ECE 3300L.01 - Lab 3

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

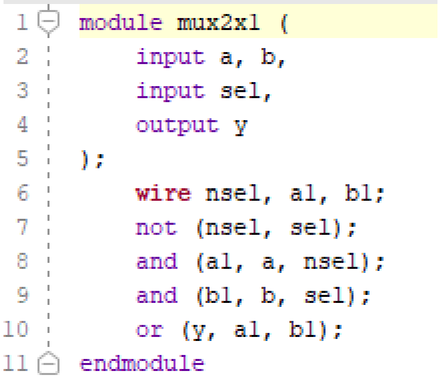
Professor Mohamed Aly
Group Q: Kevin Tang(015429622) &
Jared Mocling(015215057)

July 2nd, 2025

Objective: In this lab, you will design a 16-to-1 multiplexer (MUX16x1) using gate-level 2x1 multiplexers in Verilog. You will control the MUX using the push buttons on the Nexys A7 board, which must behave like switches using toggle logic. This requires implementing a debouncing system and a toggle flip-flop for each select bit. The output of the multiplexer will be shown on LED0, while the inputs will be fed from the 16 switches (SW[15:0])

Code

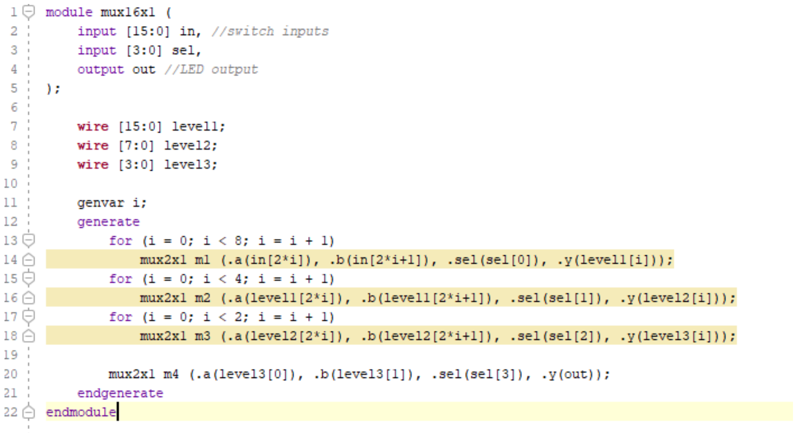
2x1 Mux Module:



```
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
```

Here is our 2x1 module. This module addresses the inputs and outputs of the 2x1 module

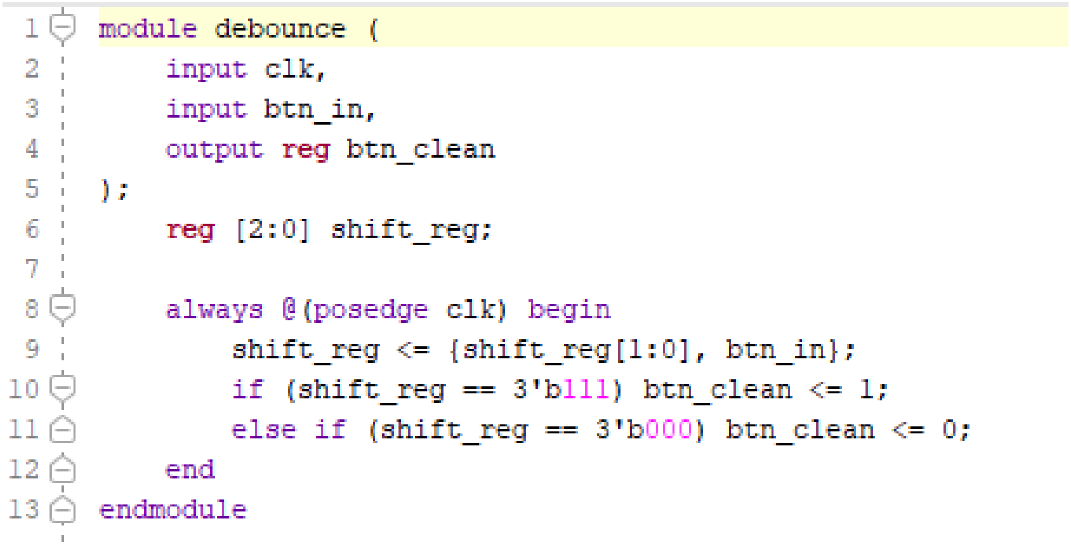
16x1 Mux Module:



```
module mux16x1 (
    input [15:0] in, //switch inputs
    input [3:0] sel,
    output out //LED output
);

    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
```

Using our 2x1 mux module, we create our 16x1 mux by running for-loops for each level of the mux.

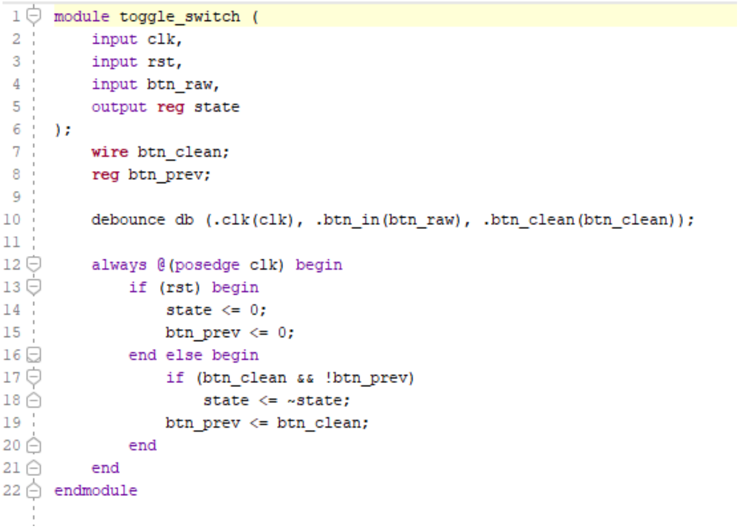
Debounce Module:



```
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
```

Our debounce module is created so that the signal of the button can get a clear number and to get this clean signal btn_clean = 1 when the input signal is at 1111

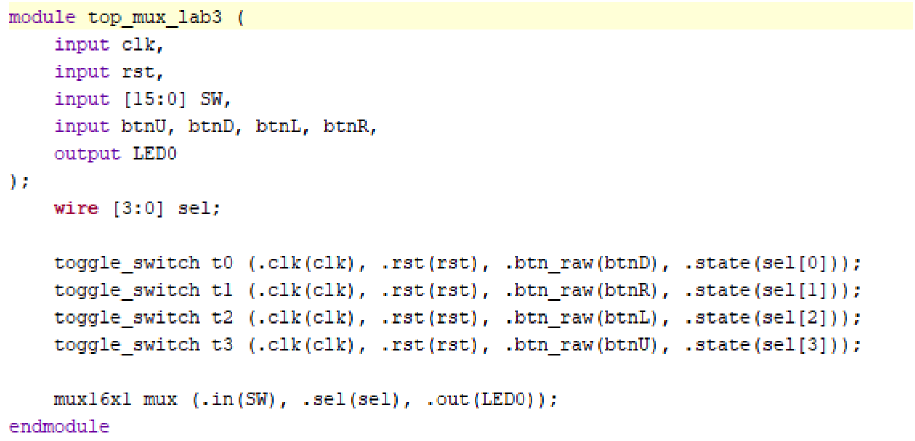
Toggle Switch Module:



```
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
```

Here the toggle switch module uses the debounce module to make sure that one switch is the only trigger. This uses if-else statements to make sure that the toggle on or off is based on which switch is selected.

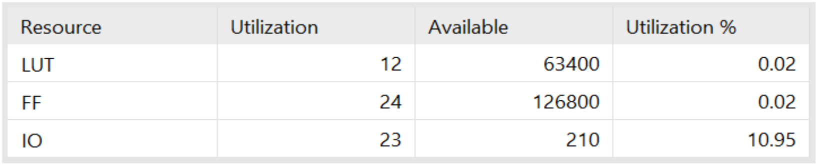
Top-Level Module:



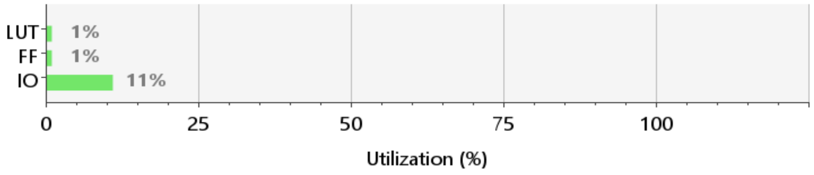
```
module top_mux_lab3 (
    input clk,
    input rst,
    input [15:0] SW,
    input btnU, btnD, btnL, btnR,
    output LED0
);
    wire [3:0] sel;

    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
```

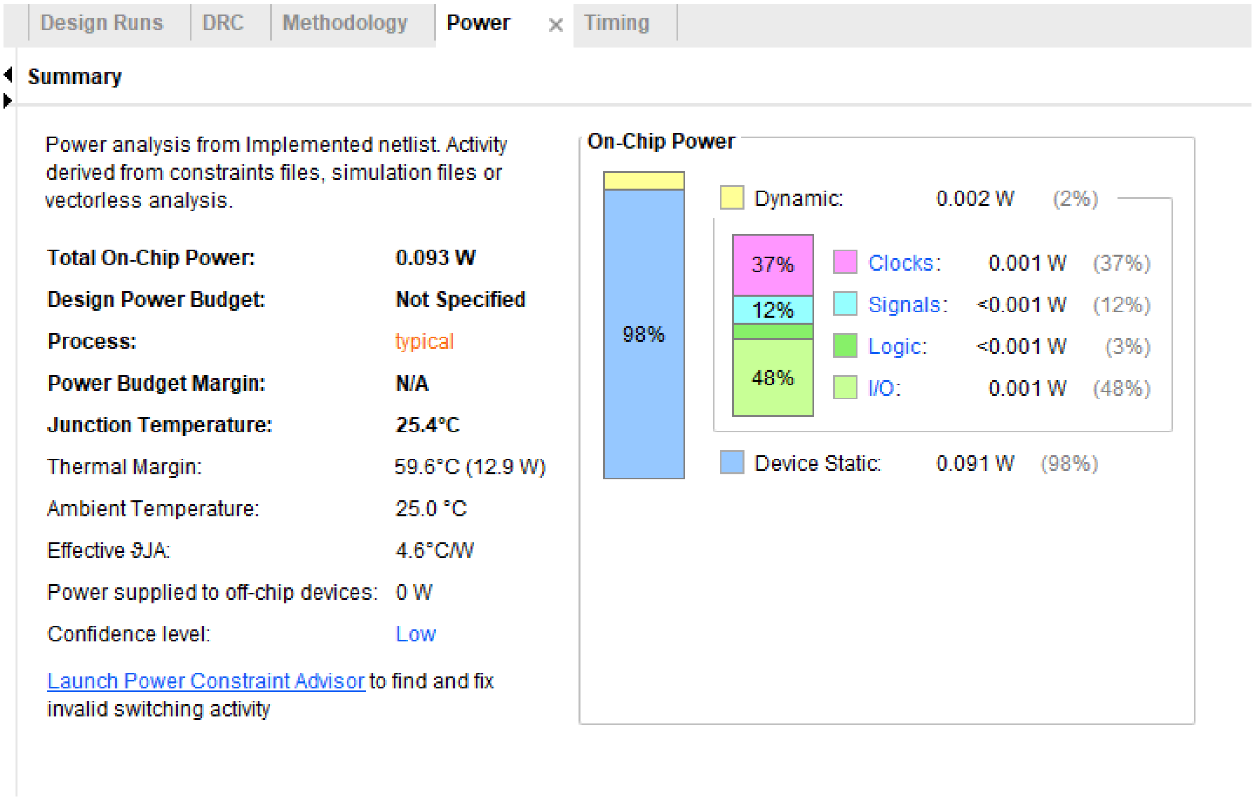
Here this combines all of our modules by using our toggle switch module to verify if the button that is selected is the same as the input lines.

Utilization Table:



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





Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

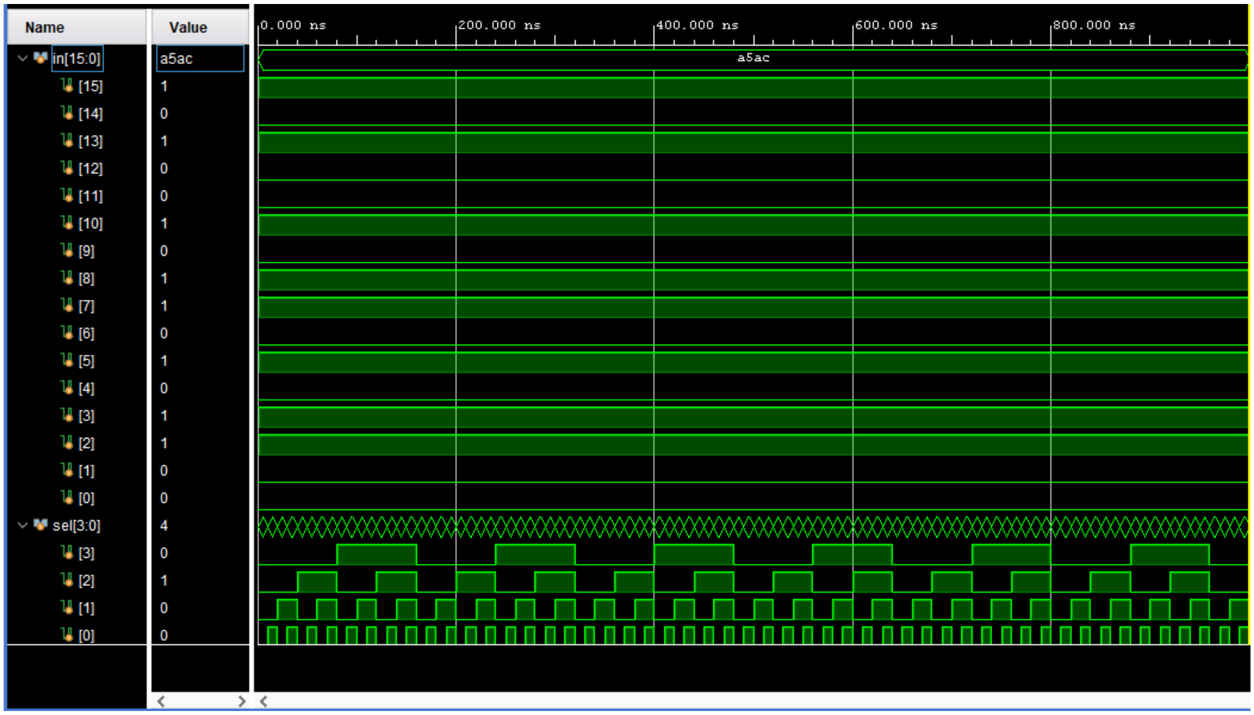
Total On-Chip Power:	**0.093 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (12.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

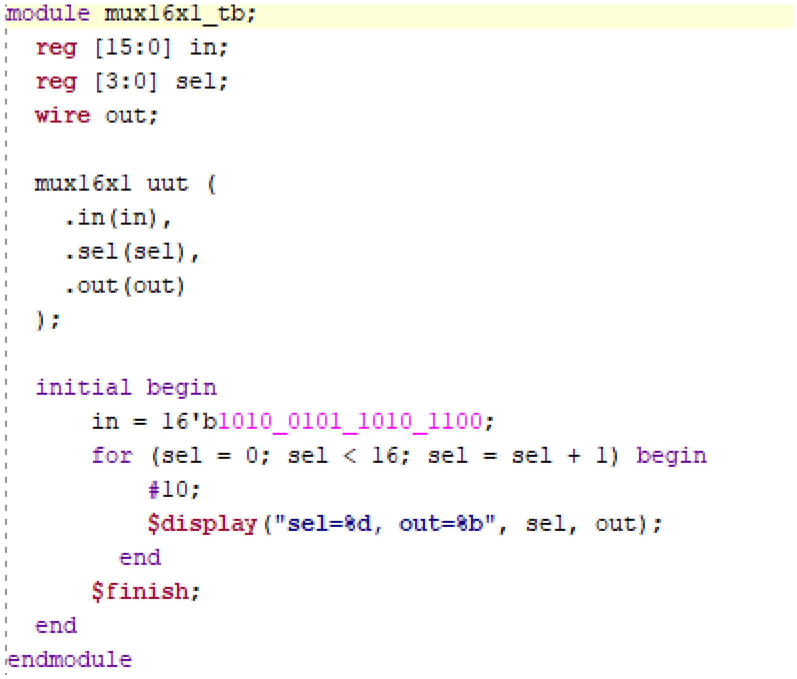
On-Chip Power

	Dynamic:	0.002 W	(2%)
	Clocks:	0.001 W	(37%)
	Signals:	<0.001 W	(12%)
	Logic:	<0.001 W	(3%)
	I/O:	0.001 W	(48%)
	Device Static:	0.091 W	(98%)

Simulation Table



Name	Value
in[15:0]	a5ac
[15]	1
[14]	0
[13]	1
[12]	0
[11]	0
[10]	1
[9]	0
[8]	1
[7]	1
[6]	0
[5]	1
[4]	0
[3]	1
[2]	1
[1]	0
[0]	0
sel[3:0]	4
[3]	0
[2]	1
[1]	0
[0]	0



```
module mux16x1_tb;
  reg [15:0] in;
  reg [3:0] sel;
  wire out;

  mux16x1 uut (
    .in(in),
    .sel(sel),
    .out(out)
  );

  initial begin
      in = 16'b1010_0101_1010_1100;
      for (sel = 0; sel < 16; sel = sel + 1) begin
          #10;
          $display("sel=%d, out=%b", sel, out);
        end
      $finish;
  end
endmodule
```

This testbench tests the 16x1 multiplexor module. We imputed a 16-bit value (1010 0101 1010 1100) and ran bits 0-15 for the sel values. The goal was to test the inputs.

Contributions:

Kevin Tang (50%) - board demo, compiled code
Jared Mocling (50%)- , test bench code, Synthesis reports
We both worked on the lab report together.