Cover Page

Name: Ethan Lingad

Student ID: 862268541

Course Title: CS120A

Lab Section: 22

Lab Title: Lab 04. Sequential Logic Design

Overview:

In this lab, we were given three parts to complete each having a specific function. In the first part, we were instructed to implement a sequential circuit that was discussed in class. Then for part 2, we were instructed to design and implement a state machine. Then for part 3, we were instructed to design a time multiplexing circuit for a four-LED display.

Analysis:

For Part 1:

1. Copy and paste the given code
2. Fill in the missing code
3. analyze the truth table that was given that is representing the Flight Attendant System

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

For Part 2:

1. Instructed to build a rising edge detector
2. copy and paste the given code

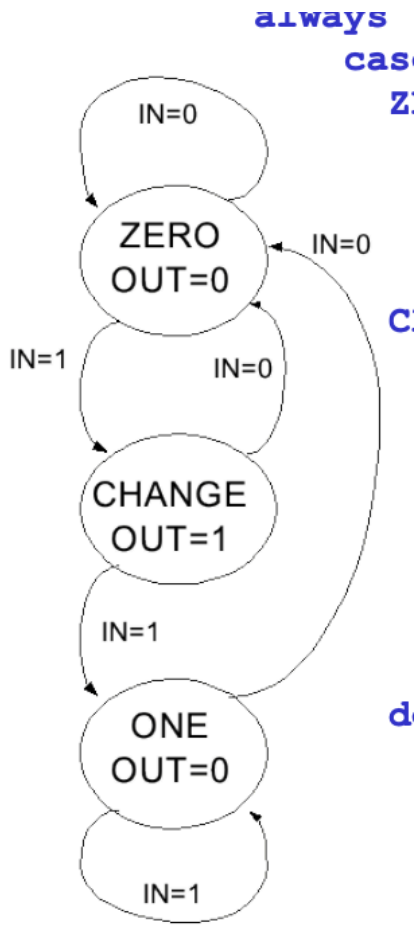
3. Use this link to help understand the FSM design and code

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

4. Fill in the missing code

Demonstration-1:

The state diagram was provided in the link that was given in the verilog code



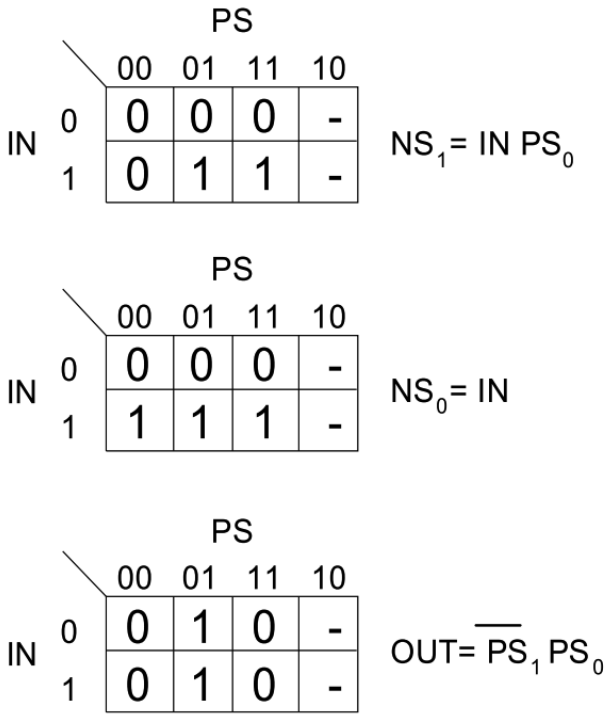
Demonstration-2:

Transition Table			
NextStateOutputState Input	D1	D0	Output

000	0	0	0
001	0	1	0
010	0	0	1
011	1	1	1
110	0	0	0
111	1	1	0

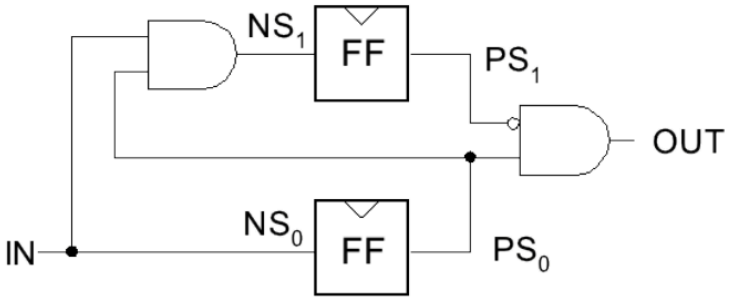
Demonstration-3:

Excitation Equations



Demonstration-4:

Sequential Logic Result



Records:

Part 1:

Design.sv

```
// Code your design here
`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
```

testbench.sv

```
`timescale 1ns/1ns;


module testbench;

      //list of input ports..
      reg clk;
      //add other input ports that you will need. For example the
following is for
      //flight attendant system.
      reg Call;
      reg Cancel;
      // add the required input ports for the module you are testing.

      //list of output ports..
      wire Light; //add output ports as required to test your module

      //instantiate your module to be tested here and connect ports
appropriately
      //your_module_name uut
(.port_name_in_module(port_name_in_testbench),...)
      //As an example module instantiation to test flight attendant
call system is given below:

      fasystem_bh uut(.call_button(Call), .cancel_button(Cancel),
.light_state(Light), .clk(clk));

      always #1 clk=~clk; // this will create a clock signal with fr
equency of(1/2ns=500MZ)

      initial begin
```

```
            $dumpfile("your_vcd_file_name.vcd");
            $dumpvars;

            clk=0; //this will initialize clk to logic 0 at time 0ns

            //Start giving stimulation to input ports as required to
test your module here.
            //A sample stimulus set to test flight attendant call
system
            // is given below as an example:

            Call=0; Cancel=0;
            #17;  //wait for appropriate amount of time so that you
can see

            #13;
            Call=1; Cancel=1;
            #27;
            Call=0; Cancel=1;
            #29;

            $finish; //this is very important. Otherwise your test
will
            //never stop as clock will keep running infinitely
according to line 23.

      end
endmodule
```

Part 2:

```
// Code your design here
`timescale 1ns/1ns;
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 ;
// Define your FSM states
localparam ZERO = 'd0;
localparam CHANGE = 'd1;
localparam ONE = 'd2;
// 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 slow_clk ) begin
r_state <= c_state ;
end
```

```
endmodule


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
```

Part 3:

```
// 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
```

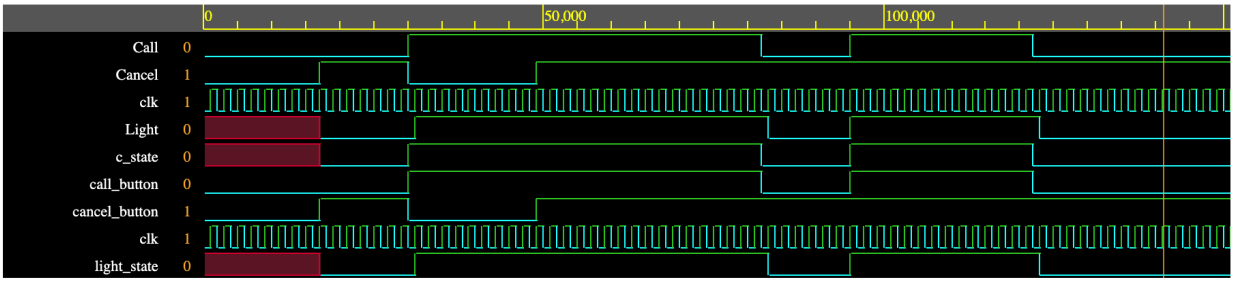
```
4'b1111 : begin // F
a = 1'b0 ;
e = 1'b0 ;
f = 1'b0 ;
g = 1'b0 ;
end
endcase
end
endmodule
```

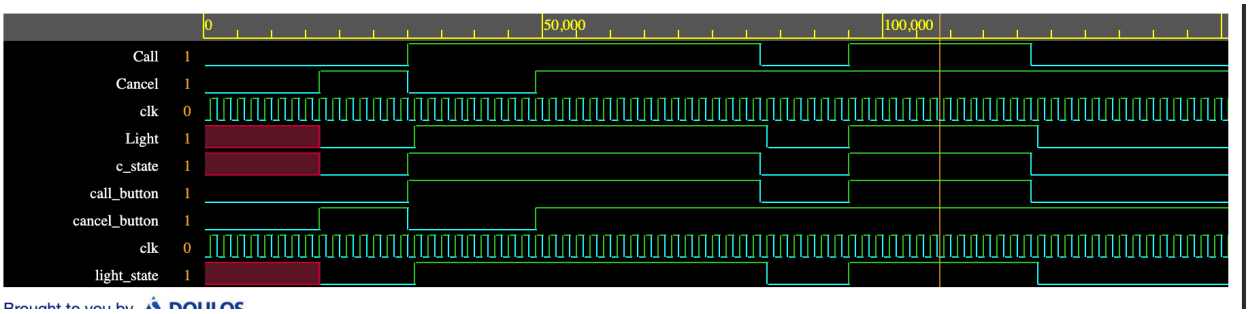
```
module dispmux_main_bh(
input clk , // Clock signal
input sw0, // Switch input
input sw1, // Switch input
input sw2, // Switch input
input sw3, // Switch input
output [3:0] an , // LED selector
output [7:0] sseg // Segment signals
 );
wire [7:0] in0;
wire [7:0] in1;
wire [7:0] in2;

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] );

bcdto7led_bh c2(sw0, sw1, sw2, sw3,
                in1[0],in1[1],in1[2],in1[3],
in1[4],in1[5],in1[6],in1[7] );
```

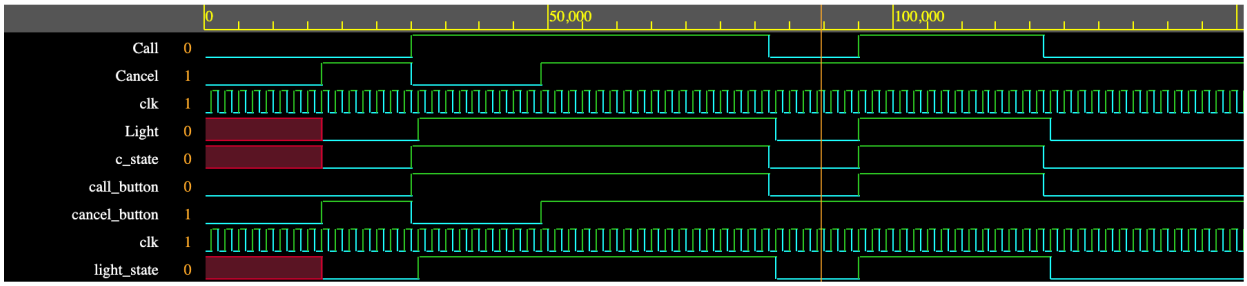
```
 bcdto7led_bh c3(sw0, sw1, sw2, sw3,
                 in2[0],in2[1],in2[2],in2[3],
in2[4],in2[5],in2[6],in2[7] );

bcdto7led_bh c4(sw0, sw1, sw2, sw3,
                in3[0],in3[1],in3[2],in3[3],
in3[4],in3[5],in3[6],in3[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


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 ;
```

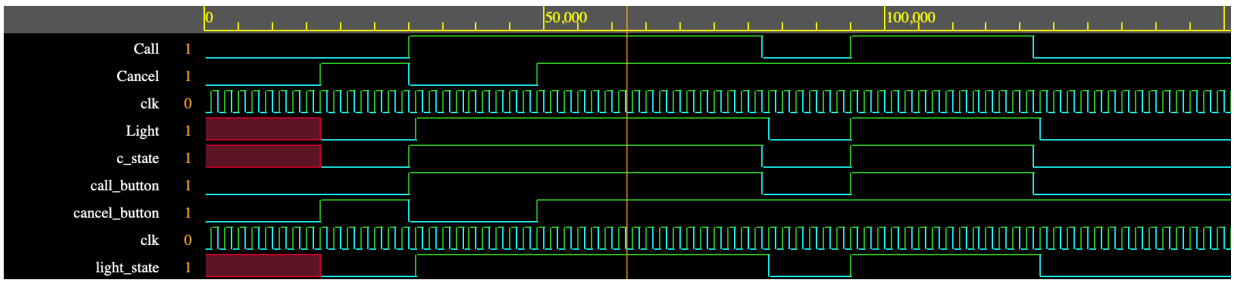
```
2'b10 : sseg = in2 ;
2'b11 : sseg = in3 ;
endcase
end
// Decoder ***********************************
always @(*) begin
case (r_qreg[1:0])
2'b00 : an = ~(4'b0001) ;
2'b01 : an = ~(4'b0010) ;
2'b10 : an = ~(4'b0100) ;
2'b11 : an = ~(4'b1000) ;
endcase
end
// Counter ***********************************
always @(*) begin
c_next = r_qreg + 'd1;
end
always @(posedge clk) begin
 r_qreg <= c_next ;
end
endmodule
```

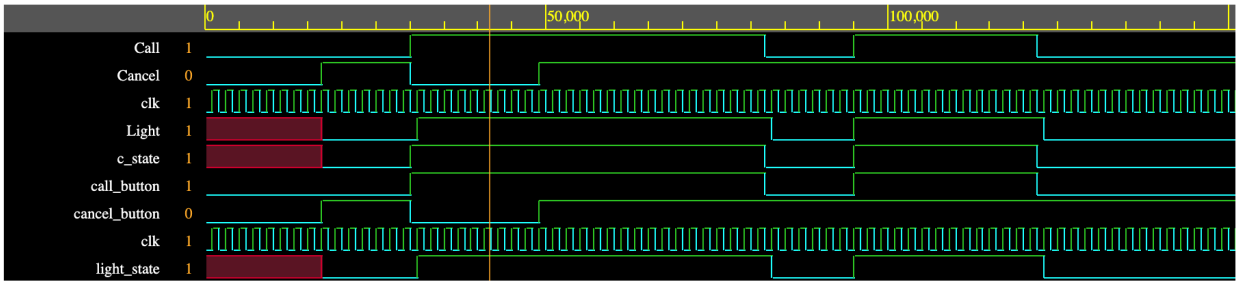
waveforms for part 1:

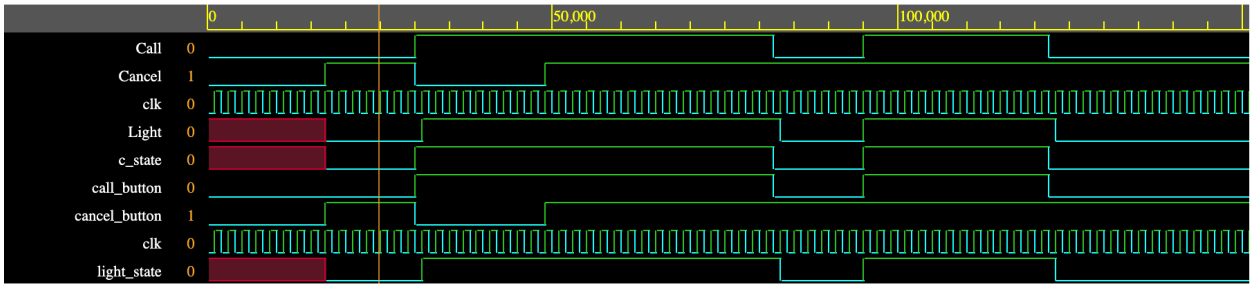


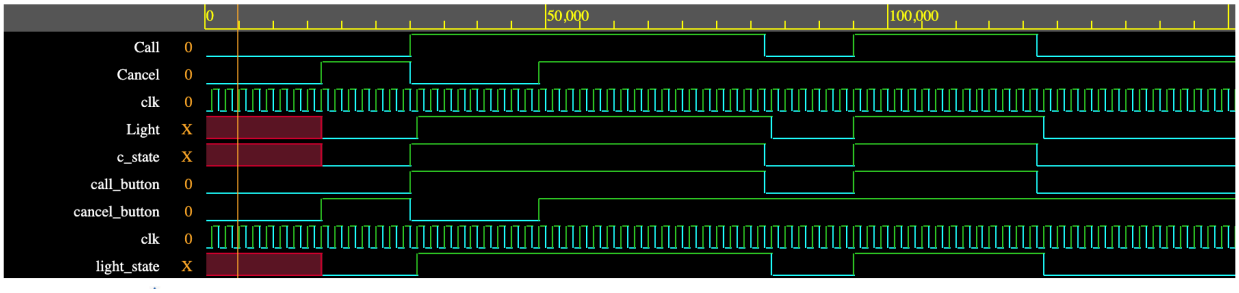



0
50,000
100,000
Call 0
Cancel 1
clk 1
Light 0
c_state 0
call_button 0
cancel_button 1
clk 1
light_state 0


0
50,000
100,000
Call 1
Cancel 1
clk 0
Light 1
c_state 1
call_button 1
cancel_button 1
clk 0
light_state 1


0
50,000
100,000
Call 1
Cancel 0
clk 1
Light 1
c_state 1
call_button 1
cancel_button 0
clk 1
light_state 1


0
50,000
100,000
Call 0
Cancel 1
clk 0
Light 0
c_state 0
call_button 0
cancel_button 1
clk 0
light_state 0


0
50,000
100,000
Call 0
Cancel 0
clk 0
Light X
c_state X
call_button 0
cancel_button 0
clk 0
light_state X

Discussion:

The system does work according to the specifications. According to the output of the waveforms, the result for the given code shows the correct output. During the lab, I didn't encounter any issues with Verilog. This means that I was able to do production without any complications. As of now, I don't think there is any way to improve the systems as it doesn't seem necessary.

Conclusion:

The purpose of this lab is to understand sequential circuits and how they can be implemented in practice. It's important to understand how memory works and how combinational logic can be incorporated into making sequential circuits. It is also important to understand how to follow certain guidelines when creating circuits like creating finite state machines.

Questions:

Part 1:

If the "clock" signal is of very low frequency, then the sequential circuit will become very slow as a result. It is better to have higher frequency in order to have faster responses. The sequential circuit will be much smoother if it has a higher frequency.