Experiment # 3
Introduction to Digital Logic Simulation and
Xilinx Schematic Editor

1. Synopsis:
This lab introduces a CAD tool (Computer Aided Design tool) called Xilinx Schematic Editor, which is used in several courses (besides EE201L) at USC to design and simulate digital hardware. In this lab, we will implement a programmable NAND gate design using Xilinx Schematic Editor.

2. Lecture/Demo Video:
Before coming to the lab session, you are required to watch the “Introduction to Xilinx Schematic Editor” video (posted at the class website). It is a screen-capture of an entire session of Xilinx Schematic Editor starting from invoking the tool to schematic entry, test fixture creation and simulation.
3. Prelab:

3.1 Watch the *Introduction to Xilinx Schematic Editor* lecture/demo video. (10 pts)
   - Watched / Did not watch it yet.

3.2 Please install Xilinx Tools (Xilinx ISE, ChipScope, and Modelsim) on your home laptop / desktop. The step-by-step procedure to install the tools has been posted to the class website. (20 pts)
   - Installed / Did not install yet.

After having watched the lecture/demo video, answer the following Prelab questions:

Q 3.3: Two nets with same names (labels) but not “physically” connected to each other are “logically” connected to each other. (5 pts)
   - True / False

Q 3.4: If there is a solid blue square dot at the junction crossing two wires, these wires are ________________ (connected/ not connected) to each other.
   - A ________________ (hollow red square dot / solid blue square dot) indicates a dangling end of a wire. (5 pts)

Q 3.5: Mr. and Ms. Bruin made the schematics for a 1-bit adder. After saving the schematics, they wanted to run the Modelsim - Simulate Behavioral Model (Under Processes). What did they forget to do? (5 pts)

Q 3.6: Mr. and Ms. Bruin finally manage to get Modelsim to run but have no clue about how to verify their design from there. What do they need to do? (5 pts)

Q 3.7: How do you create a symbol? (5 pts)

Q 3.8: How do you switch between sheets? (5 pts)
   - For going to the next sheet:
   - For going back to the previous sheet:

Q 3.9: What is the purpose of labelling an instance of a design (for example each of the five 2-to-4 decoders in a tree-decoder forming a 4-to-16 decoder)? (5 pts)
4. Procedure:

4.1 Follow the guidelines given in the tutorial exercise (posted at the class website) and implement and simulate the 4-bit adder. This is just to familiarize you with the Xilinx Schematic Editor and Modelsim.

4.2 Programmable NAND gate, need for such a device:

In interfacing memory to a microprocessor, one needs to decode higher order bits of the address sent out by the processor (in a memory read or write transaction). For example, the Intel 8088 processor puts out a 20 bit address on 20 address lines \([A(19:0)]\). Since \(2^{20} = 1\) Mega locations = \(2^8 \times 2^{12} = 256\) of 16 K locations; you can use 256 16KB memory chips to fill up the 1 megabyte address space.

In a small application, you may not fully populate the entire 1 megabyte address space with memory chips. Instead, you may have a few 16KB memory chips occupying selected 16KB slots of the 256 slots. Depending on the slot you choose to use for a 16KB memory chip, you need to select that 16KB memory chip, only when (if and only if) a particular combination of the 8 higher order address bits \(A(19:12)\). For example, if your 16KB memory chip is occupying the first 16KB slot after the bottom half-megabyte, then \(A(19:12) = 1000\ 0000\). So we need to generate a low-active chips select signal \(\overline{CS}\) when \(A(19:12) = 1000\ 0000\). Thirty years back such address decoding was done using TTL chips. A TTL design is shown above. This design uses an 8-input NAND gate and several inverters. This means too many chips and too much wiring.

We are looking for a one-chip solution which can be used for a number of combinations of \(A(19:12)\), (not just suitable for only one combination, \(A(19:12) = 1000\ 0000\)). What if we make a configurable (programmable) NAND gate where the user can choose invert or not to invert an input before the signal reaches the NAND gate.

So, one possible one-chip solution is to package an 8-input NAND gate with 8 XOR gates and provide to the user the 8 INV control pins as shown on the side. The only problem with this design is that it has too many pins and the cost of the chip is too much.
If we forego the ability to invert all 8 inputs and agree to invert only 7 inputs at most, then we have 8 choices of inverting, namely “choose to invert zero inputs” to “choose to invert 7 inputs”. These 8 choices (namely, \( \text{inv}_a, \text{inv}_b, \text{inv}_c, \text{inv}_d, \text{inv}_e, \text{inv}_f, \text{inv}_g = 0,0,0,0,0,0,0 \) standing for zero inverters to \( \text{inv}_a, \text{inv}_b, \text{inv}_c, \text{inv}_d, \text{inv}_e, \text{inv}_f, \text{inv}_g = 1,1,1,1,1,1,1 \) standing for seven inverters) can be encoded into three bits. Hence, we will have three programming pins, called \( \text{p}[2:0] \). The 8 combinations of three pins are converted to the 8 combinations of the 7 control lines by the “Special encoder” shown in the adjacent figure. We call the design in the adjacent box, “an 8-bit programmable NAND gate”. Instead of building this directly, here in this lab, we will build a 4-bit programmable NAND gate in part 1 of the lab and use two of these (and additional logic) to build the 8-bit programmable NAND gate.

4.3 Create a new project under “C:/xilinx_projects/”. Here, we are going to implement (and simulate) on Xilinx Schematic Editor, a programmable NAND gate. In part 1 we implement a 4-bit programmable NAND gate and in part 2 an 8-bit programmable NAND gate. Partially complete designs are given to you. You need to complete the design and simulate it. Understand the specifications and complete the design before proceeding.

4.4 Specifications for the 4-bit programmable NAND gate- Part 1

<table>
<thead>
<tr>
<th>( \text{p}[1:0] )</th>
<th>( \text{p}0 )</th>
<th>( \text{p}1 )</th>
<th>( \text{inv}_a )</th>
<th>( \text{inv}_b )</th>
<th>( \text{inv}_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

a, b, c, d -- Inputs to the NAND gate
\( \text{p}[1:0] \) -- Programming inputs (2-bits) The indicate the number of inputs that need to be inverted.

\( \text{p}[1:0] \) can take values in the range 0-3
The two extreme cases are
1. when no input is inverted (\( \text{p}[1:0] = 0 \))
2. 3 of the 4 inputs are inverted (\( \text{p}[1:0] = 3 \))

When \( \text{p}[1:0] = 0 \) None of the inputs are inverted, Output \( \text{y}_{\text{bar}} = (a.b.c.d)' \)

When \( \text{p}[1:0] = 1 \) 1 input is inverted (a is inverted), Output \( \text{y}_{\text{bar}} = (a'.b.c.d)' \)

When \( \text{p}[1:0] = 2 \) 2 inputs are inverted (a & b are inverted), Output \( \text{y}_{\text{bar}} = (a'.b'.c.d)' \)

When \( \text{p}[1:0] = 3 \) 3 inputs are inverted (a, b & c are inverted), Output \( \text{y}_{\text{bar}} = (a'.b'.c'.d)' \)
4.5 Derive the boolean expression for the signals (inv_a, inv_b, inv_c) and draw the AND – OR implementation of the control signals (inv_a, inv_b, inv_c) for the special encoder.

4.6 Get your paper-pen design checked by the TA and then complete the design in Xilinx Schematic Editor.

4.7 After completing the schematic entry part, create a verilog test fixture to test the design. A sample incomplete verilog test fixture is already provided to you (as part of the .zip file for this project). Following the guidelines given in the xilinx schematic entry tutorial exercise, complete the test fixture to simulate the design exhaustively. That is, the simulation patterns should exhaust all possible input combinations. Use Modelsim to simulate the design using the test fixture you wrote. You can seek help from your TA in creating the verilog test fixture file.

Do not hesitate to seek help as this is your first simulation exercise using a CAD tool and we do expect that students will experience difficulty.

4.8 After completing Part 1 and ensuring that the design works correctly, create a symbol of the 4-bit programmable NAND gate. (use the tutorial exercise for guidelines.)

4.9 For Part 2, design an 8-bit programmable NAND gate (with the following specification), using 2 of the 4-bit programmable NAND gates (designed in part1) and some additional logic.

4.10 Specifications for the 8-bit programmable NAND gate:

\[ p \text{ [2:0]} \text{ -- Programming inputs (3-bits)} \]

Indicates the number of inputs that need to be inverted.

\( p \) can take values in the range 0-7

The two extreme cases are

1. when no input is inverted \((p = 0)\)
2. 7 of the 8 inputs are inverted \((p = 7)\)

When \( p \text{ [2:0]} = 0 \) None of the inputs are inverted, Output \( y\text{ _bar} = (a.b.c.d.e.f.g.h)' \)

When \( p \text{ [2:0]} = 1 \) 1 input is inverted \((a\text{ is inverted})\), Output \( y\text{ _bar} = (a'.b.c.d.e.f.g.h)' \)

When \( p \text{ [2:0]} = 2 \) 2 inputs are inverted \((a \& b \text{ are inverted})\), Output \( y\text{ _bar} = (a'.b'.c.d.e.f.g.h)' \)

\( \cdots \)

When \( p \text{ [2:0]} = 7 \) 7 inputs are inverted \((a, b, c, d, e, f, g \text{ are inverted})\),

\( \text{Output } y\text{ _bar} = (a'.b'.c'.d'. e'.f'.g'.h)' \)
Hints for arriving at the Part 2 design:

Suppose we wire 4 of the 8 inputs (a, b, c, d) to one 4-bit programmable NAND gate (instance label: TOP) and the other 4 inputs (e, f, g, h) to the other 4-bit programmable NAND gate (instance label: BOT for bottom) as shown on the side.

Each of the two 4-bit programmable NAND gates will provide us ability to invert 3 inputs. So, as is, we can invert at most 6 inputs. But we need to invert a total of 7 inputs. So let us have an external XOR gate in the “d” input of the TOP to invert the “d” input for certain appropriate combinations of p2, p1, p0.

So the modified suggested design is as shown below. Complete the function table for the “Special Encoder” below.

<table>
<thead>
<tr>
<th>Truth Table for the Special Encoder for Part 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

4.11 Complete the truth table above and also decide what gate you wish to use for combining the top_y_bar and bot_y_bar into the overall y_bar. You need to submit this.

4.12 As in part 1, create the schematic in Xilinx Schematic Editor, create a verilog test fixture and verify your design.
5. Lab Report:

Name:_________________________          Date: _________________________

Lab Session:_____________________        TA's Signature: __________________

For TAs: Prelab (out of 65): ______  Report (out of 85): ______

Comments:

Q 5. 1: Submit the completed truth table and the final gate producing the overall y_bar. (15 pts)

![Truth Table for the Special Encoder for Part 2](image)

Q 5. 2: Print and attach the following items to this report.
Your TA will familiarize you with the commands to take printouts.

(a) the schematics (prog_nand_4_inp.sch, prog_nand_8_inp.sch), (30 pts)
(b) the verilog test fixture file (prog_nand_4_inp_tb.v, prog_nand_8_inp_tb.v), (10 pts)
(c) the portion of the Part 1 waveform showing the combination of inputs a,b,c,d (10 pts) for which the output signal, y_bar, becomes zero, when p1,p0 = 1,0
and the portion of the Part 2 waveform showing the combination of inputs a,b,c,d,e,f,g,h for which the output signal, y_bar, becomes zero, when p2,p1,p0 = 1,1,0.

Q 5. 3: Are the following, valid simulation commands? Try them on the modelsim tool and answer.(10 pts)

VSIM>run 20ns  (Valid / Invalid)
VSIM>run 20    (Valid / Invalid)
VSIM>run 20 ns (Valid / Invalid)
VSIM>run 20 cycles (Valid / Invalid)