ECE 3300L

Lab Report #3

Group A

Edwin Estrada (#015897050)

Michelle Lau (#016027427)

July 2, 2025

Design

2x1 Mux 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
```

Rudimentary 2x1 module. If sel is high, the state of b is output. If sel is low, the state of a is output.

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
```

To make a 16x1 mux using only 2x1 muxes, we start with 8 2x1 muxes being selected by the first select bit. The output of those 8 muxes are then connected to 4 more 2x1 muxes which are selected by the second select bit. The output of those 4 muxes are connected to 2 more 2x1 muxes which are selected by the third select bit. Finally, the output of those two muxes are connected to one more mux, selected by the last bit. The output of the last mux is the output of the 16x1 mux. We expedite this process by using the generated block and looping.

Debounce Module

```
module debounce (
input clk,
input btn_in,
output reg btn_clean
);
    reg [2:0] shift_reg;

    initial begin
        shift_reg = 3'b000;
    end

    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
```

The main issue with using buttons for input is the output of a button can vary wildly while in the process of pressing it. In other words, the output can jump between high and low many times even if you pressed the button once. To fix this, we introduce a debounce module which makes sure the output of the button is high for at least 3 clock cycles before outputting a high signal on the next clock cycle. When the output of the button is low (released) for at least 3 clock cycles, the output signal of the button is low.

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 utilizes the debounce module to ensure that there are no unintended triggers. The state of the toggle switches when we get a clean signal from the debounce module (in other words, when we press the button). We also include a btn_prev register so the toggle_switch isn't constantly switching states when we're holding down the button. Once we toggle the switch, btn_prev will stop the state from switching again while holding the button as we release from holding.

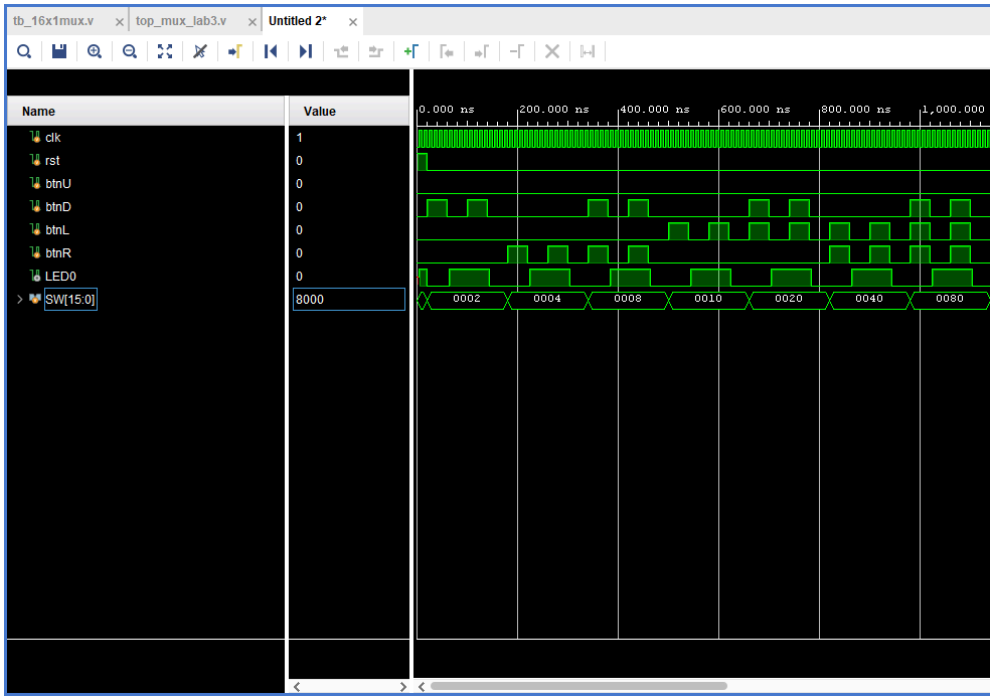
High Level Implementation

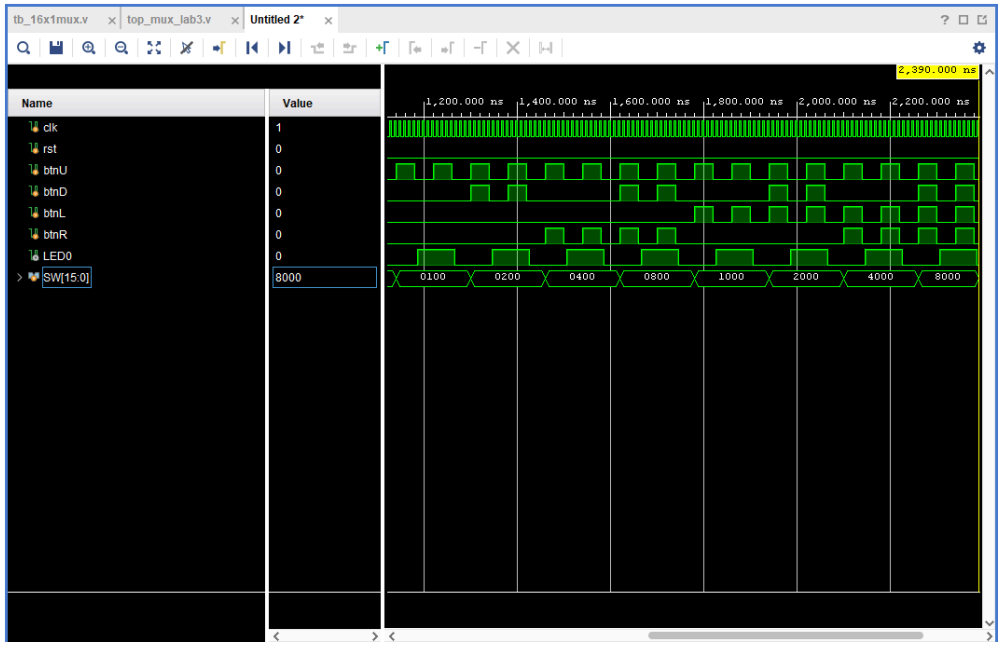
```
module top_mux_lab3 (
    input clk,
    input rst,
    input [15:0] SW,
    input btnU, btnD, btnL, btnR,
    output LED0
);
    wire [3:0] sel;
    toggle_switch t0 (.clk(clk), .rst(rst), .btn_raw(btnD), .state(sel[0]));
    toggle_switch t1 (.clk(clk), .rst(rst), .btn_raw(btnR), .state(sel[1]));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

We implement a toggle_switch module for each button that we intend to use for each bit in our select. That way, we can press its respective button to put that select bit on high and press it again to put that bit on low. We then connect the output of those toggle_switch into our select for the 16x1 mux, and connect our 16 data inputs to our switches. Finally, the output of the mux is connected to an onboard LED.

Simulation Waveform

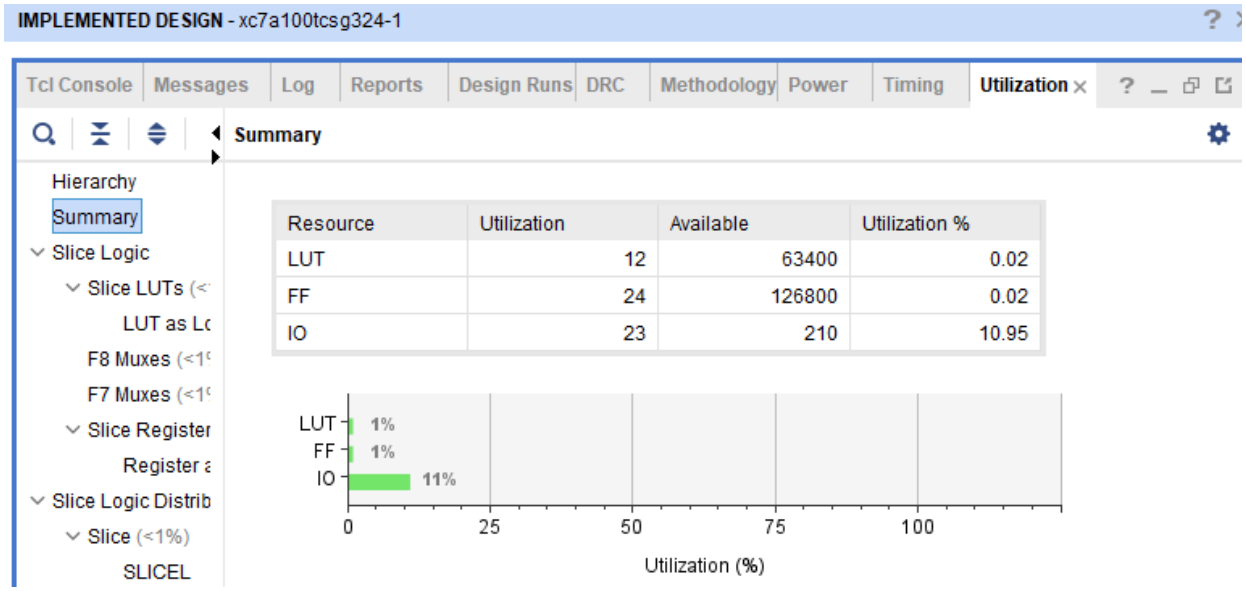
If one of the btn's is high for at least 4 rising clock edges, the sel or its respective button will turn on. To turn off a part of the sel, we "release" the button for 4 clock cycles before "holding" the button for 4 clock cycles again. We start with all bits of select on low with the first switch being turned on. This results in the LED being turned on. We then toggle the down button to put the first bit of the select on high, selecting the second switch, while also turning on the second switch to turn on the LED. As soon as we toggle the down button again to put the first select bit on low, the LED turns off, this process continues for all 15 switches. Simulated below



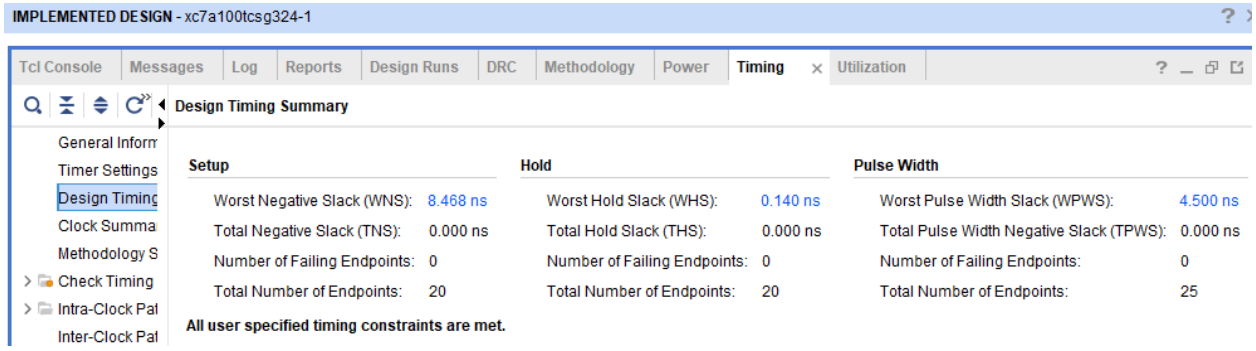


Implementation

Utilization Table:



Timing Summary:



Contribution

Michelle Lau (50%) - Implementation and board demo
Edwin Estrada (50%) - Verilog programming and test benching

Board Demo

https://youtu.be/22VLtozwRRY