EECE2323	Digital Logic Design Lab	Lab 0

XOR gate

1 Objective

The purpose of this lab experiment is to get you familiar with the tools and hardware development kit, PYNQ, that you will be using for all EECE2323 labs this semester. The first experiment is a tutorial. By stepping through all the instructions, you will learn techniques and tools you will use throughout the course.

2 Conventions

The following conventions are used throughout all lab manuals.

Text Style	Use	Example
boldface	Actions you need to take	**Draw a wire from A to B.**
	Names of programs or tools	Open Vivado Simulator, **XSIM**, and enter...
italics	Menus and menu items	Select *File* → *Quit* to exit.
	Buttons	Click *Finish* to continue.
monospace	User interface objects	Enter a name into the Net Name field.
	Names of symbols	Insert two nand2 gates.
	Text you enter into fields	Call your element xorgate.
	Items you pick from a list	Choose Create Block Design from the list.

You should read through the entire lab manual so that you are familiar with the concepts and the procedure to be followed. Additional information that is important to understanding the experiment may not be printed in boldface, but you should read it anyway.

3 Overview

In this lab you will implement an EXCLUSIVE-OR (XOR) gate out of NAND gates using the tools available in the **Xilinx Vivado** software package. The XOR gate is a very useful element which is used in many applications such as encryption and decryption. This XOR gate is only an example.

The logic symbol and truth table for an XOR gate are shown in Figure 1(a). A schematic showing how to build an XOR out of NAND gates is shown in Figure 1(b). There are several possible ways to build an XOR from NANDs; this one uses the minimum number of gates.

This lab will teach you how to use the **Xilinx Vivado** design tools. The steps taken in this lab manual, as well as the processes and procedures that are described, will be repeated throughout the semester. This assignment will introduce you to these practices.

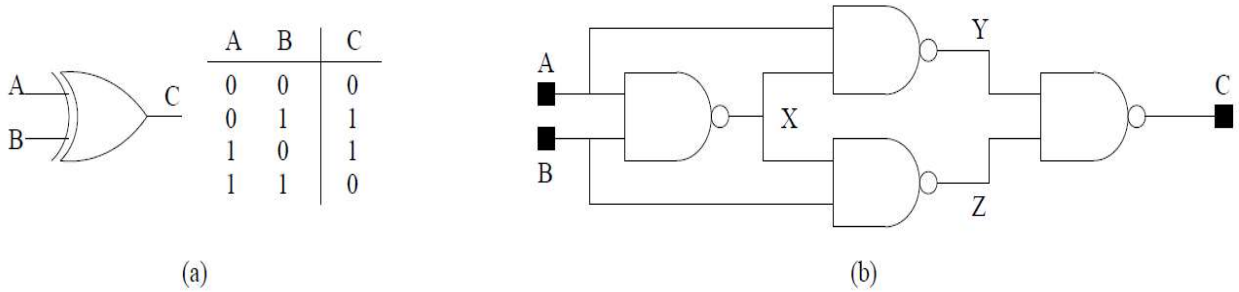

Figure 1: XOR gate symbol and truth table (a), Implementation of XOR using NANDs (b)

First, you will recreate the schematic shown in Figure 1 with **Intellectual Property Integrator (IPI)**. Then, you will create a testbench and simulate your design with Vivado Simulator. *You will include a screen capture of your schematic, testbench, and the completed simulation with your final submission.*

Once the simulation is complete, you will use the **Xilinx Vivado** synthesis and implementation tools to translate your design to a bitstream that can be used to program the Zynq's programmable logic (PL) section on the TUL PYNQ Z2. Finally, you will use **Vivado Hardware Manager** to download the program to the Zynq's PL section (equivalent to an ARTIX-7 Field Programmable Gate Array (FPGA)) and test it in hardware. *Then demonstrate the working hardware to the TAs for the second part of the lab grade. You can upload a short video (~10 seconds) of your demonstration with your report submission.*

For this tutorial lab, your grade will consist of the grade for the simulation and the grade for the hardware. There are no prelab assignments and no lab report for this lab. All other labs will have one or more prelab assignments, one or more simulation/hardware grades, and one lab report grade.

4 XOR Gate

The first and only part of this lab assignment is to create the XOR gate out of NAND gates. Future labs will often have two or more sections of work to be done.

4.1 Prelab

There is no prelab assignment for this part of the lab.

4.2 Entering Your Design

The **Xilinx Vivado** toolkit is accessed through an interface called **Vivado IDE**. All of the design and implementation tools can be accessed, run, and controlled through **Vivado IDE**.

Launch Vivado IDE from VLAB by selecting Vivado from the *Start* menu:

Start → All Programs → Xilinx Design Tools → Vivado 2020.1

4.2.1 Starting a New Project

*After launching the Vivado tool, click on the **Create Project** to open new project wizard. Then choose **lab0** as the project name. From the "Choose Project Location menu" browse to a file on your Z; drive. Create a subdirectory for your project. Select **RTL Project** but do not specify sources now. Finally, in the Default Part tab and in the Specify section, choose **xc7z020clg400-1** under the part category. By pressing Next, you will see the project summary. Press Finish to close the wizard.*

When you have created a new project your **Vivado IDE** window should look like Figure 2. Lab manual directions will often refer to the various "panes" of the **Vivado IDE** window, as labeled here.

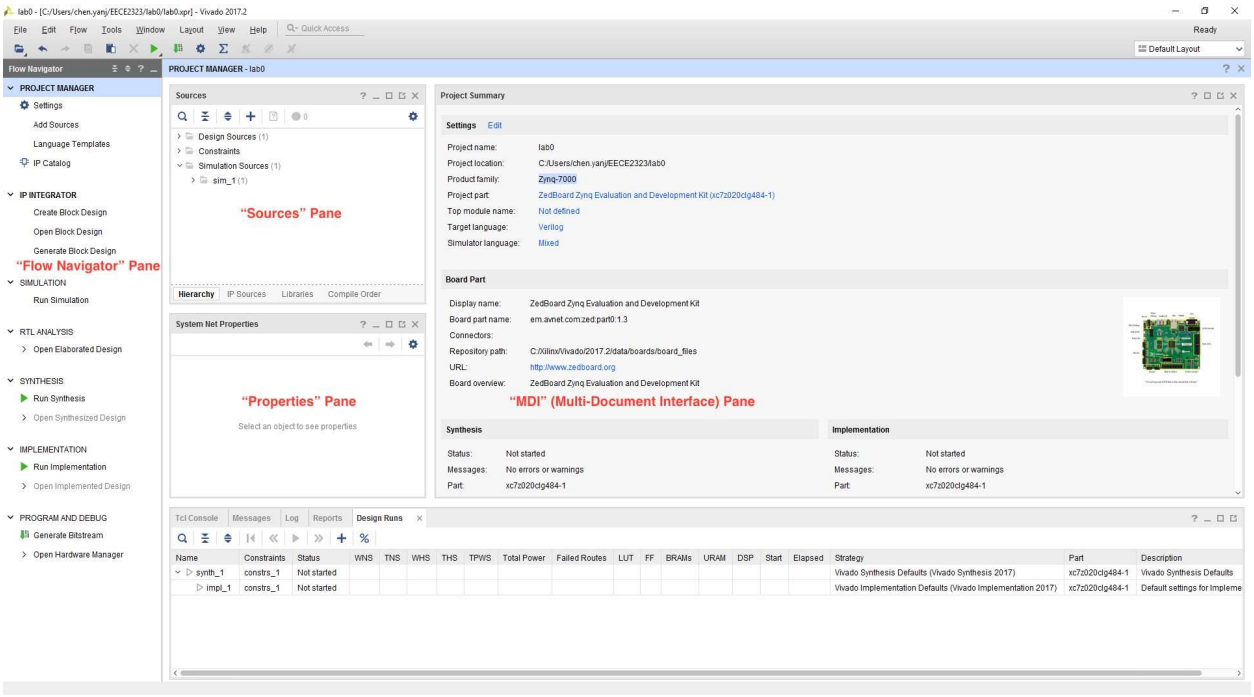

Figure 2: Vivado IDE Main Window

4.2.2 Creating Block Design

The first step in this lab assignment is to recreate the schematic design for the XOR gate using **IPI**. First, we must create a new Block Design to add to our project, then we can add the desired IPs in the canvas.

In the Flow pane select *Create Block Design* **from the IP Integrator menu. In the** Design Name **field, enter** xorgate**. Do not change the location. Then click the** *Ok* **button. The "Diagram" canvas appears.**

To see the project summary, click on the Window tab of the Vivado IDE, choose Project Summary. You can also open the Project Summary tab by clicking on the $\sum$ icon in the Vivado IDE toolbar.

Now there are two tabs, one for the block design you just created (`xorgate.bd`) and one for the "Project Summary". The Project Summary contains information about the project and allows access

to the log files that are generated at different points in the process. Now click on the diagram tab to go back to the Block Design canvas.

You should see an empty canvas in the MDI pane, as shown in Figure 3. If not, double-click on the entry for your block design file (xorgate.bd) in the Sources pane.

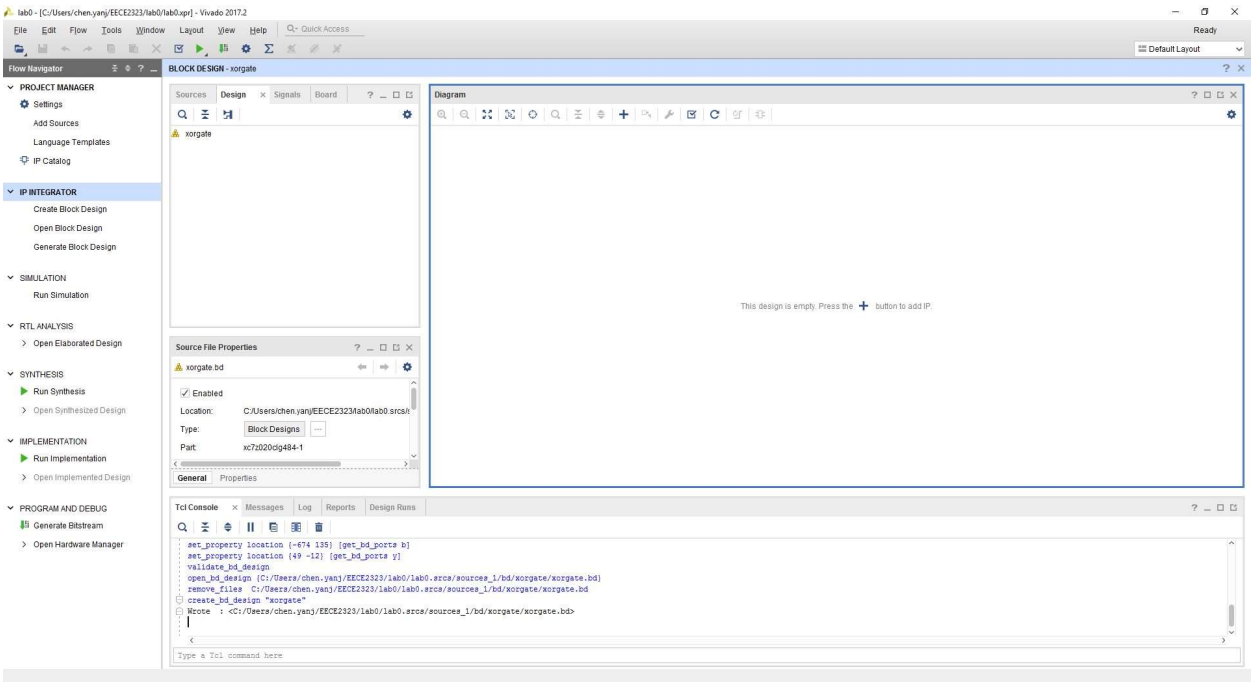

Figure 3: IPI empty canvas

Diagrams can be "floated" into their own window. You may find it easier to work with a Block Design window that takes up the entire screen. To float a document from the MDI pane, right-click on its tab at the top of the MDI pane and select *Float*. Once floated, you can re-dock the diagram in the MDI pane by selecting *Dock Window* from the top right of the window.

4.2.3 Entering a Design with IPI

We will now enter the schematic shown in Figure 1(b). Before that, you should add the Basic Elements repository to your project.

Select Settings **under the Project Manager section in the Flow Navigator pane, Figure 4. Then in the IP section, Figure 5, select add repository and point to the** basic _elements **folder (.zip version of this folder is available for download on Canvas's course webpage) which includes Basic Elements repository. To make the basic elements visible in the IP catalog, press** Refresh All**. You should add four 2-input NAND gates to your canvas. For that click on the canvas, then press Ctrl+I to open IP catalog. In the IP catalog these gates are called** XUP 2-input NAND**, drag and drop four of them into your canvas. Click on the canvas or press ESC to close the IP catalog. Add wires by pointing to the gate ports and dragging the pencil toward the target port,**

connecting the NAND gates together, as shown in Figure 6. If you notice the possible connections are shown with a green check mark beside the interface.

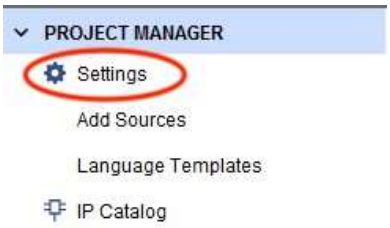

Figure 4: Project Settings (Circled)

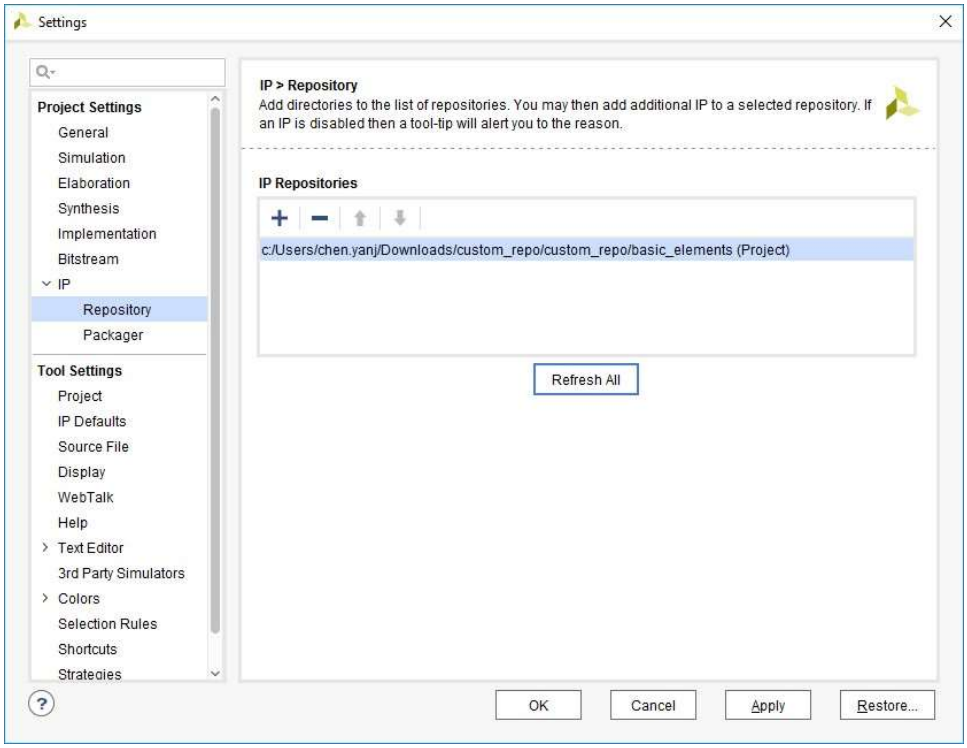

Figure 5: Project Settings, IP Section

Now you will indicate the top level ports (inputs and outputs) of your design, and name the signals. Right click on the ports that you want to make them design's I/Os, then select *Create Port* **from the menu or simply press Ctrl+K after left clicking on the port. Then in the Properties pane name the NAND gates as nand2_0 to 3 after selecting them. Similarly, you can choose the names of the circuit's inputs** *a* **and** *b*, **and the output** *c*. **You do not need to name the intermediate wires, but you can if you wish. When you are done your block design should look like Figure 6.**

It is generally a good idea at this point to run the **DRC** (Design Rules Check) or **Validate Design** to ensure that your circuit is properly constructed. Click on the validate design button (Figure 7) on the toolbar or press F6 after selecting the diagram tab. If any errors or warnings are displayed

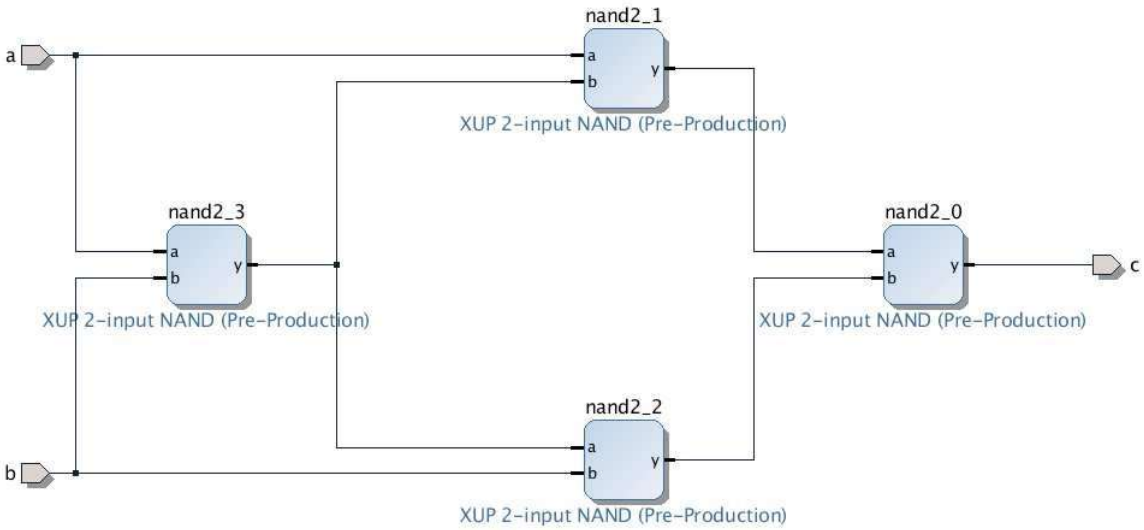

Figure 6: Completed XOR Block Design

in the Transcript pane, ask the TA for assistance as the error messages are sometimes difficult to interpret.

Figure 7: Validate Design button (circled)

4.2.4 Creating A Hierarchy From Your Block Design

Usually, a Block Design contains multiple IPs which could be packed together to create a higher level hierarchy or IP. In our design we can use this feature to pack all NAND gates together. To do so, please select all gates, then right click on one of them and choose Create Hierarchy **from the list. After that, change the name of the hierarchy to xor_2 by modifying the** Name **field in the Properties pane, denoting 2 input XOR logic gate. Please see Figure 8.**

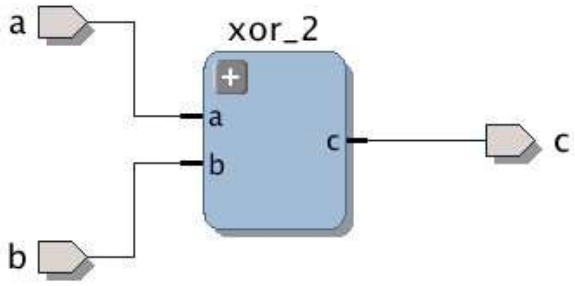

Figure 8: XOR Gate Hierarchy

	Time (ns)

Signal	0	200	300	400	550
a	0	1	0	1	0
b	0	0	1	1	0
c (after a delay)	0	1	1	0	0

Table 1: XOR Gate Test Vectors

Once you are finished, save your Block Design by clicking on the Save button on the toolbar, pressing Ctrl-S, or selecting *File → Save Block Design* **from the menu.**

4.3 Simulating Your Design

Now that you have finished constructing the circuit you must test it. To test designs, you use a software simulator which creates a computer simulation of the block design. To do this, you must specify the *test vectors*, or set of input values, that will be used to test your circuit. In this section you will use the **simulator** to simulate the circuit's response to those test vectors.

In this lab we will tell you what test vectors you should use. Since this is a very simple circuit with only two inputs, you will test all possible input combinations ($2^2 = 4$ possibilities). In some future labs it will be up to you to choose an appropriate set of vectors that demonstrates that your circuit works correctly.

4.3.1 Creating a Testbench with Vivado

The first step in a computer simulation is to create a testbench. A testbench may include three parts: Stimulus, Unit Under Test and Monitor. For our case, the testbench is a verilog module with a synthetic set of test vectors as the stimulus and xor_2 design as the unit under test. The Monitor section is integrated in the testbench module.

For us to run the simulation of our design we need to have the top-level HDL code of **xorgate** block design. Therefore, in the Sources pane right click on the **xorgate.bd** and choose *Create HDL Wrapper.* In the menu that shows-up choose *Let Vivado manage wrapper and auto-update*. Then, you will see the verilog wrapper in the Sources pane.

To create a testbench module click on the Add Sources **under the Project Manager section in the Flow Navigator pane and choose** Add or Create Simulation Sources**. In the next page click on** Create File **and create a testbench source file called** xorgate _tb**.**
Make sure that the Include all design sources for simulation **option is selected. Use Figure 9 to write your testbench code in the source file you just created. Find the testbench file in the Sources pane. In fact you will use the test vectors in Table 1 to test your** XOR **gate. Please note that you are using the HDL wrapper the you created for the xorgate block design as the UUT for your testbench.**

In the testbench code notice how the input values change at 200ns, 300ns, 400ns, and 550ns. This is the way that you will create your own test vectors in future labs. Also, If you look at the testbench code you will see the `timescale 1ns \ 1ps **statement at**

```
1  `timescale 1ns / 1ps
2
3  module xorgate_tb();
4
5      //Inputs
6      reg a=1'b0;
7      reg b=1'b0;
8      //Output
9      wire c;
10
11     //Instantiate the Unit Under Test (UUT)
12     xorgate_wrapper UUT
13        (.a(a),
14         .b(b),
15         .c(c)
16         );
17
18     //Initialize Inputs (Stimulus)
19     initial
20         begin
21
22             #200; // ------------------ Current Time: 200ns
23             a=1'b1;
24
25             #100; // ------------------ Current Time: 300ns
26             a=1'b0;
27             b=1'b1;
28
29             #100; // ------------------ Current Time: 400ns
30             a=1'b1;
31
32             #150; // ------------------ Current Time: 550ns
33             a=1'b0;
34             b=1'b0;
35
36         end
37
38
39 endmodule
```

Figure 9: XOR Gate Testbench

the top of the source file. It means that any delay (numbers preceding with #) in the code will have a unit of 1ns second and simulation will run with the precision of 1ps.

Once you are finished, save your testbench by clicking the Save button on the toolbar, pressing Ctrl-S, or selecting *File → Save* **from the menu.**

4.3.2 Running a Testbench with Simulator

To run the simulation press Run Simulation **under the Simulation section in the Flow Navigator pane. Then pick** Behavioral Simulation **from the list. You will see the Behavioral Simulation window after compilation of source files. The first step to run the simulation is to set the simulation time. We want to run the simulation for 630ns, so write 630 in the time box and make sure the unit is set to ns, see Figure 10. Then run the simulation until then by first clicking on** Restart **button and then on** Run for … **button.**

Figure 10: Simulation Toolbar: Restart button, Run all button, Run for button, time box, units list (Highlighted with a red box) and Relaunch Simulation button (Circled)

You have to compare the results that appear in your **Simulator** waveform with the results shown for the signal *c* in Table 1. Your waveform should resemble Figure 11.

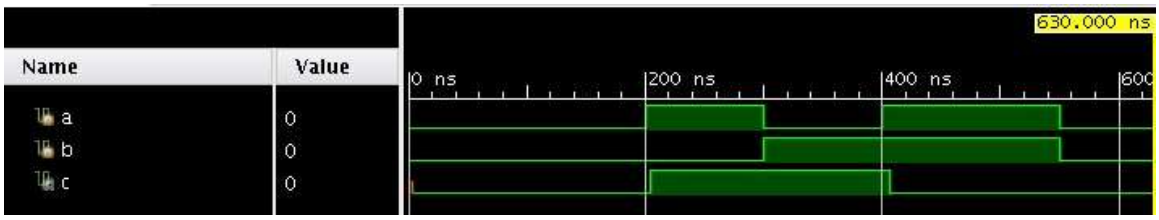

Figure 11: XSIM Waveform for XOR Gate

Let's take a closer look at the waveform. If you zoom in the waveform you will see that the response of the circuit to a change in inputs show up after some delay, see Figure 12 and Figure 13. This is due to the Propagation Delay **or** Gate Delay **of the circuit. Investigate the circuit paths from inputs to output and find out why there is a difference in the delay amount in Figure 12 and Figure 13. You can find the delay of each gate by double clicking on its IP in the block design. Open the block design by clicking on** Open Block Design **under the IP Integrator section in the Flow Navigator pane. You can come back to the Simulation window by clicking on the Simulation in the Flow Navigator pane. Now try to modify the delay of NAND gates and observe the effect on the circuit's gate delay. Change the delay of all of the NAND gates to 1 by double clicking on each of them and changing 3 to 1. Then Save the block design and regenerate the *HDL Wrapper* file:**

In the Sources pane right click on the **xorgate.bd** and choose *Create HDL Wrapper.* In the menu that shows-up choose *Let Vivado manage wrapper and auto-update*.

Go back the simulation window and click on the Relaunch Simulation **button form the simulation toolbar. After the compilation is finished run the simulation again until 630ns. The Resulting waveform should look like Figure 14 and Figure 15**.

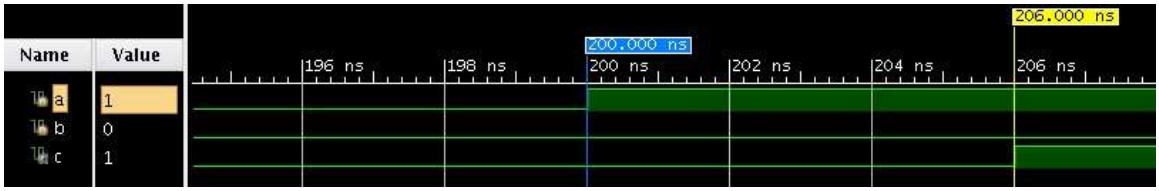

Figure 12: Propagation Delay of 6ns for a change in input "a" at t=200ns

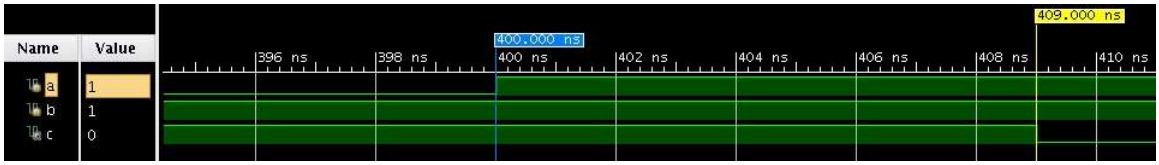

Figure 13: Propagation Delay of 9ns for a change in input "a" at t=400ns

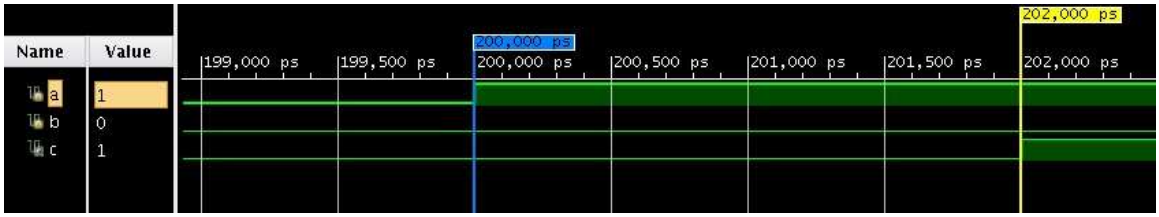

Figure 14: Propagation Delay of 2ns for a change in input "a" at t=200ns

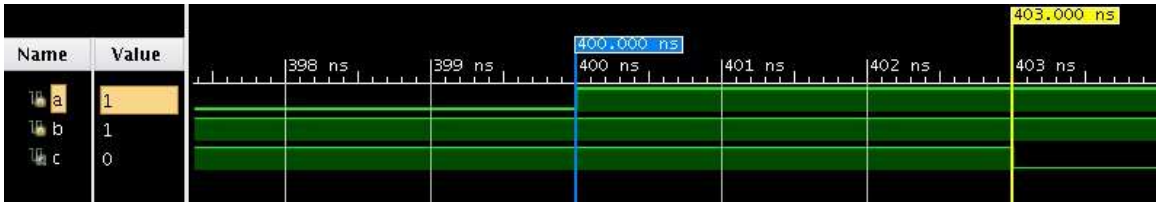

Figure 15: Propagation Delay of 3ns for a change in input "a" at t=400ns

4.4 Testing in Hardware

The last step is to test your design in hardware. To do this, you need to create a new source file that indicates how the design IOs connect to the TUL ZYNQ Z2 hardware resources. In Vivado this file is called constraints file or XDC file. The *synthesis and implementation* tools will then create a *bitstream* that can be used to program the FPGA. Finally we will connect the FPGA board to the PC and download the program.

4.4.1 Creating a Constraints File

To create a constraint or XDC file, click on Add Sources **under the Project Manager section in the Flow Navigator pane, then choose** Add or Create Constraints **and by clicking on** Create File**, create a constraint file with name of** ''xorgate''**, then press** Finish**. The constraint file is in the design sources pane. You need to double click on the name for the file to come up.**

Now we will connect the XOR gate's inputs to switches on the FPGA board, and the output to an LED. Specifying the LOC attribute for the inputs attaches them to pins on the FPGA chip; those pins are connected to devices on the FPGA board.

Follow Figure 16 to write your physical constraints in the XDC file you just created. You only need to copy the lines that are uncommented. The physical constraints are written based on Table 2. Remember to save the constraints file after you have finished.

Port	a	b	c
Device	SW0	SW1	LED0
LOC	M20	M19	R14

Table 2: LOC Attributes for XOR Gate I/Os

Investigate the XDC file syntax and find out what each line means.

```
## Switches
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports a]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports b]

## Buttons
#set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports a]
#set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports b]
#set_property -dict {PACKAGE_PIN L20 IOSTANDARD LVCMOS33} [get_ports {btns_4bits_tri_i[2]}]
#set_property -dict {PACKAGE_PIN L19 IOSTANDARD LVCMOS33} [get_ports {btns_4bits_tri_i[3]}]

## LEDs
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports c]
#set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[1]}]
#set_property -dict {PACKAGE_PIN N16 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[2]}]
#set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[3]}]

### RGBLEDs
#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { rgbleds_6bits_tri_o[0] }];
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { rgbleds_6bits_tri_o[1] }];
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { rgbleds_6bits_tri_o[2] }];
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { rgbleds_6bits_tri_o[3] }];
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { rgbleds_6bits_tri_o[4] }];
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { rgbleds_6bits_tri_o[5] }];
```

Figure 16: XOR Gate XDC file

4.4.2 Implementing a Design with Vivado Synthesizer

The process of transforming your design specification (i.e. Block Design files) into a *bitstream* that can be used to program the FPGA is called *synthesis*. **Vivado IDE** includes the **Vivado Synthesis** toolkit to perform synthesis on your design. Since synthesis is specific to a particular FPGA, it is critical that your project be designed for the right part. In this case we are using the xc7z020clg400-1 chip, which you should have set when you created a new project. If this part number does not appear in Project Settings -> General -> Project Device under the Project Manager in the Flow Navigator pane, ask a TA to help you fix it before going on.

Click on Generate Bitstream **under the Program and Debug section in the Flow Navigator pane to run the synthesis process. Vivado IDE will run the Synthesis and Implementation processes automatically**. At the top right of your screen there is a scroll bar that shows progress; you can see diagnostic output scroll past in the Transcript pane. If any errors occur they will be marked with a red circle with a cross in middle. Warnings are marked with a yellow triangle with an exclamation point. If you see errors or in your output, ask the TA for help. Some warnings can be ignored some not. Processes that complete successfully will be marked with a green circle with a check mark inside.

4.4.3 Programming the FPGA with Hardware Manager

To program the Zynq's PL section you will use the Vivado Hardware Manager. Make sure to plug the micro-USB into the board next to the power cable, as shown in Figure 17. After the bitstream is generated turn on the board and open the Hardware Manager either by clicking on the option that will pop-up at the end of bitstream generation or by clicking on Open

Hardware Manager **under the Program and Debug section in the Flow Navigator pane. Ensure that your board is connected and powered, and that the** Future Devices TUL **is selected under the** "*Connect USB Device*" **menu at the top Desktop menus. In the Hardware Manager toolbar click on the** Open target **the choose** Auto Connect **from the list. In the same toolbar click on the** Program device**. Keep the settings unchanged in the window and press** Program**. Now you are ready to test your design in hardware.**

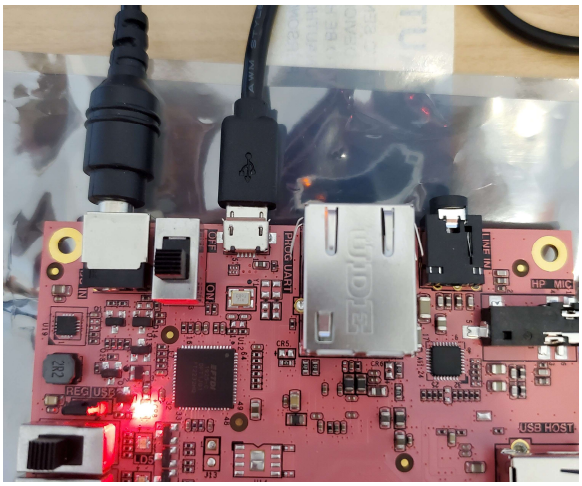

Figure 17: Micro-usb programming cable connected to the top-side of the board

4.4.4 Testing your Design

Table 3 below shows which devices on the prototype board (switches, buttons) are connected to the inputs of your design. This table will appear in every lab manual to show you how to test your design. Also recall Table 2 on page 10, which showed similar information.

For this design, switches 0 and 1 are connected to the inputs, and LED 0 (again, the right-most light) is connected to the output. Using the switches, run the same test vectors that you simulated (see Table 1 and Figure 18). Verify that the LED turns off when both switches are in the same position and turns on when the switches are in different positions.

BTN1	BTN0	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
–	–	–	–	–	–	–	–	b	a

Table 3: Hardware inputs

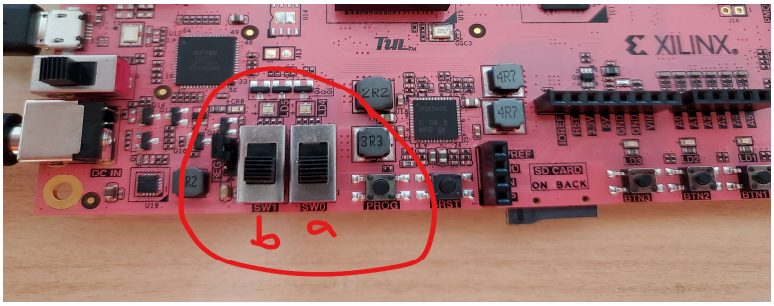

Figure 18: TUL PYNQ Z2 Switches