ELEC5566M FPGA DESIGN FOR SYSYTEM-ON-CHIP MINI PROJECT REPORT

DIGITAL LOCK USING FSM

Phaneeshwar Mallakkagari
SID: 201387234

Contents

Phaneeshwar Mallakkagari
SID: 201387234

1.Abstract

This report covers the Design, implementation and verification of a digital lock programmed using Verilog in FPGA. The module is highly parameterized with a range of different parameters used in both top-level entity and other submodules for a wide range of clock frequencies. Finite state machine is used to detect the correct sequence and this sequence is known by the user who sets up the password. The implemented Design is successfully verified with the simulation tool.

2.Introduction

In the present times, it is seen that the communication networks mostly use digital data rather than analogue because of its accuracy and higher speed. Hence digital data transmission is extensively used in the FPGA board. This Design aims to unlock the digital lock or vice versa if the correct binary code is entered as an input. In this system of Design, I have used a Mealy machine, as these FSM machines are generally sequential circuits and used as a controller. This task was demonstrated and controlled the operation by utilizing the pushbuttons and switches that are embedded on De1-SoC development board. Constructed Design is feasible for the users to control various parameters like the length of the password sequence, timers, and clock divided frequency.

3.Design Flow

The circuit design constructed is feasible for any length of the input sequence. Moreover, the inputs for this Design are clock, reset, four input keys, password control switch. The flow of this system begins by setting up the correct password using the password module, which can be accessed by setting up switch2 bit high. Once the password is set the user can turn off the switch2, this makes the Design to act as a sequence detector. When the input sequence is matched with the password sequence, it makes the output bit high, and the respective series of letters are displayed on the Hex display. Another additive feature of this Design is that the FSM machine would enter to error state if the user does not give any input to the device for a certain amount of time while locking or unlocking the system. This design gives flexibility for the user to change parameters according to the requirements to obtain desires results at the output. Furthermore, the constructed design has different functional modules, as a designer point of view, it would be efficient way of solving a problem by dividing the workload into various modules and later connect them in the top-level entity. This gives clarity to view the design and for debugging any glitches in the circuit.

4.Code Design

4.1 FSM logic

In this part of the code I have designed a four-state mealy machine for detecting n-bit sequence. in this design the next-state logic creates a signal 'next' based on the current state and inputs. This block is implemented in combinational logic. While the state memory holds the current state and the current state is updated with 'next' on the rising edge of the clock. Specifically, this logic block is implemented in sequential logic. While the output logic creates the system outputs. The output logic always depends on the current state and the inputs of the system. It begins with a start state, so when it detects any input signal it will immediately transit to button state and execute following instructions in the conditional loop. After the logical instructions are executed, there is transition from button state to wait state. Operation of this state is to tick the clock when the user does not provide any inputs. If the user presses any key on the board it would reset the timer and moves to button state. If the user does not press any key for about 7 seconds, then the timeout signal would go high and there will be a transition from wait state to error state. When the current state is in the error state, it would display 'error' on the hex display for 5 seconds and move to start state. This delay is set by using another timer module that specifically triggers when it is in error state. If the input sequence matches with the password sequence, then the output bit goes high and respective string of letters are been displayed. When the sequence is matched it would have a transition from button state to start state. This design provides a basic implementation as a sequence detector.

4.2 Password module

This module design is used for setting up the password according to the user needs. If the user needs to change the password anytime during the program run, first the user should set reset switch high and then turn of the reset switch and turn on the password switch. When password switch is turned on it will disable the fsm module and enable the password module. This is achieved with help of 1x2 de-multiplexer circuit. A simple Fsm logic is designed even for this module. It begins with initialize state, where in this state all the register variables are initialized and then it moves to Idle state. When the user press key0, there is a transition from idlestate to checkstate and then 2'b00 is stored in the sequence register. later it increments the counter and width of sequence register to accumulate next set of input bits. It moves back to idlestate after it executes specific instructions in checkstate. Similarly, if key1 is pressed then 2'b01 is stored in the sequence register and there is a shift in width of the sequence to accumulate the next input, more like a window function. This process is done till counter is equal to length of the password sequence. Once the password is set by the user, the sequence register is outputted to the fsm model.

4.3 timer module

This module design is used to set the specified amount of delay for certain task. Input clock frequency for this module is 1hz. From the design code it is seen that timer1 is declared as parameter, which represents amount of delay required by the user in seconds. Two timer blocks were called in the design, one of the blocks is used for checking the delay between two successive input sequence and another block is used with a delay of 5 seconds to display the error output.

4.4 output hex module

This module is specifically designed to control output lines coming from fsm module and password module. When the user is accessing the password module to set the password, then this design would convert respective decimal number into respective BCD using display function, for displaying it on the Hex Led. It connects the output wires coming out from password module to the actual output lines of the top-level entity. In a similar approach, this module would connect output wires of the fsm module to the outputs defined in top level entity.

4.5 frequency divider module

This module design is used to convert from one set of frequency to a required frequency. Here in this module the required output and input frequency are declared as a parameter, hence we could change the output frequency depending on the user requirement. In my design I have set the required frequency to 1hz, has this frequency is essential for two timer blocks used in the design. Here the output clock frequency is calculated to meet the requirements and it is given as follows,

Number_of_clk_cycles = 50000000/1 = 50000000

Low_output = High_output = 50000000/2 = 25000000

From the above equation it is seen that the frequency divider module would output high level signal (bit 1) for first 25 million cycles and for next 25 million cycles it would output low level signal (bit 0). And the process repeats till the input clock signal is given.

5. State Table

Current State	Input	Next State	Output (z)
Start	**Key[3:0] == 4’b1110 || Key[3:0] == 4’b1011 || Key[3:0] == 4’b1101 || Key[3:0] == 4’b1110**	**Button**	**0**
Start	**Key [3:0] == 4’b0000**	**start**	**0**

button	**Key[3:0] == 4’b1110 || Key[3:0] == 4’b1011 || Key[3:0] == 4’b1101 || Key[3:0] == 4’b1110**	**wait**	**0**
button	**Ulseq == seq**	**start**	**1**
button	**Ulseq != seq**	**error**	**0**
wait	**t1in = 1’b0 && Key [3:0] == 4’b1111**	**wait**	**0**
wait	**t1in = 1’b1 && Key [3:0] == 4’b1111**	**error**	**0**
wait	**t1in=1’b0 && Key[3:0] == 4’b1110 || Key[3:0] == 4’b1011 || Key[3:0] == 4’b1101 || Key[3:0]== 4’b1110**	**button**	**0**
error	**t2in = 1’b0**	**error**	**0**
error	**t2in = 1’b1**	**start**	**0**
initialize	**-**	**start**	**0**
ANY State	**Reset == 1’b1 | enable0 == 1’b0**	**initialize**	**0**

Table1: State table for Fsm module

Current State	**Input**	**Next state**	**Output**
Initialize	**-**	**Idlestate**	**0**
Idlestate	**Key[3:0] == 4’b1110 || Key[3:0] == 4’b1011 || Key[3:0] == 4’b1101 || Key[3:0]== 4’b1110**	**checkstate**	**1**
Idlestate	**Enable1 == 1’b0**	**Idlestate**	**0**
checkstate	**Key[3:0] == 4’b1111**	**Idlestate**	**0**
Any State	**Reset == 1’b1**	**Initialize**	**0**

Table2: State table for Password module

6. State Diagram

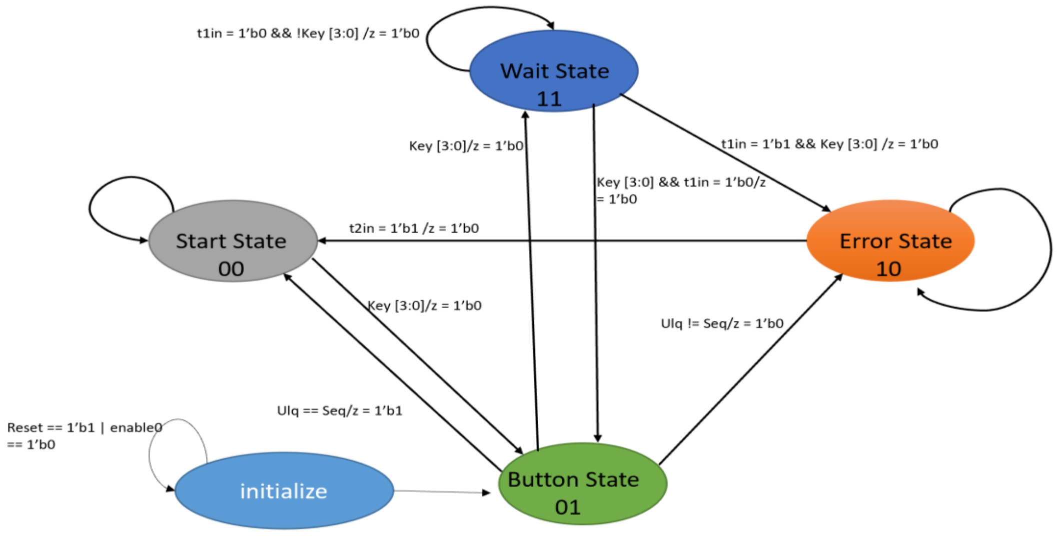


Fig1: Fsm module state diagram

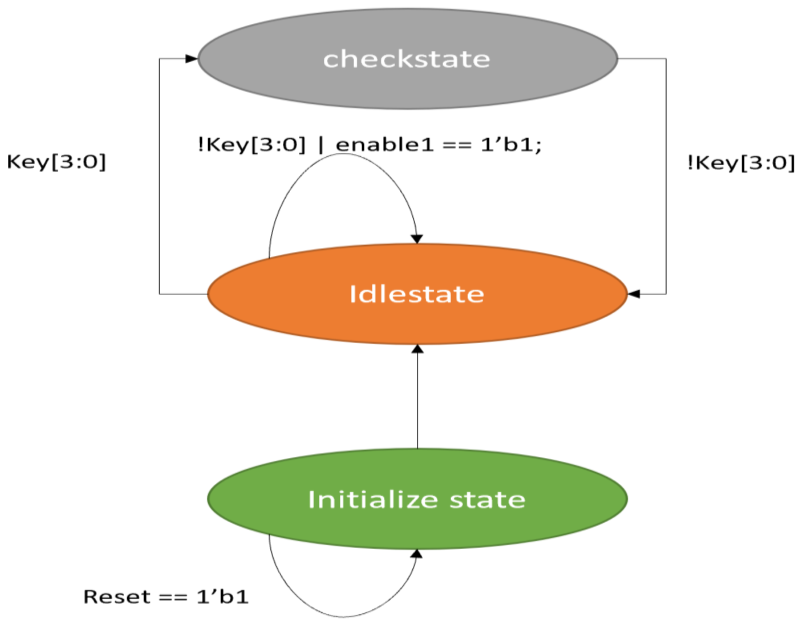


Fig2: Password module Fsm state diagram

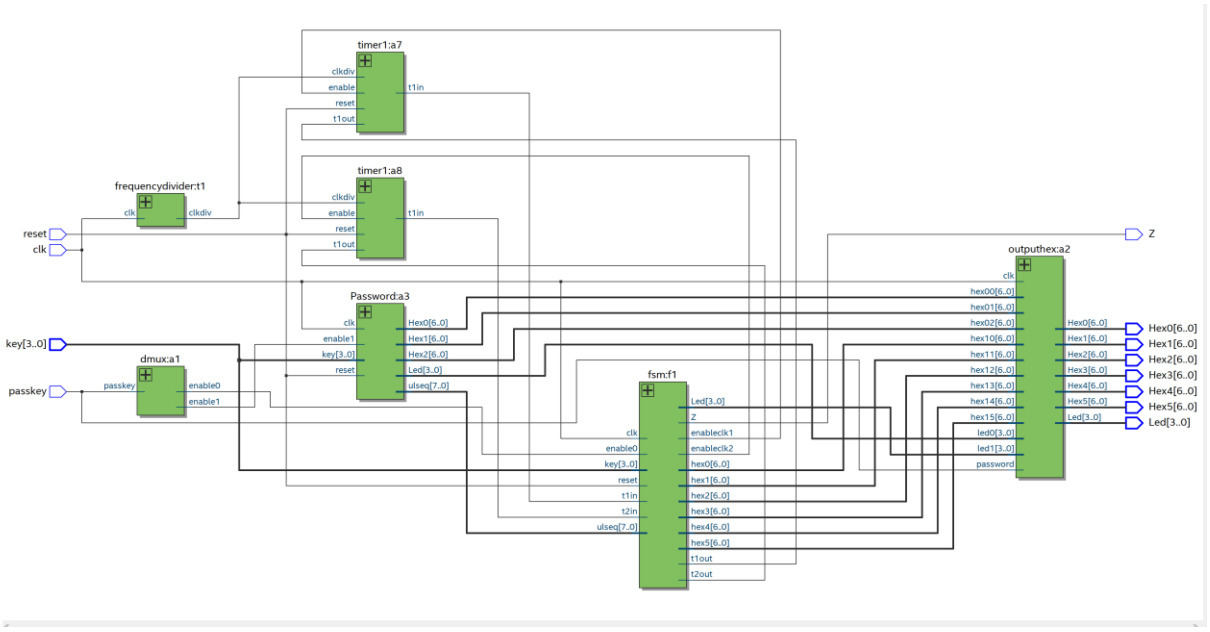
7. Verification

This task is very crucial and important in analyzing any Verilog design. The multi sim tool used is very useful and important for performing timing analysis on the constructed design or circuit. Proper testbench is designed to understand the behavioral flow of the FSM for a given input signals. The testbench of the module is verified and obtained satisfactory results. Verified for different values password sequence and achieved desired results.

8. Conclusion

In conclusion, I have learned many concepts of fsm and effective implementation of circuit level design using Verilog HDL. The code designed to run for different password sequence and achieved desired results at output by testing using only simulation tool. Many challenges were faced while debugging on the hardware but could not achieve the desired results on the board. While the design created is well parameterized and organized various functional modules in the top-level entity.

9. Appendix



RTL viewer: Block Diagram Design

Phaneeshwar Mallakkagari
SID: 201387234

Code:

1)fsm5state.v

```
module fsm5state #(
   parameter nkeys = 4, // we can define any length Password sequence
            parameter timer1 = 7, // to set clock, when the user doesnot enter the key for certain duration
            parameter timer2 = 5, // to set clock when the state machine in FSM module, is in error state
            parameter required_clk_frequency = 1,  // to set frequency of 1 hz
            parameter clk_frequency = 50000000 // Input clock frequency
                )(
            input [3:0] key,  // input push button
            input passkey,   // to control between password module and fsm module
            input reset,    // reset the process
            input clk,      // internal clock input
            output wire Z,    //output goes high when sequence is right
            output wire [3:0] Led,  // to display the button pressed
            output wire [6:0] Hex0,Hex1,Hex2,Hex3,Hex4,Hex5 // to display the output string
            );

localparam req_clk_freq = 4;
wire clkdiv_t1;
wire enable0,enable1;
wire t1out,t1in,t2out,t2in;
wire enableclk1, enableclk2;
wire [(nkeys*2)-1:0] ulseq;
wire [6:0] h10,h11,h12,h13,h14,h15,h00,h01,h02,h03,h04,h05;
wire [3:0] led0,led1;

// Respective connections are made for other modules

dmux a1 (
   .passkey (passkey),
            .enable0 (enable0),
              .enable1 (enable1)
              );


frequencydivider #(
       .required_clk_frequency (required_clk_frequency) ,
                           .clk_frequency (clk_frequency)
                           ) t1 (
                            .clk (clk),
                                       .clkdiv (clkdiv_t1)
);




timer1 #(
         .timer1 (timer1)
                          ) a7(
                          .t1out (t1out) ,
                          .enable (enableclk1),
                          .reset (reset),
                          .clkdiv (clkdiv_t1),
                          .t1in (t1in)

                          );


timer1 #(
         .timer1 (timer2)
                          ) a8(
                          .t1out (t2out) ,
                          .enable (enableclk2),
                          .reset (reset),
                          .clkdiv (clkdiv_t1),
                          .t1in (t2in)
```

```
                       );


outputhex a2 (
     .hex00 (h00),
                       .hex01 (h01),
                       .hex02 (h02),
                       .hex10 (h10),
                       .hex11 (h11),
                       .hex12 (h12),
                       .hex13 (h13),
                       .hex14 (h14),
                       .hex15 (h15),
                       .led0 (led0),
                       .led1 (led1),
                       .password (passkey),
                       .clk (clk),
                       .Led (Led),
                       .Hex0 (Hex0),
                       .Hex1 (Hex1),
                       .Hex2 (Hex2),
                       .Hex3 (Hex3),
                       .Hex4 (Hex4),
                       .Hex5 (Hex5)
                       );


Password #(
   .nkeys (nkeys)
   ) a3 (


   .key (key),
            .enable1 (enable1),
            .reset (reset),
            .clk (clk),
            .ulseq (ulseq),
            .Led (led0),
            .Hex0 (h00),
            .Hex1 (h01),
            .Hex2 (h02)
            );



fsm #(
        .nkeys (nkeys)
            ) f1 (
            .key (key),
            .enable0 (enable0),
            .reset (reset),
            .t1in (t1in),
            .t2in (t2in),
            .clk (clk),
            .ulseq (ulseq),
            .t1out (t1out),
            .t2out (t2out),
            .enableclk1 (enableclk1),
            .enableclk2 (enableclk2),
            .Z(Z),
            .Led(led1),
            .hex0 (h10),
            .hex1 (h11),
            .hex2 (h12),
            .hex3 (h13),
            .hex4 (h14),
            .hex5 (h15)
            );




endmodule
```

2) timer1.v

```
module timer1 #(
     parameter timer1 = 7    // delay
                    )(
                    input t1out,    // it is basically used as reset the counter to 0
                    input enable,   // enable signal of the clock
                    input reset,    // main reset signal
                    input clkdiv,   // clk coming from frequency divider
                    output reg t1in   // output goes high if timer reaches to a value of timer1

                    );

reg [3:0] counter = 0;



always @(clkdiv or reset or t1out) begin


            if (reset == 1'b1 | t1out == 1'b1) begin


                        counter <= 0;
                        t1in <= 0;



                    end
```

```
                if (enable == 1'b1 && t1out == 1'b0 && counter < timer1) begin

            counter <= counter + 1;  // increment the counter value till it reaches to timer 1
                        t1in <= 0;
                end

          if (enable == 1'b1 && t1out == 1'b0 && counter == timer1) begin

            t1in <= 1;  // set high output when timer expires

                end



end

endmodule
```

3) Password.v

```
module Password #(
  parameter nkeys = 4 // we can define any length Password sequence
  ) (

  input [3:0] key, // input push button
        input enable1,   // this signal would enable the module
        input reset,     // reset the process
        input clk,       // internal clock input
        output reg [(nkeys*2)-1:0] ulseq,  // to output the stored sequence fsm module
        output reg [3:0] Led,      // to display the button pressed
        output reg [6:0] Hex0, Hex1, Hex2 // to display the output string
        );
```

```
reg [7:0] counter;  // used to count upto nkeys

reg [3:0] keyprev;  // to check the previous state of the key

reg [10:0] width;   // used for storing bits in pb reg

reg [(nkeys*2)-1:0] pb; // it needs to store 2 bits data into the register for every press in button

reg [6:0] hex0,hex1,hex2;  // to display the output string

reg [3:0] led;

parameter key0 = 4'b1110, key1 = 4'b1101, key2 = 4'b1011, key3 = 4'b0111;

parameter initialize = 2'b00,idlestate = 2'b01, checkstate = 2'b10;

reg [1:0] state, next;

always @(posedge clk) begin
    if (reset == 1'b1) state <= initialize;
                if (enable1 == 1'b0) state <= idlestate;
                else state <= next;
end

always @(posedge clk) begin
    case(state)

                initialize : begin              // begins with initialize state

                                        next <= idlestate;      // transit to idlestate
```

```
                    end


                    idlestate : begin


                            if (keyprev != 4'b1111 && key [3:0] == 4'b1111) next <= idlestate;


                                                            if ((key [3:0] == key0 || key [3:0] == key1 || key [3:0]
== key2 || key [3:0] == key3) && keyprev == 4'b1111) next <= checkstate;
                                                            // transit for every push of the button
                    end


                    checkstate : begin    // respective state transition after execution of instruction
                            if (counter == nkeys) next <= idlestate;
                                                            if (counter < nkeys && key [3:0] == key0 && keyprev
== 4'b1111) next <= checkstate;
                            if (counter < nkeys && key [3:0] == key1 && keyprev == 4'b1111) next <=
checkstate;
                            if (counter < nkeys && key [3:0] == key2 && keyprev == 4'b1111) next <=
checkstate;
                            if (counter < nkeys && key [3:0] == key3 && keyprev == 4'b1111) next <=
checkstate;
                            if (key [3:0] == 4'b1111) next <= idlestate;
                    end


                    default: begin
                        state <= initialize; // default state
                    end
                    endcase
end


always @(*) begin
    if (reset == 1'b1) begin    // initializes all the regs when reset signal goes high
```

```
                            led <= 4'd0;

                            counter <= 0;

                            hex0 <= 0;

                            hex1 <= 15;

                            hex2 <= 15;

                            width <= 1;

                            pb <= 0;

                            keyprev <= 4'b1111;


        end


else begin


   case(state)


            initialize: begin           // at the begining of the state execute all the instructions


                            led <= 4'd0;
                                counter <= 0;

                                hex0 <= 0;

                                hex1 <= 15;

                                hex2 <= 15;

                                width <= 1;

                                                keyprev <= 4'b1111;

                                pb <= 0;

            end


            idlestate : begin


                                    if (keyprev != 4'b1111 ) begin     // to update the state of the push
button

                                                    keyprev <= 4'b1111;
```

```
                                                  end


                                                  if ((key [3:0] == key0 || key [3:0] == key1 || key [3:0]
== key2 || key [3:0] == key3) && keyprev == 4'b1111) begin
                                                   // for any key press ,change of state
                                                         led <= counter;
                                                         hex0 <= counter;


                                                  end


                  end


                  checkstate : begin


                 if (counter == nkeys) begin     // if counter = nkeys then display set on hex led and
output the whole sequence to Fsm module
                          led <= counter;
                                               hex0 <= 5;
                                               hex1 <= 6;
                                               hex2 <= 7;
                                                         ulseq[(nkeys*2)-1:0] <= pb [(nkeys*2)-1:0];
                       end


                      if (counter < nkeys && key [3:0] == key0 && keyprev == 4'b1111) begin


                           pb[width -:1] <= 2'b00;   // store 2'b00 when key zero is pressed
                           width <= width + 2;       // increment the bits, ex pb[1:0] = 2'b00, next
process pb[3:2] = 2'b01. In this way each key is assigned
                           counter <= counter + 1;   // to one set of bits,   // to increment counter
value to check if it has reached the maximum length of sequenc
                           led <= counter;
                           hex0 <= counter;
                           keyprev <= key0;
```

```
                    end
                // Similarly like above steps, i have defined for other keys

                    if (counter < nkeys && key [3:0] == key1 && keyprev == 4'b1111) begin

                            pb[width -:1] <= 2'b01; // store 2'b01 when key 1 is pressed
                            width <= width + 2;
                            counter <= counter + 1;
                                                    led <= counter;
                            hex0 <= counter;
                            keyprev <= key1;

                    end

                   if (counter < nkeys && key [3:0] == key2 && keyprev == 4'b1111) begin


                           pb[width -:1] <= 2'b10; // store 2'b10 when key 1 is pressed
                           width <= width + 2;
                           counter <= counter + 1;
                                                   led <= counter;
                           hex0 <= counter;
                           keyprev <= key2;

                    end

                   if (counter < nkeys && key [3:0] == key3 && keyprev == 4'b1111) begin

                           pb[width -:1] <= 2'b11; // store 2'b11 when key 1 is pressed
                           width <= width + 2;
                           counter <= counter + 1;
                                                   led <= counter;
```

```
                                    hex0 <= counter;

                                    keyprev <= key3;

                         end


                         if (key [3:0] == 4'b1111) begin

                                                  led <= counter;

                                                                 hex0 <= counter;

                                               end


                            end


                            default: begin

                        state <= initialize;

                            end


                    endcase


end


Hex0 <= hex0;

Hex1 <= hex1;

Hex2 <= hex2;

Led  = led;

end


endmodule
```

4) Outputhex.v

```
module outputhex (

    input [6:0] hex00,hex01,hex02,

                    input [6:0] hex10,hex11,hex12,hex13,hex14,hex15,
                        input [3:0] led0,led1,
                        input password,
                        input clk,
                        output reg [3:0] Led,
                        output reg [6:0] Hex0,Hex1,Hex2,Hex3,Hex4,Hex5
                        );

localparam empty = 15;

function [6:0] display;   // display function is used to convert the decimal input to required Bcd value

input [3:0] disp;

case (disp)

            4'd0: display = 7'b1000000; // 0
   4'd1: display = 7'b1111001; // 1
            4'd2: display = 7'b0100100; // 2
            4'd3: display = 7'b0110000; // 3
            4'd4: display = 7'b0011001; // 4
            4'd5: display = 7'b0000111; // T
            4'd6: display = 7'b0000111; // E
            4'd7: display = 7'b0010010; // S
            4'd8: display = 7'b0001001; // K
            4'd9: display = 7'b0100111; // C
            4'd10:display = 7'b0100011; // O
            4'd11:display = 7'b1000111; // L
            4'd12:display = 7'b0101011; // N
            4'd13:display = 7'b1000001; // U
            4'd14:display = 7'b0101111; // R
            4'd15:display = 7'b1111111; // Empty
```

```
    default: display = 7'bx;


endcase


endfunction




always @(posedge clk) begin




                        if (password == 1'b0) begin   // if password switch is low connect outputlines to fsm
module


                           Hex0 <= display(hex10);

                                     Hex1 <= display(hex11);

                                     Hex2 <= display(hex12);

                                     Hex3 <= display(hex13);

                                     Hex4 <= display(hex14);

                                     Hex5 <= display(hex15);

                                     Led  <= led1;

                        end


                        if (password == 1'b1) begin   // if password switch is high connect outputlines to
password module


                                     Hex0 <= display(hex00);

                                     Hex1 <= display(hex01);

                                     Hex2 <= display(hex02);

                                     Hex3 <= display(empty);

                                     Hex4 <= display(empty);

                                     Hex5 <= display(empty);

                                     Led  <= led0;

                        end
```

```
end


endmodule
```

5) fsm.v

```
module  fsm #(
    parameter nkeys = 4   // we can define any length Password sequence
              )(
               input [3:0] key, // input push button
               input  reset,   // reset the process
                         input t2in,    // input from timer block, specifically for error block, to display error for
5 secs
                         input t1in,    // input from timer block, specifically for wait block
                         input enable0,  // enable input of this module
                         input clk,      // clock
                         input  [(nkeys*2)-1:0]  ulseq,   //  input  password  sequence  coming  from  password
module
                         output reg t2out,         // to reset the timer clock
                         output reg t1out,         // to reset the timer clock
                         output reg enableclk1,    // enable input for timer
                         output reg enableclk2,    // enable input for timer
                         output reg Z,       //output goes high when sequence is right
                         output reg [3:0] Led,   // to display the button pressed
                         output reg [6:0] hex0,hex1,hex2,hex3,hex4,hex5 // to display the output string
                         );

localparam initialize = 3'b000, start = 3'b001, button = 3'b010, error = 3'b011, wait1 = 3'b100;


localparam key0 = 4'b1110, key1 = 4'b1101, key2 = 4'b1011, key3 = 4'b0111;


reg [3:0] state, next;
reg [nkeys:0] counter; // used to count upto nkeys

reg [3:0] keyprev; // to check the previous state of the key

reg [(nkeys*2)-1:0] seq; // it needs to store 2 bits data into the register for every press in button

reg unl; // used to change the display, like UNlock or lock, everytime when user type correct sequence

reg z;

reg [3:0] led;

reg [(nkeys*2)-1:0] width; // used for storing bits in seq reg

reg [3:0] h0,h1,h2,h3,h4,h5; // to display the output string

always @(posedge clk) begin   // Next State logic

    if (reset == 1'b1 | enable0 == 1'b0) state <= initialize;   // goes to initialize state

            else state <= next;   // update the states
end

always @(*) begin    // designed the whole logic as combinational block,  State Memory

     case(state)

                            initialize: begin

                                next = start;

                            end
```

```
                        start: begin


                                       if ( (key[3:0] == key0 || key[3:0] == key1 || key[3:0] == key2 ||
key[3:0] == key3) && keyprev == 4'b1111) next = button;
                                                                    else   next = start;
                end


                                      button: begin


                                            if(key [3:0] == key0 && keyprev == 4'b1111 && counter < nkeys)
next = button;
                    if(key [3:0] == key1 && keyprev == 4'b1111 && counter < nkeys)     next = button;
                                                           if(key [3:0] == key2 && keyprev == 4'b1111 &&
counter < nkeys)     next = button;
                                                           if(key [3:0] == key3 && keyprev == 4'b1111 &&
counter < nkeys)     next = button;
                                                           if (counter == nkeys && ulseq == seq && key [3:0]
== 4'b1111)     next = start;
                                                           if ((counter == nkeys) && (ulseq != seq) && key [3:0]
== 4'b1111)   next = error;
                    if (key [3:0] == 4'b1111 && counter < nkeys)              next = wait1;
                end


                                      wait1: begin


                                            if ((key[3:0] == key0 || key[3:0] == key1 || key[3:0] == key2 ||
key[3:0] == key3 ) && keyprev == 4'b1111) next = button;
                    if (t1in == 1'b1)                  next = error;
                                                           if (keyprev != 4'b1111 && key [3:0] == 4'b1111) next
= wait1;
                                                           else                      next = wait1;


                                      end


                                      error: begin
```

```
                                    if (t2in == 1'b1) next = start;
                                                else        next = error;


                                end




                                default : next = initialize;


                                endcase
end


always @(clk or reset or enable0) begin       // Output Logic , Sequential block using non-blocking
statements


if (reset == 1'b1 | enable0 == 1'b0) begin
  z <= 1'b0;
        width <= 1;
        counter <= 0;
        led <= 0;
        enableclk2 <= 1'b0;
  enableclk1 <= 1'b0;
        h0 <= 15;
        h1 <= 15;
        h2 <= 15;
        h3 <= 15;
        h4 <= 15;
        h5 <= 15;
        unl <= 0;
        seq <= 0;
end


else begin
```

```
    case(state)


                                    initialize: begin            // initialize all the variables
                                        z <= 1'b0;
                    width <= 1;
                    counter <= 0;
                    led <= 0;
                    enableclk2 <= 1'b0;
            enableclk1 <= 1'b0;
                    h0 <= 15;
                    h1 <= 15;
                    h2 <= 15;
                    h3 <= 15;
                    h4 <= 15;
                    h5 <= 15;
                    unl <= 0;
                    keyprev <= 4'b1111;
                                                  seq <= 0;
                      end


                      start: begin


                                      if ((key[3:0] == key0 || key[3:0] == key1 || key[3:0] == key2 || key[3:0] ==
key3) && keyprev == 4'b1111) begin
                                                         z <= 1'b0;
                                                         enableclk1 <= 1'b1;
                                                 end


                                                 if(unl == 1'b0) begin    // to display unlock


                                                         counter <= 0;
                                                         keyprev <= 4'b1111;
                                                         led <= 4'b1111;

                                                         width <= 1;

                                                         h0 <= 8;

                                                         h1 <= 9;

                                                         h2 <= 10;

                                                         h3 <= 11;

                                                         h4 <= 12;

                                                         h5 <= 13;

                                                 end


                                                 if(unl == 1'b1) begin  // to display lock


                                                         counter <= 0;

                                                         led <= 4'b1111;

                                                         keyprev <= 4'b1111;

                                                         width <= 1;

                                                         h0 <= 8;

                                                         h1 <= 9;

                                                         h2 <= 10;

                                                         h3 <= 11;

                                                         h4 <= 15;

                                                         h5 <= 15;

                                         end


                         end


                                 button: begin


                                                 if(key [3:0] == key0 && keyprev == 4'b1111 && counter < nkeys)
begin



                                                seq [width -:1] <= 2'b00;
                                                width <= width + 2;
                                                counter <= counter + 1;
                                                led <= 4'b0001;
                                                keyprev <= key0;

                                        end

                        if(key [3:0] == key1 && keyprev == 4'b1111 && counter < nkeys) begin

                                                keyprev <= key1;
                                                seq [width -:1] <= 2'b01;
                                                width <= width + 2;
                                                counter <= counter + 1;
                                                led <= 4'b0010;



                                        end

                                        if(key  [3:0]  ==  key2  &&  keyprev  ==  4'b1111  &&
counter < nkeys) begin

                                                keyprev <= key2;
                                                seq [width -:1] <= 2'b10;
                                                width <= width + 2;
                                                counter <= counter + 1;
                                                led <= 4'b0011;



                                        end

                                        if(key  [3:0]  ==  key3  &&  keyprev  ==  4'b1111  &&
counter < nkeys) begin
```

```
                                        keyprev <= key3;

                                        seq [width -:1] <= 2'b11;

                                        width <= width + 2;

                                        counter <= counter + 1;

                                        led <= 4'b0100;




                                end


                                if (counter == nkeys && ulseq == seq && key [3:0]
== 4'b1111) begin // to check if the sequence is right display either lock or unlock


                                        unl <= ~unl;

                                        enableclk2 <= 1'b0;

                                        enableclk1 <= 1'b0;

                                        z <= 1'b1;


                                end


                                if ((counter == nkeys) && (ulseq != seq) && key [3:0]
== 4'b1111) begin // to check if the sequence is wrong ,go to erro state


                                        enableclk2 <= 1'b1;


                                end


                                if (key [3:0] == 4'b1111 && counter < nkeys) begin

                                  z <= 1'b0;

                                end


        end
```

```
                              wait1: begin


                                             if ((key[3:0] == key0  || key[3:0] == key1  || key[3:0] == key2  ||
key[3:0] == key3 ) && keyprev == 4'b1111) begin
                                                         t1out <= 1'b1;  // reset the timer
                                                        end


                     if (t1in == 1'b1) begin   // if timer expiers then go to error state
                                                         enableclk1 <= 1'b0;
                                                                enableclk2 <= 1'b1;
                                                        end


                                                        if  (keyprev  !=  4'b1111  &&  key  [3:0]  ==  4'b1111)
begin
                                                          keyprev = 4'b1111;
                                                        end


                                                        else begin
                                                          t1out <= 1'b0;
                                                        end


                              end


                     error: begin


                                             if (t2in == 1'b1) begin   // wait till the timer expires till 5 sec, then
ges to start state


                                                         enableclk2 <= 1'b0;
                                                                enableclk1 <= 1'b0;


                                                        end


                                                        else begin            // display error
```

```
                                t2out <= 1'b0;
                                    z <= 1'b0;
                                    led <= 4'b1111;
                                    h0 <= 14;
                                h1 <= 10;
                                h2 <= 14;
                                h3 <= 14;
                                h4 <= 6;
                                h5 <= 15;
                                    keyprev <= 4'b1111;

                               end

                     end

                     default : next <= initialize;

endcase
end

hex0 <= h0;
hex1 <= h1;
hex2 <= h2;
hex3 <= h3;
hex4 <= h4;
hex5 <= h5;
Led <= led;
Z <= z;

end
```

```
endmodule
```

6) frequencydivider.v

```
module frequencydivider #(
    parameter required_clk_frequency = 1,
                    parameter clk_frequency = 50000000
                    )(
                     input clk,  // input clock
                            output reg clkdiv  // outptut the required frequency
                            );


localparam integer clkcycle = clk_frequency / required_clk_frequency; // calculate number of clock cycles required
reg[10:0] counter = 0;


always @(posedge clk) begin


                    if (counter < (clkcycle / 2)) begin // set high for first half of the calculated value
       counter <= counter + 1;
                      clkdiv <= 1;
                     end




                    if (counter >= (clkcycle / 2) && counter < clkcycle) begin    // set low for next half of the calculated value, to obtain required frequency
                                counter <= counter + 1;
                                               clkdiv <= 0;
                    end


                    else begin
                       counter <= 0;
                    end
```

```
end

endmodule
```

7) demux.v

```
module dmux (

    input passkey,
            output wire enable0,
            output wire enable1
            );

reg s0 = 1'b1;
assign enable0 = s0 & (~passkey);
assign enable1 = s0 & (passkey);

endmodule
```

8) fsm5state_tb.v

```
`timescale 1 ns / 100 ps

module fsm5state_tb;

localparam nkeys = 4;
localparam timer1 = 10;
localparam timer2 = 5;
localparam required_clk_frequency = 1;
localparam clk_frequency = 50000000;
reg [3:0] key;
reg passkey;
reg reset;

reg clk;

reg [3:0] control;

wire Z;

wire [3:0] Led;

wire [6:0] Hex0;

wire [6:0] Hex1;

wire [6:0] Hex2;

wire [6:0] Hex3;

wire [6:0] Hex4;

wire [6:0] Hex5;


fsm5state #(

  .nkeys (nkeys),

  .timer1 (timer1),

  .timer2 (timer2),

  .required_clk_frequency (required_clk_frequency),

  .clk_frequency (clk_frequency)

  ) a1 (


  .key (key),

  .passkey (passkey),

  .reset (reset),

  .clk (clk),

  .Z (Z),

  .Led (Led),

  .Hex0 (Hex0),

  .Hex1 (Hex1),

  .Hex2 (Hex2),

  .Hex3 (Hex3),

  .Hex4 (Hex4),

  .Hex5 (Hex5)
  );



localparam clockfrequency = 50_000_000;

localparam clockPeriod = (1_000_000_000.0 / clockfrequency);

localparam halfPeriod = clockPeriod / 2;


always #(halfPeriod) clk = ~clk;



initial begin


      clk = 0;

                          key = 4'b1111;

                          passkey = 0;

                          reset = 1;


   # 10;

                         reset = 0;

                         passkey = 1;

                         key = 4'b1111;

                         @(posedge clk);

     key = 4'b1111;


   # 30;

                         reset = 0;

                         passkey = 1;

                         key = 4'b1110;

                         @(posedge clk);

     key = 4'b1111;
   # 30;
                         reset = 0;
                         passkey = 1;
                         key = 4'b1101;
                         @(posedge clk);
     key = 4'b1111;



   # 30;
                         reset = 0;
                         passkey = 1;
                         key = 4'b1011;
                         @(posedge clk);
     key = 4'b1111;


            # 30;
                         reset = 0;
                         passkey = 1;
                         key = 4'b0111;
                         @(posedge clk);
     key = 4'b1111;





   # 30;
                         reset = 0;
                         passkey = 0;
                         key = 4'b1111;
                         @(posedge clk);
     key = 4'b1111;


            # 30;
```

```
                        reset = 0;

                        passkey = 0;

                        key = 4'b1110;

                        @(posedge clk);

  key = 4'b1111;


       # 30;

                        reset = 0;

                        passkey = 0;

                        key = 4'b1101;

                        @(posedge clk);

  key = 4'b1111;


       # 30;

                        reset = 0;

                        passkey = 0;

                        key = 4'b1011;

                        @(posedge clk);

  key = 4'b1111;


  # 30;

                        reset = 0;

                        passkey = 0;

                        key = 4'b0111;

                        @(posedge clk);

  key = 4'b1111;



 # 30;

                        reset = 0;

                        passkey = 0;

                        key = 4'b1111;
                        @(posedge clk);
     key = 4'b1111;


   $display("%b ns\tsimulation Finished",$time);


end
endmodule
```