

### Lab 2 – Introduction to Xilinx Vivado Design Environment

---

#### Objective

After completion of this lab exercise you will be able to use the Vivado logic design environment to capture, simulate, test, and download a logic circuit to a Basys 3 Board.

#### Design Flow Overview

The heart (brains) of the Digilent Basys 3 development boards, which we'll be using in lab, is the Xilinx Artix FPGA. An FPGA is a type of programmable logic device – a Field Programmable Gate Array. Conceptually, an FPGA contains thousands and thousands of logic gates whose inputs and outputs can be connected to or disconnected from each other or to/from inputs and output devices like switches and LEDs by applying “programming voltages”. If a given interconnection is “programmed” with a logic 1 then a connection is made otherwise it is not. Thus, when we “program” an FPGA we ARE NOT WRITING SOFTWARE but in essence specifying the hardware connections between logic components – we are wiring the circuit! Below are the steps used to implement modern logic designs using programmable logic devices.

**Design Entry:** You enter your design into the system through an HDL (Hardware Description Languages such as Verilog, VHDL or System Verilog). In general, your design may include different individual logic gates, combinational logic blocks, and/or sequential logic blocks.

**Verification:** Simulators are used for functional and timing verification of the design. Functional simulation verifies the logic behavior of the system without any knowledge of the underlying target device and does not provide any timing information. Timing simulation provides various timing analysis after the design has been compiled for a specific target device.

**Implementation:** Design tools like Vivado translate your design into an optimized format suitable to your target device. The output of this step is a *bit stream file* (or a .bit file) that can be downloaded into the hardware. The bit stream is the list of connections that are to be made within the FPGA.

Figures 1&2 show more detailed design flow diagrams for FPGA devices. As mentioned above, our projects are targeted for Xilinx Artix 7 FPGAs.

***If you have already finished the pre-lab, jump to section Creating Constraint File on page 10.***

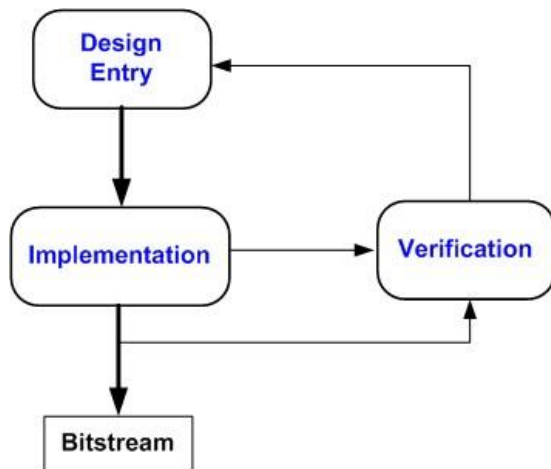


Figure 1

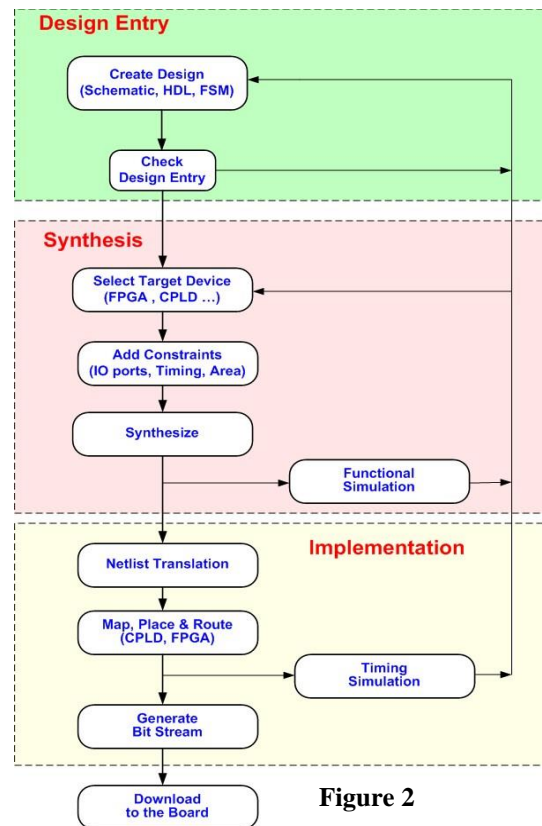


Figure 2

## Pre-lab Assignment

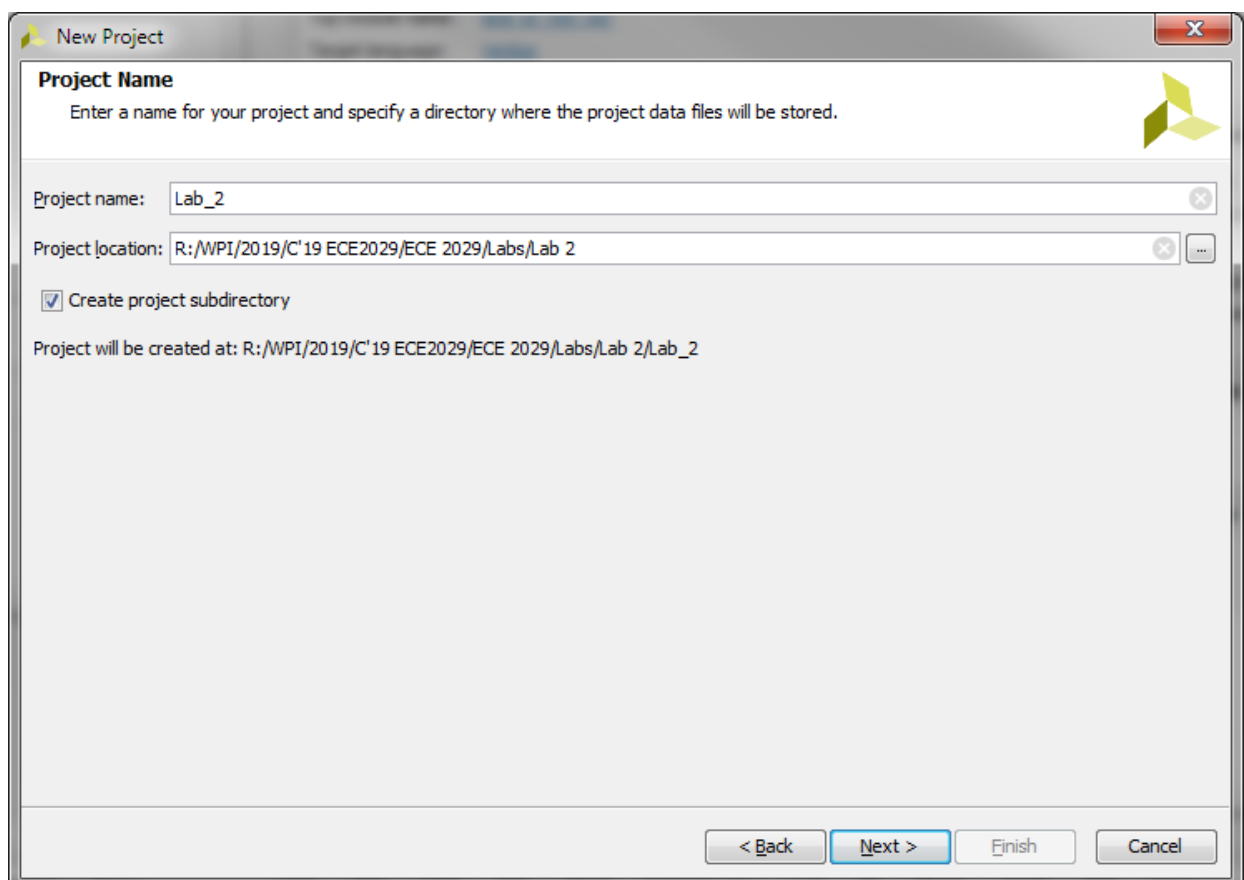
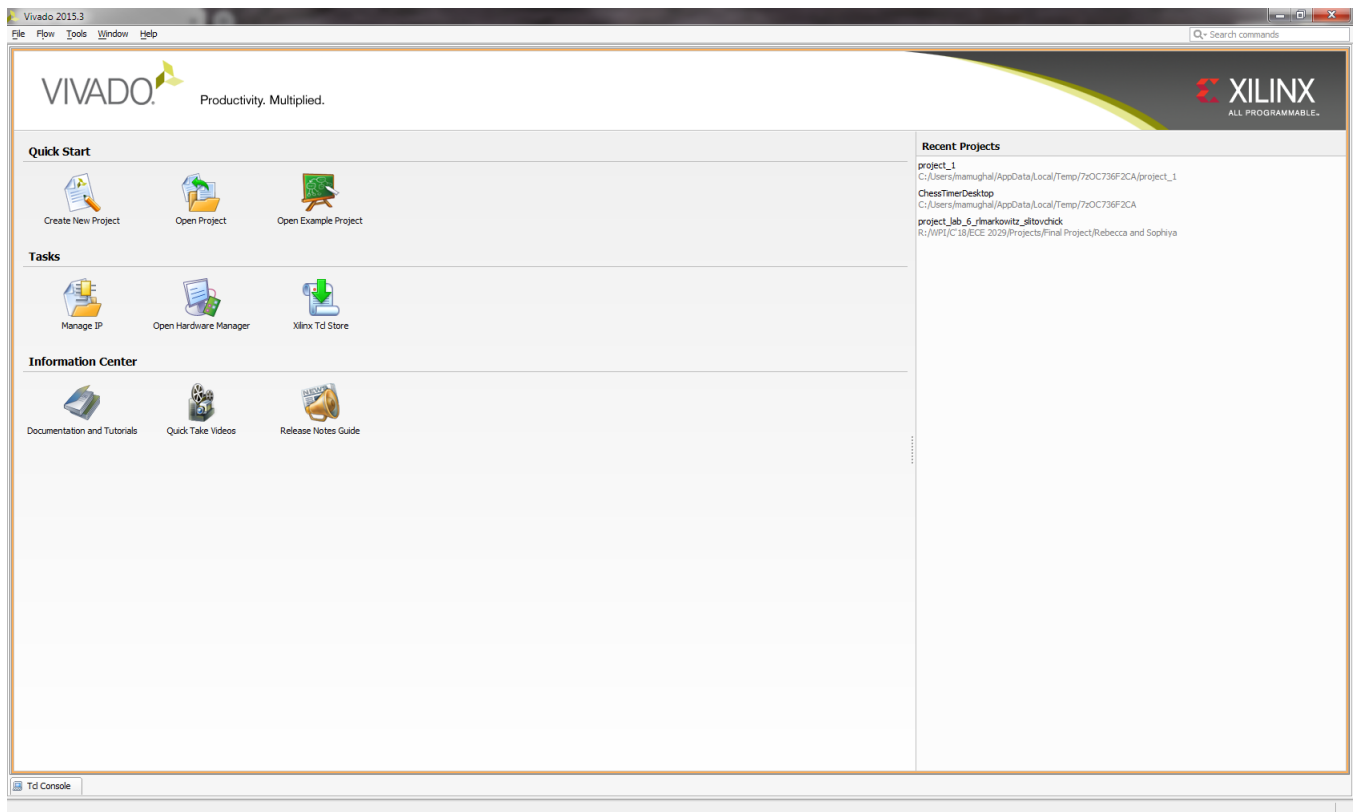
Usually, laboratory exercises will have pre-lab assignments which are to be completed before your lab session and must be signed-off by the TA during your lab session.

Pre-labs help you to become oriented to the problem before you enter lab, help complete your design in advance and prevent wasting time in lab. This second lab is a straightforward tutorial and does not have a pre-lab.

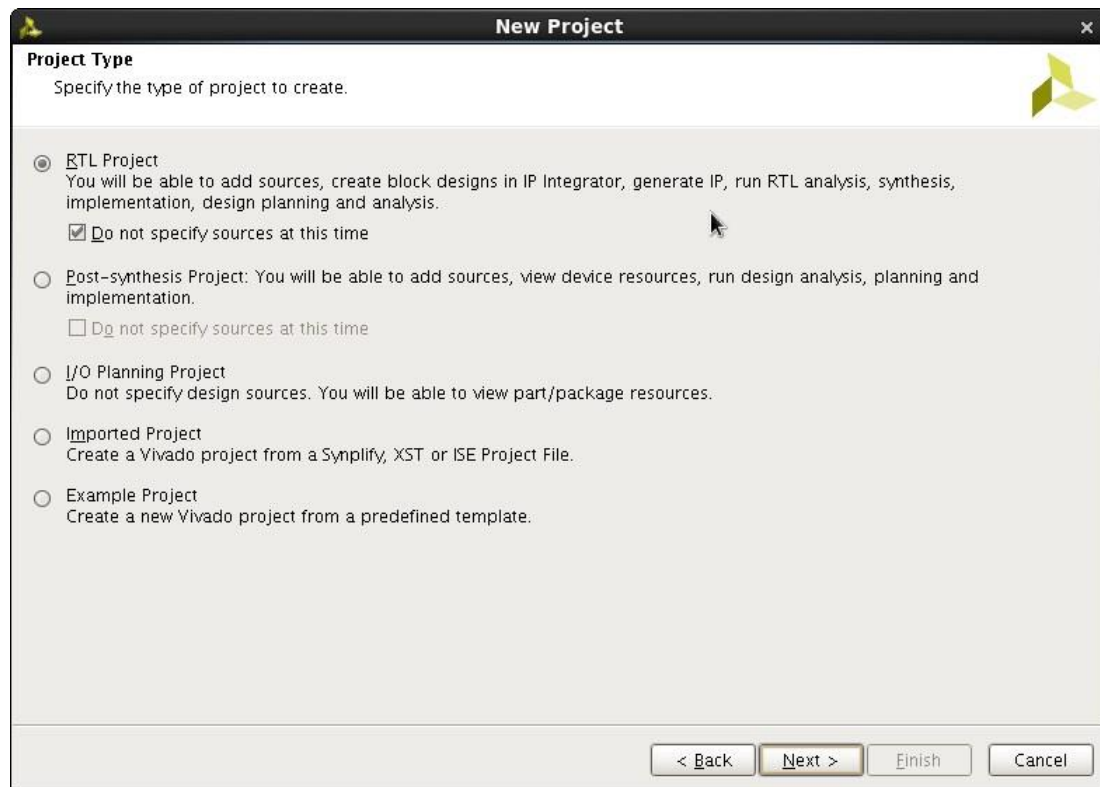
## Assignment

In this second lab you will start by implementing the basic logic gates AND, OR, NOT and the compound exclusive OR gate (XOR) and verify their truth tables. Then you will implement a multi-gate circuit. You will do this using the Verilog hardware description language.

1. Start Vivado Design Suite by clicking on Vivado icon and then selecting Create New Project from the menu. And click next to move to next step.
2. Enter a project name. You should use **your network drive** as the project location for your files. Make sure there are **no spaces or special characters (- and \_ are allowed)** in the folder name that you create for your work. If you must create your project on the local disk be sure to back it up to your network drive when you're done at the end of lab. Lab machines can be re-imaged at any time and you could lose your work if it is not on your network drive! Each partner should have a copy of all the work!



Click Next and then select the RTL project type. Be sure to check the “Do not specify sources at this time” box and click Next:



3. Click next and select the correct Xilinx Artix 7 FPGA that is on the Basys3 board (XC7A35T-1CPG236C)

**New Project**

**Default Part**  
Choose a default Xilinx part or board for your project. This can be changed later.

Select: ☒ Parts ☐ Boards

Filter

Product category: All Speed grade: -1

Family: Artix-7 Temp grade: All Remaining

Package: cpg236

Reset All Filters

Search:

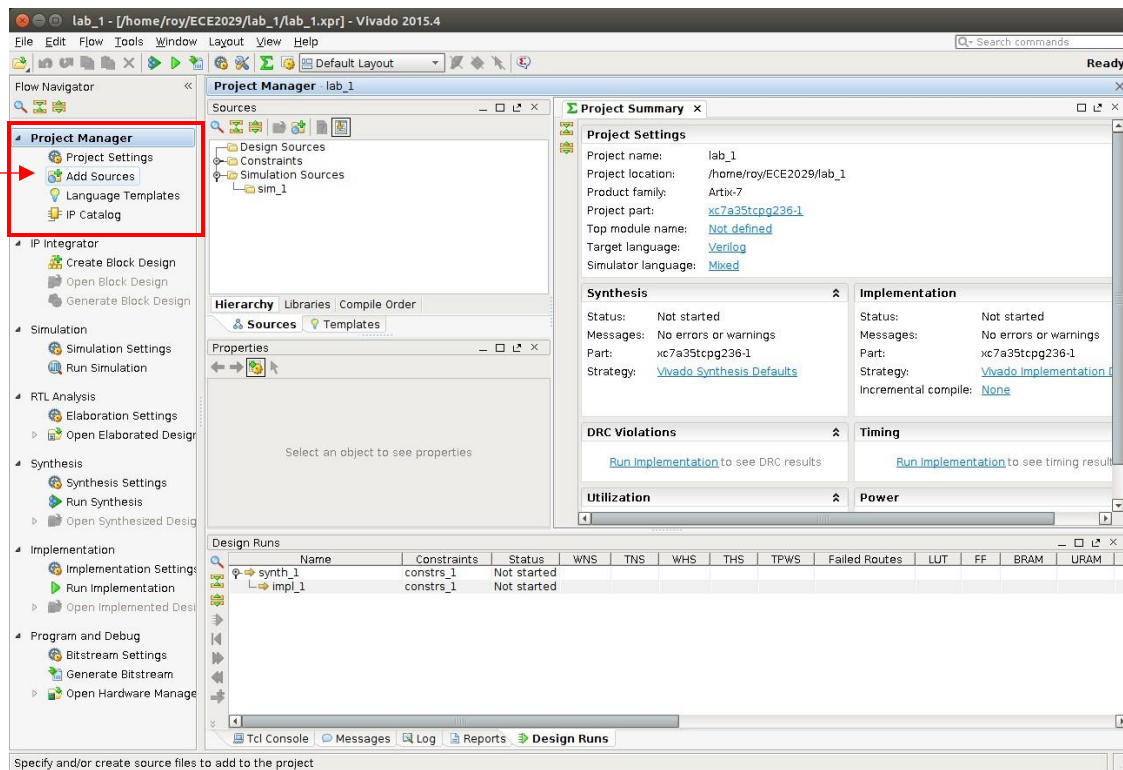
Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs	L E
xc7a15tcpg236-1	236	25	45	20800	2	2	106	10
xc7a35tcpg236-1	236	50	90	41600	2	2	106	20
xc7a50tcpg236-1	236	75	120	65200	2	2	106	32

< Back Next > Finish Cancel

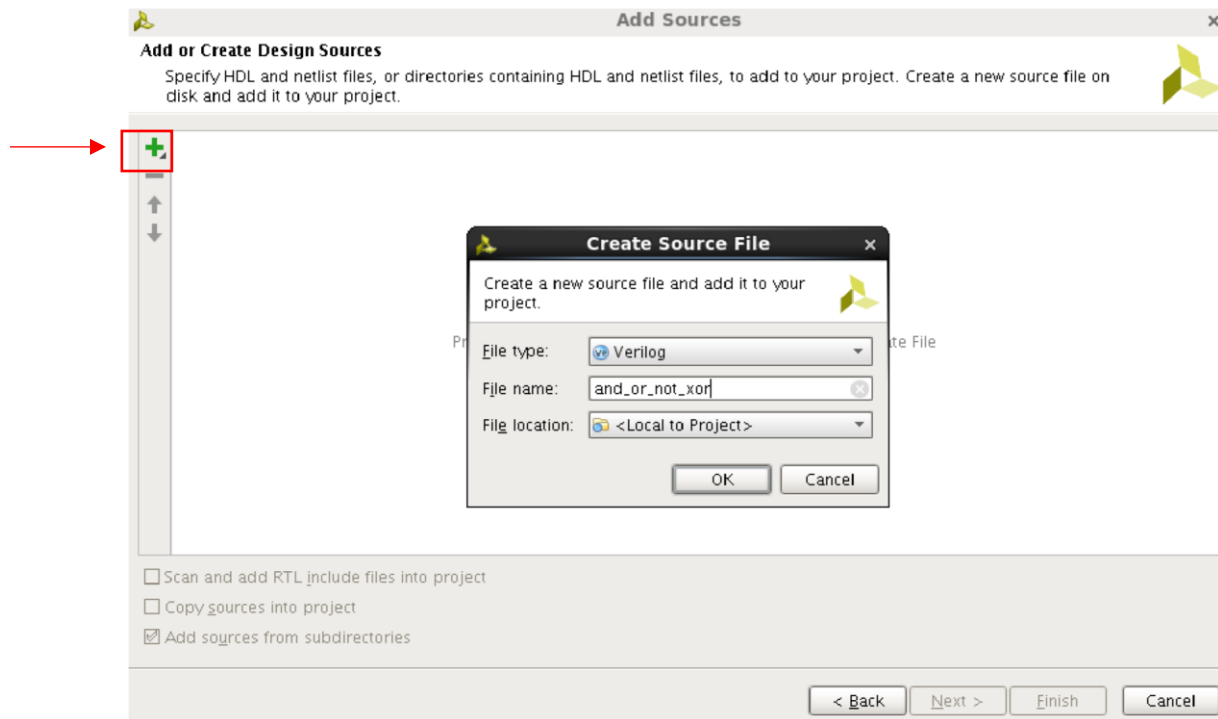
Click Next, and then Finish on the New Project Summary Page. The Project Window opens:

## Design Entry

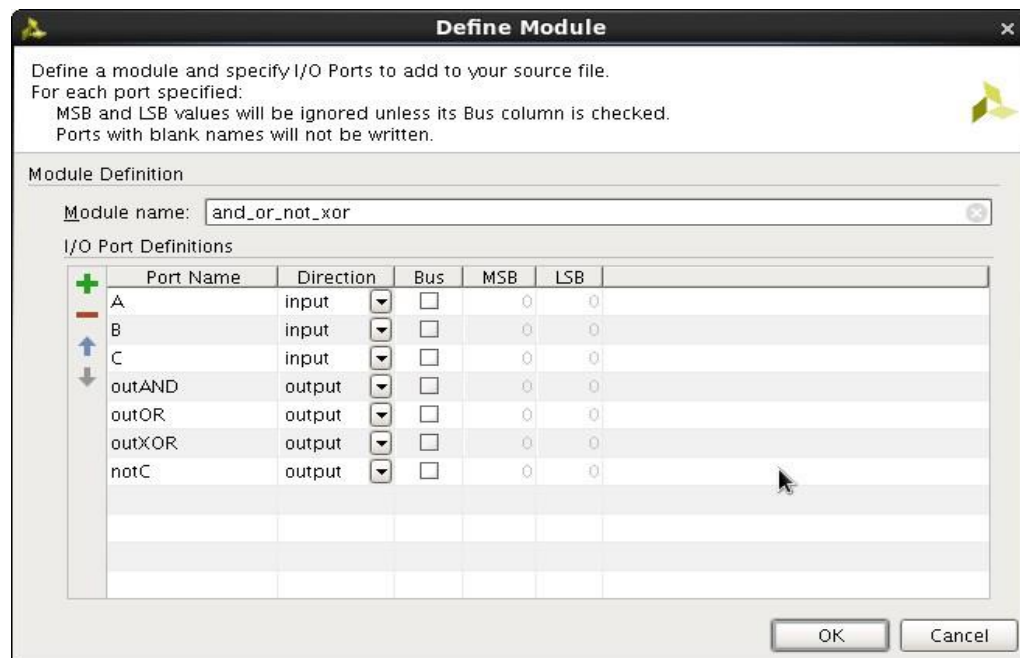
- Under the Project Manager, select Add Sources to begin designing your new project.



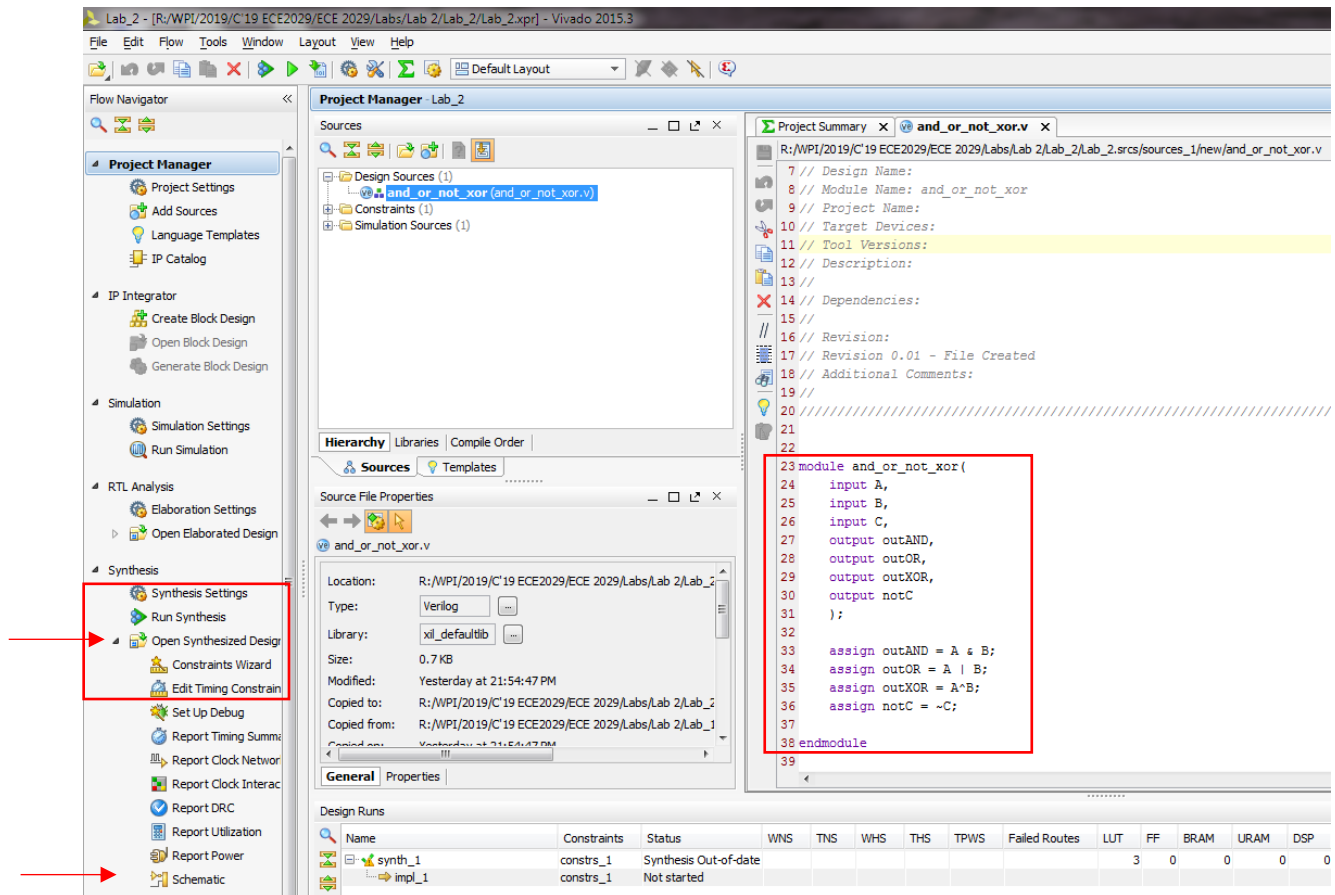
5. Select Next and then select Create File (click on the + symbol) and enter the file name for the circuit.



6. Then click OK and Finish. We can now specify the inputs and outputs to create our AND, OR, NOT and XOR gates. We will use three slide switches as inputs and 4 LEDs on the Basys3 board to display the outputs of each gate. Inputs A and B will be the inputs to the AND, OR and XOR gates, and C will be the input to the NOT gate.

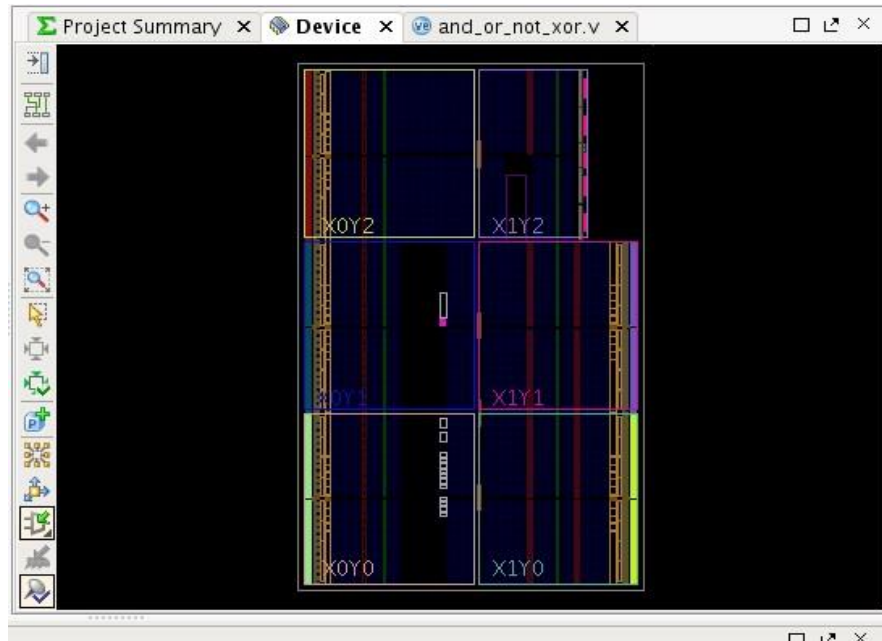


7. Click OK. Back in the Project Manager Sources window double-click the new and\_or\_not\_xor.v file and you will then see the Verilog file appear in the window on the right. Add your name and a description of this file to the header description. You should ALWAYS complete the comment block at the top of each Verilog module that you write. This is basic professionalism.
8. We can now add the Verilog statements to design our gates. Enter the code below to your and\_or\_not\_xor.v file.

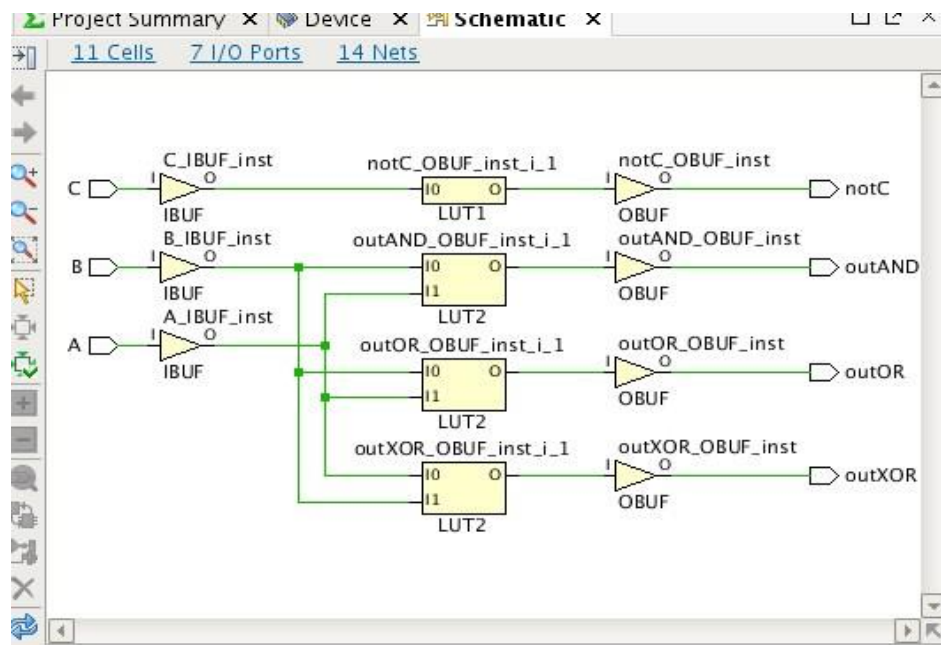


9. Now we can synthesize the design. Click Run Synthesis in the Project Manager window. After synthesis is complete there should be no errors or warnings reported. If you open the synthesized design you can see a device level representation (this is mostly empty since we just have a very simple design that only uses a tiny, tiny fraction of the available FPGA resources).





10. You can also look at a schematic representation to see the input and output buffers and the Look Up Tables (LUT) used. LUTs can be used to implement any truth table with a certain number of inputs. Notice that since AND, OR and XOR all have A and B as inputs they are implemented in the same LUT.



## Creating Constraint File

- Before we can implement the design we need to specify the FPGA pins that will be used for the SW inputs and LED outputs. The [Digilent Basys 3 Reference Manual](#) provides detailed information about the Basys3 Starter board. Figure 16 shows the overall architecture of the board with its major components. Basys3 board provides sixteen sliding switches (SW0-SW15) and five push buttons (BTN5-BTN0) that can be used as inputs. Each of these switches is connected to an associated IO pin of the FPGA. Figure 16 in the Basys3 Reference Manual shows to which pins these digital IO devices are connected. The full address of these IO's (and everything on the board) is available in the [user constraint file, UCF](#) (also available on CANVAS under important course material module).

Both the slide switches and push buttons are active high, in other words when they are “on” they connect  $V_{CC0}$  = Logic 1 to the FPGA IO pin. Note that there is no de-bouncing circuitry for these inputs. There are sixteen surface mounted LED's on the board that can be used as outputs (LD15-LD0). The LED's are all active high, so to turn an LED ON you need to apply a logic high to the corresponding IO pin of the FPGA. For [Basys 3 board](#), see the FPGA pins connected to switches SW0, SW1 and SW2 and LEDs LED0, LED1, LED2 and LED3 below:

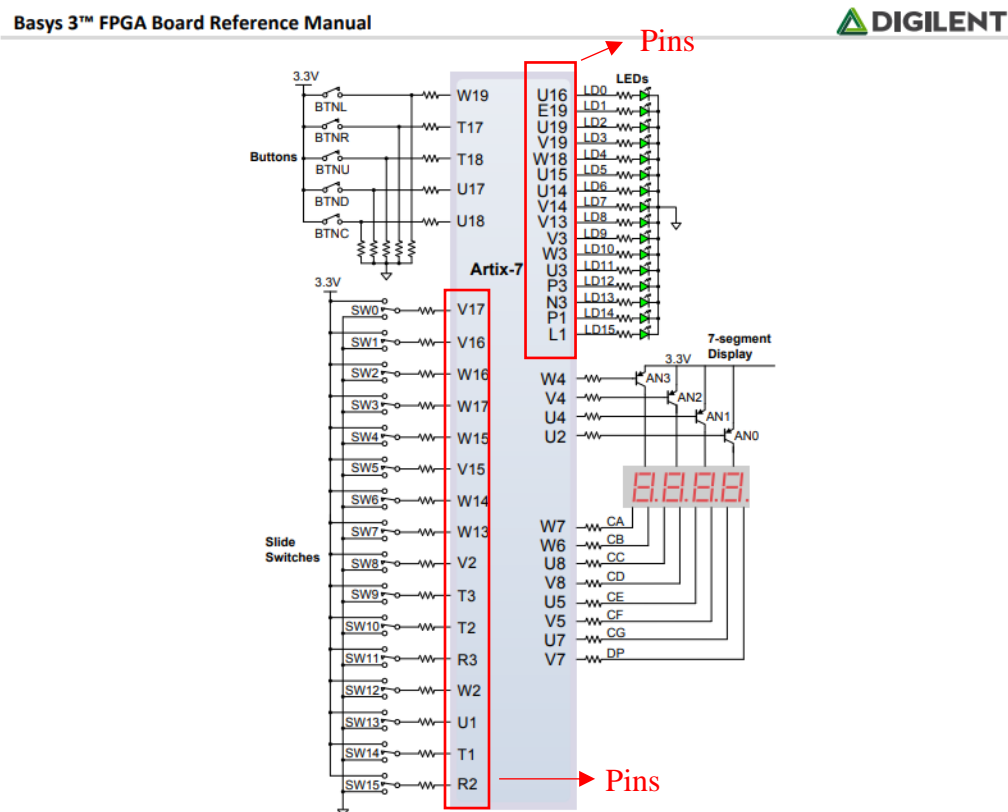
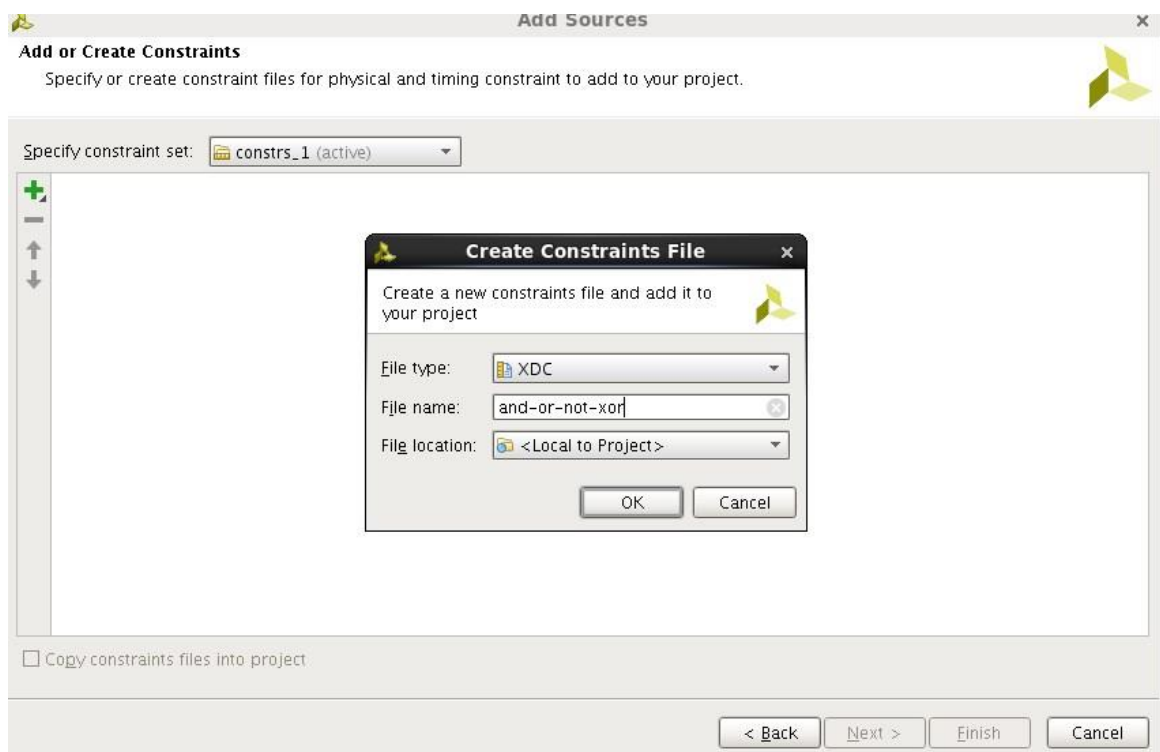


Figure 16. General purpose I/O devices on the Basys 3.

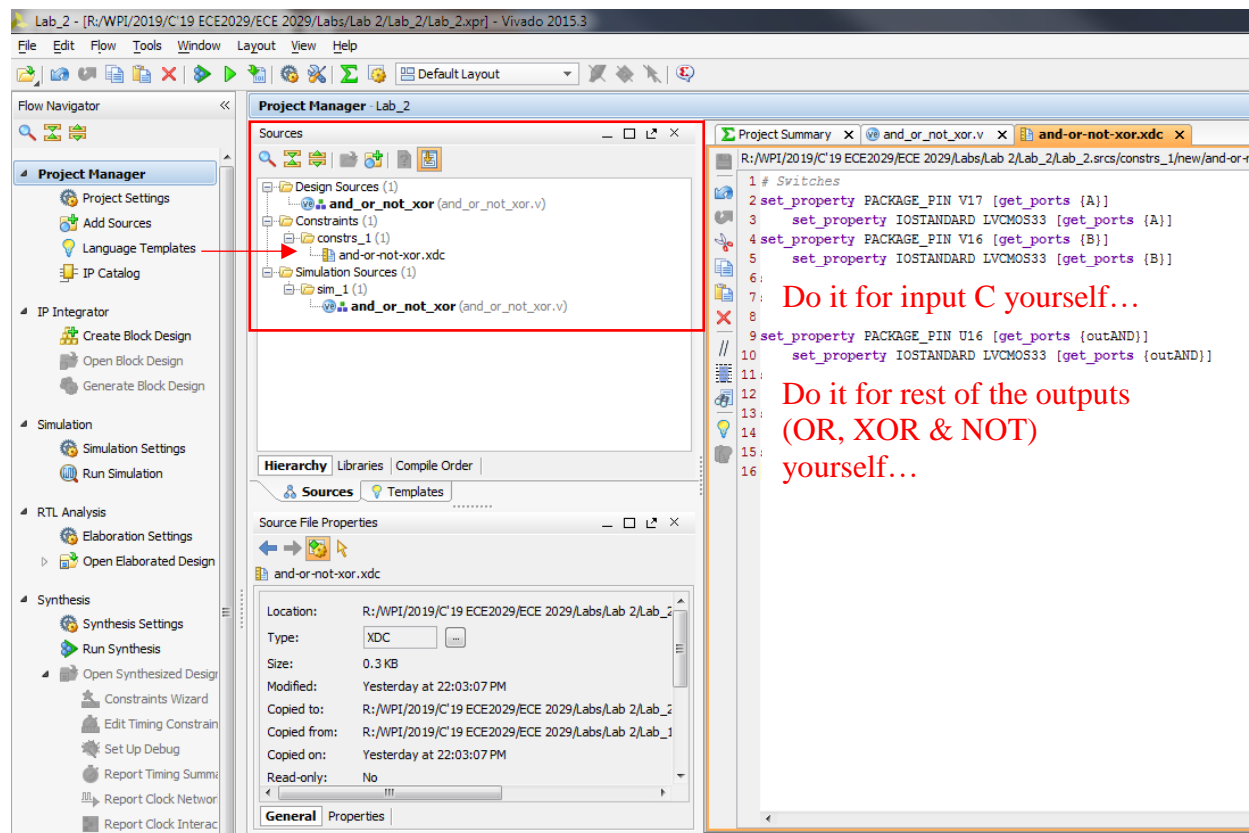
Next click Add Sources and select **Add or create constraints**



And name the constraints file.



12. In the sources window select the constraints file through the hierarchy Constraints => constrs\_1 => and\_or\_not\_xor.xdc. We can now add location constraints (or address) for all the inputs and outputs. These constraints specify the pins to use for each signal and what type of interface. Like mentioned earlier, you can download [user constraint file for Basys3 XDC here](#) – just copy the pins only you are using for the design. Or, just type in this file and modify it as needed. Remember the listing of pins (e.g. W17) to which the LED and switches are connected are given in the constraint file.

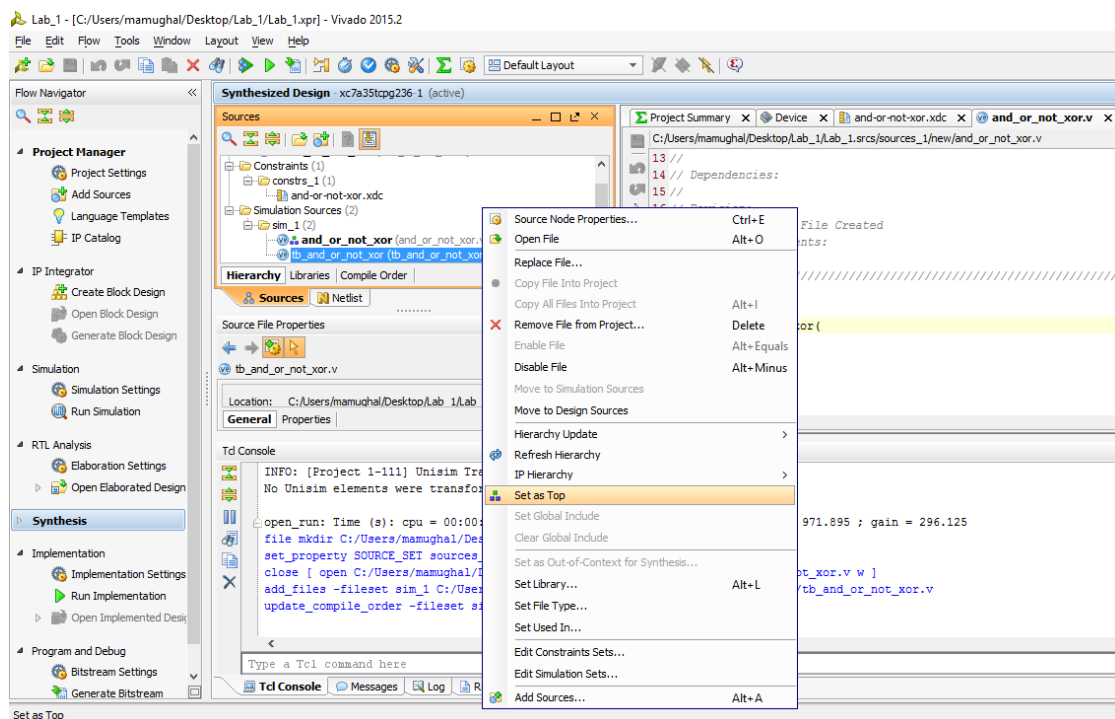


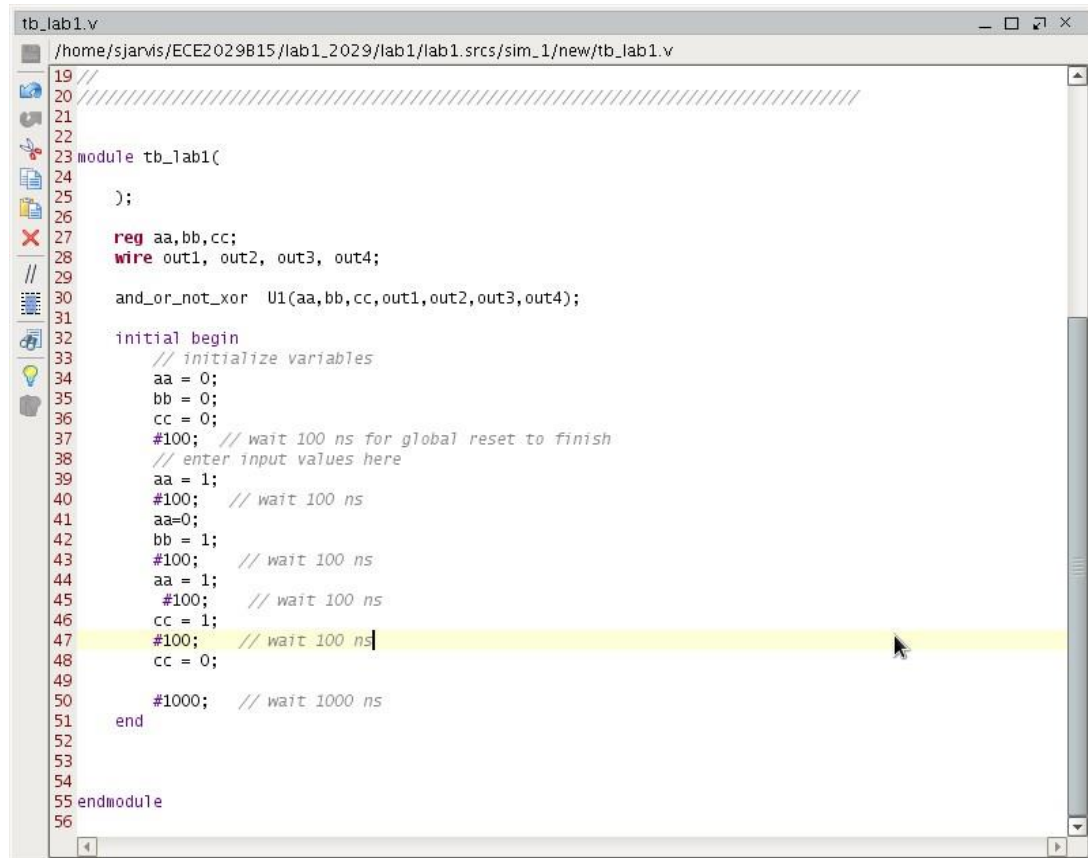
## Generating a Test-bench Waveform for Functional Verification

### Functional (Behavioral) Simulation for Design Verification

In this section you will learn how to create and use a Verilog Test Bench to verify the functional behavior of your design. Verilog is a simulation language as well as synthesis language. In a HDL simulation, the design you want to test is instantiated in the simulator. You will need to add the Verilog that asserts all input combinations you want to test. For our simple logic gates there are only 6 input combinations we need to consider (A=0 B=0, A=1 B=0, A=0 B=1, A=1 B=1 and C=0, C=1) so you can easily complete an “exhaustive test” of all possible inputs.

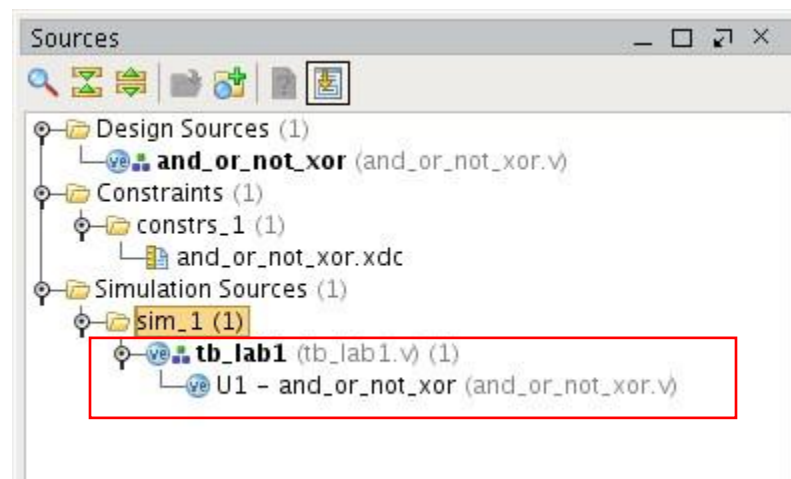
13. Under the Project Manager Simulation Sources, right click on sim\_1(1) and Add Sources. Choose Add or create simulation sources. Create a file and name it tb\_and\_or\_not\_xor or similar (tb\_lab1). The tb stands for testbench. Click finish.
14. Right click the new testbench file and Set it as Top module. Double click on the file name to open it in the editor window and add the following Verilog code. Notice that you are *instantiating* the and\_or\_not\_xor module you will define as U1. We are using simulator input signals called aa, bb and cc as inputs A, B, and C respectively and have assigned simulator outputs out1, out2, out3, out4 as outAND, outOR, outXOR, and notC respectively. Then we set the values of aa, bb and cc to the all the settings we wish to test.





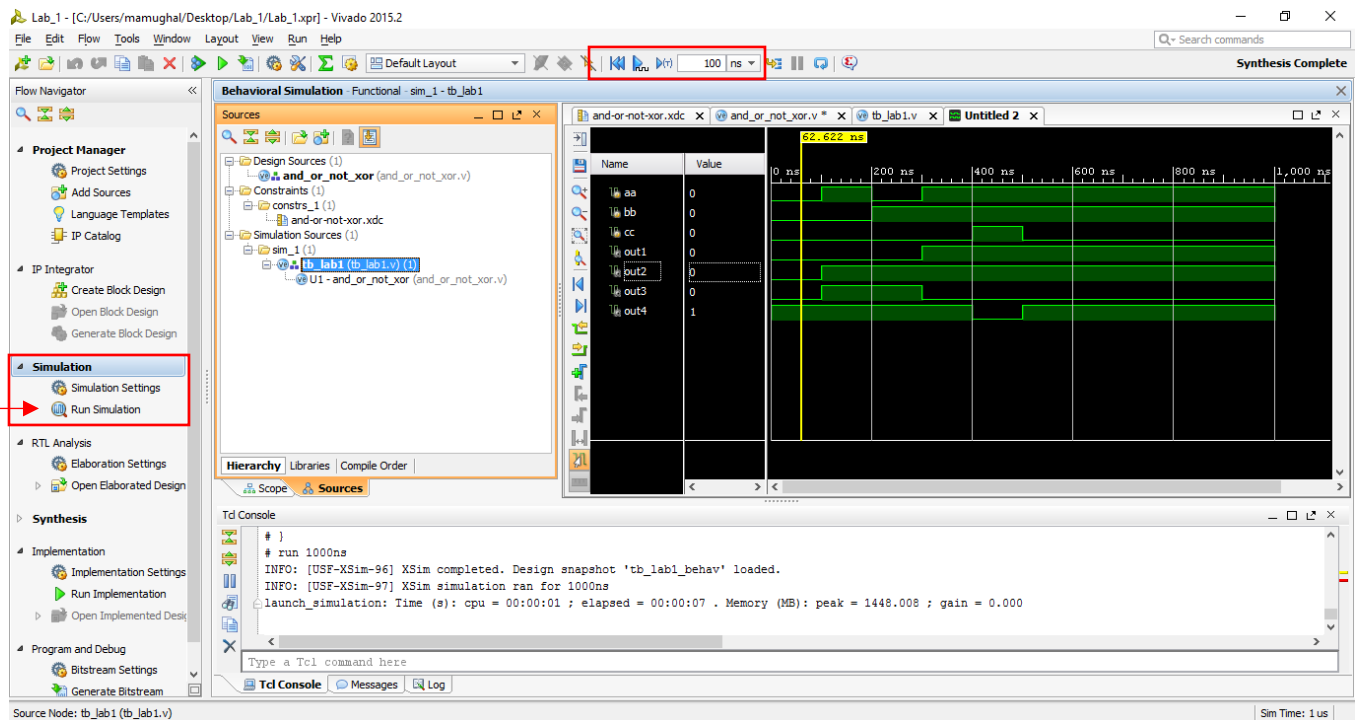
```
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module tb_lab1(
24
25 );
26
27 reg aa,bb,cc;
28 wire out1, out2, out3, out4;
29
30 and_or_not_xor U1(aa,bb,cc,out1,out2,out3,out4);
31
32 initial begin
33     // initialize variables
34     aa = 0;
35     bb = 0;
36     cc = 0;
37     #100; // wait 100 ns for global reset to finish
38     // enter input values here
39     aa = 1;
40     #100; // wait 100 ns
41     aa=0;
42     bb = 1;
43     #100; // wait 100 ns
44     aa = 1;
45     #100; // wait 100 ns
46     cc = 1;
47     #100; // wait 100 ns
48     cc = 0;
49
50     #1000; // wait 1000 ns
51 end
52
53
54
55 endmodule
56
```

15. Afterwards you will see your testbench in bold under sim\_1 with the module and\_or\_not\_xor under it.

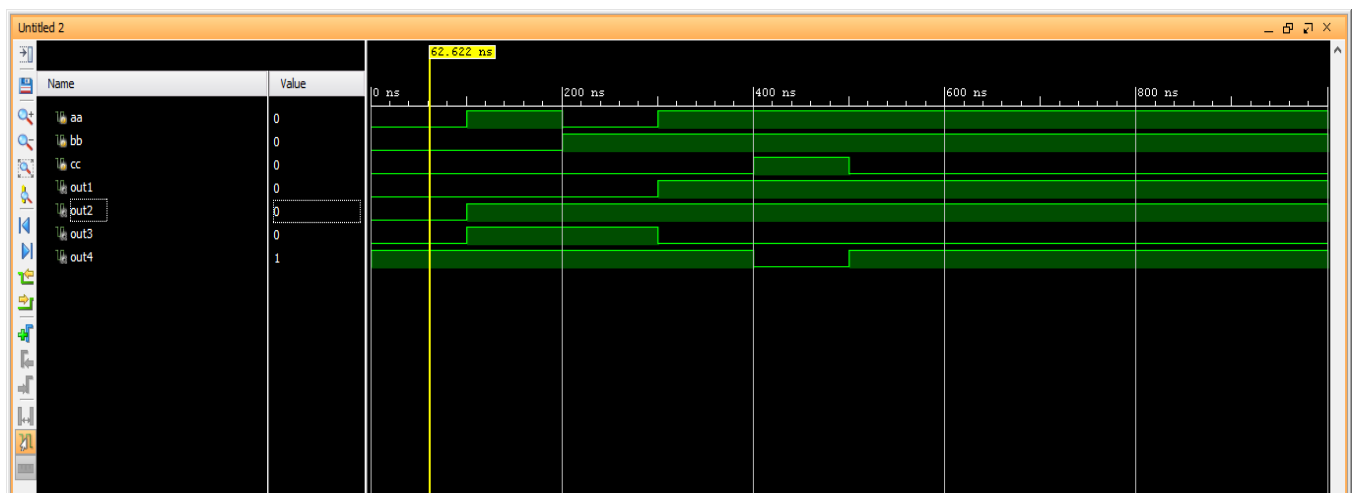


16. Select your testbench file then click on Run Simulation then Run Behavioral Simulation. Vivado will whirl and spin for a second before opening a simulation window in which you

can't see a thing! In the simulation window hit the Run All button then right click in the window and select full view. Now verify the logic for each gate in the timing diagram.



You should now see a timing diagram like this.



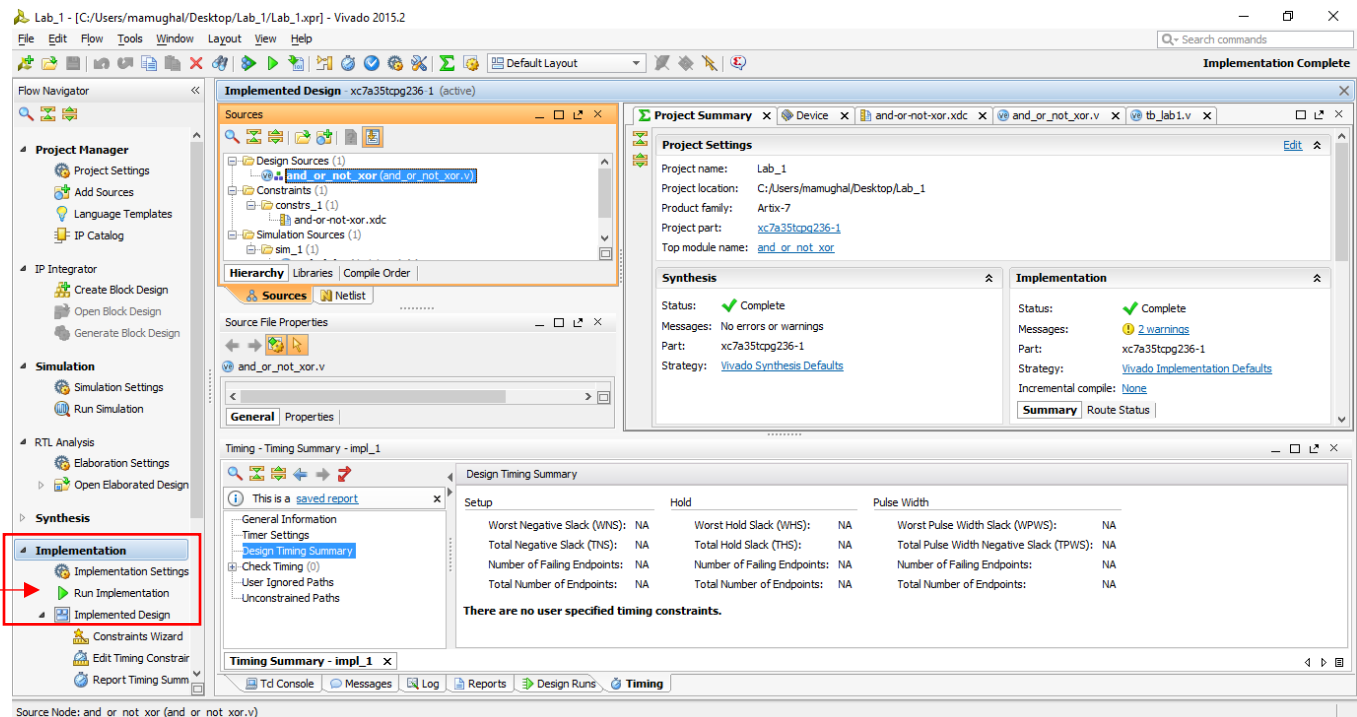
17. Are the outputs of the simulation as you expected them to be?



## Synthesis & Simulation for Synthesis and Timing Verification

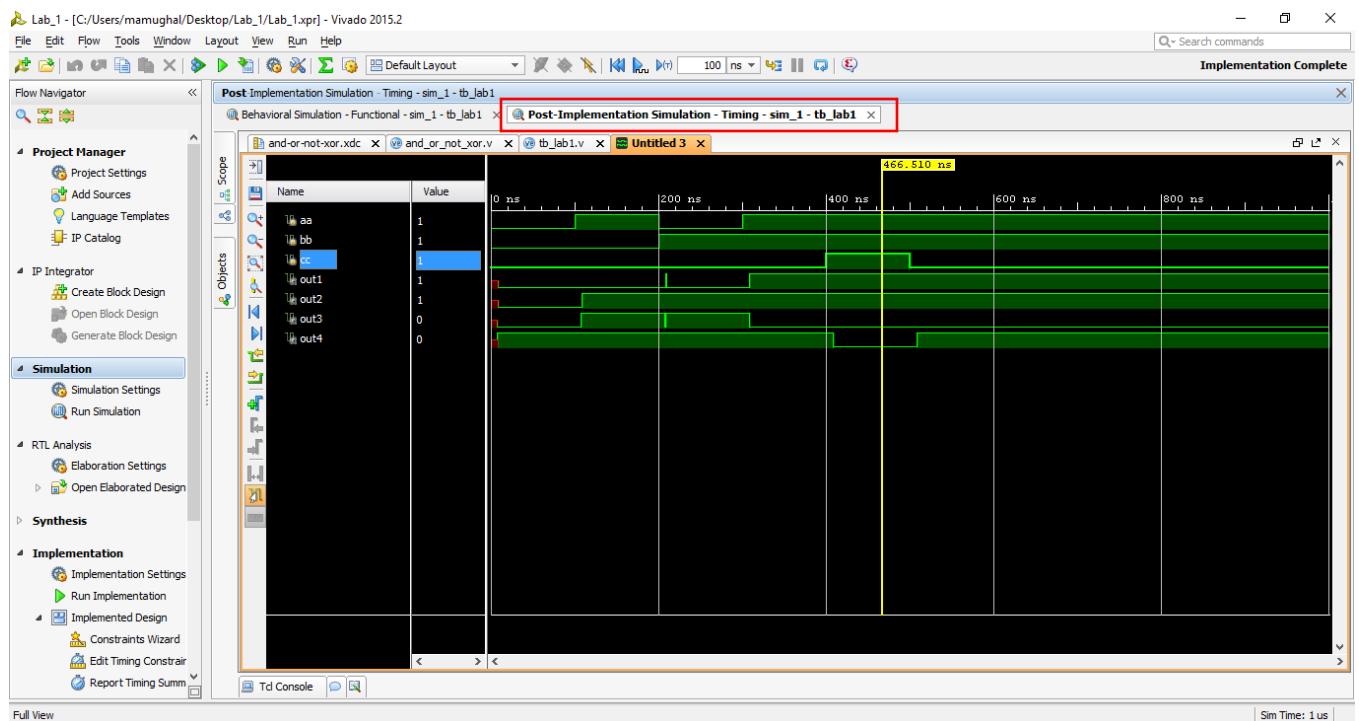
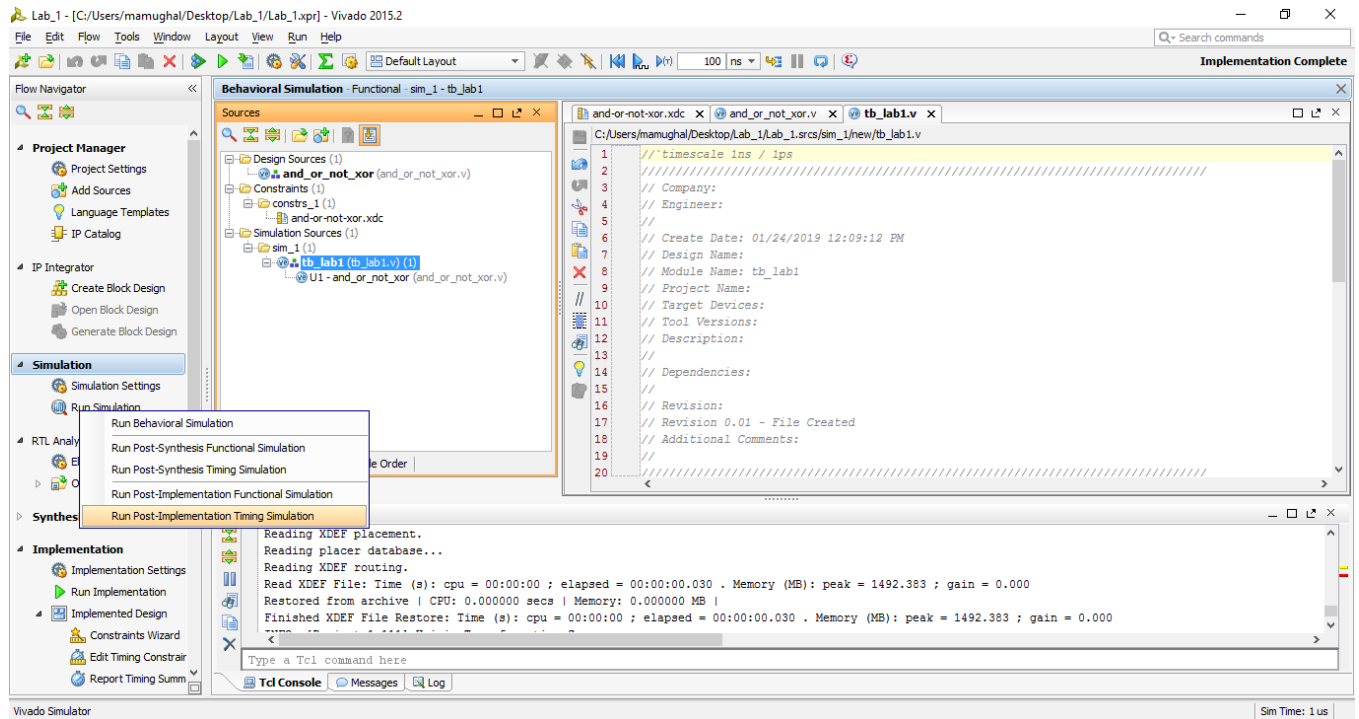
You have completed your design entry and functional verification of your Verilog module without any assumptions about the underlying target. In this section you will synthesize your design targeted for Basys3 FPGA device.

18. Under Design Sources, select your `and_or_not_xor` module then click Run Implementation (takes few seconds). You will/may find there are two warnings after implementation. This is because we have not specified any timing constraints. For this simple combinational circuit we can ignore this.



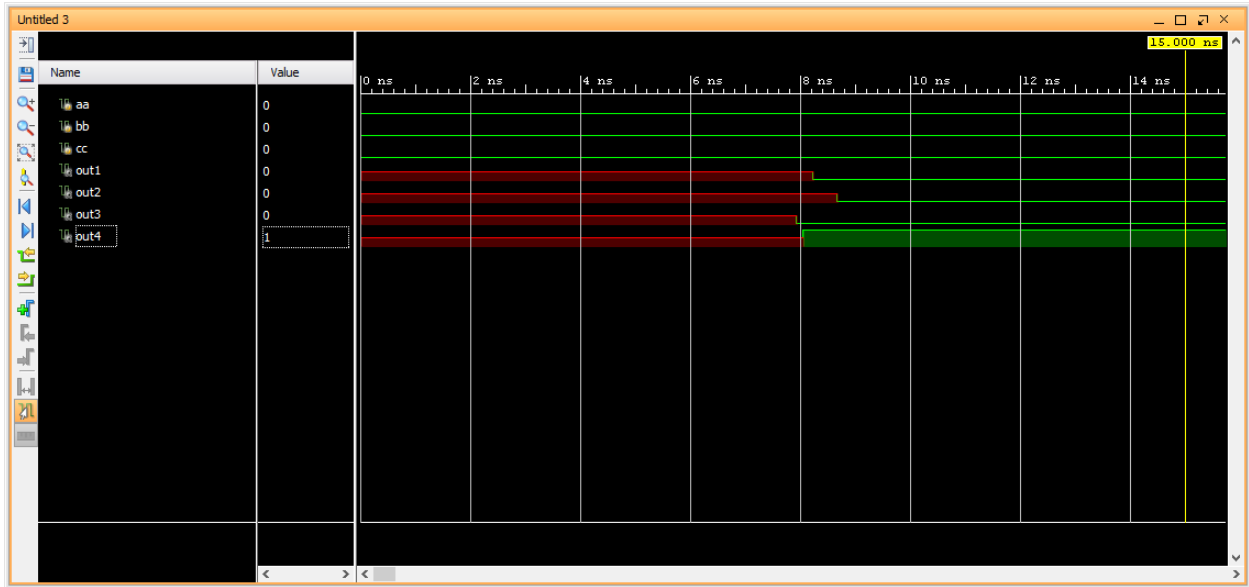
19. Now select your testbench file and then Run Simulation/ Run Post-implementation Timing Simulation. Again, Vivado will go off and simulate and open a window which you can't read. Hit the Run All followed by the Zoom Fit button again.





20. This simulator models the expected propagation delays resulting from the actual lay-out of the gates and pin connections on the FPGA. You can see the propagation delays from the time an input changes until the corresponding change in output occurs. Hit the Zoom + button to see in more detail. What's going on just after 200 ns? Check your testbench

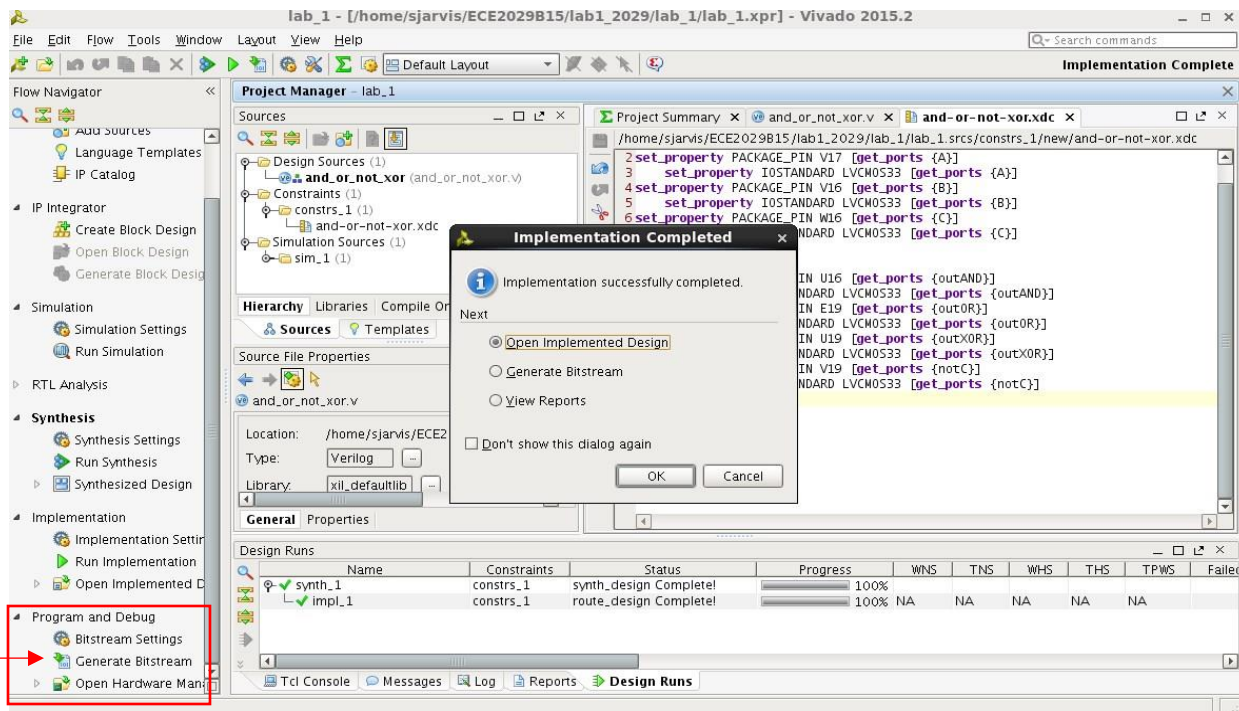
file. Notice that we changed aa=0 and bb=1 without a delay in between. Behaviorally, we wanted those inputs to change values exactly simultaneously. However, in the actual implementation that is not physically possible and both aa and bb must have been equal to 1 for a nanosecond and we see the slight “glitch” in the outputs. If you go back into your test bench and but a #100 after aa=00 and before bb=1 then this “glitch” will not occur in the post-implementation simulation.



21. Use the zoom and the cursor measuring tools on the toolbar to measure the propagation delay of your design as it is actually laid out for implementation on. Are the outputs of the simulation as you expected them to be?

## Implementation and Downloading

22. Under Program and Debug click Generate Bitstream. This generates the downloadable bitstream file (\*.bit).



23. The Next step is to program the FPGA with the bitsream. On the Basys3 board make sure that JP1 is set to the JTAG mode and connect the USB cable to the board and turn on the power.

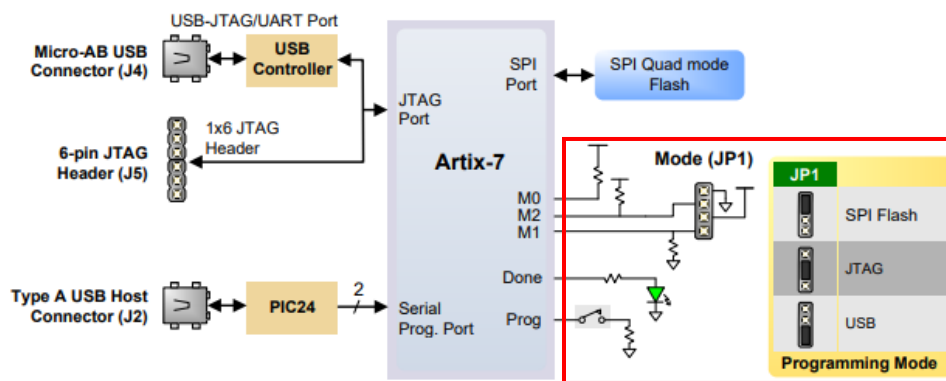
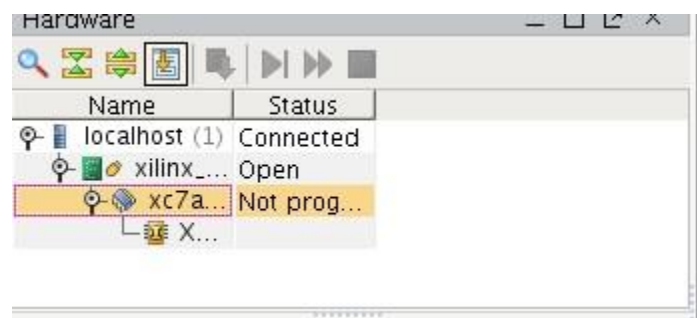


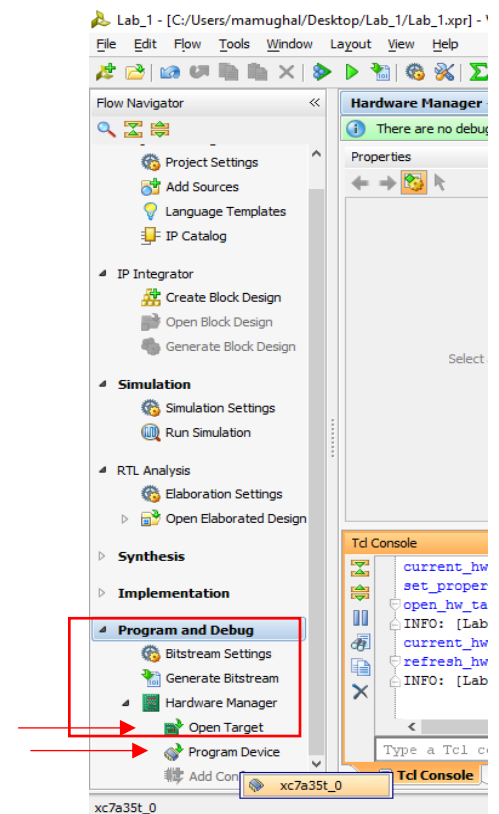
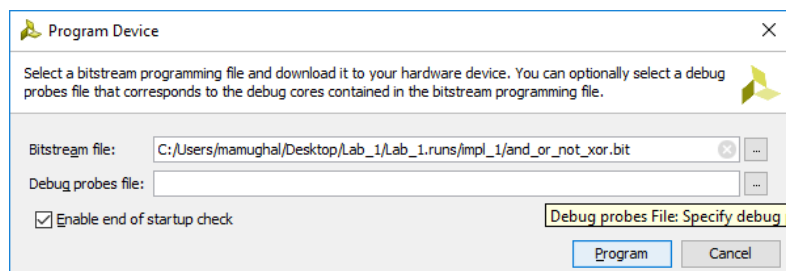
Figure 3. Basys 3 configuration options.

During JTAG programming, a .bit file is transferred from the PC to the FPGA using the onboard Digilent USB-JTAG circuitry (port JP1). You can perform JTAG programming any time after the Basys 3 has been powered on.

24. Select the Hardware Manager in the Project Manager and select Open Target and then Auto Connect. You should see the Xilinx xc7a35t\_0 in the Hardware Window.



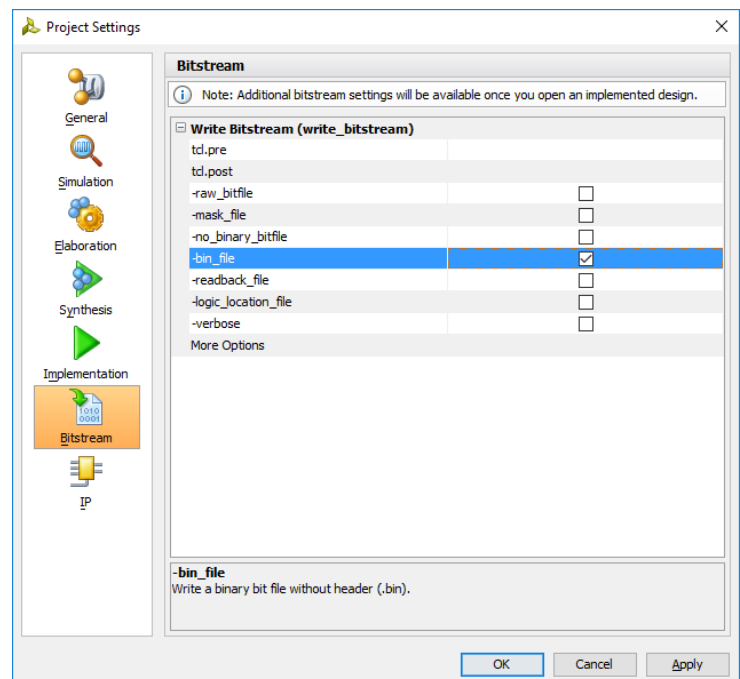
25. Click on Program Device and select the and\_or\_not\_xor.bit bitstream file (automatically filled in). Then select Program (ignore the warning about the missing debug core and the rule violation).

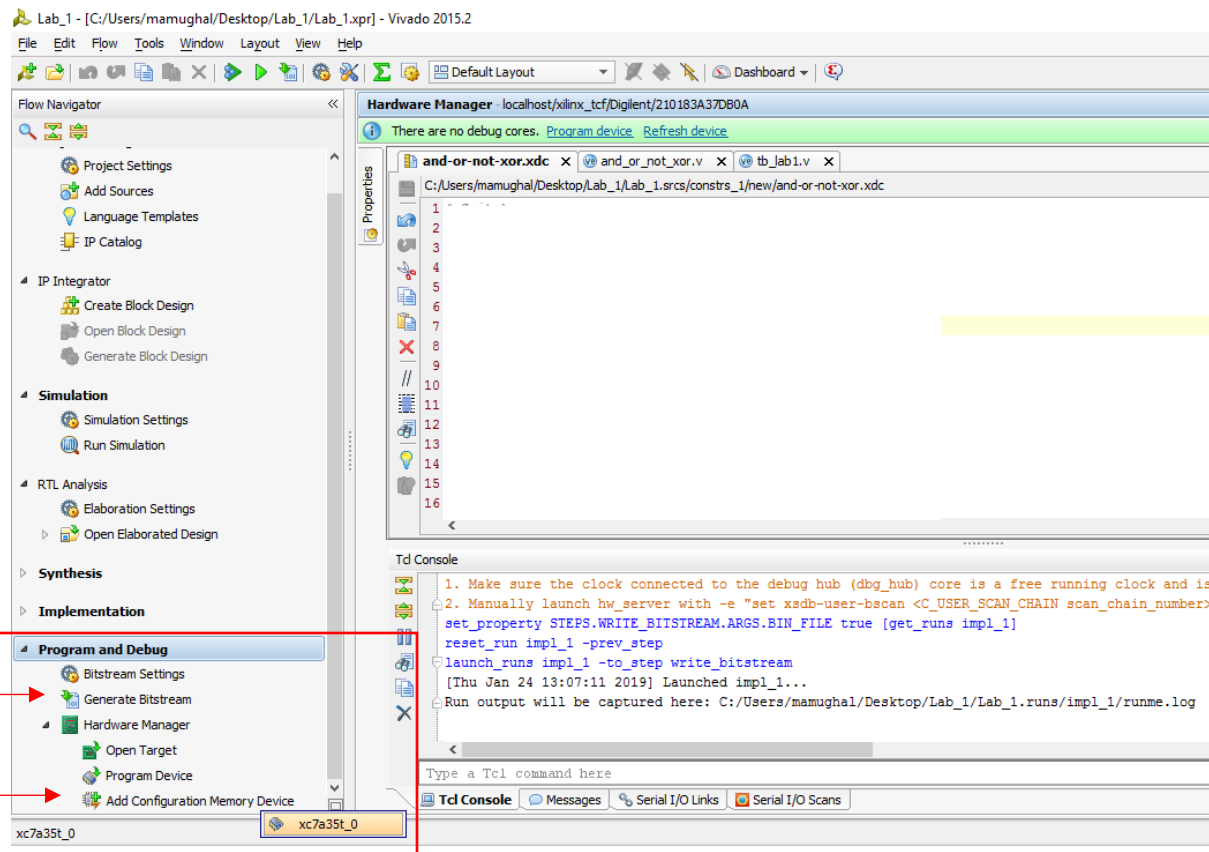


26. Now you can test your design by toggling SW0, SW1 and SW2 and see the various outputs on LD3-LD0. Try different combination of inputs and make sure that your design works properly.

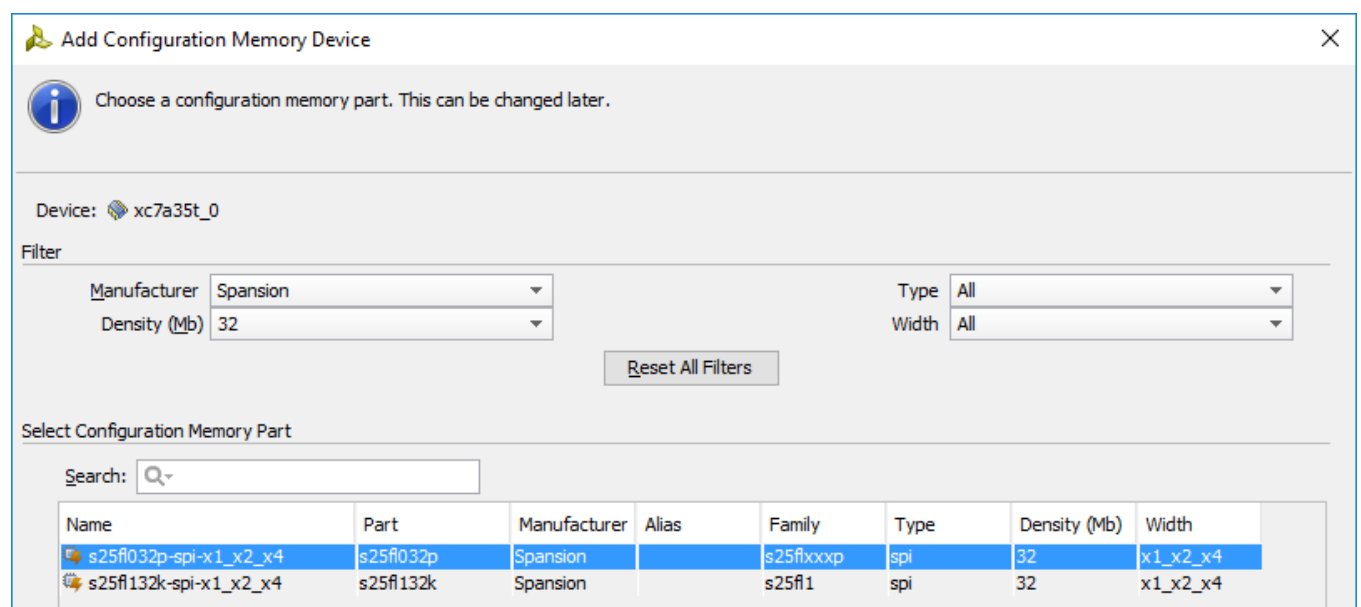
## Programming the Serial Flash

27. Close/minimize the Hardware manager to go back to the Project Manager window.
28. The FPGA is a volatile device so the bit file will be lost after power is removed. We can also choose to write the program to the QSPI serial flash on the Basys3 board. Then the bit file will load from the flash on power up.
29. First we need to create a *bin* file to be able to program the serial flash.
30. Click on the Bitstream Settings in the Program and Debug section of the Project manager and Select the *bin\_file* option.
31. Click OK then click on Generate Bitstream. After generation, if you look in the lab\_1 => lab\_1.runs => impl\_1 directory you will see a .bit file and a .bin file.

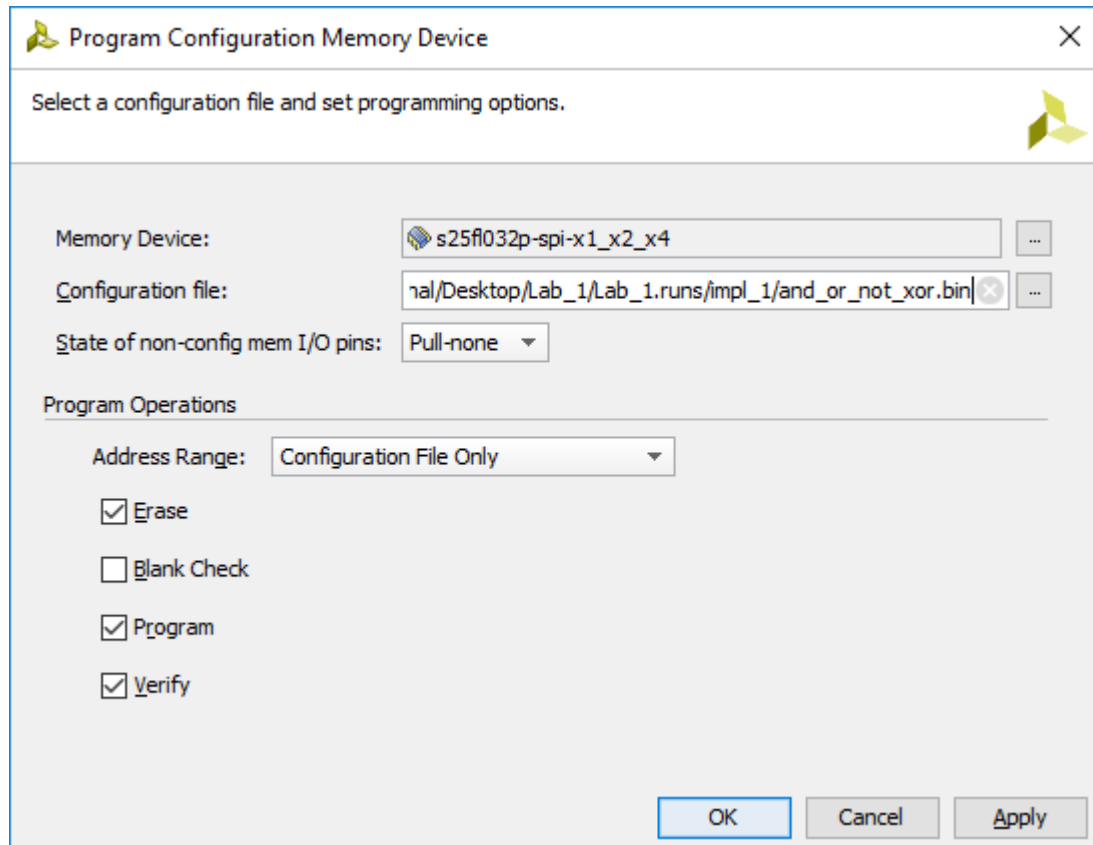




32. Use the Hardware Manager to connect to the Basys3 board then right-mouse click on the FPGA and select Add Configuration Memory Device. We now need to select the serial flash device that is on the Basys3 board. Select Spansion as the Manufacturer, then select the 32Mb device and OK. Then click OK for the pop-up.



33. Select the Configuration File (and\_or\_not\_xor.bin) in the impl\_1 directory and click OK. Note: this will erase any existing design in the QSPI flash (including the configuration file that shipped with the Basys3 board). The QSPI Flash will then be programmed with the and\_or\_not\_xor.bin file. Once programmed, you can power off the Basys3 board and change the JP1 jumper mode from JTAG to QSPI (see step 23). Power back on the board and after a few seconds the Done LED will turn on indicating that your design has been automatically loaded into the FPGA from the serial flash. You can verify your and\_or\_not\_xor design by moving the slider switches and observing the leds as before.



## Additional Experiments

Now that you know how to pass signals from a switch on the Basys 3 board to LEDs on the board, you know all the basics necessary procedures to build logic circuits on FPGA. Given this newfound knowledge, you are to implement and test the following logic circuits:

Input A	Input B	Input C	Output	Function
SW0	SW1	N/A	LD6	NAND
SW0	SW1	N/A	LD7	NOR
SW0	SW1	N/A	LD8	XNOR

**In order to get TA sign-off, add the above functionalities to your existing design and create the appropriate simulations to verify your design, and then load the project onto the Basys 3 board.**

### Summary

- You should now be able to describe a Design flow for digital systems.
- You learned how to use Vivado Design Studio tools to enter a design.
- Synthesize and download your implementation to a target board
- You learned how to use the simulator to create stimulus signals, verify functional and timing behavior of a digital system.