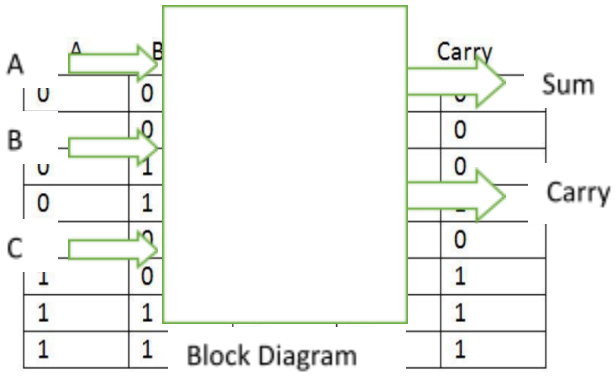
Lab 2

Tasks to be done in this lab:

1. Design and implement a 3-bit adder using Full Adder and verify its functionality.
2. Implement a 3x3 Binary Multiplier using the Full Adder and verify its functionality.
3. Test the functionality of both on the Basys3 Board (remote access) using VIO IP

1. 3-bit Full Adder Implementation:

It is a combinational circuit used for three bits addition. The block diagram and truth table are shown below:



Truth Table

The expressions for Sum and Carry Output are as follows:
Sum= A^B^C
Carry=(A^B).C + AB

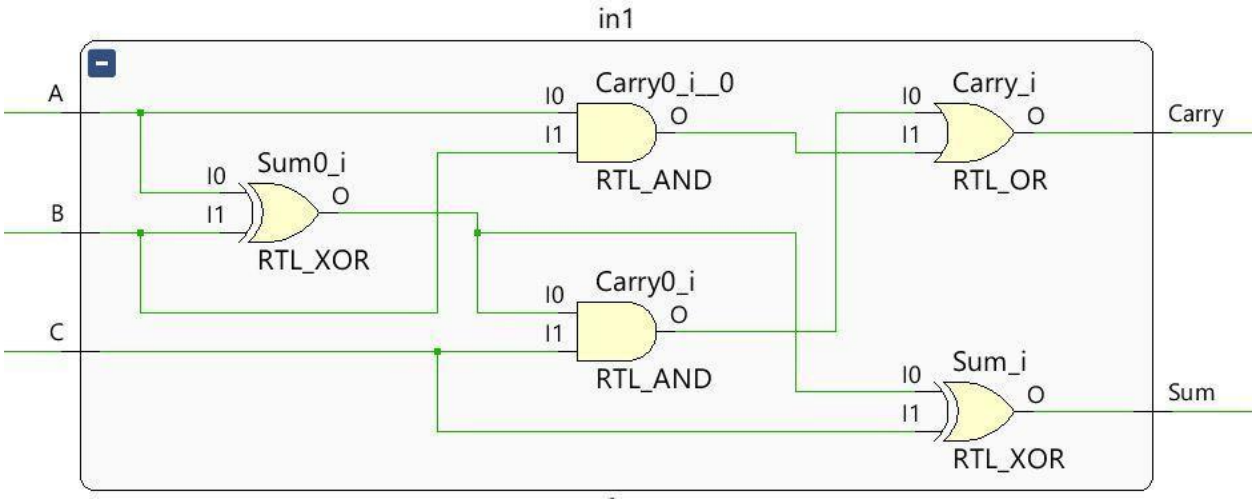


Fig: 1 Gate Level Circuit Diagram of Full Adder

To implement the 3-bit Full Adder, three full adders are connected in cascade as shown in the circuit diagram in Fig:2

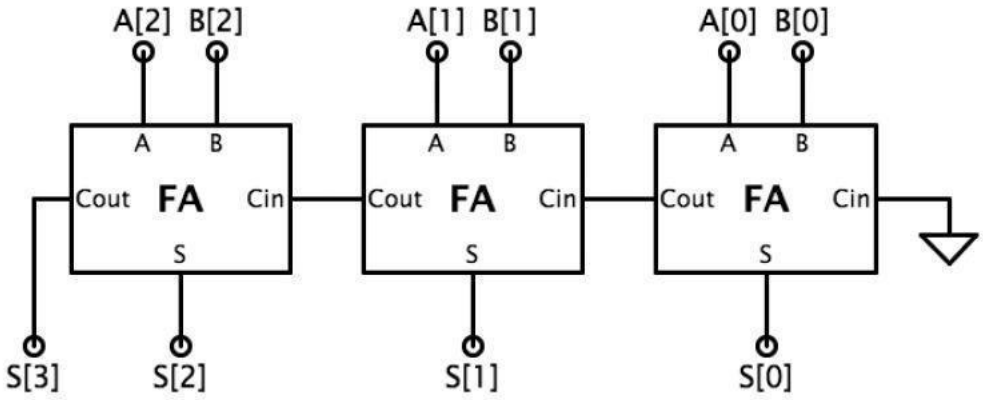
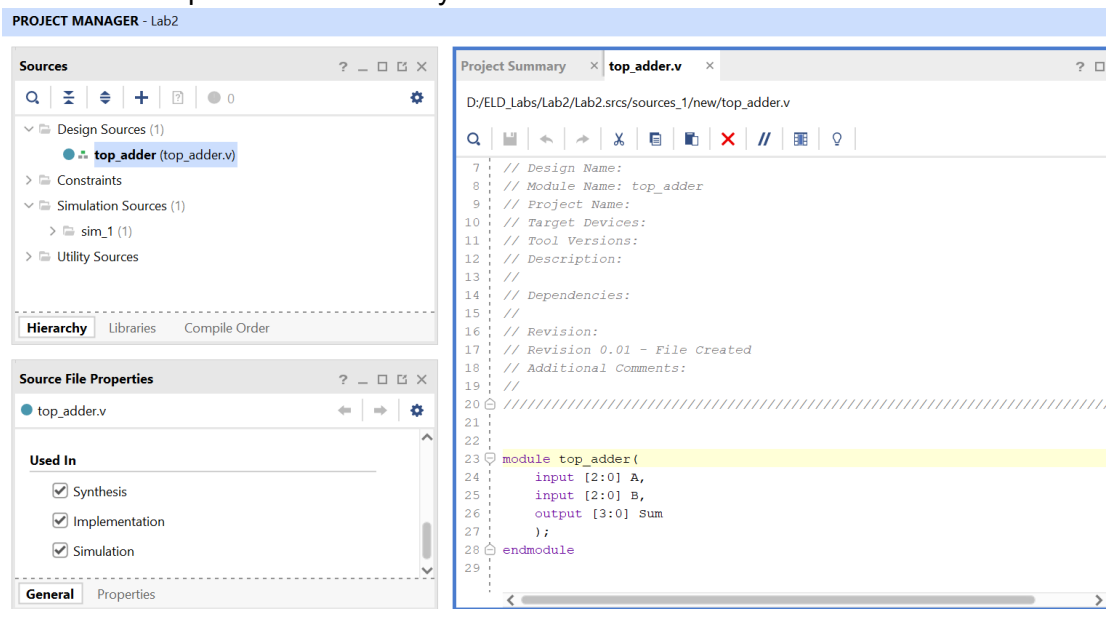


Fig 2: Circuit diagram for 3-bit adder using Full Adders

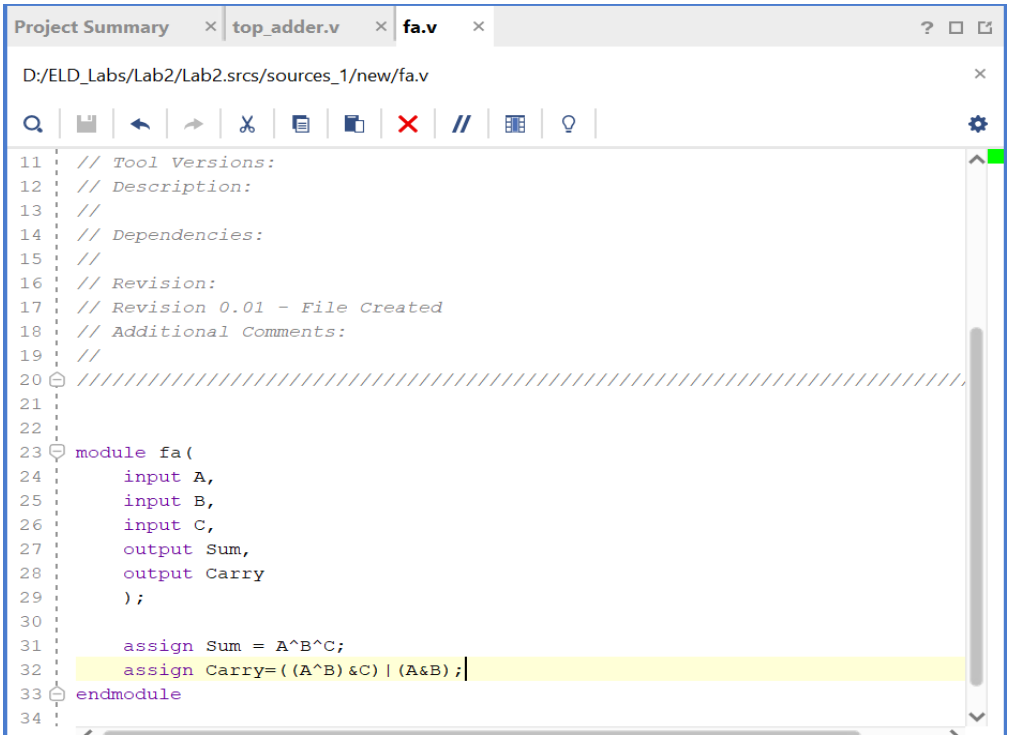
Steps to implement the 3-bit Adder on Vivado:

1. Create a New Project.
2. Select Project Type as RTL project.
3. In the Add Sources window, first we will create the Verilog file for the top module(top_adder.v).
4. You don't need to add constraints at this time; we will do it in the later stages.
5. Add the board as Basys3 board.

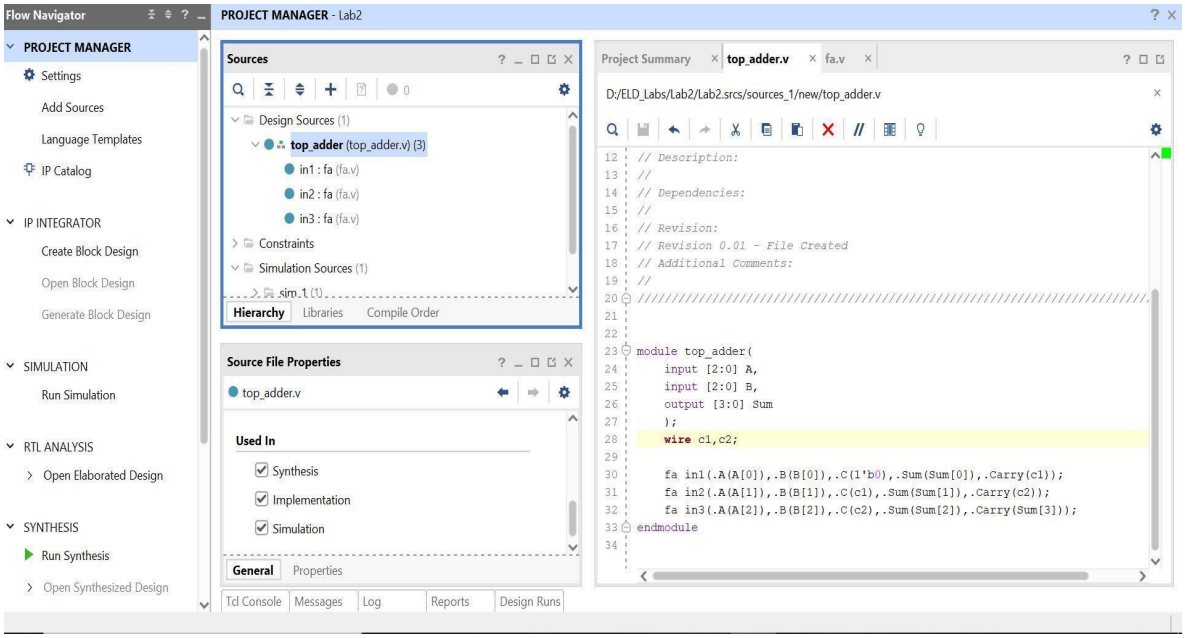
6. Make the port declaration, two input ports(A, B) of 3 bits each, and one output port Sum of 4-bit width.
7. After completing Step 6, your project must have a module top_adder, which should have the module and port declaration only as shown below



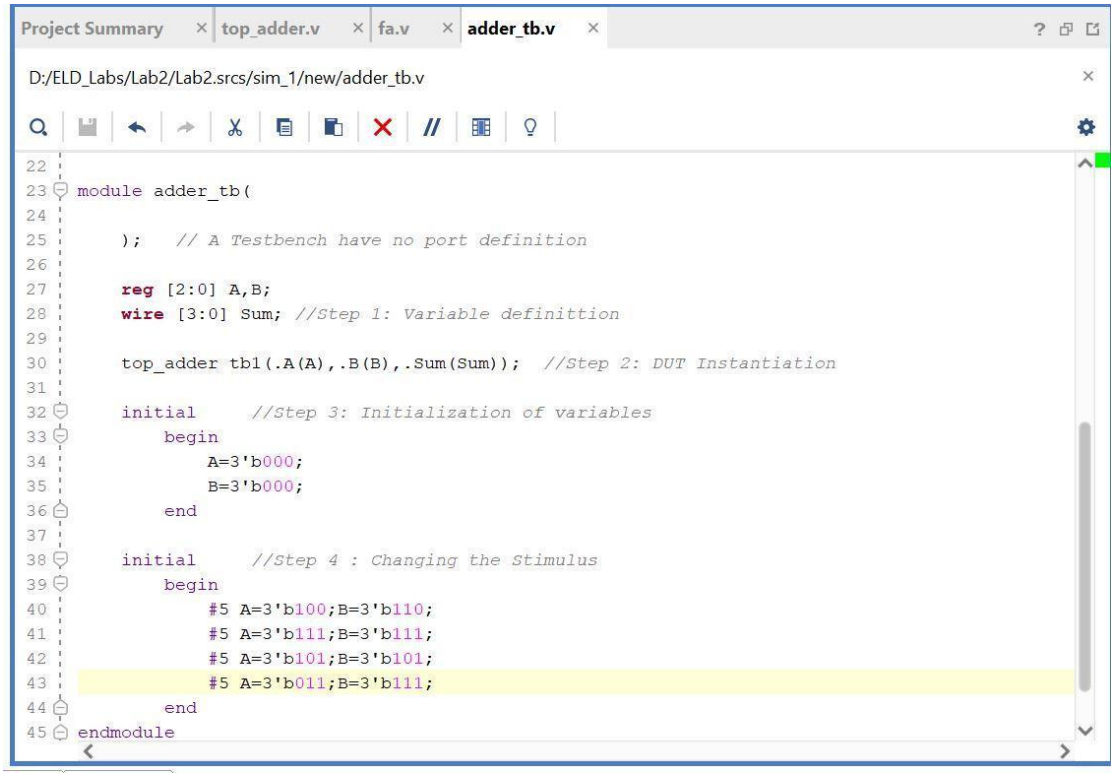
8. Now we will add the functionality of the full adder. To do so, add design sources and create one more Verilog file fa.v. Add the three input ports (A, B, C) and two output ports (Sum, Carry). Implement the functionality of the full adder. The fa.v should look like the one shown in the screenshot below:



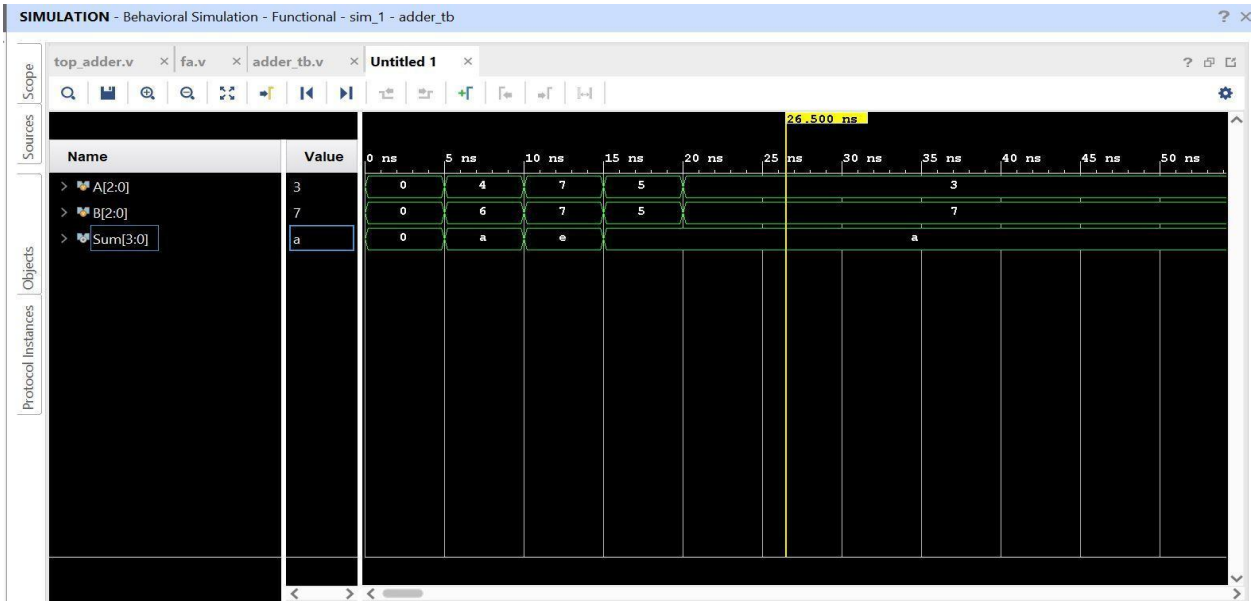
9. Now we will instantiate the full adder module in the top module in the and make the connections in the same manner as shown in Fig:2. After instantiations your top module should look like as shown in the below screenshot:



10. The next step is to verify the functionality using testbench. For that, add a simulation source and write the testbench, as discussed in Lab1. Your testbench file should look like the one shown below

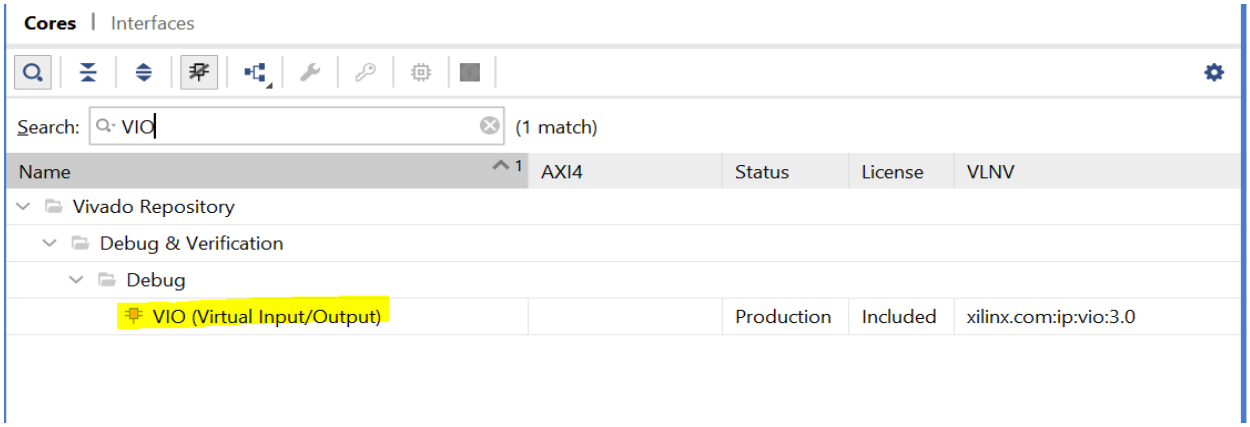


11. Run Simulation and verify the functionality.

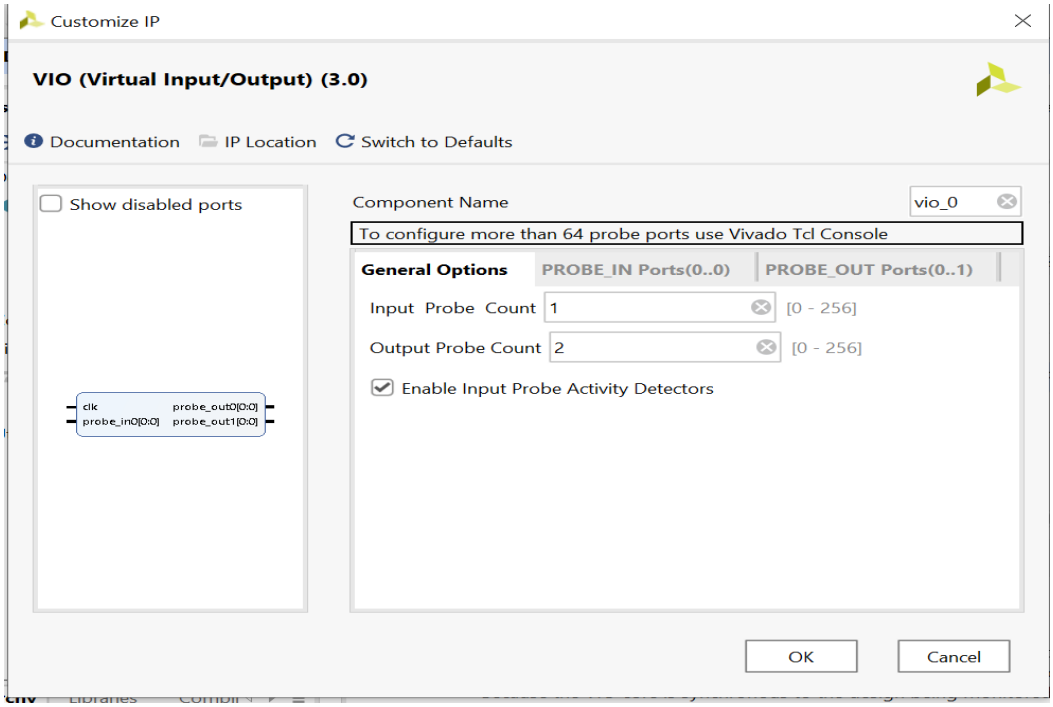


The next part of this lab is to test the functionality of the design on the FPGA. As we are going to access the board using a remote connection, we will make use of VIO IP to give the stimulus and monitor the output. To add the IP use the following steps:

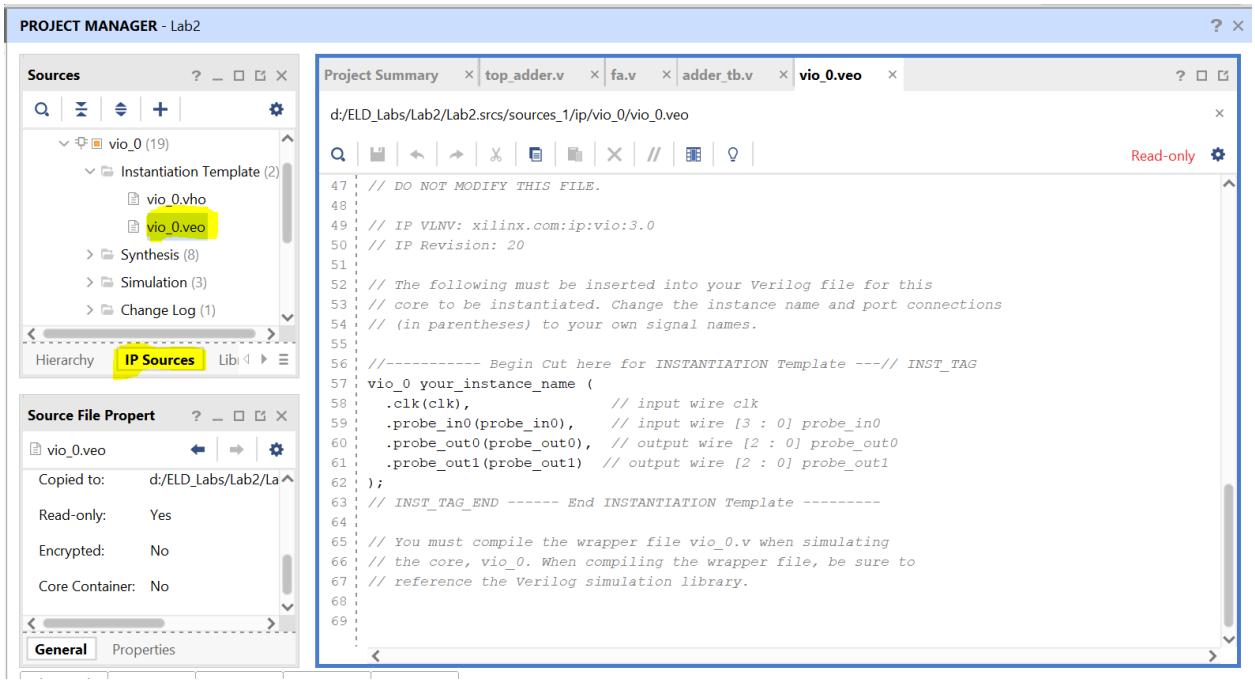
1. In the right pane, click on IP Catalog.
2. In the search box type VIO and click on the VIO IP as shown below



3. Once you double click on the IP name, a window will appear as shown below, which will contain the Input and Output Probes field. Now, if the FPGA board would have been available physically, we could have used switches to give input and LEDs to check the output. Because we are accessing the board remotely, we need some mechanism to give stimulus and monitor output. For that, VIO is very useful. Using the output probes of VIO, we can give stimulus as desired, and using the input probes of VIO, we can monitor the output. As we have two inputs, we need two output probes of 3 bits each and one Input probe of 4 bits to monitor the Sum output. You can change the probe width by going into the PROBE_IN and PROBE_OUT window.



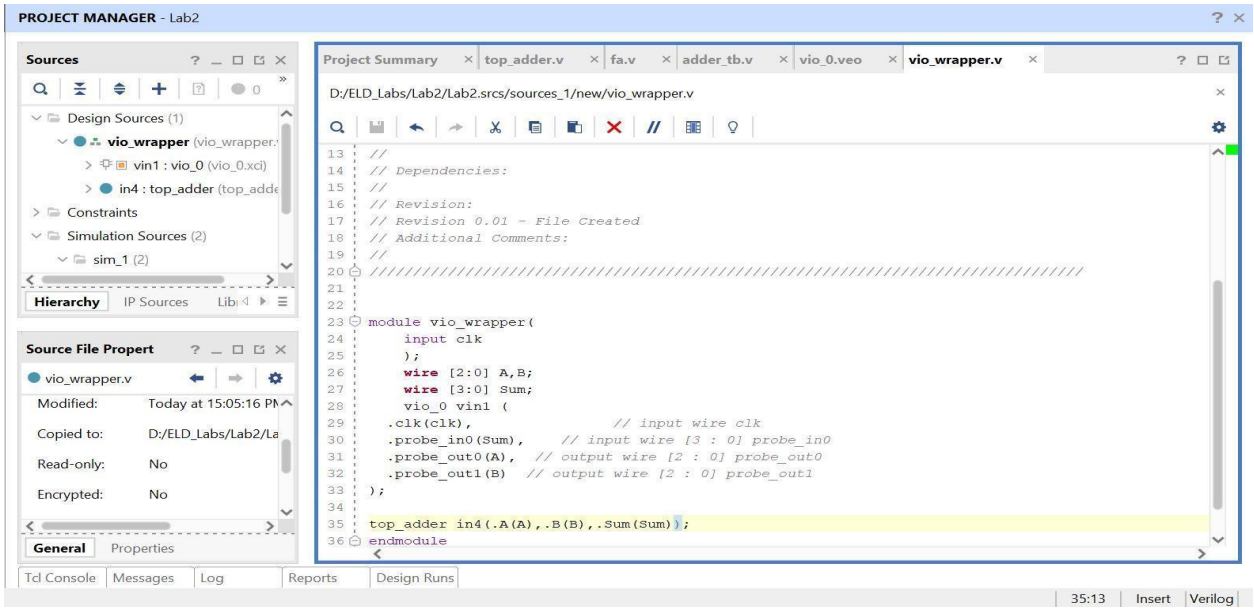
4. Once you are done, click OK.
5. Now go to IP Sources->Instantiation template and open .veo file as shown below:



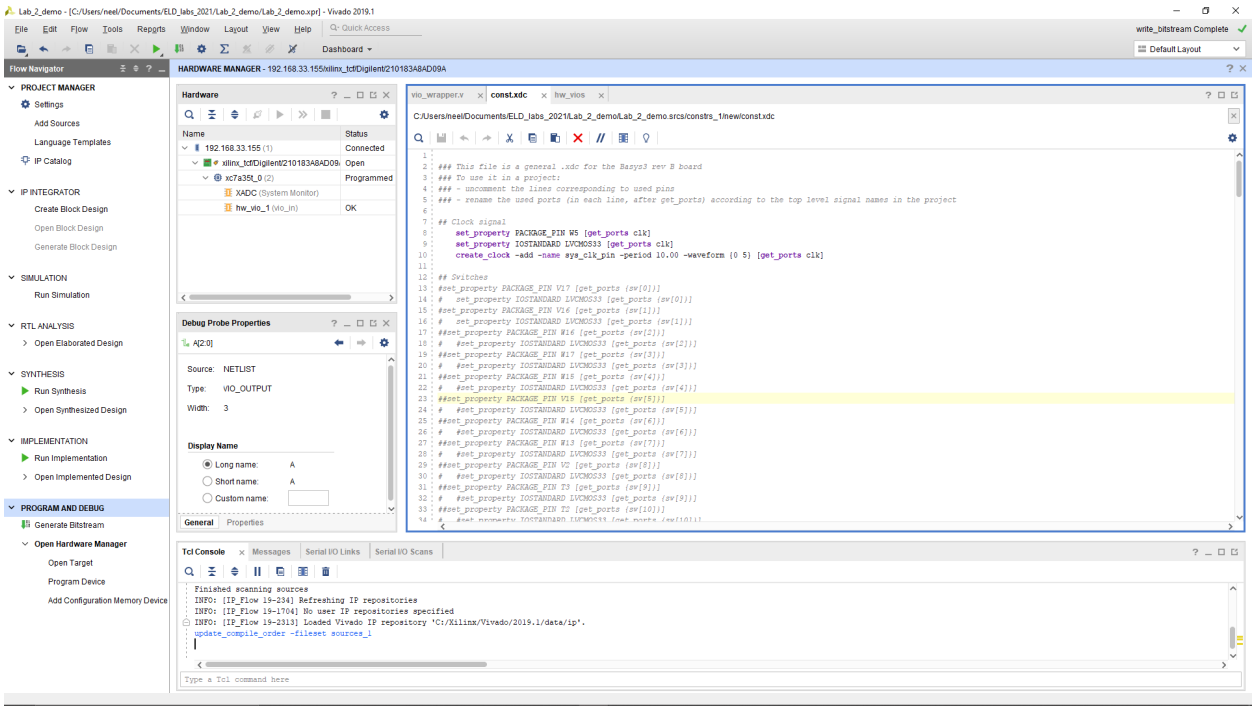
6. Copy the instantiation template.

Next, we need to make a few changes to our design. We need to connect out top module to the VIO. For that, follow the steps as given below:

1. Add a new design source that has one input port for Clock as VIO IP will need the Clock. We will call this module as VIO Wrapper module.
2. Connect the VIO and top module of the design in the wrapper module, as shown below.

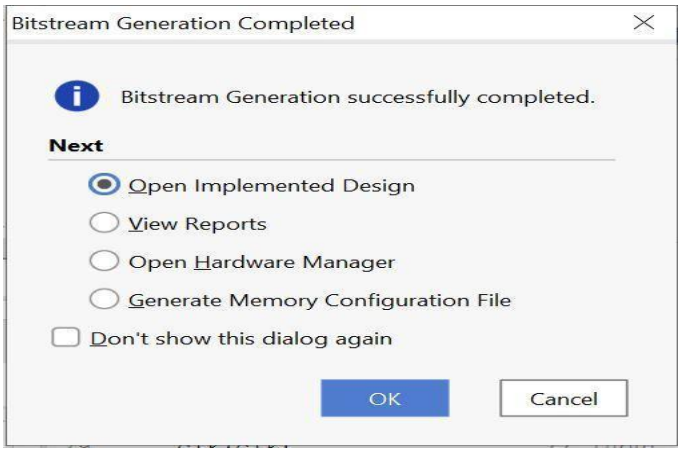


3. The next step is to add a constraint file. Click on add sources -> Add or create constraints->create file->write file name and click OK.
4. Now go to the constraints file we made in step 3 and copy the master xdc of Basys3(can be easily found on the internet) and uncomment the lines as shown in the screenshot below:

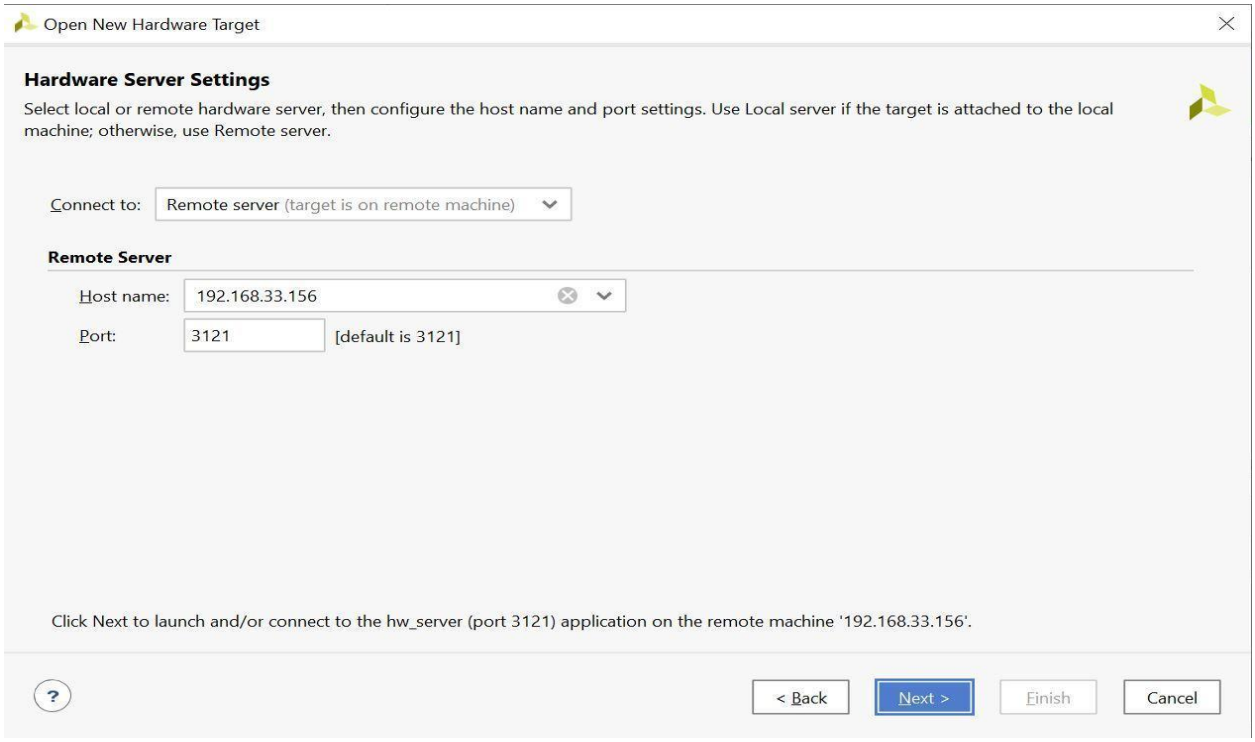


In Step 4 we have connected the CLK pin of the board to the clk pin of vio wrapper module. **Please note that we have changed the "CLK100MHZ" to clk as highlighted.**

5. The next step is to generate the bitstream. Click on "Generate Bitstream" on the right pane. After the successful generation of the bitstream, you will get the following window.

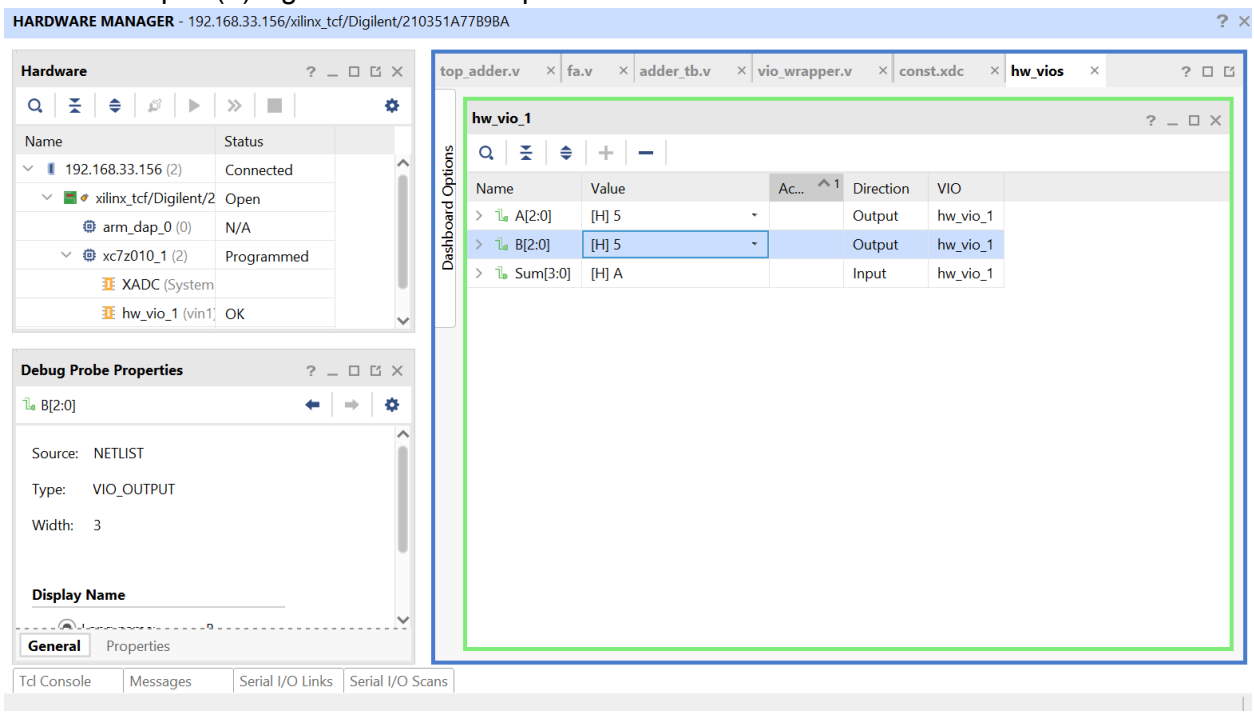


6. Click on Open hardware Manager and then click on Open Target followed by clicking on Open New Target.
7. Click Next and in the "Connect to" dropdown select Remote Server and fill the details as shown in the following screenshot and click next. It will take some time to connect to the board.



8. Once connected, program the device using the bitstream we created in Step 5.
9. Once programmed, the window will look like as below.If not opened automatically, open the hw_vios.

10. Click on the plus(+) sign and add all the ports.



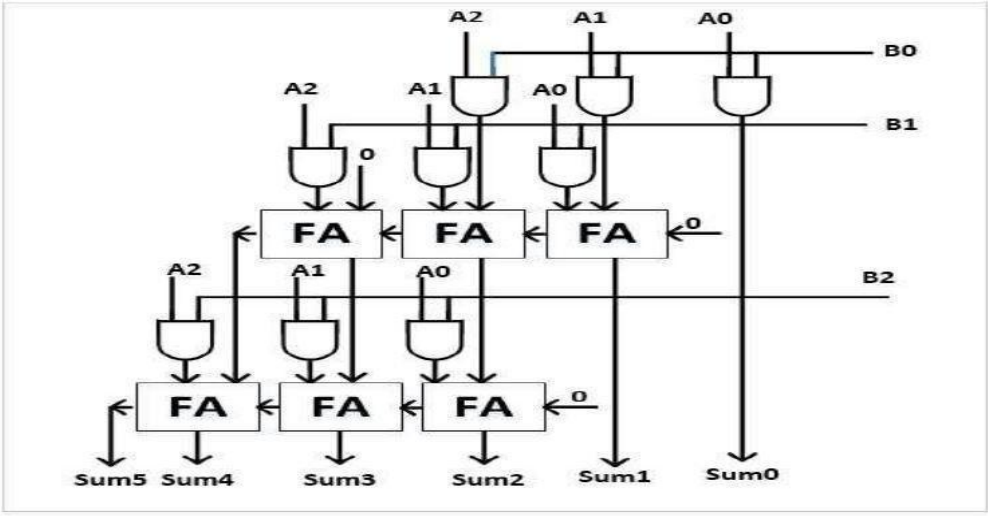
11. In the value dropdown, you can change the values of the inputs, and correspondingly, the output will appear automatically. Change the inputs and check whether the design is functioning correctly or not.

2. 3X3 Multiplier:

Multiplication is a very common operation in digital systems. For that reason, custom-designed multipliers are often used. The goal of this lab is to make you familiar with the design and implementation of such a multiplier. Multiplication of a 3-bit number B(2:0) by a 3-bit number A(2:0) can be illustrated as follows.

			A2	A1	A0	
		X	B2	B1	B0	
			A2B0	A1B0	A0B0	
		A2B1	A1B1	A0B1		
+	A2B2	A1B2	A0B2			
Sum5	Sum4	Sum3	Sum2	Sum1	Sum0	

Implementation using Full Adder: The circuit diagram for the implementation of 3X3 Multiplier is given below:

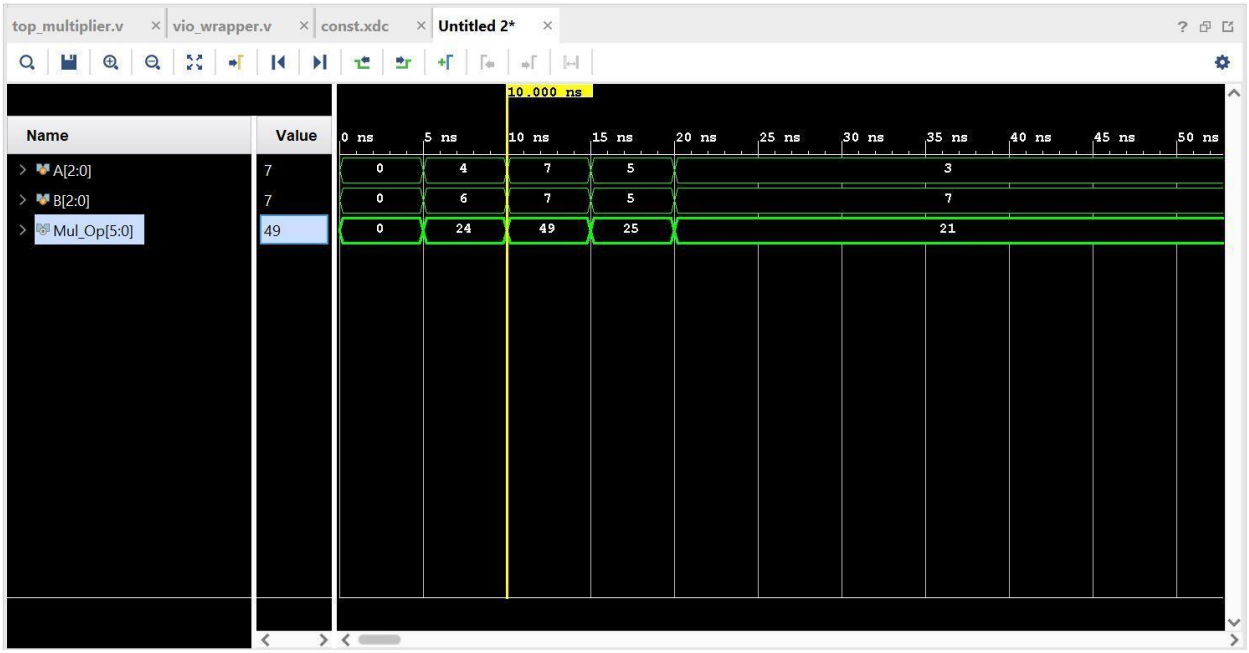


- The full adder module will remain the same.
- The top module will be changed according to the circuit given above. After making the changes, the top module will look like the one shown below:

```
module top_multiplier(
    input [2:0] A,
    input [2:0] B,
    output [5:0] Mul_Op
    );

    wire c1,c2,c3,c22,c32;
    wire s1,s2;
    assign Mul_Op[0]=A[0] & B[0];
    fa in1(.A(A[0] & B[1]),.B(A[1] & B[0]),.C(1'b0),.Sum(Mul_Op[1]),.Carry(c1));
    fa in2(.A(A[2] & B[0]),.B(A[1] & B[1]),.C(c1),.Sum(s1),.Carry(c2));
    fa in3(.A(A[0] & B[2]),.B(s1),.C(1'b0),.Sum(Mul_Op[2]),.Carry(c22));
    fa in4(.A(A[2] & B[1]),.B(1'b0),.C(c2),.Sum(s2),.Carry(c3));
    fa in5(.A(A[1] & B[2]),.B(s2),.C(c22),.Sum(Mul_Op[3]),.Carry(c32));
    fa in6(.A(A[2] & B[2]),.B(c3),.C(c32),.Sum(Mul_Op[4]),.Carry(Mul_Op[5]));
endmodule
```

- Minor changes need to be done in testbench as the output width is 6 bits now.

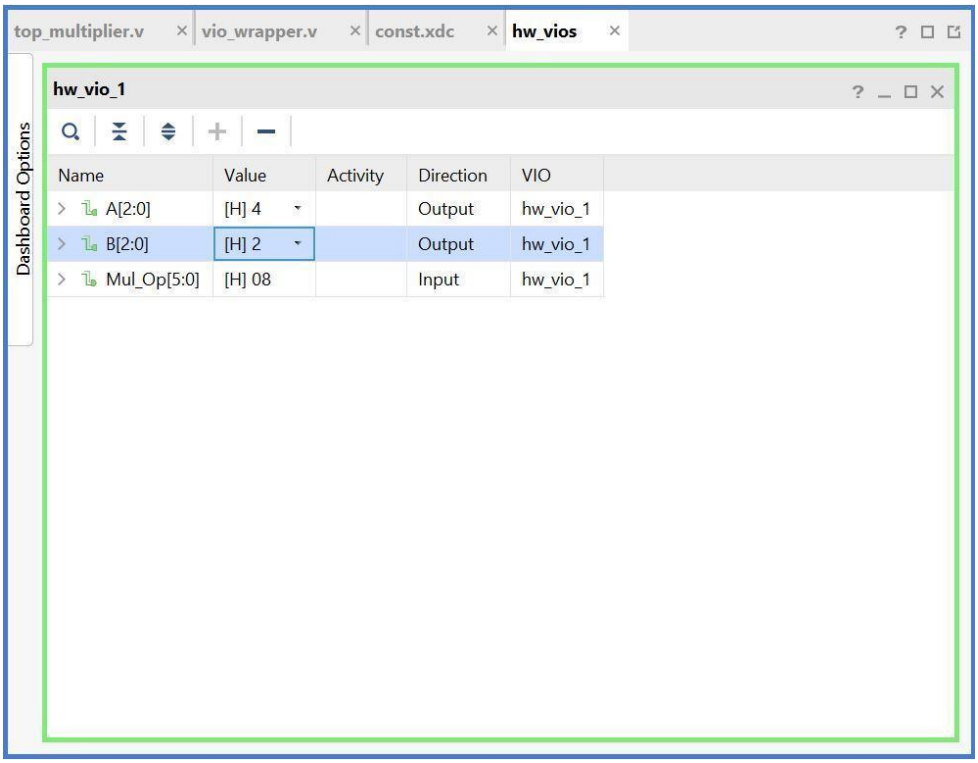
```
module mul_tb(

    );

    reg [2:0] A,B;
    wire [5:0] Mul_Op; //Step 1: Variable definittion

    top_multiplier tb1(.A(A),.B(B),.Mul_Op(Mul_Op));  //Step 2: DUT Instantiation

    initial     //Step 3: Initialization of variables
        begin
            A=3'b000;
            B=3'b000;
        end

    initial     //Step 4 : Changing the Stimulus
        begin
            #5 A=3'b100;B=3'b110;
            #5 A=3'b111;B=3'b111;
            #5 A=3'b101;B=3'b101;
            #5 A=3'b011;B=3'b111;
        end
endmodule
```



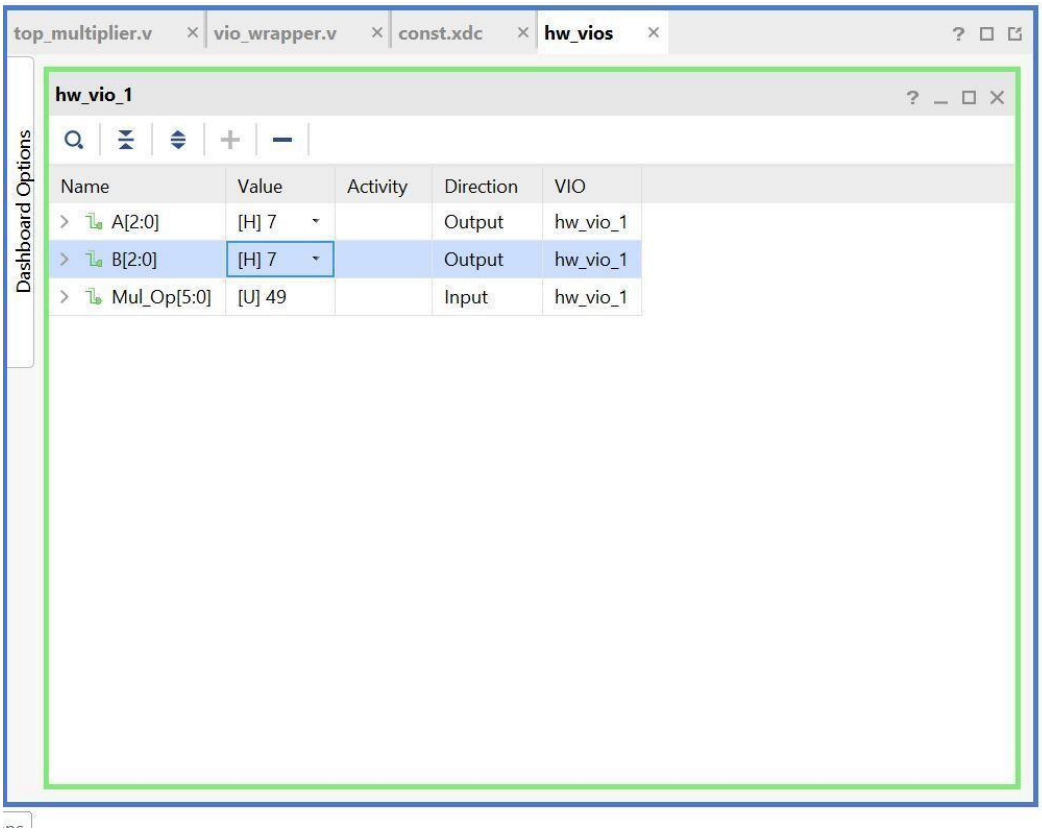
- After making the changes in the vio wrapper, it should look like the one shown below:

```
module vio_wrapper(
    input clk
    );
    wire [2:0] A,B;
    wire [5:0] Mul_Op;

    vio_0 vin1 (
  .clk(clk),                    // input wire clk
  .probe_in0(Mul_Op),     // input wire [5 : 0] probe_in0
  .probe_out0(A),  // output wire [2 : 0] probe_out0
  .probe_out1(B)   // output wire [2 : 0] probe_out1
);

top_multiplier in7(.A(A),.B(B),.Mul_Op(Mul_Op));
endmodule
```

- The constraints file will remain the same.
- Generate bitstream, connect to remote hardware, and program the device as done in the adder.
- Check the functionality by changing the stimulus of the VIO.



- You can also change the radix by right-clicking on the Value. As shown in the screenshot given below, we have changed the radix of Mul_Op to an unsigned decimal.


top_multiplier.v
vio_wrapper.v
const.xdc
hw_vios
Dashboard Options
hw_vio_1
Name
Value
Activity
Direction
VIO
A[2:0]
[H] 7
Output
hw_vio_1
B[2:0]
[H] 7
Output
hw_vio_1
Mul_Op[5:0]
[U] 49
Input
hw_vio_1