EE2702 MINI PROJECT REPORT

ELEVATOR CONTROL SYSTEM

BY – NIKHIL KUMAR YADAV
111601013
CSE(2nd year)
November 2017

INDEX

1) INTRODUCTION 3

2) What is an Elevator Controller? 4

3) About The Project 5

4) Materials Used in the project 6

5) Detailed overview of project 7

6) Conclusion 12

7) APPENDIX 13

- CODE
- CONSTRAINTS
- SCHEMATIC

INTRODUCTION

The high growth of the semiconductor industry over the past two decades has put Very Large Scale Integration in demand all over the world. The basics of digital logic theory and techniques are easily understood by the design based on VLSI technology. These are the core fundamentals of the fast, high-speed complex digital circuits. As day to day the technology is gradually improving. So obviously the designs have to be made simpler for enjoying the benefits. To do that, an Elevator Controller is modeled. In the proposed design a VERILOG RTL code is developed to control the lift moment based on the request it will get.

For that a finite state machine is developed to know from which state to state the controller is changing based on the requests from the end user. Lift is also called as Elevator or car. The design is based on the synchronous input which should be operating with a fixed sort of frequency.

What is an Elevator Controller?

An elevator is a device designed as a convenience appliance that has evolved to become an unavoidable feature of modern day urban life. An elevator is defined as, “A machine that carries people or goods up and down to different levels in a building or mine”. While a standalone elevator Isa simple electro-mechanical device, an elevator system may consist of multiple standalone elevator units whose operations are controlled and coordinated by a master controller. Such controllers are designed to operate with maximum efficiency in terms of service as well as resource utilization. This project details the design of an elevator controller using VERILOG. The Elevators/Lifts are used in multi store buildings as a means of transport between various floors. Elevator is a device designed as a convenience appliance that has evolved to become an unavoidable features of modern day in urban life normally .The lifts is controlled by Microprocessor based systems, which are costlier. It is proposed to design a low cost and compact dedicated controller. The Elevator Controller is a device used to control a lift motion and to indicate the direction of motion, and the present floor level, etc. The device control the lift motion by means of accepting the floor level as input and generate control signals (for control the lift motion) as output.

About The Project

So basically I will implement a elevator control system in the following way -

- Parameters such as current floor number,next floor request, door opened or closed, moving or idle, direction of movement etc will display on a LCD display.
- My project will be applicable for any number of floors but due to input constraints of zybo board we will stick to 16(i.e from 0-15) floors,(if there is no input constraint then my program can be easily modified to any number of group).
- For every floor there will be 2 buttons. One inside the lift and second on the floor to request lift.
- I will implement these buttons using multiplexing that is the same buttons will serve the purpose with a additional select button.
- The lift will have emergency stop button in case emergency. This button will also trigger a alarm.

My Elevator Algorithm

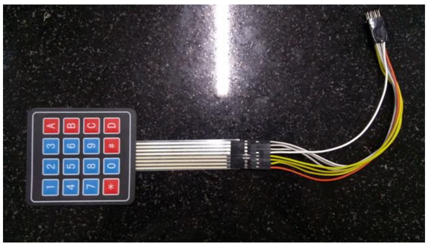
- Continue traveling in the same direction while there are remaining requests in that same direction.
- If there are no further requests in that direction, then stop and become idle, or change direction if there are requests in the opposite direction.

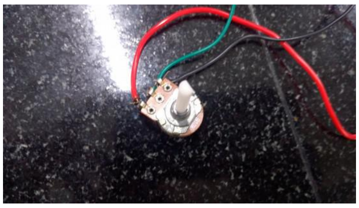
Assumptions made in project

- Doors open immediately when a lift reaches a floor. Doors close immediately before when lift moves.
- Lift takes 1 second(to display my project clearly, I can change later) to move between any two consecutive floors.

Materials Used -

Zybo board	(to program and implement our verilog code)
2*16 LCD Display	(to display current floor and the door status)
4*4 Keypad	(for input for request)
3 L293D Driver Ics	(to convert 3.3volts signal from zybo board to 5volts signal for lcd display)
10K Potentiometer	(to adjust the contrast of the display)
Power Supply	(to supply 5volts to the display and the Ics)
Breadboard	(for solderless connections)
Jumper Wires	(connecting wires)

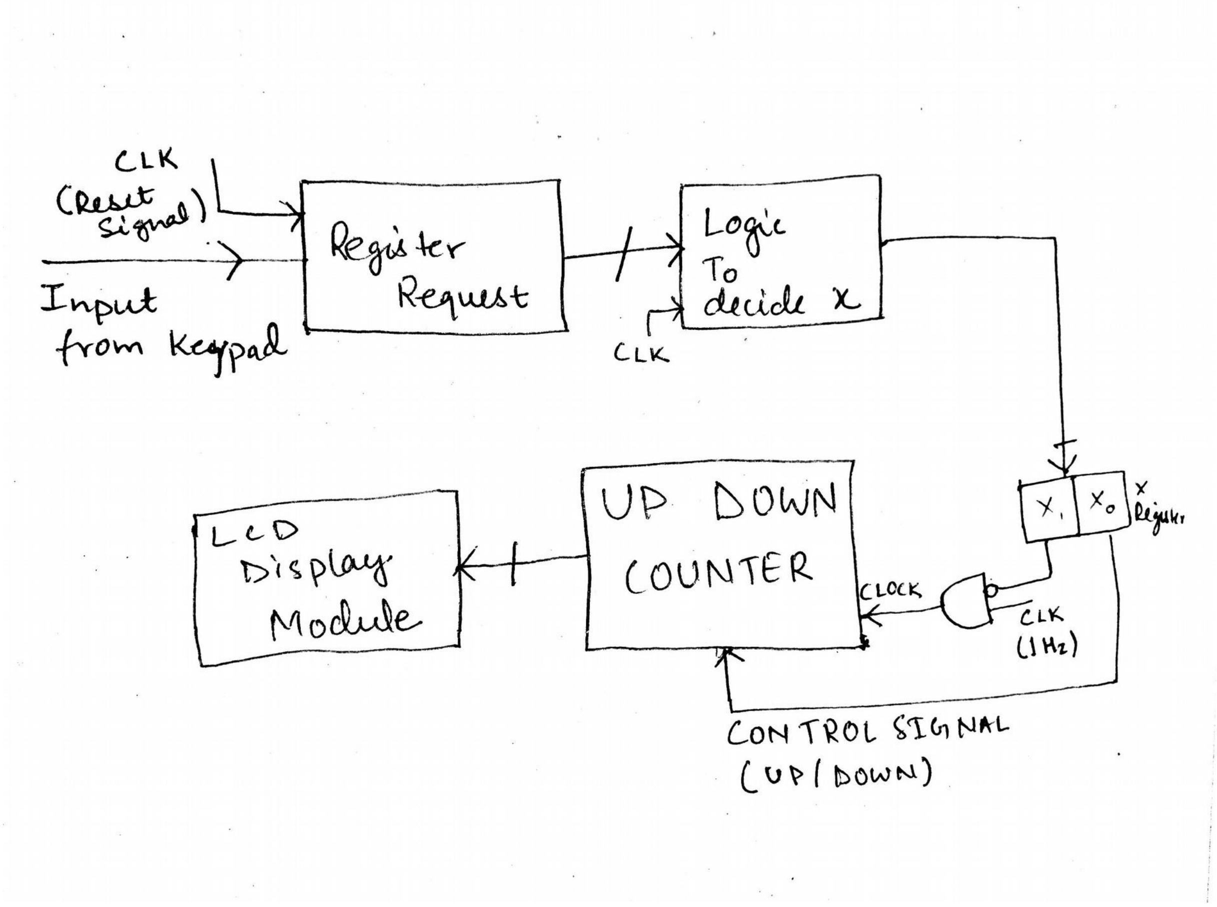




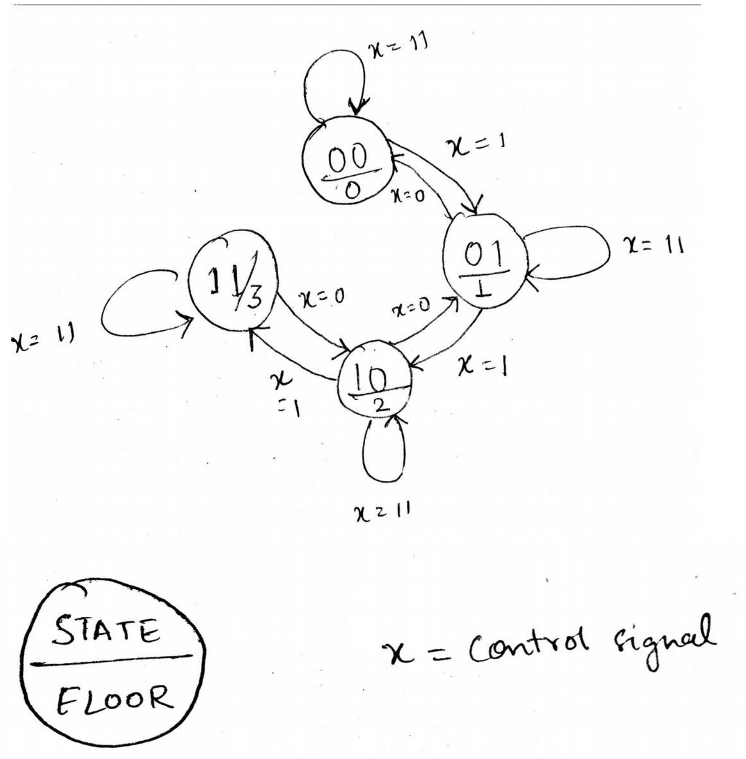
Detailed overview of project

- This project as a Mealy machine. As the current output (current floor) is a function of current state and input (present floor value and request of floor) .
- The LCD display here works as Moore machine as the display is dependent only on the current state.
- The Keypad input is taken when the set button is pressed.

Block Diagram (General)



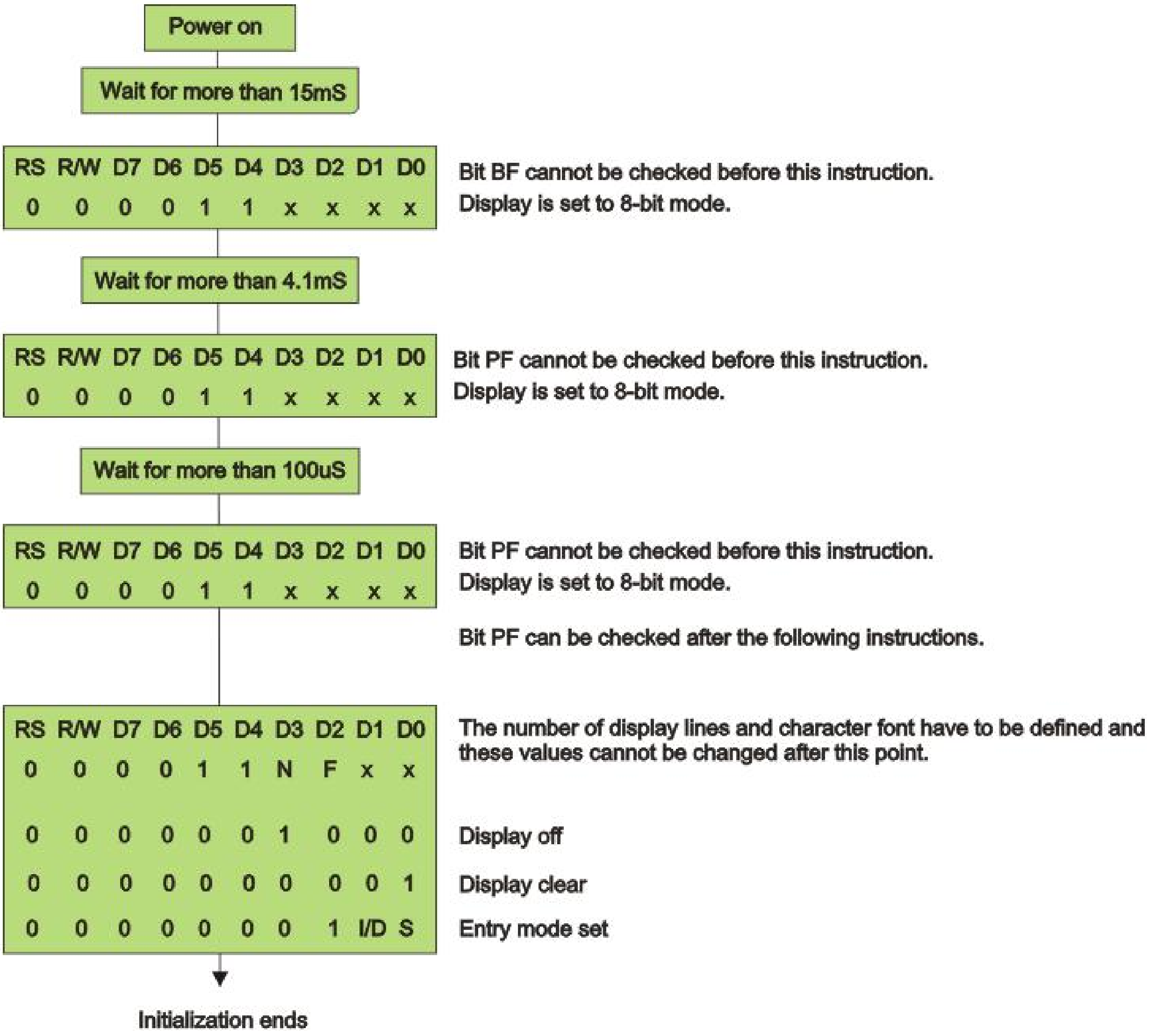
Simplified **State diagram** of the lift
(**only 4 floor** represented here) (**up/down** counter)



- x=1 mean lift travels upwards i.e more requests in the same direction.
- x=0 mean lift travels downwards i.e more requests in the same direction.
- x=11 mean lift remains stationary i.e no more pending requests.
- The control signal x is decided on the basis of current input ,current state and pending requests. This is achieved with help of if statements in verilog and with MUX in hardware.
- I have maintained a 16 bit register named request to store pending requests.
- Lcd Display module is a simple state machine. It refreshes itself every 20ms and displays the door opened or closed.

How does LCD DISPLAY work?

Lcd inialization -



Power on

Wait for more than 15mS

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	x	x	x	x

Bit BF cannot be checked before this instruction.
Display is set to 8-bit mode.

Wait for more than 4.1mS

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	x	x	x	x

Bit PF cannot be checked before this instruction.
Display is set to 8-bit mode.

Wait for more than 100uS

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	x	x	x	x

Bit PF cannot be checked before this instruction.
Display is set to 8-bit mode.

Bit PF can be checked after the following instructions.

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	1	1	N	F	x	x	The number of display lines and character font have to be defined and these values cannot be changed after this point.
0	0	0	0	0	0	1	0	0	0	Display off
0	0	0	0	0	0	0	0	0	1	Display clear
0	0	0	0	0	0	0	1	I/D	S	Entry mode set

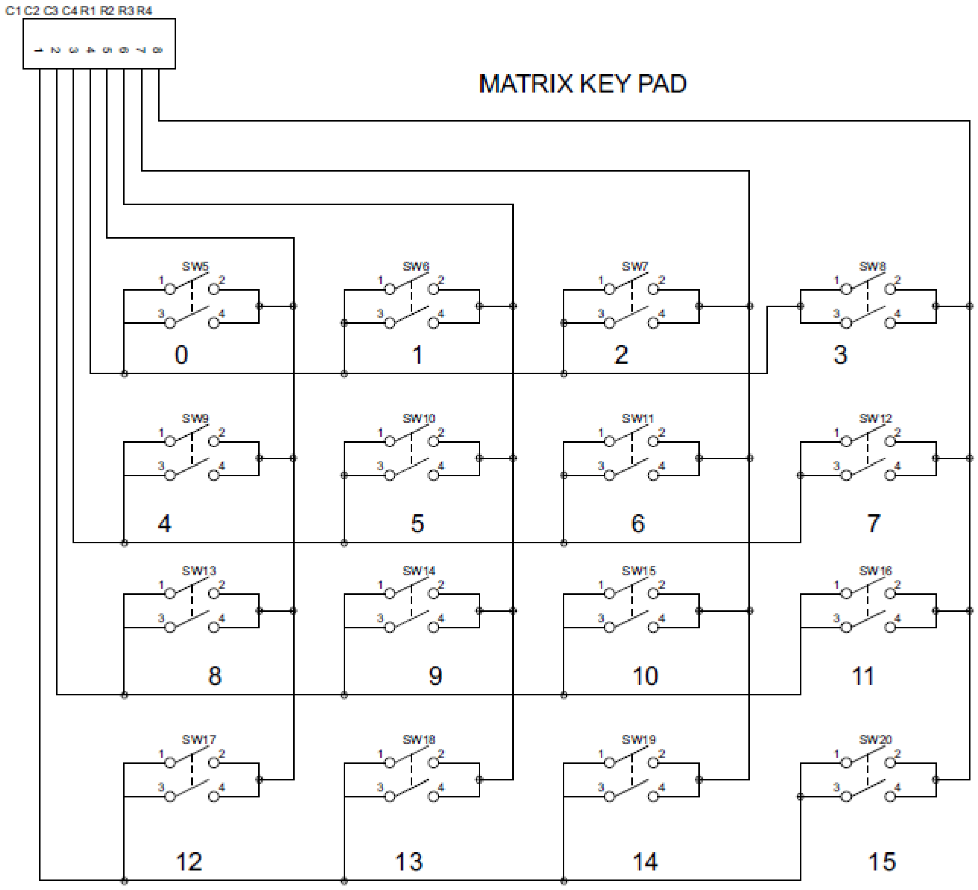
Initialization ends

After intialization we need to send data to the lcd. While sending data to print reigister select should be 1 and while writing the RW should be 0. When a data is send remember to send a pluse of enable signal during that time frame. It takes approximately 40 micro seconds to the LCD to complete one writing/reading task.

The whole thing can be implemented using state machine(Moore design). I have implemented in that way. I have used aroung 30~35(numbering is not serial because i wanted to debug it easily).

How does the matrix keypad works?

Internal circuit of the 4*4 matrix keypad

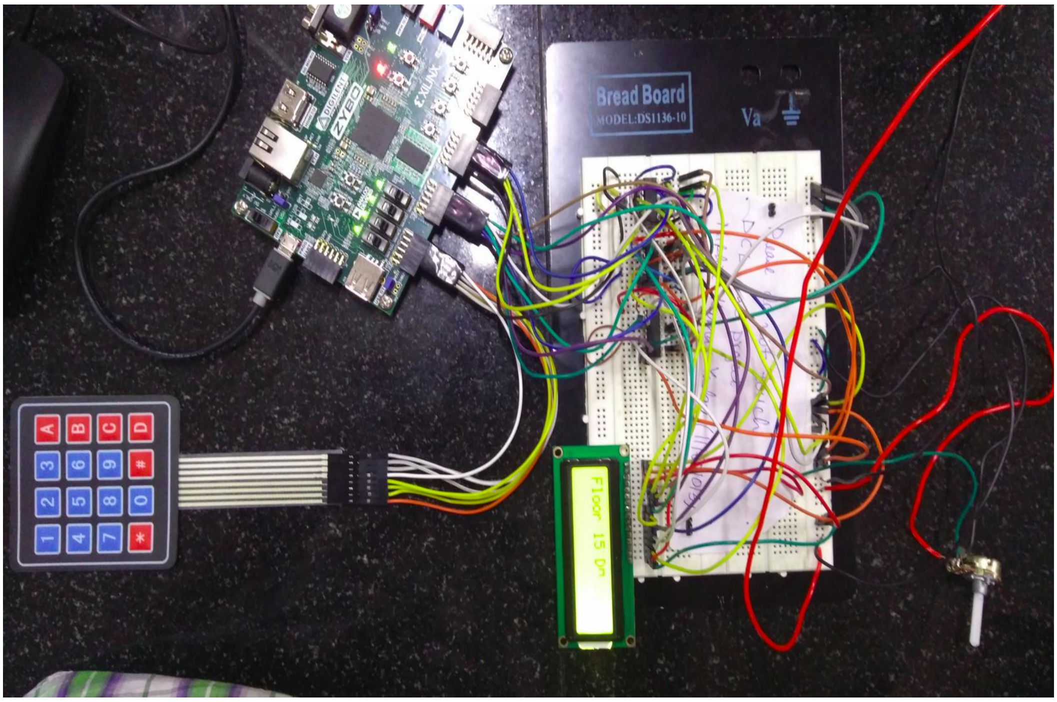


Step 1: The first step involved in interfacing the matrix keypad is to write all logic 0’s to the rows and all logic 1’s to the columns.

Step 2: Now the software has to scan the pins connected to columns of the keypad. If it detects a logic 0 in any one of the columns, then a key press was made in that column. This is because the event of the switch press shorts the column line with row line. Hence that column line is driven low.

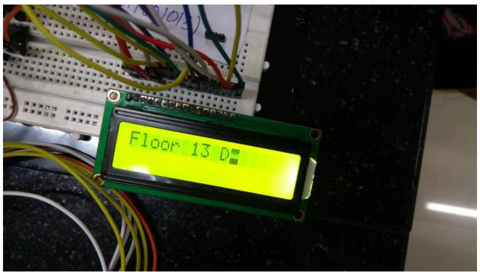
Step 3: Once the column corresponding to the key pressed is located, the next thing that the software has to do is to start writing logic 1’s to the rows sequentially (one after the other) and check if that column becomes becomes high. The logic is that if a button in that row was pressed, then the value written to that row will be reflected in the corresponding column as they are short circuited. Note: color of the lines indicate the logic values they return.

IMPLEMENTATION



Current floor 3 and door is open

Current floor 13 and door is closed and lift is moving



Conclusion

I was finally able to make a elevator control system for a building, and also was able to learn about how to operate Keypad and Lcd. I also thought of a different kind of different algorithms which can be used for a lift like I thought of giving priority to the floors and then operate the lift. Also my code general and can be extended to any number of floors with slight modifications(Current code is just for 16 floors).

APPENDIX

CODE , CONSTRAINTS and SCHEMATIC

```
`timescale 1ns / 1ps

module main(
    input clk,rset,
    input stop,
    input [3:0] row,
    output reg [3:0] col,
    output reg RS=0,
    output reg RW=0,
    output reg E=0,
    output reg [7:0] D=255,
    output reg [3:0] led,
    output reg alarm
    );

 /*//////////////////////////////////////////////////////
                      INPUTS -
          clk - zybo 125MHz onboard clock
          rset - set button to set the value of the keypad
          stop - Emergency stop the lift
          row  - to get row input for the keypad
                      OUTPUTS -
          col - 4 bit register to send signals to the keypad
          RS  - Register select for the LCD DISPLAY
          RW  - READ/WRITE for the LCD DISPLAY
          E   - Enable line of the LCD DISPLAY
          D   - 8 bit display lines
          led - Register to store the current value of the keypad
          alarm - if the lift is stopped
 //////////////////////////////////////////////////////*/

reg [3:0] floor=12;                //floor register maintains the current floor
reg [3:0] nextfloor=0;            //nextfloor register maintains the next floor to be visited
reg [31:0] count_1Hz=0;           //counter to count upto 125000000 for 1Hz clock from the 125MHz board clock
reg clk_1Hz=0;                    //1 Hz clock
reg [15:0] request=0;             //register to maintain remaining request of floors to visit
reg door=0;                       //register to indicate openning or closing of the door of the lift
reg [1:0]ud=0;            //register to maintain the direction in which the lift is travelling(or stationary)
reg flag=0;               //flag register to store the last state of register ud
reg stopedbefore=0;     //register to store if the lift was stoped before

reg clk_50Hz=0;                   //50Hz derived clock
reg [63:0] count_50Hz=0;          //counter to count upto that 50Hz clock

//Logic for the LCD DISPLAY
//Clock for the LCD DISPLAY
always@(posedge clk)              //Clock changed change to .8ms to refresh the lcd fast
begin
        if(count_50Hz==100000)
        begin
            count_50Hz<=0;
            clk_50Hz<=~clk_50Hz;
        end
        else
        begin
             count_50Hz<=count_50Hz+1;
        end
end

reg [15:0] STATE=3;               //Register to store the value of the state of the LCD
```

```
/*//////////////////////////////////////////////////////////////////
                        LOGIC TO DISPLAY
        To intialise to display -
                0x38 is used for 8-bit data initialization.
                0x1H for clearing the LCD.
        Sending Data to the LCD
                E=1; enable pin should be high
                RS=1; Register select should be high for writing the data
                Placing the data on the data registers          ascii value
                R/W=0; Read/Write pin should be low for writing the data.
//////////////////////////////////////////////////////////////////*/
always@(posedge clk_50Hz)
begin
        if(STATE==0)
        begin
            D<=8'b11110000;
            STATE<=1;
        end
        else if(STATE==1)
        begin
            STATE<=2;
            D<=8'b00000001;                  //clear screen
            RS<=0;
            E<=1;
        end
        else if(STATE==2)
        begin
          STATE<=3;
          E<=0;
        end
        else if(STATE==3)
        begin
            STATE<=4;
            D<=8'b00001100;                 //set screen  8bit communication and 16*2 cursor and blink off
            RS<=0;
            E<=1;
        end
        else if(STATE==4)
        begin
            STATE<=13;
            E<=0;
        end
        else if(STATE==13)
        begin
          D<=8'b01000110;                   //F
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=14;
        end
        else if(STATE==14)
        begin
          D<=8'b01000110;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=15;
        end
        else if(STATE==15)
        begin
          D<=8'b01101100;                   //l
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=16;
        end
        else if(STATE==16)
        begin
```

```
          D<=8'b01101100;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=17;
        end
        else if(STATE==17)
        begin
          D<=8'b01101111;                    //o
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=18;
        end
        else if(STATE==18)
        begin
          D<=8'b01101111;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=19;
        end
        else if(STATE==19)
        begin
          D<=8'b01101111;                    //o
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=20;
        end
        else if(STATE==20)
        begin
          D<=8'b01101111;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=21;
        end
        else if(STATE==21)
        begin
          D<=8'b01110010;                    //r
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=22;
        end
        else if(STATE==22)
        begin
          D<=8'b01110010;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=23;
        end
        else if(STATE==23)
        begin
          D<=8'b10100000;                    //space
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=24;
        end
        else if(STATE==24)
        begin
          D<=8'b10100000;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=30;
        end
        else if(STATE==30)                          //few STATE intentionally left
        begin
          D<=48 + (floor - (floor%10))/10 ;                   //floor most significant
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=31;
        end
        else if(STATE==31)
        begin
          E<=0;
          STATE<=32;
        end
        else if(STATE==32)
        begin
          D<=48 + (floor%10) ;                                //floor least significant
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=33;
        end
        else if(STATE==33)
        begin
          E<=0;
          STATE<=34;
        end
        else if(STATE==34)
        begin
          D<=8'b10100000;                   //space
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=35;
        end
        else if(STATE==35)
        begin
          D<=8'b10100000;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=36;
        end
        else if(STATE==36)
        begin
          D<=8'b01000100;                   //D
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=37;
        end
        else if(STATE==37)
        begin
          D<=8'b01000100;
          E<=0;
          RW<=0;
          RS<=1;
          STATE<=40;
        end
        else if(STATE==40)                          //few STATE intentionally left
        begin
          if(door==0)                               //door status
            D<=8'b11111111;
          else
            D<=8'b11011011;
          E<=1;
          RW<=0;
          RS<=1;
          STATE<=41;
        end
        else if(STATE==41)
        begin
          E<=0;
          STATE<=42;
        end
        else if(STATE==42)
            STATE<=43;
        else if(STATE==43)
            STATE<=44;
        else if(STATE==44)
            STATE<=1;
        else
          STATE<=41;
end

//Logic for the Keypad
reg [3:0] key_value;                //4 v=bit reg to store the current input
reg [16:0] count_500khz;            //counter to count upto that 500KHz clock
reg [2:0] state;                    //state variable register
reg key_flag;                       // flag variable
reg clk_500khz;                     //derived clock 500KHz
reg [3:0] col_reg;                  //4 bit column reg
reg [3:0] row_reg;                  //4 bit row reg

always @(posedge clk)               //20 ms clock
begin
    if(rset)
    begin
        clk_500khz<=0;
        count_500khz<=0;
    end
    else
    begin
        if(count_500khz>=250)
        begin
            clk_500khz<=~clk_500khz;
            count_500khz<=0;
        end
        else
            count_500khz<=count_500khz+1;
     end
end

always @(posedge clk_500khz)        //Logic to determine the input form the keypad
begin
    if(rset)
    begin
        col<=4'b0000;
        state<=0;
    end
    else
    begin
       case (state)
        0:  begin
              col[3:0]<=4'b0000;
              key_flag<=1'b0;
              if(row[3:0]!=4'b1111)
              begin
                state<=1;
                col[3:0]<=4'b1110;
              end
            else
                state<=0;
```

```verilog
            end
        1:  begin
               if(row[3:0]!=4'b1111)
               begin
                state<=5;
               end
               else
               begin
                    state<=2;
                    col[3:0]<=4'b1101;
               end
            end
        2:  begin
              if(row[3:0]!=4'b1111)
              begin
                state<=5;
              end
              else
              begin
                state<=3;
                col[3:0]<=4'b1011;
              end
             end
      3:     begin
              if(row[3:0]!=4'b1111)
              begin
                state<=5;
              end
              else
              begin
                state<=4;
                col[3:0]<=4'b0111;
              end
             end
      4:     begin
               if(row[3:0]!=4'b1111)
               begin
                    state<=5;
               end
               else
                    state<=0;
             end
      5:     begin
               if(row[3:0]!=4'b1111)
               begin
                     col_reg<=col;
                     row_reg<=row;
                     state<=5;
                     key_flag<=1'b1;
               end
               else
               begin
                    state<=0;
               end
             end
    endcase
  end
end


 always @(clk_500khz or col_reg or row_reg)       //Decoding the input from the current input

     begin

        if(key_flag==1'b1)

                begin

```

```
                     case ({col_reg,row_reg})

                      8'b1110_1110:key_value<=0;

                      8'b1110_1101:key_value<=1;

                      8'b1110_1011:key_value<=2;

                      8'b1110_0111:key_value<=3;



                      8'b1101_1110:key_value<=4;

                      8'b1101_1101:key_value<=5;

                      8'b1101_1011:key_value<=6;

                      8'b1101_0111:key_value<=7;



                      8'b1011_1110:key_value<=8;

                      8'b1011_1101:key_value<=9;

                      8'b1011_1011:key_value<=10;

                      8'b1011_0111:key_value<=11;



                      8'b0111_1110:key_value<=12;

                      8'b0111_1101:key_value<=13;

                      8'b0111_1011:key_value<=14;

                      8'b0111_0111:key_value<=15;

                     endcase

              end

   end


//To assign the current key value to the register led at the posivite edge
always@(posedge rset)
begin
   led<=key_value;
end

//Elevator logic
always@(posedge(clk))
begin
        if(count_1Hz==125000000)                    //1Hz clock
        begin
            count_1Hz<=0;
            clk_1Hz<=~clk_1Hz;
        end
        else
            count_1Hz<=count_1Hz+1;
end

//To determine the next floor based upon input , ud , flag , pending request.
/*//////////////////////////////////////////////////////////////////////////////////
```

```
                        LOGIC DETERMINE THE NEXT FLOOR
        ->      The lift Continues traveling in the same
                direction while there are remaining
                requests in that same direction.
        ->      If there are no further requests in that
                direction, then stop and become idle,
                or change direction if there are
                requests in the opposite direction.
/////////////////////////////////////////////////////////////////////////////////*/
always@(posedge clk)
begin
        if(request[led]==0 && rset==1)                  //if we recive a new request
            request[led]<=1;
        if(door==1)                                     //if door is opened it implies that the current floor is visited
            request[floor]<=0;
        if(ud==0 && led<nextfloor  && floor<led)        //if lift is going up and the entered floor is b/w nextfloor and current floor
            nextfloor<=led;
        else if(ud==1 && led>nextfloor && floor>led)    //if lift is coming down and the entered floor is b/w nextfloor and current floor
            nextfloor<=led;
        if(ud==2'b11 && flag==1)                        //if the lift is now stationary and was travelling downwards
        begin
              if(request[0]==1 && floor==0)
                nextfloor<=0;
              else if(request[1]==1 && floor>1)
                nextfloor<=1;
              else if(request[2]==1 && floor>2)
                nextfloor<=2;
              else if(request[3]==1 && floor>3)
                nextfloor<=3;
              else if(request[4]==1 && floor>4)
                nextfloor<=4;
              else if(request[5]==1 && floor>5)
                nextfloor<=5;
              else if(request[6]==1 && floor>6)
                nextfloor<=6;
              else if(request[7]==1 && floor>7)
                nextfloor<=7;
              else if(request[8]==1 && floor>8)
                nextfloor<=8;
              else if(request[9]==1 && floor>9)
                nextfloor<=9;
              else if(request[10]==1 && floor>10)
                nextfloor<=10;
              else if(request[11]==1 && floor>11)
                nextfloor<=11;
              else if(request[12]==1 && floor>12)
                nextfloor<=12;
              else if(request[13]==1 && floor>13)
                nextfloor<=13;
              else if(request[14]==1 && floor>14)
                nextfloor<=14;
              else if(request[15]==1 && floor>15)
                nextfloor<=15;
              else if(request[0]==1 && floor==0)
                nextfloor<=0;
              else if(request[1]==1 && floor<1)
                nextfloor<=1;
              else if(request[2]==1 && floor<2)
                nextfloor<=2;
              else if(request[3]==1 && floor<3)
                nextfloor<=3;
              else if(request[4]==1 && floor<4)
                nextfloor<=4;
              else if(request[5]==1 && floor<5)
                nextfloor<=5;
              else if(request[6]==1 && floor<6)
                nextfloor<=6;
              else if(request[7]==1 && floor<7)
                nextfloor<=7;
              else if(request[8]==1 && floor<8)
                nextfloor<=8;
              else if(request[9]==1 && floor<9)
                nextfloor<=9;
              else if(request[10]==1 && floor<10)
                nextfloor<=10;
              else if(request[11]==1 && floor<11)
                nextfloor<=11;
              else if(request[12]==1 && floor<12)
                nextfloor<=12;
              else if(request[13]==1 && floor<13)
                nextfloor<=13;
              else if(request[14]==1 && floor<14)
                nextfloor<=14;
              else if(request[15]==1 && floor<15)
                nextfloor<=15;
        end
        else if(ud==2'b11 && flag==0)                   //if the lift is now stationary and was travelling upwards
        begin
              if(request[0]==1 && floor==0)
               nextfloor<=0;
              else if(request[1]==1 && floor<1)
               nextfloor<=1;
              else if(request[2]==1 && floor<2)
               nextfloor<=2;
              else if(request[3]==1 && floor<3)
               nextfloor<=3;
              else if(request[4]==1 && floor<4)
               nextfloor<=4;
              else if(request[5]==1 && floor<5)
               nextfloor<=5;
              else if(request[6]==1 && floor<6)
               nextfloor<=6;
              else if(request[7]==1 && floor<7)
               nextfloor<=7;
              else if(request[8]==1 && floor<8)
               nextfloor<=8;
              else if(request[9]==1 && floor<9)
               nextfloor<=9;
              else if(request[10]==1 && floor<10)
               nextfloor<=10;
              else if(request[11]==1 && floor<11)
               nextfloor<=11;
              else if(request[12]==1 && floor<12)
               nextfloor<=12;
              else if(request[13]==1 && floor<13)
               nextfloor<=13;
              else if(request[14]==1 && floor<14)
               nextfloor<=14;
              else if(request[15]==1 && floor<15)
               nextfloor<=15;
              else if(request[0]==1 && floor==0)
               nextfloor<=0;
              else if(request[1]==1 && floor>1)
               nextfloor<=1;
              else if(request[2]==1 && floor>2)
               nextfloor<=2;
              else if(request[3]==1 && floor>3)
               nextfloor<=3;
              else if(request[4]==1 && floor>4)
               nextfloor<=4;
              else if(request[5]==1 && floor>5)
```

```
               nextfloor<=5;
              else if(request[6]==1 && floor>6)
               nextfloor<=6;
              else if(request[7]==1 && floor>7)
               nextfloor<=7;
              else if(request[8]==1 && floor>8)
               nextfloor<=8;
              else if(request[9]==1 && floor>9)
               nextfloor<=9;
              else if(request[10]==1 && floor>10)
               nextfloor<=10;
              else if(request[11]==1 && floor>11)
               nextfloor<=11;
              else if(request[12]==1 && floor>12)
               nextfloor<=12;
              else if(request[13]==1 && floor>13)
               nextfloor<=13;
              else if(request[14]==1 && floor>14)
               nextfloor<=14;
              else if(request[15]==1 && floor>15)
               nextfloor<=15;
        end
end

//Lift movement logic
always@(posedge (clk_1Hz))
begin
        if(nextfloor==floor)
        begin
            door<=1;                            //door opened
            ud<=2'b11;                          //ud = 11 means lift stationary
        end
        else
        begin
            door<=0;                            //door closed
        end
        if(nextfloor>floor && ud!=10)
        begin
            ud<=0;                              //ud = 0 means lift travelling upwards
            flag<=0;
            floor<=floor+1;                     //incrementing floor
        end
        else if(nextfloor<floor && ud!=10)
        begin
            ud<=1;                              //ud = 1 means lift travelling downwards
            flag<=1;
            floor<=floor-1;                     //decrementing floor
        end
        if(stop==1)                             //Stop the elevator
        begin
                ud<=10;                         //undetermined state
                stopedbefore<=1;
                alarm<=1;                       //alarm on
        end
        if(stopbefore==1 && stop==0)
        begin
                ud<=11;                         //Stationory
                stopbefore<=0;
                alarm<=0;                       //alarm off
        end

end
endmodule
```

```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk

##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { stop }]; #IO_L19N_T3_VREF_35 Sch=SW0

##Buttons
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { rset }]; #IO_L20N_T3_34 Sch=BTN0

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rset]

##Pmod Header JC
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { RS }]; #IO_L10P_T1_34 Sch=JC1_P
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports { RW }]; #IO_L10N_T1_34 Sch=JC1_N
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { E }]; #IO_L1P_T0_34 Sch=JC2_P
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { alarm }]; #IO_L1N_T0_34 Sch=JC2_N

##Pmod Header JD
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { D[0] }]; #IO_L5P_T0_34 Sch=JD1_P
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { D[1] }]; #IO_L5N_T0_34 Sch=JD1_N
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { D[2] }]; #IO_L6P_T0_34 Sch=JD2_P
set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { D[3] }]; #IO_L6N_T0_VREF_34 Sch=JD2_N
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { D[4] }]; #IO_L11P_T1_SRCC_34 Sch=JD3_P
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports { D[5] }]; #IO_L11N_T1_SRCC_34 Sch=JD3_N
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { D[6] }]; #IO_L21P_T3_DQS_34 Sch=JD4_P
set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33 } [get_ports { D[7] }]; #IO_L21N_T3_DQS_34 Sch=JD4_N
##Pmod Header JE
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { row[0] }]; #IO_L4P_T0_34 Sch=JE1
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports { row[1] }]; #IO_L18N_T2_34 Sch=JE2
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { row[2] }]; #IO_25_35 Sch=JE3
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { row[3] }]; #IO_L19P_T3_35 Sch=JE4
set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports { col[0] }]; #IO_L3N_T0_DQS_34 Sch=JE7
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { col[1] }]; #IO_L9N_T1_DQS_34 Sch=JE8
set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports { col[2] }]; #IO_L20P_T3_34 Sch=JE9
set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { col[3] }]; #IO_L7N_T1_34 Sch=JE10

##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L23P_T3_35 Sch=LED0
set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3
```