Lab 11

Top module

```
module top(
    input clk,
    input rst,
    input enter,
    inout [7:0] JA,
    output [3:0] anode,
    output [7:0] cathode,
    output [3:0] ps
    );

    wire clk_190;
    wire press,clr,flg;
    wire [3:0] num;
    wire [1:0] out;
    reg [3:0] ones=0,tens=0,hundreds=0,thousands=0;
    reg [3:0] count;

    always @ (posedge flg)
    begin
        if(out==2'd1)               // for PASS
        begin
            ones<=4'd2;
            tens<=4'd2;
            hundreds<=4'd0;
            thousands<=4'd1;
        end
```

```
        else if(out==2'd2)          // for FAIL
        begin
            ones<=4'd7;
            tens<=4'd3;
            hundreds<=4'd0;
            thousands<=4'd6;
        end
        else if(out==2'd3)          // for INV
        begin
            ones<=4'd5;
            tens<=4'd4;
            hundreds<=4'd3;
            thousands<=4'd8;
        end
        else                        // for nothing
        begin
            ones<=4'd8;
            tens<=4'd8;
            hundreds<=4'd8;
            thousands<=4'd8;
       end
    end

    frq_div #(19) fun1(clk,clk_190);
    debounce fun2(clk_190,clr,rst);
    debounce fun3(clk_190,press,enter);
    decoder
fun4(.clk(clk),.row(JA[7:4]),.col(JA[3:0]),.dout(num),.flg(flg));
    fsm fun5(clk,clr,press,num,flg,out,ps);
```

```
    sevenseg
fun6(.clk(clk),.clr(clr),.ones(ones),.tens(tens),.hundreds(hundreds)
,.thousands(thousands),.cathode(cathode),.anode(anode));

endmodule
```

Frequency divider

```
module frq_div(
    input clk,
    output div_clk
    );

parameter n=25;

reg [n-1:0] q=0;

always @ (posedge clk)
begin

q<=q+1;

end

assign div_clk=q[n-1];

endmodule
```

Debounce

```
module debounce(
    input clk,
    output deb_out,
    input deb_in
    );

    reg D1,D2,D3;
    always@(posedge clk)
    begin
        D1<=deb_in;
        D2<=D1;
        D3<=D2; // inversion is done to generate a pulse
    end

    assign deb_out=D1&&D2&&D3;

endmodule
```

Decoder

```
module decoder(
    input clk,
    input[3:0] row,
    output reg[3:0] col,
    output reg [3:0] dout,
    output flg
```

```
);

reg[19:0] count;  // 20 bit counter

reg flg=0,flg1=0,flg2=0,flg3=0,flg4=0;

always @(posedge clk)

begin

count<=count+1;

end

always@(posedge clk)

begin

   if(count==20'b00011000011010100000)   // 100k i.e 1khz clk
or 1ms time

      begin

      col<=4'b0111; // selecting column 1

      end

   else if(count==20'b00011000011010101000) // 100008
```

```
        begin

        if(row==4'b0111)
        begin
           dout<=4'b0001;
           flg1<=1;
        end
        else if(row==4'b1011)
        begin
           dout<=4'b0100;
           flg1<=1;
        end
        else if(row==4'b1101)
        begin
           dout<=4'b0111;
           flg1<=1;
        end
        else if(row==4'b1110)
        begin
           dout<=4'b0000;
           flg1<=1;
        end
        else
           flg1<=0;
        end

      else if(count==20'b00110000110101000000)   // 500 Hz clk
or 2ms time
```

```
    begin

    col<=4'b1011; // selecting column 2

    end

else if(count==20'b00110000110101011000) // 508

    begin

    if(row==4'b0111)
    begin
       dout<=4'b0010;    //2   //checking row wise
       flg2<=1;
    end
    else if(row==4'b1011)
    begin
       dout<=4'b0101;  //5
       flg2<=1;
    end
    else if(row==4'b1101)
    begin
       dout<=4'b1000;  // 8
       flg2<=1;
    end
    else if(row==4'b1110)
    begin
       dout<=4'b1111;   ///
       flg2<=1;
```

```
                end
                else
                   flg2<=0;
                end

        else if(count==20'b00111000011010100000)   // 500 Hz clk
or 2ms time

                 begin

                 col<=4'b1101; // selecting column 3

                 end

        else if(count==20'b00111000011010101000)

          begin

          if(row==4'b0111)
          begin
             dout<=4'b0011;    //3   //checking row wise
             flg3<=1;
          end
          else if(row==4'b1011)
          begin
             dout<=4'b0110;  //6
             flg3<=1;
          end
          else if(row==4'b1101)
```

```
        begin
           dout<=4'b1001;  // 9
           flg3<=1;
        end
        else if(row==4'b1110)
        begin
           dout<=4'b1110;   //E
           flg3<=1;
        end
        else
           flg3<=0;
        end

      else if(count==20'b01100001101010000000)   // 250 Hz clk
or 4ms time

                  begin

                  col<=4'b1110; // selecting column 4

                  end

        else if(count==20'b01100001101010001000)

          begin

          if(row==4'b0111)
          begin
              dout<=4'b1010;    //A   //checking row wise -Row 1
```

```
            flg4<=1;
        end
        else if(row==4'b1011)
        begin
            dout<=4'b1011;  //B  -row 2
            flg4<=1;
        end
        else if(row==4'b1101)
        begin
            dout<=4'b1100;  // C  -row 3
            flg4<=1;
        end
        else if(row==4'b1110)
        begin
            dout<=4'b1101;   //D   - row-4
            flg4<=1;
        end
        else
            flg4<=0;
        end

    end

    always @ (posedge clk)
    begin
      flg<=flg1 || flg2 || flg3 || flg4;
    end

Endmodule
```

Seven Segment Display

```
module sevenseg(
    input clk,
    input clr,
    input [3:0] ones,
    input [3:0] tens,
    input [3:0] hundreds,
    input [3:0] thousands,
    output [7:0] cathode,
    output [3:0] anode
    );
     reg [6:0]sseg_temp;
     reg [3:0]an_temp = 4'b1110 ;
     reg [17:0] count; //the 18 bit counter which allows us to
multiplex at 1000Hz

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

         //code for display multiple digits (do not initialize an_temp in
line 41 and comment out lines 44 to 60)
     always @ (*)
      begin
       case(count[17:16]) //using only the 2 MSB's of the counter

        2'b00 :  //When the 2 MSB's are 00 enable the fourth display
```

```
begin
 case(ones)
      4'd0 : sseg_temp = 7'b0001000; //to display A
      4'd1 : sseg_temp = 7'b0011000; //to display P
      4'd2 : sseg_temp = 7'b0100100; //to display S
      4'd3 : sseg_temp = 7'b1001111; //to display I
      4'd4 : sseg_temp = 7'b1101010; //to display N
      4'd5 : sseg_temp = 7'b1000001; //to display V
      4'd6 : sseg_temp = 7'b0111000; //to display F
      4'd7 : sseg_temp = 7'b1110001; //to display L
      4'd8 : sseg_temp = 7'b0000000; //to display 0
      default : sseg_temp = 7'b1111110; //dash
  endcase

 //sseg = ones;
 an_temp = 4'b1110;

end

2'b01:  //When the 2 MSB's are 01 enable the third display
 begin
    case(tens)
      4'd0 : sseg_temp = 7'b0001000; //to display A
      4'd1 : sseg_temp = 7'b0011000; //to display P
      4'd2 : sseg_temp = 7'b0100100; //to display S
      4'd3 : sseg_temp = 7'b1001111; //to display I
      4'd4 : sseg_temp = 7'b1101010; //to display N
      4'd5 : sseg_temp = 7'b1000001; //to display V
      4'd6 : sseg_temp = 7'b0111000; //to display F
```

```
        4'd7 : sseg_temp = 7'b1110001; //to display L
        4'd8 : sseg_temp = 7'b0000000; //to display 0
        default : sseg_temp = 7'b1111110; //dash
      endcase

  //sseg = tens;
  an_temp = 4'b1101;
 end

2'b10:  //When the 2 MSB's are 10 enable the second display
 begin
  case(hundreds)
        4'd0 : sseg_temp = 7'b0001000; //to display A
        4'd1 : sseg_temp = 7'b0011000; //to display P
        4'd2 : sseg_temp = 7'b0100100; //to display S
        4'd3 : sseg_temp = 7'b1001111; //to display I
        4'd4 : sseg_temp = 7'b1101010; //to display N
        4'd5 : sseg_temp = 7'b1000001; //to display V
        4'd6 : sseg_temp = 7'b0111000; //to display F
        4'd7 : sseg_temp = 7'b1110001; //to display L
        4'd8 : sseg_temp = 7'b0000000; //to display 0
        default : sseg_temp = 7'b1111110; //dash
    endcase

  //sseg = hundreds;
  an_temp = 4'b1011;
 end

2'b11:  //When the 2 MSB's are 11 enable the first display
```

```
        begin
         case(thousands)
              4'd0 : sseg_temp = 7'b0001000; //to display A
              4'd1 : sseg_temp = 7'b0011000; //to display P
              4'd2 : sseg_temp = 7'b0100100; //to display S
              4'd3 : sseg_temp = 7'b1001111; //to display I
              4'd4 : sseg_temp = 7'b1101010; //to display N
              4'd5 : sseg_temp = 7'b1000001; //to display V
              4'd6 : sseg_temp = 7'b0111000; //to display F
              4'd7 : sseg_temp = 7'b1110001; //to display L
              4'd8 : sseg_temp = 7'b0000000; //to display 0
              default : sseg_temp = 7'b1111110; //dash
         endcase
         //sseg = thousands;
         an_temp = 4'b0111;
        end
       endcase
      end
     assign anode = an_temp;
     assign cathode = {sseg_temp, 1'b1};
 endmodule
```

Door Lock

```
module fsm(
    input clk,
    input clr,
    input enter,
```

```
    input [3:0] num,
    input flg,
    output reg [1:0] out=0,
    output [3:0] ps
    );

    parameter [3:0]
s0=4'd0,s1=4'd1,s2=4'd2,s3=4'd3,e1=4'd4,e2=4'd5,e3=4'd6,e4=4'
d7;
    reg [3:0] ps=0,ns=0;

    always @ (posedge flg or posedge clr)
    begin
        if(clr)
            ps<=s0;
        else
            ps<=ns;
    end

    always @ (posedge flg)
    begin
        case(ps)

            s0: begin
                if(~enter)
                    if(num==4'd12)
                        ns<=s1;
                    else
                        ns<=e1;
```

```
    else
        ns<=e4;
    end

s1: begin
    if(~enter)
        if(num==4'd12)
            ns<=s2;
        else
            ns<=e2;
    else
        ns<=e4;
    end

s2: begin
    if(~enter)
        if(num==4'd13)
            ns<=s3;
        else
            ns<=e3;
    else
        ns<=e4;
    end

s3: begin
    if(enter)
        ns<=s3;
    else
        ns<=e4;
```

```
            end

        e1: begin
            if(~enter)
                ns<=e2;
            else
                ns<=e4;
            end

        e2: begin
            if(~enter)
                ns<=e3;
            else
                ns<=e4;
            end

        e3: begin
                if(enter)
                    ns<=e3;
                else
                    ns<=e4;
            end

        e4: ns<=e4;

    endcase
end

always @ (posedge enter)
```

```
    begin
        if(ps==s3)
            out<=2'b01;             // for PASS
        else if(ps==e3)
            out<=2'b10;             // for FAIL
        else if(ps==e4)
            out<=2'b11;             // for INV
        else
            out<=2'b00;             // for nothing
    end

endmodule
```

XDC File

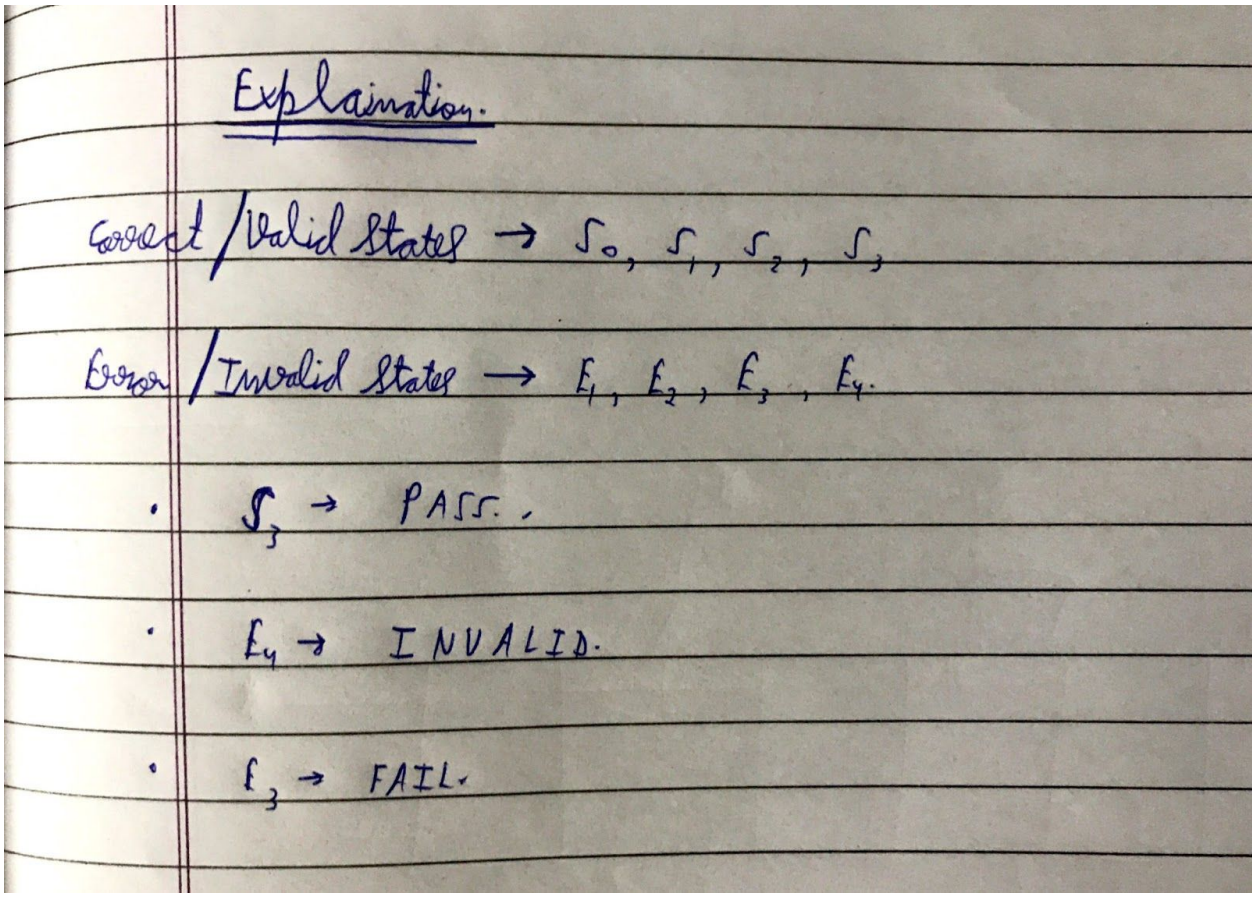
```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {ps[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ps[0]}]
set_property PACKAGE_PIN E19 [get_ports {ps[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ps[1]}]
set_property PACKAGE_PIN U19 [get_ports {ps[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ps[2]}]
```

```
#7 segment display
set_property PACKAGE_PIN W7 [get_ports {cathode[7]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[7]}]
set_property PACKAGE_PIN W6 [get_ports {cathode[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[6]}]
set_property PACKAGE_PIN U8 [get_ports {cathode[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[5]}]
set_property PACKAGE_PIN V8 [get_ports {cathode[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[4]}]
set_property PACKAGE_PIN U5 [get_ports {cathode[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[3]}]
set_property PACKAGE_PIN V5 [get_ports {cathode[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[2]}]
set_property PACKAGE_PIN U7 [get_ports {cathode[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[1]}]
set_property PACKAGE_PIN V7 [get_ports cathode[0]]
   set_property IOSTANDARD LVCMOS33 [get_ports cathode[0]]

set_property PACKAGE_PIN U2 [get_ports {anode[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode[0]}]
set_property PACKAGE_PIN U4 [get_ports {anode[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode[1]}]
```

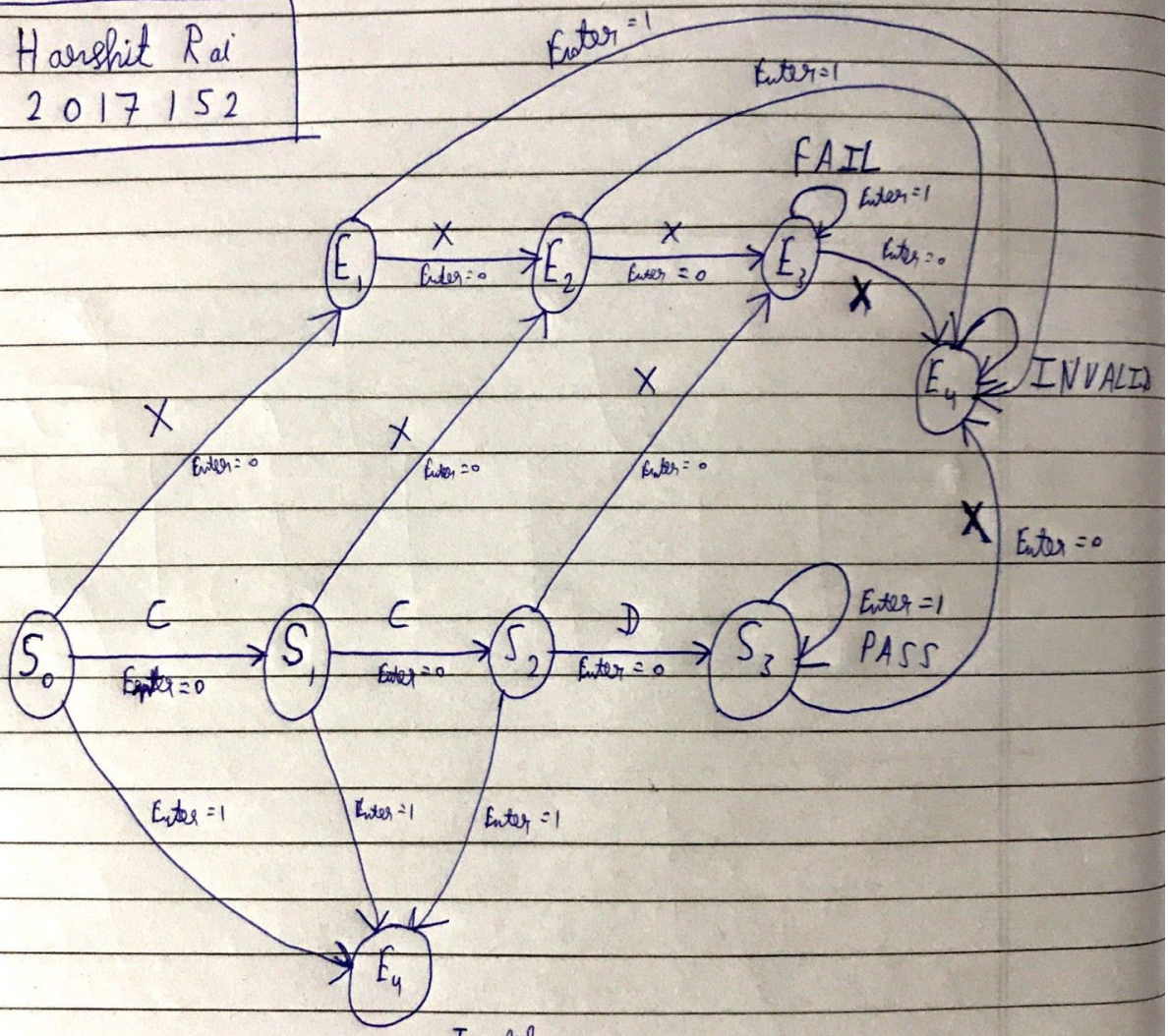
```
set_property PACKAGE_PIN V4 [get_ports {anode[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode[2]}]
set_property PACKAGE_PIN W4 [get_ports {anode[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {anode[3]}]

##Buttons
set_property PACKAGE_PIN U18 [get_ports rst]

       set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property PACKAGE_PIN T18 [get_ports enter]

   set_property IOSTANDARD LVCMOS33 [get_ports enter]

#Pmod Header JA
#Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {JA[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {JA[0]}]
#Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {JA[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {JA[1]}]
#Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {JA[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {JA[2]}]
#Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {JA[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {JA[3]}]
#Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {JA[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {JA[4]}]
```

```
#Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports {JA[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {JA[5]}]
#Sch name = JA9
set_property PACKAGE_PIN H2 [get_ports {JA[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {JA[6]}]
#Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports {JA[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {JA[7]}]
```

FSM



Harshit Rai
2017152



Invalid.