Dhirubhai Ambani Institute of Information and Communication Technology

Gandhinagar, Gujarat

EL-203 - **Embedded Hardware Design**

(Prof. Biswajit Mishra)

(Prof. Yash Agrawal)

GROUP MEMBER:

STUDENT ID	NAME	CONTRIBUTION
202201166	NISHIL PATEL	24%(CODE AND IMPLEMENTATION)
202201167	HET GANDHI	22% (STATE DIAGRAM AND IMPLEMENTATION)
202201169	JYOT VASAVA	15%(REPORT AND OTHER HELP)
202201170	TATHYA PRAJAPATI	24%(CODE AND IMPLEMENTATION)
202201171	SAHIL CHAUDHARI	15%(REPORT AND OTHER HELP)

PROBLEM:

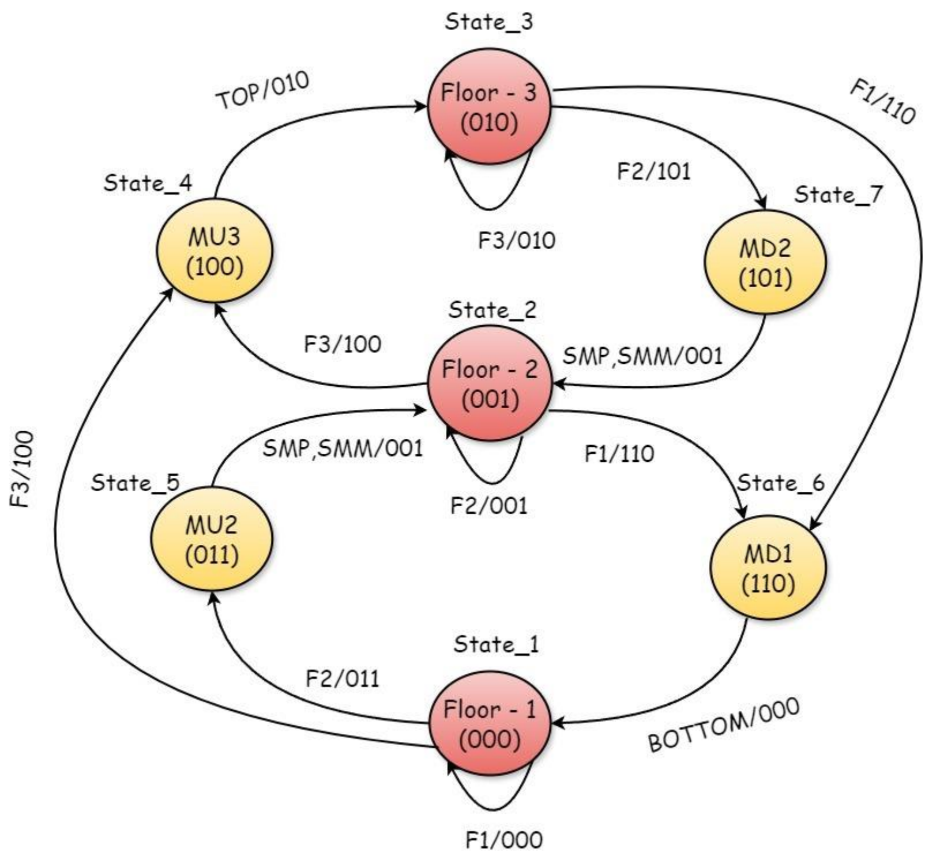
The task is to create a basic elevator setup with 3 floors using VHDL on an FPGA board. The elevator consists of three buttons labelled "Call" for each floor, with corresponding indicators showing when the elevator is on its way, plus a rack-and-pinion gear system and motor whereby a simple platform representing the lift cage can be driven up and down. The position of platform is detected by four more switches- one each for “top”, “middle-plus”, “middle-minus” and “bottom”. The two switches which detect the platform in the middle position are operated by the rack cage, rather than an isolated point on it – only when both “middle_switches” are closed is the platform properly aligned. The top and bottom position sensing arrangements don’t need this refinement because any “up” movement with the “top” switch active is illegal, as is “down” movement with the “bottom” switch active.

STATE DIAGRAM

Firstly, We started with 3 state diagram :

The problem in this 3 state FSM is that we can't considering the state in which the lift is moving we described only floor is over output at any instant and that is wrong FSM. So, next we considered 5 state FSM :

Again, the problem in 5 state FSM with 2 extra states (Moving up , Moving down) that is at any instance we are moving from floor one to floor two and same for moving second to third floor both indicating the same state that is the Moving up state but that should be define as different and need to be distinguish similarly problem is going with Moving down so we finally came to the state number 7 as shown below:



Elevator (Lift) Design

The state diagram containing 7 states, 3 state for every floor, 4 states are moving states.

In state floor-1, floor-2, floor-3represents floors, mu-2 and mu-3 states represent moving up to the first floor from ground and second floors respectively, md-1 and md-2 states represent moving down to the ground floor from first and first floors respectively.

BASIC CASE:

Floor-1

- If floor-1 receives 1 input the next state becomes mu-2 and when smp is received the current state will become floor-2.
- If floor-1 receives 2 input the next state becomes md-2 and when top is received the current state will become floor-3.

Floor-2

- If the floor-2 receives 2 input next state become mu-3 and when top is received the current state will become floor-3.
- If the floor-2 receives 0 input the next state becomes md-1 and when bottom is received the current state will become floor-1.

Floor-3

- If the floor-3 receives 1 input next state become md-1 and when smm is received the current state will become floor-2.
- If the floor-3 receives 0 input next state become md-2 and when bottom is received the current state will become floor-1.

If the any floor receives its floor number input elevator doesn't move.

CODE SNIPPET

```
module Lift_Project(
    input clock,          // Generated Clock signal
    input reset,          // Reset input
    input B1, B2, B3,  // Push buttons for floors 1, 2, and 3
    input SBT, SMM, SMP, STP,  // Sensors for bottom, middle-, middle+, top
    output reg [2:0] state,  // Output state of the lift
    output reg [1:0] motor   // Motor state output
);
    // State encoding
    localparam F1 = 3'b000,  // Floor 1
        F2 = 3'b001,  // Floor 2
        F3 = 3'b010,  // Floor 3
        MU2 = 3'b011, // Moving up to Floor 2
        MU3 = 3'b100, // Moving up to Floor 3
        MD2 = 3'b101, // Moving down to Floor 2
```

```
   MD1 = 3'b110;  // Moving down to Floor 1
reg [2:0] present_state, next_state;

//Maintaing State register
always @(posedge clock or posedge reset) begin
   if (reset)
      present_state <= F1;
   else
      present_state <= next_state;
end
// Next state logic
always @(*) begin
   case (present_state)
      F1: begin
         if (B2) next_state = MU2;
         else if (B3) next_state = MU3;
         else next_state = F1;
      end
      F2: begin
         if (B1) next_state = MD1;
         else if (B3) next_state = MU3;
         else next_state = F2;
      end
      F3: begin
         if (B1) next_state = MD1;
         else if (B2) next_state = MD2;
         else next_state = F3;
      end
      MU2: begin
         if (SMM && SMP) next_state = F2;
         else next_state = MU2;
```

```
        end
        MU3: begin
          if (STP) next_state = F3;
          else next_state = MU3;
        end
        MD2: begin
          if (SMM && SMP) next_state = F2;
          else next_state = MD2;
        end
        MD1: begin
          if (SBT) next_state = F1;
          else next_state = MD1;
        end
        default: next_state = F1;
      endcase
    end
    // Output logic
    always @(present_state) begin
      case (present_state)
        F1, F2, F3: motor = 2'b00;  // Motor stop
        MU2, MU3: motor = 2'b01;    // Motor up
        MD1, MD2: motor = 2'b10;    // Motor down
        default: motor = 2'b00;     // Motor off
      endcase
    end
    // Assign the state output for testing or debugging
    always @(present_state) begin
  // present_state = next_state;
      state <= present_state;
endmodule
```