California Polytechnic State University Pomona

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Digital Circuit Design Verilog

ECE 3300L

Report #2

Prepared by

Heba Hafez 017353323

Sean Wygant 017376658

Group Y

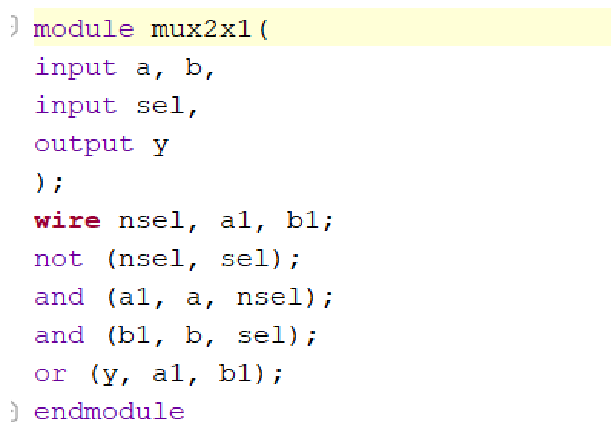
Mohamed Aly

July 1, 2025

Objective: In Verilog, students will design a 16-to-1 Multiplexer (MUX 16x1) using gate-level 2x1 multiplexers. The Nexy's A7 Board pushbuttons will be used to control the MUX with switch life behavior through applying debouncing and toggle flip-flops per bit. The MUX output will be displayed on LED0, with inputs fed from the 16 switches.

Code and Explanation:

<u>Mux 2x1 Gate</u>



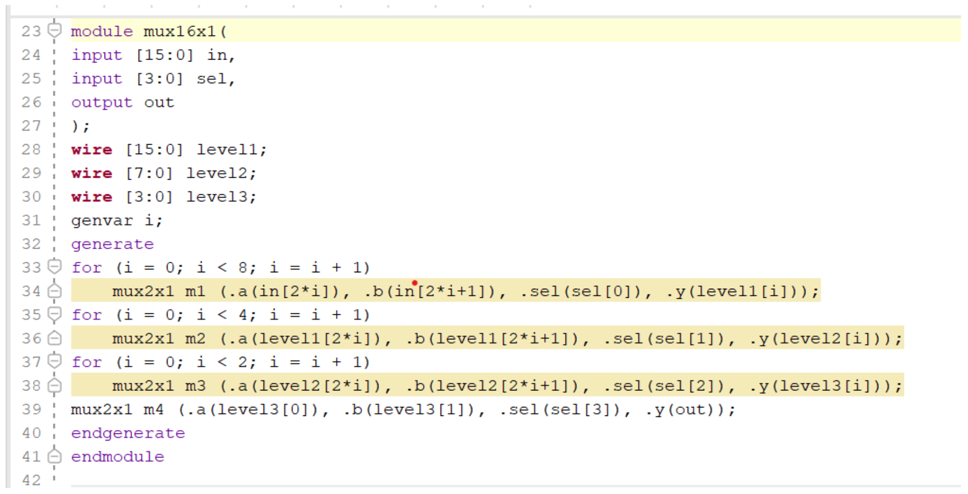
```
module mux2x1(
input a, b,
input sel,
output y
);
wire nsel, a1, b1;
not (nsel, sel);
and (a1, a, nsel);
and (b1, b, sel);
or (y, a1, b1);
endmodule
```

Based on the select (sel) inputs a or b are selected.

Not will invert the select inputs, and will choose a if select is 0 and b if select is 1.

Then the or will give the output.

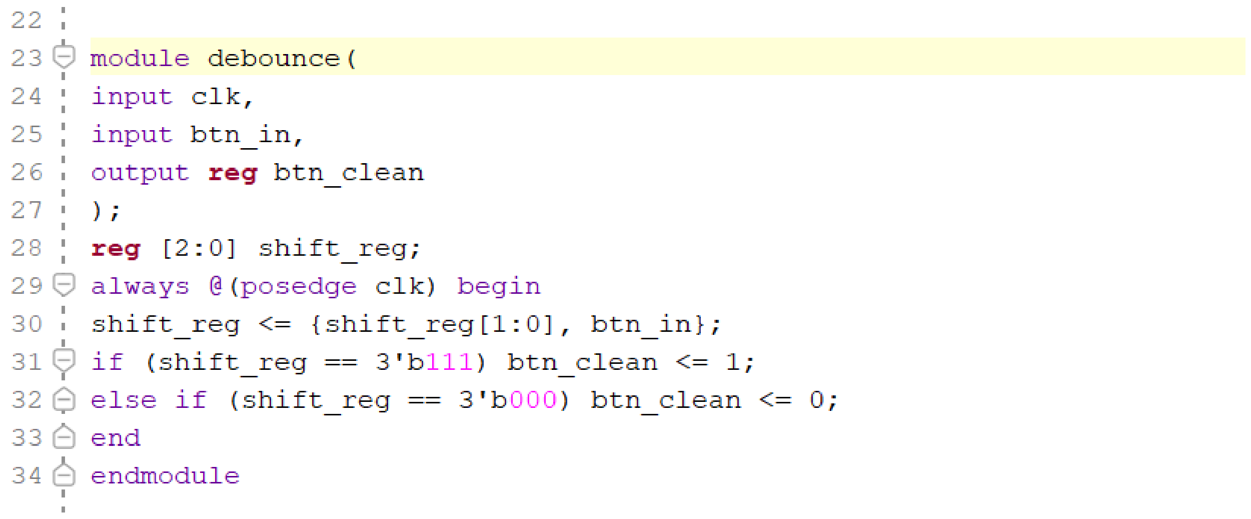
MUX 16x1 using Generate Loops



```
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
```

Using the 2x1 Muxs there's a sequence and loop to create the 16x1 Mux. This calls back to the mux 2x1 module. "genvar i" is used in the loop to instantiate the multiple muxs.

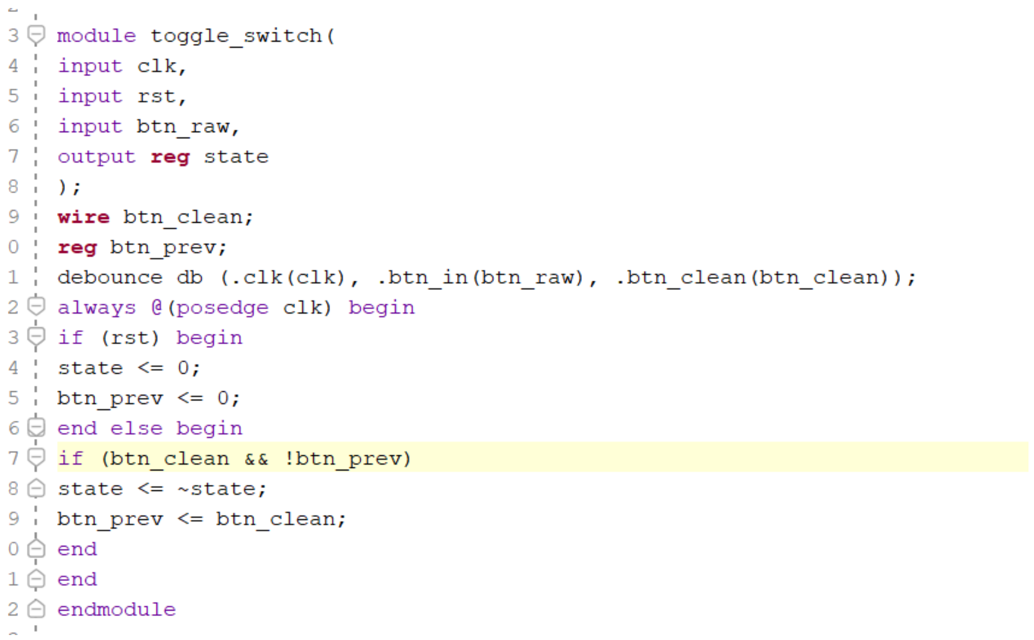
Debounce Module



```
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
```

In order to debounce the noise we filter the circuit by using a 3-bit shift register . Within the if-else statement, it states that if the register is "3'b111" so 111 the button will be on/1 , if it is "3'b000" or 000 it will be 0ff/0.

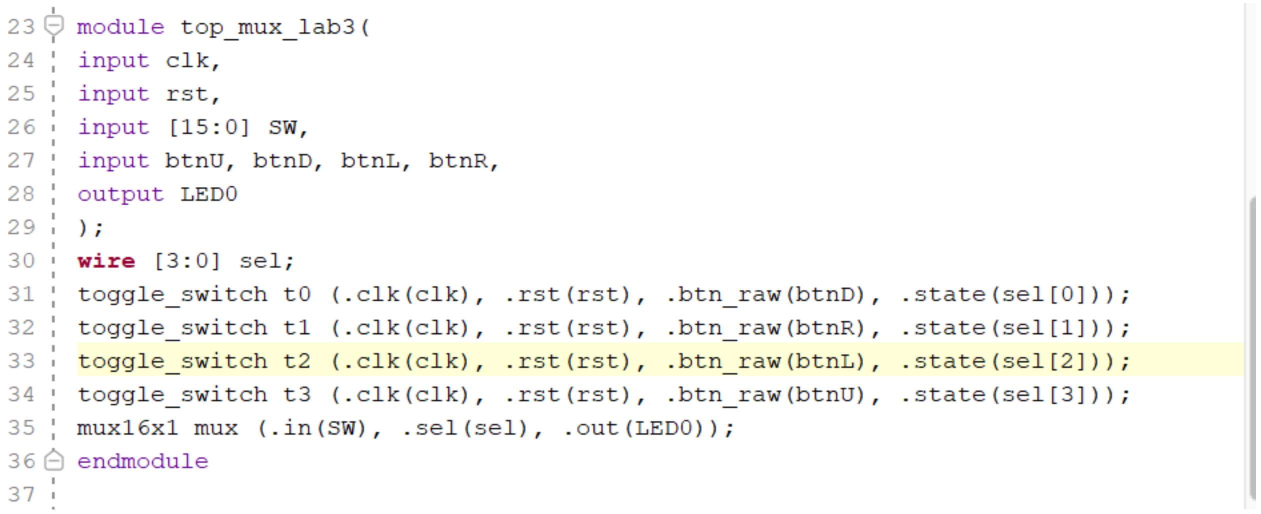
Toggle Flip-Flop



```
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
```

Each time the button is pressed, the state of the button is toggled. This is done by toggling the rising edges (state <= ~ state) of btn_clean and btn_prev then resetting (rst) them to start over.

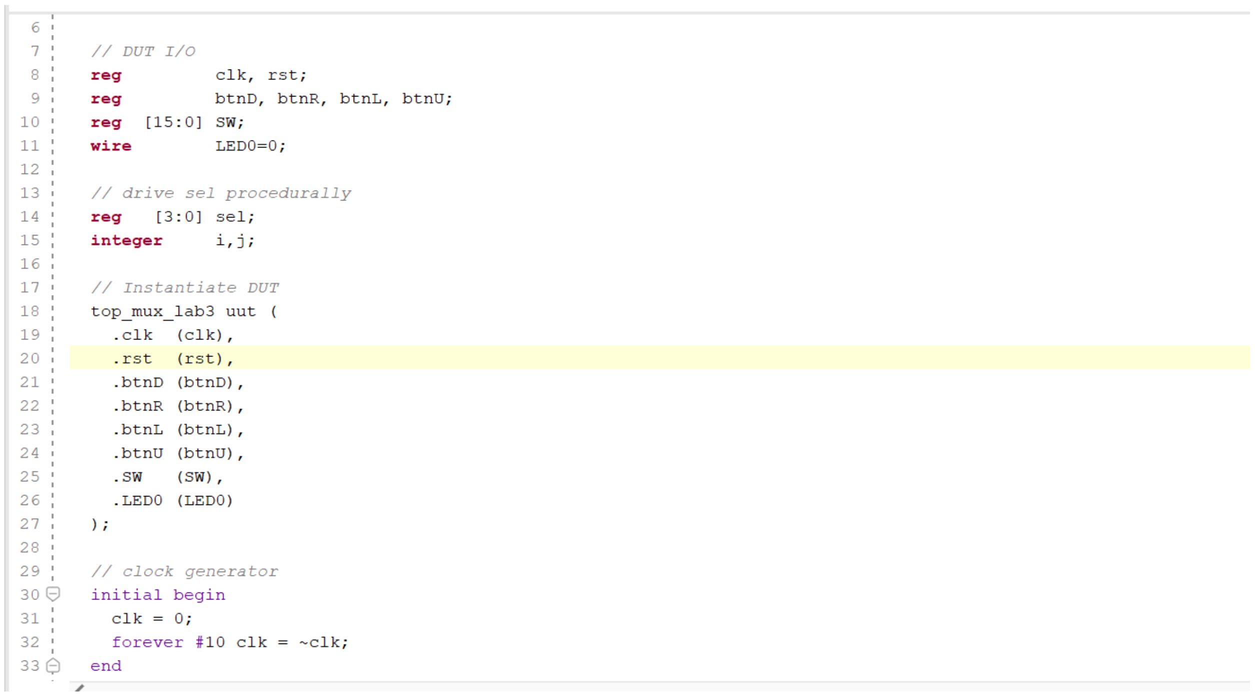
Top-Level Module



```
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
```

The data inputs switch 0-16 of the mux, output led 0, and buttons up, down, left, and right are declared, as well as the clock and reset. Each button toggles one bit of the select signal and drives it to the LED. This calls back to the toggle_switch and debounce modules.

Testbench

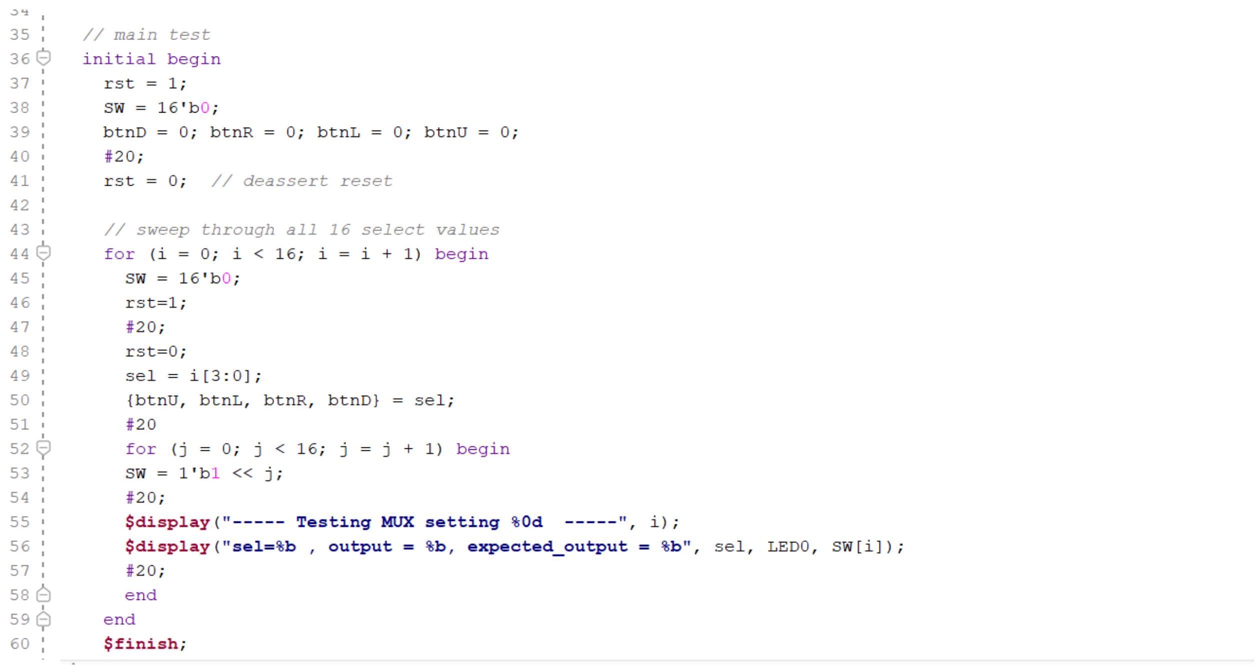

```

  // DUT I/O
  reg          clk, rst;
  reg          btnD, btnR, btnL, btnU;
  reg   [15:0] SW;
  wire         LED0=0;

  // drive sel procedurally
  reg    [3:0] sel;
  integer      i,j;

  // Instantiate DUT
  top_mux_lab3 uut (
    .clk  (clk),
    .rst  (rst),
    .btnD (btnD),
    .btnR (btnR),
    .btnL (btnL),
    .btnU (btnU),
    .SW   (SW),
    .LED0 (LED0)
  );

  // clock generator
  initial begin
    clk = 0;
    forever #10 clk = ~clk;
  end
```

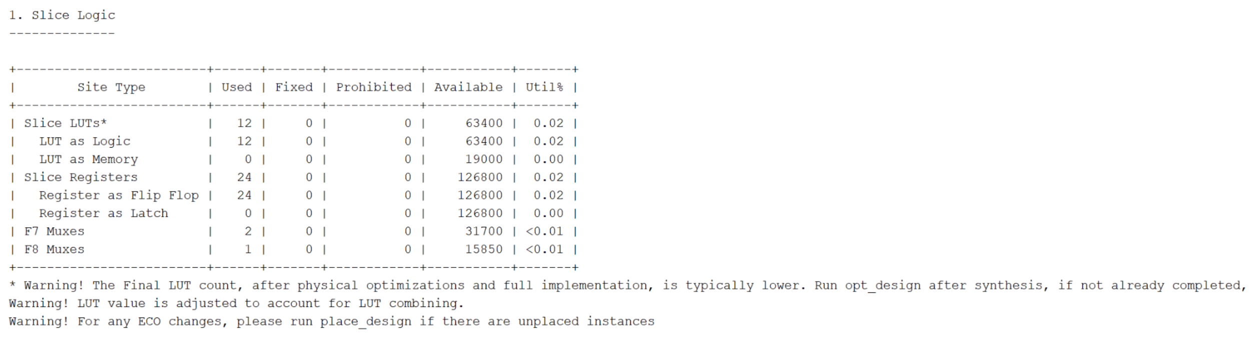

```

  // main test
  initial begin
    rst = 1;
    SW = 16'b0;
    btnD = 0; btnR = 0; btnL = 0; btnU = 0;
    #20;
    rst = 0;  // deassert reset

    // sweep through all 16 select values
    for (i = 0; i < 16; i = i + 1) begin
      SW = 16'b0;
      rst=1;
      #20;
      rst=0;
      sel = i[3:0];
      {btnU, btnL, btnR, btnD} = sel;
      #20
      for (j = 0; j < 16; j = j + 1) begin
      SW = 1'b1 << j;
      #20;
      $display("----- Testing MUX setting %0d  -----", i);
      $display("sel=%b , output = %b, expected_output = %b", sel, LED0, SW[i]);
      #20;
      end
    end
    $finish;
```

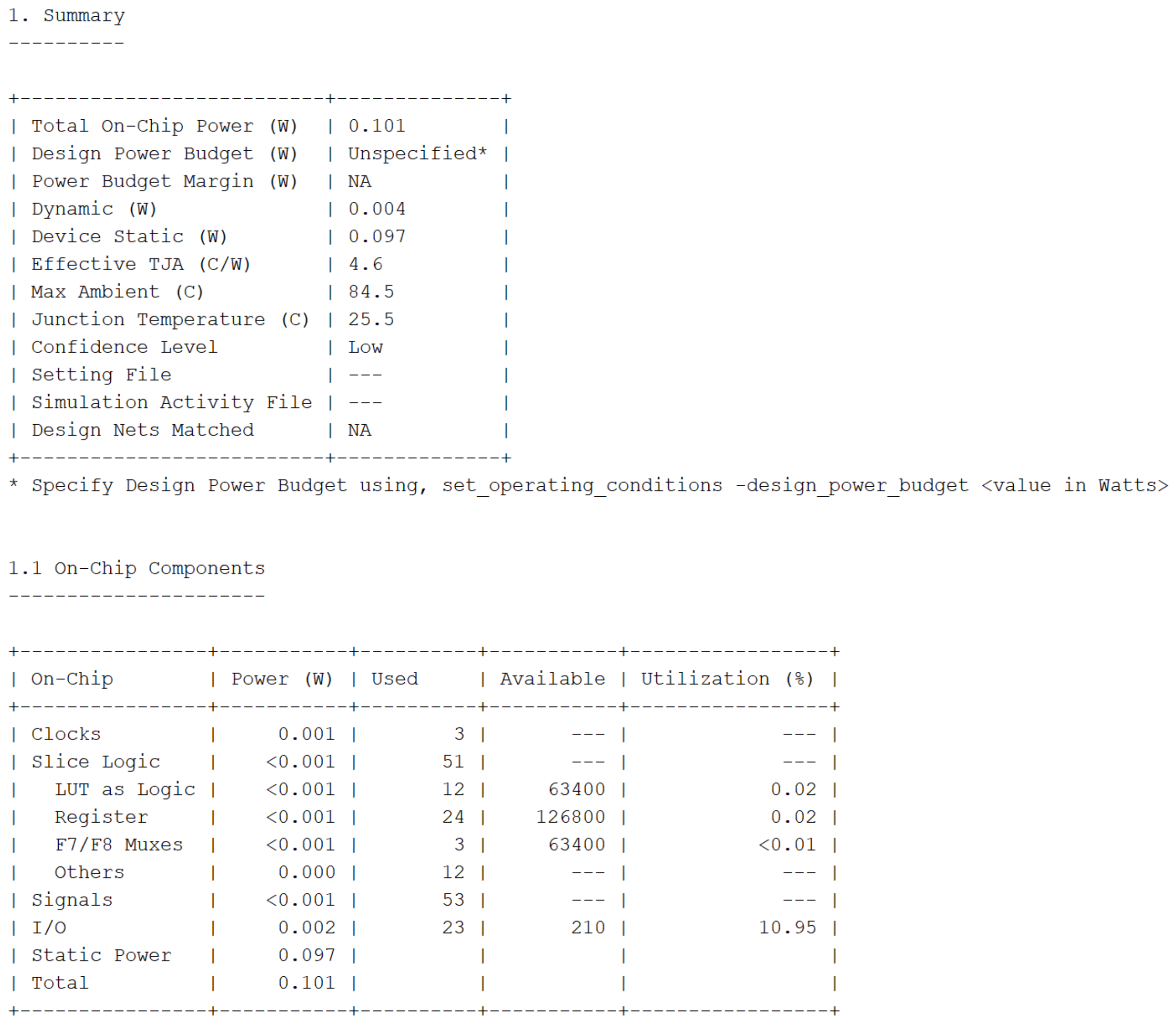
All buttons, the clock, reset, and switches are declared as well as integers i and j to be used in our for loop. The first initial begin is for the clock to generate a 50MHz clock. Then the main test loop resets the DUT and initializes the signals, ending with rst=0 to loop the reset. The buttons port back to the top module to call the debounce and flip flop. The outer loop then sweeps through the 4-bit values 0-15. The inner loop sets one switch bit high at a time, then displays the select, LED0, and other outputs.

Vivado Utilizations (LUTs, FFs, Power):



```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        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 |
+-------------------------+------+-------+------------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is typically lower. Run opt_design after synthesis, if not already completed,
Warning! LUT value is adjusted to account for LUT combining.
Warning! For any ECO changes, please run place_design if there are unplaced instances
```



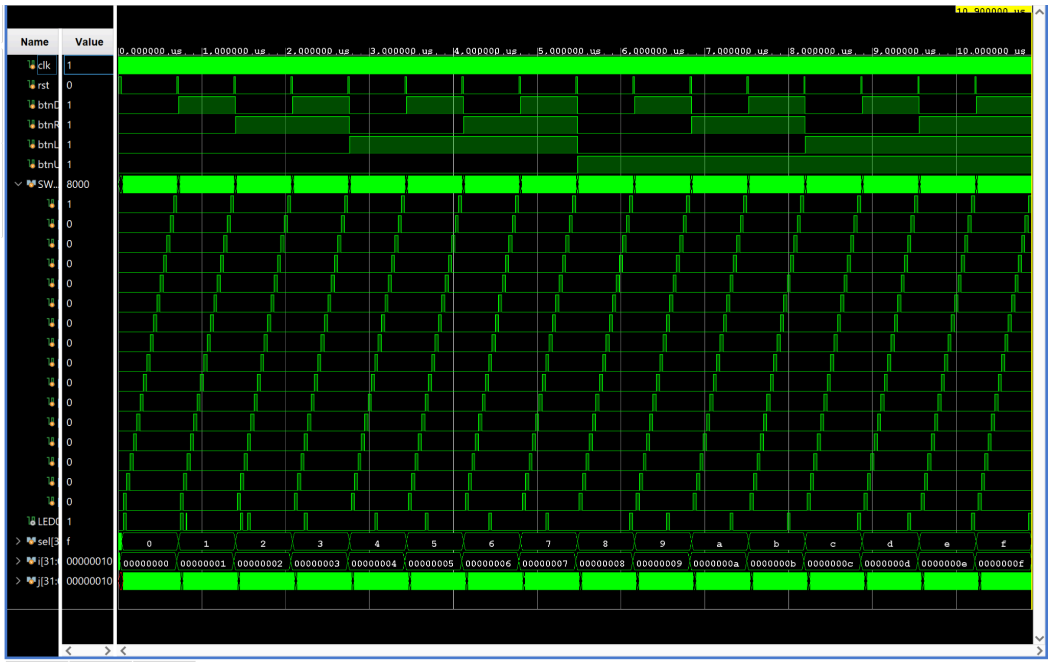
```
1. Summary
----------

+--------------------------+--------------+
| Total On-Chip Power (W)  | 0.101        |
| Design Power Budget (W)  | Unspecified* |
| Power Budget Margin (W)  | NA           |
| Dynamic (W)              | 0.004        |
| 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           |
+--------------------------+--------------+
* Specify Design Power Budget using, set_operating_conditions -design_power_budget <value in Watts>


1.1 On-Chip Components
----------------------

+----------------+-----------+----------+-----------+-----------------+
| On-Chip        | Power (W) | Used     | Available | Utilization (%) |
+----------------+-----------+----------+-----------+-----------------+
| Clocks         |     0.001 |        3 |       --- |             --- |
| Slice Logic    |    <0.001 |       51 |       --- |             --- |
|   LUT as Logic |    <0.001 |       12 |     63400 |            0.02 |
|   Register     |    <0.001 |       24 |    126800 |            0.02 |
|   F7/F8 Muxes  |    <0.001 |        3 |     63400 |           <0.01 |
|   Others       |     0.000 |       12 |       --- |             --- |
| Signals        |    <0.001 |       53 |       --- |             --- |
| I/O            |     0.002 |       23 |       210 |           10.95 |
| Static Power   |     0.097 |          |           |                 |
| Total          |     0.101 |          |           |                 |
+----------------+-----------+----------+-----------+-----------------+
```

(Testbench with waveform) Screenshot Proofs:



Video Link:

https://youtu.be/a1tPf9ari9A

Partner Contribution:

The lab was worked on together in person, so the work is 50/50. Heba started the project and code, then Sean fixed up the testbench. Heba was able to get the hardware to work while Sean edited the test bench to properly show all switches on the simulations. Heba then put together the lab report with Sean's screenshots and utilizations while Sean demonstrated the board and edited the video. All work was mutual with input on the code, report, and video done together.

Conclusion: For Lab 3 we finally got the basics down from the first two labs on how Vivado works, so this time, software was our sole focus. All files were set up properly, and the hardware worked as expected after some small editing inside our constraint file. However, when it came to testing the lab through simulation, our testbench had troubles after switch 4. We properly needed to get all 16 switches to be read through our loop to properly reflect the toggling of the 2x1 Muxs.