

Lab 3. Adding a Custom Hardware IP, and interfacing it with Software

Objective

In this lab, we will add a Custom hardware IP (a user-defined Verilog block), which will be implemented on the FPGA and interface it to the software running on the PowerPC.

- A Custom IP (Verilog code) is used to implement a multiplier. The Verilog code reads the values from two registers (say R1 and R2) and writes the multiplied output to a third register (R3).

- Software is used for testing the multiplier. The software code will write values to registers R1 and R2, read the multiplication result from R3, and display the values of R1, R2 and R3 on a hyperterminal screen.

Overview

Create a Base System (or use the one created in Lab 2) consisting of a PowerPC, RS232 interface and a PLB BRAM. In this lab, we will add a Custom IP – which will multiply 2 32-bit numbers stored in 2 registers and write the result to a third register.

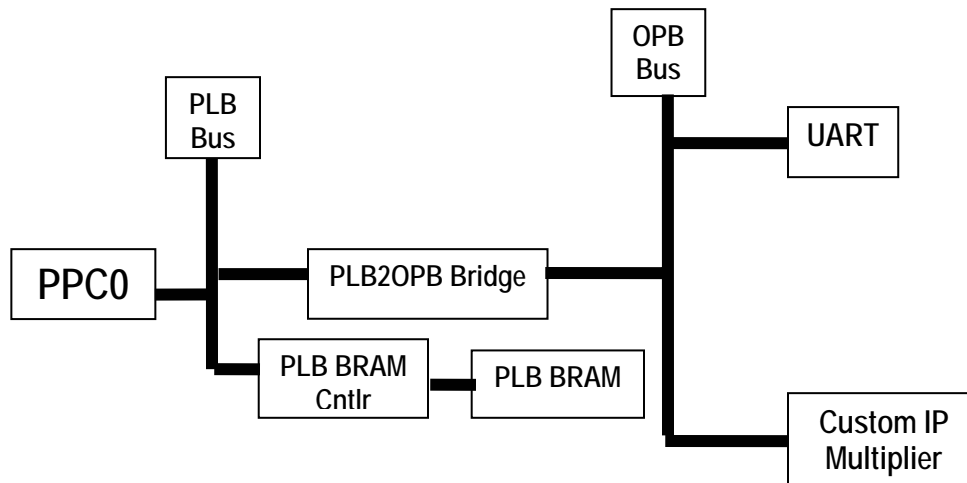


Figure 1. Complete System

Procedure

Copy the base system created in Lab 2 or create a new base system in a directory called lab3.

After the design is setup using the Base System Builder (BSB), add the Custom IP using the following steps.

1. Under the Hardware tab, select 'Create/Import Peripheral'. Click Next on the Welcome window. Select 'Create templates for a new peripheral'. Click Next.

In the Repository or Project window, select 'To an XPS project' and browse to the directory where your directory Lab3 is located. Click Next.

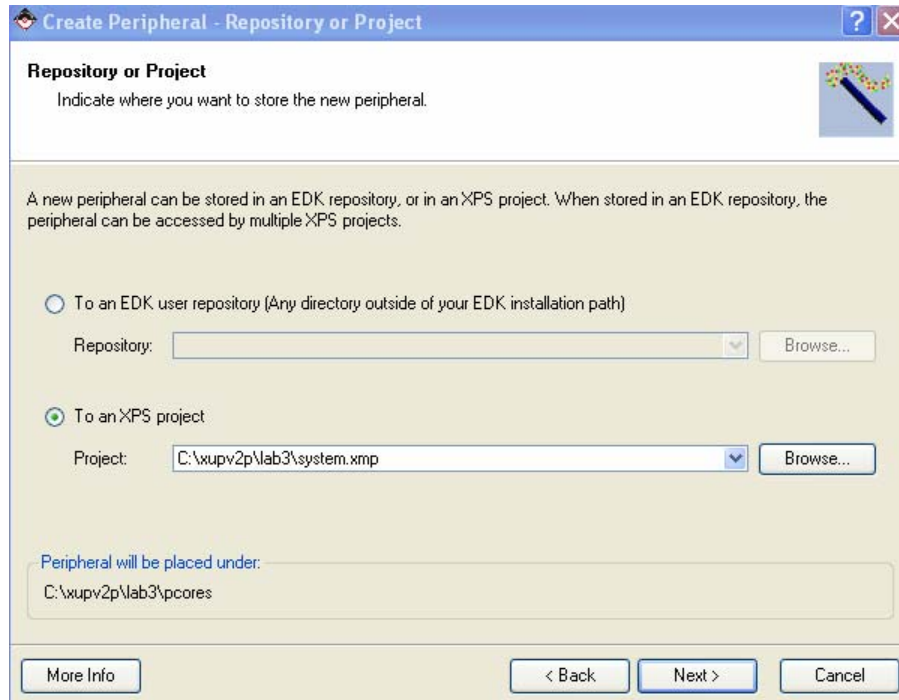


Figure 2. Create/ Import Peripheral

2. Name the new peripheral 'multiply' (this will be the name of your Verilog module). The rest of the settings remain unchanged. Click Next.

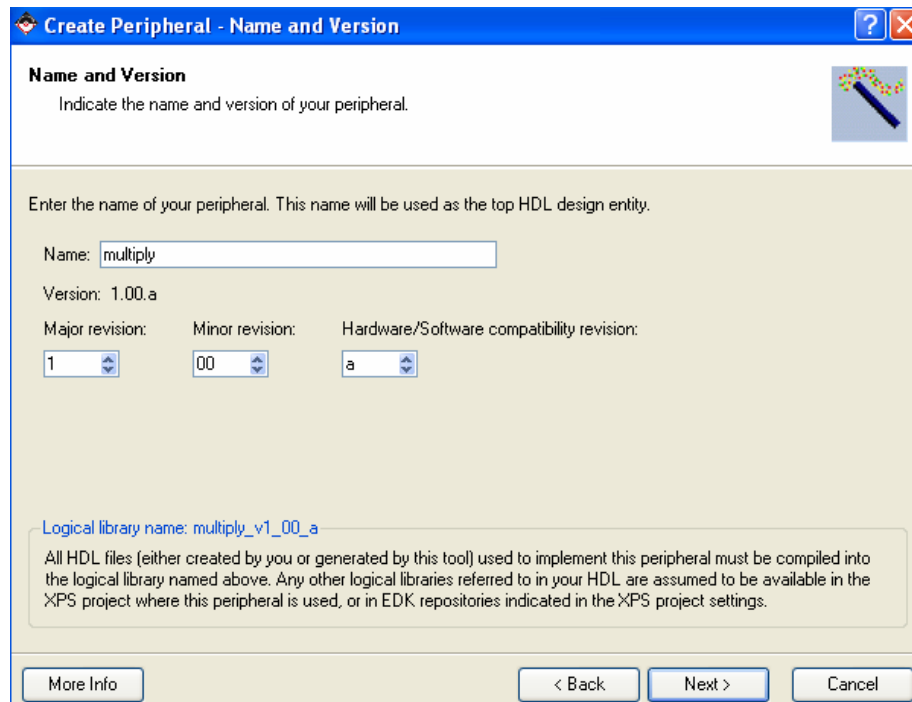


Figure 3. Name and version

- Now, the Bus Interface Window will open. We want to connect the IP to the On-chip Peripheral Bus (OPB). Select OPB and click Next.
In the IPIF Services window, check 'User logic S/W register support'. Our custom IP will interface with the software using these software registers. Click Next.

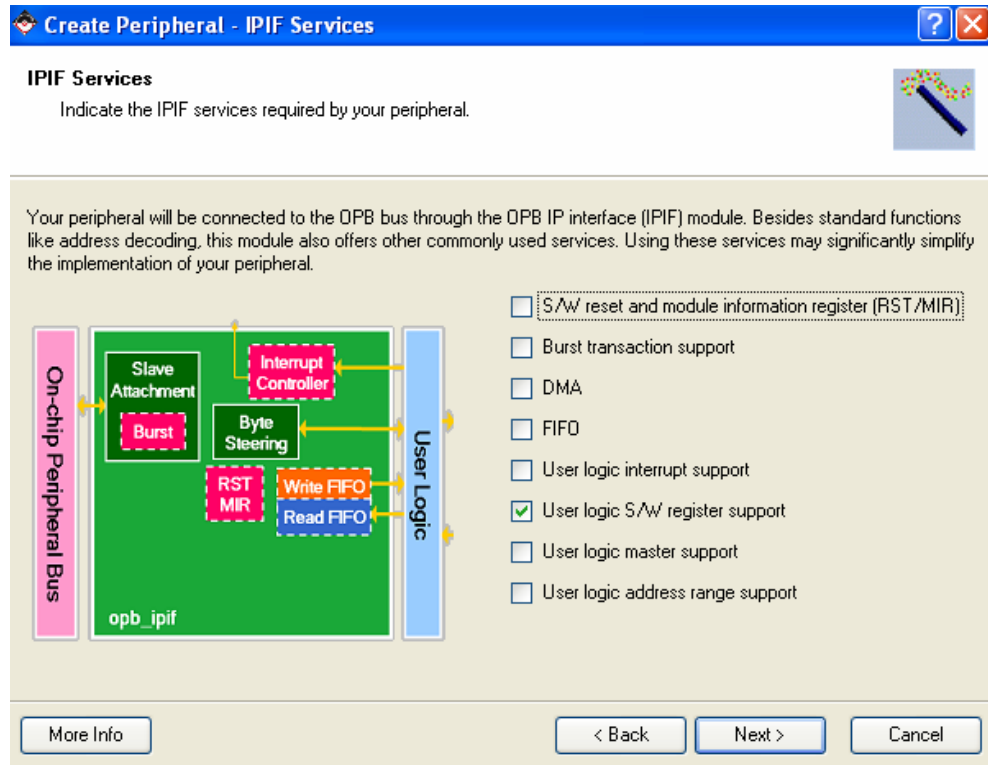


Figure 4. IPIF Services

- In the User S/W Register window, set the 'Number of software accessible registers' to 3, and set 'Data width' of each register to 32. We need two 32-bit registers to hold the numbers to be multiplied, and a third 32-bit register to hold the multiplication result.
Note that we can only specify the register widths to be identical. When we use the 'Multiply' IP, we will ensure that the multiplier and multiplicand are 16 bit values. Click Next.
- In the IP Interconnect (IPIC) window, some ports are selected by default for the custom IP. These ports are used for communication between the user logic and OPB bus. Accept the default setting and click Next.
- In the window that shows up next, uncheck 'Generate BFM simulation platform for ModelSim-SE or ModelSim-PE'. We do not need the simulation support for this exercise. Click Next.

- In Peripheral Implementation Support window (shown in Figure 5), click on 'Generate stub 'user_logic' template in Verilog instead of VHDL'. A Warning message pops up. The Warning message states that even though the user_logic stub will be created in Verilog, it will have a wrapper around it written in VHDL. XPS uses VHDL for interfacing with rest of the peripherals. Click OK. We will be making our changes in the Verilog file 'user_logic.v'. This Verilog file can be found in 'pcores/multiply_v1_00_a/hdl/verilog' in your lab3 directory.

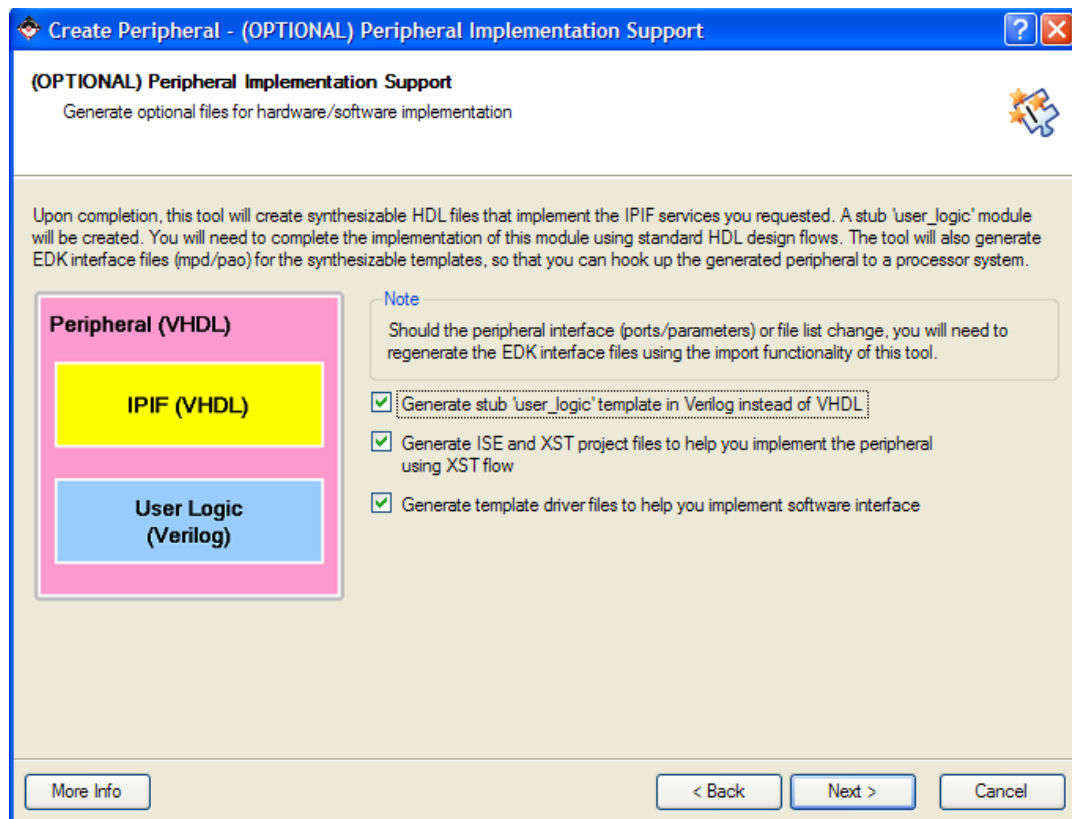


Figure 5. Select Verilog Stub

- Open the user_logic.v file and notice the three registers *slv_reg0*, *slv_reg1* and *slv_reg2* declared as 'reg [31:0]'. Also, find Bus2IP_Clk and Bus2IP_Reset in the input port list. Add the following code to the verilog file to implement the multiplier.

```

always @(posedge Bus2IP_Clk)
  if(Bus2IP_Reset == 1)
    slv_reg2 = 0;
  else
    slv_reg2 = slv_reg0 * slv_reg1;

```

Also, comment out the portion of the code where *slv_reg2* is being used in the

first always block. In a Verilog code, a 'reg' cannot be written to in two always blocks.

9. We have created the IP and will now import it to XPS. From the 'Hardware' tab, select 'Create or Import Peripheral'. In the Peripheral Flow window, select 'Import existing peripheral' and click Next.
10. In the Name and Version window, choose 'multiply' from the drop-down list. Check 'Use version' and accept the default version number 1.00.a. Click Next.

In the Source File Types window, check HDL source files. Click Next.

11. In the HDL Source Files window, select User existing Peripheral Analysis Order file (*.pao) to locate HDL source files and dependent library files. Browse to lab3\pcores\multiply_v1_00_a\data\multiply_v2_1_0.pao (Figure 6). Use HDL language as *Mixed* to implement your peripheral. Click Next.

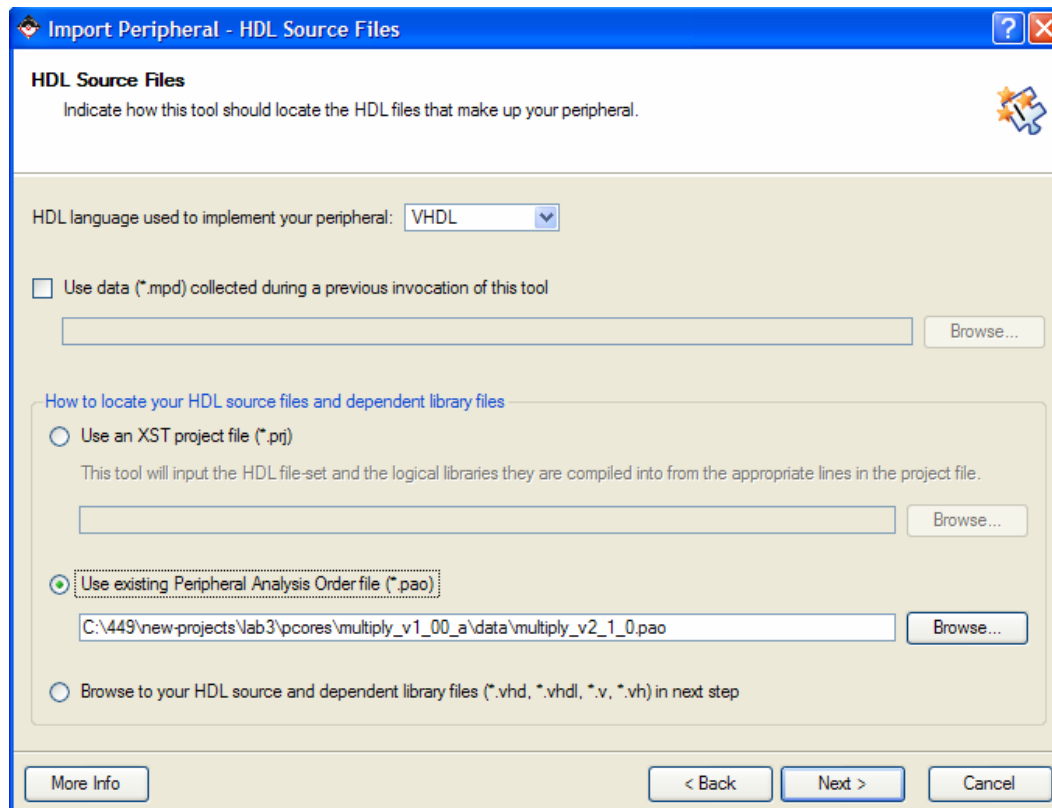


Figure 6. HDL Source Files

Now, the HDL analysis information window appears. Click Next.

12. If there is no error in the code, you should see the Bus Interface window as shown in Figure 7. Check OPB Slave (SOPB) under On-chip Peripheral Bus Interface. Click Next.

A Congratulations window should appear on completion. Click Finish.

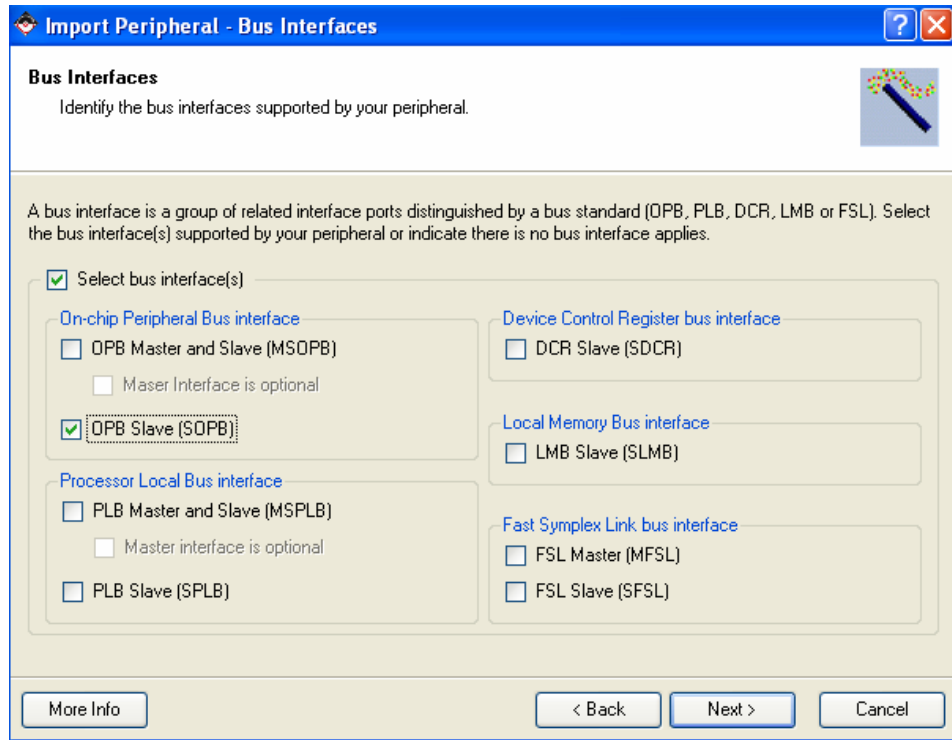


Figure 7. Import Peripherals – Bus Interfaces

13. Now, “Multiply” should appear in the IP Catalog under Project Repositories. Right click on ‘multiply’ and click on Add IP.

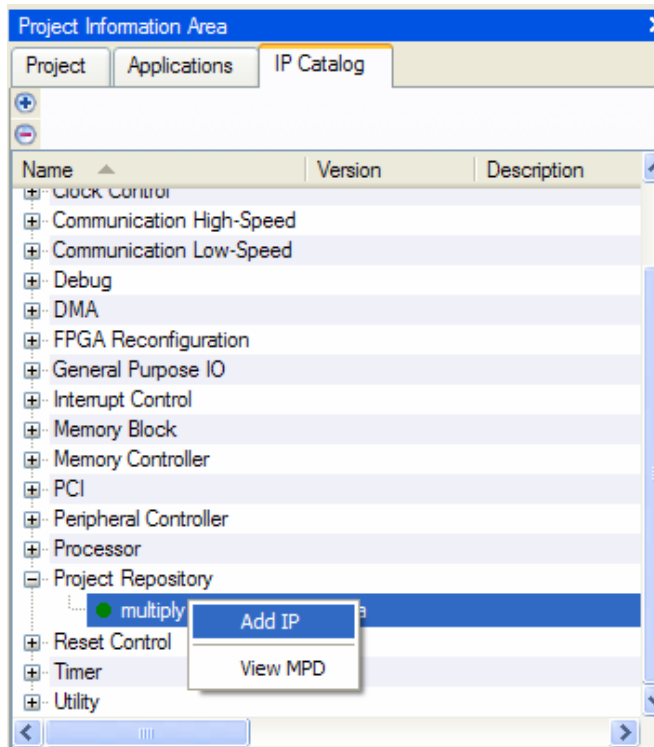


Figure 8. Add IP 'Multiply'


14. In Bus Interfaces in 'System Assembly', connect the multiply_0 instance to the OPB Bus.

Now, Select 'Ports' and select 'All' from Connection Filters to see all available ports. Change the Net name of 'OPB_Clk' to 'sys_clk_s' for the "multiply_0" instance.

Now, select 'Addresses' tab. Set the 'Size' of the multiply_0 instance to 64KB and click Generate Addresses.

15. Click on Hardware → Generate Netlist.

Now, the Hardware part is complete. Write a testbench in Software to test the multiplier.

16. Click  or select Software -> Generate Libraries and BSPs to generate libraries for the project.

17. Now, add a source file in the Application Tab (follow steps from Lab2). The source file (testbench) should write values to the registers 'slv_reg0' and 'slv_reg1' and read the multiplication result from 'slv_reg2'. The value stored in each of these three registers should be printed on the hyperterminal screen.

Some hints for the .c file:

- You will need to include xparameters.h and multiply.h (which are located in lab3/ppc405_0/include directory).

- Look for functions to read and write to a register in multiply.h
- The `RegOffset` value for `'slv_reg0'` is 0, and the three 32-bit registers are consecutively located in memory.
- Use a 'for' loop to write different values (varying from 0 to 7) in `'slv_reg0'` and `'slv_reg1'` and read the corresponding multiplication result from `'slv_reg2'`, and print the three values on the hyperterminal screen.

Deliverables

Submit a report containing the following information:

- Your commented .c file [2 points]
- Commented verilog code [2 points]
- The output seen on the hyperterminal[2 points]
- How long does it take for the content of `'slv_reg2'` register to appear on hyperterminal screen after the values are written to the registers `'slv_reg0'` and `'slv_reg1'`. You can assume that you are just printing `'slv_reg2'` register value on the hyperterminal. [4 points]