EECS120A Logic Design
Department of Electrical Engineering
University of California – Riverside

Laboratory #4
EECS 120 A
Spring 2020

LABORATORY # 4
LAB MANUAL

Sequential Logic Design

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.
2. Usage of function generator for external "clock" input for Basys FPGA boards via PMOD input/output connectors.
3. Creating and using symbol libraries within Xilinx ISE.
4. Using buses in schematic capture.
5. Vector entries in configuration files, control of external clocks.

Equipment

- PC or compatible
- Function Generator (Agilent 33120A)
- Digilent's Basys Spartan-3E FPGA Evaluation Board

Software

- Xilinx ISE Design Software Suite
- ModelSim XE III modeling software
- Digilent's Adept ExPort Software

Parts

- Connecting wires

PART 1. Flight Attendant Call System

In this FPGA application development experiment, we will implement and test the "flight attendant call system" discussed in class. Since in this lab we have to implement registers we need a clock source. The board allows the specification of both internal and external clock sources. In this lab, we use the internal clock source. Here, the first step is to configure the board such that the internal clock runs at 25 Mhz (JP 4 has to be close in the 25 Mhz position).

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.

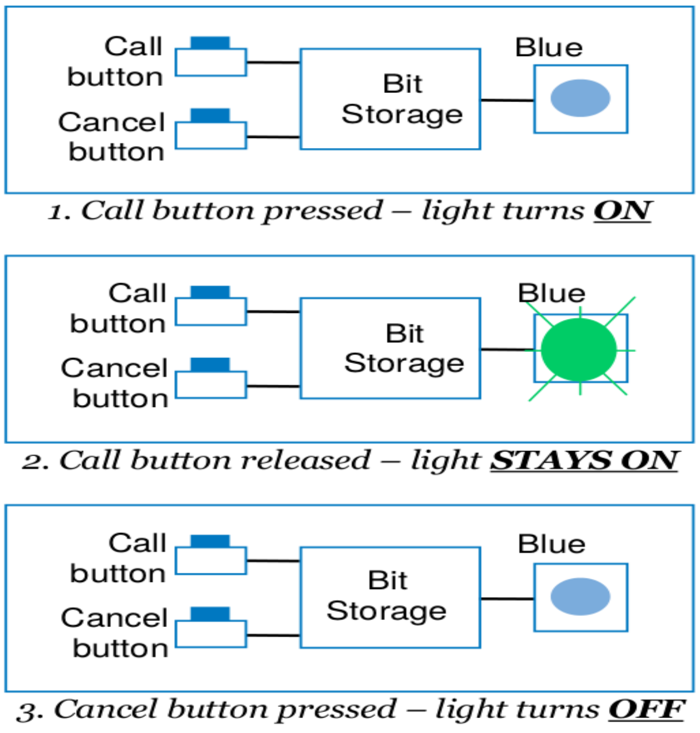
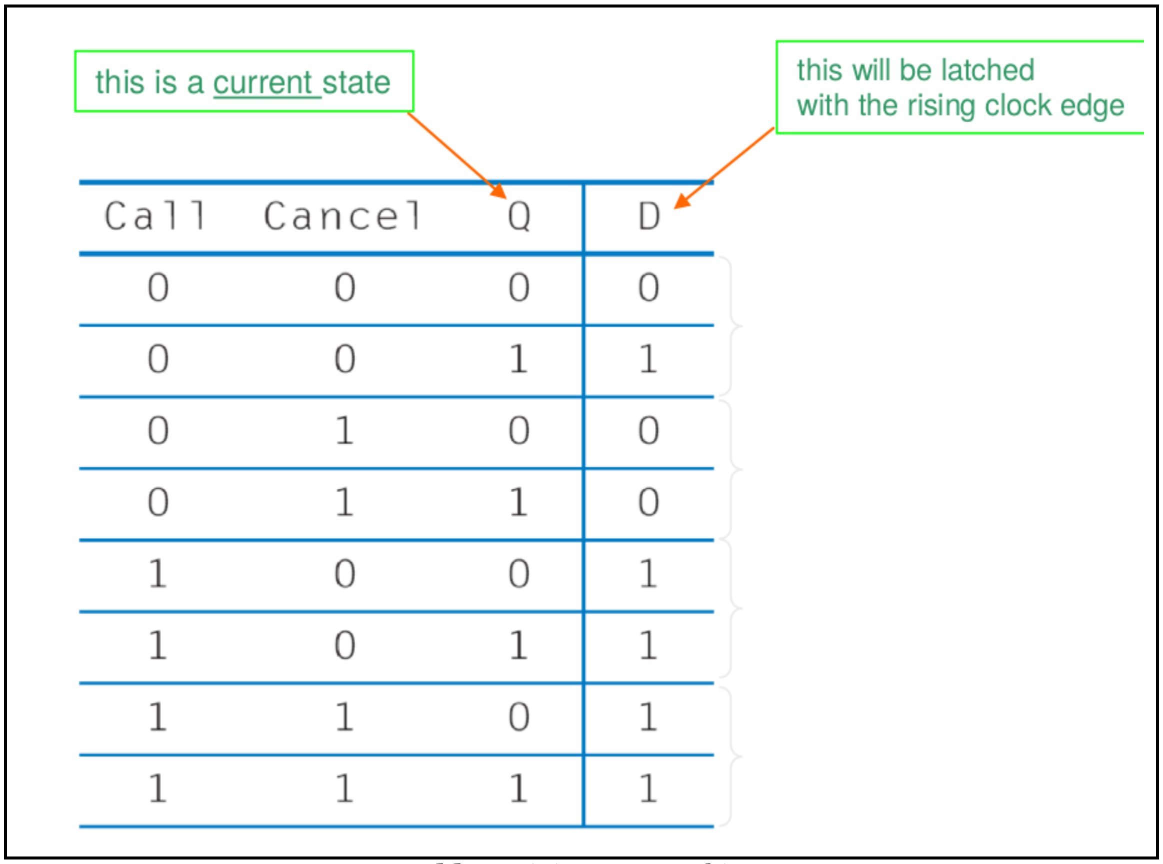

Call button
Cancel button
Bit Storage
Blue
1. Call button pressed – light turns ON
Call button
Cancel button
Bit Storage
Blue
2. Call button released – light STAYS ON
Call button
Cancel button
Bit Storage
Blue
3. Cancel button pressed – light turns OFF

Figure 1. Flight Attendant System State Machine Description

<u>System Analysis and Implementation</u>

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

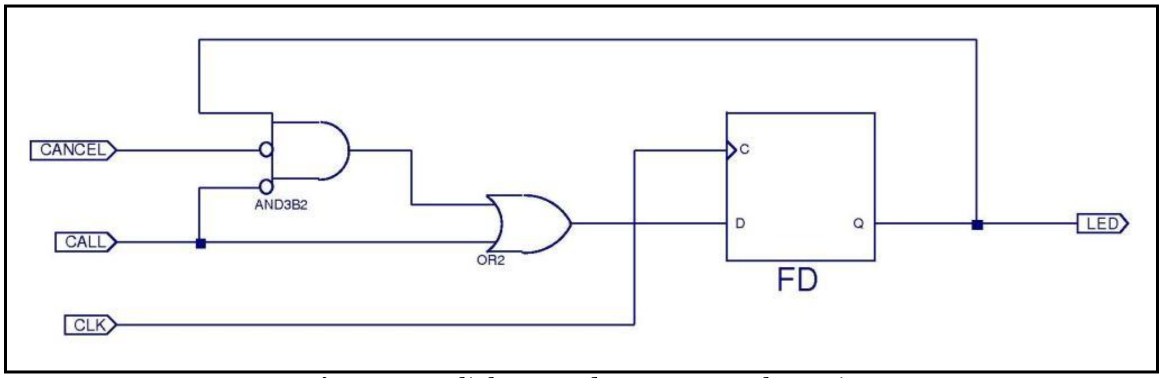


Figure 2. Flight Attendant System Schematic

Conduct the Behavioral Simulation

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 both structural or behavioral modeling, in this lab our aim is to practice behavioral modeling. The following is the module interface of the required system.

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

reg c_state ;

// Combinatorial block
always @(*) begin

        case ({call_button,cancel_button})

        2'b00:  Your code ;
        2'b01:  Your code ;
        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 systems has three inputs 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. Once the Verilog module has been implemented, your can use the test bench form part A in the task of validating your code. In order to synthesize your Verilog code your will require to map the call and cancel button to the buttons in the board as described in part A.

Demonstration

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

PART 2. Rising-edge Detector

Objective

In this assignment, it is required to construct a Finite State Machine (FSM) state/output diagram, derive excitation equations and implement it on the Basys Board.

Specification

In this part your job is to implement the proposed FSM developed in verilog. To facilitate the process of testing the code, your FSM module should have the following ports.

```
module edgedetector_bh(
 input  wire clk,
 input  wire signal,
 output reg outedge    );

wire slow_clk ;

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

localparam ZERO = 'd0;
localparam CHANGE = 'd1;
localparam ONE = 'd2;

// http://www-inst.eecs.berkeley.edu/~cs150/sp12/agenda/lec/lec17-FSM.pdf

clkdiv c1(clk, slow_clk );

// Comb. logic.

always @(*) begin
```

```
 case (r_state)

ZERO   : Your code ;
CHANGE  : Your code ;
ONE  :  Your code ;

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

 endcase

end

//  --------------------------------
// Seq. logic
//  --------------------------------
always @( posedge slow_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);

 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.

PART 3 : LED Display Time Multiplexing Circuit

The Digilent Basys Board contains four seven segment LED displays with decimal points. To reduce the number of used of FPGA's I/O pins 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.

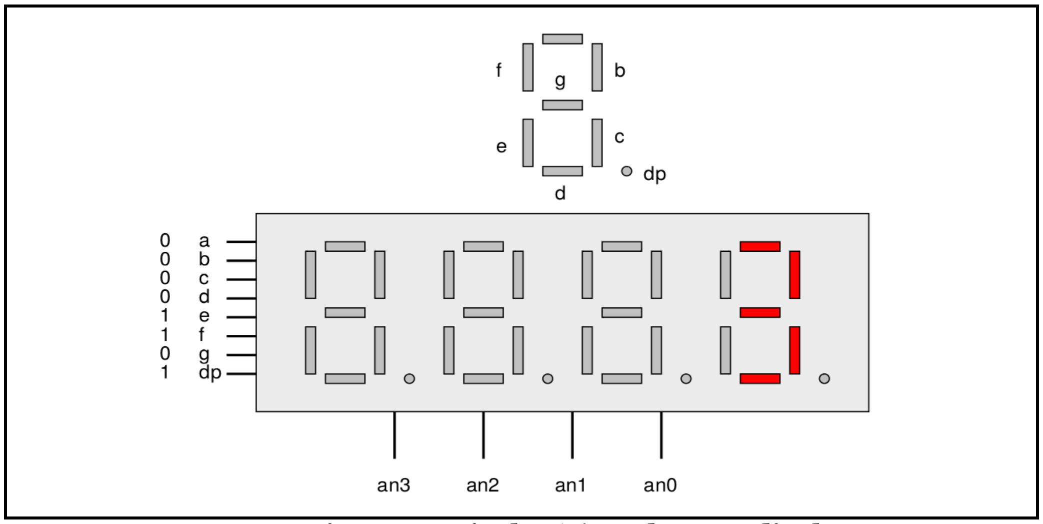


Figure 7. Display '3' on the LED display

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 digram 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.

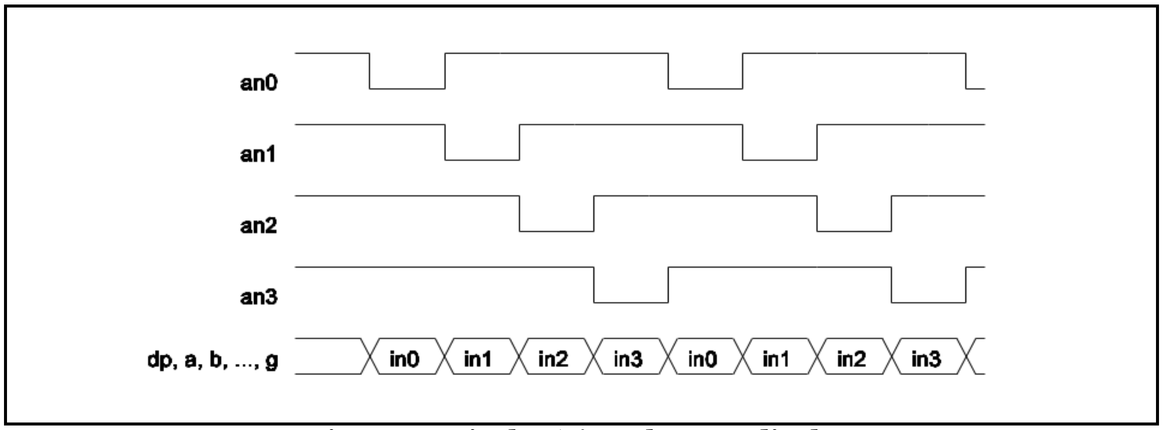


Figure 8. Display '3' on the LED display

One of possible realizations is shown in the block diagram of Figure 9 . Use it as a guide to implement the circuit and verify (simulate ONLY) its performance

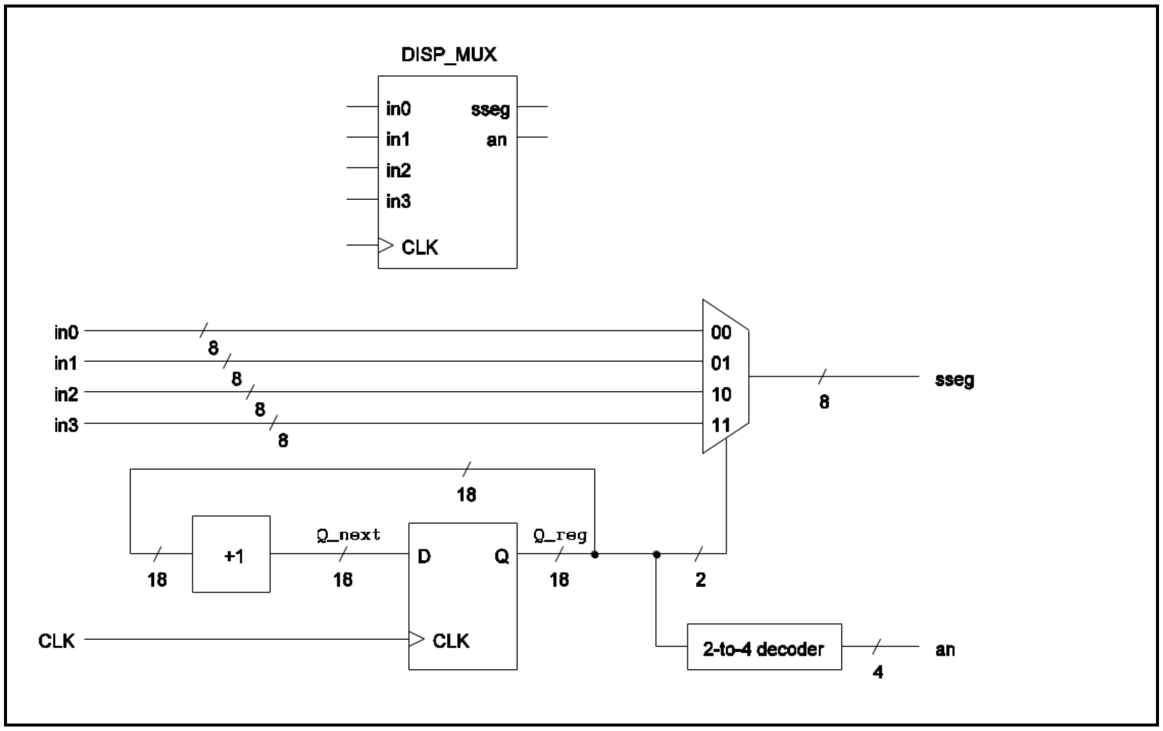


Figure 8. Symbol and Block diagram of the time-multiplexing circuit.

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

// --------------------------------
// 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
```

Questions & Deliverables

1. In part 1, what will happen if the “clock” signal is of very low frequency (1 Hz)?

2.In part2, shows the FSM diagram of rising edge detector?

3.In part2, how you can tune the frequency of slow clock?

4. Design test benches for all three parts and show the waveform for each (Please upload the source code as well).