Dhirubhai Ambani
Institute of Information and Communication Technology

EL467 - Digital Programming

- Prof. Yash Agarwal

Implementation of Elevator Control System using Verilog HDL

Group members:

201901078 - Jaydip Mokariya
201901100 - Shubham Patel

NOTE: [Project PPT link]

- **Abstract**

Nowadays construction of high rise buildings are increasing in which the primary function of connectivity is fulfilled by Elevators. They help in the situation of transportation and evacuation in the building. The aim of the project is to design and implement the elevator control system using Verilog hardware descriptive language (HDL). The elevator controller is based on the concept of finite state machine technology. According to FSM technology, the elevator process can be defined with the help of different states. In FSM technology, there is change from one state to another state likewise in the elevator system there is change from one floor to another. XILINX has been used as a code writing platform and results were simulated using Modelsim simulator. This project discusses the implementation of the elevator system and also discusses simulated results.

- **Model description**

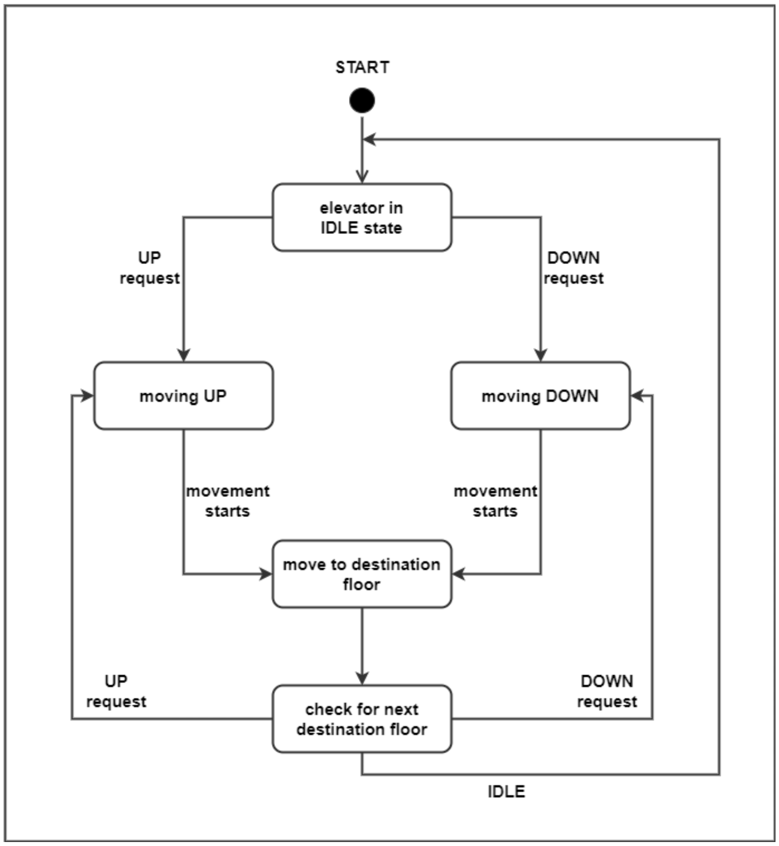
We have created elevator system in Verilog. It takes in a 4-bit input req_floor representing the floor number that a user has requested, and a clock signal clk. The code uses a binary list floor_status to keep track of which floors have been requested. When a new floor is requested, the code updates floor_status to reflect this. The code also keeps track of the highest requested floor in up_floor and the lowest requested floor in down_floor. These values are updated every time a new floor is requested. The elevator has a current floor, represented by curr_floor, and a target floor represented by target_floor. The state variable is a 3-bit value that represents the current state of the elevator. The possible values are 2'b00, 2'b01, and 2'b11, which correspond to the IDLE, MOVING DOWN, and MOVING UP, respectively. The target_floor register is a 4-bit binary number that represents the floor that the elevator is currently moving towards. The curr_floor register is a 4-bit binary number that represents the current position of the elevator. The behavior of the elevator is determined by several always blocks, which specify the behavior of the module at different times.

The first always block specifies the initial condition for the elevator, which is that the target_floor is set to the req_floor when the down_floor and up_floor are the same and equal to the target_floor.

The second always block updates the floor_status register with the requested floor when a new request is made.

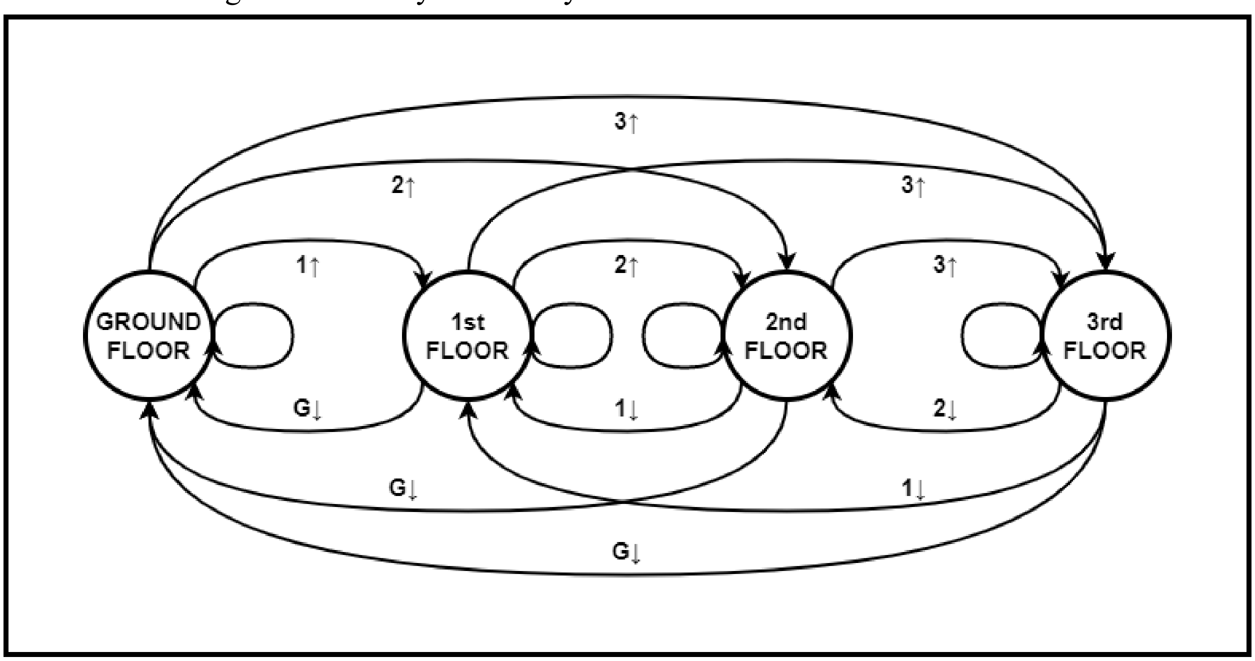
The third always block is used to update up_floor and down_floor variables . If new requested floor is above up_floor then we set up_floor=req_floor and we will also update down_floor in similar manner. Last always block is used to control lift. We have used disk scheduling algorithm for this code. We will move in one direction until we reach the last requested floor in that direction and then we will change the direction and we will do the same for other direction as well . And while we are moving from current floor to target floor , at every floor we will check floor_status . So is current floor is requested in the past then we can stop on that floor for some time and then move ahead.

- **Work flow chart**



- **Diagram**

FSM state diagram for 4 story elevator system:



- **Coding**

Elevator main module code:

```
module elevator(input [3:0] req_floor,input clk);

  reg floor_status[7:0];//binary list to represent floor_status
  reg [2:0] state=2'b00;
  reg[3:0] up_floor=4'b0000;//represents highest requested floor
  reg[3:0] down_floor=4'b0000;//represents lowest requested floor
  reg[3:0] target_floor=4'b0000;//target floor
  reg[3:0] curr_floor=4'b0000;// current floor
  always @ (*) //intial condition for elevator
    begin
      if(down_floor==up_floor && target_floor==up_floor)
        begin
          target_floor=req_floor;
        end
    end
  always @ (req_floor)//updating floor_status list
  begin
    floor_status[req_floor]=1'b1;//1 represents that the floor is in
the list
  end

  always @ (*) begin //updating up_floor and down_floor on every
request
      if(req_floor>up_floor)
        begin
          up_floor=req_floor;
        end
      else if(req_floor<down_floor)
        begin
          down_floor=req_floor;
        end
     end
  always @(posedge clk) begin
     if(up_floor==0 && down_floor==0 && curr_floor==0)
      begin
        state=2'b00;
        target_floor=0;
```

```
      end
  case (state)
    //2'b11 ==> moving up
    //2'b01 ==> moving down
    //2'b00 ==> idle
    // idle state: wait for a target floor
    2'b00:
      if(target_floor>curr_floor)//if target_floor is above curr_floor
then set state=2'b11
          begin
            state=2'b11;
          end
      else if(target_floor<curr_floor)
          begin
            state=2'b01;
          end
      else
        state=2'b00;

    2'b11:
      if(floor_status[curr_floor]==1)//move up until last floor reached
          begin
            #10
            floor_status[curr_floor]=0;

            //curr_floor++;
          end
    else if(curr_floor==target_floor || curr_floor==7)
        begin
          target_floor=down_floor;
          up_floor=0;
          state=2'b01;
          curr_floor--;
        end
       else
        begin
          state=2'b11;
          curr_floor++;
        end
```

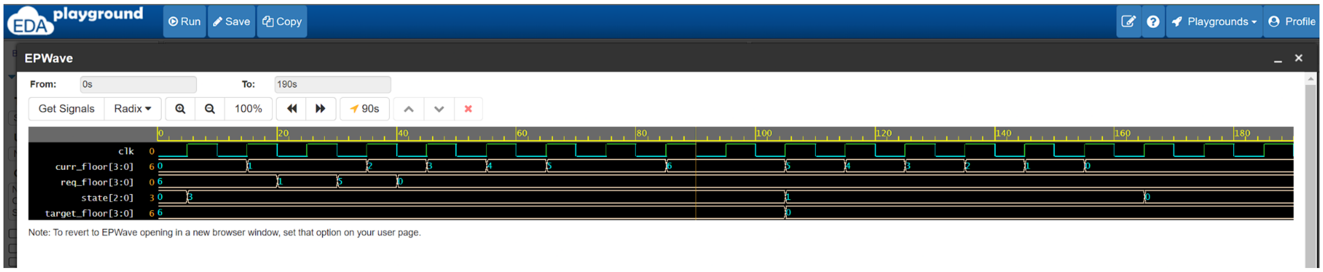
```
    2'b01://move down until down floor reached
         if(floor_status[curr_floor]==1)
          begin
             #10
            $display("aa");
             floor_status[curr_floor]=0;
          end
    else if(curr_floor==target_floor || curr_floor==0)
        begin
          target_floor=up_floor;
          down_floor=0;
          state=2'b11;
          curr_floor++;
        end
      else
        begin
          state=2'b01;
          curr_floor--;
        end
  endcase
end

endmodule
```

Elevator test bench module code:

```
module testbench;

// input signals
  reg clk;
  reg [3:0] req_floor;

// output signals



// instantiate the elevator controller
  elevator elevatorcontroller(req_floor,clk);
```

```
// test bench code
initial begin
  // set initial values for the input signals
  $dumpfile("abc.vcd");
  $dumpvars(0,testbench);
  clk = 0;

  req_floor = 4'b0110;

  // run the test for 100 clock cycles
  #20;

  // set the target floor to 3
  req_floor = 4'b001;

  // run the test for 100 clock cycles
  #10;

  // set the target floor to 1
  req_floor = 4'b0101;

  // run the test for 100 clock cycles
  #10;

  // set the target floor to 0
  req_floor = 4'b0000;

  // run the test for 100 clock cycles
  #100;
  //targetFloor=4'b0100;
  #50;

  $finish;
end

// clock generator
always #5 clk = ~clk;

endmodule
```

- **Output waveform**



- **Conclusion**

  This project explains how to implement and simulate an elevator controller using Verilog HDL. We learned the basic idea of how the normal elevators run in many cases. The resources used for designing this system are very less which makes it cost efficient as compared to other technologies. The state changing behavior of FSM technology can be used in elevators for floor changing. The most challenging and time consuming part is debugging but we learned something new.