PROJECT 1

Getting to know Verilog and Xilinx ISE

Through examples of basic combinational and sequential circuits, you will get to know about Verilog HDL and the Xilinx ISE development environment. You will also get a flavor of the FPGA design workflow.

Introduction

In this lab, you will learn the basics of Verilog as a hardware description language, and get more familiar with the Xilinx ISE as your simulation and synthesis tool. You need to provide the simulation, validate the design, and explain the procedure to implement it on an actual board as you were taught (since the course is online, the mere explanation is enough). You need to provide your own test bench for the simulation and accurately describe the combinational logic circuit and the procedures to the simulation. The accepted testbench output should be from Xilinx Isim simulator.

The second part of this lab involves the design of a simple sequential circuit. You are provided with the schematics, and you are expected to provide the code as your design, perform the simulations, and implement a small design revolving around counters.

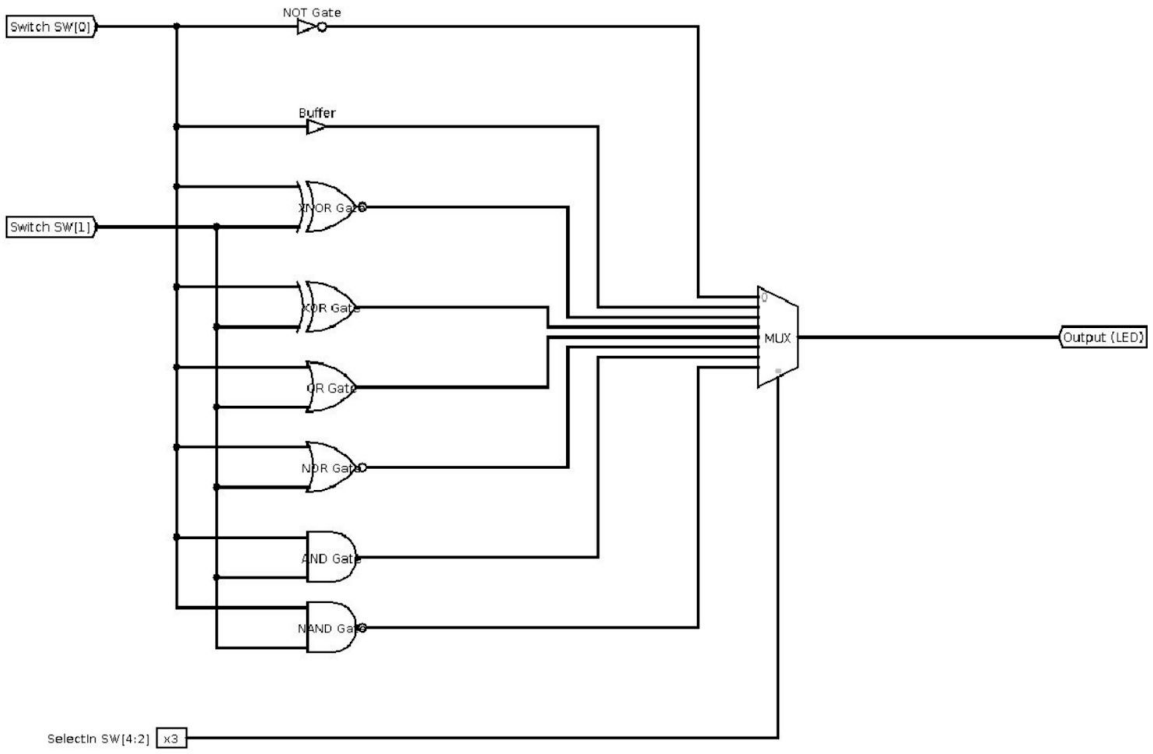
Remark

Remember to review course materials and additional resources to get yourself more familiar with the concepts and Verilog language. The more experienced you be in Verilog, the easier the entire course would be for you.

Combination Circuit Example

In this part, you will design the hardware and the testbench for a combinational logic circuit to be implemented on an FPGA board. This combinational logic consists of eight gates, NAND, AND, NOR, OR, XNOR, XOR, NOT, Non-inverting buffer (BUFF) gates and a multiplexer at the output to select between the outputs of the gates.

This circuit has its outputs switched to one output using an 8:1 multiplexer. The inputs to the gates, in an actual setting, would be shared with two slider switches on the Nexys3 board.

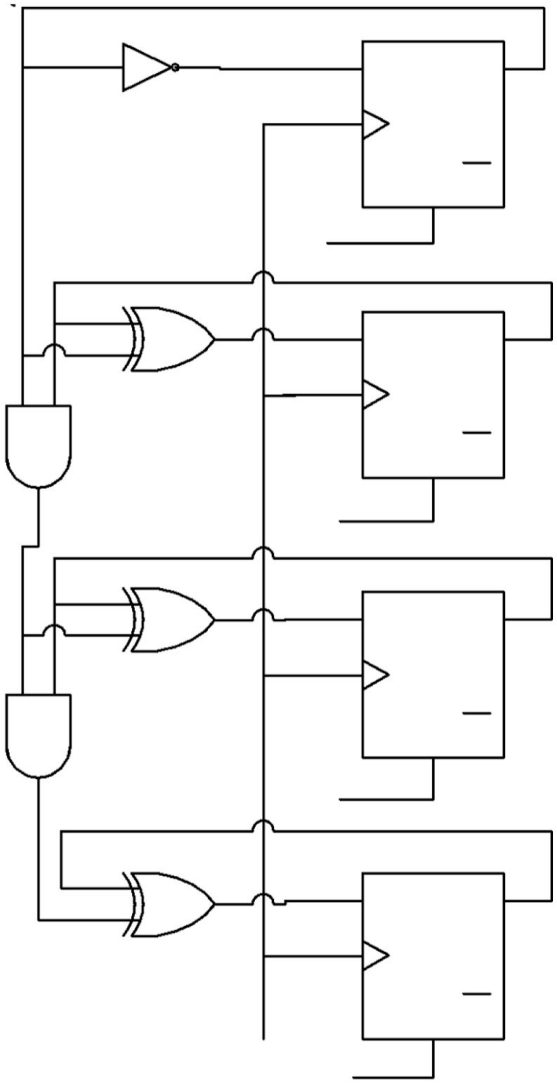


Sequential Circuit Example

In the previous part, you learned the FPGA design workflow with a combinational logic design. Now we move to the sequential circuits. In this part, you will design and perform simulations, and explain the real implementations, of several small projects revolving around counters. It is important to review the lecture materials to refresh your mind before proceeding. We will follow three steps: first, we design a 4-bit counter using gate-level hardware description based on schematics, and perform the simulation. Second, we design the same counter using a higher-level abstraction, and perform simulations. In the final step, we create a simple clock divider using a counter and simulate this clock and describe our findings.

4-Bit Counter: Translating the Schematics

The following figure shows the schematics of a 4-bit counter with D flip-flops.



With knowledge you acquired from M51A, you should be able to easily recreate the same schematics through the use of K-maps. The Xilinx ISE does provide an interface for schematics creation and edition, schematic 1 based design methods are no longer in use in the real world. Our job here is to translate the schematics into Verilog code. We begin by creating a new Verilog module. Since we are translating the schematics, each gate should be mapped to an operator, and each flip-flop should be mapped to one line in the edge-sensitive always block. Remember to use non-blocking assignment <= in edge-sensitive always blocks.

Write a verilog module that describes this schematic, and show the counter in simulation for your report.

In the next section, we will see another way to describe a counter, this time a bit easier.

4-Bit Counter: Modern Version

The counter you just created using gate / flip-flop level logic seems pretty complicated. Fortunately, Verilog HDL provides a higher level abstraction. The following code snippet shows the same counter written in a different fashion:

```
// Example Verilog code for the counter
reg [3:0] a;
always @ (posedge clk)
  if (rst)
    a <= 4'b0000;
  else
    a <= a + 1'b1;
```

When synthesized, the code will produce the same results as the previous one. With that powerful tool, we can simply write HDL code to achieve the purpose. However, you should always remember that you are writing hardware, rather than software program. Whatever code you write, you should be aware of the hardware that is underlying.

Create a new module, write the above code snippet, and test it in simulation. Does it give the same out puts as the previous one?

Clock Divider: Counter in Action

One of the most important applications for counters is to create clock dividers. For example, if we have a 4-kHz clock, and we want to create a 1-kHz clock called *clk_1kHz*, the easiest thing is to create a 2 bit counter, and flip the *clk_1khz* signal every time the counter resets. Now your job is to create an flashing LED with a frequency of 1-Hz. Since we are not working with a real FPGA at the moment, you should create an output wire and consider “1” as its light-on state, and “0” for the light-off state. In your test bench, design a 10kHz clock and use that.

Remember to Respond

Note that for this and other projects, your submission should contain every material that is explicitly mentioned in the syllabus. In addition, please make sure that your written document contains responses to the following requirements as well:

- What is a ".ucf" file? How would you use it for Nexys3 in an actual setting on a real board to connect the inputs of the combinational circuitry to the switches on the FPGA?
- Providing all the schematics of your designs is mandatory

Deliverables

Your submission must comply with all the requirements mentioned in the syllabus, and will be graded accordingly.

1. Your video demo recording (ONLY screen recording of the computer you have been using for this assignment, and your verbal explanations of main steps, including top-module designs, the roles of main code blocks, how to simulate, what the waveforms mean, etc.
2. Your written report
3. Your codes and files

For more information on file-formats and submission guidelines, please refer to the syllabus.