CS M152A Lab 0

Introduction to 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

This lab, you will learn the basics of Verilog HDL, the Xilinx ISE and the workflow of FPGA design. There are two parts in the lab. The first part involves the simulation, implementation and testing of a combinational logic circuit. You will provided with the Verilog code and testbench. You are expected to simulate the design using the testbench on Xilinx ISIM simulator. Then you will implement the design on the Spartan 6 FPGA on the Nexys3 board and test the working of the design on the board.

The second part involves a simple sequential circuit. You are provided with schematics and sample code, and you are expected perform simulations and implement a small design revolving around counters.

The following is a table of contents that are used in this lab.

File	**Note**
./lab_0.pdf	The lab manual.
./fpga_fundamentals.pdf	Theory and practical guide to FPGA design and implementation using Xilinx ISE.
./src/	The folder containing example code
./src/Nexys3.ucf	The User Constraint File that includes all the pin mappings of the Nexys3 board.

New to FPGA?

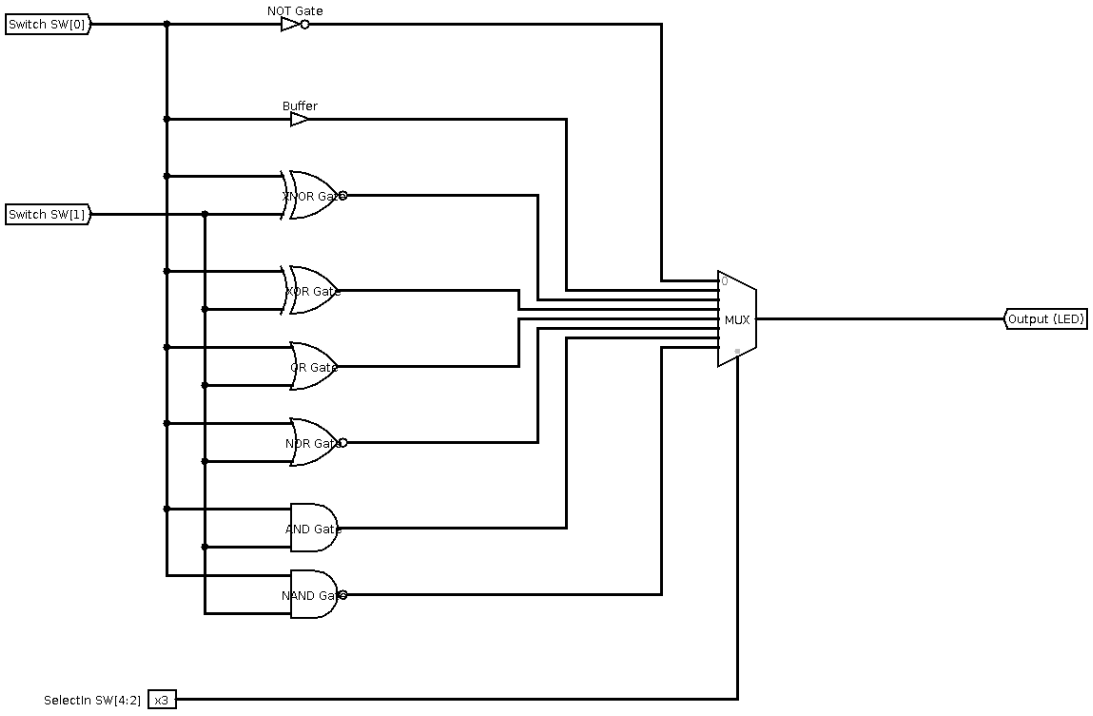
If you are new to FPGA, part I of the *fpga_fundamentals* slides provide some of the background knowledge that you need to know before you start the lab.

Combination Circuit Example

In this demonstration, you will simulate, implement and test a combinational logic circuit on the FPGA board. The combinational logic circuit 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. The circuit consists of eight logic gates with their outputs switched to one output using an 8:1 multiplexer. The inputs to the gates are shared to with two slider switches on the Nexys3 board. Switch SW[0] is as input for all single input gates. Both SW[0] and SW[1] is used

for inputting to two-input gates. The schematic for this implementation is given as the following figure. The select inputs for the multiplexer is entered using the slider switches, SW[4:2].

You can find the sample code in the *src/combinational_gates* folder. You should go through the part II of the *fpga_fundamentals* slides, which will guide you through the process of implementing this demo project.



Sequential Circuit Example

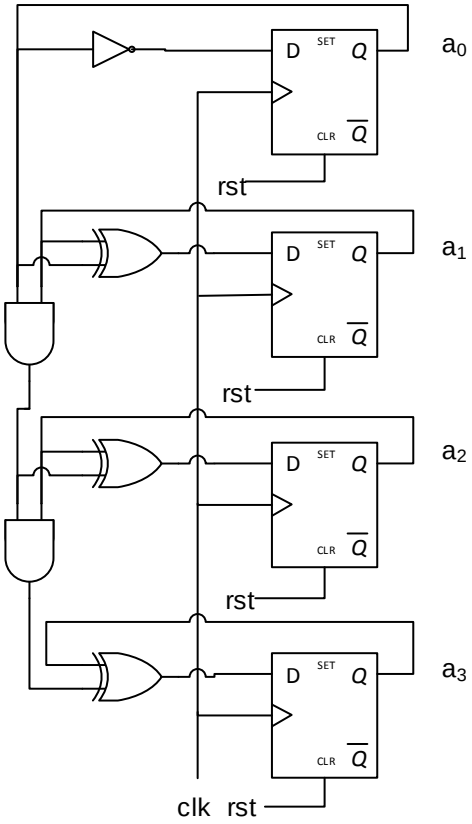
In the first part, you learned the FPGA design workflow through examples of combinational circuit. Now we are moving to sequential circuits. In this part, you will implement perform simulations and real implementations of several small projects revolving around counters. We will follow three steps: first, we implement a 4-bit counter using gate-level hardware description based on schematics, and perform simulation. Second, we implement the same counter using a higher level abstraction, and perform simulation. In the final step, we create a simple clock divider using a counter, and implement the design on the FPGA

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[1], schematic 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. The following code snippet shows the example code that represents a0.




```
// Example Verilog code
// for a0

reg a0;

always @ (posedge clk)

  if (rst)

    a0 <= 1'b0;

  else

    a0 <= ~a0;
```

Finish the translation for the whole circuit and test it in simulation. If you see red Xs in the value, it means that the values are not *initialized*. You should reset the circuit by setting `rst` first, and then proceed.

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. 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

[1] http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_schematic_overview.htm

software program. Whatever code you write, you should be aware of the hardware that is underlying.

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

Create a new module, paste 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-Hz clock, and we want to create a 1-Hz clock called `clk_1hz`, the easiest thing is to create a 1 bit counter, and flip the `clk_1hz` signal every time the counter resets. Now your job is to create a flashing LED with a frequency of 1-Hz.

To put the code onto the board, you'll need to use the User Constraint File(UCF). UCF lists all the available pin mappings in the FPGA in the following format:

```
Net "your_signal_name<bit_index>" LOC = XX | IOSTANDARD = LVCMOS33; # More details about the pin
```

For example:

```
Net "sw<0>" LOC = T10 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L29N_GCLK2, Sch name = SW0
```

You only need to uncomment the pins that you'll use in the UCF to activate the connection. Our Nexys3 has a 100MHz master clock. You can access it through uncommenting the "clk" net in the UCF. You'll also need to uncomment one of the LED lines to connect your counter output with an LED. For more information on how to map your design to the FPGA board using UCF, you can go back and check the combinational gates example. Using that clock and a counter, you should be able to create 1-Hz clock. Then, after hooking the 1-Hz clock to one of the LEDs, you should be able to create an LED flashing with 1-Hz frequency.

Deliverables

When you finish, you should demo the following parts of your design to the TA:

1. Simulation and Implementation of the combinational gates example
2. Simulation of two 4-bit counters
3. Implementation of LED flashing with 1-Hz frequency