EE/CS120A Logic Design
Department of Electrical Engineering
University of California – Riverside

Laboratory #3
EE/CS 120 A
Fall 2024

LABORATORY # 3

Sequential Logic Design (EDA Playground)

Type text here

Type text here Type text here

Objectives

Lab 4 contains 3 parts: **Part 1** – implementation of a sequential circuit discussed in class; **Part 2** – design and implementation of a state machine; **Part 3** – design of time multiplexing circuits for four-LED display. Its purposes are to get familiar with:

1. Clock synchronous state machine design, synthesis and implementation.

Equipment

- PC or compatible

Software

- EDA Playground

Parts

N/A

PART 1. Flight Attendant Call System

Part A: In this application development experiment, we will implement and test the "flight attendant call system" discussed in class.

Specification

The Flight Attendant System functions according to the following rules:

Flight attendant call button

Press **CALL**: light turns on

- Stays on after button released

Press **CANCEL**: light turns off

And is shown diagrammatically in Figure 1.

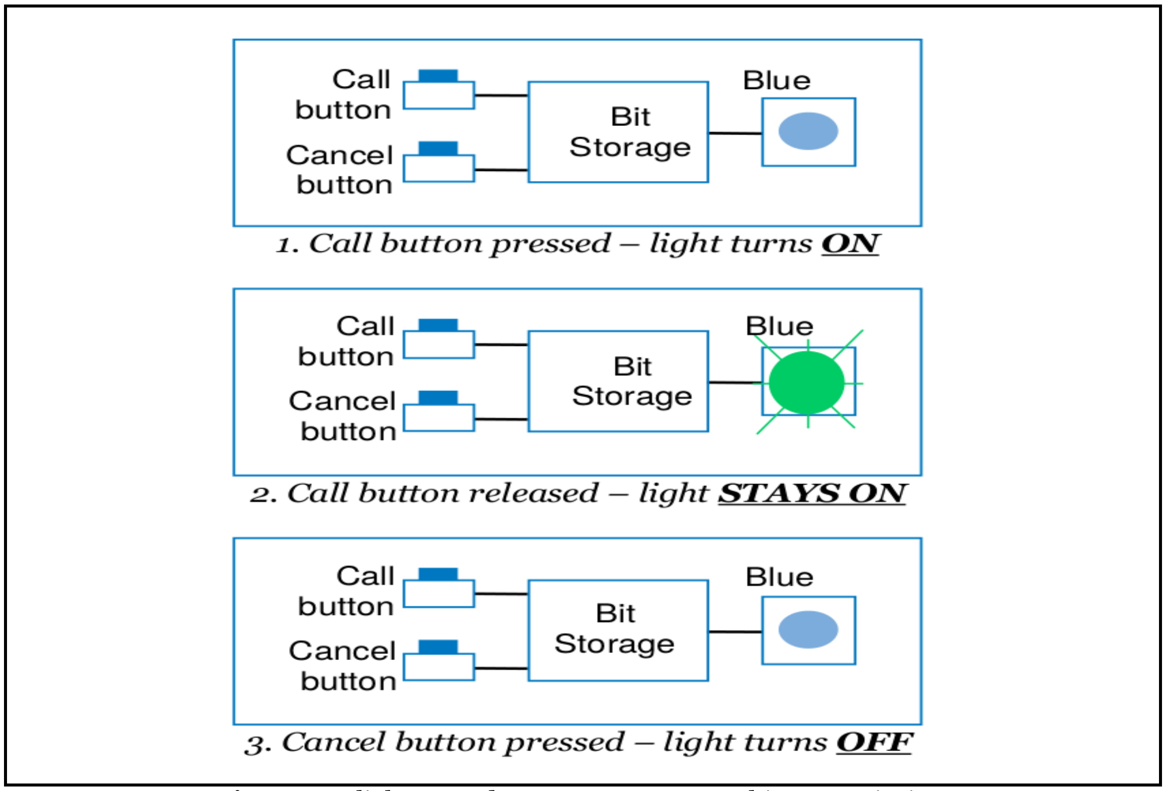
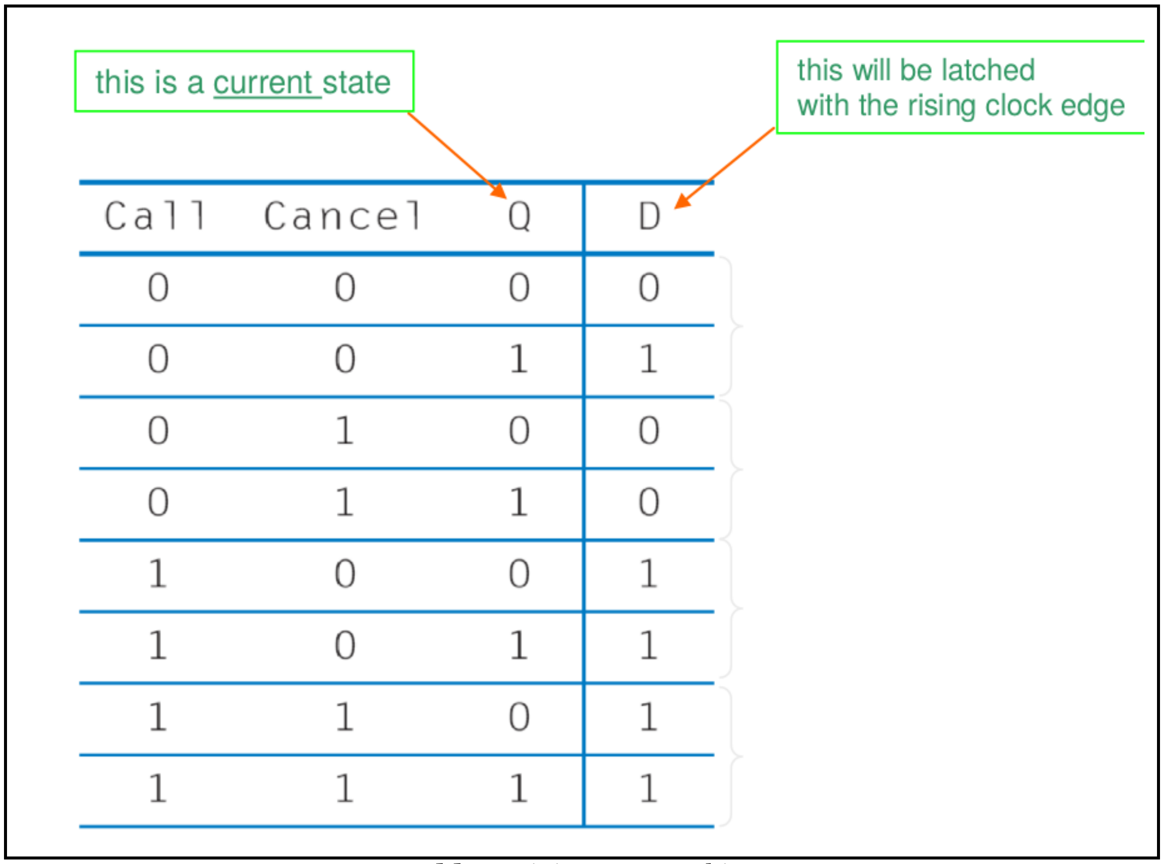

Call button
Cancel button
Bit Storage
Blue
1. Call button pressed – light turns ON
Call button
Cancel button
Bit Storage
Blue
2. Call button released – light STAYS ON
Call button
Cancel button
Bit Storage
Blue
3. Cancel button pressed – light turns OFF

Figure 1. Flight Attendant System State Machine Description

System Analysis and Implementation

As discussed in class from the problem description we can obtain the following state output/transition table.


this is a current state
this will be latched with the rising clock edge

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

Table 1. Finite State Machine

Derive **excitation equation** which leads to the following implementation schematic

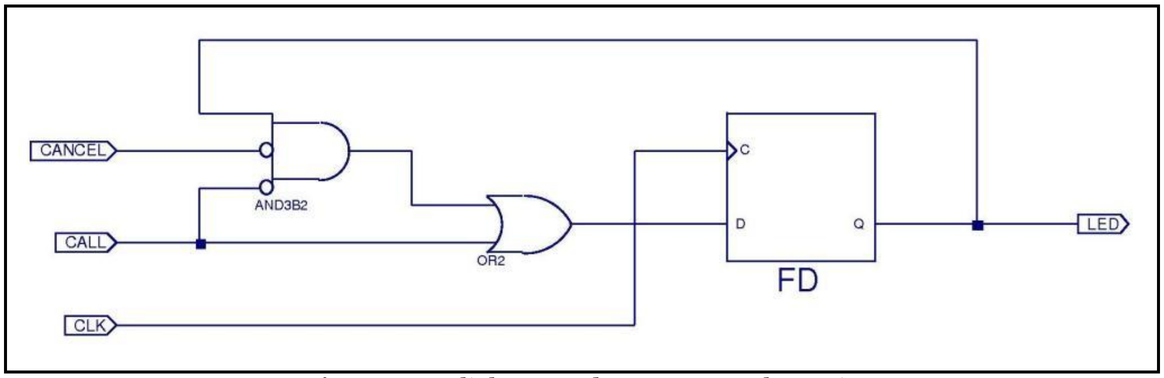


Figure 2. Flight Attendant System Schematic

Conduct the Behavioral Simulation

Part B: In this part of the lab your job is to implement the flight attendant call system using Verilog. Although this system can be implemented using either structural or behavioral modeling, in this lab our aim is to practice behavioral modeling. The following is the module interface of the required system. Please complete the missing code.

```
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:  //your code
          2'b11:  //your code

          default : c_state = 'd0 ;

          endcase

end

// Sequential block

always @( posedge clk ) begin
```

```
        light_state  <= c_state ;
end
endmodule
```

According to this specification, the Verilog flight attendant system has three input signals (clock, call button and cancel button) as well as one output signal (the light state signal). Notice that this system requires memory. This is, somewhere in the module, the proposed circuit has to have the capability of storing the current state of the system. In addition, the new state of the system can be implemented in a combinatorial block.

Demonstration

Demonstrate what the application performs according to specs, both the schematics and the verilog code.

Questions

1. What will happen if the "clock" signal is of very low frequency (1 Hz)?

PART 2. Rising-edge Detector

Objective

In this assignment, it is required to construct a Finite State Machine (FSM) state/output diagram.

Specification

Part A: The rising edge detector is a circuit that generates a short, one-clock-cycle pulse (called a tick) when the input signal changes from '0' to '1'. It is usually used to indicate the onset of a slow time-varying input signal.

Part B: In this part your job is to implement the proposed FSM developed in part A in verilog. To facilitate the process of testing the code, your FSM module should have the following ports. Please complete the missing code.

```
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
            Your code ;
      end

      ONE   :   begin
            Your code ;
      end

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

  endcase
```

```
end

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

endmodule
```

The edge detector module can be implemented using two blocks. One combinatorial block to compute the FSM next state and another sequential block to store the FSM state. In addition, if we use the clocks in the board, we will not be able to see the rise edge event in the LEDs as the clocks in the board run at a high frequency. To produce a clock that goes slower, the following code is provided.

```
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
```

In this code, the signal clk is coming from the board while the signal clk_out is the one that drives the implemented FSM.

Demonstration

1. Derive a state diagram from the spec's description.
2. Show the output/transition table. $\rightarrow$ Add the current state, next state, signal and the outedge
3. Derive the excitation equations.
4. Design a sequential logic circuit that implements the excitation equation.

PART 3 : LED Display Time Multiplexing Circuit

Part A: The Digilent Basys Board contains four seven segment LED displays with decimal points. To reduce the number of FPGA's I/O pins used, it is required to use a time-multiplexing sharing scheme. That is, the four displays have their enable signals but share eight common signals to light the segments. All signals are active-low (i.e., enabled when a signal is '0'). The schematic of displaying '3' on the right-most LED is shown in Figure 7.

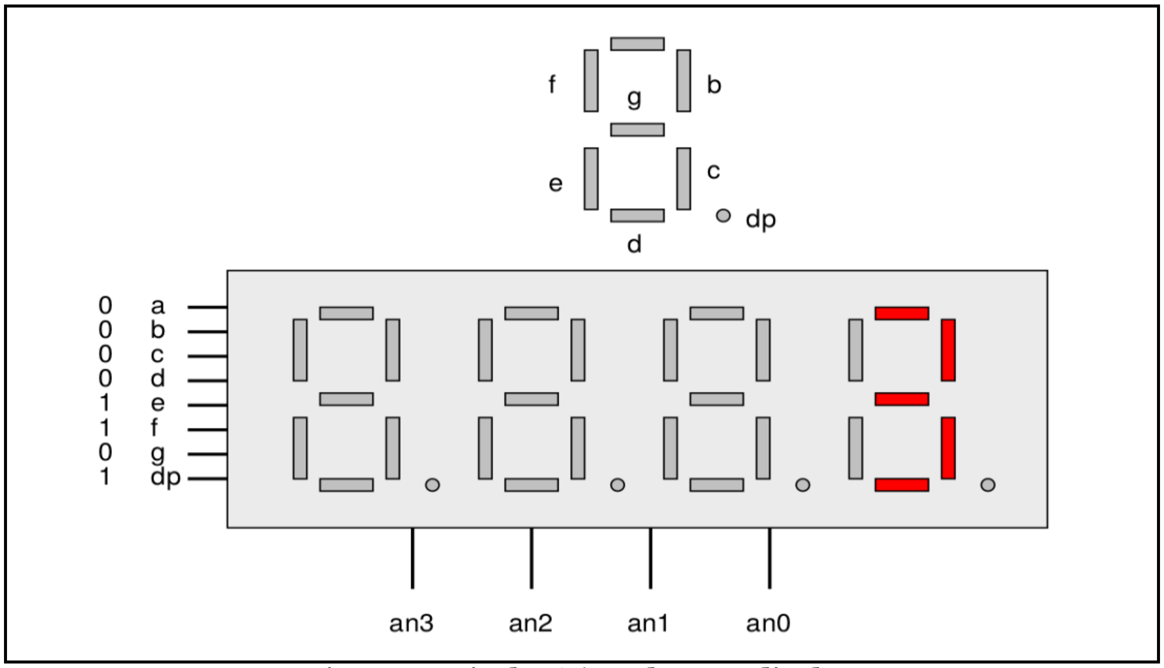


Figure 7. Display '3' on the LED display

Note that the enable signal (i.e., an) is '1110'. This configuration clearly can enable only one display at a time. We can time-multiplex the four LED patterns by enabling the four displays in turn, as shown in the simplified timing diagram in

Figure 8. If the refreshing rate of the enable signal is fast enough, the human eye cannot distinguish the on and off intervals of the LEDs and perceives that all four displays are lit simultaneously. This scheme reduces the number of I/O pins from 32 to 12 (i.e., eight LED segments plus four enable signals) but requires a time multiplexing circuit.

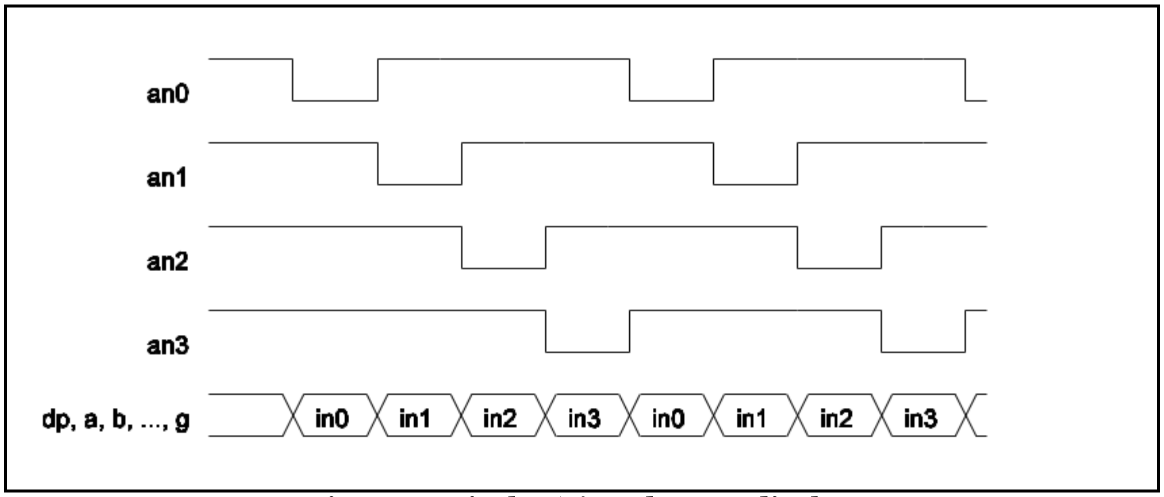


Figure 8. Display '3' on the LED display

One possible realization of the DISP_MUX is shown in the block diagram of Figure 9. Use it as a guide to implement the circuit and verify (simulate ONLY) its functionality. Feel free to design your own implementation of the DISP_MUX if you like. Only the functionality is important for this lab.

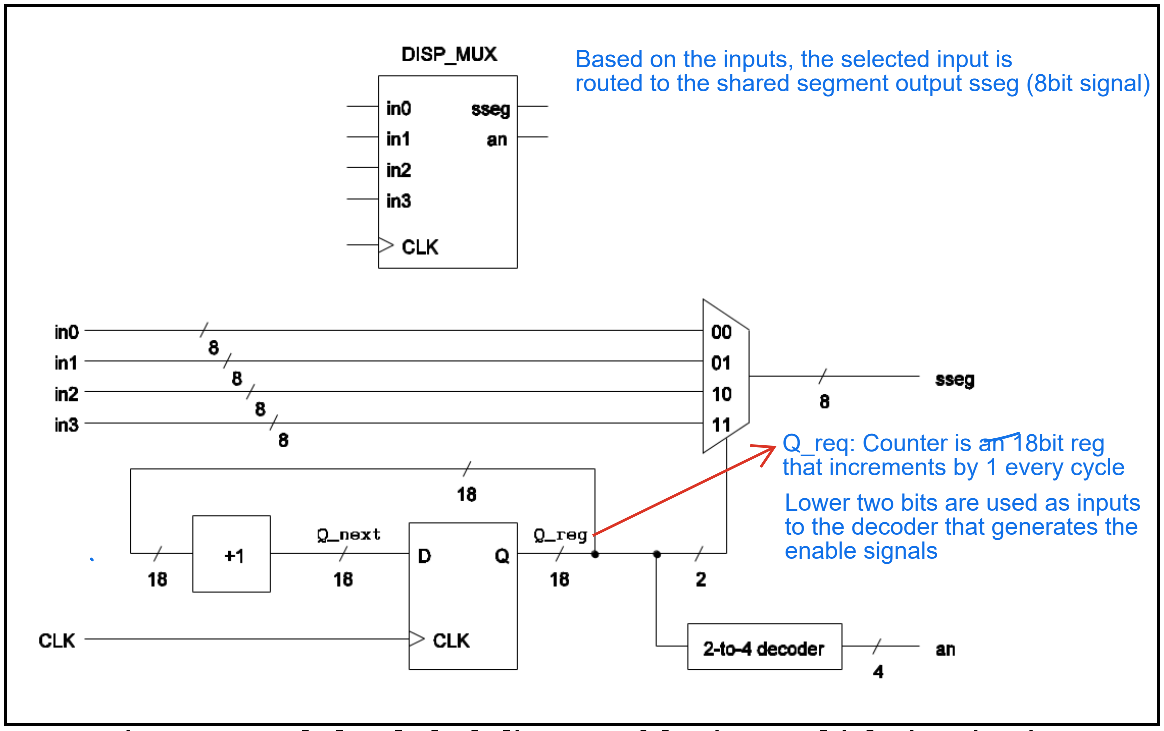

DISP_MUX
in0
in1
in2
in3
CLK
sseg
an
Based on the inputs, the selected input is routed to the shared segment output sseg (8bit signal)
00
01
10
11
8
18
Q_next
Q_reg
D
Q
+1
2
2-to-4 decoder
4
Q_req: Counter is ~~an~~ 18bit reg that increments by 1 every cycle
Lower two bits are used as inputs to the decoder that generates the enable signals

Figure 8. Symbol and Block diagram of the time-multiplexing circuit.

7-Segment Display

Copy-paste the following hexto7segment verilog code (from lab3).

```
`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
```

Figure 9. HEX-TO-LEDSEG Verilog Code

<u>Part B</u>: In this part your goal is to implement in verilog the circuit described in figure 8. The input and output signals of the main module in your behavioral implementation should be as follows:

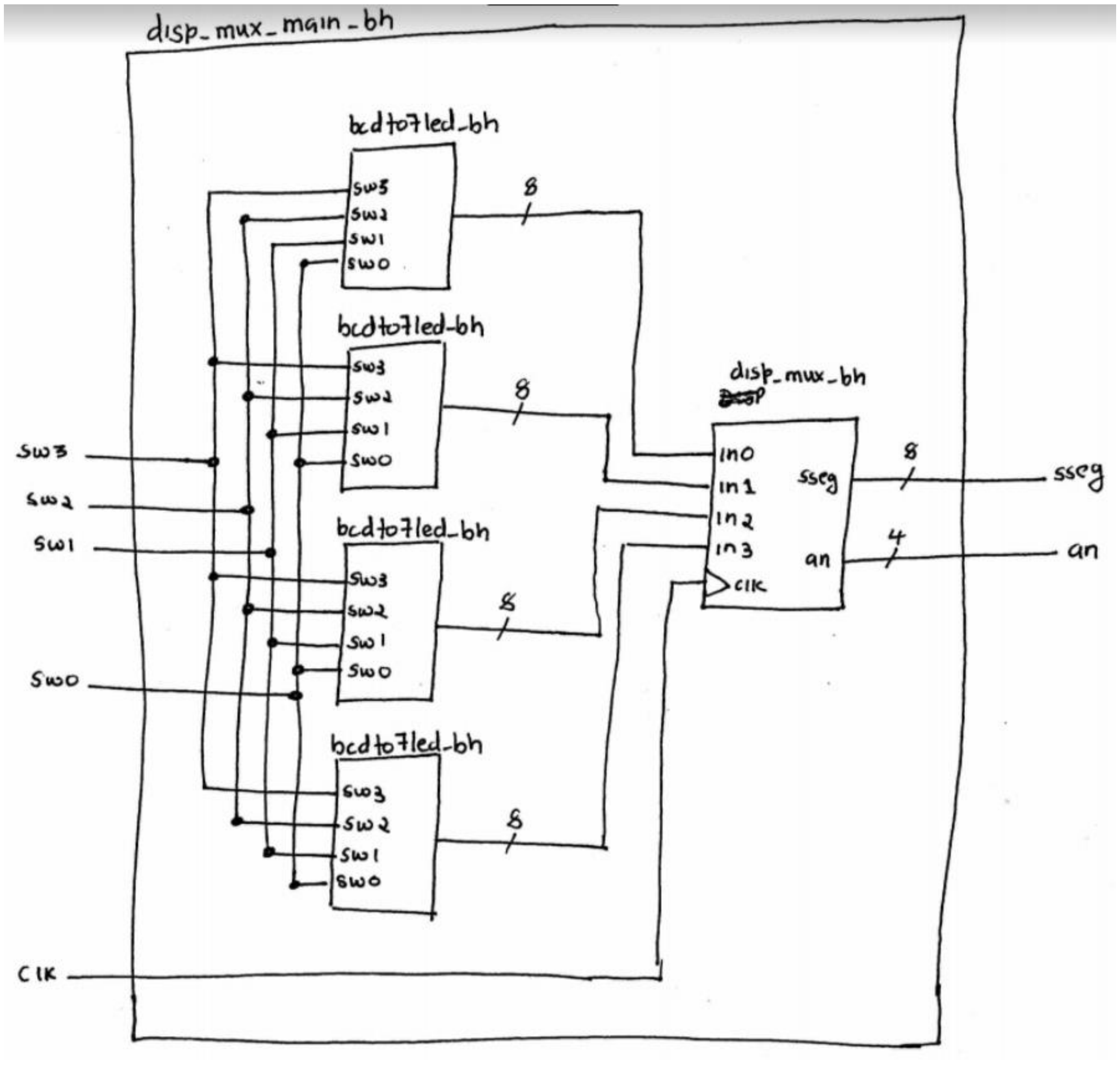
```
module dispmux_main_bh(
 input clk ,  // Clock signal
 input sw0, // Switch input
 input sw1, // Switch input
 input sw2, // Switch input
 input sw3, // Switch  input
 output [3:0] an ,  // LED selector
 output [7:0] sseg  // Segment signals
   );

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

// --------------------------------
// Module instantiation  bcdto7led
// --------------------------------

bcdto7led_bh c1(sw0, sw1, sw2, sw3,
                in0[0],in0[1],in0[2],in0[3], in0[4],in0[5],in0[6],in0[7]  );
// Your code (3 more modules)

// --------------------------------
// Module instantiation Mux
// --------------------------------

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

endmodule
```

In addition, you should use the verilog code in figure 9. Notice that the decoder and the multiplexor shown in figure 8 can be implemented using case statements. Moreover, the counter can be implemented using a sequential block. Finally, to synthetize your code in the given board, you can use the following source files.

```
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 ;
reg [16:0] c_next ;

// Mux ***************************************

always @(*) begin

          case (r_qreg[1:0])
          2'b00 : sseg = in0 ;
          2'b01 : sseg = in1 ;
          2'b10 : sseg = in2 ;
          2'b11 : sseg = in3 ;
endcase

end

// Decoder ************************************

always @(*) begin

          case (r_qreg[1:0])
          2'b00 : an = ~(4'b0001) ;
          2'b01 : an = ~(4'b0010) ;
          2'b10 : an = ~(4'b0100) ;
          2'b11 : an = ~(4'b1000) ;
          endcase

end

// Counter ************************************
always @(*) begin
          c_next = r_qreg + 'd1;
end

// Register
always @(posedge clk) begin
  r_qreg <= c_next ;
end

endmodule
```

Refer to the following image to get an intuition on how the modules are connected to each other:



Demonstration

Since no testbench is given, in the lab report, only the design codes (**the parts you implement must be included**) need to be included in the report. Schematics are optional.

Presentation and Report

Must be presented according to the general EE120A lab guidelines.

Prelab

1. Familiarize yourself with Verilog design code and EDA Playground.
2. Review Lectures.
3. Try to answer all the questions, prepare logic truth tables, do all necessary computations.