California Polytechnic State University, Pomona
Department of Electrical and Computer Engineering

Digital Circuit Design Using Verilog Laboratory
ECE 3300L

Lab 3

16x1 Multiplexer Using Nested 2x1 MUXes with Debounced Toggle Select Control

By

Jetts Crittenden (ID# 015468128) and
Evan Tram(#ID 016404570)

7/2/2025

Design:

The design for this project revolves around nesting 2x1 muxes into a 16x1 mux. The code for the 2x1 is set up at the gate level and provides basic 2x1 functions. Then the 16x1 mux is created by instantiating separate layers of the 2x1 muxes using genvar. Each level of the mux divides the selected quantity by half based on the select inputs, sel[0], sel[1], and sel[2]. Then the final output is chosen by sel[3] after the previous stages. This allows us to simulate the 16x1 mux. However, since we are intending to use toggle buttons for the selection switches, we need to integrate an FF for each button that remembers the previous state.

This also requires a debouncing function to prevent the FF's from toggling too many times. This is because push buttons like those on the NEXYS A7 bounce for a few milliseconds, creating unclean signals. We clean these signals by reading the state of the button for a few clock cycles, and if the last three states have been 1, then it sets the clean signal to on. Integrating all of these design principles, we get a 16x1 with proper toggleable selection buttons.

Code:

MUX 2x1:

```
module Mux2x1(
    input a,
    input 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
```

MUX 16x1:

```
module mux16x1(
    input [3:0] sel ,
    input [15:0] in ,
```

```
    output out
    );

    wire [7:0] level1;
    wire [3:0] level2;
    wire [1:0] level3;
        genvar i,j,k;
        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 (j = 0; j < 4; j = j + 1)
            Mux2x1 m2 (.a(level1[2*j]), .b(level1[2*j+1]),
.sel(sel[1]), .y(level2[j]));
            for (k = 0; k < 2; k = k + 1)
            Mux2x1 m3 (.a(level2[2*k]), .b(level2[2*k+1]),
.sel(sel[2]), .y(level3[k]));
            Mux2x1 m4 (.a(level3[0]), .b(level3[1]), .sel(sel[3]),
.y(out));
         endgenerate

endmodule
```

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
```

Debounce:

```
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
```

XDC Snippet:

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {clk}];


##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
```

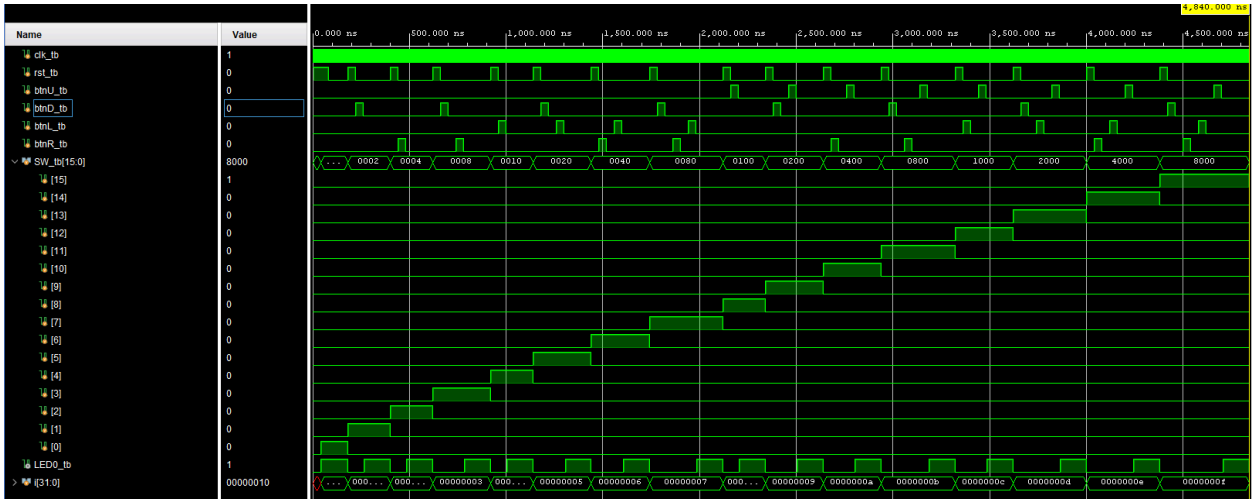
Simulation:

Test Bench Code:

```
module Tb_top_mux;

  reg clk = 0;
  reg [15:0] SW = 0;
  reg reset = 0;
  reg btnU = 0, btnD = 0, btnL = 0, btnR = 0;
  wire LED0;

  top_mux_lab3 DUT (
    .clk(clk),
    .rst(reset),
    .SW(SW),
    .btnU(btnU),
    .btnD(btnD),
    .btnL(btnL),
    .btnR(btnR),
    .LED0(LED0)
  );

  always #5 clk = ~clk;  // 100 MHz clock

  // Button press tasks (40ns pulse)
  task press_btnD(); begin btnD = 1; #40; btnD = 0; #40; end endtask
  task press_btnR(); begin btnR = 1; #40; btnR = 0; #40; end endtask
  task press_btnL(); begin btnL = 1; #40; btnL = 0; #40; end endtask
  task press_btnU(); begin btnU = 1; #40; btnU = 0; #40; end endtask
```

```
  // Select bits by toggling buttons based on value
  task select_value(input [3:0] value);
    begin
      reset = 1; #40; reset = 0;

      if (value[0]) press_btnD();
      if (value[1]) press_btnR();
      if (value[2]) press_btnL();
      if (value[3]) press_btnU();
    end
  endtask

  integer i;

  initial begin
    SW = 16'b0;
    reset = 0;
    btnU = 0; btnD = 0; btnL = 0; btnR = 0;

    for (i = 0; i < 16; i = i + 1) begin
      SW = 16'b1 << i;  // Set only one bit high
      select_value(i[3:0]);
      #100;

      $display("Testing sel=%0d, SW=%b, LED0=%b", i, SW, LED0);
      if (LED0 !== SW[i])
        $fatal(" Mismatch at sel=%0d: expected %b", i, SW[i]);
    end

    $display(" All tests passed.");
    $finish;
  end

endmodule
```

Sample Waveform:



Simulation shows proper outputs for the Mux inputs.

Implementation: Resource utilization table(s):

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	12	0	0	63400	0.02
LUT as Logic	12	0	0	63400	0.02
LUT as Memory	0	0	0	19000	0.00
Slice Registers	24	0	0	126800	0.02
Register as Flip Flop	24	0	0	126800	0.02
Register as Latch	0	0	0	126800	0.00
F7 Muxes	2	0	0	31700	<0.01
F8 Muxes	1	0	0	15850	<0.01

Look-Up Tables: 12 out of 63,400 are used, and all are configured as logic (not memory).
Slice Registers: 24 out of 126,800 are used, all as flip-flops.
Muxes (F7/F8): Used

Total On-Chip Power (W)	0.100
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	0.003
Device Static (W)	0.097
Effective TJA (C/W)	4.6
Max Ambient (C)	84.5
Junction Temperature (C)	25.5
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Total Chip Power: 0.100 Watts (Low)
Dynamic Power: 0.003 Watts (Extremely Low Power variation)
Static Power: 0.097 Watts (Dominant)
Slice Registers: 24 out of 126,800 are used, all as flip-flops.
Muxes (F7/F8): Low temperature with low max.

Timing summary:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.328 ns	Worst Hold Slack (WHS):	0.137 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	20	Total Number of Endpoints:	20	Total Number of Endpoints:	25

All user specified timing constraints are met.

WNS (Worst Negative Slack) 8.468 ns
TNS (Total Negative Slack) 0.000 ns
THS (Total Hold Slack) 0.000 ns
Clock Frequency 100 MHz

Group video link:

https://youtu.be/BAjskOn-4bg?si=-PpDiN8cB6hGoqZf

Contributions:

The contributions to this lab were 50/50. We both continue to work and collaborate on the main development files using the provided resources. The testbench was developed separately by both of us. We then discussed our methods and optimized our testbench into one efficient bench. Jetts Crittenden recorded the YouTube demonstration, and the simulation analysis was performed by Evan Tram. Overall, we contributed to an equal and efficient workload. This allowed both of us to understand and digest the material and the lessons of this lab demonstration.