Vivado Tutorial and Lab

ECE 366- Computer Organization

February 10, 2025

1. Tasks to be done in this lab.

1. Design and implement a half-adder (using a data flow approach).
2. Using the half-adders designed in step 1 to implement a full adder.
3. Implement a 4-bit adder/subtractor (adds/subtracts two 4-bit inputs) using the full adders designed in step 2.

2. Download

a. Windows: AMD Unified Installer Windows
   i. Double click on the downloaded exe file and follow on-screen installation instructions.

b. Linux: AMD Unified Installer Linux
   i. cd <folder_in_which_you_downloaded_the_bin_file>
   ii. chmod +x <bin_file>
   iii. ./<bin_file>
   iv. follow on-screen installation instructions.

c. Detailed User Guide and Installation Link from AMD: https://docs.amd.com/r/en-US/ug973-vivado-release-notes-install-license/Navigating-Content-by-Design-Process

3. Step-by-step guide for implementing half-adder.

Follow the steps shown in the figures one by one.

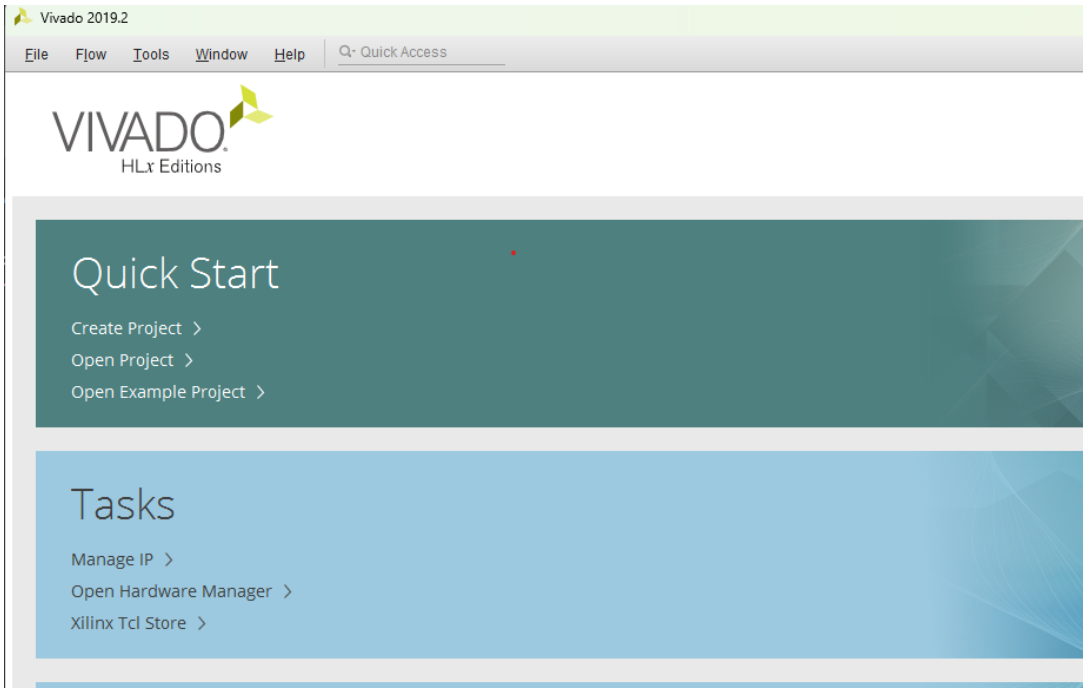

Figure 1: Create New Project

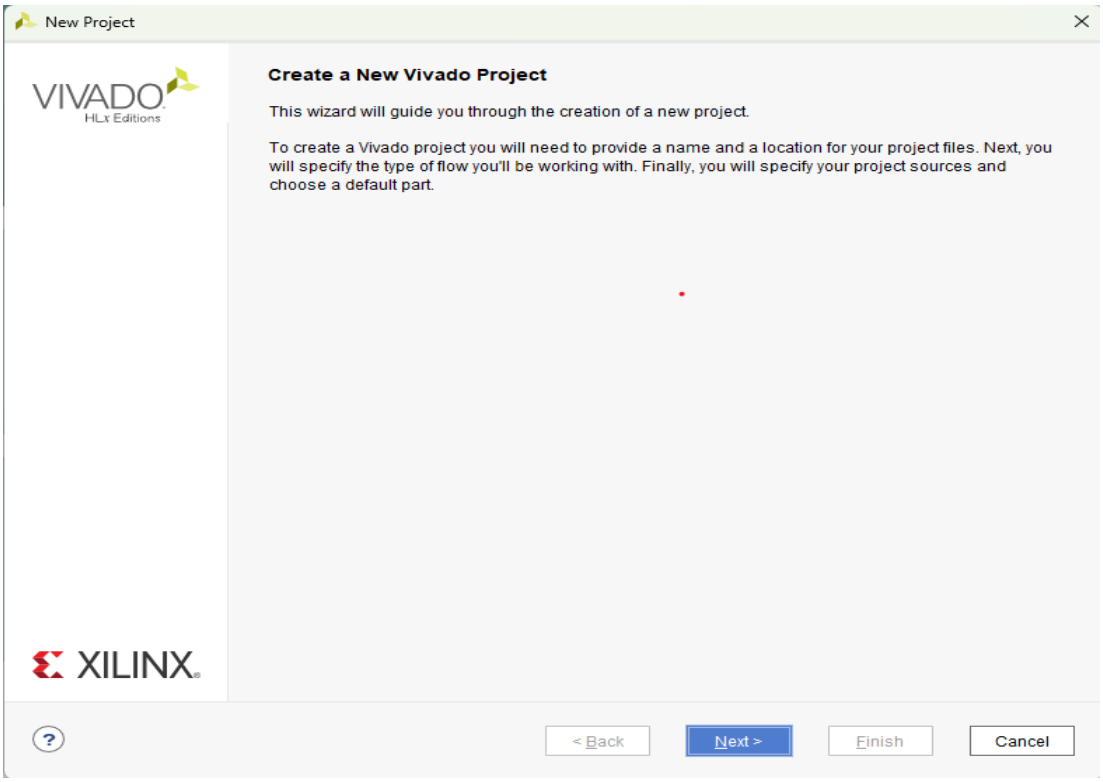

Figure 2: Click Next

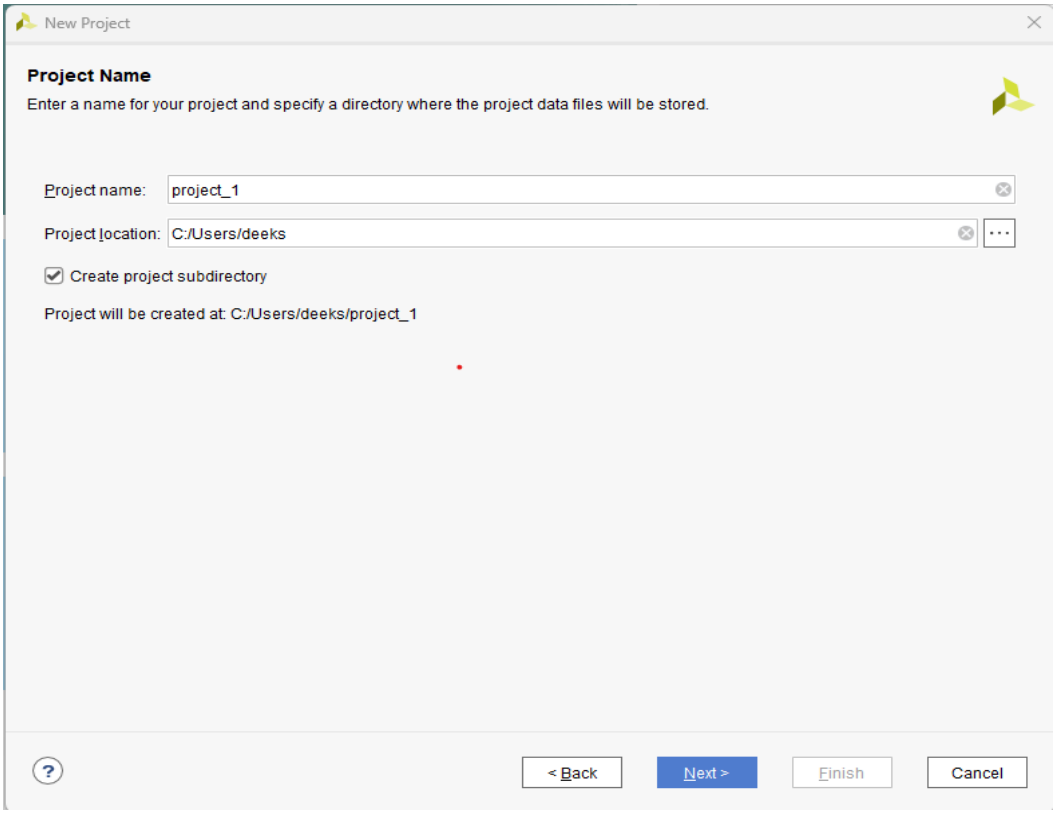

Figure 3: Give your project a name and set the location of your project

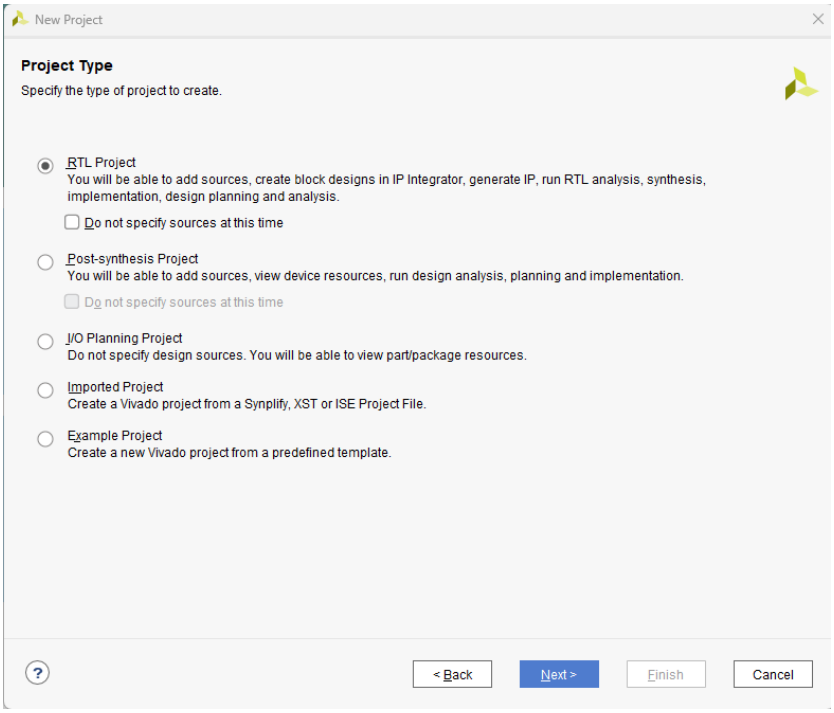

Figure 4: Select the type of project you wish to create (RTL Project in our case, uncheck the option "Do not specify.....")

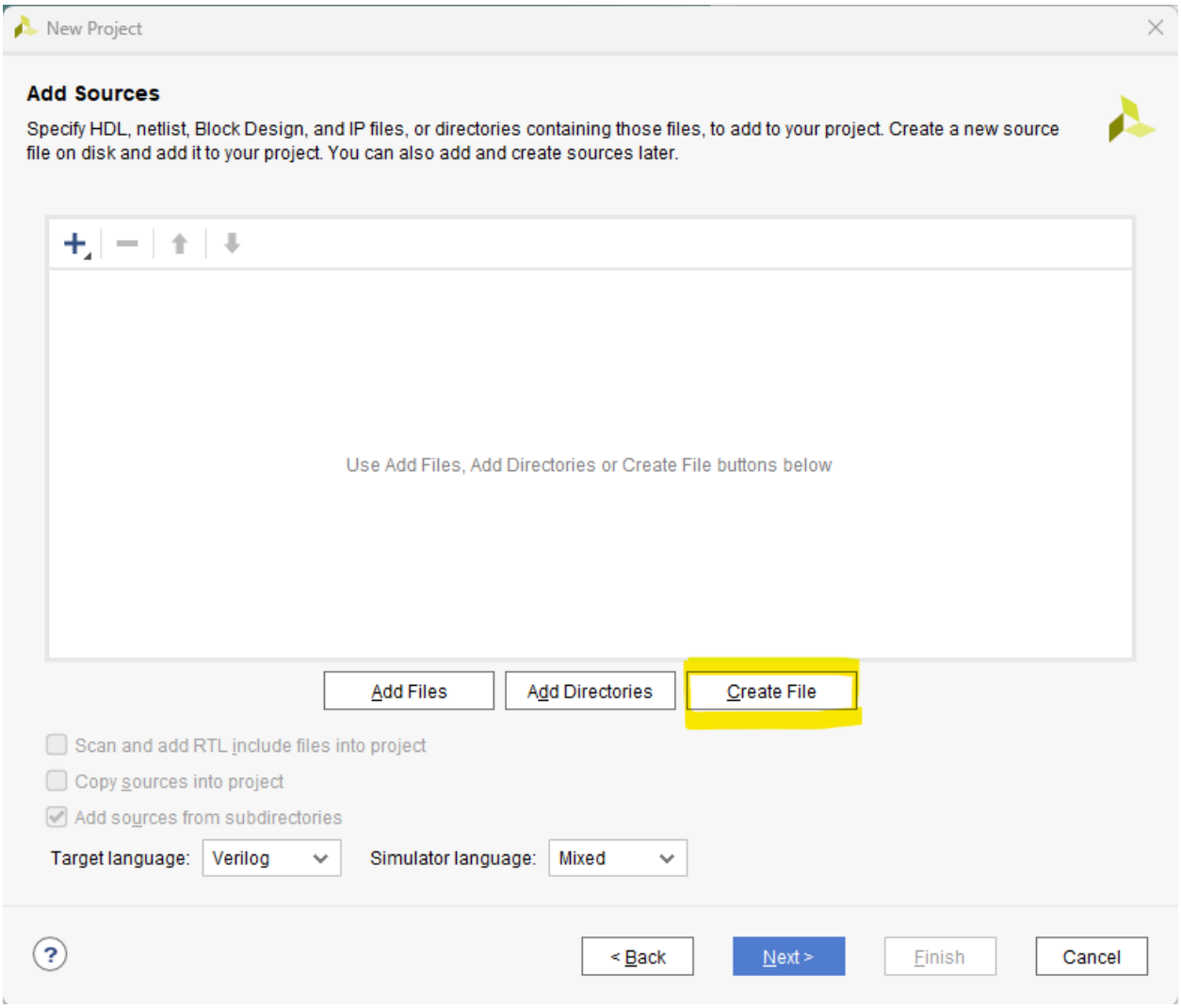

Figure 5: In Add Sources, click on Create File to create new files

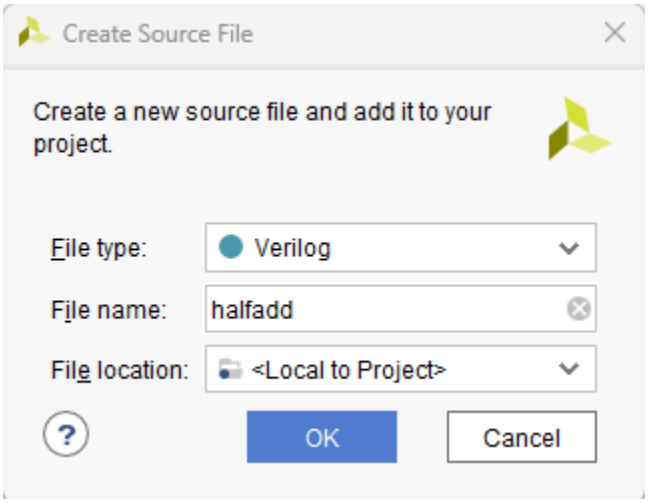

Figure 6: Give a suitable name to your verilog file and click OK.

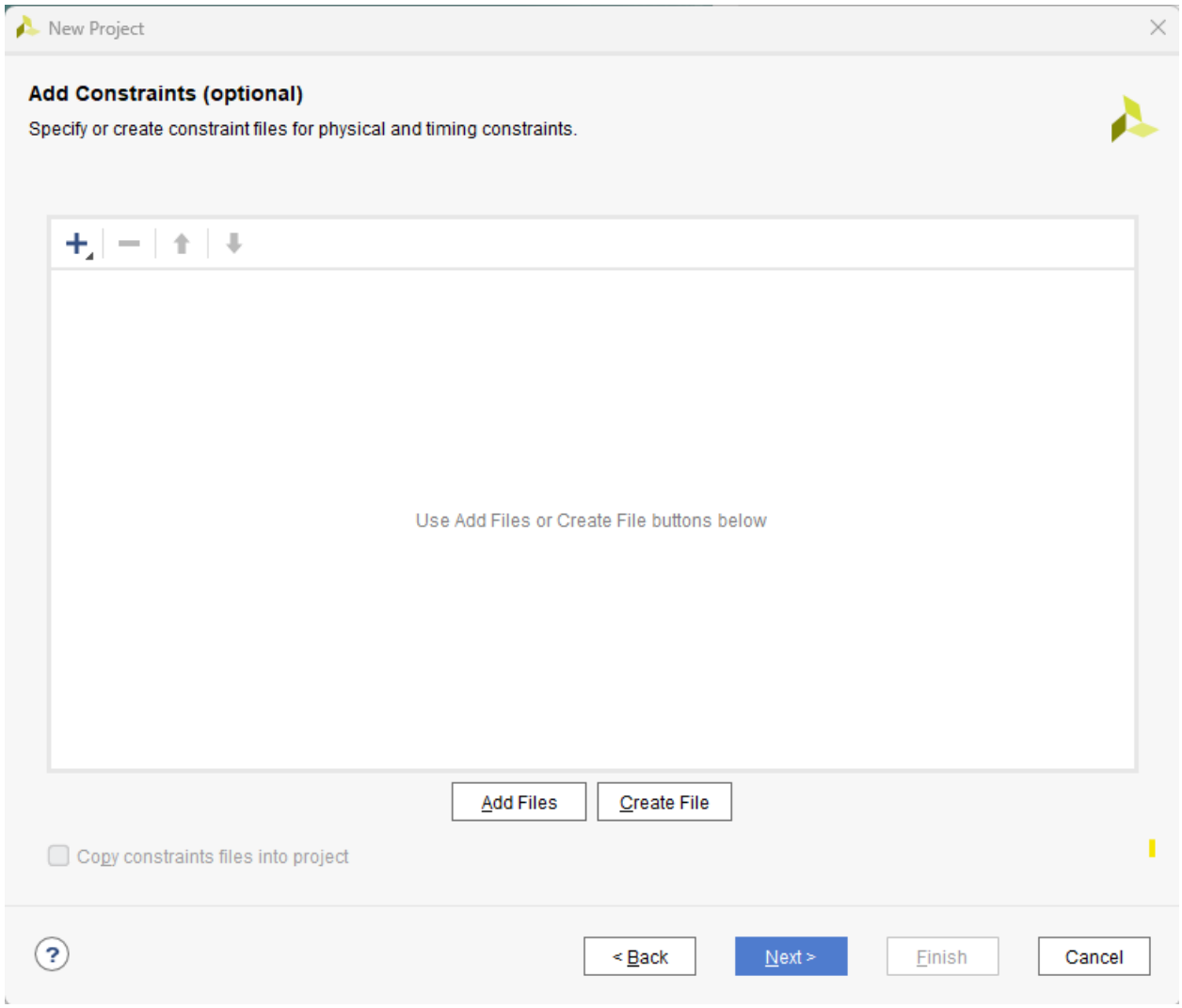

Figure 7: Add Constraints if required (click Next for our case)

These settings remain the same for any ECE 366 project.

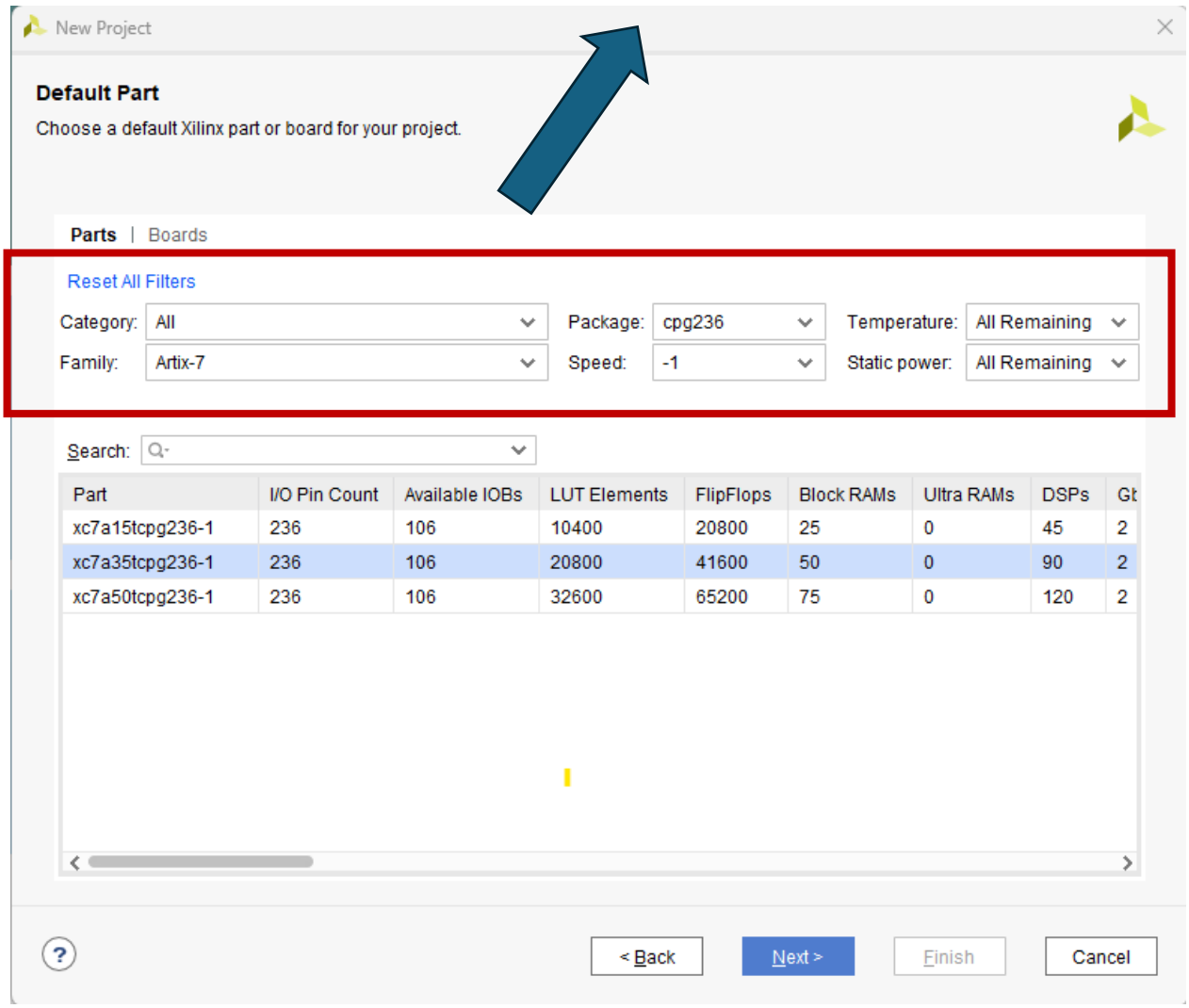

Figure 8: Select the target FPGA device

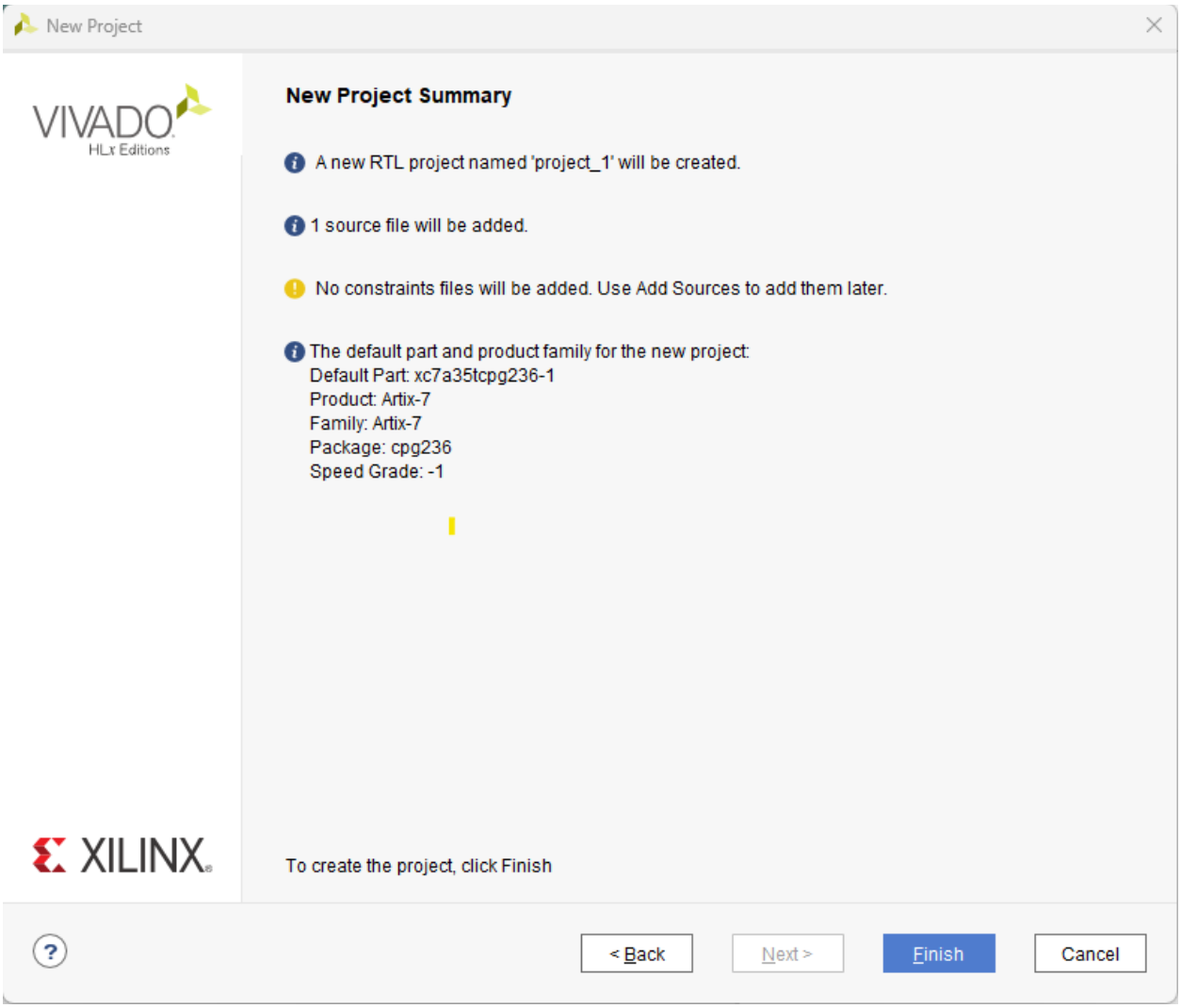

Figure 9: Check the project summary, click Finish.

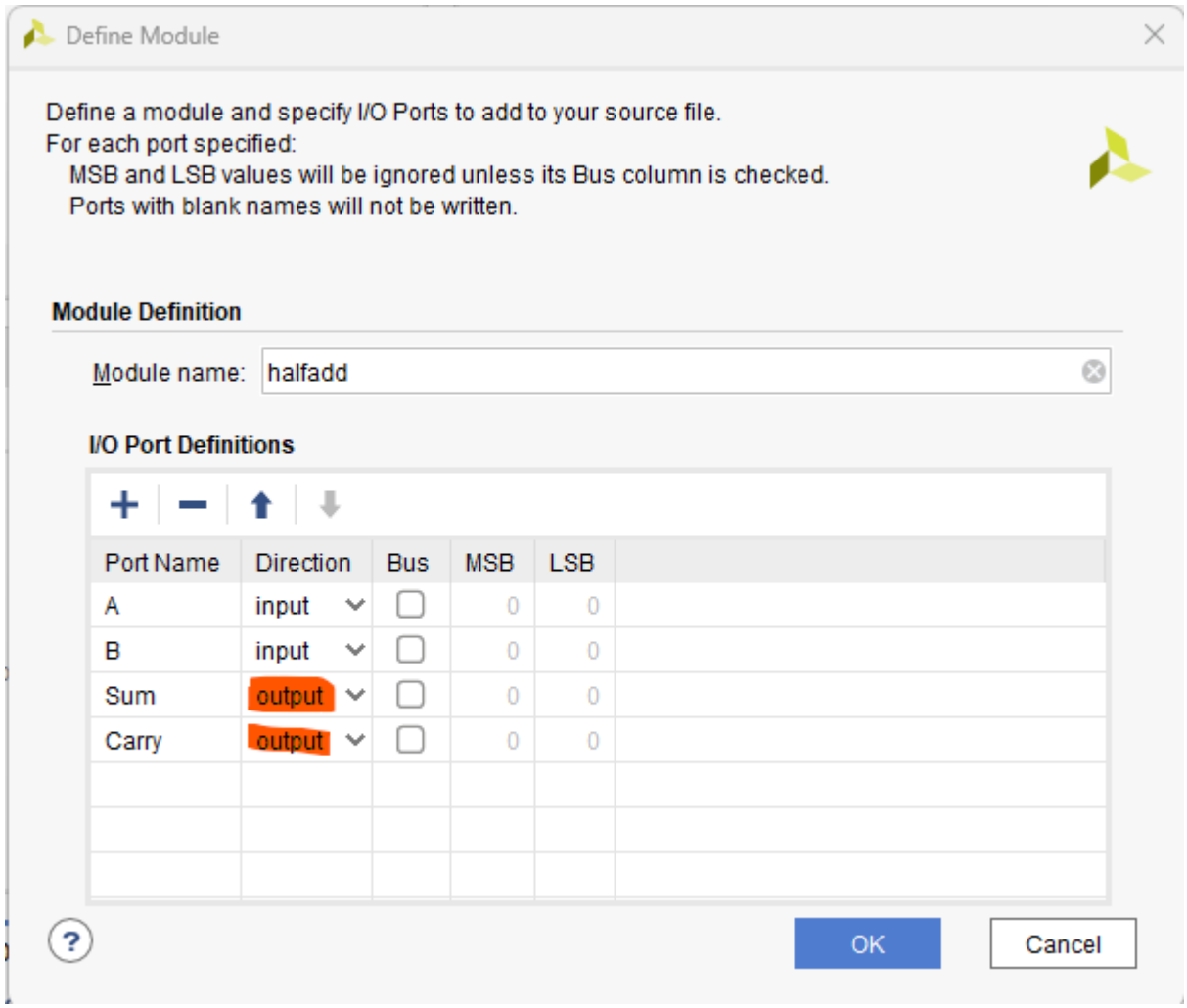

Figure 10: Mention the IOs of the verilog files created

Figure 11: You can see the files added in Sources window. Now open the files and code the required logic.

Project Summary | halfadd.v *

C:/Users/deeks/project_1/project_1.srcs/sources_1/new/halfadd.v

```
// Company:
// Engineer:
//
// Create Date: 02/07/2025 03:19:13 PM
// Design Name:
// Module Name: halfadd
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module halfadd(
    input A,
    input B,
    output Sum,
    output Carry
    );
    assign Sum = A ^ B;
    assign Carry = A & B;
endmodule
```

Figure 12: Half adder code added in the halfadd.v file and save the file.

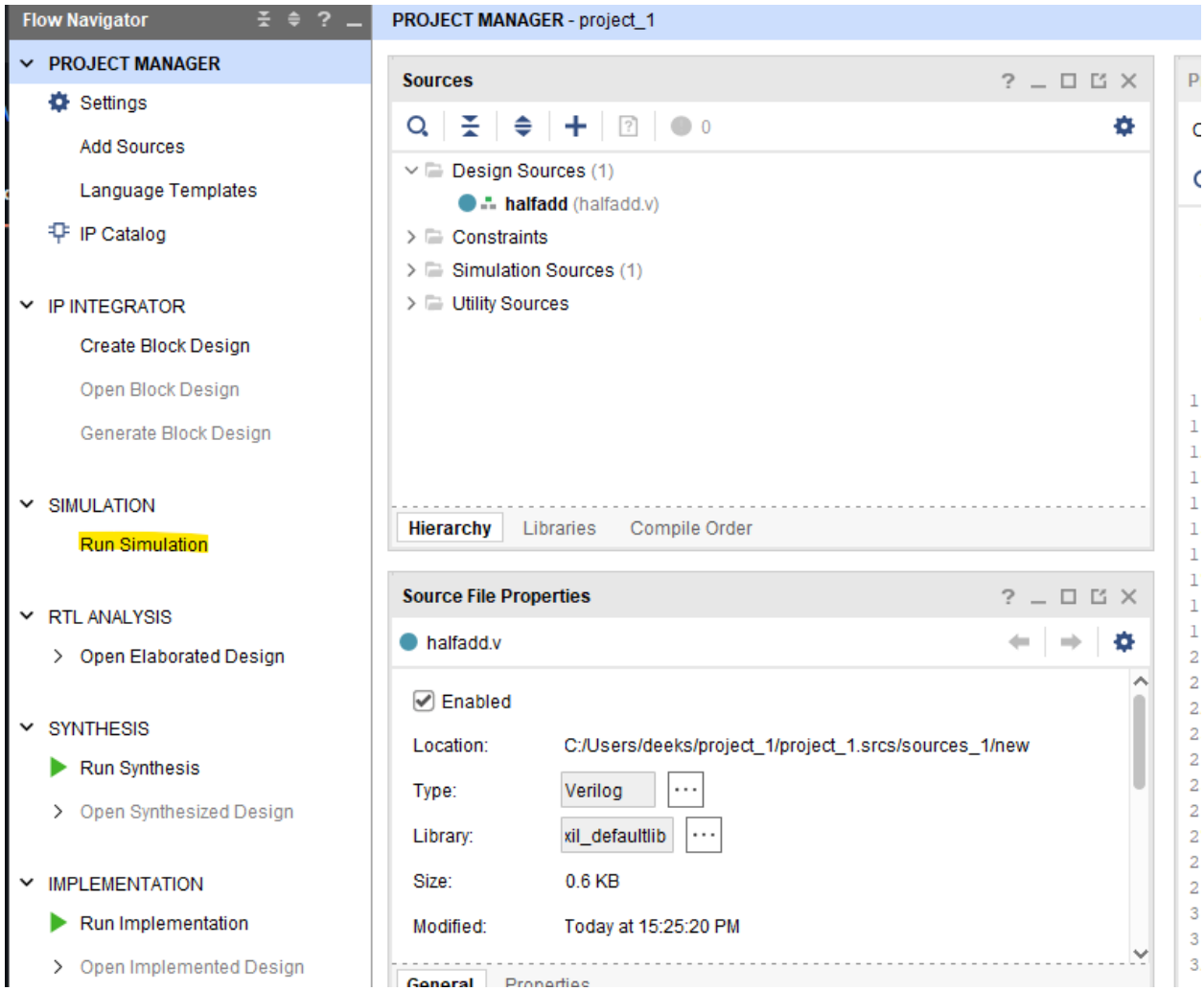

Figure 13: Click on Run Simulation and select Run Behavioral Simulation.

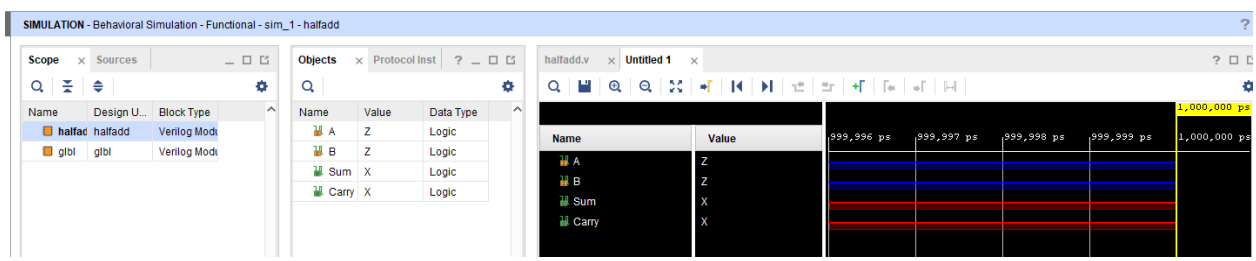

Figure 14: Behavioral Simulation will open with the following windows.

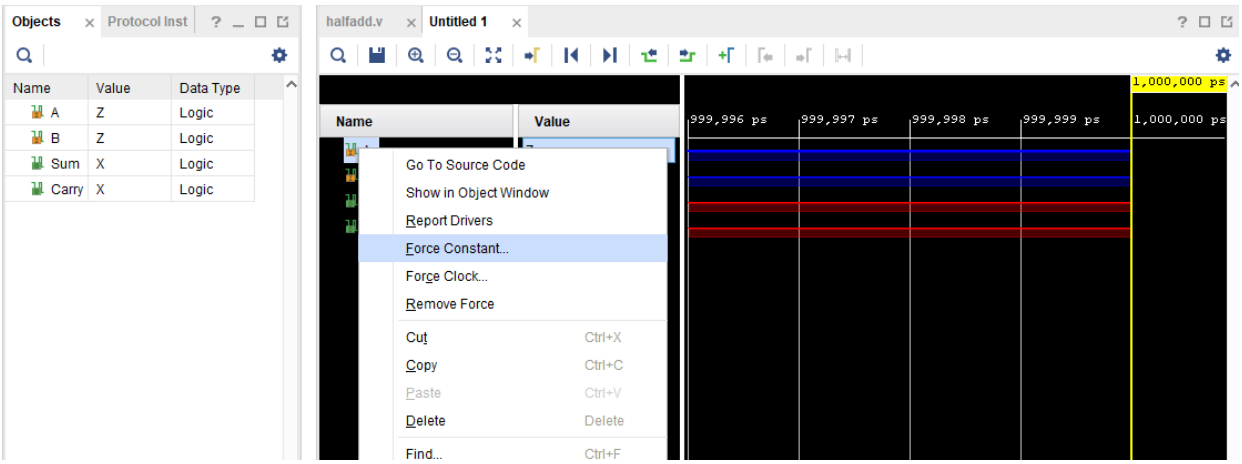

Figure 15: Force constant or clocks to the Inputs to generate various test sequences.

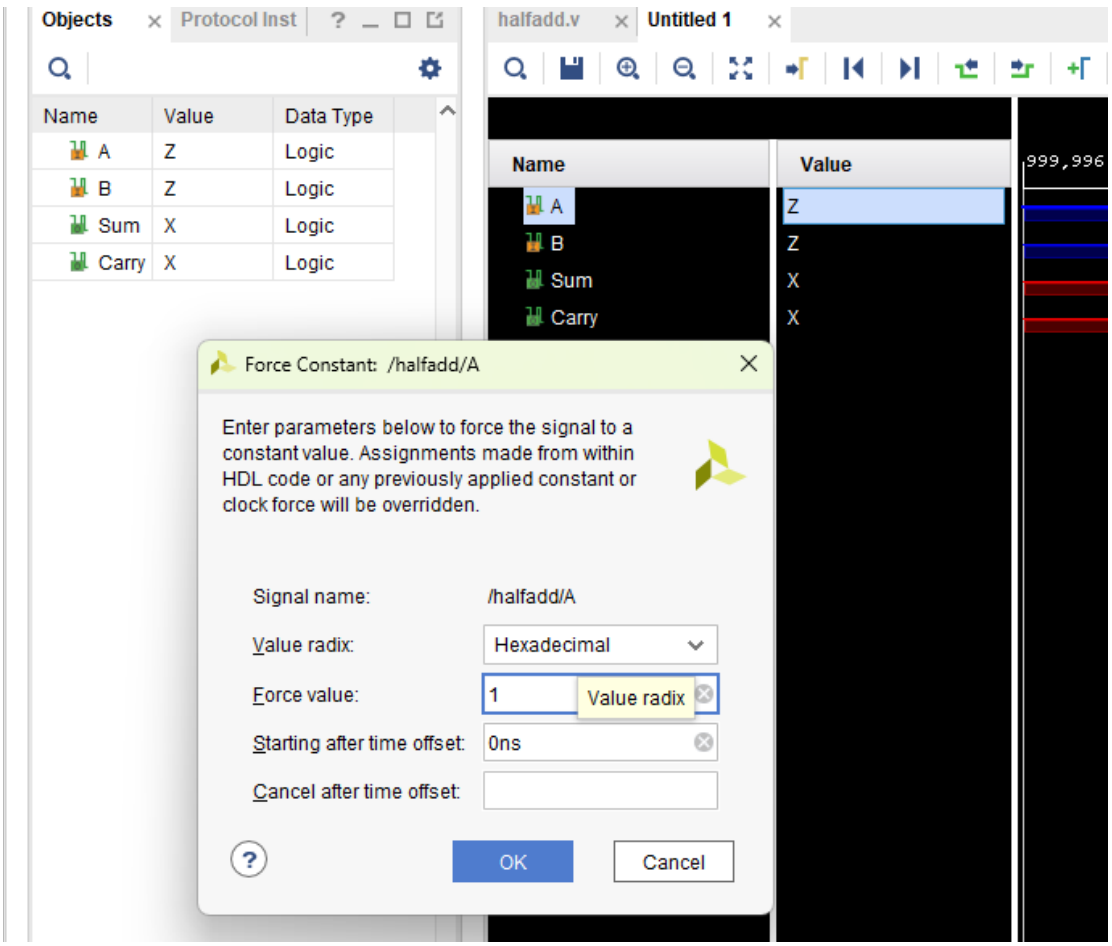

Figure 16: Give values in the Force value.

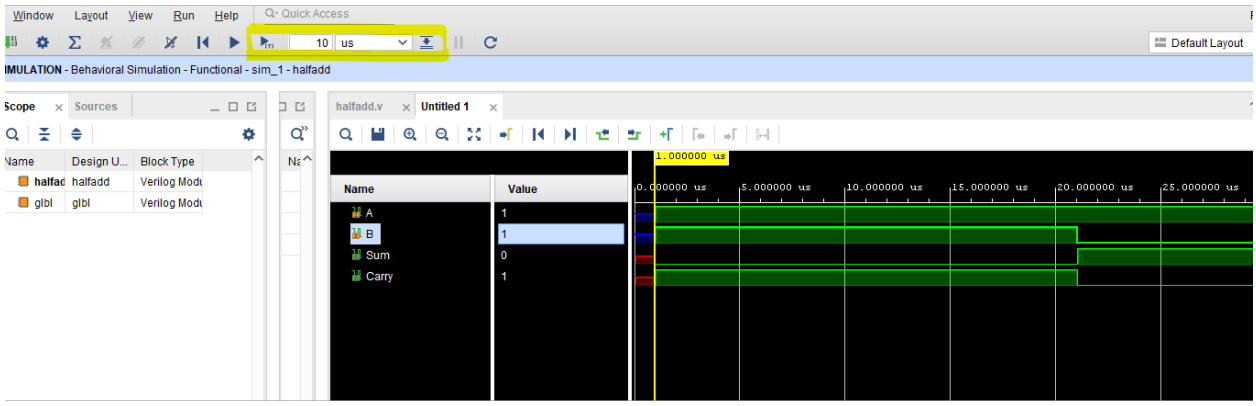

Figure 17: Verify the outputs generated for corresponding Inputs. Click on the Run option for the runs.

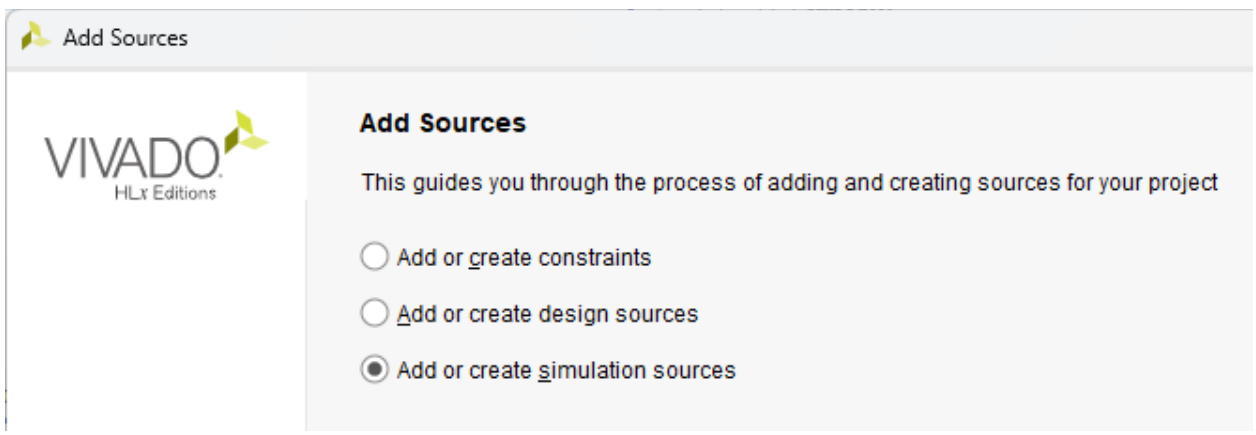

Figure 18: Another method is to run with the testbench file. Use the Add or create simulation sources option to add the testbench file.

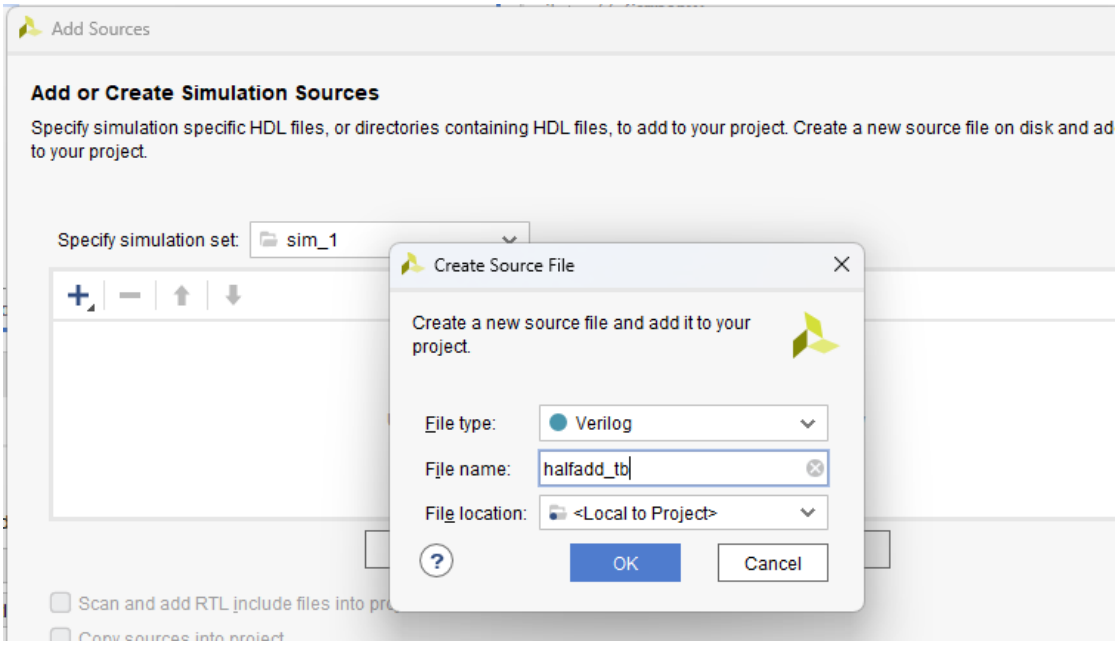

Figure 20: Name the file and click OK.

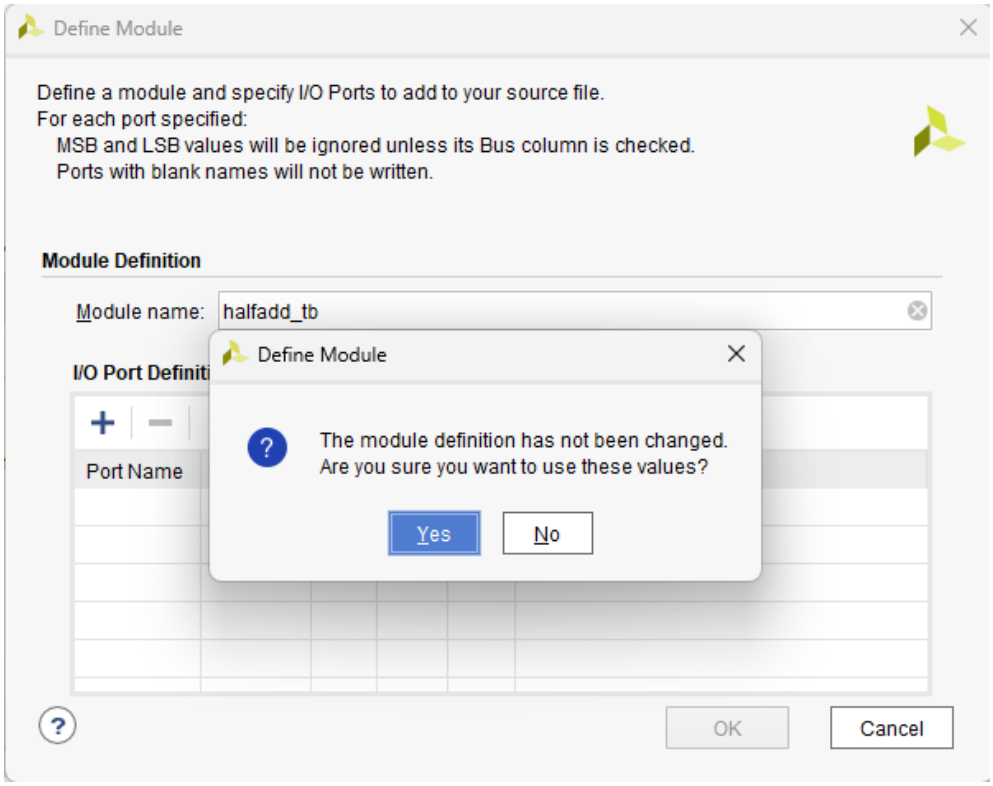

Figure 20: Click Yes and then OK.

Project Summary | halfadd.v | halfadd_tb.v *

C:/Users/deeks/project_1/project_1.srcs/sim_1/new/halfadd_tb.v

```
////////////////////////////////////////////////////////////////////////////////
module halfadd_tb( ); // Testbench module

    // Declare testbench signals
    reg A, B;  // Inputs to the halfadd module
    wire Sum, Carry;  // Outputs from the halfadd module
    // Instantiate the halfadd module
    halfadd uut (
        .A(A),
        .B(B),
        .Sum(Sum),
        .Carry(Carry)
    );
    // Test vector generation
    initial begin
        // Display the output for each combination of inputs
        $display("A B | Sum Carry");
        $display("------------");

        // Test all combinations of A and B
        A = 0; B = 0;
        #10;  // Wait for 10 time units
        A = 0; B = 1;
        #10;
        A = 1; B = 0;
        #10;
        A = 1; B = 1;
        #10;
        // End the simulation
        $finish;
    end
    // Monitor outputs
    initial begin
        $monitor("%b %b | %b %b", A, B, Sum, Carry);
    end
endmodule
```

Figure 21: Add the test bench code as shown above and save the file.

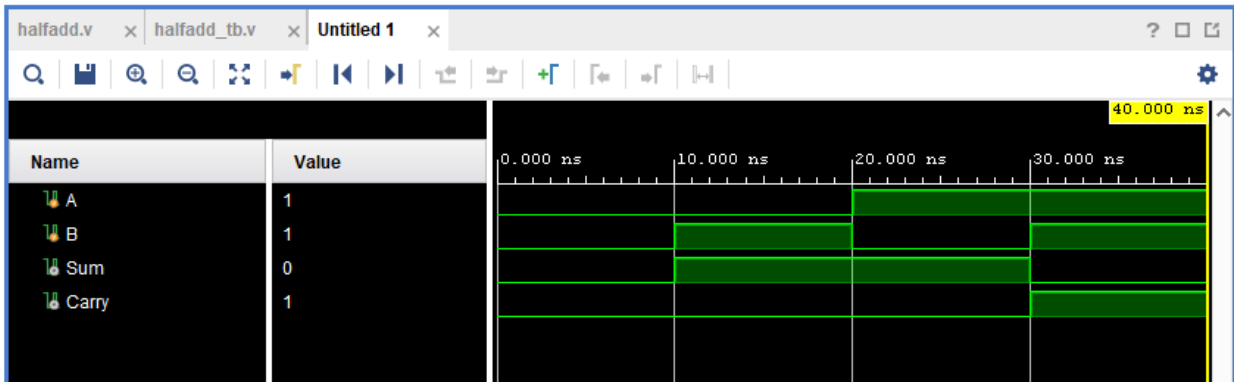

Figure 22: Click on Simulation and Run Behavioral Simulation. Behavioral Simulation will open with the following windows.

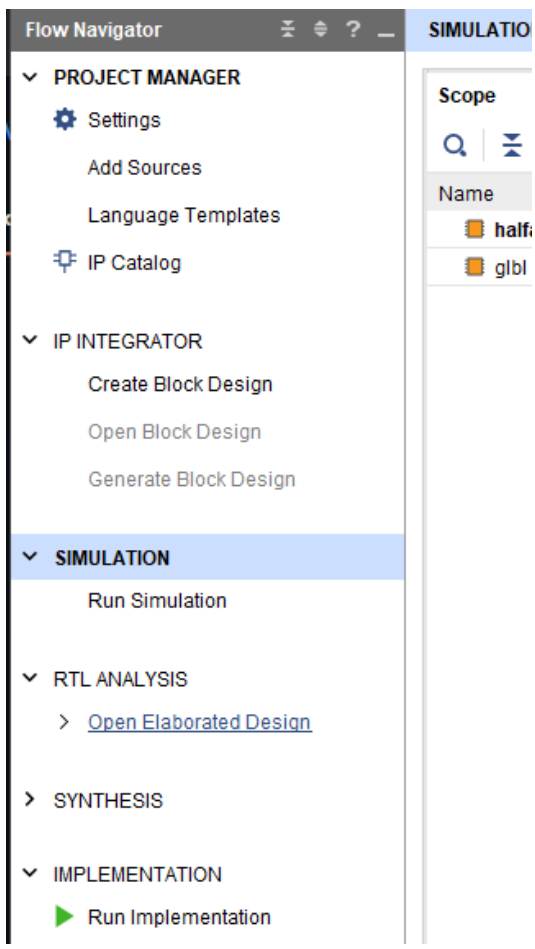

Figure 23: Now click on Open Elaborated Design from RTL ANALYSIS.

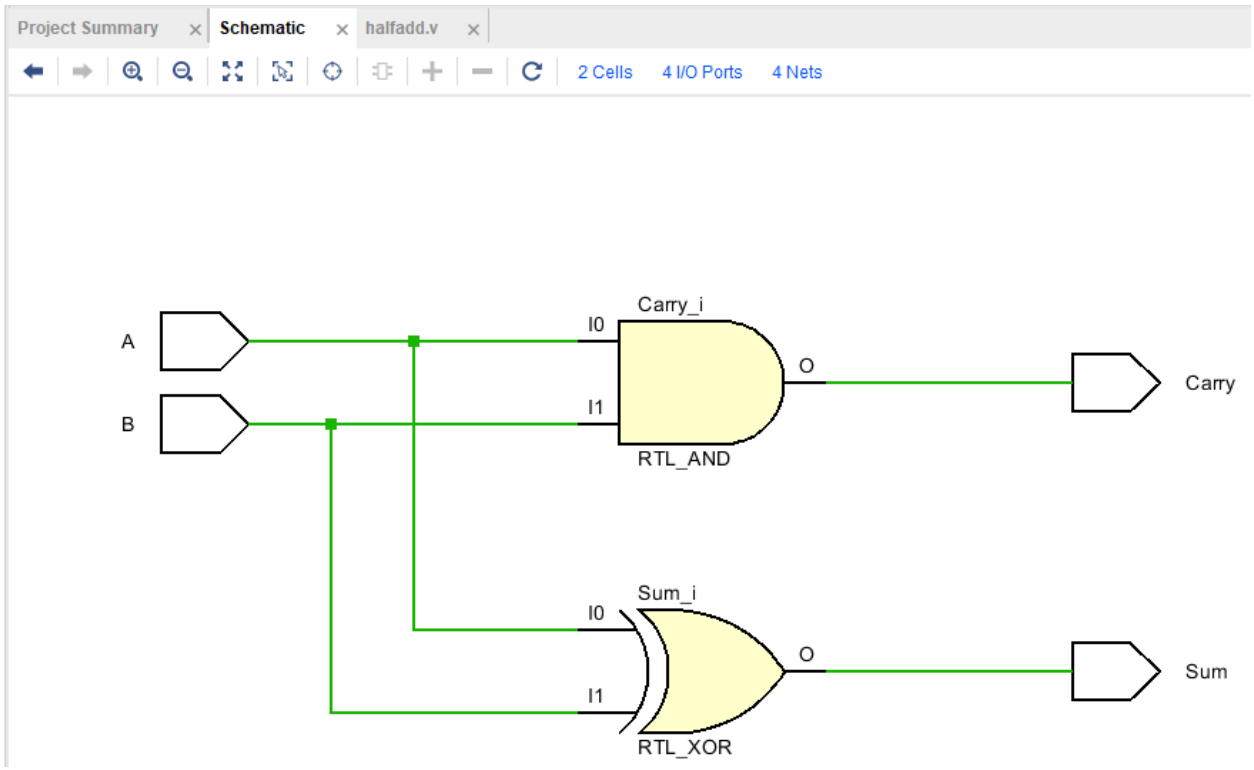

Figure 24: Schematic will be displayed.

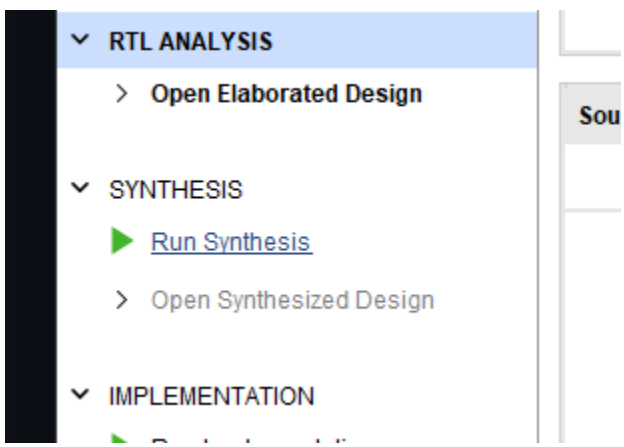

Figure 25: Now click on Run Synthesis.

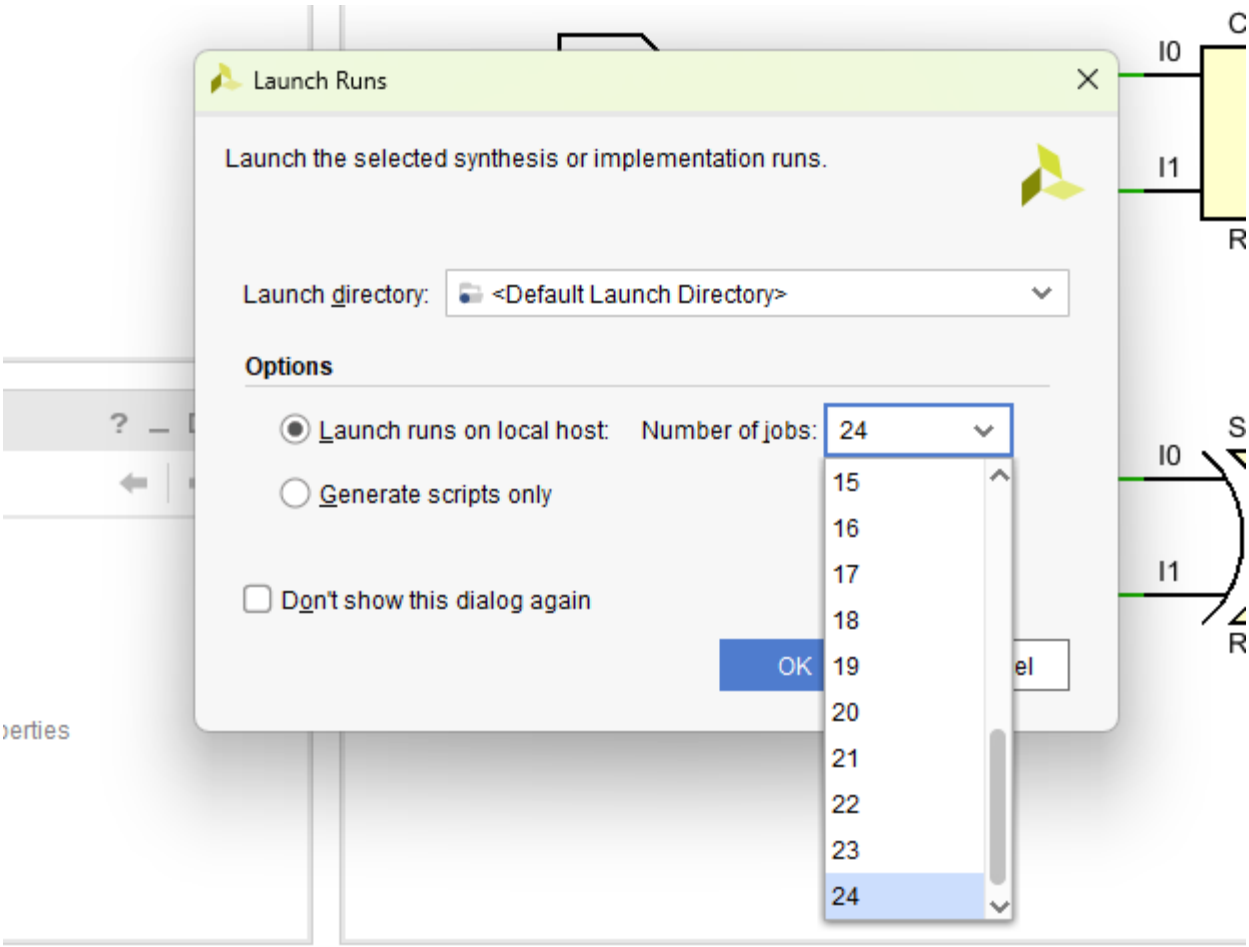

Figure 26: Select the Number of jobs according to your machine and click OK.

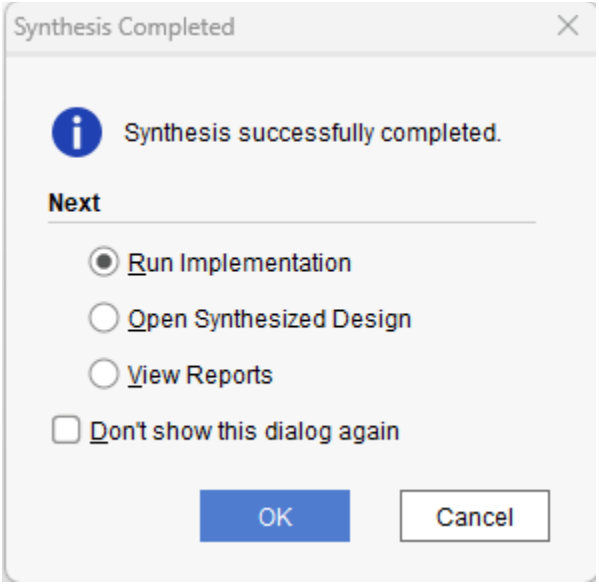

Figure 27: After successful synthesis, run the Implementation.

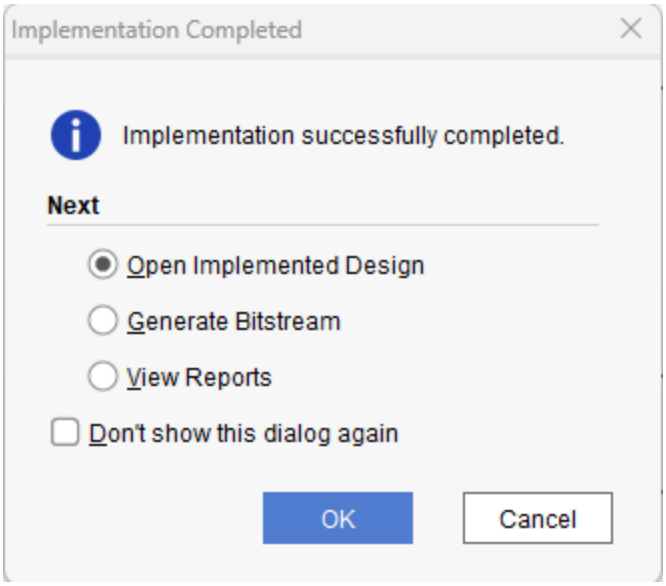

Figure 28: Open the Implemented Design after successful Implementation.

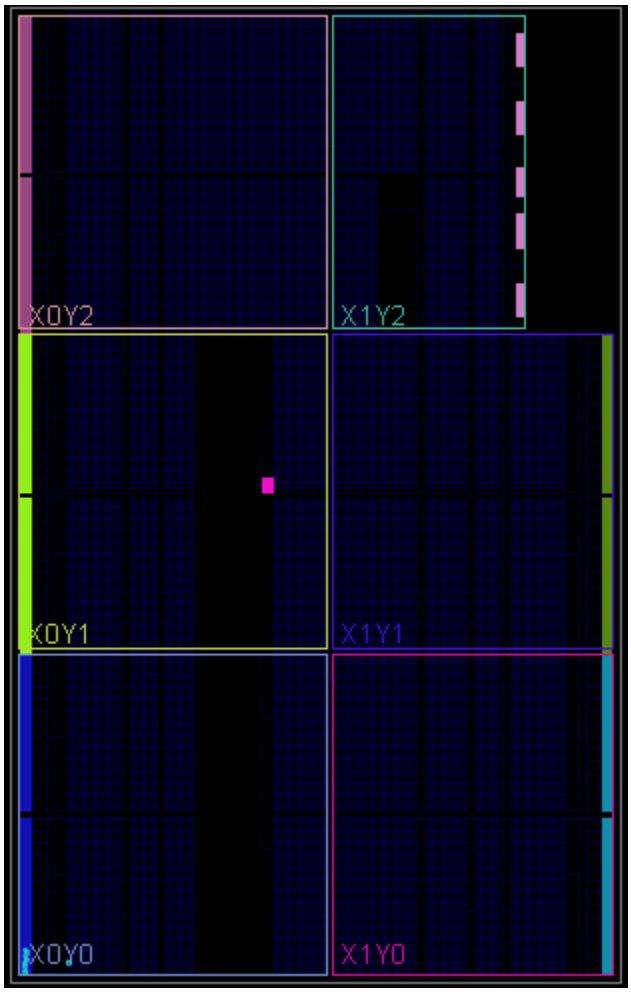

Figure 29: The Artix7 FPGA architecture

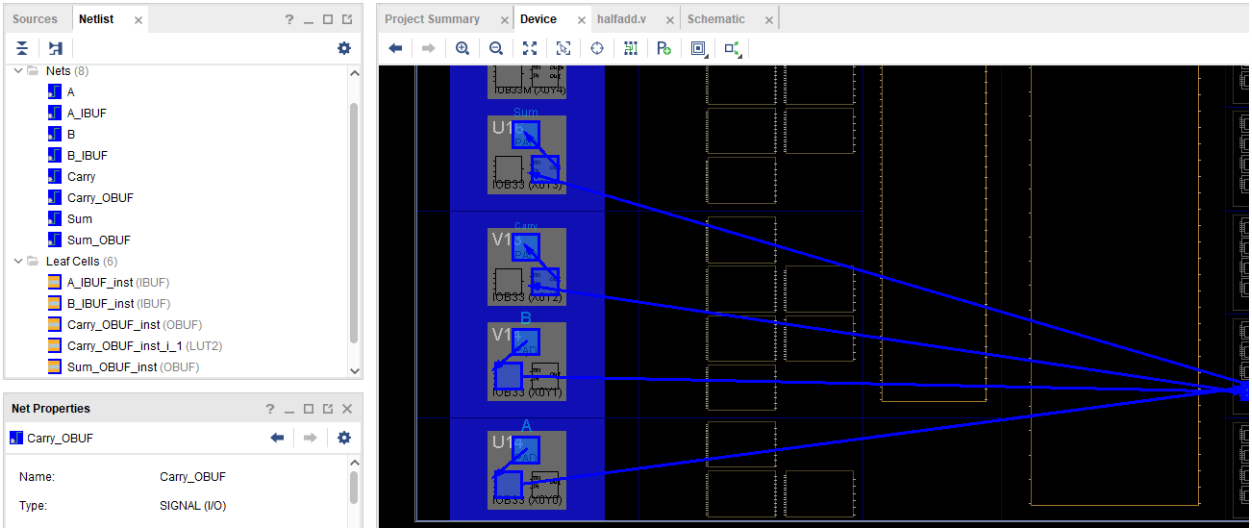

Figure 30: Zoom In to see the implemented Logic, and select different Nets to visualise the connections.

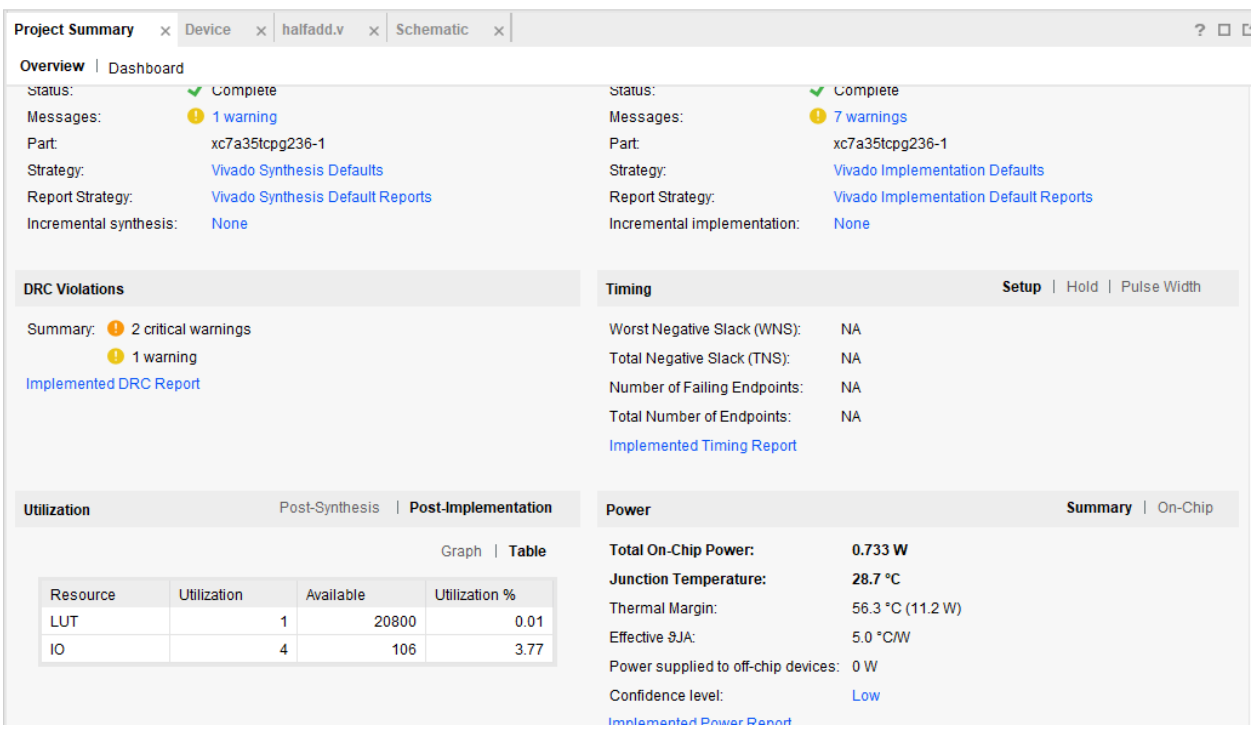

Figure 31: Project Summary gives details on resource utilization, timing and other details.

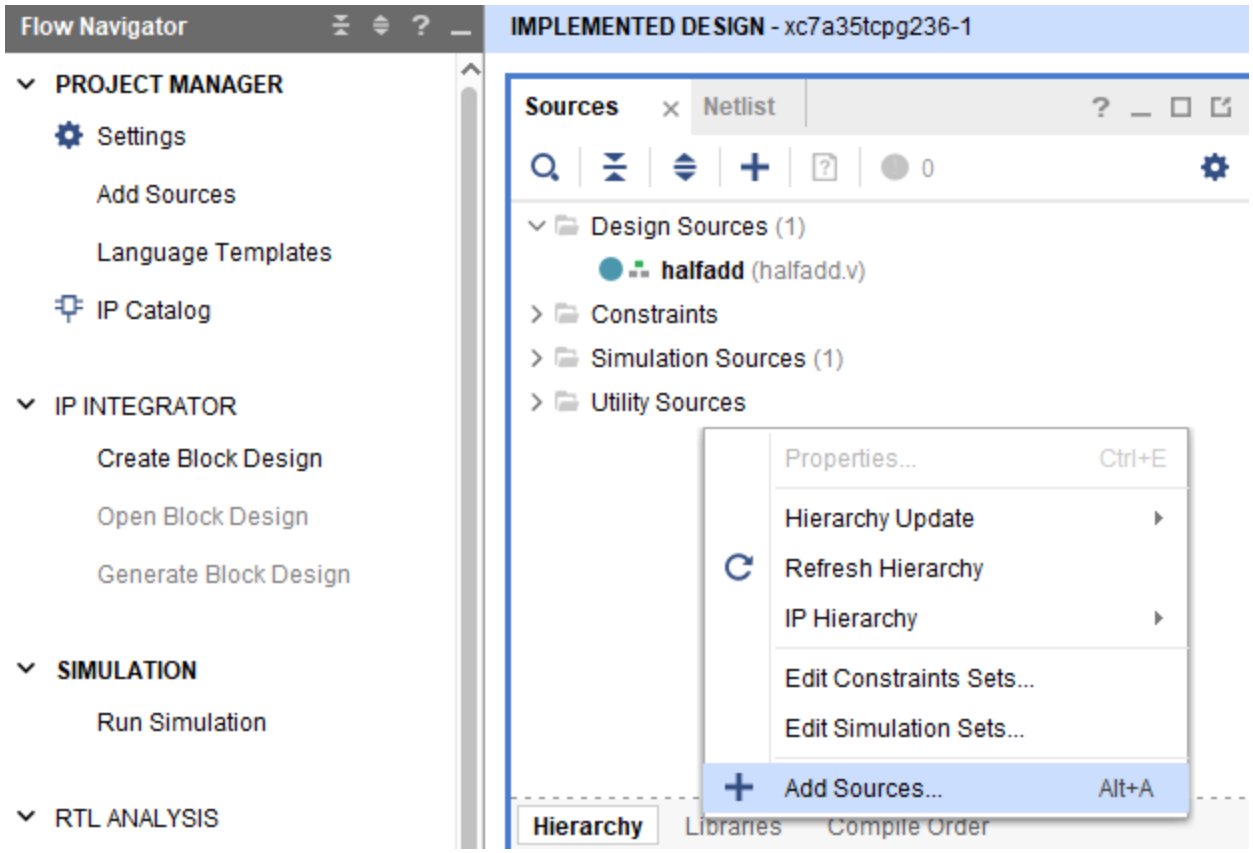

Figure 32: After successful implementation add the .xdc file into the project.

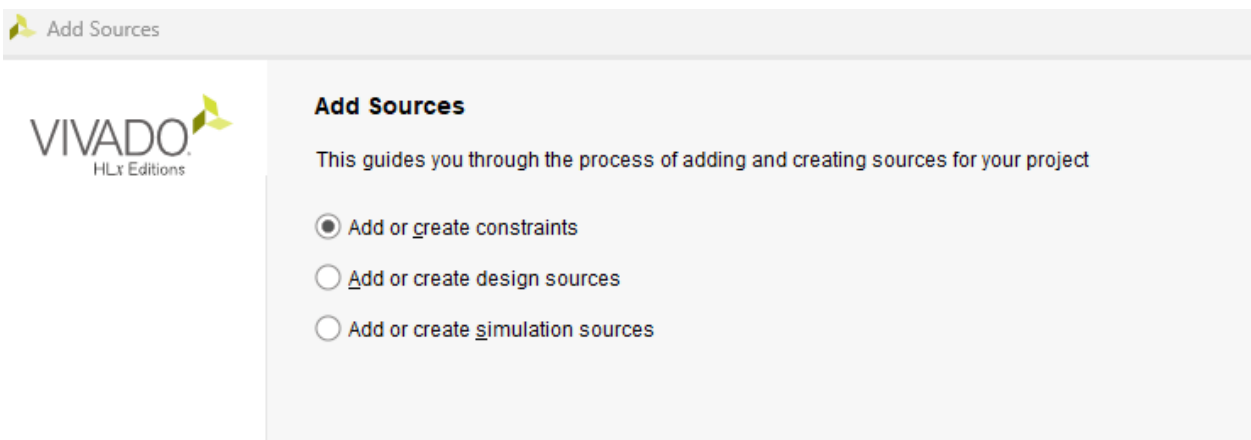

Figure 33: Select Add constraints option.

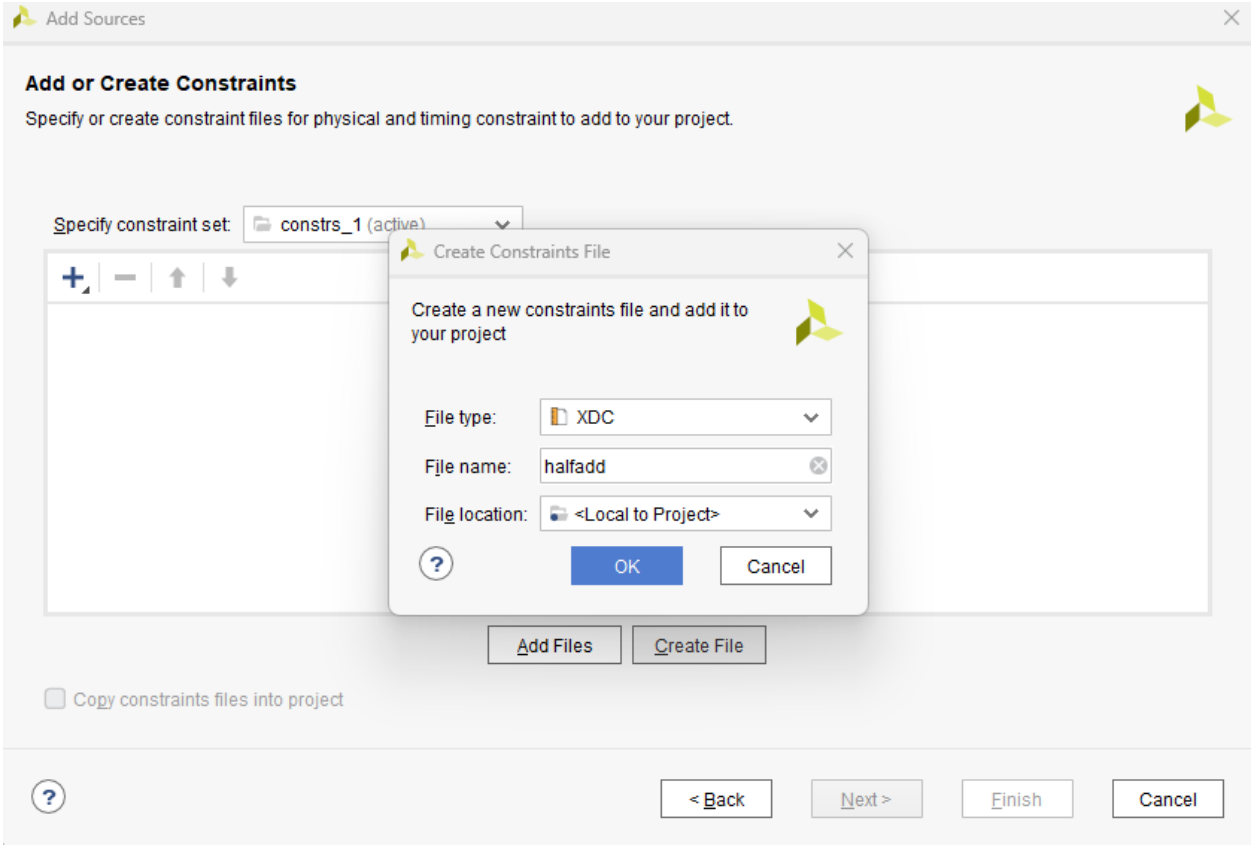

Figure 34: Create File and give a name to your .xdc file

Figure 35: .xdc file shows up in the Sources window under Constraints. Now open the file and write the constraints to make connections between the IOs and the peripherals on the board. (set_property PACKAGE_PIN <pin> [get_ports <port>]; set_property IOSTANDARD LVCMOS33 [get_ports <port>])

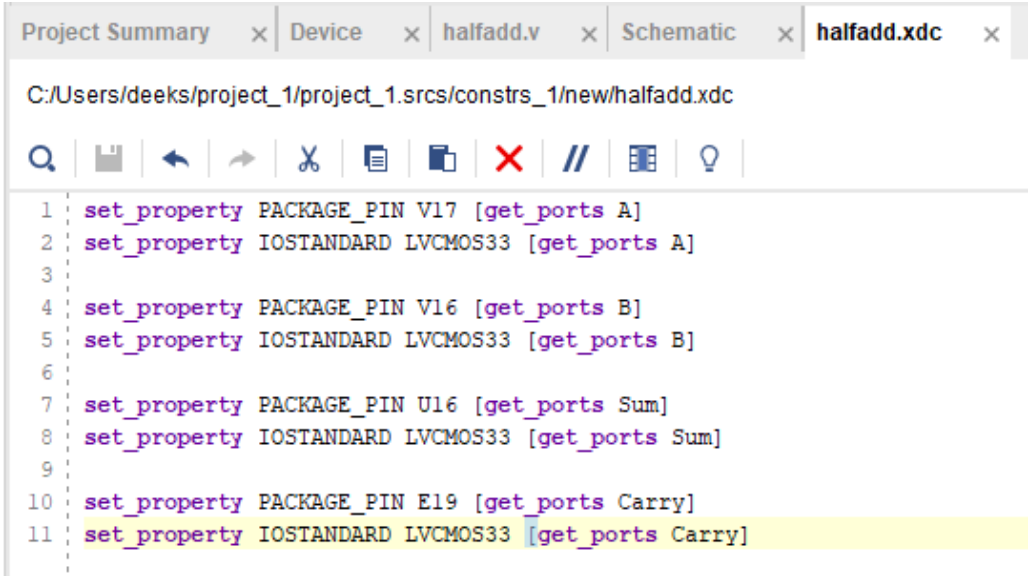

Figure 36: The constraint file is added according to the basys3 board input and output pins. (Source)

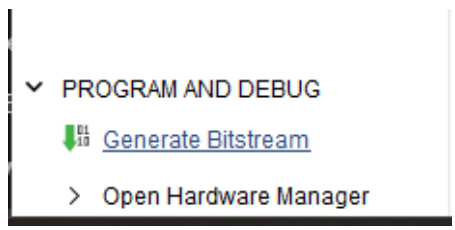

Figure 37: After completing the .xdc file, click on Generate Bitstream.

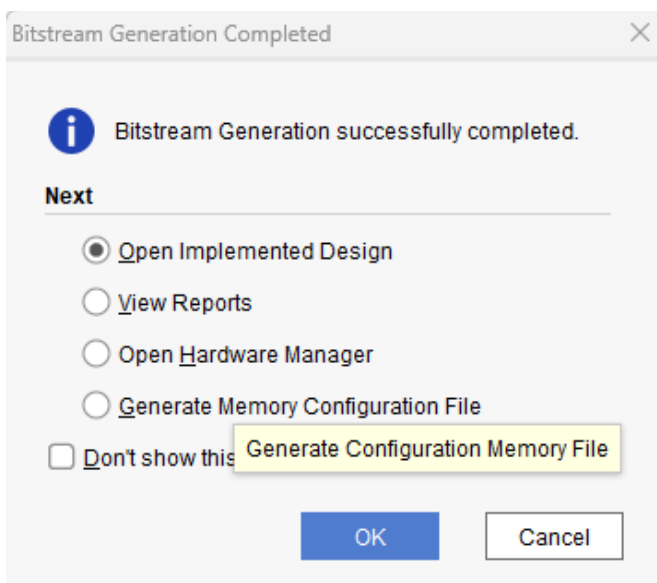

Figure 38: The above window will appear on successful Bitstream Generation.