ECE3300L Lab 3

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

Group X
Czyrone Agbayani (BID: 014766336)
Caleb Jala-Guinto (BID: 015446587)

Objective:

The objective of this lab is to design and implement a 16-to-1 multiplexer (MUX16x1) on the Nexys A7 FPGA board using gate-level 2-to-1 multiplexers in Verilog. The multiplexer selects one of 16 switch inputs (SW\[15:0]) to be displayed on a single LED (LED0), based on a 4-bit select signal. Instead of using simple switches for the select lines, the lab introduces the use of pushbuttons with toggle behavior. To make this possible, students must implement a debouncing system to filter out signal noise and a toggle flip-flop to store each select bit. This lab provides hands-on experience with building hierarchical digital circuits, managing mechanical input issues, and applying real-time logic on FPGA hardware.

MUX2x1 Gate-Level Verilog:

```
1  module mux2x1 (
2      input a, b,
3      input sel,
4      output y
5  );
6      wire nsel, a1, b1;
7      not (nsel, sel);
8      and (a1, a, nsel);
9      and (b1, b, sel);
10     or (y, a1, b1);
11 endmodule
```

This is a simple MUX2x1 module. If sel is high, the output is the state of b, but if sel is low, the output is the state of a. This will help later with making the MUX16x1

MUX16x1 Using Generate Loops:

```
23 module mux16x1(
24     input [15:0] in,
25     input [3:0] sel,
26     output out
27 );
28     wire [15:0] level1;
29     wire [7:0] level2;
30     wire [3:0] level3;
31
32     genvar i;
33     generate
34         for (i = 0; i < 8; i = i + 1)
35             mux2x1 m1 (.a(in[2*i]), .b(in[2*i+1]), .sel(sel[0]), .y(level1[i]));
36
37         for (i = 0; i < 4; i = i + 1)
38             mux2x1 m2 (.a(level1[2*i]), .b(level1[2*i+1]), .sel(sel[1]), .y(level2[i]));
39
40         for (i = 0; i < 2; i = i + 1)
41             mux2x1 m3 (.a(level2[2*i]), .b(level2[2*i+1]), .sel(sel[2]), .y(level3[i]));
42
43         mux2x1 m4 (.a(level3[0]), .b(level3[1]), .sel(sel[3]), .y(out));
44     endgenerate
45 endmodule
```

This module uses eight 2x1 MUXes to make a 16x1 MUX. It uses the 2x1 MUXes to select the different levels of bits. It starts by using eight MUXes to connect to four MUXes, then those four MUXes connect to two MUXes, and finally those two MUXes connect to one MUX, which determines the output of the 16x1 MUX. This process is seen through the generate block and the for loops.

Debounce Module:

```
23  module debounce (
24      input clk,
25      input btn_in,
26      output reg btn_clean
27  );
28      reg [2:0] shift_reg;
29
30      always @(posedge clk) begin
31          shift_reg <= {shift_reg[1:0], btn_in};
32          if (shift_reg == 3'b111) btn_clean <= 1;
33          else if (shift_reg == 3'b000) btn_clean <= 0;
34      end
35  endmodule
```

This module helps prevent the buttons from jumping from high to low too many times. This module works by making sure the button is high for at least three clock cycles before outputting the signal on the next clock cycle. This goes the same when the button is low.

Toggle Flip-Flop:

```
23  module toggle_switch (
24      input clk,
25      input rst,
26      input btn_raw,
27      output reg state
28  );
29      wire btn_clean;
30      reg btn_prev;
31
32      debounce db (.clk(clk), .btn_in(btn_raw), .btn_clean(btn_clean));
33
34      always @(posedge clk) begin
35          if (rst) begin
36              state <= 0;
37              btn_prev <= 0;
38          end else begin
39              if (btn_clean && !btn_prev)
40                  state <= ~state;
41              btn_prev <= btn_clean;
42          end
43      end
44  endmodule
```

This module uses the debounce module to make sure that there are no unintentional triggers. This works by having the state of the toggle switch when it gets a clean signal from the button. There are also parameters in the code to make sure it isn't constantly switching states whenever the button is held down.

Top-Level Module:

```
23  module top_mux_lab3 (
24      input clk,
25      input rst,
26      input [15:0] SW,
27      input btnU, btnD, btnL, btnR,
28      output LED0
29  );
30      wire [3:0] sel;
31
32      toggle_switch t0 (.clk(clk), .rst(rst), .btn_raw(btnD), .state(sel[0]));
33      toggle_switch t1 (.clk(clk), .rst(rst), .btn_raw(btnR), .state(sel[1]));
34      toggle_switch t2 (.clk(clk), .rst(rst), .btn_raw(btnL), .state(sel[2]));
35      toggle_switch t3 (.clk(clk), .rst(rst), .btn_raw(btnU), .state(sel[3]));
36
37      mux16x1 mux (.in(SW), .sel(sel), .out(LED0));
38  endmodule
```

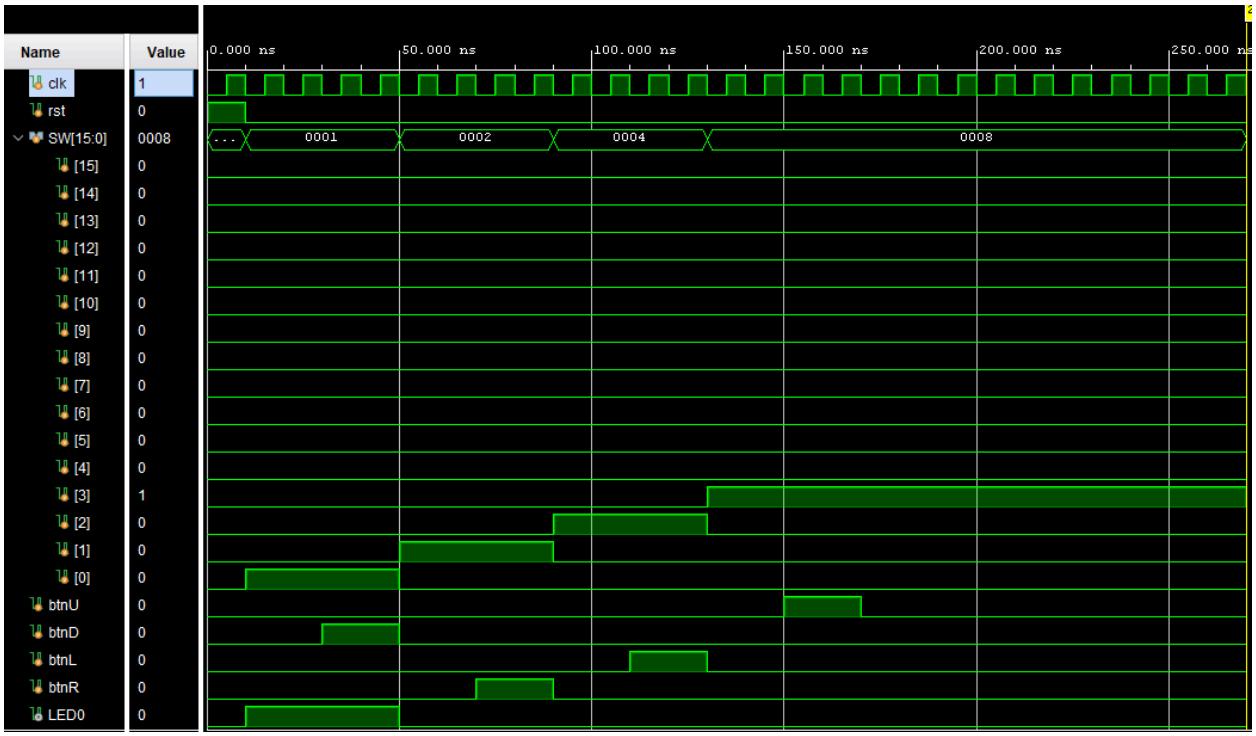
This module implements a toggle_switch method on each button used. The buttons are used to communicate each bit in our select. Their outputs are connected to the MUX16x1, which are connected to the sixteen data inputs to the switches. This then outputs onto LED0.

XDC file:

```
# Clock signal
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
```

Test Bench:



Video Link:
https://youtube.com/shorts/7H1bLrLX0Xw

Contributions:
Czyrone (50%) - Physical Demo, Debugging Code
Caleb (50%) - Debugging Code and Explaining Coding Process
Both worked on the lab report together and troubleshooted code

Reflections:

In this lab, a 16-to-1 multiplexer was built using gate-level 2-to-1 multiplexers in Verilog and controlled by the push buttons on the Nexys A7 FPGA board. To make the push buttons behave like toggle switches, a debouncing system and a toggle flip-flop were implemented for each select bit. This allowed each button press to reliably change the value of one of the four select bits, which controlled which of the 16 switches was routed to the output. The selected input value was displayed in real time on LED0. By combining digital design with input filtering and state-based control, this lab demonstrated how to manage noisy mechanical signals and implement clean, user-driven selection logic using hardware description language and FPGA tools.