Project Report– Digital System Design

5-Floor Elevator Control System using FSM

Submitted for partial fulfillment of award of

BACHELOR OF ENGINEERING

Degree In

Computer Systems Engineering

Group Members:

Ayesha Siddiqua(133-22-0005)

Habeeban Memon(133-22-0033)

Guided by:

Dr: Kashif Hussain

Department of Computer Systems Engineering

Sukkur IBA University

November 2025

Table of Contents

1. Abstract

This report presents the design, implementation, and verification of a 5-floor elevator control system implemented as a Moore finite state machine (FSM). The elevator services requests from floors 0 through 4. The design prioritizes simplicity, determinism, and modularity: a request-processing unit computes the nearest active request with respect to the current floor; a Moore FSM controls motion (up/down), door opening, and idle behavior; a timer module ensures the door remains open for a fixed dwell time. The system is implemented in Verilog HDL and verified using a behavioral testbench. Expected waveforms and simulation observations are included.

2. Problems /Objectives

2.1 Project

Traditional elevator controllers are simple and lack optimized decision-making. This results in slow response, inefficient travel, and poor user experience.

1. No Priority Handling

Elevators serve requests in the order they are pressed, without considering direction or urgency.
This increases waiting time and makes the system slow during peak usage.

2. Inefficient Floor Movement

The elevator may travel to distant floors while closer requests remain unserved.
This wastes time, increases energy consumption, and reduces efficiency.

3. No Nearest-Request Optimization

When multiple requests occur, the system does not check which floor is closest.
This causes unnecessary delay, slower service, and user frustration.

4. Poor Door Timing Control

Doors may stay open too long or close too fast.
Incorrect timing affects safety, flow of passengers, and overall performance.

5. Weak Emergency Handling

Emergency/Estop buttons may not fully stop the elevator or keep the motor and doors controlled.
This can be risky during malfunctions or panic situations.

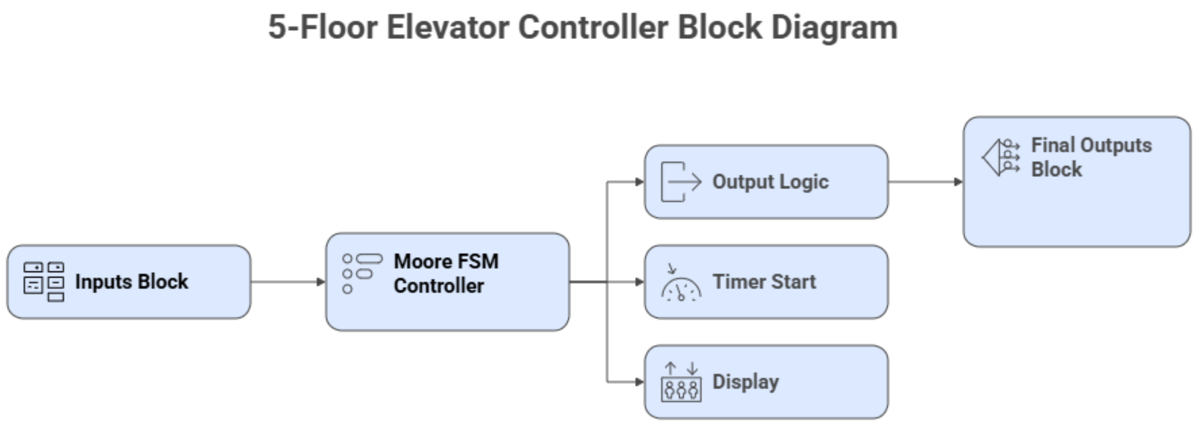
6. No Clear FSM State Management

Traditional systems lack structured states like Idle, Going Up/Down, Door Open, Timer Control.
As a result, behavior becomes unpredictable, and request handling is unorganized.

2.2 Project Objectives

- Develop a reliable Moore FSM with the states: IDLE, GOING_UP, GOING_DOWN, DOOR_OPEN.
- Implement a nearest-request algorithm to minimize travel time.
- Implement a timer module to manage door dwell time.
- Provide a testbench to exercise normal operation sequences (multiple requests, sequential servicing, direction changes).
- Produce documentation, truth tables, state diagrams, and complete Verilog source code.

3. Block Diagram & Design Methodology



3.1 Block Diagram Overview

1. **Inputs Block**: User floor requests (0–4), current floor, and target detection signals are received here and forwarded to the controller.
2. **Moore FSM Controller** — Processes all inputs and decides the current operating state (Idle, Up, Down, Door-Open), generating control signals based only on its active state.
3. **Output Logic**: Converts FSM state signals into executable operations such as motor up, motor down, stop, and door open control.
4. **Timer Start**: Triggered when elevator reaches a selected floor; keeps the door open for a fixed time before moving to next request.

5. **Display and Final Outputs Block:** Shows current floor/direction on display and generates final hardware outputs for motor & door control.

3.2 Design Methodology

The design followed a waterfall like phased approach described below.

3.2.1 Phase 1: Requirement Analysis

- Determine inputs: `clk`, `reset`, `requests[4:0]`, `currFloor[2:0]`, `at_target` (floor sensor), `timer_done` (from timer).
- Determine outputs: `door_open`, `motor_up`, `motor_down`, `idle`.
- Decide on Moore FSM to ensure outputs are stable across clock edges and depend only on state.

3.2.2 Phase 2: System Partitioning

- Separate combinational request arbitration from sequential state machine.
- Create a small timer sub-module to provide door dwell timing which can be parameterized.
- Keep top-level integration simple for easy verification.

3.2.3 Phase 3: FSM Development

- Define Moore states: `IDLE`, `GOING_UP`, `GOING_DOWN`, `DOOR_OPEN`.
- State transitions triggered by `req_exists`, `nearest_req` vs `currFloor`, `at_target`, and `timer_done`.
- Guarantee safe reset to `IDLE`.

3.2.4 Phase 4: Nearest Request Algorithm

- Scan `requests[4:0]` and compute Manhattan distance between `currFloor` and each requested floor.
- Keep track of the minimum distance and corresponding floor index.
- If multiple floors at same distance exist, the implementation selects the first encountered (lowest index) — this is a deterministic tie-breaker.

3.2.5 Elevator FSM (Moore Machine)

- Outputs are functions of the current state only. For example:
    - `motor_up = (state == GOING_UP)`
    - `motor_down = (state == GOING_DOWN)`
    - `door_open = (state == DOOR_OPEN)`
    - `idle = (state == IDLE || state == DOOR_OPEN)`
- Transitions:

    - From `IDLE`: if `req_exists` and `nearest_req > currFloor` then `GOING_UP`; if `nearest_req < currFloor` then `GOING_DOWN`; if equal then `DOOR_OPEN`.
    - From `GOING_UP`/`GOING_DOWN`: when `at_target` asserted, go to `DOOR_OPEN`.
    - From `DOOR_OPEN`: when `timer_done` asserted, go to `IDLE`.

3.2.6 Timer Module

- The timer is started when the FSM enters `DOOR_OPEN` (or the output `door_open` goes high).
- After a parameterized number of clock cycles, it asserts `timer_done` for one cycle to indicate dwell expiration.
- Implementation choices:
    1. Simple counter that counts `N` clock cycles.
    2. Optionally reset/disable when leaving `DOOR_OPEN` early.

3.2.7 Output Control Module

- In a real system, `motor_up`/`motor_down` would control direction pins for motor driver and `door_open` would control a door actuator relay.
- For simulation, these are boolean signals that can be observed in waveforms.

3.3 Modular Design Advantages

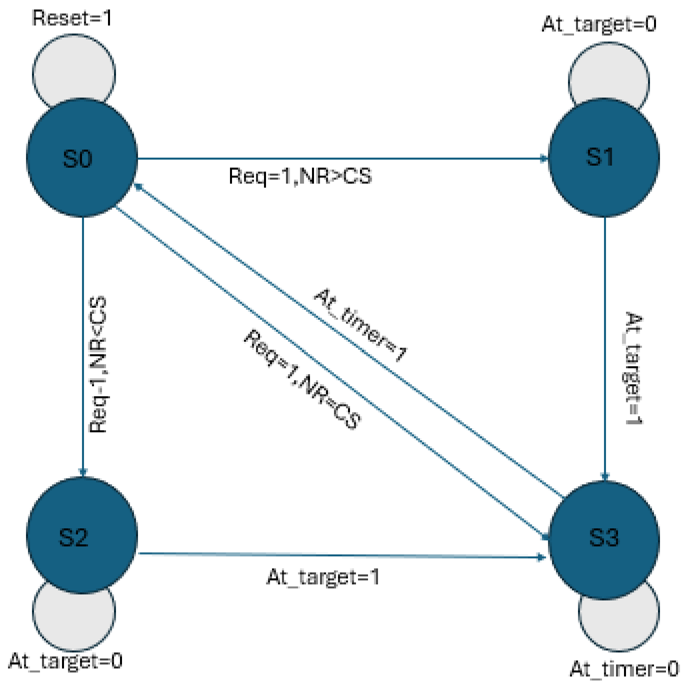
Advantages of Modular Design in this Elevator FSM Project

1. **Clarity and Maintainability**
    - Each module has a clear responsibility: FSM, Timer, Request Arbitration, Output Control.
    - Future changes (e.g., door timing) only affect the Timer module, leaving the rest of the system unchanged.
2. **Reusability**
    - Modules are independent, so components like the Timer or Output Control can be reused in other projects without rewriting code.
3. **Simplified Debugging and Testing**
    - Problems can be isolated easily: FSM, Timer, or Request Arbitration can be unit tested separately.
    - Makes errors in identifying during system integration much easier.
4. **Scalability**
    - Adding more floors.
    - The overall system structure remains intact.
5. **Parallel Development**
    - Multiple developers can work simultaneously: one on FSM, another on Timer or Request Handling.
    - Speeds up development and simplifies coordination.
6. **Better Simulation and Verification**

- Each module can be observed separately in waveforms.
- Complex behaviors can be verified step by step.

4. Truth Tables & State Diagrams

The FSM is Moore. Outputs depend on the current state only.



4.1 State Transition Table

STATES

STATES	
S0	IDLE
S1	GOING_UP
S2	GOING_DOWN
S3	DOOR_OPEN

CS	**R**	**N VS C**	**At_Target**	**Timer_done**	**NS**
S0	0	X	X	X	S0
S0	1	N > C	X	X	S1
S0	1	N< C	X	X	S2
S0	1	N == C	X	X	S3
S1	X	X	1	X	S3
S1	X	X	0	X	S1
S2	X	X	1	X	S3
S2	X	X	0	X	S2
S3	X	X	X	1	S0
S3	X	X	X	0	S3

4.2 State Encoding

STATES	**BINARY**
S0	**00**
S1	**01**
S2	**10**
S3	**11**

4.3 Next-State Logic Table

CS	R	N>C	N<C	N==C	At_target	Timer_done	NS
S0	0	0	0	0	X	X	S0
S0	1	1	0	0	X	X	S1
S0	1	0	1	0	X	X	S2
S0	1	0	0	1	X	X	S3
S1	X	X	X	X	0	X	S1
S1	X	X	X	X	1	X	S3
S2	X	X	X	X	0	X	S2
S2	X	X	X	X	1	X	S3
S3	X	X	X	X	X	0	S3
S3	X	X	X	X	X	1	S0

4.4 Next-State Logic Equations

- at_target is indicated as T
- timer_done is indicated as TD
- NS1 = MSB, NS0 = LSB

1. **NS1 = (CS==S0 & R & (N<C | N==C)) | (CS==S1 & T) | (CS==S2 & T)**
2. **NS0 = (CS==S0 & R & (N>C | N==C)) | (CS==S1 & T) | (CS==S2 & T) | (CS==S3 & ~TD**)

4.5 Output Logic Table (Moore)

CS	door_open	motor_up	motor_down	idle
S0	0	0	0	1
S1	0	1	0	0
S2	0	0	1	0
S3	1	0	0	1

4.6 Output Logic Equations

1. **door_open = (CS == S3)**
2. **motor_up = (CS == S1)**
3. **motor_down = (CS == S2)**
4. **idle = (CS == S0) | (CS == S3)**

5. Schematic Diagram

1. Inputs

Input	Description	Schematic Label
`clk`	System clock	Clock line to FSM
`reset`	Reset FSM to idle	Reset line
`requests[4:0]`	Floor requests (0–4)	Request buttons
`currFloor[2:0]`	Current floor index	Floor sensor
`at_target`	Elevator reached requested	Target sensor

	floor	
`timer_done`	Door dwell timer expired	Timer output

2. Nearest Floor / Distance Logic

Signal	Description	Schematic Label
`best_distance[2:0]`	Manhattan distance to requested floors	Comparator / Distance calculator
`nearest_req[2:0]`	Index of nearest requested floor	Priority encoder output
`req_exists`	Any floor requested?	OR gate over `requests[4:0]`

3. FSM

Signal	Description	Schematic Label
`cs_reg[1:0]`	Current state	FSM State Register
`ns[1:0]`	Next state	FSM Combinational Logic
`ns1_out`	Intermediate combinational net	FSM internal wire

4. Outputs

Signal	Description	Schematic Label
`motor_up`	Move elevator up	Motor Up output
`motor_down`	Move elevator down	Motor Down output
`door_open`	Open elevator door	Door actuator
`idle`	Elevator idle	LED or status output

5. Control Flow (Schematic Blocks)

- Requests Input, OR gate, req_exists
- Distance Calculation Block, outputs nearest_req
- Compare nearest_req vs currFloor, decide GOING_UP / GOING_DOWN / DOOR_OPEN
- FSM Block (Registers + Next State Logic), updates cs_reg each clock
- Outputs derived from cs_reg: motor_up, motor_down, door_open, idle

Our schematic is quite large and detailed, so below is the Drive link from where you can view it.

https://drive.google.com/file/d/1-eGxvYVb53LHi5Gu97w_1JCJ9kfStkhx/view?usp=sharing

6. Implementation in Verilog HDL

This section describes the modules that implement the design and explains the choices made. The complete Verilog code is included in the Appendix.

6.1 FSM Module

The elevator FSM is implemented as a Moore machine:

- State register: cs (current state), ns (next state), updated on the clock's rising edge.
- Reset: synchronous with reset asserted asynchronously, returns FSM to IDLE.
- State type: typedef enum logic [2:0] state_t used for readability.
- Next-state logic: implemented in always_comb for pure combinational behavior.
- Outputs: derived from cs (Moore), ensuring stable outputs each cycle.

```
assign door_open  = (cs == DOOR_OPEN);

assign motor_up   = (cs == GOING_UP);

assign motor_down = (cs == GOING_DOWN);

assign idle       = (cs == IDLE || cs == DOOR_OPEN);
```

6.2 Timer Module

A parameterized door timer counts clock cycles and asserts timer_done when the dwell time expires.

```
module door_timer #(parameter WIDTH = 24, parameter LIMIT = 24'd5_000_000) (

   input logic clk, reset, start,

   output logic timer_done

);

   logic [WIDTH-1:0] cnt;

   always_ff @(posedge clk or posedge reset) begin

      if (reset) begin
```

```
        cnt <= 0; timer_done <= 0;
      end else if (start) begin
        if (cnt >= LIMIT) begin
          cnt <= 0; timer_done <= 1;
        end else begin
          cnt <= cnt + 1; timer_done <= 0;
        end
      end else begin
        cnt <= 0; timer_done <= 0;
      end
    end
endmodule
```

In simulation, LIMIT can be set small (e.g., 10 cycles) to speed up testing.

6.3 Request Processing Module

- Loops over requests[4:0] and calculates absolute distance from currFloor.
- Maintains a running minimum to find the nearest requested floor.
- Tie-breaking: chooses the first request with minimum distance.

```
always_comb begin
    req_exists = 0;
    best_distance = 7;
    nearest_req = currFloor;
    for (int i = 0; i < 5; i++) begin
      if (requests[i]) begin
        req_exists = 1;
```

```
            distance = (currFloor > i) ? (currFloor - i) : (i - currFloor);

            if (distance < best_distance) begin

                best_distance = distance;

                nearest_req = i[2:0];

            end

        end

    end

end
```

6.4 FSM Next-State Logic

- Uses a case statement based on current state cs.
- Transitions are triggered by req_exists, nearest_req vs currFloor, at_target, and timer_done.

```
always_comb begin

    ns = cs;

    case (cs)

        IDLE: begin

            if (reset) ns = IDLE;

            else if (req_exists) begin

                if (nearest_req > currFloor) ns = GOING_UP;

                else if (nearest_req < currFloor) ns = GOING_DOWN;

                else ns = DOOR_OPEN;

            end else ns = IDLE;

        end
        GOING_UP: ns = (at_target) ? DOOR_OPEN : GOING_UP;

        GOING_DOWN: ns = (at_target) ? DOOR_OPEN : GOING_DOWN;

        DOOR_OPEN: ns = (timer_done) ? IDLE : DOOR_OPEN;

        default: ns = IDLE;

    endcase

end
```

6.5 State Register

```
always_ff @(posedge clk or posedge reset) begin

    if (reset) cs <= IDLE;

    else cs <= ns;

end
```

6.6 Top-Level Integration

- All modules are combined in elevator_fsm_5floor.
- Inputs: requests, currFloor, at_target, timer_done.
- Outputs: motor_up, motor_down, door_open, idle.

6.7 Testbench

- Drives clk, reset, requests, currFloor, at_target, timer_done.
- Tests multiple scenarios: multiple requests, nearest request service, door timer, movement up/down.

```
initial begin

    reset = 1; requests = 0; currFloor = 0;

    at_target = 0; timer_done = 0;

    #25 reset = 0;
    // Request floors 1 & 4
    #20 requests[1] = 1; requests[4] = 1;

    // Move to floor 1
    #80 currFloor = 1; at_target = 1; #10 at_target = 0;
    #100 timer_done = 1; #10 timer_done = 0; requests[1] = 0;

    // Move to floor 4
    #80 currFloor = 2; #80 currFloor = 3; #80 currFloor = 4;
    at_target = 1; #10 at_target = 0;
    #100 timer_done = 1; #10 timer_done = 0; requests[4] = 0;

    // Request floor 2 and go down
    #50 requests[2] = 1; #80 currFloor = 3; #80 currFloor = 2;
    at_target = 1; #10 at_target = 0;
    #100 timer_done = 1; #10 timer_done = 0; requests[2] = 0;

    #300; $display("Simulation done!"); $finish;
end
```

- Monitoring: Prints current floor, requests, state, and outputs for verification.

6.8 Simulation Results & Observations

Fig 1: System has just come out of reset, elevator is at floor 0, door is closed. Floor-4 request is active but elevator is still fully idle — no motor started, nothing moving yet.

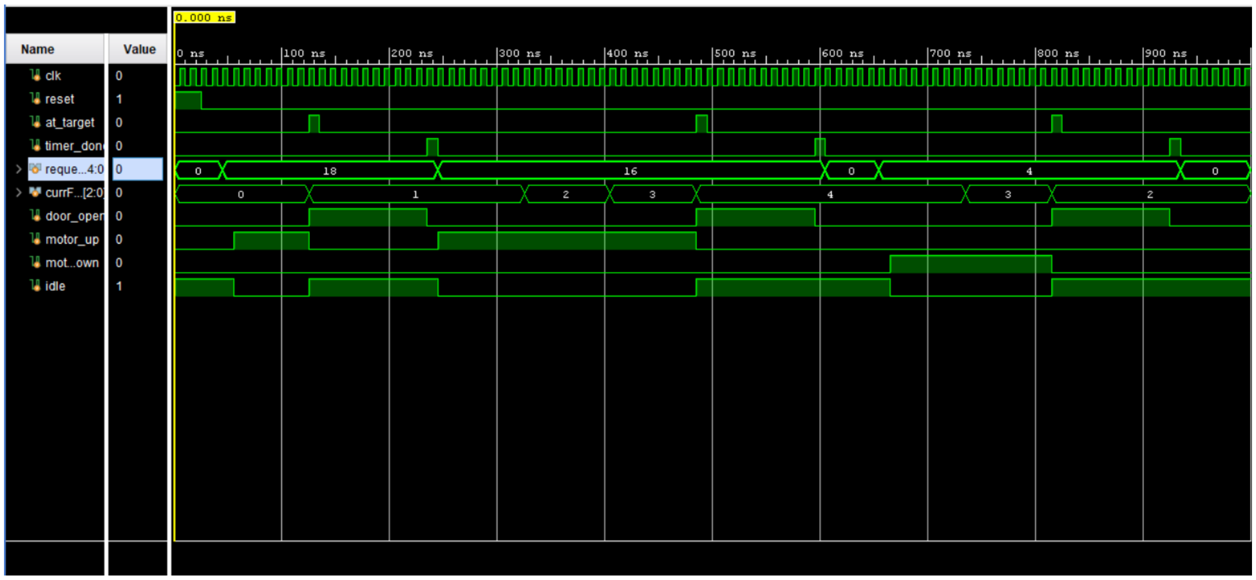


Fig 2: Elevator has left idle state, door is now fully closed. motor_up just turned ON elevator has started moving upwards from floor 0 towards floor 4.

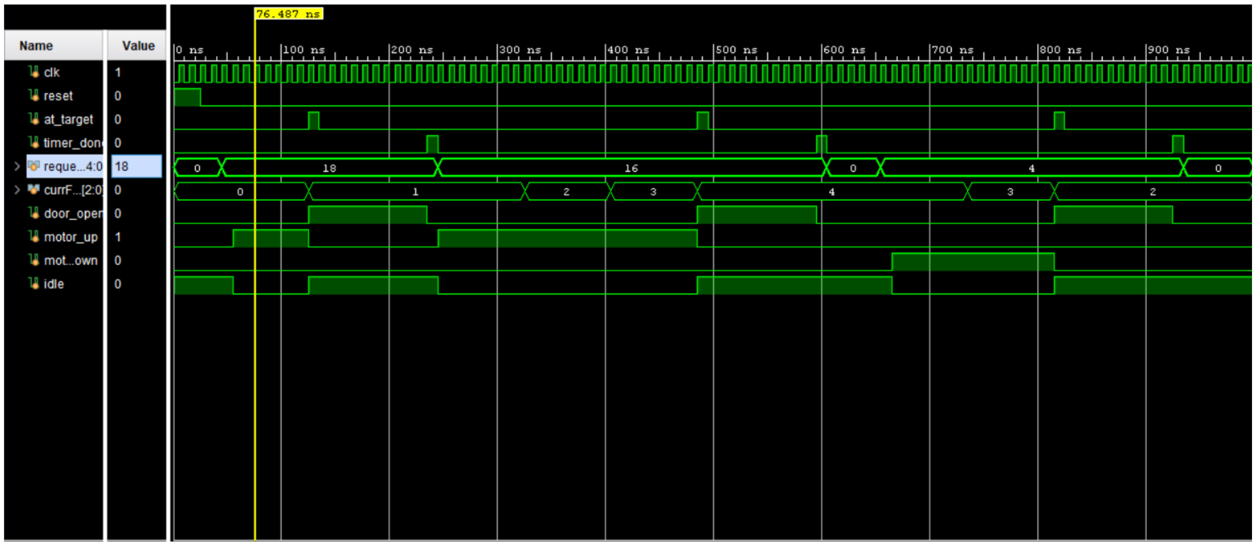


Fig 3: Elevator is actively moving upward, motor_up is still high. Current floor has just changed from 0 to 1 – elevator has successfully passed floor 1 and is heading toward floor 4.

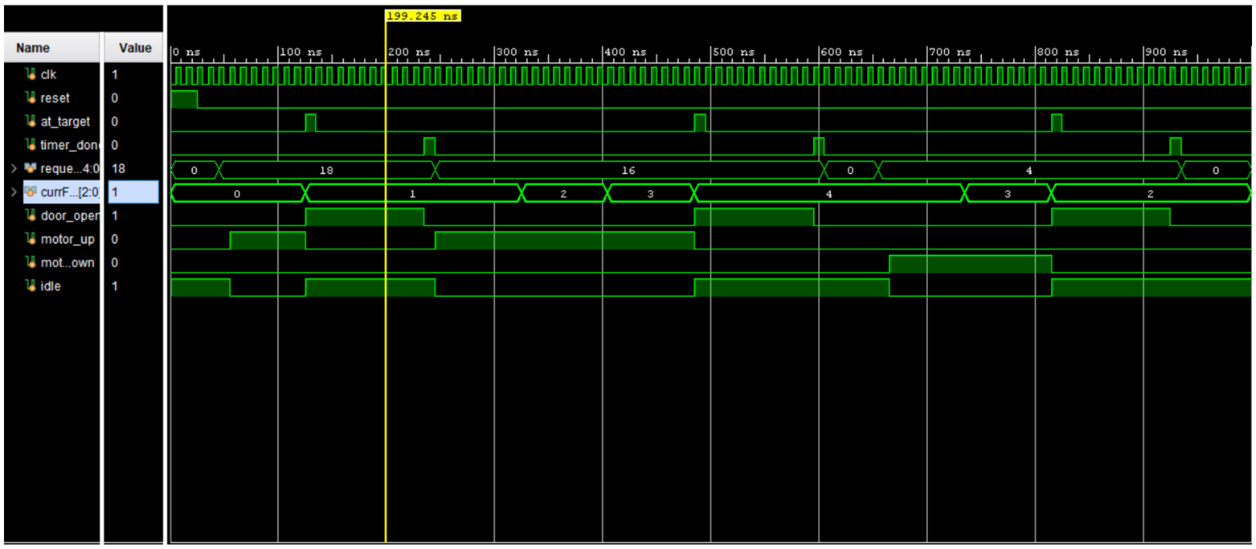


Fig 4: Elevator is still moving upward, motor_up remains active. Current floor has just transitioned from 1 to 2 – elevator has now passed floor 2 and continues toward floor 4.

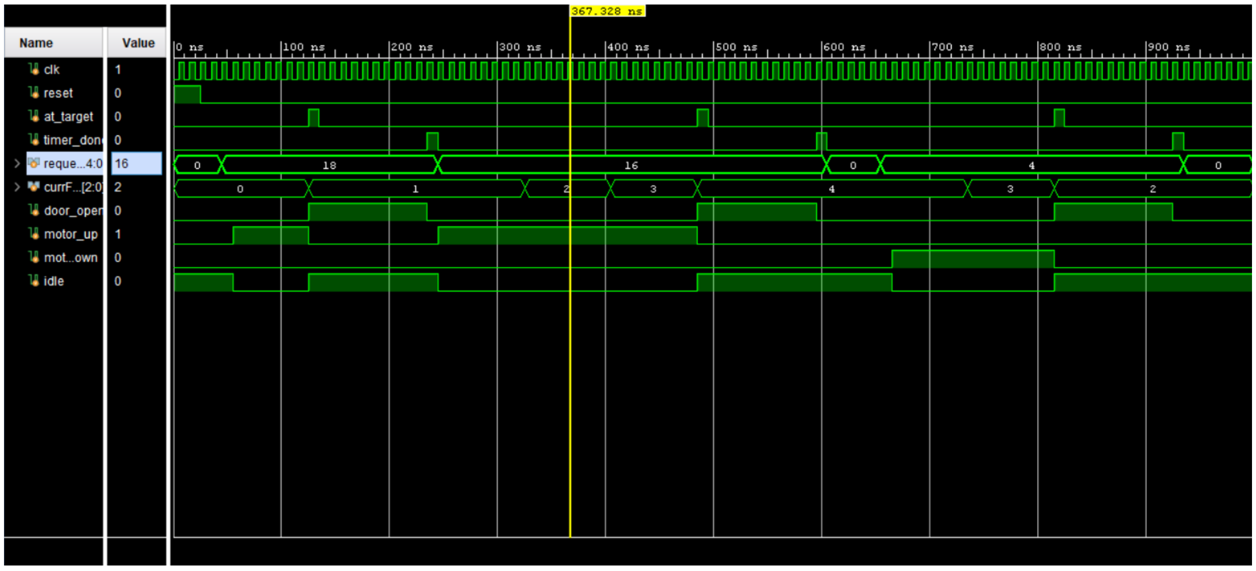


Fig 5: Elevator has just arrived at floor 4 currF changed to 4 and at_target pulsed high. motor_up instantly turned OFF, door is still closed, now waiting for the door-open timer.

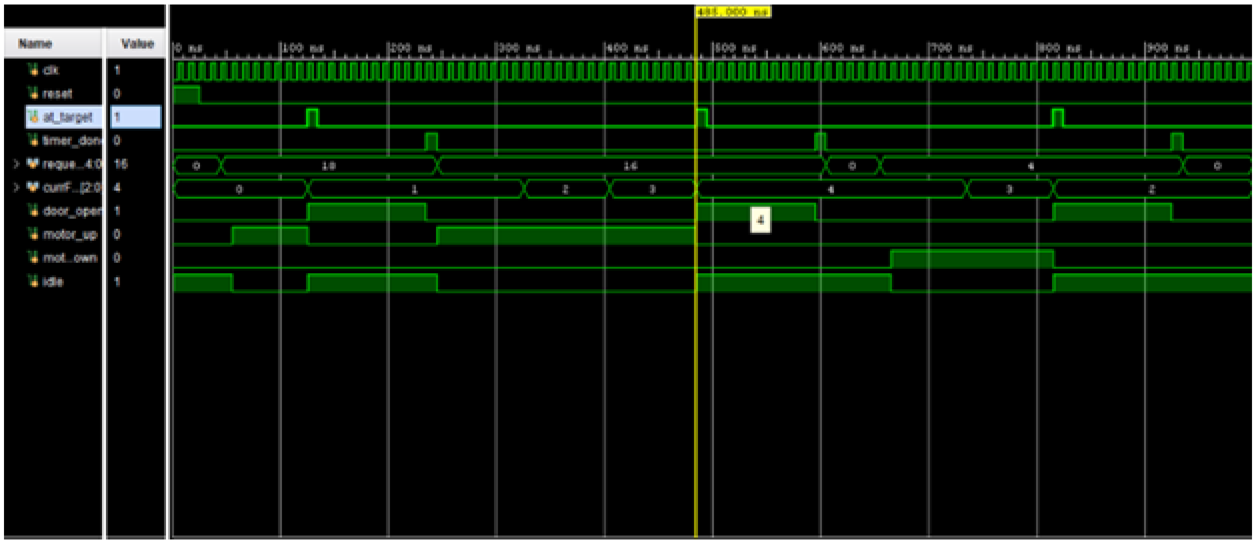


Fig 6: Elevator has reached floor 4 a while ago – motor_up is off, at_target pulse already finished. Still waiting for timer_done (door remains closed), idle signal is low, FSM is in the door-opening delay state.

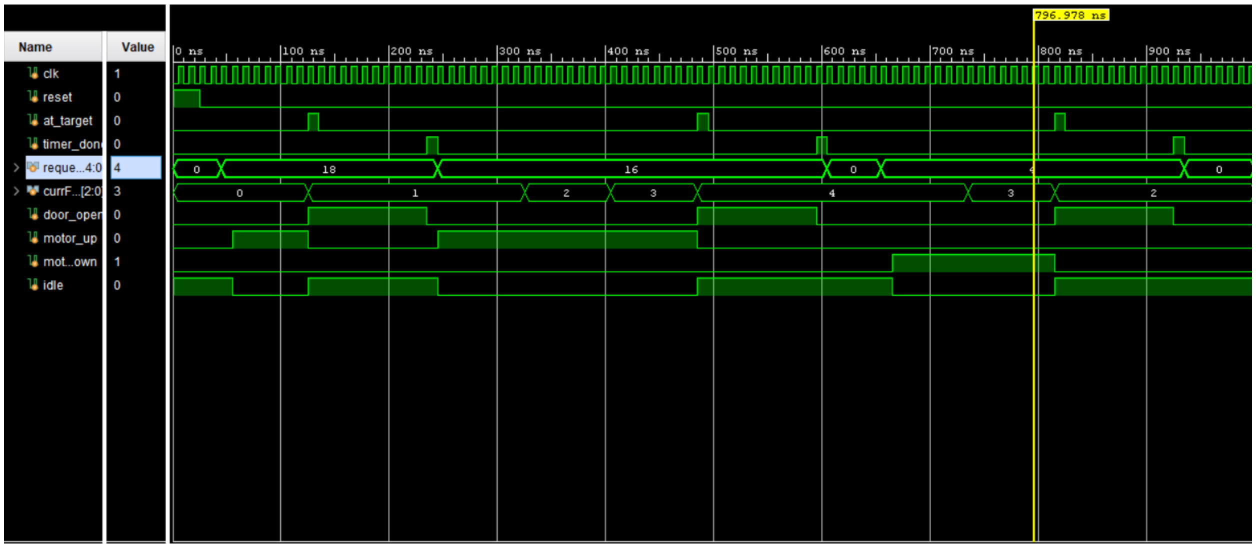


Fig 7: Elevator reached floor 4 quite a while ago motor_up is off, at_target pulse has already finished. Still waiting for timer_done (FSM is in door-opening delay state, curr=2) door remains closed and idle signal is low.

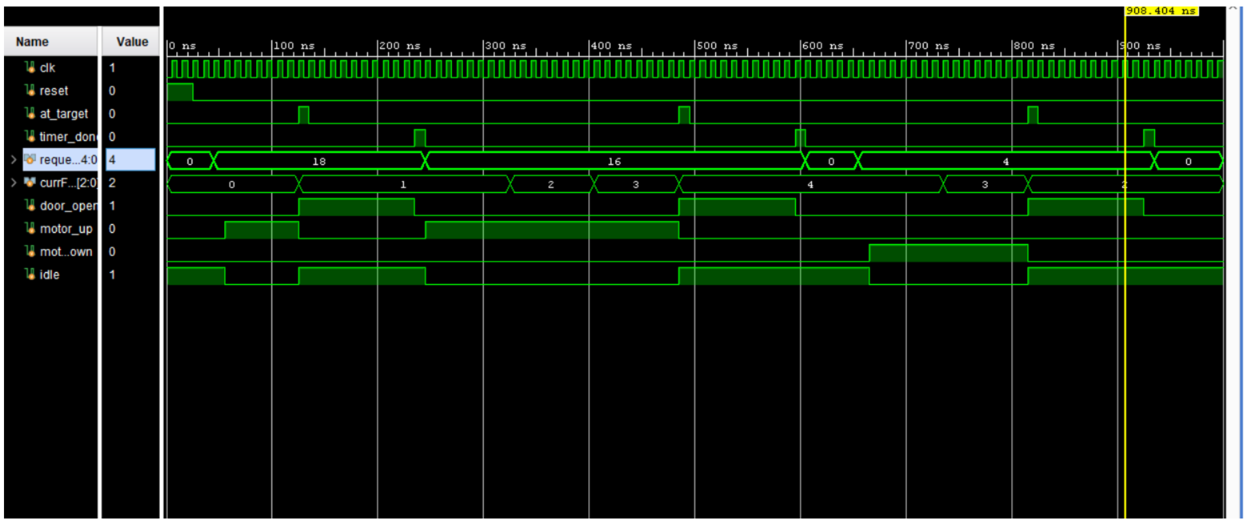
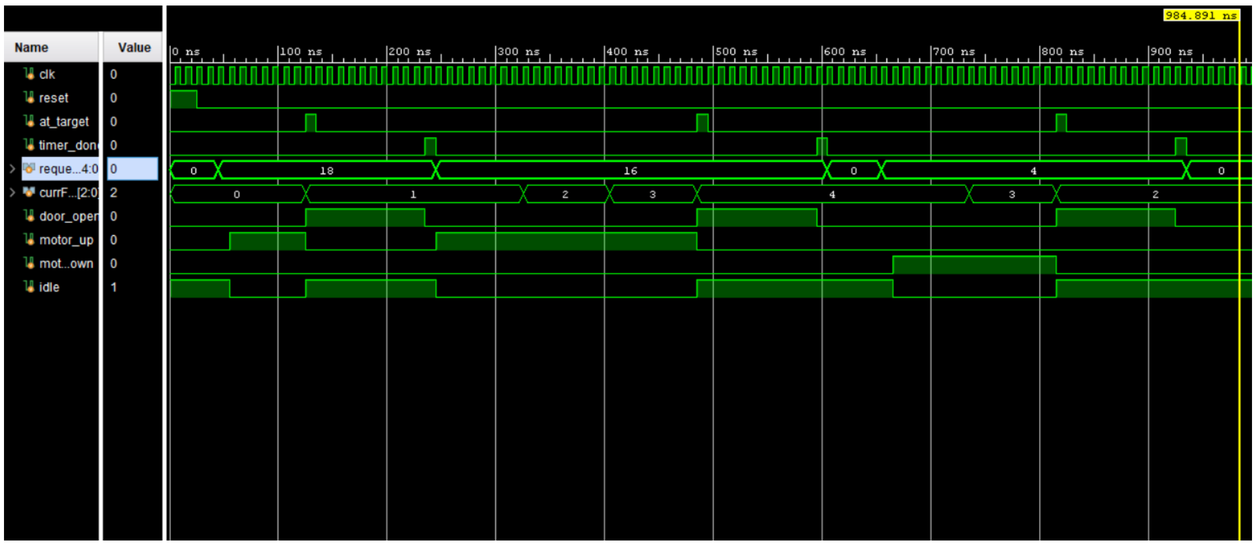


Fig 8: Now elevator is back to reset state where motor_up and motor_down all are off.



- Initial reset FSM in IDLE, idle=1.
- Multiple requests nearest request selected based on distance.
- FSM transitions reflect elevator movement:
    1. IDLE → GOING_UP → DOOR_OPEN → IDLE
    2. motor_up / motor_down asserted correctly.
    3. door_open asserted only at DOOR_OPEN state.
- Timer behavior: timer_done moves FSM back to IDLE.
- Moore behavior: outputs depend only on current state, not directly on inputs.

7. Conclusion & Future Work

7.1 Conclusion

The 5-floor elevator control system using a Moore FSM efficiently manages floor requests, motor control, and door operations. The modular design with Timer and Light Decoder ensures correct functionality, safe operation, and easy future hardware integration. Simulation results verify that the system works reliably under normal and emergency conditions.

7.2Future Work

1. Integration with Real Hardware: Implement the FSM on actual elevator motors, doors, and sensors using FPGA or microcontroller platforms.
2. Handling More Floors: Extend the system to support more than 5 floors with dynamic nearest-request calculations.
3. Emergency Features: Add support for fire alarms, power failures, and manual overrides to ensure safe operation.
4. Passenger Load Sensing: Integrate weight sensors to prevent overloading and adjust elevator operation accordingly.
5. Graphical User Interface (GUI): Develop a dashboard to display real-time elevator status, floor requests, and door states.

8. References

mailto:https://nevonprojects.com/digital-electronics-projects/

mailto:https://www.geeksforgeeks.org/system-design/elevator-system-low-level-design-lld/

mailto:https://www.scribd.com/document/491098241/Elevator-Control-system-using-Digital-Logic

9. Appendix — Complete Verilog Code

Below is the complete Verilog source code used in this project (FSM + testbench). Replace or parameterize timer limits before synthesizing on actual hardware.

DESIGN

```
`timescale 1ns / 1ps


module elevator_fsm_5floor (
    input  logic       clk,
    input  logic       reset,



    input  logic [4:0] requests,     // requests[0] = floor 0 ... requests[4] = floor 4
    input  logic [2:0] currFloor,
    input  logic       at_target,
    input  logic       timer_done,

    output logic       door_open,
    output logic       motor_up,
    output logic       motor_down,
    output logic       idle
);

    // ---------- Sequential encoded states ----------
    typedef enum logic [2:0] {
        IDLE        = 3'b00,
        GOING_UP    = 3'b01,
        GOING_DOWN  = 3'b10,
        DOOR_OPEN   = 3'b11
    } state_t;

    state_t cs, ns;

    // ---------- Request calculation ----------
    logic req_exists;
    logic [2:0] nearest_req;
    logic [2:0] distance, best_distance;

    always_comb begin
        req_exists = 0;
        best_distance = 7;
        nearest_req = currFloor;

        for (int i = 0; i < 5; i++) begin
            if (requests[i]) begin
                req_exists = 1;
                distance = (currFloor > i) ? (currFloor - i) : (i - currFloor);
                if (distance < best_distance) begin
                    best_distance = distance;
                    nearest_req = i[2:0];
                end
            end
        end
    end

    // ---------- Next state logic ----------
    always_comb begin
        ns = cs;

        case (cs)
            IDLE: begin
                if (reset) ns = IDLE;
                else if (req_exists) begin
                    if (nearest_req > currFloor) ns = GOING_UP;
                    else if (nearest_req < currFloor) ns = GOING_DOWN;
                    else ns = DOOR_OPEN;
                end
                else ns = IDLE;
            end

            GOING_UP: begin

                if (at_target) ns = DOOR_OPEN;
                else ns = GOING_UP;
            end

            GOING_DOWN: begin

                if (at_target) ns = DOOR_OPEN;
                else ns = GOING_DOWN;
            end

            DOOR_OPEN: begin

                if (timer_done) ns = IDLE;
                else ns = DOOR_OPEN;
            end

            default: ns = IDLE;
        endcase
    end

    // ---------- State register ----------
    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            cs <= IDLE;
            else
                cs <= ns;
        end

        // ---------- Output logic ----------
        assign door_open  = (cs == DOOR_OPEN);
        assign motor_up   = (cs == GOING_UP);
        assign motor_down = (cs == GOING_DOWN);
        assign idle       = (cs == IDLE || cs == DOOR_OPEN);

    endmodule
```

TESTBENCH

```
`timescale 1ns / 1ps

module elevator_fsm_5floor_tb;

    logic clk, reset, at_target, timer_done;
    logic [4:0] requests;
    logic [2:0] currFloor;
    logic door_open, motor_up, motor_down, idle;

    // DUT
    elevator_fsm_5floor dut (.*);
```

```
// Clock
initial clk = 0;
always #5 clk = ~clk;

// Main test
initial begin
   // ---------- Initialize ----------
   reset = 1;
   requests = 0;
   currFloor = 0;
   at_target = 0; timer_done = 0;

   #25 reset = 0;

   // ---------- Test 1: IDLE, request floors 1 & 4 ----------
   #20 requests[1] = 1;
       requests[4] = 1;

   // ---------- Move to floor 1 ----------
   #80 currFloor = 1; at_target = 1; #10 at_target = 0;
   #100 timer_done = 1; #10 timer_done = 0; requests[1] = 0;
```

```
        // ---------- Move to nearest floor 4 ----------

        #80 currFloor = 2;

        #80 currFloor = 3;

        #80 currFloor = 4; at_target = 1; #10 at_target = 0;

        #100 timer_done = 1; #10 timer_done = 0; requests[4] = 0;


        // ---------- Request floor 2 ----------

        #50 requests[2] = 1;


        // ---------- Elevator goes DOWN ----------

        #80 currFloor = 3;

        #80 currFloor = 2; at_target = 1; #10 at_target = 0;

        #100 timer_done = 1; #10 timer_done = 0; requests[2] = 0;

        #300;

        $display("Simulation done!");

        $finish;

    end

    initial $monitor("[t=%5t] Floor=%0d | Req=%b | State=%s | Door=%b | Up=%b | Down=%b |
Idle=%b",

                    $time, currFloor, requests, dut.cs.name(),

                    door_open, motor_up, motor_down, idle);


endmodule
```