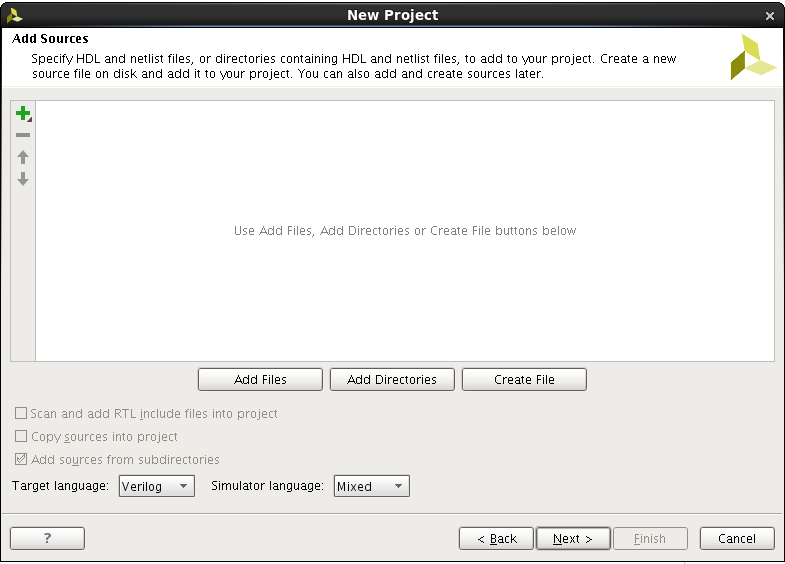
**Lab 9: Demonstration of IP Integrator**

**Exercise 9.1: Refer to the video “Lab\_9\_IP\_Integrator” uploaded in the drive**

**Exercise 9.2: Implement a 2-bit x 2-bit Multiplier using IP Integrator. Follow the steps shown below.**

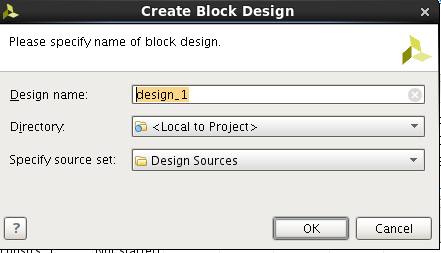
1. Follow same process as earlier labs for creating a project. Do not add any resources to the project by clicking “Next” instead of “Create File” when the following “Add Sources” window appears.



1. Once the Project is created, Click on “Create Block Design” which appears on the “Flow Navigator” window.



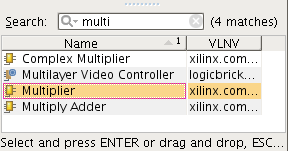
1. Give a proper name to the design. Here the name specified is design\_1.



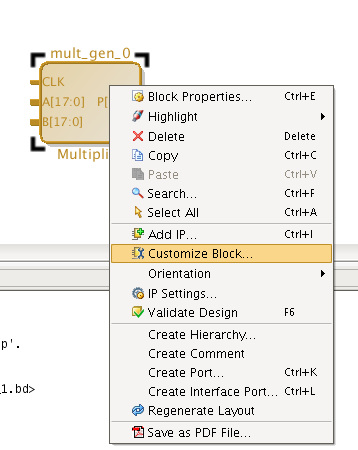
1. This will create a block design window in the right hand side. Click on “Add IP” option shown below.



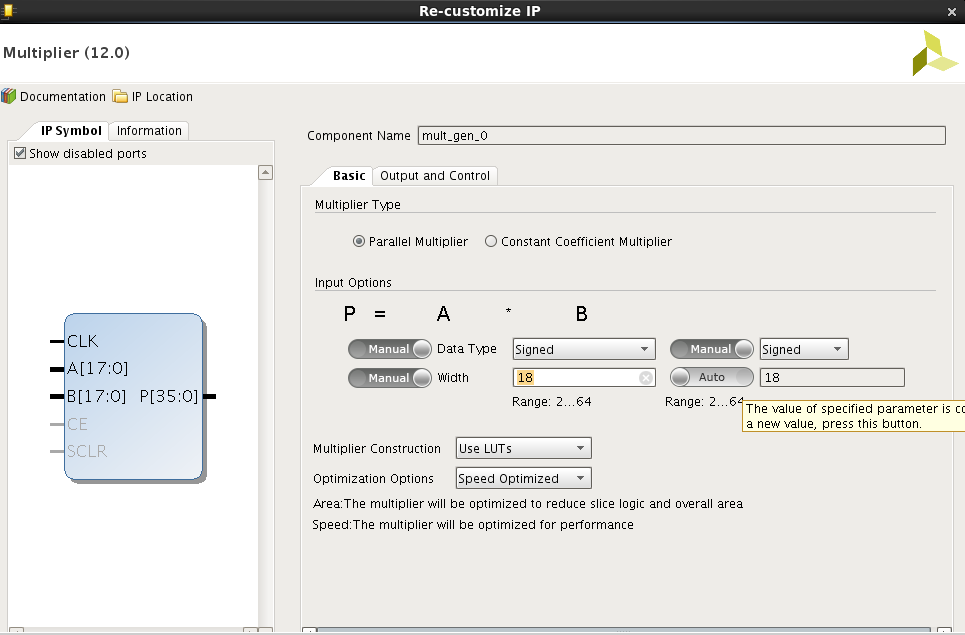
1. After clicking on ‘Add IP”, an IP catalog window will be opened where you can select and add IPs that are needed for the design. In this example add “Multiplier”.

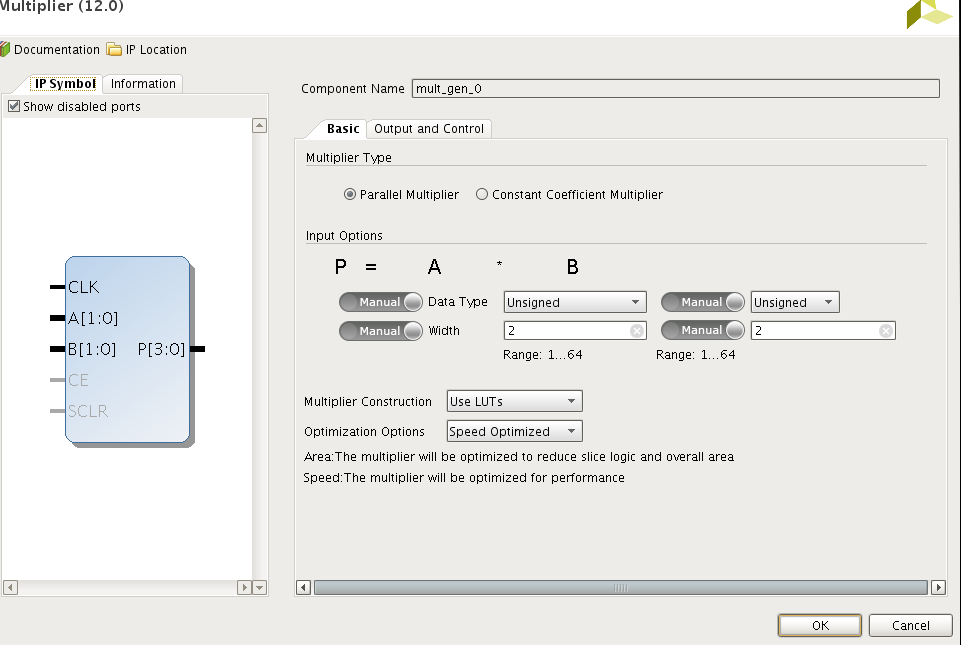
****

1. This will create a default multiplier IP block. To customize this block as per your needs, right click on the IP block and select “Customize Block”.

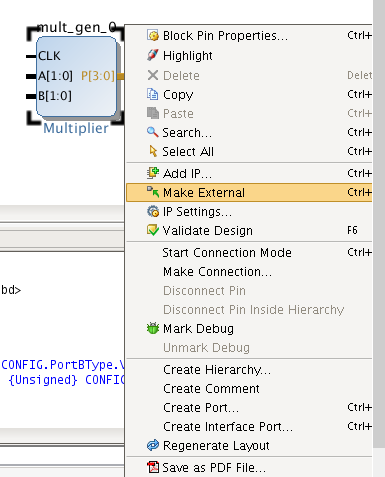
****

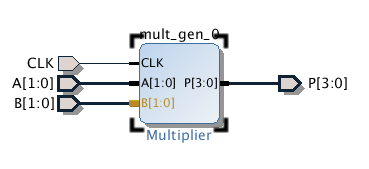
1. Customize the multiplier for 2x2 unsigned multiplication. This design requires clock.

****

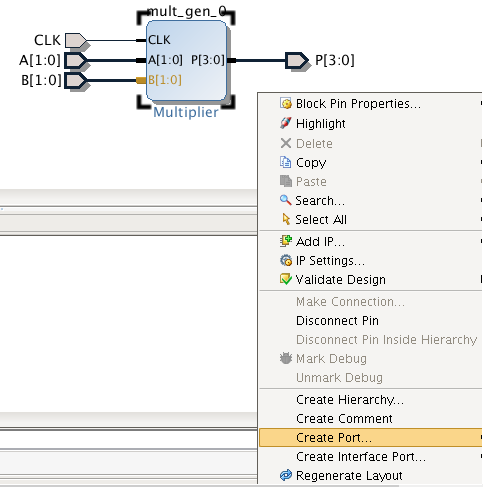
****

1. Now you have to add ports to the pins of IP by right clicking on the corresponding nodes and select “Make External”. Repeat this step for all nodes.

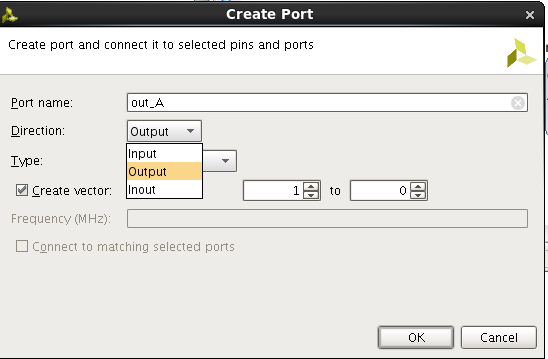
****

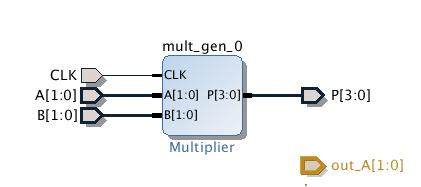
****

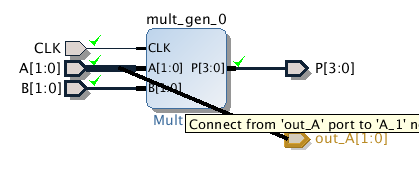
1. You can also add extra ports if necessary. For example if you want to route any input/intermediate signal as output. Right click anywhere in design window and select create port.

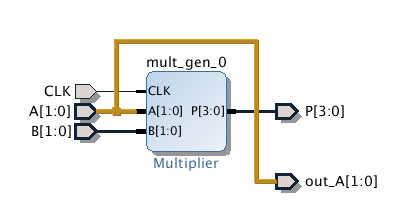
****

1. You can configure the port to be input/output. Here the input A is directly connected to output out\_A. Connection between nodes can be made by right click (on a pin), drag and connect to another.

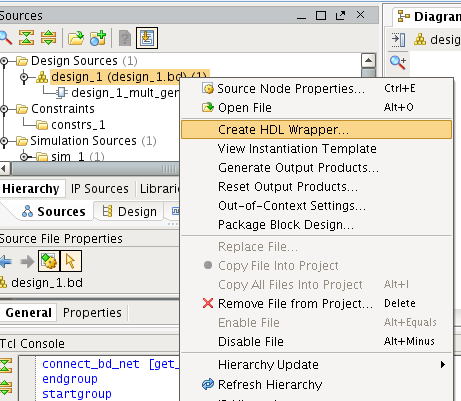
****

****

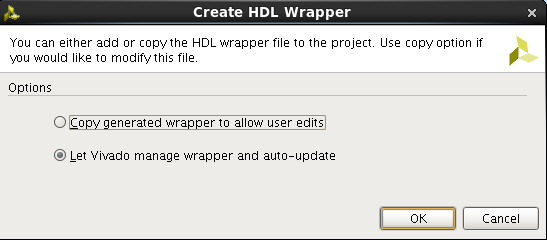
****

****

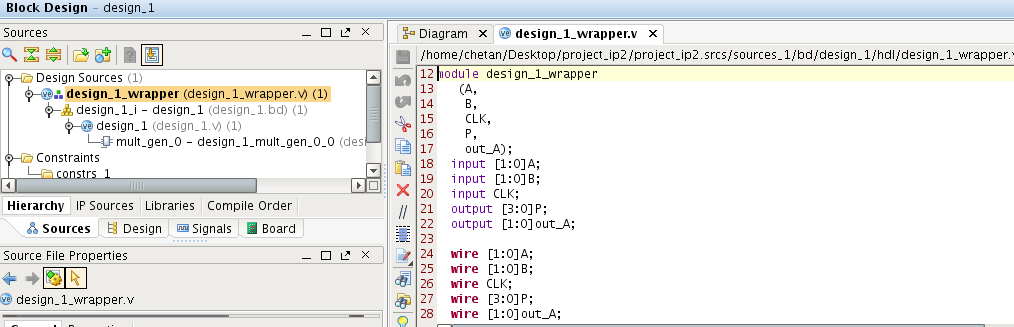
1. Now the design is complete. To be able to use and test this design you have to create a Verilog wrapper. This is done by right clicking on design\_1.bd in sources window and select “Create HDL Wrapper”.

****

1. You can either copy the generated Verilog file and reuse in other modules or let vivado manage it. Here let Vivado manage the wrapper by clicking ok for the default option as shown below.

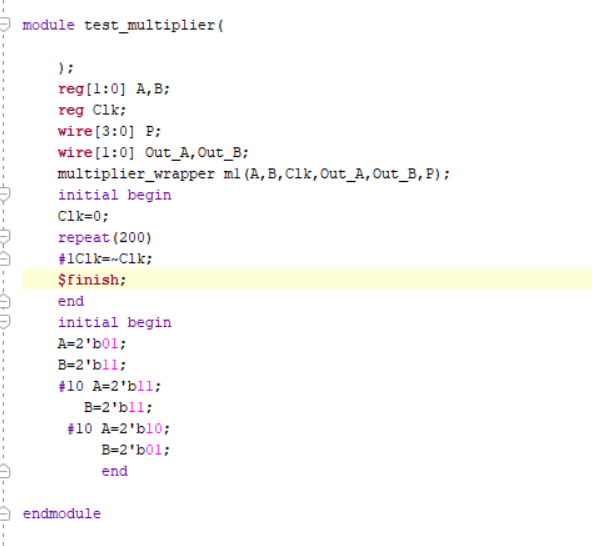
****

1. The Verilog module is generated which has instantiation of the multiplier IP.

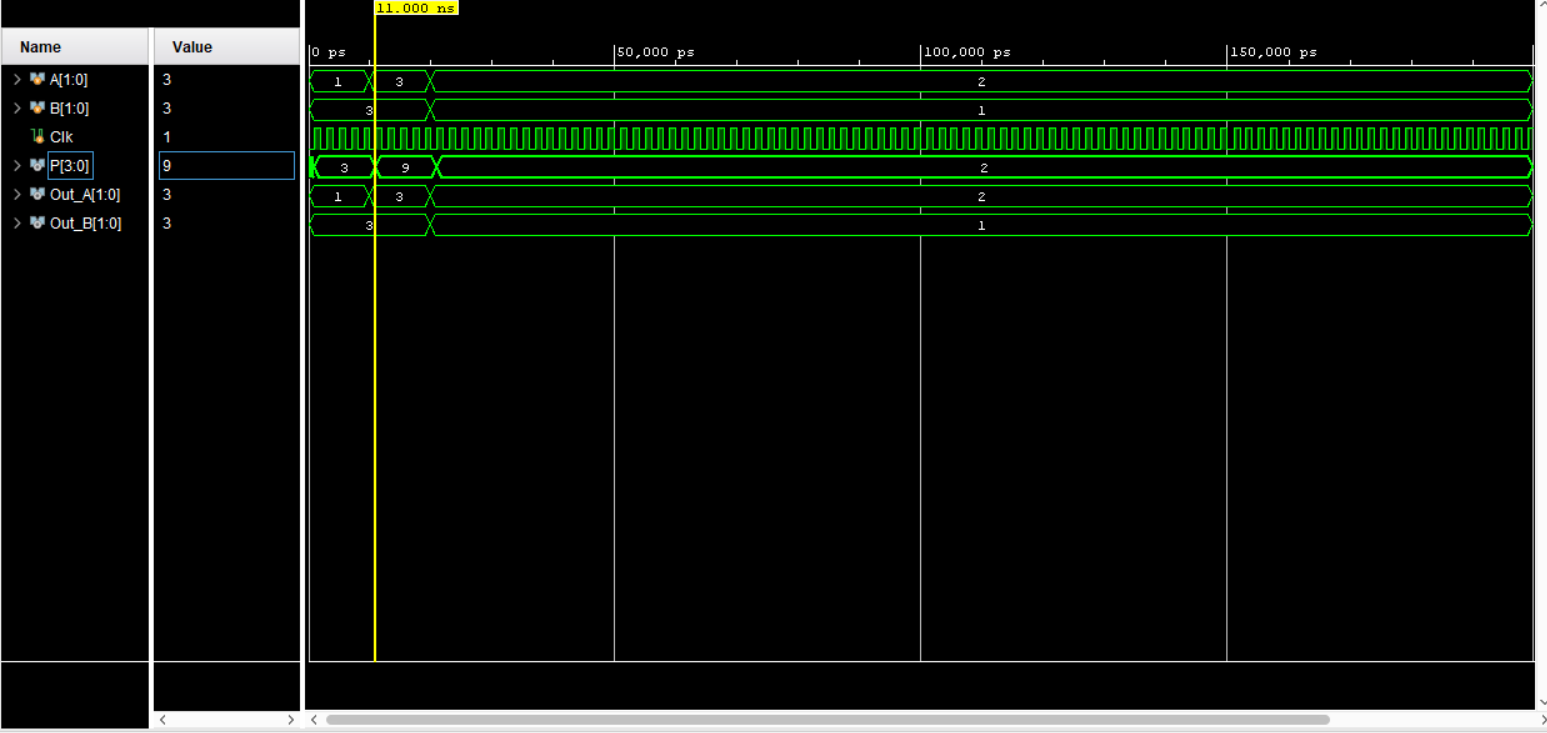
****

1. After this the remaining steps will be same as seen in previous labs. (Note: Clock is need for the design. CLK can be of 100 MHz and there is no need for clock division here).
2. Write appropriate test bench **Test\_Design.v** and test the **design\_1\_wrapper.v** module.

**Question: Paste the image test bench code Test\_design.v.**

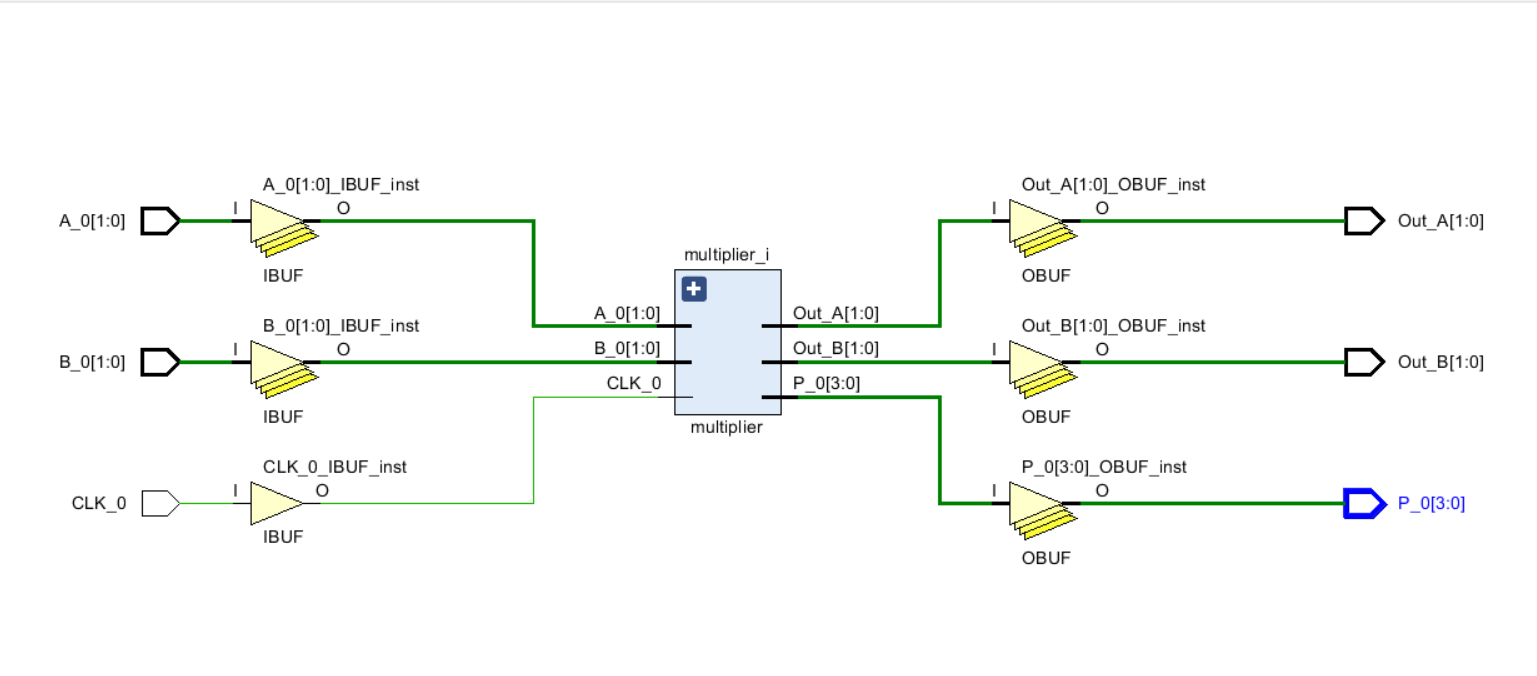
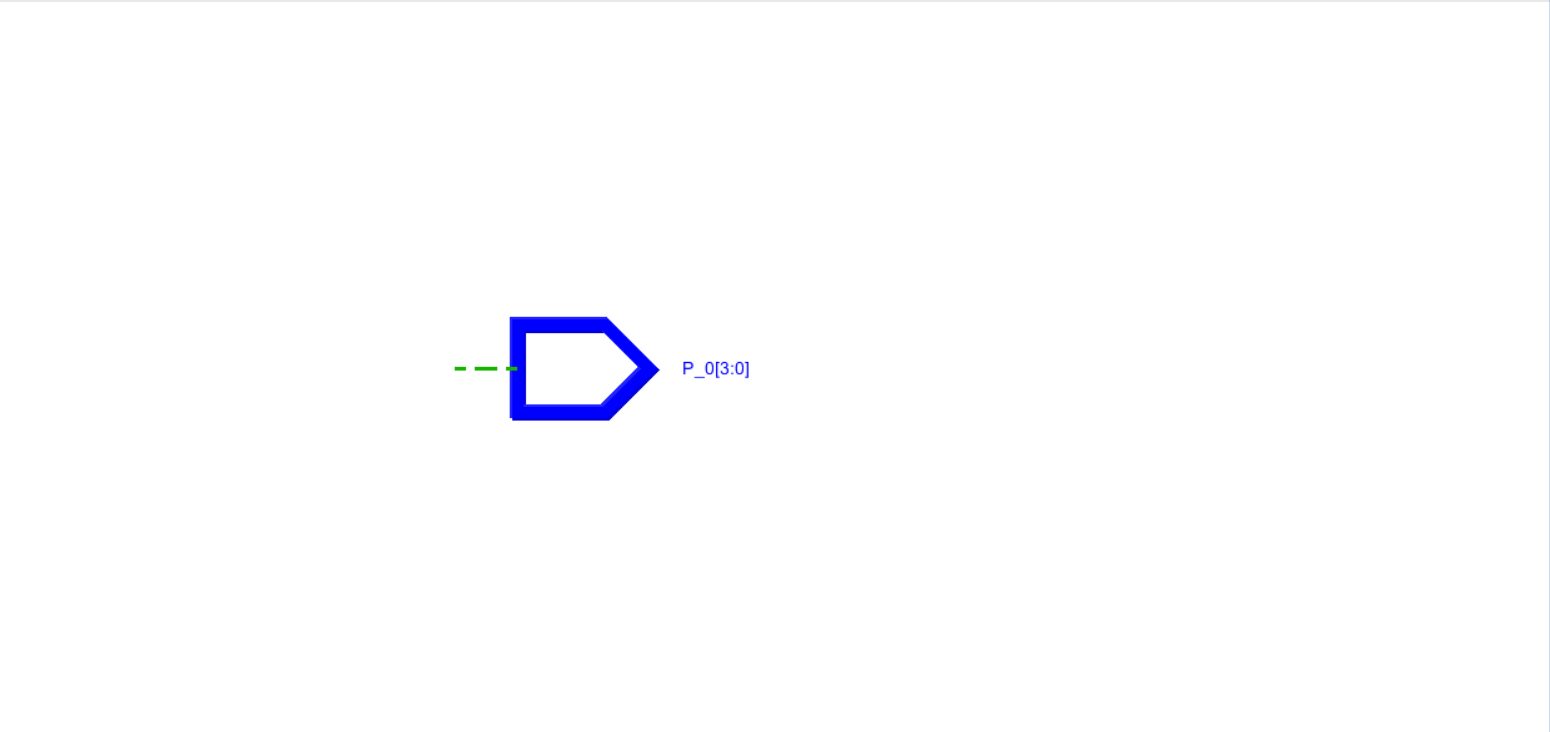
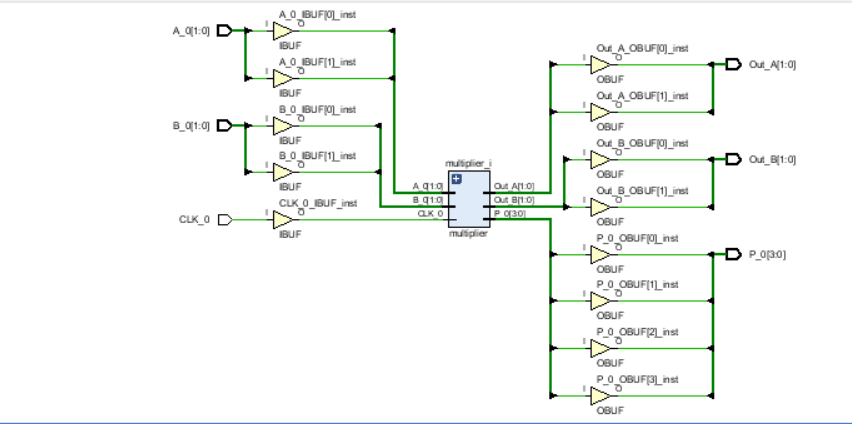
Answer: 

**Question: Paste the image showing the simulated waveforms.**

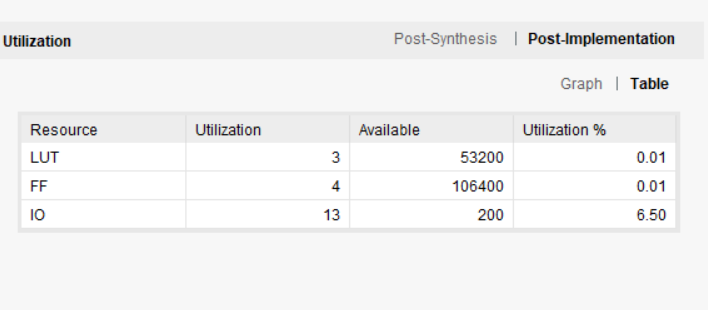
Answer: 

1. Plan your I/O mapping (using **I/O planning** option) such that actual input **Clk** is connected to internal clock pin **Y9,** all inputs connected to DIP switches and outputs are connected to LEDs (Refer to old documents or CMS for Pin information). Save the mapping information as **design\_1.xdc**.
2. Synthesize (**Run Synthesis**) and Implement the design (**Run Implementation)**.

**Question: Paste the image showing the schematic after synthesis.**

Answer: After Elaborated Design Schematic: After Synthesis: After Implementation: 

**Question: Check the summary report and report hardware utilization for the implementation.** (Note the utilization in your observation book)

Answer: 6.52%

1. **Generate Bitstream** and port your design on to FPGA (**Open Hardware Manager**🡪 **New Target**🡪… **Program Device**)
2. **Check the output on FPGA.**

**List the concepts you learnt from this lab (Conclusions/Observations)**

Answer: I learnt the implementation of a basic 2\*2 multiplier using Xilinx IP, where it works exactly like the generation of a conventional Verilog code however in the IP there are inbuilt blocks which serve the same purpose. Then there is a wrapper which converts this block into a Verilog code automatically. We should provide a testbench to check whether the block is working as intended. Then in the I/O Planning we need to map the input and output ports to DIPs(the LED lights) so that we can view our results. Rest of the procedure is similar to the earlier labs wrt to running synthesis,implementation,generating bitstream.