Intro to Digital Logic, Lab 2
An Introduction to Verilog and Digital Components (Part 2)

Lab Objectives

In lab #1 you learned the basics of entering a design for simulation in Quartus II, and how to wire up a basic gate on the breadboard. In this lab we will take things further. You will also implement a more complex design on the breadboard, and will complete the flow of mapping designs from Verilog to the FPGA inside the DE1 SoC.

Design Problem – Multi-level Logic on the Breadboard.

Electronics can be very cold and impersonal. So let's change that by having your DE1 recognize you and only you (assuming you only have 9 other friends with carefully chosen student ID numbers…). We want a circuit that will light up an LED when it sees the last digit of your student ID # and no other (including invalid codes such as 1111). The list of all possible codes is given below – find your student ID number under "meaning", and design your circuit to match just that pattern. Your goal is to design the circuit out of Inverter, NAND, and NOR gates, and use the fewest number of gates as possible. Note that to make it fair to those with more difficult student ID #'s, inverters hooked directly to a switch output are free.

First, design the circuit by hand, optimizing with Boolean logic as needed. Then, implement the design on the breadboard. You may use '00, '02, and '04 chips as needed, but cannot use any other types of gates.

Note that for all design problems, you will be graded 100 points on correctness, style, testing, etc. Most labs will also have a bonus category, with which you can get up to 20 bonus points, though only through effort above and beyond just getting it working (i.e. not everyone will get bonus points, and on some labs no-one will get all 20). For this lab, the bonus goal is to use as FEW logic gates as possible. Each gate (an individual Inverter, an individual NAND, and an individual NOR) cost the same, though any inverters connected directly to a switch input are free. The fewer the number of gates, regardless of the number of chips, the better the grade.

SW3	SW2	SW1	SW0	Meaning
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6

0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Design Problem – Multi-Digit Recognizer in the FPGA.

Read the Verilog Tutorial on the website up to (but not including) Register Transfer Level (RTL) Code.

The circuit you developed for the previous section is good, but recognizing only one digit is a bit simplistic. We'd like to scale it up to handle more digits, though wiring it all on the breadboard will be way too much work. Instead, we'll develop a Verilog version and load that into the FPGA.

First, complete the Quartus II tutorial you started in lab #1. This will involve loading the mux2_1 design into the DE1 FPGA and testing it with the switches on the board. This will show you the complete flow for creating FPGA-based designs.

Next, develop the logic for a circuit that will recognize the bottom two digits of your student ID number. SW3-SW0 will be the bottom digit, encoded like in the previous section. SW7-SW4 will be the next digit up, using the similar code (i.e. when SW7==1 and SW6, SW5, and SW4 are each 0, the digit encoded is 8). To make things easier I have provided a structure for the file below that will help you get started, and hook things up to the proper inputs and outputs. Note that we do NOT care about the number of gates in the Verilog version – we'll start worrying about FPGA efficiency in later labs.

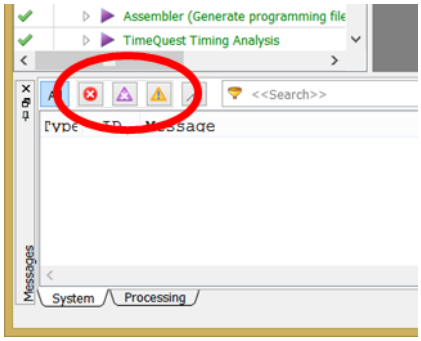
```
// Top-level module that defines the I/Os for the DE-1 SoC board

module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
   output   [6:0]    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output   [9:0]    LEDR;
   input    [3:0]    KEY;
   input    [9:0]    SW;

   // Default values, turns off the HEX displays
   assign HEX0 = 7'b1111111;
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

   // Logic to check if SW[3]..SW[0] match your bottom digit,
   // and SW[7]..SW[4] match the next.
   // Result should drive LEDR[0].

endmodule

module DE1_SoC_testbench();
   wire  [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   wire  [9:0] LEDR;
   reg   [3:0] KEY;
   reg   [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR,
.SW);

   // Try all combinations of inputs.
   integer i;
   initial begin
      SW[9] = 1'b0;
      SW[8] = 1'b0;
      for(i = 0; i <256; i++) begin
         SW[7:0] = i; #10;
      end
   end
endmodule
```

Note that the testbench uses an "initial" statement to generate the input patterns, with #10 delays, for loops, etc. These are great for quickly writing testbenches, but should not appear in the code that actually gets put into the FPGA.

Make sure to test your design in simulation BEFORE mapping it to the FPGA. This will speed up your development time for this lab, and will be critical as your designs get larger.

Quartus II Tip: Many of the "warning" messages that Quartus prints are actually early warnings of errors in your design. You should fix your Verilog to remove all warnings before simulating your design, and your final project should compile without warnings.

To quickly find errors or warnings, the upper-right corner of the message window has filtering icons that will just show errors, critical warnings, or warnings (left to right) in the message window:



Note however that Quartus II does have some useless warning messages. We have set up the tool to filter out many of them, but if you find a warning message that you don't think anyone should see, please show your TA. We will augment the system to ignore these, or explain why that actually is a real problem with your code.

During simulation you'll likely notice two new kinds of waveforms – red and blue. Red (or the value X) represents an unknown value – since our testbench doesn't specify the value of the KEY inputs, their value is unknown. Blue (or the value Z) represents a signal that is disconnected – since our circuit doesn't make any connection to the LEDRs other than LEDR[0], the others are disconnected. Red and Blue lines are a way for the simulator to get your attention and have you think about whether those values are what you actually want.

Note: do NOT remove the 1-digit design from the breadboard when you develop the 2-digit design in Verilog, you'll need to demo both to the TAs. When you load your design into the FPGA, the default design that connects the breadboard to the DE1 sliders and LEDs is overwritten, isolating the breadboard signals from the rest of the DE1. If you turn the DE1 off and then on again, the default configuration will be back.

Lab Demonstration/Turn-In Requirements

A TA needs to "Check You Off" for each of the tasks listed below. 100 points for correct design, plus up to a possible 20 bonus points for the most efficient breadboard circuit.

- Turn in to the TA the circuit diagram for the breadboard circuit.
- Turn in to the TA a printout of the Verilog for the 2-digit design.
- Simulate in ModelSim your 2-digit recognizer for the TA.
- Demonstrate to the TA your 1-digit and 2-digit ID recognizers working in hardware.
- Tell the TA how many hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).