LABORATORY_3

Sequential Logic Design

Hannah Hwang | SID : 862419233 | CS120A Section 025
Adithya Chander | SID : 862429692 | CS120A Section 025
Calvin Hong | SID : 862416275 | CS120A Section 025

Overview

This is a three part lab.

In part I we explore the logic behind a flight attendant call system with the given specifications:

Press CALL : light ON

- Stays ON after button released

Press CANCEL : light OFF

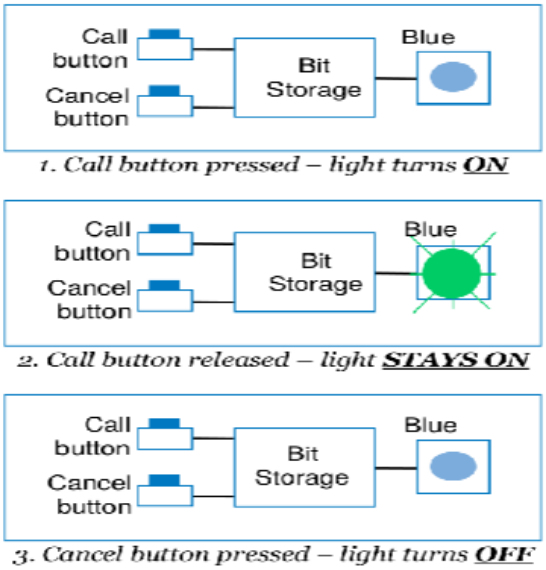


Figure 1. Flight Attendant System State Machine Description

Given the state output/input table and the implementation schematic, we derived an excitation equation and also implemented the call system in Verilog using the provided code and inputting two additional cases for 2'b10 and 2'b11.

In Part II, we are given the written schematic for a rising edge detector – "a circuit that generates a short one-clock-cycle pulse when the input changes from 1 to 0." With this information, we drew a finite state machine diagram in part A and in part B we developed the proposed FSM in Verilog with the provided template. We also implemented an edge detector module.

4. Design a sequential logic circuit that implements the excitation equation

In Part III, we explored the logic behind an LED display time multiplexing circuit where the four separate LED displays have individual enable signals but share eight common signals to light the segments.

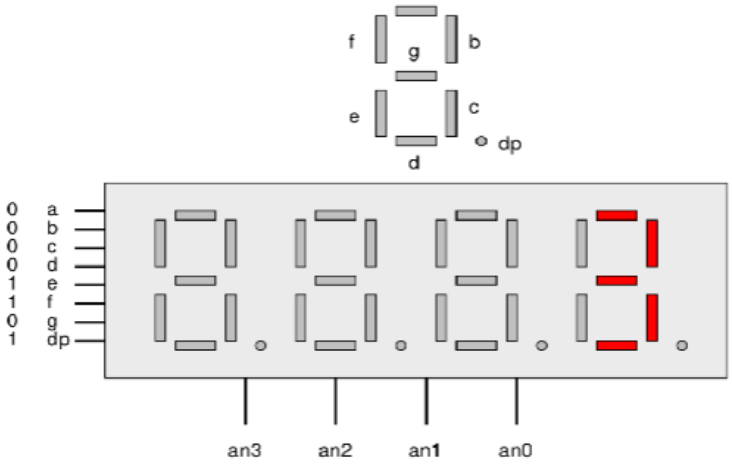


Figure 2. Display '3' on LED Display

We were also provided with a possible realization of the DISP_MUX and experimented with different design choices to implement the circuit. Lastly, we implemented the system on Verilog with the provided starter code.

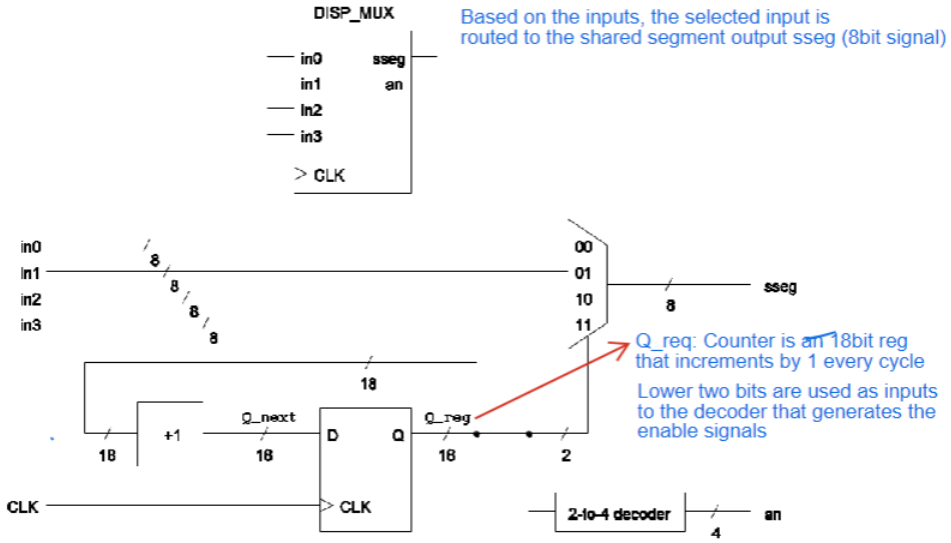


Figure 3. Symbol and block diagram of the time-multiplexing circuit

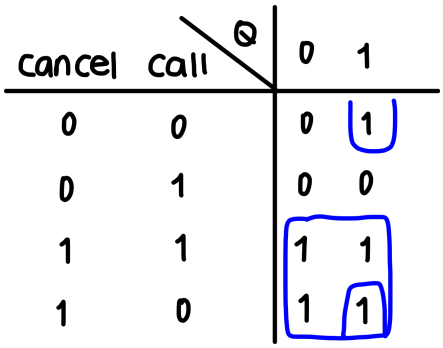
For all three parts of this lab, we did not have to implement a test bench so there is no direct result. Rather, the lab expanded our theoretical understanding of sequential logic by providing multiple examples.

Analysis

Part I

Call	Cancel	Q	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Figure 4. Finite State Machine input/output table



K-map representation of Figure.4

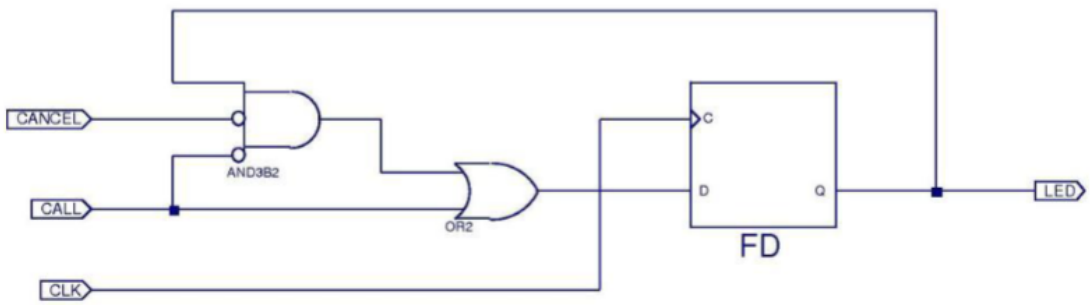


Figure 5. Flight attendant system schematic

Excitation Equation (SOP) : = Call + (Cancel' * Q) (Q's initial value is 0)

Part II

1. Derive a state diagram from the spec's description.

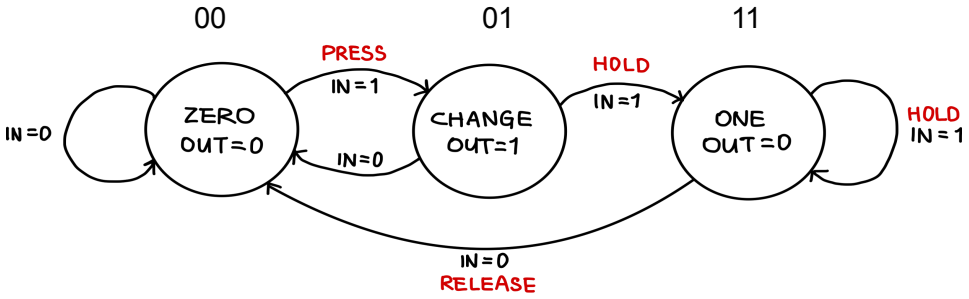


Figure 6. Rising-edge Detector Finite State Machine diagram

2. Show the output/transition table.

IN (signal)	CS (Current State)	NS (Next State)	OUT (outedge)
0	00	00	0
1	00	01	0
0	01	00	1
1	01	11	1
0	11	00	0
1	11	11	0

Figure 7. Rising-edge Detector output/transition table

IN (signal)	S0	S1	S0+	S1+	OUT (outedge)
0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	0	0	1
1	0	1	1	1	1
0	1	1	0	0	0
1	1	1	1	1	0

Figure 7.1. Output/transition table with states separated into an input for each bit

3. Derive the excitation equations.

Excitation Equations (SOP) : NS = IN * CS(right bit)
NS(right bit) = IN
OUT = !CS(left bit) * CS(right bit)

4. Design a sequential logic circuit that implements the excitation equation.

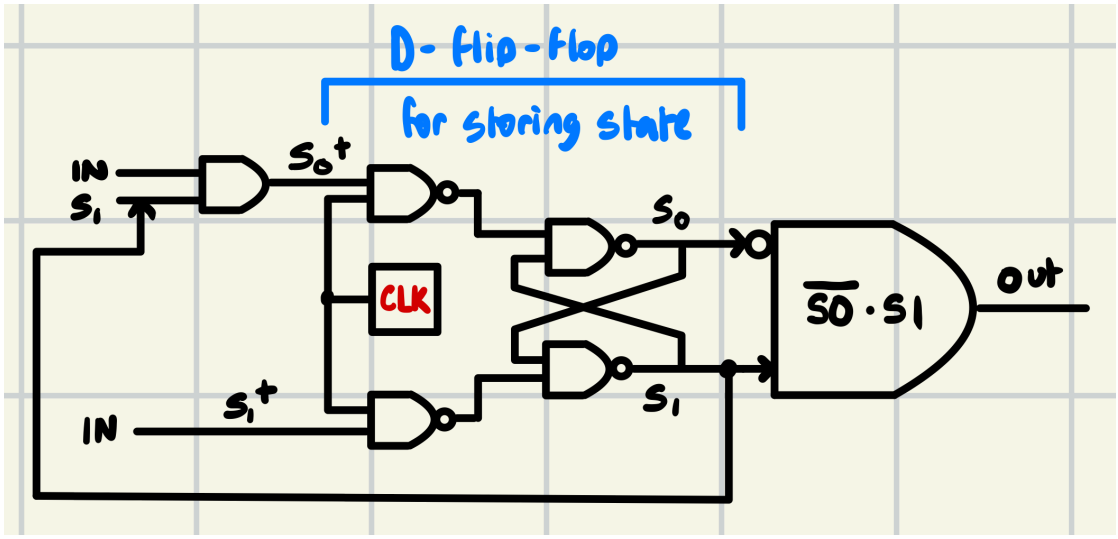


Figure 8. Circuit diagram based on truth table + FSM

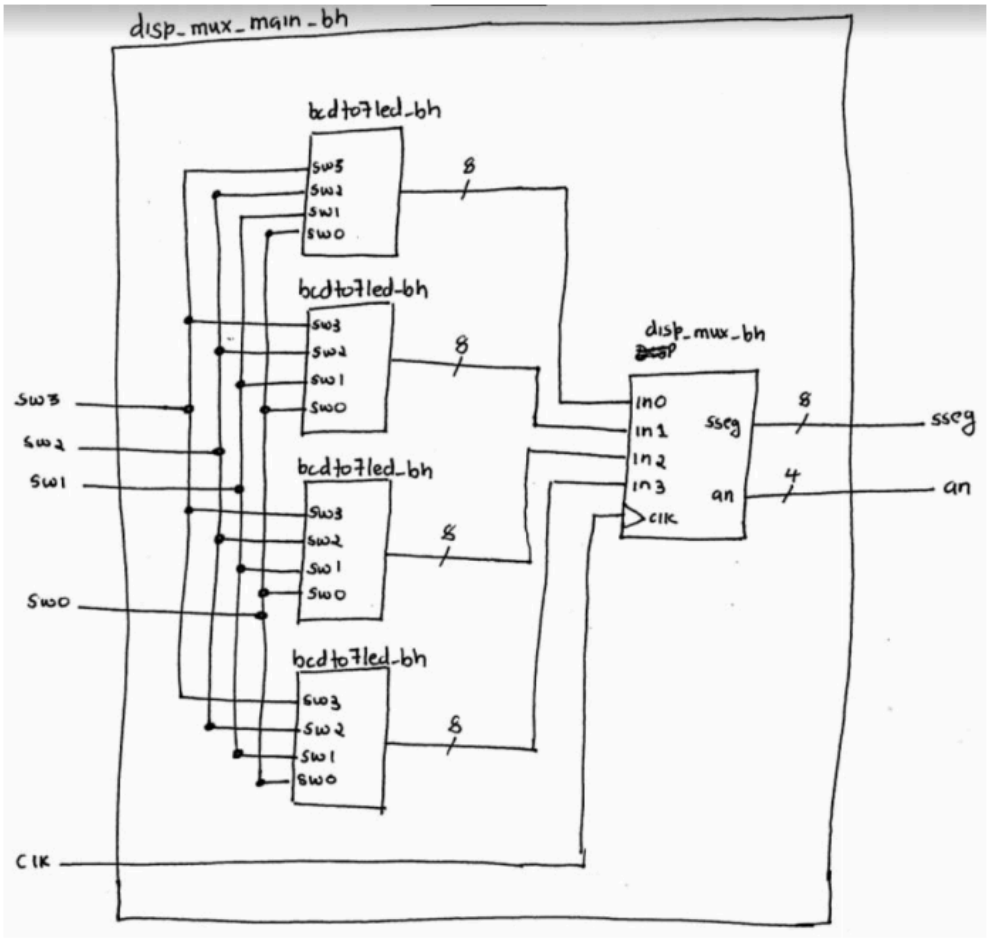
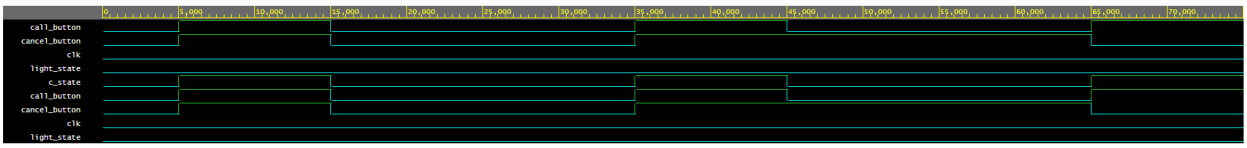


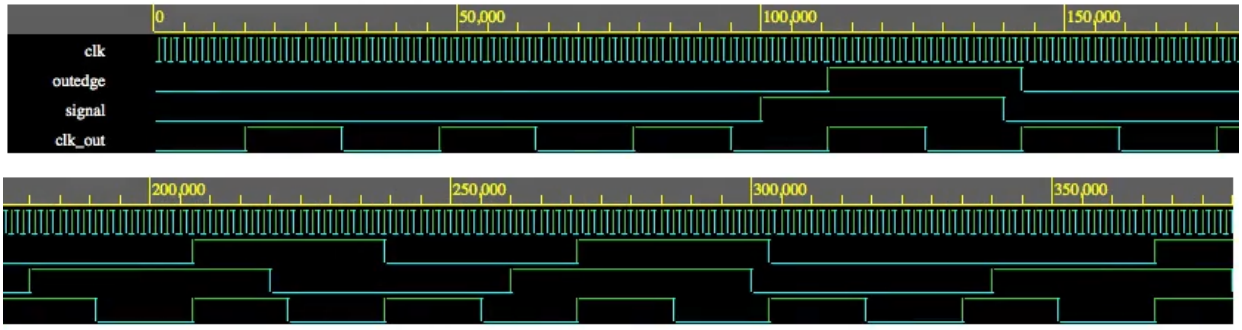
Figure 9. Visual diagram for module connections in part III

Records

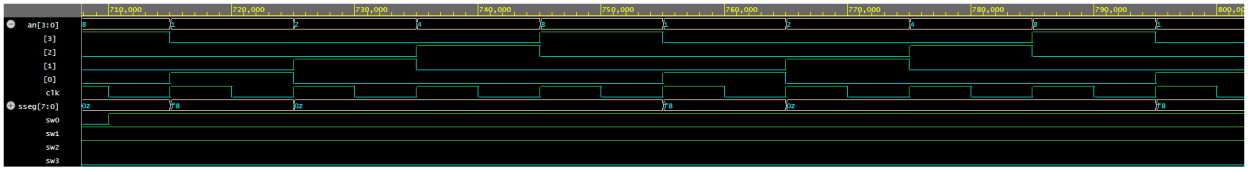
Part 1 output waveform:



Part 2 output waveforms:



Part 3 output waveforms:



Part I_testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1ns/1ps
module fasystem_bh_tb();
  // Inputs
  reg call_button;
  reg cancel_button;
  reg clk = 'd0;
  // Outputs
  wire light_state;

  // Bidirs


  // Instantiate the UUt
  fasystem_bh UUT(
    .call_button(call_button),
    .cancel_button(cancel_button),
    .clk(clk),
```

```
  .light_state(light_state)
);


always #1 clk <= -clk;

initial begin
  call_button = 0;
  cancel_button = 0;

end

initial begin
  $dumpfile("dump.vcd"); $dumpvars;

  #5
  call_button = 1;
  cancel_button = 1;
  $display("Test Case 8");
  if (light_state != 1'b1) $display("Result is incorrect");

  #10
  call_button = 0;
  cancel_button = 0;
  $display("Test Case 2");
  if (light_state != 1'b1) $display("Result is incorrect");

  #10
  call_button = 0;
  cancel_button = 0;
  $display("Test Case 1");
  if (light_state != 1'b0) $display("Result is incorrect");

  #10
  call_button = 1;
  cancel_button = 1;
  $display("Test Case 7");
  if (light_state != 1'b1) $display("Result is incorrect");

  #10
  call_button = 0;
  cancel_button = 1;
  $display("Test Case 4");
  if (light_state != 1'b1) $display("Result is incorrect");
```

```
    #10
    call_button = 0;
    cancel_button = 1;
    $display("Test Case 3");
    if (light_state != 1'b0) $display("Result is incorrect");

          #10
    call_button = 1;
    cancel_button = 0;
    $display("Test Case 5");
    if (light_state != 1'b1) $display("Result is incorrect");

    #10
    call_button = 1;
    cancel_button = 0;
    $display("Test Case 6");
    if (light_state != 1'b1) $display("Result is incorrect");

    // finish simulation

    $finish;
  end
endmodule
```

Part I_design.sv

```
`timescale 1ns/1ps

module fasystem_bh(
  input wire clk,
  input wire call_button,
  input wire cancel_button,
  output reg light_state
);

//Internal wire
  reg c_state;

//Combinatorial block
  always @(*) begin

    case({call_button, cancel_button})
```

```
    2'b00: c_state = light_state ?'d1 : 'd0;
    2'b01: c_state = 'd0;
    2'b10: c_state = 'd1;
    2'b11: c_state = 'd1;

    default : c_state = 'd0;

    endcase
  end

//Sequential block

always @(posedge clk) begin
  light_state <= c_state;
end
endmodule
```

Part II_testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1ns/1ps
module lab3part2tb;
  // inputs
  reg clk;
  reg signal;

  // outputs
  wire outedge;

  edgedetector_bh uut(
    .clk(clk),
    .signal(signal),
    .outedge(outedge)
  );

initial begin
  signal = 'd0;
  clk = 'd0;
end

// clock signal
always #1 clk <= !clk;
  initial begin
```

```
    $dumpfile("dump.vcd"); $dumpvars;
    //Initialize inputs

    //Wait 100 ns for global reset to finish
    // #20;
    #100
    signal = 1;
    // #250000
    #40;
    signal = 0;
    // #250000
    #40;
    signal = 1;
    // #250000
    #40;
    signal = 0;
    // #250000
    #40;
    signal = 1;
    // #250000
    #40;
    signal = 0;
    // #250000
    #40;
    signal = 1;
    // #250000
    #40
    signal = 0;
    // finish simulation
          $finish;
  end
endmodule
```

Part II_design.sv

```
module edgedetector_bh(
input wire clk,
input wire signal,
output reg outedge );  // An indicator when a rising edge is detected - 0 to 1
wire slow_clk ;

reg [1:0] c_state ;
reg [1:0] r_state ;
```

```
  // Define your FSM states
localparam ZERO = 'd0; // 0
localparam CHANGE = 'd1; // 0 to 1
localparam ONE = 'd2; // 1

  // EECS150 - Digital Design Lecture 17 - Finite State Machines Revisited

  // Code for clkdiv module is given below. Create a new Verilog module in the
  //same project with the given code.

  clkdiv c1(clk, slow_clk );

  // Comb. logic.

  always @(*) begin

    case (r_state)

      ZERO : begin
           c_state = signal ? CHANGE : ZERO ;
           outedge = 'd0 ;
      end

      CHANGE : begin
        c_state = signal ? ONE : ZERO ;
           outedge = 'd1 ;
      end

      ONE : begin
           c_state = signal ? ONE : ZERO ;
        outedge = 'd0 ;
      end

      default : begin
           c_state = ZERO ;
           outedge = 'd0 ;
      end

    endcase
end

// Seq. logic
always @( posedge clk ) begin
           r_state <= c_state ;
```

```
end

endmodule
```

Clock divider to slow down signal for LEDs observation

```
module clkdiv(clk,clk_out); // Clock divider to slow down the signal for LEDs observation

  input clk;
  output clk_out;

  reg [15:0] COUNT;

  assign clk_out=COUNT[15];

  always @(posedge clk)
  begin
          COUNT = COUNT + 1;
  end

endmodule
```

Part III_design.sv

```
// Code your design here
//`timescale 1ns / 1ps

module bcdto7led_bh(
input wire sw0 ,
input wire sw1 ,
input wire sw2 ,
input wire sw3 ,
output reg a ,
output reg b ,
output reg c ,
output reg d ,
output reg e ,
output reg f ,
output reg g ,
output reg dp
);
// Internal wire
wire [3:0] bundle ;
assign bundle = {sw3,sw2,sw1,sw0 } ;
always @(*) begin
a = 1'b1 ;
```

```
b = 1'b1 ;
c = 1'b1 ;
d = 1'b1 ;
e = 1'b1 ;
f = 1'b1 ;
g = 1'b1 ;
dp = 1'b1;
case ( bundle )
4'b0000 : begin // 0
a = 1'b0 ;
b = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
end
4'b0001 : begin // 1
b = 1'b0 ;
c = 1'b0 ;
end
4'b0010 : begin // 2
a = 1'b0 ;
b = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
g = 1'b0 ;
end
4'b0011 : begin // 3
a = 1'b0 ;
b = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
g = 1'b0 ;
end
4'b0100 : begin // 4
b = 1'b0 ;
c = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b0101 : begin // 5
a = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
```

```
f = 1'b0 ;
g = 1'b0 ;
end
4'b0110 : begin // 6
a = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b0111 : begin // 7
a = 1'b0 ;
b = 1'b0 ;
c = 1'b0 ;
end
4'b1000 : begin // 8
a = 1'b0 ;
b = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b1001 : begin // 9
a = 1'b0 ;
b = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b1010 : begin // A
a = 1'b0 ;
b = 1'b0 ;
c = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b1011 : begin // B
c = 1'b0 ;
d = 1'b0 ;
```

```
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b1100 : begin // C
a = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
end
4'b1101 : begin // D
b = 1'b0 ;
c = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
g = 1'b0 ;
end
4'b1110 : begin // E
a = 1'b0 ;
d = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
4'b1111 : begin // F
a = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
endcase
end
endmodule
```

```
module disp_mux_bh(
input clk ,
```

```
input wire [7:0] in0 ,
input wire [7:0] in1 ,
input wire [7:0] in2 ,
input wire [7:0] in3 ,
output reg [3:0] an ,
output reg [7:0] sseg
 );
  reg [16:0] r_qreg = 0;
  reg [16:0] c_next = 0;

  always @(*) begin
    case (r_qreg[1:0])
    2'b00 : begin
      sseg = in0;
            an = 4'b0001;
    end
    2'b01 : begin
      sseg = in1;
      an = 4'b0010;
    end
    2'b10 : begin
      sseg = in2;
      an = 4'b0100;
    end
    2'b11 : begin
      sseg = in3;
      an = 4'b1000;
    end
    endcase
  end

  always @(posedge clk) begin
    r_qreg <= (r_qreg + 'd1);
  end
endmodule

module dispmux_main_bh(
  input clk,
  input sw0,
  input sw1,
  input sw2,
  input sw3,
  output [3:0] an,
  output [7:0] sseg
```

```
);

  wire [7:0] in0;
  wire [7:0] in1;
  wire [7:0] in2;
  wire [7:0] in3;

  bcdto7led_bh c1(sw0,sw1,sw2,sw3,in0[0],in0[1],in0[2],in0[3],in0[4],in0[5],in0[6],in0[7]);
  bcdto7led_bh c2(sw0,sw1,sw2,sw3,in0[0],in0[1],in0[2],in0[3],in0[4],in0[5],in0[6],in0[7]);
  bcdto7led_bh c3(sw0,sw1,sw2,sw3,in0[0],in0[1],in0[2],in0[3],in0[4],in0[5],in0[6],in0[7]);
  bcdto7led_bh c4(sw0,sw1,sw2,sw3,in0[0],in0[1],in0[2],in0[3],in0[4],in0[5],in0[6],in0[7]);


  disp_mux_bh c5(
    .clk(clk),
    .in0(in0),
    .in1(in1),
    .in2(in2),
    .in3(in3),
    .an(an),
    .sseg(sseg));

endmodule
```

Part III_testbench.sv

```
// Code your testbench here
// or browse Examples

`timescale 1ns/1ps

module lab3part3tb;

// inputs
  reg clk;
  reg sw0;
  reg sw1;
  reg sw2;
  reg sw3;

// outputs
  wire[3:0] an;
  wire[7:0] sseg;

  // instantiate the unit under test (UUT)
```

```
  dispmux_main_bh uut(
    .clk (clk) ,
    .sw0 (sw0) ,
    .sw1 (sw1) ,
    .sw2 (sw2) ,
    .sw3 (sw3) ,
    .an (an) ,
    .sseg (sseg)
  );

initial begin

    clk = 0;
    sw0 = 0;
    sw1 = 0;
    sw2 = 0;
    sw3 = 0;
end

// clock signal
always #5 clk <= !clk;

initial begin
  $dumpfile("dump.vcd"); $dumpvars;
  // initialize inputs
  #10
  sw0 = 0;
  sw1 = 0;
  sw2 = 0;
  sw3 = 0;

  // wait 100 ns for global reset to finish
  #100
  sw0 = 0;
  sw1 = 0;
  sw2 = 0;
  sw3 = 0;

  // add stimulus here
  #100
  sw0 = 0;
  sw1 = 1;
  sw2 = 0;
  sw3 = 0;
```

```
  #100
sw0 = 1;
sw1 = 1;
sw2 = 0;
sw3 = 0;

  #100
sw0 = 0;
sw1 = 0;
sw2 = 1;
sw3 = 0;
  #100
sw0 = 1;
sw1 = 0;
sw2 = 1;
sw3 = 0;
  #100
sw0 = 0;
sw1 = 1;
sw2 = 1;
sw3 = 0;
  #100
sw0 = 1;
sw1 = 1;
sw2 = 1;
sw3 = 0;
  #100
sw0 = 0;
sw1 = 0;
sw2 = 0;
sw3 = 1;
  #100
sw0 = 1;
sw1 = 0;
sw2 = 0;
sw3 = 1;
  #100
sw0 = 0;
sw1 = 1;
sw2 = 0;
sw3 = 1;
  #100
sw0 = 1;
```

```
  sw1 = 1;
  sw2 = 0;
  sw3 = 1;
    #100
  sw0 = 0;
  sw1 = 0;
  sw2 = 1;
  sw3 = 1;
    #100
  sw0 = 1;
  sw1 = 0;
  sw2 = 1;
  sw3 = 1;
    #100
  sw0 = 0;
  sw1 = 1;
  sw2 = 1;
  sw3 = 1;
    #100
  sw0 = 1;
  sw1 = 1;
  sw2 = 1;
  sw3 = 1;
  $finish;
end
endmodule
```

Discussion

For part 1, we were asked to implement a Flight attendant calling system, involving two inputs for “calling” the attendant and “canceling” the original call input. We needed to make sure that the “call” input always had priority over the “cancel” input if it was turned on.

The provided specifications included a diagram as to how the state machine would work, and by using the provided truth table and schematic, we were able to solve for an excitation equation, that was the SOP representation of how the circuit worked. It used Q(which was the current state) as an input alongside the two button inputs to calculate an output for the next state, which would be updated every time the clock input refreshed. We aren’t sure if the design that we came up with works as intended, as we weren’t provided a testbench with which we would be able to test our code, but our design matches the SOP representation of the provided truth table and the circuit schematic.

Additionally, we weren’t sure if the circuit diagram that we came up with was correct or not, but we did the best we could given the excitation SOP equations that we were able to come up with.

For part 2, we were asked to construct an FSM state diagram. There weren’t a lot of instructions as to how we would implement that, but we referenced the provided lab videos to understand how we would draw out the diagram. Using the video and the explanations it provided, we were able to draw and understand our own FSM state diagram, with three states for zero, change, and one.

Similar to part 1 of the lab, we were once again not provided with a testbench design that we could use to test our code, so we aren’t sure if our code works as intended or not in practice. However, we were able to use our FSM state diagram to implement the “CHANGE” and “ONE” states the verilog design that we were provided, by assigning the output values to either 0 or 1 depending on the value of “signal” and assigning that to the current state for use as the input for the next state calculation.

Conclusion

The purpose of this lab was to learn how sequential logic circuits work, i.e. implementing a circuit that would use its own output as an input in its next output calculation. We called each of these output calculations “states” and used each state to calculate the state that came afterwards.

In Part 1 we implemented a flight attendant call system using a call and cancel light. We were given a transition table, and were told to derive excitation equations and write verilog test cases for the circuit that we designed.

Part 2 allowed us to understand how to use and implement a rising edge detector, where the clock of the circuit was run at various speeds to showcase how ticks work in a circuit that has a clock in it. We made an FSM diagram, an output/transition table, and then implemented the code in verilog.

In Part 3 we used the knowledge that we got from the earlier sections about how a clock can allow us to update a circuit’s state in order to implement the provided schematic for how the LED Display would display the correct values for each number. We did so using a separate module for each of the individual displays, and linked them together using a combination of switches, a clock, and a multiplexer to decide which of the signals would be outputted (i.e. which lights would turn on).

Questions

Part I

- **What will happen if the “clock” signal is of very low frequency (1Hz)?**
  A slower clock signal implies that the circuit itself would run at a slower pace, as the “FD” block in the given schematic would update less frequently and therefore the circuit itself would not reassign its state as often. The circuit could only change its state once a second, as opposed to when a clock signal is faster, where the circuit would be updating its state more frequently.