Lab Report

VLSI System Design

ELE301P

EXPERIMENT 9

By Kadambi Narasimhan Akash
COE19B005

08 Nov 2021

INDEX

Verilog Programs

Lift Control System

Objective:

→ Design a lift group system for 4 lift which serves a 10-story building.
→ Controller should be designed with an aim to minimize the average waiting time of the Passengers.

Theory:

Concept:

We are given to design a Control System for given 4 lifts that co-ordinate.
Firstly, let us understand what are the inputs and outputs.
Inputs: A 12-bit input that uses First two MSBs for accessing 1 lift among 4, and other 10 bits denote 10 floors.
Outputs: Changes in states of the lift.

To Simplify, first let us understand what to do in case of a single lift. Once any request comes it holds in its buffer. Till all requests are satisfied it keeps on running. Our job is to set what direction the lift must start moving. If we optimize this using any greedy approach then there will be possibility of huge starvation. So, the average waiting time increases. So, the best thing we do in case of single lift is give preference to its direction. This follows the following algorithm:

→ If currently at rest, then go to the nearest requesting floor.
→ If moving, then service all the requests in its direction first before changing direction.

This is highly suitable for a single lift system. But here we have 4-lifts that we can co-ordinate and reduce the average waiting time further. Here also we give preference to the direction and do extra tasks to minimize it. The algorithm is as follows:

→ If currently at rest, then initially prefer the direction in which nearest request appears.

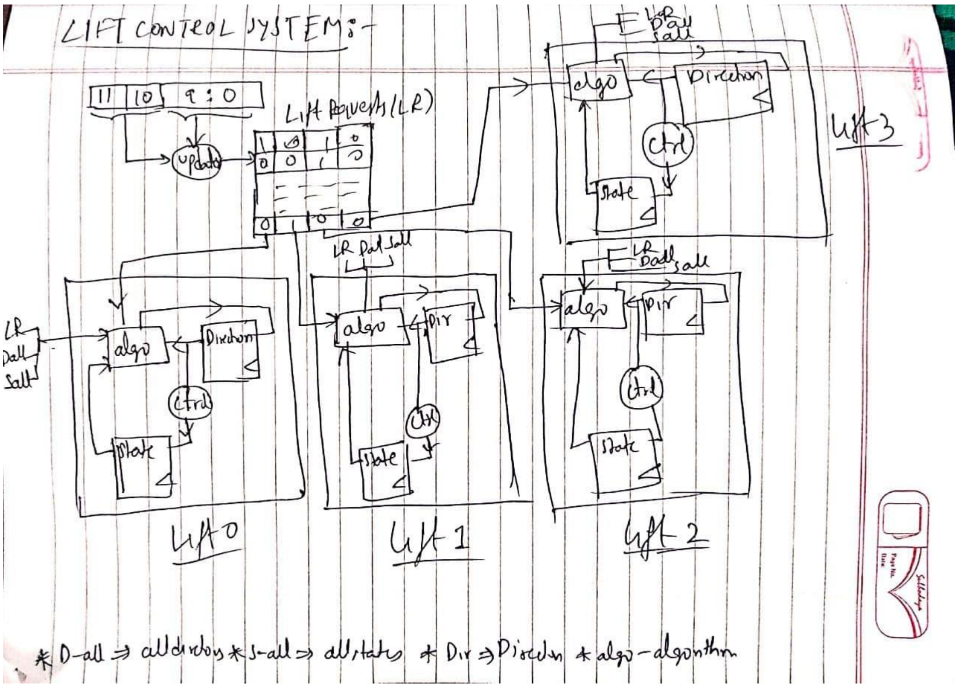
→ If moving, then initially prefer the direction it is moving.

→ Now to decide the final direction, for each other lift, the current lift sees if the nearest requested floor will be serviced first by the lift earlier than this lift or not. If yes then changes its direction to service the other requests which can reduce the average waiting time or else it remains in same direction. This can be divided into two types.

→ **Type 1:** Let the nearest floor requested in the initial direction of the lift is '**i**'. Now consider the case that in another lift **L**, it is moving in the same direction and the floor '**i**' is also requested for the lift **L**. Now the current lift **Lc**, sees if already the **L** has crossed the floor or not. If yes, then it stops and checks for other lifts. If not, then **Lc** computes if **L** can reach '**i**' earlier than **Lc**. If yes, change **Lc** direction or else do this for other lifts.

→ **Type 2:** Let the nearest floor requested in the initial direction of the lift is '**i**'. Now consider the case that in another lift **L**, it is moving in the opposite direction and the floor '**i**' is also requested for the lift **L**. Now the current lift **Lc**, sees if already the **L** has crossed the floor or not. If yes, then it stops and checks for other lifts. If not, then **Lc** checks if any requests are there for it beyond the state of **L**. If yes, then it stops and checks for others. If not, then it computes whether **L** can reach '**i**' earlier than **Lc**. If yes, change **Lc** direction or else do this for other lifts.

Diagram:



Code:

1. Main Codes:

```
module LFT(Inp);
    input[11:0] Inp;
    reg[3:0][9:0] L;
//Holds Lift Requests For all 4 Lifts (0: Not-Requested 1: Requested)
    reg[3:0][1:0] D;
//Holds the Current Direction of Lifts(0: At-Rest 1: Going-Up 2: Going-Down)
    reg[3:0][3:0] S;
//Holds the Current States of all 4 lifts(0 to 9 to denote 10 floors)
    reg[3:0][3:0] R;
//Holds the State towads which the Lift is Going(0 to 9 to denote 10 floors)
```

```
    //Following variables used for calculation purposes as we need to
minimize average waiting time

    reg[3:0][3:0] H;//Holds Hop Count of Lifts

    reg[3:0][9:0] KL;//Temporary registers to store Lift requests of
particular Lift

    reg[9:0] TL;//Temporary registers to store Lift requests of
particular Lift

    reg[3:0] Oflag;//Flag Holds if the Lift is Open or Not

    reg[3:0] DFlag;

    //Flag to indicate change the direction so wait time reduced

    integer i,j,r;//Iterators

    reg[3:0] Flag;//Intially when at rest, used to decide in which direction
the request is near.(0-Bottom 1-Top)


    initial

    begin

        L=0;D=0;S=0;R=0;H=0;

    end


//Initializing Requests

    always@(Inp)

    begin

        L[Inp[11:10]]=L[Inp[11:10]] | Inp[9:0];//ORed because when already
request is ON, till its request completed it won't turn it off.

    end
```

```
//Displays Changes In Lift States
    always@(L,S,D,Oflag)
    begin
    $display("-------------------LIFTS---------------------");
        for(integer n=9;n!=-1;--n)
        begin
            for(integer m=0;m<4;++m)
            begin
                TL=L[m];
                if(n==S[m])
                    if(Oflag[m])
                        $write("[]");
                    else if(D[m]==1)
                        $write("^ ");
                    else if(D[m]==2)
                        $write("v ");
                    else
                        $write("I ");
                else if(TL[n]==1)
                    $write("+ ");
                else
                    $write("- ");
            end
            $write("\n");
        end
    $display("-------------------LIFTS---------------------");
    end
```

//Now for each lift an always block is used

Lift-i always block

endmodule

The always block code is as follows for a lift L0, all others have similar configuration:

```
always@(L[0])
    begin
        R[0]=S[0];//initially made same to make it decide the direction.
        D[0]=0;//As initially Lift is at rest


        //Finding Which Direction is Near
        for(i=S[0]+1;i<10 && L[0][i]==0;++i);
        for(j=S[0]-1;i!=-1 && L[0][j]==0;--j);
        if(i==10)
            Flag[0]=0;
        else if(j==-1)
            Flag[0]=1;
        else if(i-S[0]<=S[0]-j)
            Flag[0]=1;
        else
            Flag[0]=0;
```

```
//Till all requests are satisfied keep running
while(L[0]!=0)
begin
    if(L[0][S[0]]==1)//Services The Floor
    begin
        #1 Oflag[0]=1;
        #1 Oflag[0]=0;
        L[0][S[0]]=0;
    end

    if(S[0]==R[0])//at reached state or floor
    begin
        if(D[0]==1 || Flag[0]==1)//if going up currently
        begin
            //Finding if any requests are there above, if any the
nearest is obtained in 'i'
            i=S[0]+1;
            while(i<10 && L[0][i]==0)
            begin
                i=i+1;
            end
```

```
if(i<10)//Yes at top floor someone requested
Begin
//See if Other Lifts are near them by calculating hop count
        for(integer k=0;k<4;++k)
        begin
            KL[0]=L[k];
            if(KL[0][i]==1)
                begin
    //If other one moves in same direction or is at rest
                if(D[k]==1 || D[k]==0)
                begin
              //Going towards it so calculate hop count
                    if(i>=S[k])
                    begin
                        H[k]=i-S[k];
                    end
              //already crossed the floor set to high
                    else
                    begin
                        H[k]=10;
                    end
                end
    //If other one moves in opposite direction or is at rest
                else if(D[k]==2)
                begin
              //already crossed the floor set to high
```

```
                        if(i>S[k])
                        begin
                            H[k]=10;
                        end
                    //Going towards it so calculate hop count
                        else
                        begin
                            H[k]=S[k]-i;
//If there are other requests to this lift at top of other states then set
high hop count
                            for(integer p=S[k]+1;p<10;++p)
                            begin
                                if(L[0][p]==1)
                                    H[k]=10;
                            end
                        end
                    end
                end
                else
                    H[k]=10;
            end


            DFlag[0]=0;
```

//Change Direction if other lifts can service the top floors earlier than this.

```
                for(integer k=0;k<4;++k)
                begin
                    if(H[k]<H[0])
                        DFlag[0]=1;
                end
                if(DFlag[0]==1)//Changes
                    D[0]=2;
                else//Remains Same
                    D[0]=1;
            end
            else//If no request at top exists
            begin
                if(L[0]==0)//If no requests at bottom also
                    D[0]=0;
                else
                    D[0]=2;
            end


            //Get what state to goto next
            if(D[0]==1)
                R[0]=i;
            else if(D[0]==2)
            begin
```

```
        //Finds Nearest Bottom Request
        for(r=S[0]-1;r!=-1 && L[0][r]==0;--r);
        R[0]=r;
    //Boundary Condition When lift is at the least floor
        if(R[0]==15)
        begin
            D[0]=1;
            R[0]=i;
        end
    end
end
else if(D[0]==2 || Flag[0]==0)//if going down currently
begin

    //Finding if any requests are there below, if any the
nearest is obtained in 'i'
    i=S[0]-1;
     //See if bottom floors requested
    while(i!=-1 && L[0][i]==0)
    begin
        i=i-1;
    end
```

```
        if(i!=-1)//Yes at bottom floor someone requested
        begin
//See if Other Lifts are near them by calculating hop count
                for(integer k=0;k<4;++k)
                begin
                    KL[0]=L[k];
                    if(KL[0][i]==1)
                        begin
        //If other one moves in same direction or is at rest
                        if(D[k]==2 || D[k]==0)
                        begin
                       //Going towards it so calculate hop count
                            if(i<=S[k])
                            begin
                                H[k]=S[k]-i;
                            end
                             //already crossed the floor set to high
                            else
                            begin
                                H[k]=10;
                            end
                        end
        //If other one moves in opposite direction or is at rest
                        else if(D[k]==1)
                        begin
```

```
                         //already crossed the floor set to high
                         if(i<S[k])
                         begin
                             H[k]=10;
                         end
                   //Going towards it so calculate hop count
                         else
                         begin
                             H[k]=i-S[k];
//If there are other requests to this lift at bottom of other states then
set high hop count
                      for(integer p=S[k]-1;p>0;--p)
begin
                                 if(L[0][p]==1)
                                     H[k]=10;
                             end
                         end
                     end
                     end
                 else
                     H[k]=10;
             end

             DFlag[0]=0;
//Change Direction if other lifts can service the bottom floors earlier than
this.
```

```
        for(integer k=0;k<4;++k)
        begin
            if(H[k]<H[0])
                DFlag[0]=1;
        end
        if(DFlag[0]==1)//Changes
            D[0]=1;
        else//Remains Same
            D[0]=2;
end
else//If no request at bottom exists
begin
    if(L[0]==0)//If no requests at top also
        D[0]=0;
    else
        D[0]=1;
end


//Get what state to go next
if(D[0]==2)
    R[0]=i;
else if(D[0]==1)
begin
//Finds Nearest Top Request
    for(r=S[0]+1;r<10 && L[0][r]==0;++r);
```

```
            R[0]=r;
          //Boundary Condition When lift is at the Top most floor
            if(R[0]==10)
            begin
                D[0]=2;
                R[0]=i;
            end
        end
    end
end


//Direction Decided
if(D[0]==1)//Go up
begin
    S[0]=S[0]+1;
    #1;
end
else if(D[0]==2)//Go Down
begin
    S[0]=S[0]-1;
    #1;
end
```

```
            else//Stop
            begin
                #1;
            end
        end
    end
```

2. Test Bench:

```
module LFT_tb;
    reg[11:0] Inp;
    LFT lft(Inp);
    initial
    begin
        #10 Inp=12'b000000110010;
        #1  Inp=12'b010010010010;
        #1  Inp=12'b010100000010;
        #1  Inp=12'b000100000001;
        #1  Inp=12'b100100010000;
        #1  Inp=12'b110101010010;
        #1  Inp=12'b100000000001;
        #1  Inp=12'b110000000001;
    end
endmodule
```

Results:

Some Random lift requests are given every time to the control system as in test bench. The States of Lifts are shown Pictorially as Follows:

Output:

Following are the notations used in output:

'+': Requested

'-': Not-Requested

'^': Going-up-towards-it

'v': Going-down-towards-it

'[]': Lift-Open

'I': Lift-at-Rest

To understand the simulation better the series of images are made into a video here:(Click on the Link or copy the link)

Ouput Simulation or link: https://drive.google.com/file/d/1ltKs-bdkjOT9-7CykZgbTF94CDUtOXkq/view?usp=sharing

Conclusion:

Hence a Lift Control System is made in an aim with minimizing passenger waiting time.

Applications:

Most buildings with high number of floors have lifts in it. In most of them waiting time is considerably high if the lifts are highly used. As, the experiment done here aids in reduction of average waiting time, it can be used in apartments, malls, etc. for better usage.

Verilog Questions

Question no.1

Answer the below questions:

a. What is the output sequence for the given input data sequence 0010101101101101111?
b. What is the behavior of the above Finite state machine?
c. Decide the Flip-flop which can be used by avoiding external logic gates.

Ans.

a) Let S0 be the initial state of the FSM. As this is a moore circuit Output depends only on the state it is present. For the given input sequence we show the output as follows:

Current State	Output	Input	Next State
S0	0	0	S0
S0	0	0	S0
S0	0	1	S1
S1	1	0	S1
S1	1	1	S0
S0	0	0	S0
S0	0	1	S1
S1	1	1	S0
S0	0	0	S0
S0	0	1	S1
S1	1	1	S0
S0	0	0	S0
S0	0	1	S1
S1	1	1	S0
S0	0	0	S0
S0	0	1	S1
S1	1	1	S0
S0	0	1	S1
S1	1	-	-

b) This FSM is a moore circuit that remains in the same state if input is 0 or changes to the other state when input is 1 and output also gets complemented.

c) The FSM has same behaviour as a T-Flip Flop. Consider T as input to the FSM. So, when T=1 it Toggles the state that corresponds to complemented output, and if T=0 it remains in same state.