Lab 3
3300L.E01

16x1 Multiplexer

Group U
Justin Herrera - Pandarb004
Nathan Rahbar - 9athan

Design

Constraints

For Lab 3, our team implemented a 16x1 multiplexer with nested 2x1 MUXes. We had a constraints file in which we initialized our switches, buttons and LED.

```
##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
```

```
## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
```

```
##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { rst }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { btnU }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { btnL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { btnR }]; #IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { btnD }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

In the constraints, we matched our switch, button, and LED names to what was provided in the lab report.

Debounce

```
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
```

This module filters the noise from button presses. It accepts a dirty raw signal and outputs the cleaned signal from the button. At every positive edge of the clock cycle, this module reads the 3 bits registered from the raw button signal. If every bit is 1, the button is considered pressed and 1 is outputted as the cleaned button signal.

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
```

This 2x1 MUX module is used as the basis for the 16x1 MUX. The output is based on the sel input.

16x1 MUX

```
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
```

This module implements the 2x1 mux at a higher level. This 16x1 MUX utilizes four select bits to choose an output.

Toggle Switch

```
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
```

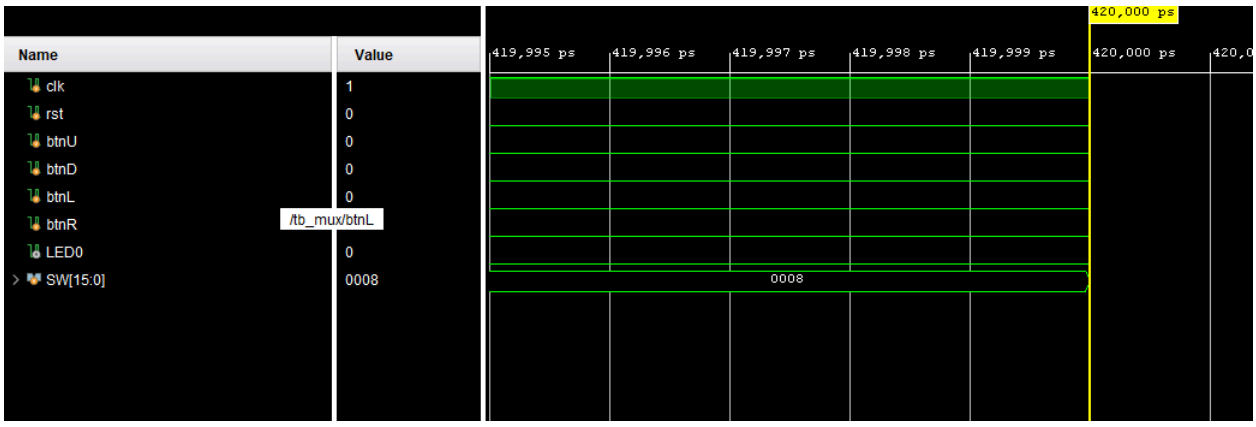
This module aims to implement a toggle switch with the debounced button input. This acts as a switch as in when the button is pressed and released it flips the state between 0 and 1. This module stores the previous button state and compares it against the current. If the previous button state is 0 and the current is 1, this flips the state to 1.

Top 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
```

This is our top module. We use a 4 bit vector named select to determine the output. Each button controls a different select bit which is declared by t0, t1, t2, and t3. This module also allows output to the singular LED.

Testbench



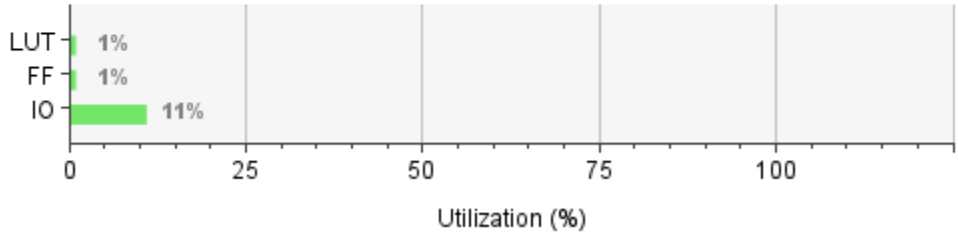
Test Bench Code

```
module tb_mux(
    );
    reg clk, rst, btnU, btnD, btnL, btnR;
    wire LED0;
    reg [15:0] SW;

    top_mux_lab3 tb(
                            .clk(clk),
                            .rst(rst),
                            .btnU(btnU),
                            .btnD(btnD),
                            .btnL(btnL),
                            .btnR(btnR),
                            .SW(SW),
                            .LED0(LED0)
                           );
    initial
        begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial
        begin
        rst = 1;
        #20
        rst = 0;

        SW = 16'b0000000000001000;
        rst = 0;
        btnD = 1;
        btnU = 0;
        btnL = 0;
        btnR = 1;
        #100

        btnD = 0;
        btnR = 0;

        #100
        btnD = 1;
        btnR = 1;

        #100
        btnD = 0;
        btnR = 0;
        #100 $finish;
    end
endmodule
```

Our test bench is used to test the top module.

Implementation

Resource	Utilization	Available	Utilization %
LUT	12	63400	0.02
FF	24	126800	0.02
IO	23	210	10.95



In our implementation, we were able to analyze utilization and see our Flip Flop and LUT counts. This is a simple 16x1 MUX, so utilization was not particularly high. We do see an 11% I/O percentage. This makes sense because we are using 16 switches, 4 buttons and 1 LED which is about 21 I/O signals.

Contributions

Justin - Testbench, Report. - 50%
Nathan - MUX 2x1, 16x1, Toggle Switch, Top Module, Constraints. - 50%

Demo

https://www.youtube.com/watch?v=CMR2qkeVA7A