LAB 3
Switches to turn on LEDs on FPGA board

By Nicholas Williams and Faris Khan

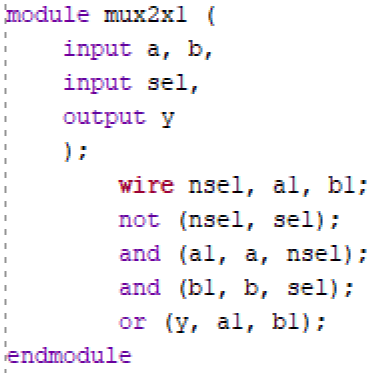
ECE3300

7/2//2025

Introduction

In this lab we were tasked to create a 16 x 1 multiplexer using gate-level 2x1 multiplexer in verilog. The method we used to give our inputs came from our buttons which functioned as toggle switches. 4 buttons made a 4 digit binary number which could get 0 to 15 values. This allowed us to give the input to the 16 x 1 multiplexer. We also had to use the debouncing function due to when a button is pressed it is counted more than once for a few milliseconds which isn't what we want for this lab.

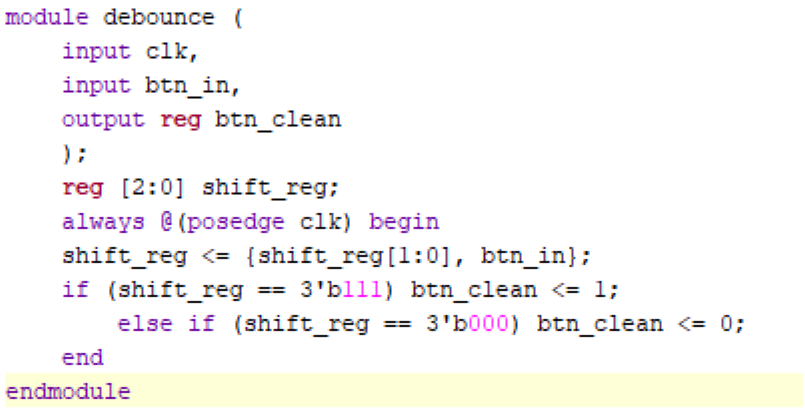
Mux2x1 Module:



```
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
```

Code provided by the professor which defines the logic of our nested 2x1 muxes.

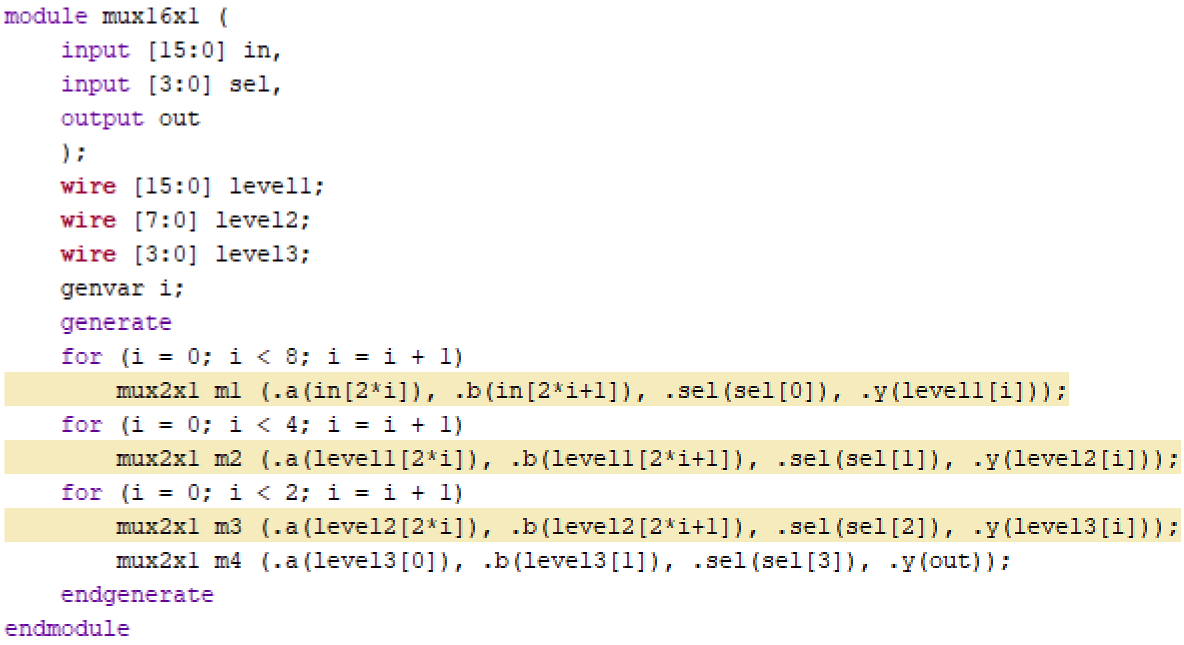
Debounce Module:



```
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
```

In order to filter the signals to get rid of any unnecessary triggers we implement a debounce module for this. Code provided by the professor.

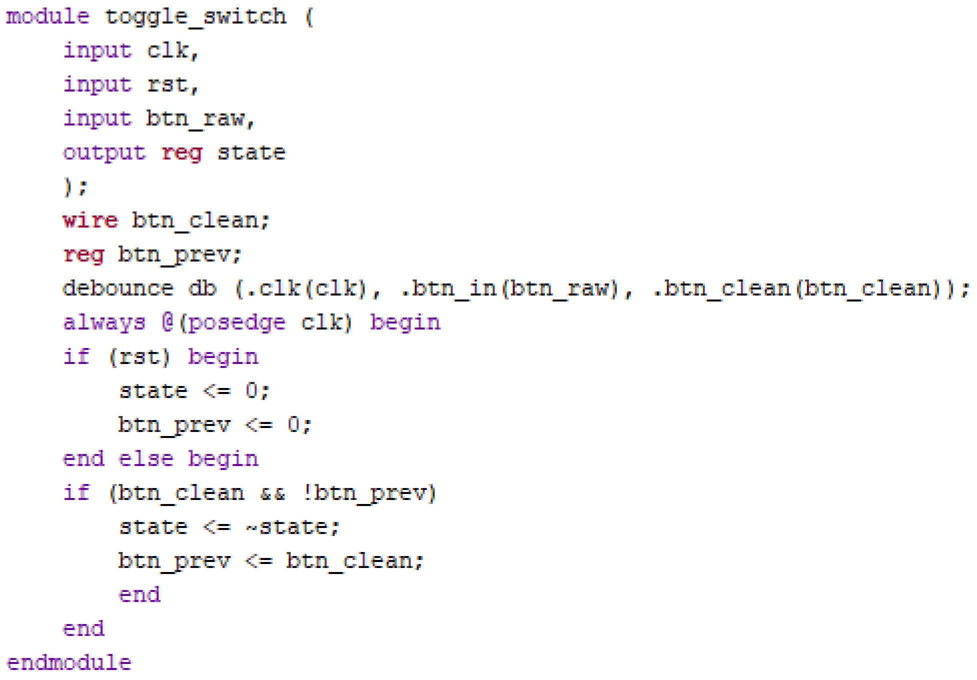
16x1 Mux Module:



```
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
```

Code for the 16x1 mux provided by the professor. Defines a multiplexer with 16 inputs and 4 select bits with the nested 2x1 muxes.

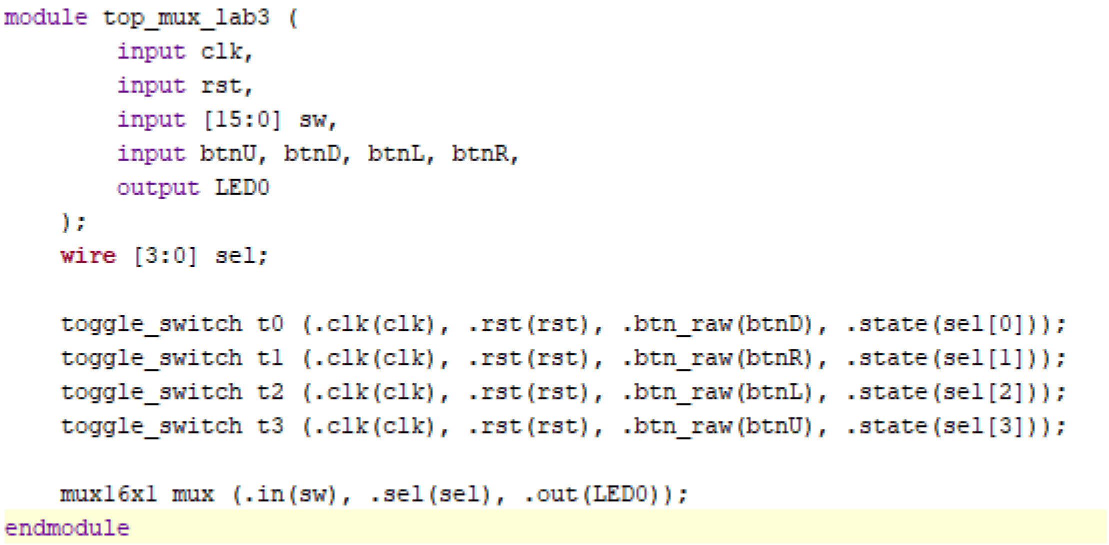
Toggle_switch Module:



```
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
```

The toggle_switch module implements flipping the output state each time a button is pressed. It uses a debounce module to filter out noise from the raw button input (btn_raw). The logic detects a rising edge on the clean signal (btn_clean) to trigger the toggle. On reset, the state is initialized to 0, and the button press history is cleared.

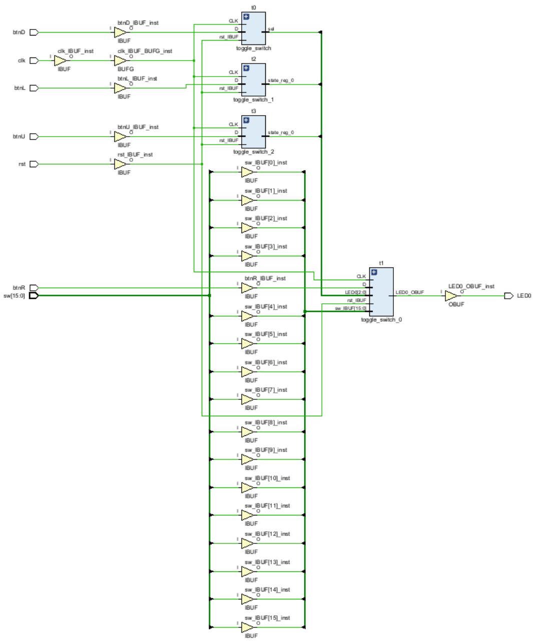
Top-level Mux Module:

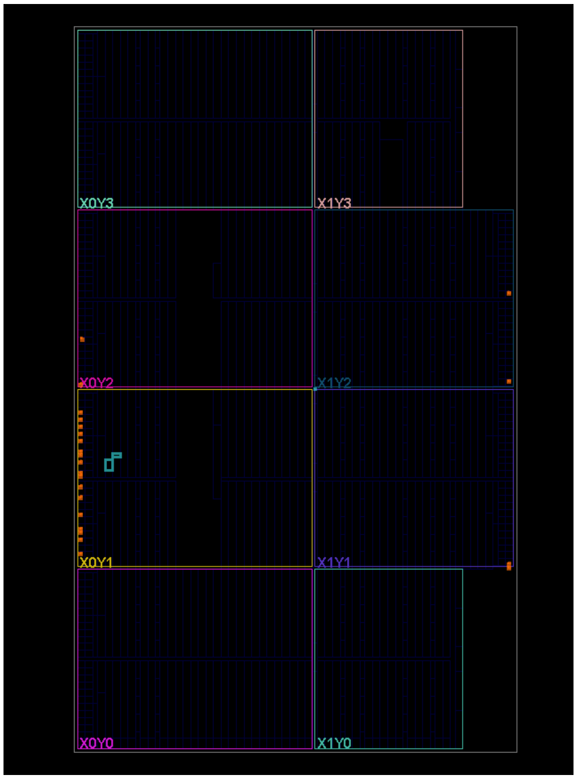


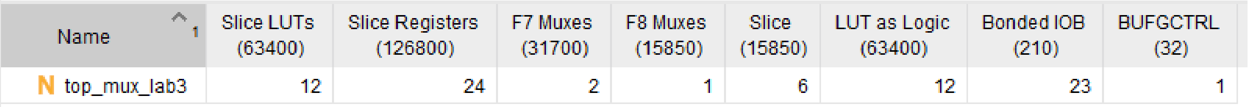
```
module top_mux_lab3 (
        input clk,
        input rst,
        input [15:0] sw,
        input btnU, btnD, btnL, btnR,
        output LED0
    );
    wire [3:0] sel;

    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
```

The top_mux_lab3 module integrates four toggle_switch instances to control the 4-bit select input sel of a mux16x1. Each push-button is wired to a toggle switch, which toggles one of the bits of sel[3:0]. The 16-bit input sw is passed into the multiplexer, and the selected bit (based on sel) is output to LED0. This design lets you manually select one of the 16 inputs using debounced toggle buttons to choose the selector bits.

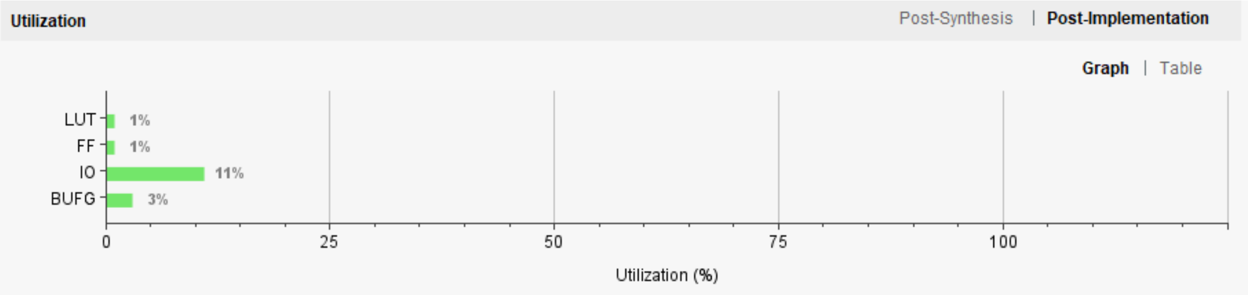
Data





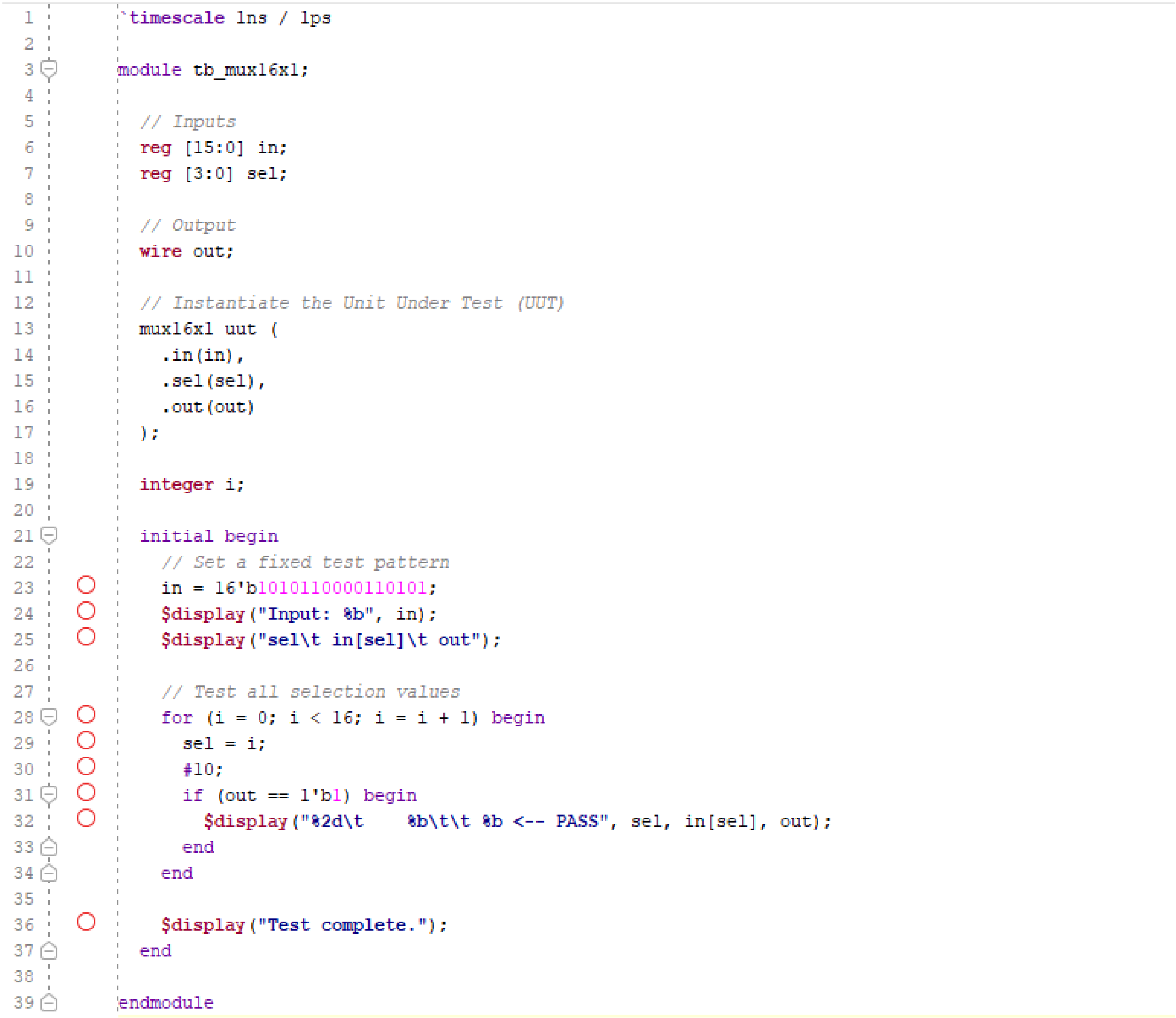


Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
N top_mux_lab3	12	24	2	1	6	12	23	1



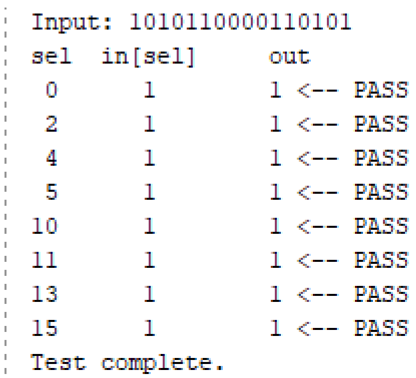
Data Analysis

Testbench



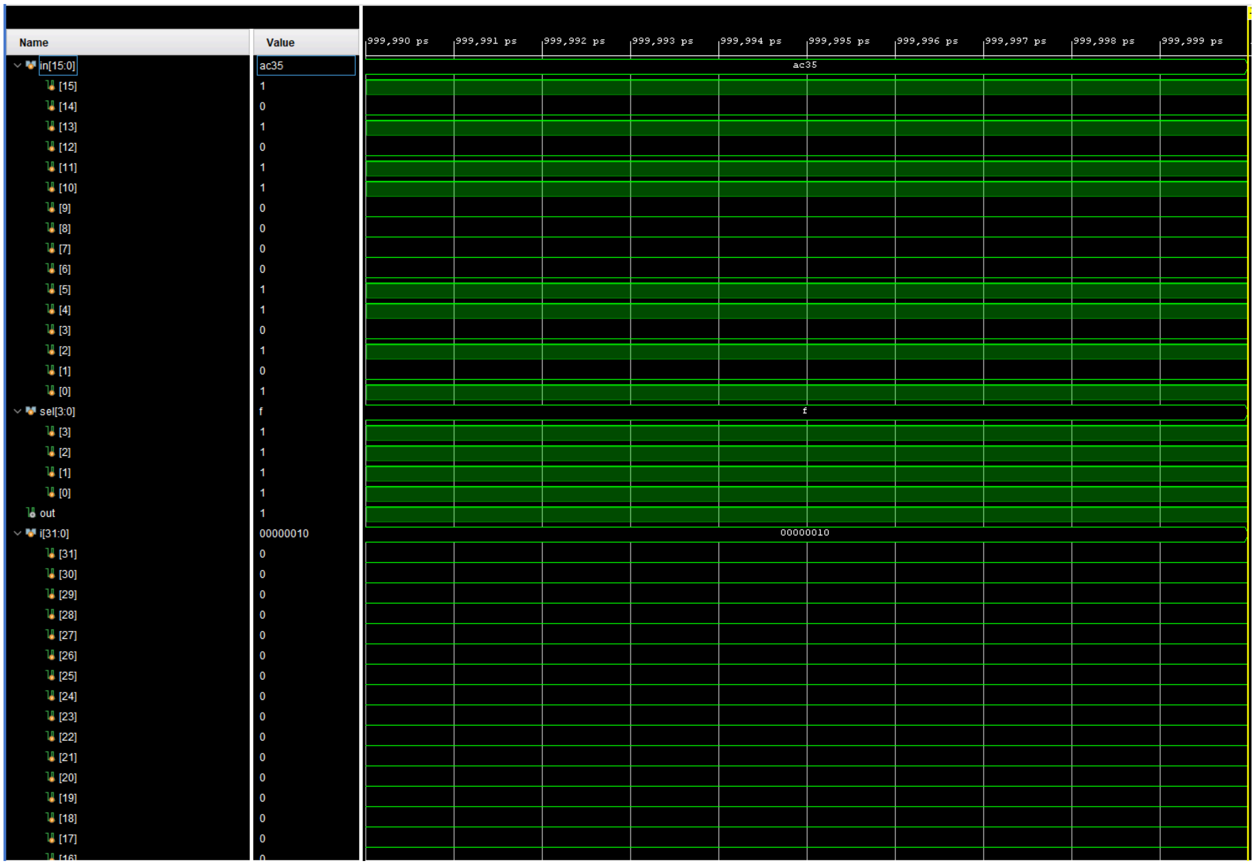
```verilog
`timescale 1ns / 1ps

module tb_mux16x1;

  // Inputs
  reg [15:0] in;
  reg [3:0] sel;

  // Output
  wire out;

  // Instantiate the Unit Under Test (UUT)
  mux16x1 uut (
    .in(in),
    .sel(sel),
    .out(out)
  );

  integer i;

  initial begin
    // Set a fixed test pattern
    in = 16'b1010110000110101;
    $display("Input: %b", in);
    $display("sel\t in[sel]\t out");

    // Test all selection values
    for (i = 0; i < 16; i = i + 1) begin
      sel = i;
      #10;
      if (out == 1'b1) begin
        $display("%2d\t    %b\t\t %b <-- PASS", sel, in[sel], out);
      end
    end

    $display("Test complete.");
  end

endmodule
```

In this an input is defined and we are comparing each bit of the value. Based on the position of the 1s, it will output: the select bit, the value at the select indice, and the output of the led which will be 1 if it passes.



```
Input: 1010110000110101
sel  in[sel]     out
 0      1        1 <-- PASS
 2      1        1 <-- PASS
 4      1        1 <-- PASS
 5      1        1 <-- PASS
10      1        1 <-- PASS
11      1        1 <-- PASS
13      1        1 <-- PASS
15      1        1 <-- PASS
Test complete.
```

Here is the screenshot of the output.



Name	Value
in[15:0]	ac35
[15]	1
[14]	0
[13]	1
[12]	0
[11]	1
[10]	1
[9]	0
[8]	0
[7]	0
[6]	0
[5]	1
[4]	1
[3]	0
[2]	1
[1]	0
[0]	1
sel[3:0]	f
[3]	1
[2]	1
[1]	1
[0]	1
out	1
i[31:0]	00000010
[31]	0
[30]	0
[29]	0
[28]	0
[27]	0
[26]	0
[25]	0
[24]	0
[23]	0
[22]	0
[21]	0
[20]	0
[19]	0
[18]	0
[17]	0
[16]	0

Here is the waveform screenshot.

Conclusion

In this lab we made a working 16x1 multiplexer using 2x1 multiplexers in verilog which combined to make one big 16x1 multiplexer. The switches ran whether the input was hi or low and the led was the output which showed the hi or low status. We used the debouncing function to be able to use the switches to drive the 4 bit selector for the 16x1 mux. In this lab

Contributions

Nicholas Williams- 50% uploaded youtube video, helped make the Testbench file, showed demo, introduction on lab report and conclusion, help with data screenshots.

Faris Khan- 50% Worked mainly on the testbench file with the simulations, included the screenshots of the code and added explanations for each of them.