EE 121 October 1, 2002
Digital Design Laboratory Handout #3

**Laboratory Assignment #1**
**Introduction to Xilinx Foundation Design Software**
Due date: Thursday, October 10, at the beginning of class
*To be completed individually*

## 1. Objectives

- Become familiar with the Xilinx Foundation software package:
  - Project Manager: creates/manages projects
  - Schematic Editor
  - Logic Simulator: graphical schematic simulator
- Create schematics and macros for a new project
- Simulate your schematics

## 2. Create new project

Install Xilinx Foundation software on your home PC or locate the computer cluster in Packard 128. You will need a class computer account and an SUID with access permission to use the computers in Packard 128.

Launch Xilinx Foundation Project Manager. Select **`Create a New Project`** and hit OK. Name the new project **`Lab1`**, select flow to be schematic, choose Spartan2 from the left pop-up menu, and choose 2S100TQ144 from the middle pop-up menu. The Xilinx Spartan 2S100TQ144 programmable logic device will be used throughout EE121, and so all projects should be of this type. Finally, choose a directory on drive **`Z:`** in which to store the project files.

Once the new project has been created and loaded, you will see a list of the project files on the left and a set of buttons on the right. Launch the schematic editor by pressing the schematic editor button in the design entry group on the right. The schematic editor contains a drawing region in the center and buttons on the top and left. The buttons on the left are for selecting and drawing.

## 3. XOR gate

Create an **XOR** gate using Figure 1 as a guide.

### 3. Creating an XOR Gate

First add the components that will be needed (two **INV** gates, two **AND2** gates, and one **OR2** gate). To add a component, first press the **`Symbols toolbox`** button on the left. This brings up a new window containing a list of available components. Next choose the desired component from the list, place the component in the drawing area by clicking in the drawing area, and then right click and choose **`Select/Drag Mode`** when done. Do this for all five needed gates.

After adding all the components, the next step is to add the inputs and outputs. The **XOR** gate has inputs, **X** and **Y**, and one output, **Z**. To add an input/output, press the **`Hierarchy`**

**connector** button, name the terminal, and choose a terminal type, i.e., input or output. Do this for both inputs and the output.

Once the components and inputs/outputs are in place, the next step is to add wires. To add a wire, press the **Draw Wires** button, then in the drawing area click on one of the two terminals you want to connect with a wire and then click on the other. You can use the **Select and Dra'** tool (the arrow button on the left) to move and stretch wire segments. Using Figure 1, wire the components of the **XOR** gate.

Finally, label the internal nodes of the **XOR** gate. To label an internal node, double click on the wire and then give the node a name. Do this for the four internal nodes; name them **X_L**, **Y_L**, **XNY**, and **XYN** as in Figure 1. **Always label the internal nodes of a schematic.**

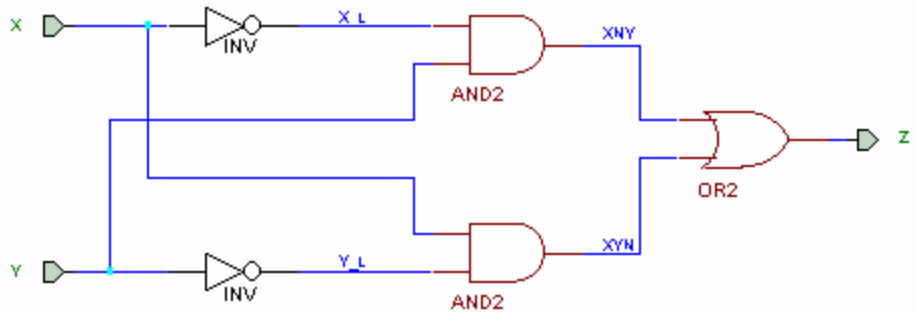


**Figure 1**. XOR Gate Schematic.

Save the schematic by choosing **Save As** from the File menu and naming the file **XORgate.sch**. Note that if you just choose **Save** or press Ctl-S the first time you save your file, Foundation will assign the file a name automatically, and the automatic name probably won't be very revealing. Print the schematic. You can adjust the page size of your schematic by choosing **File:Page Setup** and then selecting a sheet format. Choose a sheet size that is just large enough for your schematic so that the printout will be readable and not a printout of a very large, mostly unused sheet.

## 3.1. Simulating the XOR gate

To verify that the **XOR** gate functions properly, you must simulate it. To launch the simulator, press the **Simulator** button in the top menu bar of the schematic editor. The Logic Simulator has two rows of buttons at the top, a list of signals down the left side, and the signal waveforms directly to the right of the signal list.

### Simulating the Old-Fashioned Way

To simulate the **XOR** gate, first choose a step size of 1 ns (popup menu on top) and choose 1 ns/div (directly above the signal list, the two buttons on the left and right increase or decrease the time per division). Also, set the simulation type to functional using the popup menu in the first row of buttons at the top.

Next, add the desired signals by choosing **Signal:Add Signals** and double click on the signals that you want to see simulated (**X**, **Y**, **Z**, **X_L**, **Y_L**, **XNY**, **XYN**). Then click on close.

In order to simulate the **XOR** gate, you apply different combinations of inputs and observe the output. To set the state of an input, select the signal by clicking on it in the signal list, then click on the **Logical States** button in the second row of buttons at the top. Then choose the state of the input (high or low), and then click on close. Once all the inputs have been assigned a state, you can simulate one step by clicking the **Simulation Step** button in the first row of buttons at the top. Observe the output to make sure it is correct.

Verify the truth table and print the simulation results. Your simulation results should be the signal traces in the graphical Waveform Viewer; the simulator does not generate truth tables.

| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Next, use the simulator to observe the result of changing the inputs from **X**=0, **Y**=0 to **X**=1,**Y**=1. Do this first with the simulation type set to functional, and then with the simulation type set to unit. The unit simulation type adds a unit delay to the output of each gate. You should see a glitch (a short fluctuation in the output) when the **XOR** gate is simulated with unit delays. Explain why the glitch occurs and print the simulation results.

Try the built-in counter as a means of generating continuously changing test inputs. Press the **Select Stimulators** button on the bottom row of buttons at the top. This brings up a window showing the different stimulators built into the simulator. The row of LEDs next to the label 'Bc' represents the 16 bits of the built-in 16-bit counter. You can assign these bits to the signals you are simulating as a means of generating test inputs. Select **X** from the signal list and then press on the rightmost LED of the Bc row. This assigns **B0** to **X**. Now assign **B1** to **Y**, and press close. Next, go to **Options:Preferences** and under the simulation tab make sure that the **B0** frequency is 500 MHz and the **B0** period is 2ns. Click on OK, then use the **Simulation Step** button to step through the different test inputs generated by the counter. This is a useful technique for debugging, especially when there are a lot of inputs.

**Simulating with Scripts**

Writing a simulation script is usually more efficient than simulating by hand. Scripts allow you to automate repetitive testing tasks and spare you the work of adding and stimulating your signals each time you test a circuit. You can write your script in any text editor, or use the built-in script editor by choosing **Tools->Script Editor** from the Foundation Logic Simulator and clicking **Create Empty Script**. For a full list of simulation commands, choose **Help->Simulation Macros Help** from the Script Editor.

Here is an example script that performs the same tests we did by hand. Lines beginning with a vertical bar "|" are comments.

```
| EE121
| Script to test the XOR gate in Lab 1

| Issue some initialization commands
delete_signals
restart
greset
| Simulate in functional mode
set_mode functional
| Set the stepsize; i.e., the default simulation time interval
stepsize 1 ns
| Set the simulation precision...increasing precision time
| decreases timing accuracy but speeds up the simulation.
| Defaults to 100 ps.
sim_precision 125 ps
| Add signals to the watch window.
Watch X Y Z X_L Y_L XNY XYN

| Test the truth table
assign X 0
assign Y 0
| The sim command runs the simulation for one step size.
sim
assign X 0
assign Y 1
sim
assign X 1
assign Y 0
sim
assign X 1
assign Y 1
sim

| Test the transition from all 0's to all 1's in functional mode.
assign X 0
assign Y 0
sim
assign X 1
assign Y 1
sim

| Test the transition from all 0's to all 1's in functional mode.
set_mode unit
assign X 0
assign Y 0
sim
assign X 1
assign Y 1
sim

| Use counters
| At 8 ns, assign 0 to X, then increment X (modulo 2) every 2ns 8 times
wfm X @8ns = 0 (2ns = inc by 1) * 8
```

```
| At 8 ns, assign 0 to Y, then increment Y (modulo 2) every 4ns 4 times
wfm Y @8ns = 0 (4ns = inc by 1) * 4
sim 16ns
```

The script might not seem useful with such a small design, but as your designs become larger you will find them essential.

### 3.3. Creating a XOR gate macro

In order to use a schematic easily within another schematic, it is best to create a symbol macro. To create a macro out of the **XOR** gate schematic, select **`'Hierarchy->Create Macro Symbol from Current Sheet`** from the schematic editor. The symbol name should be **XORgate**, the inputs should be **X** and **Y**, and the output should be **Z**. Verify this, then click OK. You can now use the **XOR** gate in other schematics by selecting it from the components list.

## 4. Full adder

Create a new schematic called **FullAdder** using Figure 2 as a guide. The input signals are **A**, **B**, and **CIN** (carry in) and the outputs are **S** and **COUT** (carry out). This schematic uses the **XORgate** macro that you created earlier. Make sure you label the internal nodes **AXORB**, **AB**, **ACIN**, and **BCIN**. After you have created the schematic, create a macro from the schematic and name it **FullAdder**.

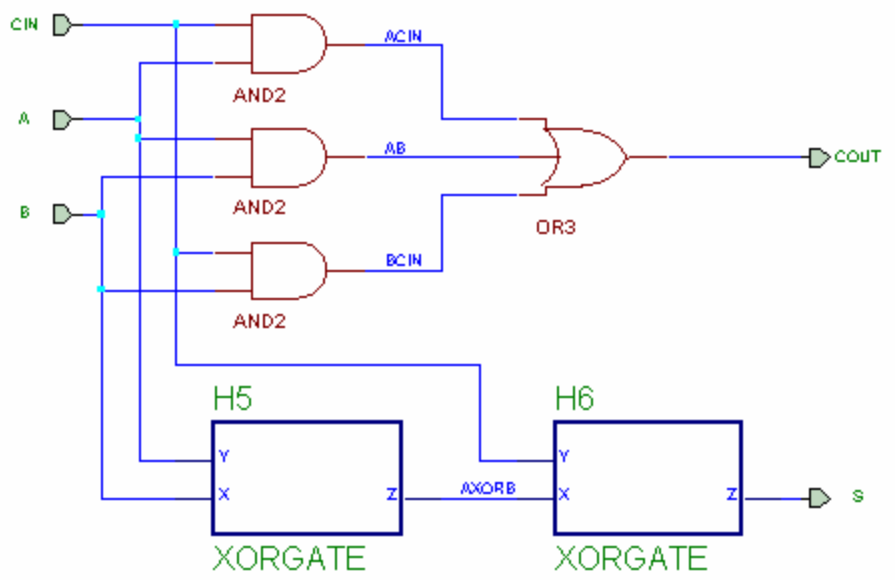


**Figure 2**. Full adder schematic.

Simulate the schematic and verify the truth table below (functional-mode simulation) with a script. Make sure you always include internal nodes in your simulation. Also, make sure that **`FullAdder.sch`** is the only schematic listed in the main Project Manger window, that is, the only schematic in your project). Including more than one schematic in a project can confuse the

simulator and cause errors; thus it is a good idea to always include only one schematic (the main top-level schematic) in your projects. All other (lower-level) schematics should be converted to and included as macros. To remove a schematic, right click on it in the list of project files and select **Remove**. To add a schematic, right click on the project name in the list of project files and select **Add**.

Alternatively, you can simulate a macro by choosing File->Simulate Single Component from the Simulator.

| A | B | CIN | S | COUT |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

After verifying the truth table above, print the simulation results.

## 5. 4-bit adder

Create a new schematic called **Adder4** using Figure 3 as a guide. You will use buses in this schematic. A bus is an array of signals, and is displayed as a thicker line than a regular wire. When using a bus, the naming convention is to give the bus a name in the form of *bus_name*[X:Y], where *bus_name* is the variable name and X is the most significant bit and Y is the least significant bit. Individual signals in the bus are named *bus_nameN* where *N* is between X and Y. Individual signals can be accessed by wiring a regular wire from a bus to a terminal and giving the internal node the name corresponding to the desired signal, or by using the **Draw Bus Taps**' tool.

After creating the **Adder4** schematic, simulate the 4-bit adder with a script for additions using both hexadecimal and decimal notation (functional-mode simulation). Again, only one schematic should be listed in the main project window. At this point, **Adder4.sch** is the top-level schematic and thus should be the only schematic file listed.

You might find the following simulation commands useful:

```
| Add the busses A[3:0] and B[3:0] to the watch window
vector A A3 A2 A1 A0
vector B B3 B2 B1 B0
watch A B
```

---

```
| Change A and B to decimal
radix dec A B
| Change A and B to hexadecimal
radix hex A B
```

---

```
| Assign hexadecimal values to A and B
assign A 2\h
assign B C\h
```

---

```
| Stimulate A, B and CIN with a pattern
pattern A   0\h D\h 7\h 5\h F\h
pattern B   6\h F\h 8\h 3\h 0\h
pattern CIN 0   0   1   1   1
(sim) * 5
```

Print the simulation results. Make a macro out of **Adder4**.

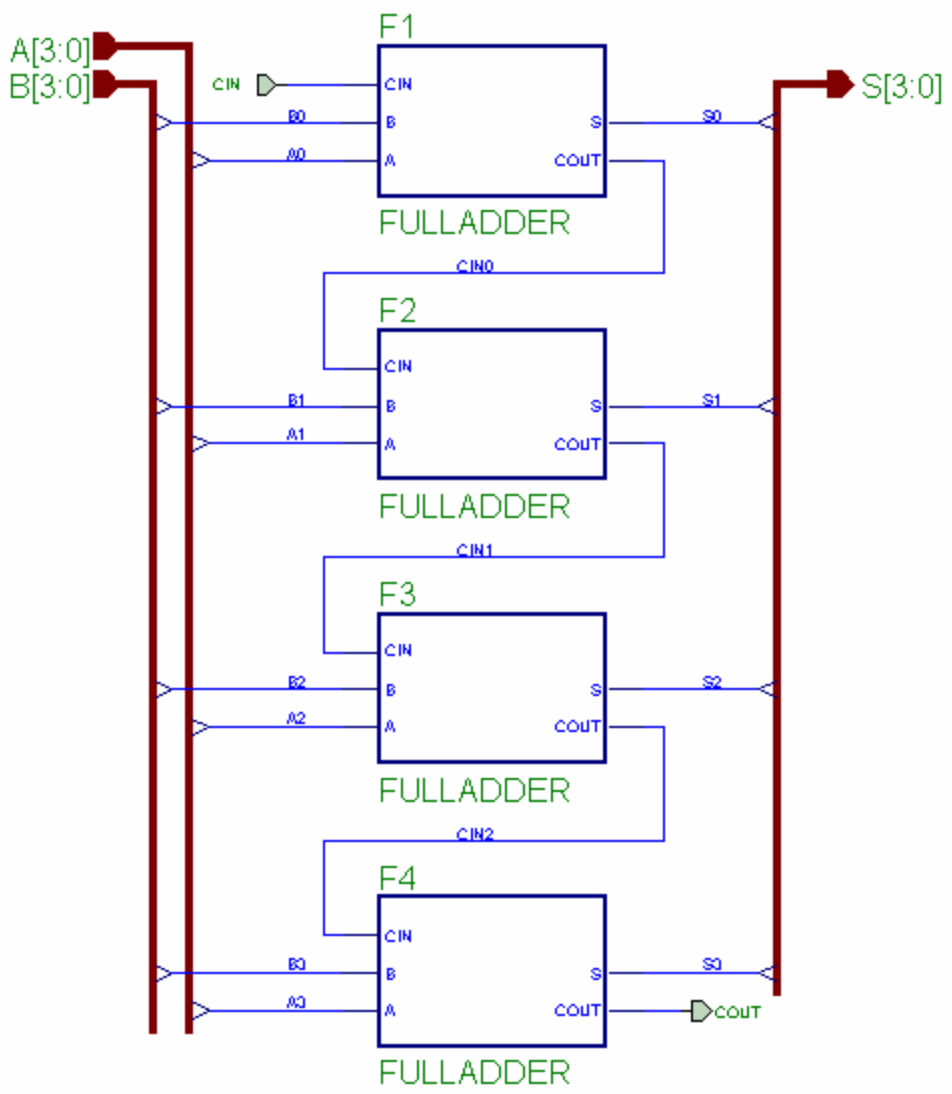


**Figure 3**. Adder4 schematic.

## 6. 16-bit adder

Create a new schematic called **Adder16** using Figure 4 as a guide. In this schematic you will learn how to pull a piece of a bus out of a bus (e.g., **A**[7:4] out of **A**[15:0]). Write a script to

simulate the 16-bit adder using decimal numbers and check the results for correctness (functional mode). Print the simulation results. Next, simulate the 16-bit adder in unit-delay mode. In particular, simulate the case where all inputs are 0 transitioning to all 1s. Print and explain the results.

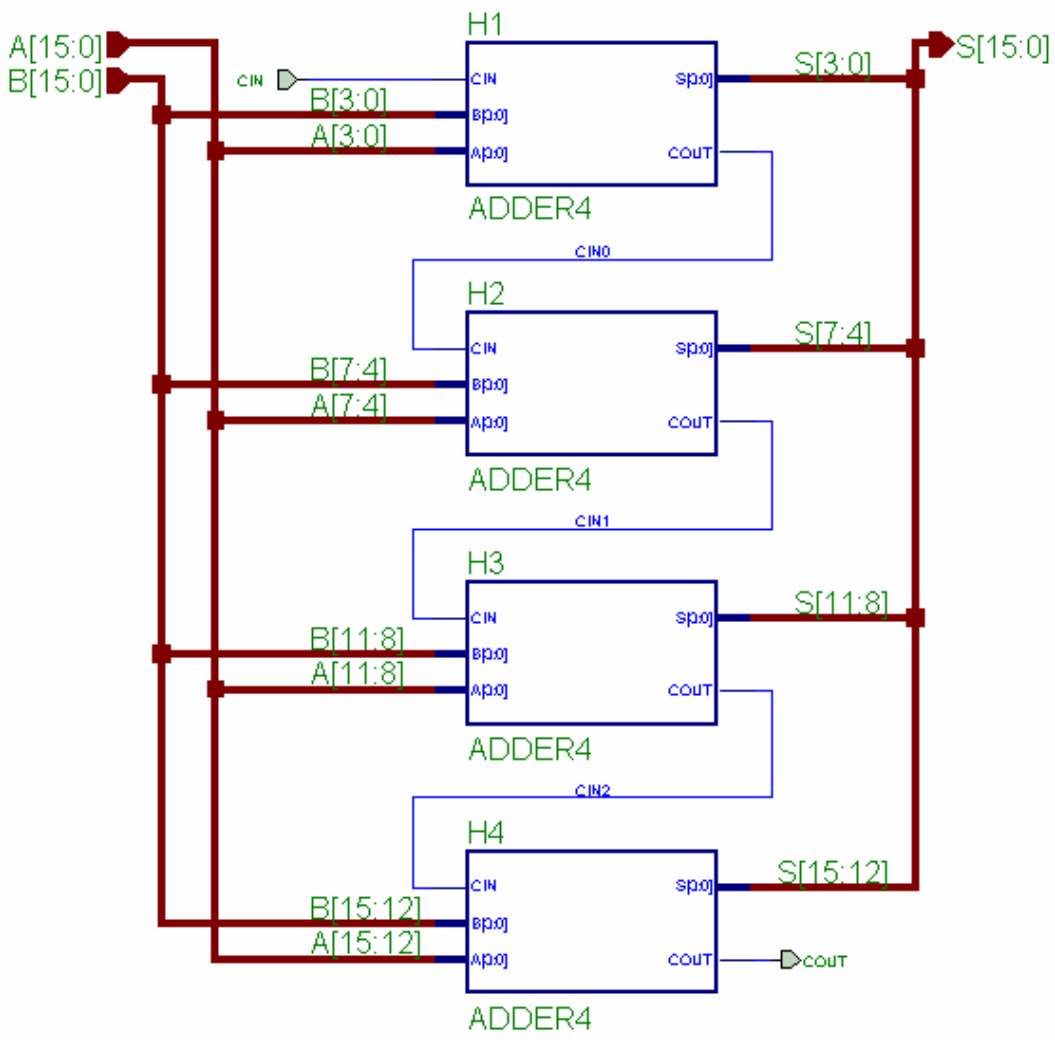


**Figure 4**. Adder16 schematic.

## 7. What to turn in

1. **XOR** gate schematic
2. **XOR** gate simulation results
   a. truth table verification
   b. inputs transitioned from 0s to 1s (both functional and unit-mode simulations)
   c. explanation of glitch in unit-mode simulation.
3. 1-bit adder simulation results (verification of truth table) and script
4. 4-bit adder simulation results (using both decimal and hexadecimal numbers) and script
5. 16-bit adder simulation results and script
   a. addition using decimal numbers in functional-mode

b. addition resulting when inputs are transitioned from 0s to 1s in unit-mode
c. explanation of the simulation results in unit-mode