ECE3300L

Summer 2025 Semester

Lab 3

Kevin Yu

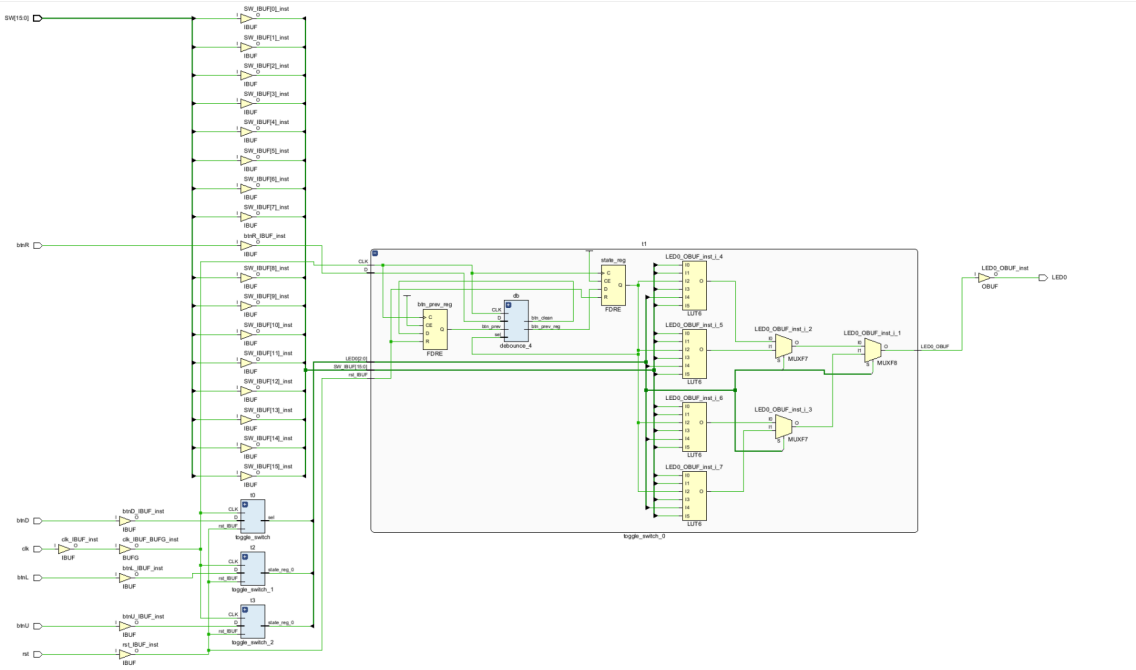
Noah Bocanegra

7/2/25

For the third lab, we are tasked with designing and verifying the code for a 16x1 multiplexer using 16 switches and 4 push buttons that determine which output will be displayed. We will implement a clk input to help determine whether a button press is a false/ unintended input. The reset button will clear the output selected.

The youtube link of our Demo:

We were given the code of different modules, starting with the 2x1 MUX that we were building our 16x1 off of. We began by first starting with the 2x1 and checking the schematic and adding pieces until we had our final schematic. The end result was:



We can see all our 16 inputs and the select for the push buttons which pass through our toggle switch code in order to get clean and correct push registers. These then pass through the different MUX stages until we get our final output.

The code we were given, we initially start with a 2x1 MUX:

```
module mux2x1 (
 input a, b,
 input sel,
 output y
);
 wire nsel, a1, b1;
 not (nsel, sel);
 and (a1, a, nsel);
 and (b1, b, sel);
 or (y, a1, b1);
endmodule
```

Where our a and b are the data and the sel is determined by whether or not the button is 0 or 1. It is a simple and effective mux that we can loop and level in order to achieve a 16x1 mux. The next module to add was the 16x1 mux loop:

```
module mux16x1 (
input [15:0] in,
input [3:0] sel,
output out
);
wire [15:0] level1;
wire [7:0] level2;
wire [3:0] level3;
genvar i;
generate for (i = 0; i < 8; i = i + 1)
mux2x1 m1 (.a(in[2*i]), .b(in[2*i+1]), .sel(sel[0]), .y(level1[i]));
for (i = 0; i < 4; i = i + 1)
mux2x1 m2 (.a(level1[2*i]), .b(level1[2*i+1]), .sel(sel[1]), .y(level2[i]));
for (i = 0; i < 2; i = i + 1)
mux2x1 m3 (.a(level2[2*i]), .b(level2[2*i+1]), .sel(sel[2]), .y(level3[i]));
mux2x1 m4 (.a(level3[0]), .b(level3[1]), .sel(sel[3]), .y(out));
endgenerate
endmodule
```

Here in this code we are looping and stacking the 2x1 muxes in order to achieve a 16x1. We first go from 16 inputs to 8 by cycling through the mux 8 times, each time selecting between 2 inputs which after going through the loop will leave us with 8. Then with those 8 we go through another loop again and repeat, this time being left with 4, and then again we do another level looping twice to be left with 2, and the final level we don't need to loop because we only have 2 inputs left and that's how we select the single output, making a 16x1 mux.

Another important part not specifically related to the MUX concept is making sure our button presses are registered, since unlike a switch they are only active for a short period of time. We need a debounce function to ensure our button inputs are correctly read by using the clock timing to store a 0 or 1 based on a button being pressed. This way only correctly read button pushes will be registered and if the press is too short or for some reason not read correctly it will be ignored.

```
module debounce (
input clk,
input btn_in,
output reg btn_clean
);
reg [2:0] shift_reg;
always @(posedge clk) begin
shift_reg <= {shift_reg[1:0], btn_in};
if (shift_reg == 3'b111) btn_clean <= 1;
else if (shift_reg == 3'b000) btn_clean <= 0;
```

```
end
endmodule
```

With a clean button reading now we need to use a flip flop to store that button press, otherwise it will keep resetting and only register a select for a clock cycle. The given code was:

```
module toggle_switch (
input clk,
input rst,
input btn_raw,
output reg state
);
wire btn_clean;
reg btn_prev;
debounce db (.clk(clk), .btn_in(btn_raw), .btn_clean(btn_clean));
always @(posedge clk) begin
if (rst) begin
state <= 0;
btn_prev <= 0;
end else begin
 if (btn_clean && !btn_prev)
state <= ~state;
btn_prev <= btn_clean;
end
end
endmodule
```

Here we can use the debounce function we created and create a toggle module, where we will store the button pressed until either the button is pressed again or the reset button is pushed. This way we can actually select without the button press immediately clearing. We can also select multiple buttons at once this way, since they'll be stored.

The final piece of the given code was the Top-Level Module:

```
module top_mux_lab3 (
input clk,
input rst,
input [15:0] SW,
input btnU, btnD, btnL, btnR,
output LED0
);
wire [3:0] sel;
toggle_switch t0 (.clk(clk), .rst(rst), .btn_raw(btnD), .state(sel[0]));
toggle_switch t1 (.clk(clk), .rst(rst), .btn_raw(btnR), .state(sel[1]));
toggle_switch t2 (.clk(clk), .rst(rst), .btn_raw(btnL), .state(sel[2]));
```

```
toggle_switch t3 (.clk(clk), .rst(rst), .btn_raw(btnU), .state(sel[3]));
mux16x1 mux (.in(SW), .sel(sel), .out(LED0));
endmodule
```

Here this module ties the button and mux parts together, where the module determines the buttons pushed and toggles them, then passes the output of sel[x] to the mux16x1 module in order to actually do the 16x1 reduction. Now that we have the code loaded in to our project and we understand what was happening in the code, we built a test bench and programmed the board.

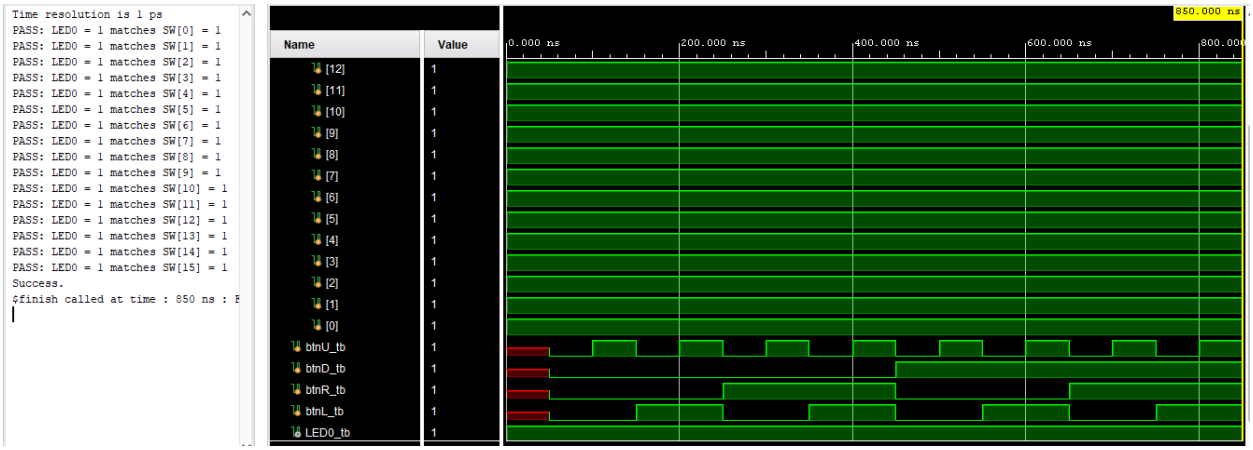
Our testbench code:

```
module lab3_tb(

    );

    reg rst_tb;
    reg clk_tb;
    reg [15:0] SW_tb;
    reg btnU_tb;
    reg btnD_tb;
    reg btnR_tb;
    reg btnL_tb;
    wire LED0_tb;

    initial
        begin
            clk_tb = 1;
        end

     always
        begin
            #5 clk_tb = clk_tb + 5;
        end

            top_mux_lab3 X (
    .clk(clk_tb),
    .rst(rst_tb),
    .SW(SW_tb),
    .btnU(btnU_tb),
    .btnD(btnD_tb),
    .btnR(btnR_tb),
    .btnL(btnL_tb),
    .LED0(LED0_tb)
    );
```

```
initial
  begin
    rst_tb = 1'b1;
    SW_tb = 16'b1111_1111_1111_1111;
  #50
    rst_tb = 1'b0;
    btnD_tb = 1'b0;
    btnR_tb = 1'b0;
    btnL_tb = 1'b0;
    btnU_tb = 1'b0;
    if (LED0_tb === SW_tb[0])
    $display("PASS: LED0 = %b matches SW[0] = %b", LED0_tb, SW_tb[0]);
  #50
    btnD_tb = 1'b0;
    btnR_tb = 1'b0;
    btnL_tb = 1'b0;
    btnU_tb = 1'b1;
    if (LED0_tb === SW_tb[1])
    $display("PASS: LED0 = %b matches SW[1] = %b", LED0_tb, SW_tb[1]);
  #50
    btnD_tb = 1'b0;
    btnR_tb = 1'b0;
    btnL_tb = 1'b1;
    btnU_tb = 1'b0;
    if (LED0_tb === SW_tb[2])
    $display("PASS: LED0 = %b matches SW[2] = %b", LED0_tb, SW_tb[2]);
  #50
    btnD_tb = 1'b0;
    btnR_tb = 1'b0;
    btnL_tb = 1'b1;
    btnU_tb = 1'b1;
    if (LED0_tb === SW_tb[3])
    $display("PASS: LED0 = %b matches SW[3] = %b", LED0_tb, SW_tb[3]);
  #50
    btnD_tb = 1'b0;
    btnR_tb = 1'b1;
    btnL_tb = 1'b0;
    btnU_tb = 1'b0;
    if (LED0_tb === SW_tb[4])
    $display("PASS: LED0 = %b matches SW[4] = %b", LED0_tb, SW_tb[4]);
  #50
    btnD_tb = 1'b0;
    btnR_tb = 1'b1;
```

```
    btnL_tb = 1'b0;
    btnU_tb = 1'b1;
    if (LED0_tb === SW_tb[5])
    $display("PASS: LED0 = %b matches SW[5] = %b", LED0_tb, SW_tb[5]);
#50
    btnD_tb = 1'b0;
    btnR_tb = 1'b1;
    btnL_tb = 1'b1;
    btnU_tb = 1'b0;
    if (LED0_tb === SW_tb[6])
    $display("PASS: LED0 = %b matches SW[6] = %b", LED0_tb, SW_tb[6]);
#50
    btnD_tb = 1'b0;
    btnR_tb = 1'b1;
    btnL_tb = 1'b1;
    btnU_tb = 1'b1;
    if (LED0_tb === SW_tb[7])
    $display("PASS: LED0 = %b matches SW[7] = %b", LED0_tb, SW_tb[7]);
#50
    btnD_tb = 1'b1;
    btnR_tb = 1'b0;
    btnL_tb = 1'b0;
    btnU_tb = 1'b0;
    if (LED0_tb === SW_tb[8])
    $display("PASS: LED0 = %b matches SW[8] = %b", LED0_tb, SW_tb[8]);
#50
    btnD_tb = 1'b1;
    btnR_tb = 1'b0;
    btnL_tb = 1'b0;
    btnU_tb = 1'b1;
    if (LED0_tb === SW_tb[9])
    $display("PASS: LED0 = %b matches SW[9] = %b", LED0_tb, SW_tb[9]);
#50
    btnD_tb = 1'b1;
    btnR_tb = 1'b0;
    btnL_tb = 1'b1;
    btnU_tb = 1'b0;
    if (LED0_tb === SW_tb[10])
    $display("PASS: LED0 = %b matches SW[10] = %b", LED0_tb, SW_tb[10]);
#50
    btnD_tb = 1'b1;
    btnR_tb = 1'b0;
    btnL_tb = 1'b1;
    btnU_tb = 1'b1;
```

```
        if (LED0_tb === SW_tb[11])
        $display("PASS: LED0 = %b matches SW[11] = %b", LED0_tb, SW_tb[11]);
    #50
        btnD_tb = 1'b1;
        btnR_tb = 1'b1;
        btnL_tb = 1'b0;
        btnU_tb = 1'b0;
        if (LED0_tb === SW_tb[12])
        $display("PASS: LED0 = %b matches SW[12] = %b", LED0_tb, SW_tb[12]);
    #50
        btnD_tb = 1'b1;
        btnR_tb = 1'b1;
        btnL_tb = 1'b0;
        btnU_tb = 1'b1;
        if (LED0_tb === SW_tb[13])
        $display("PASS: LED0 = %b matches SW[13] = %b", LED0_tb, SW_tb[13]);
    #50
        btnD_tb = 1'b1;
        btnR_tb = 1'b1;
        btnL_tb = 1'b1;
        btnU_tb = 1'b0;
        if (LED0_tb === SW_tb[14])
        $display("PASS: LED0 = %b matches SW[14] = %b", LED0_tb, SW_tb[14]);
    #50
        btnD_tb = 1'b1;
        btnR_tb = 1'b1;
        btnL_tb = 1'b1;
        btnU_tb = 1'b1;
        if (LED0_tb === SW_tb[15])
        $display("PASS: LED0 = %b matches SW[15] = %b", LED0_tb, SW_tb[15]);
    #50
        $display("Success.");
        $finish;
    end

endmodule
```



In our testbench we go through and simulate the buttons being pressed and check if the correct output is actually displaying. We can see in our console that the different tests passed as the expected output LED0 matched what switch should have been selected based on the buttons pressed.

Some of our project data:

LUTS and Flip Flops:

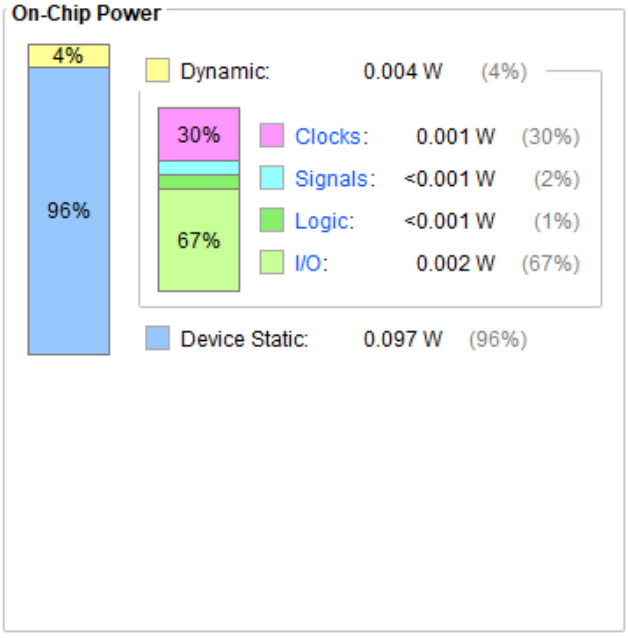
Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
top_mux_lab3	12	24	2	1	23	1

Power Report:

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.101 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (12.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



The work on this lab between us was 50/50. We worked together to figure out different issues such as errors causing the bitstream not to generate or issues with our test bench code. The work was split up evenly and we wrote the report together as well on a joint document.