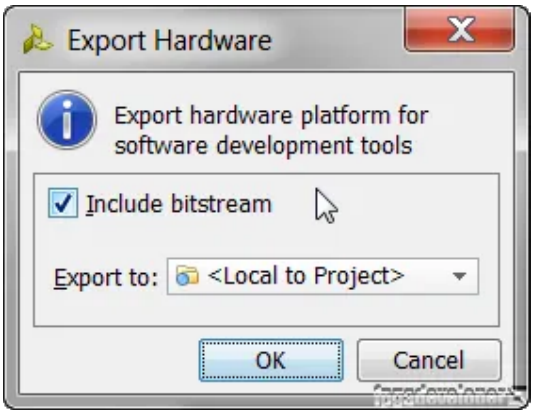
Export the hardware design to SDK

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.

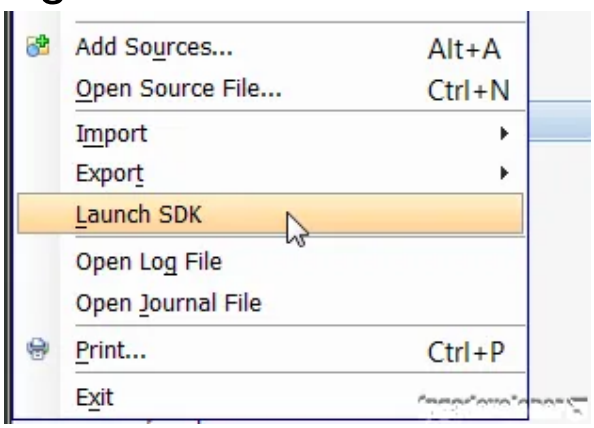
1. In Vivado, from the File menu, select “Export->Export hardware”.



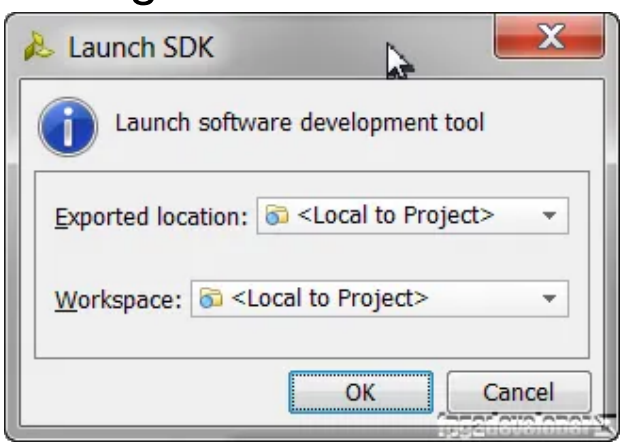
2. In the window that appears, tick “Include bitstream” and click “OK”.



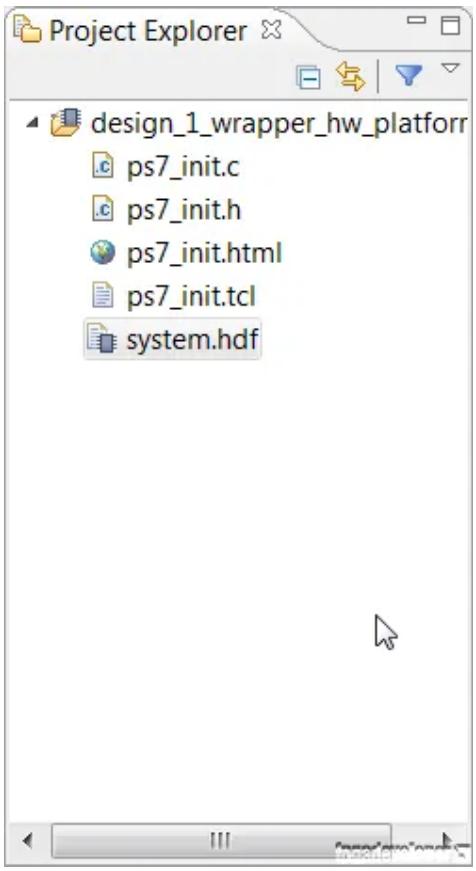
3. Again from the File menu, select “Launch SDK”.



4. In the window that appears, use the following settings and click “OK”.



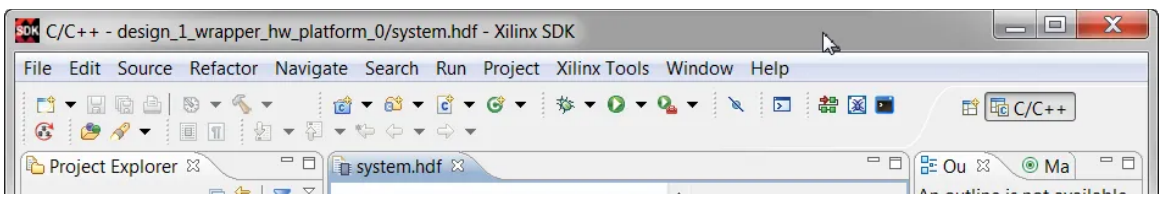
At this point, the SDK loads and a hardware platform specification will be created for your design. You should be able to see the hardware specification in the Project Explorer of SDK as shown in the image below.

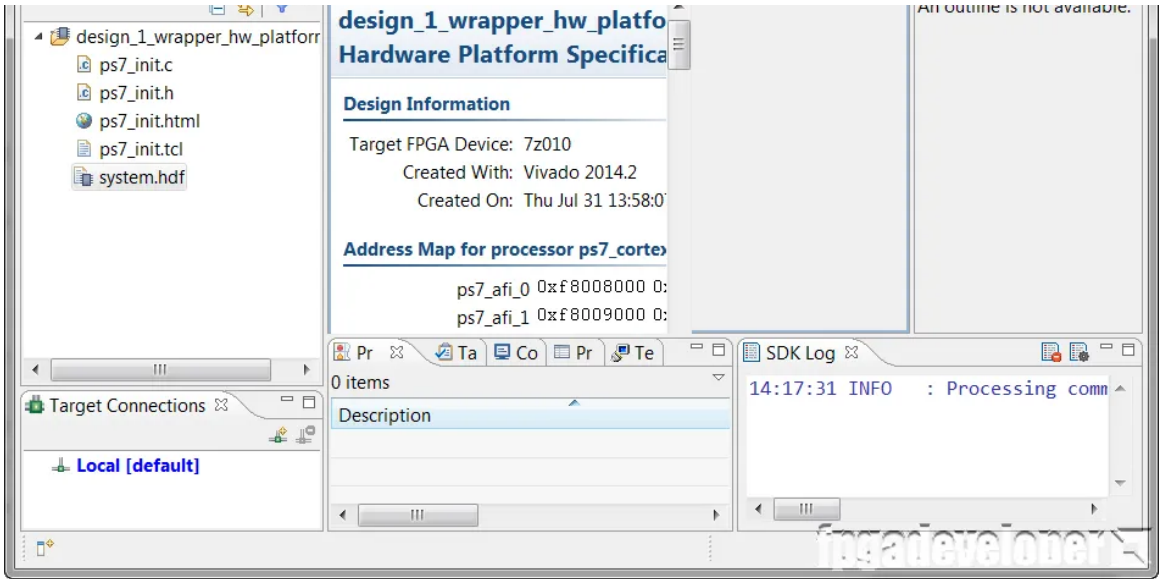


You are now ready to create a software application to run on the PS.

Create a Software application

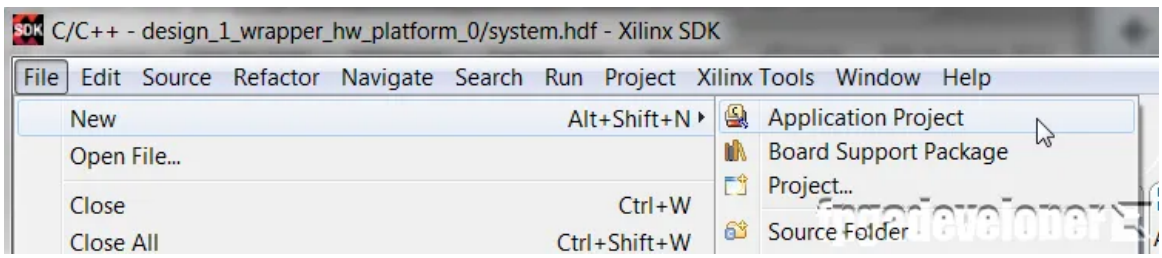
At this point, your SDK window should look somewhat like this:



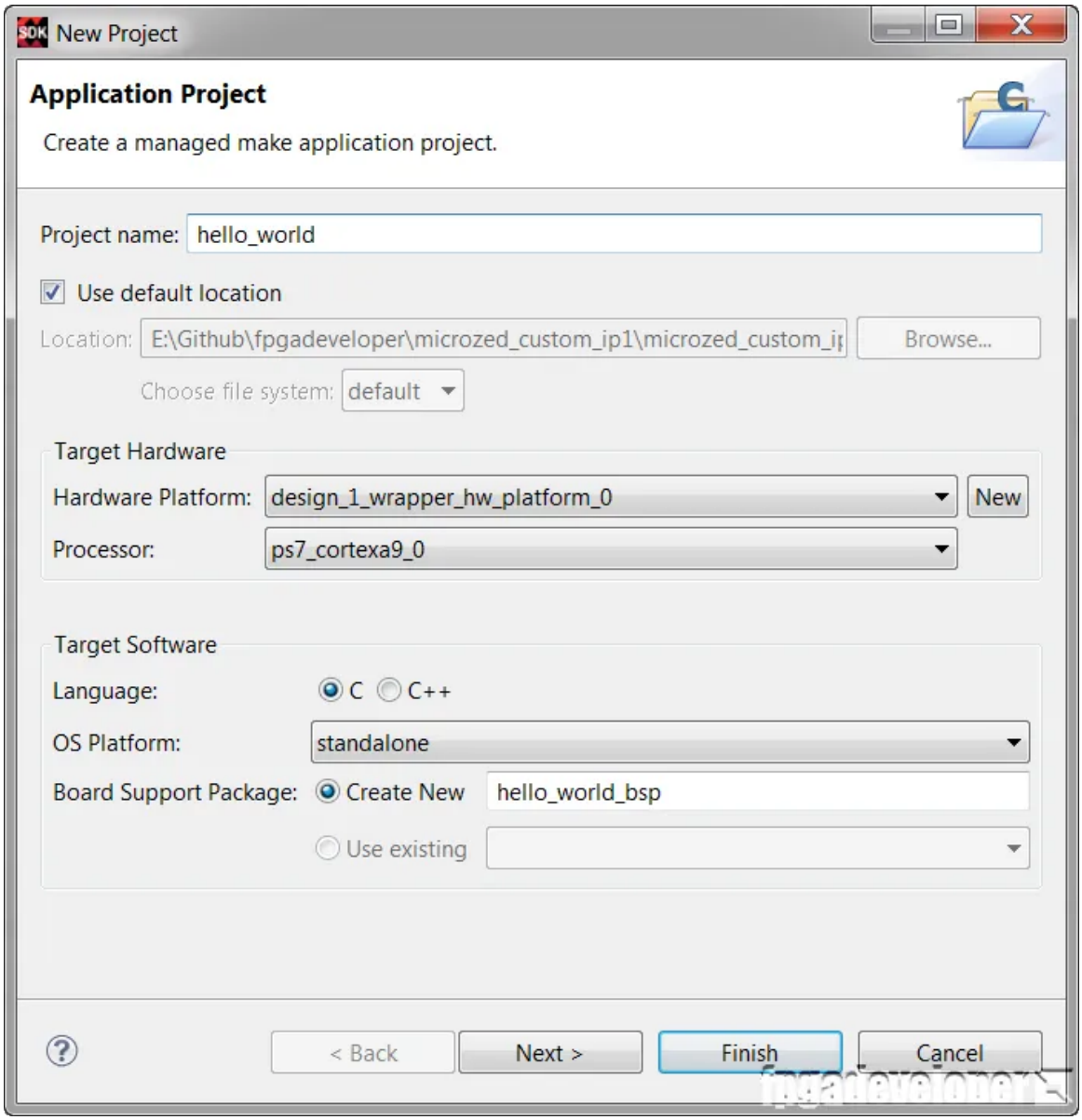


To make things easy for us, we'll use the template for the hello world application and then modify it to test the multiplier.

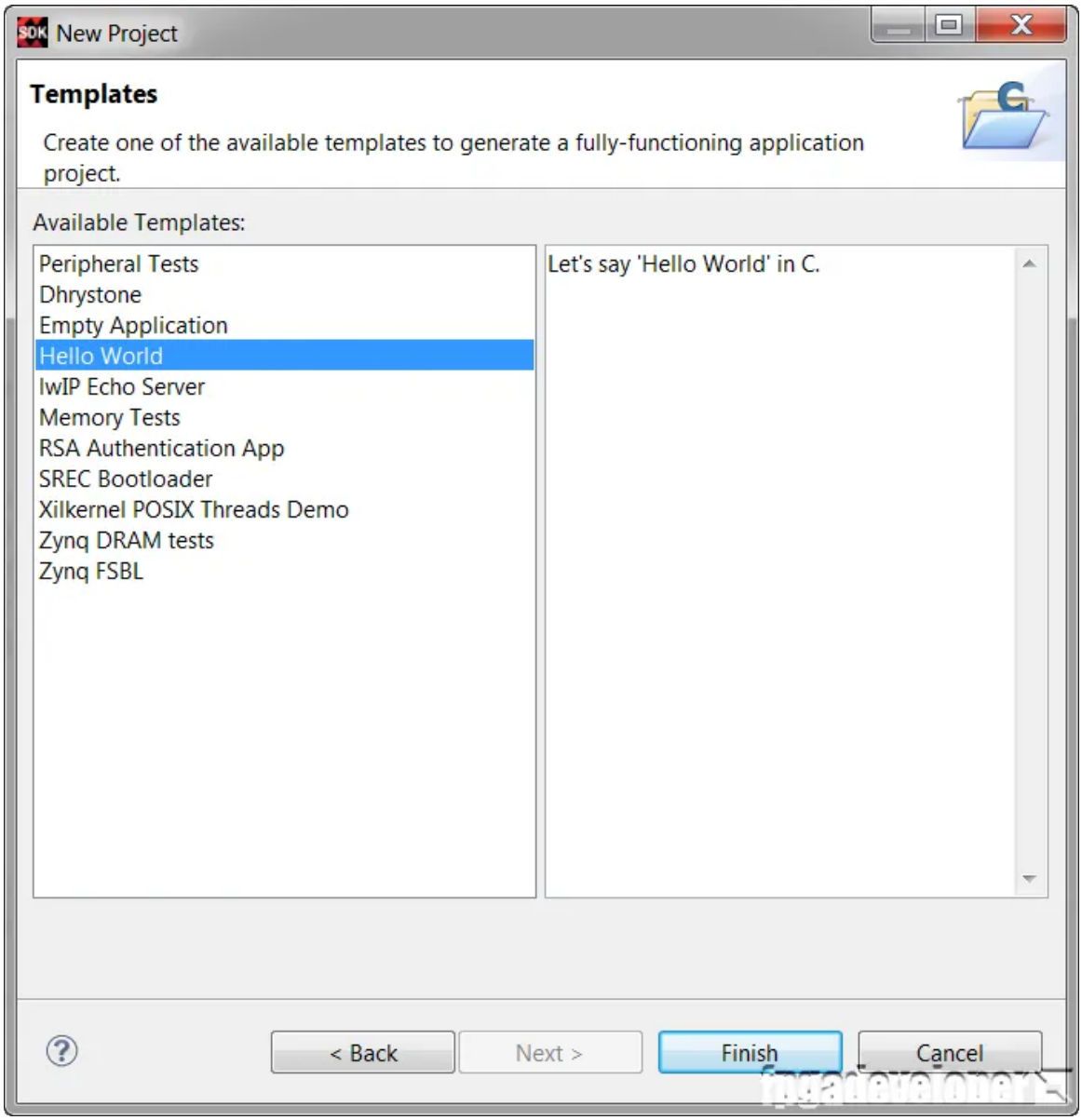
1. From the File menu, select New->Application Project.



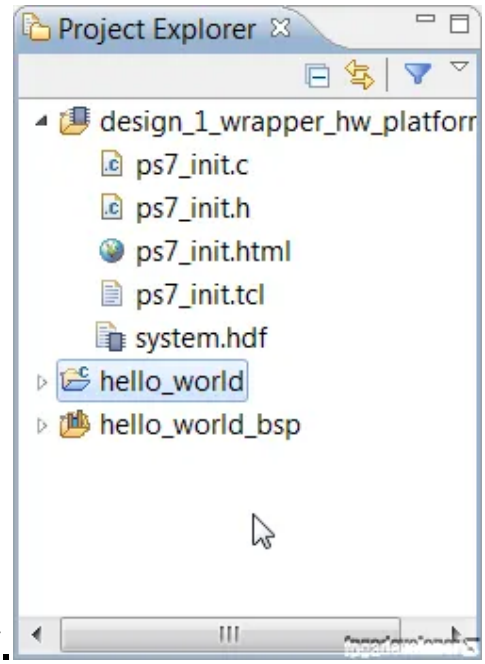
2. In the first page of the New Project wizard, choose a name for the application. I've chosen `hello_world`. Click "Next".



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



4. The SDK will generate a new application which you 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.

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"
```

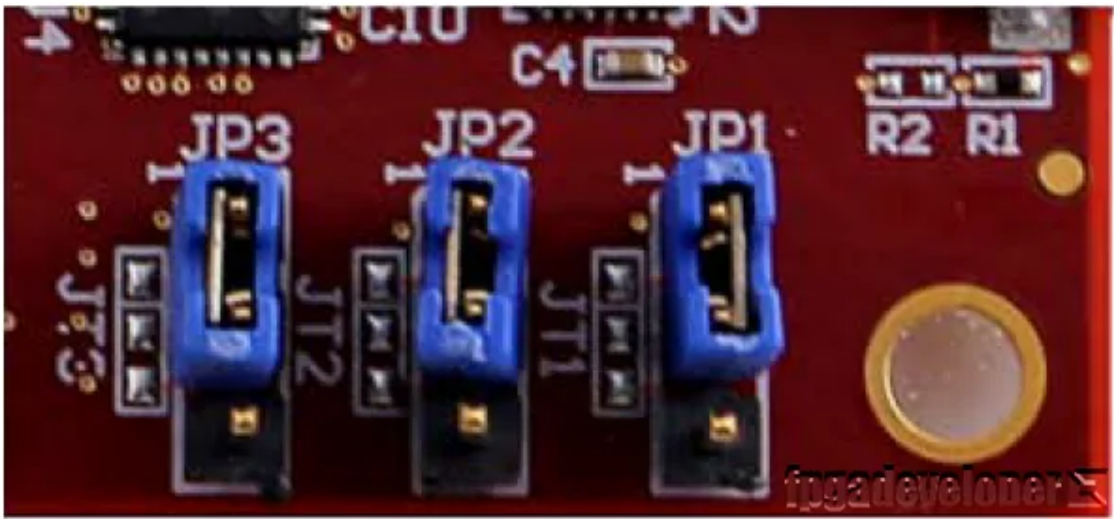
```
Xuint32 *baseaddr_p = (Xuint32
*)XPAR_MY_MULTIPLIER_0_S00_AXI_BASEADDR;

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;
}
```

Save and close the file.

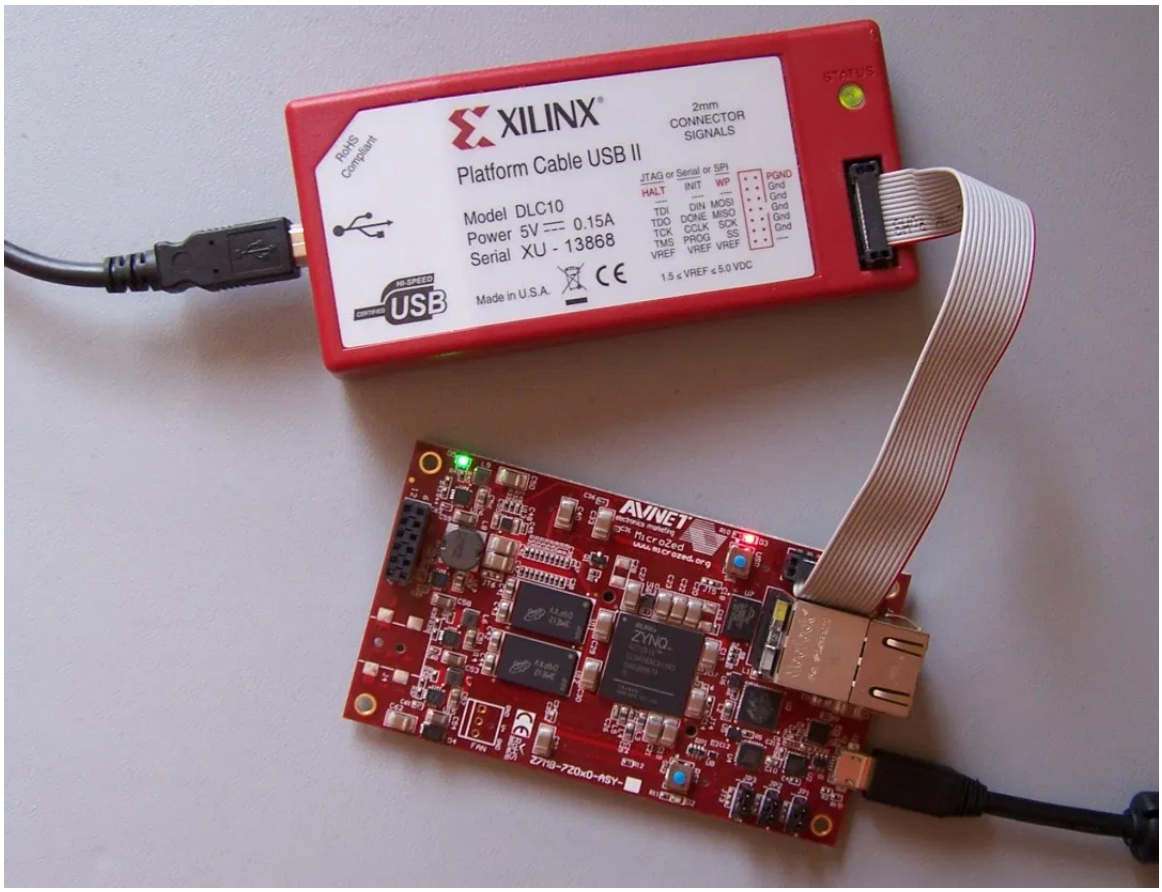
Test the design on the hardware

To test the design, we are using the MicroZed board from Avnet. Make the following setup before continuing:

1. On the MicroZed, set the JP1, JP2 and JP3 jumpers all to the 1-2 position.

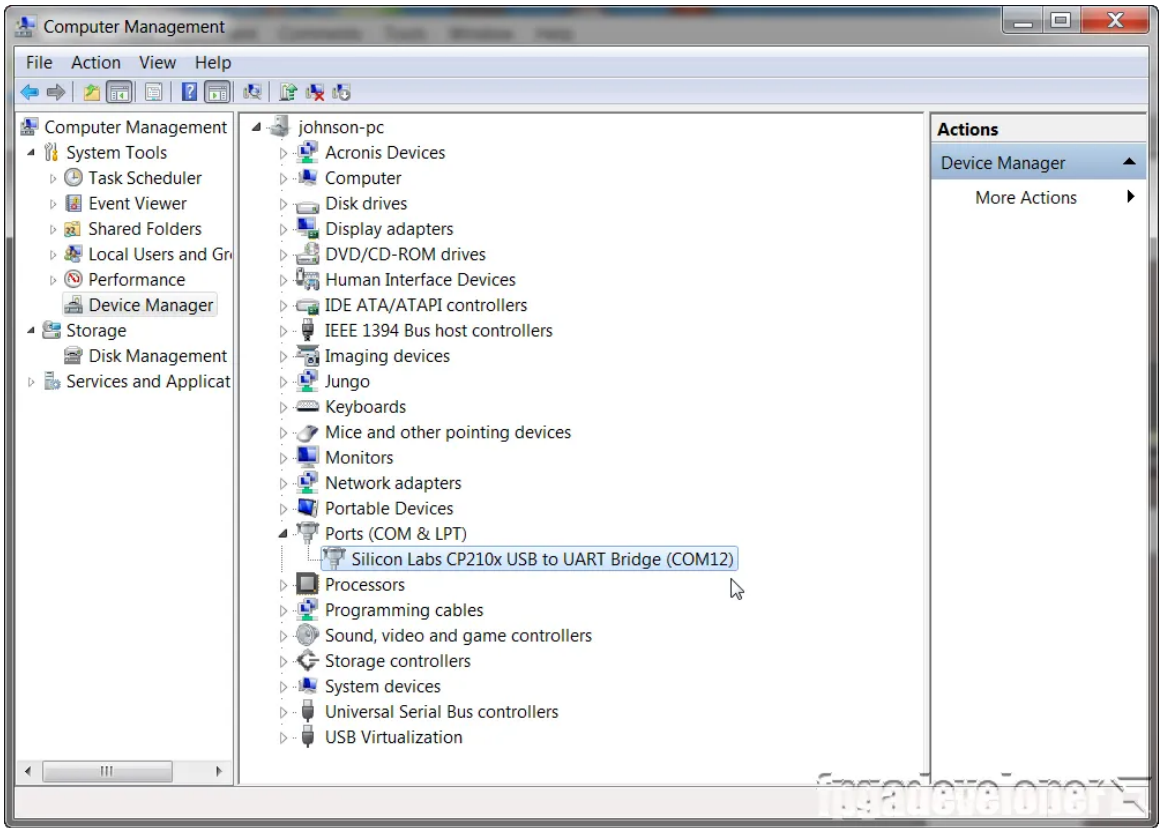


2. Connect the USB-UART (J2) to a USB port of your PC.
3. Connect a Platform Cable USB II programmer (or similar device) to the JTAG connector. Connect the programmer to a USB port of your PC.



Now you need to open up a terminal program on your PC and set it up to receive the test messages. I use Miniterm because I'm a Python fan, but you could use any other terminal program such as Putty. Use the following settings:

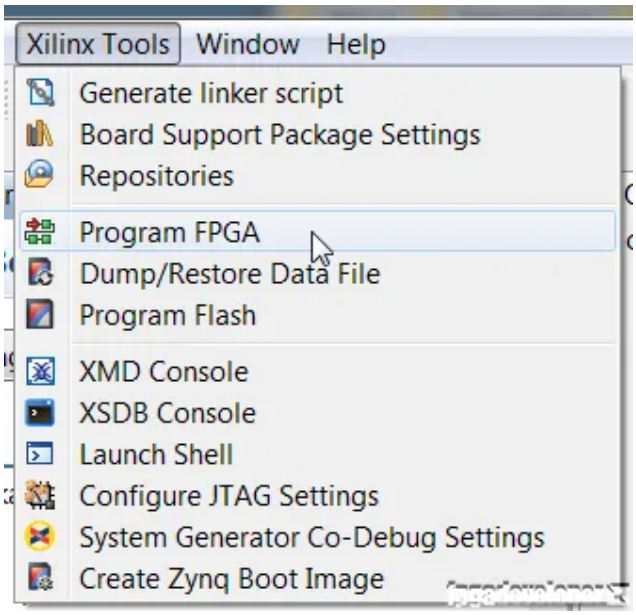
- Comport – check your device manager to find out what comport the MicroZed popped up as. In my case, it was COM12 as shown below.



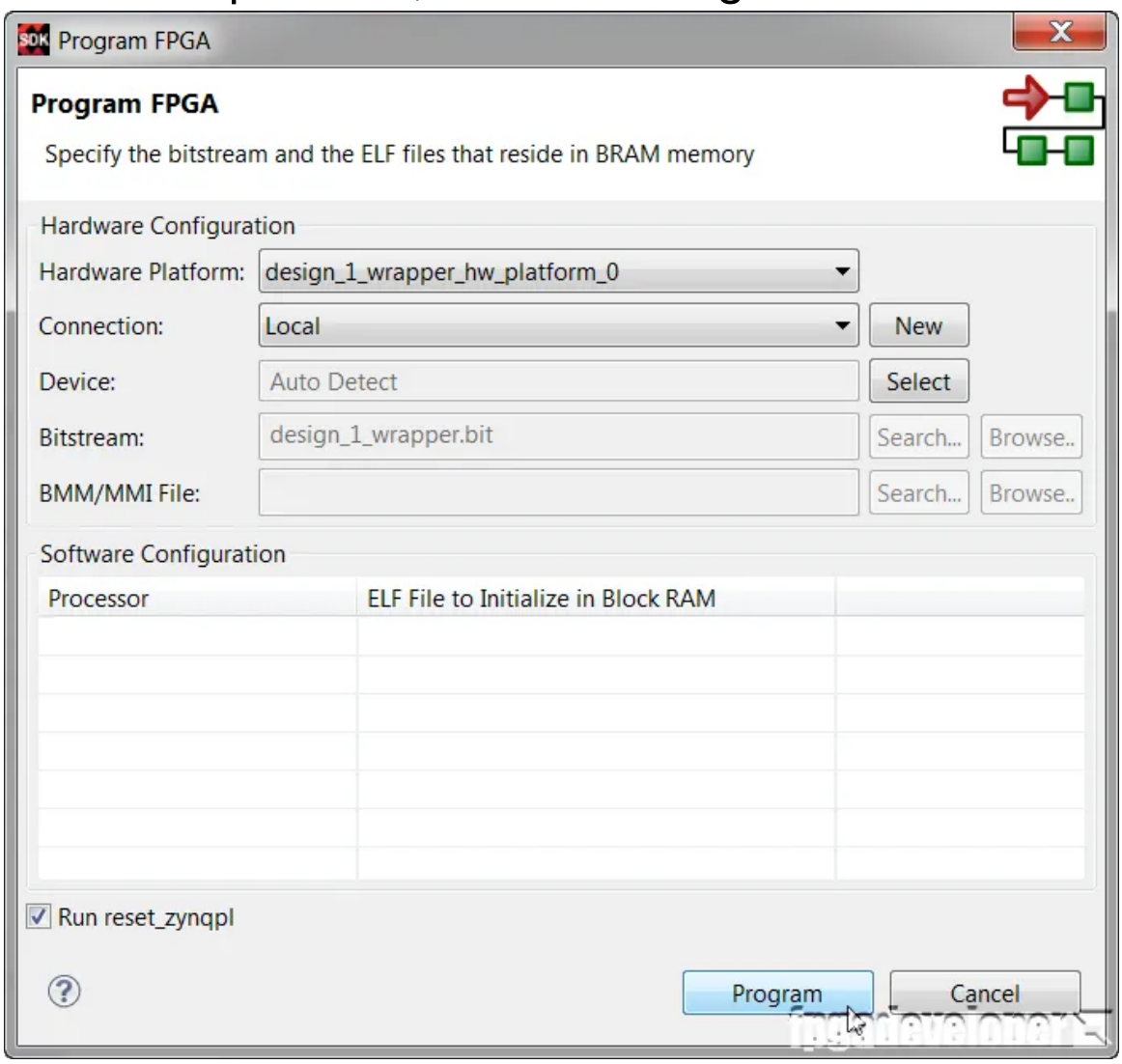
- Baud rate: 115200bps
- Data: 8 bits
- Parity: None
- Stop bits: 1

Now that your PC is ready to receive the test messages, we are ready to send our bitstream and software application to the hardware.

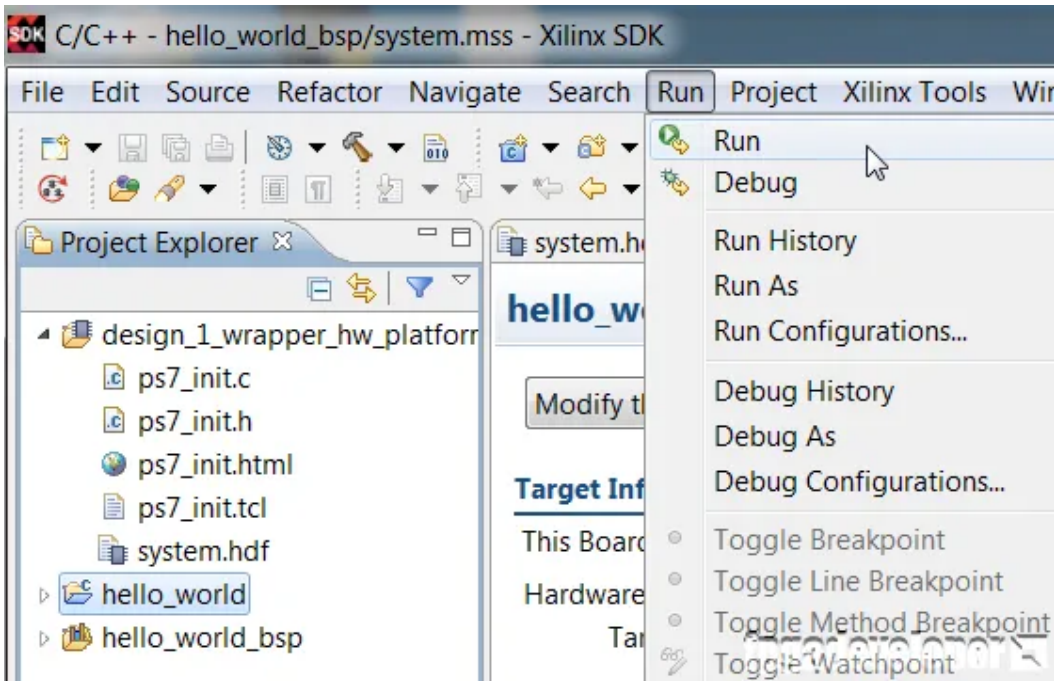
1. In the SDK, from the menu, select Xilinx Tools->Program FPGA.



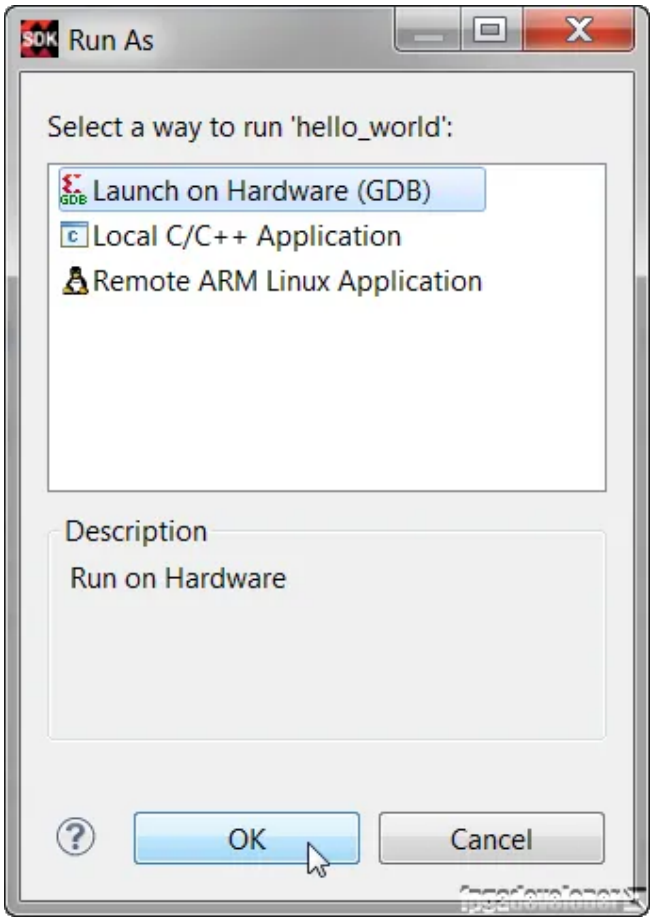
2. In the Program FPGA window, we select the hardware platform to program. We have only one hardware platform, so click “Program”.



3. The bitstream will be loaded onto the Zynq and we are ready to load the software application. Select the `hello_world` folder in the Project Explorer, then from the menu, select Run->Run.



4. In the Run As window, select “Launch on Hardware (GDB)” and click “OK”.



5. The application will be loaded on the Zynq PS and it will be executed. Look out for the results in your terminal window!

We're sending two 16-bit inputs 0x02 and 0x03 and the result is 0x06 as expected.

Source code

The TCL build script and source code for this project is shared on Github here:

https://github.com/fpgadeveloper/microzed-custom-ip

For instructions on rebuilding the project from sources, read my post on version control for Vivado projects.

Jeff is passionate about FPGAs, SoCs and high-performance computing, and has been writing the FPGA Developer blog since 2008. As the owner of Opsero, he leads a small team of FPGA all-stars providing start-ups and tech companies with FPGA design capability that they can call on when needed.