Andrew Scott
E155
Lab 1
September 14, 2015

Lab 1: Utility Board Assembly

Introduction

In this lab, a utility board was assembled and then tested. The board was then used to power a seven-segment display by writing Verilog code that took input from four switches and displayed the hex number of the binary input from the switches.

Design and Testing Methodology

On the hardware side, there were not that many design decisions to be made. I followed the step by step instructions to do all of the soldering, using solder paste for the surface mounts and regular solder wire for the through-hole mounts. For the seven segment display, I decided to bring wires from all of the pins on the display to holes on the breadboard below the display in the order in which their pins appeared on the board. This allowed me to add the resistors and connect to the utility board pins in order (the top utility board pin being used for the display connected to the top resistor/display pin). To test the hardware, I first checked that there were no short circuits in the board by using a multimeter to check the resistance between v_{in} and ground, and since the resistance was around 0.1 MΩI knew there was no short. I then tested the power by hooking the board up to 5V power and checking the voltage on all of the voltage regulators. Finally, I tested the LED's, switches and seven segment display by writing Verilog code and checking that all the outputs from the LED's and the display were the same as they were supposed to be according to the simulation I performed of my Verilog code.

On the software side, there were several design choices to make. I decided to implement the seven segment display control logic as its own module, partly because it was cleaner that way and partly because I will most likely need to use it again in the future and want to have it easily accessible. Another decision I made for the seven segment display was to use some boolean equations for each of the segments that I wrote in E85, which I simplified and were very easy to write in Verilog. The other software design choice I had to make was how to make the last LED flash at 2.4 Hz. I decided to do this using a counter that counted up to a certain number (calculated to be the number of 40 MHz clock cycles in a single 2.4 Hz flash) and then flip the status of the LED every time this number was reached. Although this solution did not feel optimal, I was unsure of any other good way to use the fast clock cycle for something changing as slow as the LED was. To test my software, I ran a simulation in ModelSim and tried every combination of switches, confirming that the output on the LED's was correct. I also tested the seven segment display logic by running the Verilog code on my board and checking that the correct symbol was displayed for each combination of switches pressed. Finally, I timed how long my flashing LED took to flash 10 times, which came out to just over 4 seconds thus showing that it was flashing at around 2.4 Hz.

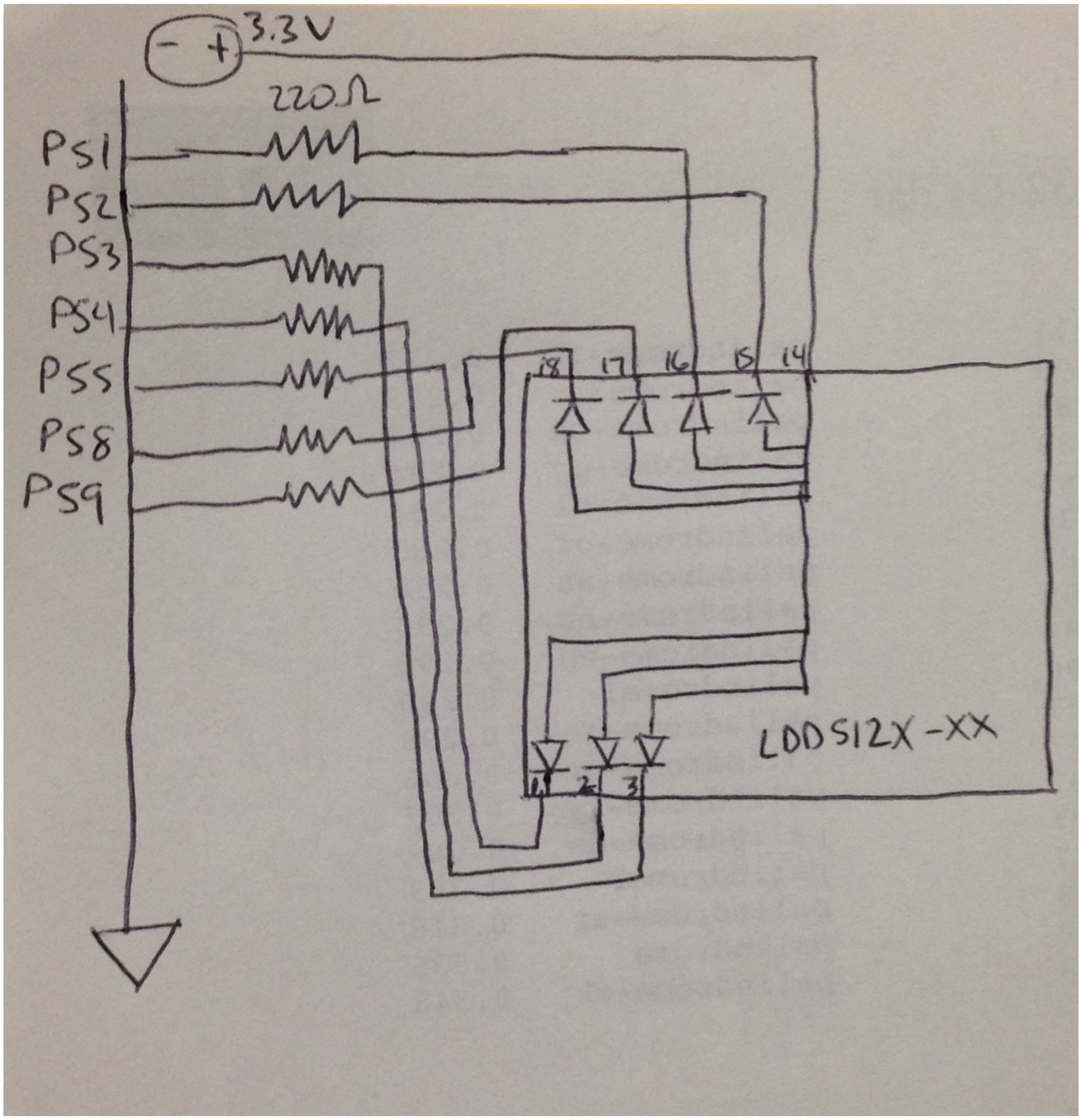
Results and Discussion

I was able to accomplish all of the lab tasks after some debugging. Everything on the board works as it is supposed to. Some of the solder joints are not as pretty as I would like them to be, but this was my first experience with soldering so I am not too disappointed with how the utility board turned out.

Conclusion

In this lab I put together a utility board and then used it to control a seven segment display. Everything performed as expected after some debugging. In all, the lab took about 8 hours to complete.

Breadboard Schematic



Verilog Code

```
// Andrew Scott

/* Every module should
begin with a comment section that includes your name and email address, the date of
creation, and a brief summary of its purpose, so that somebody else can understand what
the module does and get ahold of you if they need help. */

/*
Andrew Scott
ascott@hmc.edu
Created on September 10, 2015

This module is the top level module for e155 lab 1. It takes input from four
switches and then controls the output of a line of LEDs as well as a seven segment
display based on the input from the switches.
*/
module lab1_as (input logic clk,
                    input logic [3:0] s,
                    output logic [7:0] led,
                    output logic [6:0] seg);

    logic[31:0] count;
    logic old_led;

    // make sure count and old_led are initialized properly
    initial
        begin
        old_led <= 1'b1;
        count <= 32'b0;
        end

    // logic for all of the ouput LEDs
    assign led[0] = s[0];
    assign led[1] = ~s[0];
    assign led[2] = s[1];
    assign led[3] = ~s[1];
    assign led[4] = s[2];
    assign led[5] = ~s[2];
    assign led[6] = s[2] & s[3];
    assign led[7] = old_led;


    always_ff @(posedge clk)
        begin
        // count each clock cycle and then flip our blinking LED every 8320000
        // cycles (will result in blinking at a rate of 2.4 Hz)
        count <= count + 1;
        if (count === 32'd8320000)
            begin
            old_led = ~old_led;
            count <= 32'b0;
            end
        end

    // create an instance of the seven segment display
    seven_seg_disp display(s, seg);
endmodule


/*
Andrew Scott
ascott@hmc.edu
Created on September 10, 2015
```

```
This module contains the logic for controlling a common anode seven segment
display with a four bit input. The module uses a truth table  to determine when
to light up each segment of the display.
*/
module seven_seg_disp(input logic [3:0] s,
                    output logic [6:0] seg);

always_comb
case(s)
    4'b0000 : seg = 7'b1000000;
    4'b0001 : seg = 7'b1111001;
    4'b0010 : seg = 7'b0100100;
    4'b0011 : seg = 7'b0110000;
    4'b0100 : seg = 7'b0011001;
    4'b0101 : seg = 7'b0010010;
    4'b0110 : seg = 7'b0000010;
    4'b0111 : seg = 7'b1111000;
    4'b1000 : seg = 7'b0000000;
    4'b1001 : seg = 7'b0011000;
    4'b1010 : seg = 7'b0001000;
    4'b1011 : seg = 7'b0000011;
    4'b1100 : seg = 7'b0100111;
    4'b1101 : seg = 7'b0100001;
    4'b1110 : seg = 7'b0000110;
    4'b1111 : seg = 7'b0001110;
    default : seg = 7'b0000000;
endcase
endmodule
```