Project Report– Digital System

MULTI-ELEVATOR SYSTEM

BACHELOR OF ENGINEERING

Degree In

Computer Systems Engineering

Group Members:
Muhammad Usman
(CMS ID: 133-22-0016)
Asma Channa
(CMS ID: 133-22-0042)

Guided by:
Dr: Kashif Hussain,

Department of Computer Systems Engineering
Sukkur IBA University

TABLE OF CONTENTS

1. ABSTRACT

Modern multi-storey buildings rely heavily on efficient elevator systems to manage vertical transportation. As passenger demand increases, the performance and intelligence of elevator controllers become crucial for minimizing waiting time, reducing congestion, and optimizing energy consumption.
This project presents the design and implementation of a multi-elevator control system using Finite State Machines (FSMs) and Verilog HDL. The system handles internal cabin requests, floor-level up/down requests, door timing, movement timing, and dynamic scheduling using the Nearest Car Algorithm.
Two elevators serving six floors are implemented, each equipped with an individual FSM controller. A centralized Request Manager assigns floor requests to elevators based on distance, direction, and current state. The complete system is simulated in Xilinx Vivado, and the behavior is validated through extensive waveform analysis. The results show that the designed system performs stable movement, correct scheduling, reliable door operation, and accurate stopping at requested floors, achieving an efficient and scalable multi-elevator control solution.

2. PROBLEM DEFINITION & OBJECTIVES

2.1 Problem Definition

In multi-floor buildings, the distribution of elevator requests poses several challenges:

- Handling simultaneous up/down requests across different floors
- Assigning these requests to the most suitable elevator
- Ensuring minimal waiting time for passengers
- Controlling door opening and closing transitions
- Maintaining safe and efficient elevator movement

Traditional single-elevator controllers fail to optimize performance in buildings with high user traffic. Therefore, an intelligent and modular multi-elevator controller is required.

2.2 Objectives

The objectives of this project are:

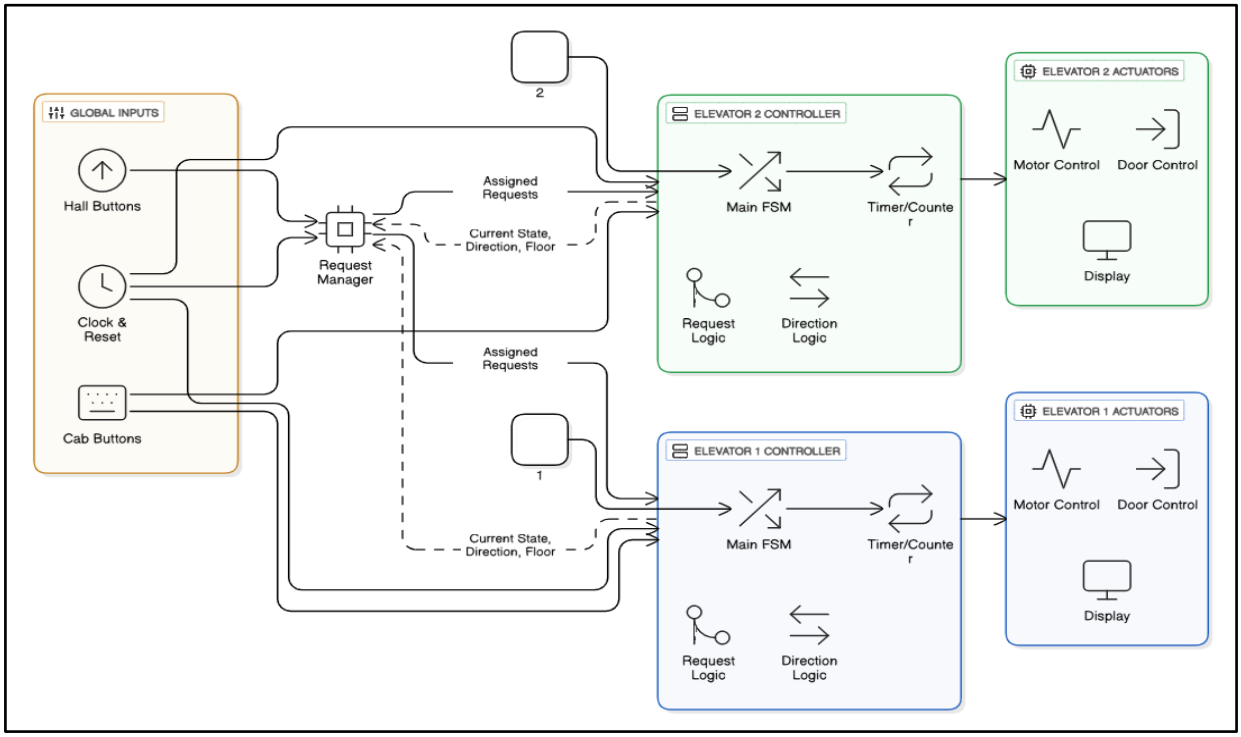
- To design a modular multi-elevator control system using Verilog HDL.
- To implement an FSM-based elevator controller capable of movement, stopping, and door logic.
- To develop a Request Manager that intelligently assigns floor requests to the nearest elevator.
- To validate the system using testbenches and simulation waveforms.
- To ensure the design is scalable, accurate, and synthesizable for FPGA implementation.

3. BLOCK DIAGRAM & DESIGN METHODOLOGY

3.1 Block Diagram Overview

The complete system consists of three major modules:

- elevator controller (fsm) controls movement, door, and request servicing.
- request manager assigns external up/down requests to the ideal elevator.
- top-level multi-elevator system integrates both elevator fsms and request manager.



3.2 Design Methodology

3.2.1 Elevator FSM (Mealy-style FSM)

Each elevator uses a 5-state FSM:

State	Binary	Description
IDLE	000	No movement, no active requests
MOVING	001	Elevator moving up/down
DOOR_OPENING	010	Door transitioning from closed to open
DOOR_OPEN	011	Door open for passengers
DOOR_CLOSING	100	Door closing after service

The FSM controls:

- Movement between floors
- Door opening and closing
- Stopping logic at requested floors
- Timer-dependent events

3.2.2 Timer Module (Integrated in FSM)

Two internal timers are used:

1. **MOVE_DELAY:** time required to move one floor
2. **DOOR_OPEN_TIME:** duration for which door remains open

Timer functions:

- Regulate movement speed
- Handle door transitions
- Control FSM timing-based state switches

3.2.3 Request Manager Module

The Request Manager:

- Collects floor up/down button presses
- Stores them in `pending_up` and `pending_down` registers
- Uses Nearest Car Algorithm for assignment
- Evaluates distance, direction, and idle status
- Assigns the request to the optimal elevator

3.3 Modular Design Advantages

- Easily extendable to more elevator
- Each module independently testable
- Clean hierarchical structure
- Reduced complexity due to FSM partitioning
- Scheduling logic isolated from movement logic

4. TRUTH TABLES & STATE DIAGRAMS

4.1 State Transition Table

Current State	Condition	Next State
IDLE	Active request at current floor	DOOR_OPENING
IDLE	Any request in building	MOVING
MOVING	Timer expired & should_stop==1	DOOR_OPENING
DOOR_OPENING	Timer ≥ DOOR_TRANSITION	DOOR_OPEN
DOOR_OPEN	Timer ≥ DOOR_OPEN_TIME	DOOR_CLOSING
DOOR_CLOSING	Active requests pending	MOVING
DOOR_CLOSING	No requests pending	IDLE

4.2 State Encoding

State	Symbol	Binary
IDLE	S0	000
MOVING	S1	001
DOOR_OPENING	S2	010
DOOR_OPEN	S3	011
DOOR_CLOSING	S4	100

4.3 Next-State Logic Table

Extracted from your Verilog FSM:

Current State	Input Signals	Next State
IDLE	next_active_req==1	MOVING/DOOR_OPENING
MOVING	timer==0 & should_stop==1	DOOR_OPENING
DOOR_OPENING	timer>=DOOR_TRANSITION-1	DOOR_OPEN
DOOR_OPEN	timer>=DOOR_OPEN_TIME-1	DOOR_CLOSING
DOOR_CLOSING	pending req?	MOVING

Current State	Input Signals	Next State
DOOR_CLOSING	no pending req	IDLE

4.4 Next-State Logic Equations

FSM symbolic terms:

- $S2^+$ = Transition to DOOR_OPEN
- $S1^+$ = Transition to MOVING
- $S0^+$ = Transition to IDLE

S_2^+ (Next State Bit 2):

$$S_2^+ = \bar{S}_2 \cdot S_1 \cdot S_0 \cdot T$$

Explanation: Goes high only when in DOOR_OPEN (011) and timer done

S_1^+ (Next State Bit 1):

$$S_1^+ = \bar{S}_2 \cdot \bar{S}_1 \cdot \bar{S}_0 \cdot R \cdot C + \bar{S}_2 \cdot \bar{S}_1 \cdot S_0 \cdot T \cdot S + \bar{S}_2 \cdot S_1 \cdot \bar{S}_0 \cdot \bar{T} + \bar{S}_2 \cdot S_1 \cdot S_0 \cdot \bar{T}$$

Simplified: $S_1^+ = \bar{S}_2 \cdot \bar{S}_1 \cdot \bar{S}_0 \cdot R \cdot C + \bar{S}_2 \cdot \bar{S}_1 \cdot S_0 \cdot T \cdot S + \bar{S}_2 \cdot S_1 \cdot (\bar{T} + \bar{S}_0 \cdot \bar{T})$

S_0^+ (Next State Bit 0):

$$S_0^+ = \bar{S}_2 \cdot \bar{S}_1 \cdot \bar{S}_0 \cdot R \cdot \bar{C} + \bar{S}_2 \cdot \bar{S}_1 \cdot S_0 \cdot (\bar{T} + \bar{S}) + \bar{S}_2 \cdot S_1 \cdot \bar{S}_0 \cdot T + \bar{S}_2 \cdot S_1 \cdot S_0 \cdot \bar{T} + S_2 \cdot \bar{S}_1 \cdot \bar{S}_0 \cdot T \cdot R$$

Explanation: Complex equation handling transitions to MOVING and DOOR_OPEN states

4.5 Output Logic Table

Output	Active In States
door_open	DOOR_OPEN
moving	MOVING
requests_served	DOOR_OPEN

4.6 Output Encoding

Output	Description
moving = 1	Elevator is currently moving

Output	Description
door_open = 1	Door fully open
direction	00: idle, 01: up, 10: down

4.7 Output Logic Equations

Based on output logic equations:

- **door_open = S3**
- **moving = S1**
- **requests_served = S3 · S0**

door_open (D):

$$D = \bar{S}_2 \cdot S_1 \cdot S_0 + S_2 \cdot \bar{S}_1 \cdot \bar{S}_0$$

Door is open in DOOR_OPEN (011) and DOOR_CLOSING (100) states

moving (M):

$$M = \bar{S}_2 \cdot \bar{S}_1 \cdot S_0$$

Elevator is moving only in MOVING (001) state

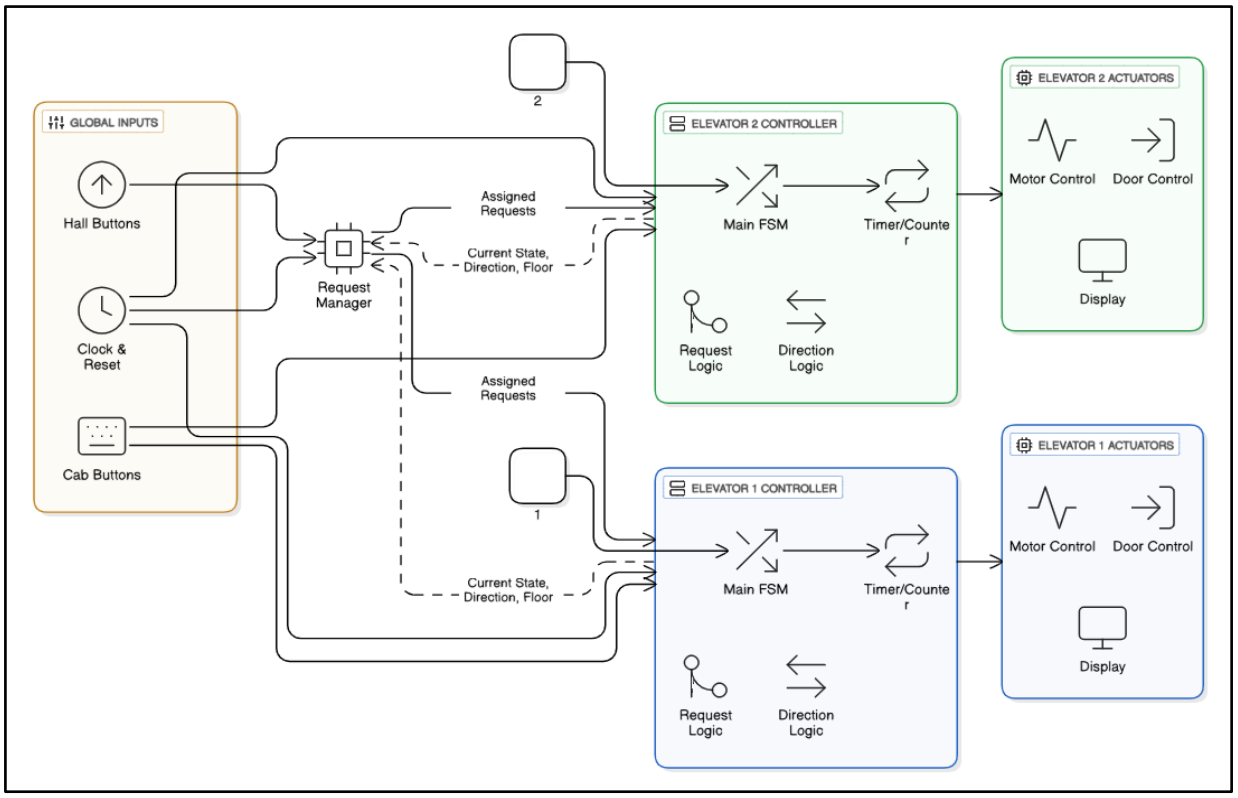
requests_served (RS):

$$RS = \bar{S}_2 \cdot S_1 \cdot S_0$$

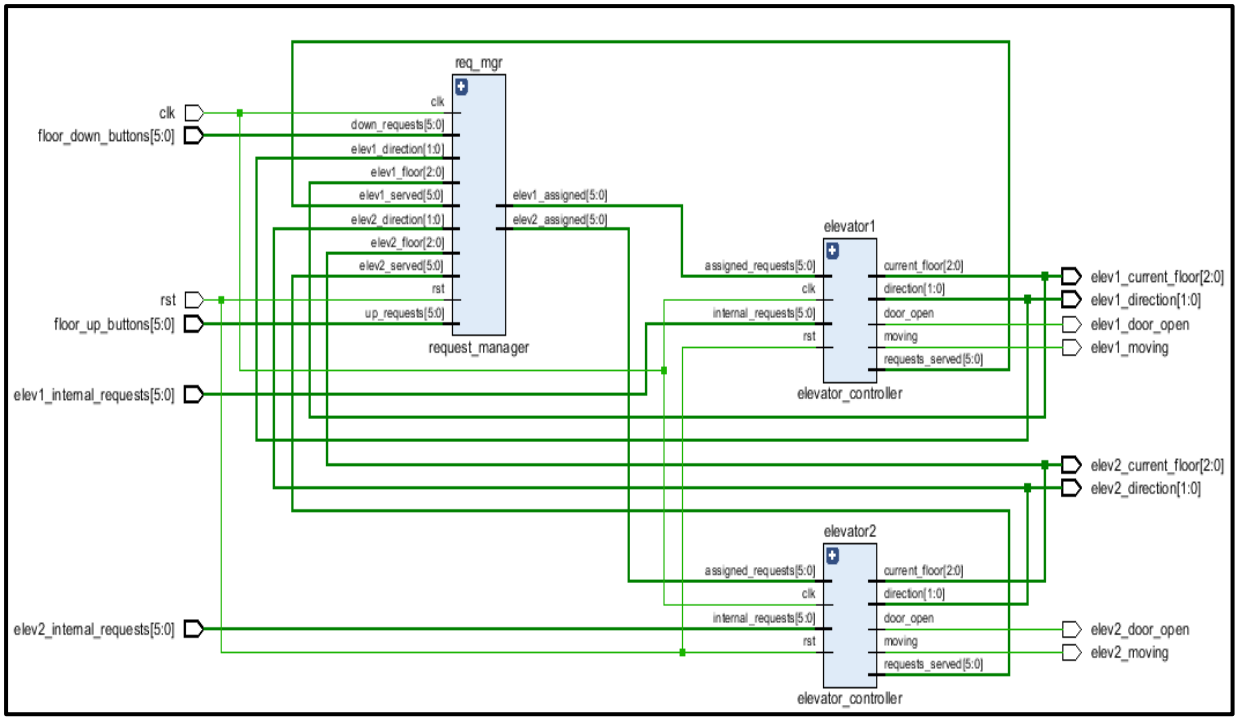
Request served signal active in DOOR_OPEN (011) state

5. SCHEMATIC DIAGRAM

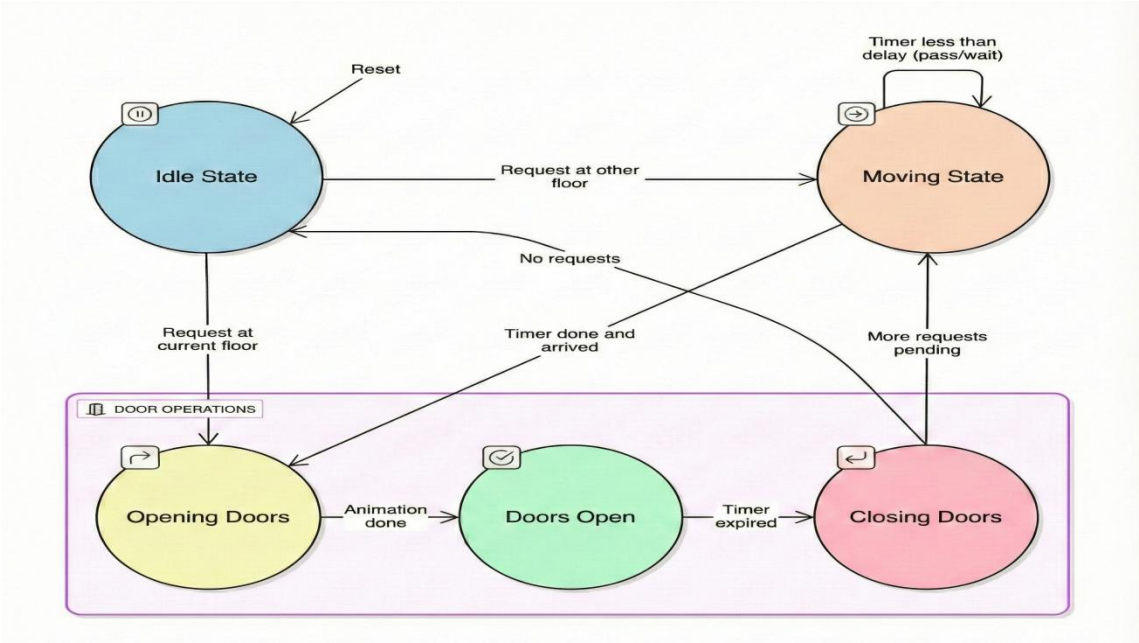
5.1 Block Diagram



5.2 System Architecture: Multi-Elevator Control



5.3 FSM State Diagram



Idle State	Open_door=0, Moving=0, request_served=0
Moving State	Open_door=0, Moving=1, request_served=0
Opening Doors	Open_door=0, Moving=0, request_served=0
Doors Open	Open_door=1, Moving=0, request_served=1
Closing Doors	Open_door=1, Moving=0, request_served=0

6. IMPLEMENTATION IN VERILOG HDL

6.1 Elevator FSM Module

```
module elevator_controller #(
    parameter NUM_FLOORS = 6,
    parameter FLOOR_BITS = 3,
    parameter MOVE_DELAY = 10,
    parameter DOOR_OPEN_TIME = 5,
     parameter DOOR_TRANSITION = 3
)(
    input wire clk,
    input wire rst,
    input wire [NUM_FLOORS-1:0] internal_requests,
    input wire [NUM_FLOORS-1:0] assigned_requests,
    output reg [FLOOR_BITS-1:0] current_floor,
    output reg [1:0] direction,
    output reg door_open,
    output reg moving,
    output reg [NUM_FLOORS-1:0] requests_served
);

    // State definitions
    localparam [2:0]
        IDLE          = 3'b000,
        MOVING        = 3'b001,
        DOOR_OPENING  = 3'b010,
        DOOR_OPEN     = 3'b011,
        DOOR_CLOSING  = 3'b100;

    // Direction definitions
    localparam [1:0]
        DIR_IDLE = 2'b00,
        DIR_UP   = 2'b01,
        DIR_DOWN = 2'b10;

    // Internal registers
    reg [2:0] state, next_state;
    reg [NUM_FLOORS-1:0] active_requests;
    reg [NUM_FLOORS-1:0] next_active_requests;
    reg [5:0] timer;
    reg [1:0] next_direction;

    // Helper signals for request checking
    wire has_req_above;
    wire has_req_below;
    wire should_stop;

    // Request management
    always @(*) begin
        next_active_requests = active_requests | internal_requests | assigned_requests;

        if (state == DOOR_OPEN && active_requests[current_floor])
            next_active_requests[current_floor] = 1'b0;
    end

    // Check for requests above current floor
    assign has_req_above = (current_floor < 5 && next_active_requests[5]) ||
                           (current_floor < 4 && next_active_requests[4]) ||
                           (current_floor < 3 && next_active_requests[3]) ||
                           (current_floor < 2 && next_active_requests[2]) ||
                           (current_floor < 1 && next_active_requests[1]);

    // Check for requests below current floor
    assign has_req_below = (current_floor > 0 && next_active_requests[0]) ||
                           (current_floor > 1 && next_active_requests[1]) ||
                           (current_floor > 2 && next_active_requests[2]) ||
                           (current_floor > 3 && next_active_requests[3]) ||
                           (current_floor > 4 && next_active_requests[4]);

    // Check if should stop at current floor
    assign should_stop = next_active_requests[current_floor] &&
                         ((direction == DIR_UP && !has_req_above) ||
                          (direction == DIR_DOWN && !has_req_below) ||
                          (direction == DIR_UP) ||
                          (direction == DIR_DOWN));

    // Direction decision logic
    always @(*) begin
        next_direction = direction;

        case (state)
            IDLE: begin
                if (|next_active_requests) begin
                    if (next_active_requests[current_floor])
                        next_direction = DIR_IDLE;
                    else if (has_req_above)
```

```
                    next_direction = DIR_UP;
                else
                    next_direction = DIR_DOWN;
            end else begin
                next_direction = DIR_IDLE;
            end
        end

        MOVING: begin
            if (direction == DIR_UP) begin
                if (!has_req_above) begin
                    if (has_req_below)
                        next_direction = DIR_DOWN;
                    else
                        next_direction = DIR_IDLE;
                end
            end else if (direction == DIR_DOWN) begin
                if (!has_req_below) begin
                    if (has_req_above)
                        next_direction = DIR_UP;
                    else
                        next_direction = DIR_IDLE;
                end
            end
        end

        DOOR_CLOSING: begin
            if (has_req_above)
                next_direction = DIR_UP;
            else if (has_req_below)
                next_direction = DIR_DOWN;
            else
                next_direction = DIR_IDLE;
        end

        default: next_direction = direction;
    endcase
end

// Sequential logic
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state <= IDLE;
        current_floor <= 0;
        direction <= DIR_IDLE;
        door_open <= 0;
        moving <= 0;
        active_requests <= 0;
        timer <= 0;
        requests_served <= 0;
    end else begin
        state <= next_state;
        active_requests <= next_active_requests;
        direction <= next_direction;

        case (state)
            IDLE: begin
                door_open <= 0;
                moving <= 0;
                requests_served <= 0;
                timer <= 0;
            end

            MOVING: begin
                moving <= 1;
                door_open <= 0;

                if (timer < MOVE_DELAY) begin
                    timer <= timer + 1;
                end else begin
                    timer <= 0;
                    if (direction == DIR_UP && current_floor < NUM_FLOORS-1)
                        current_floor <= current_floor + 1;
                    else if (direction == DIR_DOWN && current_floor > 0)
                        current_floor <= current_floor - 1;
                end
            end

            DOOR_OPENING: begin
                moving <= 0;
                if (timer < DOOR_TRANSITION) begin
                    timer <= timer + 1;
                    door_open <= 0;
                end else begin
                    timer <= 0;
                    door_open <= 1;
                end
            end

            DOOR_OPEN: begin
```

```
                    door_open <= 1;
                    moving <= 0;
                    requests_served[current_floor] <= 1;

                    if (timer < DOOR_OPEN_TIME)
                        timer <= timer + 1;
                    else
                        timer <= 0;
                end

                DOOR_CLOSING: begin
                    moving <= 0;
                    requests_served <= 0;
                    if (timer < DOOR_TRANSITION) begin
                        timer <= timer + 1;
                    end else begin
                        timer <= 0;
                        door_open <= 0;
                    end
                end

                default: begin
                    door_open <= 0;
                    moving <= 0;
                    requests_served <= 0;
                    timer <= 0;
                end
            endcase
        end
    end

    // Next state logic
    always @(*) begin
        next_state = state;

        case (state)
            IDLE: begin
                if (|next_active_requests) begin
                    if (next_active_requests[current_floor])
                        next_state = DOOR_OPENING;
                    else
                        next_state = MOVING;
                end
            end

            MOVING: begin
                if (timer == 0 && should_stop) begin
                    next_state = DOOR_OPENING;
                end
            end

            DOOR_OPENING: begin
                if (timer >= DOOR_TRANSITION - 1)
                    next_state = DOOR_OPEN;
            end

            DOOR_OPEN: begin
                if (timer >= DOOR_OPEN_TIME - 1)
                    next_state = DOOR_CLOSING;
            end

            DOOR_CLOSING: begin
                if (timer >= DOOR_TRANSITION - 1) begin
                    if (|next_active_requests)
                        next_state = MOVING;
                    else
                        next_state = IDLE;
                end
            end
            default: next_state = IDLE;
        endcase
    end


endmodule
```

6.2 Request Manager Module

```
module request_manager #(
    parameter NUM_FLOORS = 6,
    parameter NUM_ELEVATORS = 2,
    parameter FLOOR_BITS = 3
)(
    input wire clk,
    input wire rst,
    input wire [NUM_FLOORS-1:0] up_requests,
    input wire [NUM_FLOORS-1:0] down_requests,
```

```
    input wire [FLOOR_BITS-1:0] elev1_floor,
    input wire [FLOOR_BITS-1:0] elev2_floor,
    input wire [1:0] elev1_direction,
    input wire [1:0] elev2_direction,
    input wire [NUM_FLOORS-1:0] elev1_served,
    input wire [NUM_FLOORS-1:0] elev2_served,
    output reg [NUM_FLOORS-1:0] elev1_assigned,
    output reg [NUM_FLOORS-1:0] elev2_assigned
);

    reg [NUM_FLOORS-1:0] pending_up;
    reg [NUM_FLOORS-1:0] pending_down;

    integer i;

    // Register and clear requests
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            pending_up <= 0;
            pending_down <= 0;
        end else begin
            pending_up <= (pending_up | up_requests) & ~(elev1_served | elev2_served);
            pending_down <= (pending_down | down_requests) & ~(elev1_served | elev2_served);
        end
    end

    // Assignment logic - Nearest Car Algorithm
    always @(*) begin
        elev1_assigned = 0;
        elev2_assigned = 0;

        for (i = 0; i < NUM_FLOORS; i = i + 1) begin
            if (pending_up[i] || pending_down[i]) begin
                if (distance_score(i, elev1_floor, elev1_direction) <=
                    distance_score(i, elev2_floor, elev2_direction)) begin
                    elev1_assigned[i] = 1;
                end else begin
                    elev2_assigned[i] = 1;
                end
            end
        end
    end

    // Distance score calculation
    function [7:0] distance_score;
        input [FLOOR_BITS-1:0] request_floor;
        input [FLOOR_BITS-1:0] elevator_floor;
        input [1:0] elevator_dir;
        reg [7:0] distance;
        begin
            distance = (request_floor > elevator_floor) ?
                       (request_floor - elevator_floor) :
                       (elevator_floor - request_floor);

            if (elevator_dir == 2'b01 && request_floor < elevator_floor)
                distance = distance + 50;
            else if (elevator_dir == 2'b10 && request_floor > elevator_floor)
                distance = distance + 50;

            distance_score = distance;
        end
    endfunction

endmodule
```

6.3 Top-Level Multi-Elevator System

```
module multi_elevator_system #(
    parameter NUM_FLOORS = 6,
    parameter NUM_ELEVATORS = 2,
    parameter FLOOR_BITS = 3
)(
    input wire clk,
    input wire rst,
    input wire [NUM_FLOORS-1:0] floor_up_buttons,
    input wire [NUM_FLOORS-1:0] floor_down_buttons,
    input wire [NUM_FLOORS-1:0] elev1_internal_requests,
    input wire [NUM_FLOORS-1:0] elev2_internal_requests,
    output wire [FLOOR_BITS-1:0] elev1_current_floor,
    output wire [FLOOR_BITS-1:0] elev2_current_floor,
    output wire [1:0] elev1_direction,
    output wire [1:0] elev2_direction,
    output wire elev1_door_open,
    output wire elev2_door_open,
    output wire elev1_moving,
    output wire elev2_moving
);
```

```
    wire [NUM_FLOORS-1:0] elev1_assigned;
    wire [NUM_FLOORS-1:0] elev2_assigned;
    wire [NUM_FLOORS-1:0] elev1_served;
    wire [NUM_FLOORS-1:0] elev2_served;

    // Request Manager
    request_manager #(
        .NUM_FLOORS(NUM_FLOORS),
        .NUM_ELEVATORS(NUM_ELEVATORS),
        .FLOOR_BITS(FLOOR_BITS)
    ) req_mgr (
        .clk(clk),
        .rst(rst),
        .up_requests(floor_up_buttons),
        .down_requests(floor_down_buttons),
        .elev1_floor(elev1_current_floor),
        .elev2_floor(elev2_current_floor),
        .elev1_direction(elev1_direction),
        .elev2_direction(elev2_direction),
        .elev1_served(elev1_served),
        .elev2_served(elev2_served),
        .elev1_assigned(elev1_assigned),
        .elev2_assigned(elev2_assigned)
    );

    // Elevator 1 Controller
    elevator_controller #(
        .NUM_FLOORS(NUM_FLOORS),
        .FLOOR_BITS(FLOOR_BITS)
    ) elevator1 (
        .clk(clk),
        .rst(rst),
        .internal_requests(elev1_internal_requests),
        .assigned_requests(elev1_assigned),
        .current_floor(elev1_current_floor),
        .direction(elev1_direction),
        .door_open(elev1_door_open),
        .moving(elev1_moving),
        .requests_served(elev1_served)
    );

    // Elevator 2 Controller
    elevator_controller #(
        .NUM_FLOORS(NUM_FLOORS),
        .FLOOR_BITS(FLOOR_BITS)
    ) elevator2 (
        .clk(clk),
        .rst(rst),
        .internal_requests(elev2_internal_requests),
        .assigned_requests(elev2_assigned),
        .current_floor(elev2_current_floor),
        .direction(elev2_direction),
        .door_open(elev2_door_open),
        .moving(elev2_moving),
        .requests_served(elev2_served)
    );

endmodule
```

6.4 Testbench for Single Elevator

```
`timescale 1ns / 1ps
module tb_single_elevator;
    reg clk, rst;
    reg [5:0] internal_req;
    reg [5:0] assigned_req;
    wire [2:0] current_floor;
    wire [1:0] direction;
    wire door_open;
    wire moving;
    wire [5:0] req_served;

    elevator_controller #(.NUM_FLOORS(6)) uut (
        .clk(clk),
        .rst(rst),
        .internal_requests(internal_req),
        .assigned_requests(assigned_req),
        .current_floor(current_floor),
        .direction(direction),
        .door_open(door_open),
        .moving(moving),
        .requests_served(req_served)
    );

    // Clock generation - 10ns period
    initial begin
```

```
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Test stimulus
    initial begin
        $display("=======================================");
        $display("Test: Single Elevator Basic Operations");
        $display("=======================================");

        // Initialize
        rst = 1;
        internal_req = 6'b000000;
        assigned_req = 6'b000000;
        #20 rst = 0;
        #50;

        // Test 1: Request floor 3 from floor 0
        $display("\n[TEST 1] Time=%0t: Requesting floor 3", $time);
        internal_req[3] = 1;
        #10 internal_req[3] = 0;

        // Wait for elevator to arrive
        wait(current_floor == 3 && door_open == 1);
        $display("[TEST 1] Time=%0t: SUCCESS - Arrived at floor 3, door open", $time);

        // Wait for door to close
        wait(door_open == 0);
        $display("[TEST 1] Time=%0t: Door closed", $time);

        #100;

        // Test 2: Request floor 1 (going down)
        $display("\n[TEST 2] Time=%0t: Requesting floor 1", $time);
        internal_req[1] = 1;
        #10 internal_req[1] = 0;

        wait(current_floor == 1 && door_open == 1);
        $display("[TEST 2] Time=%0t: SUCCESS - Arrived at floor 1, door open", $time);

        #200;

        $display("\n=======================================");
        $display("ALL TESTS PASSED!");
        $display("=======================================");
        $finish;
    end

    // Monitor signals
    initial begin
        $monitor("Time=%0t | Floor=%0d | Dir=%0d | Door=%b | Moving=%b | State=%b",
                 $time, current_floor, direction, door_open, moving, uut.state);
    end

endmodule
```

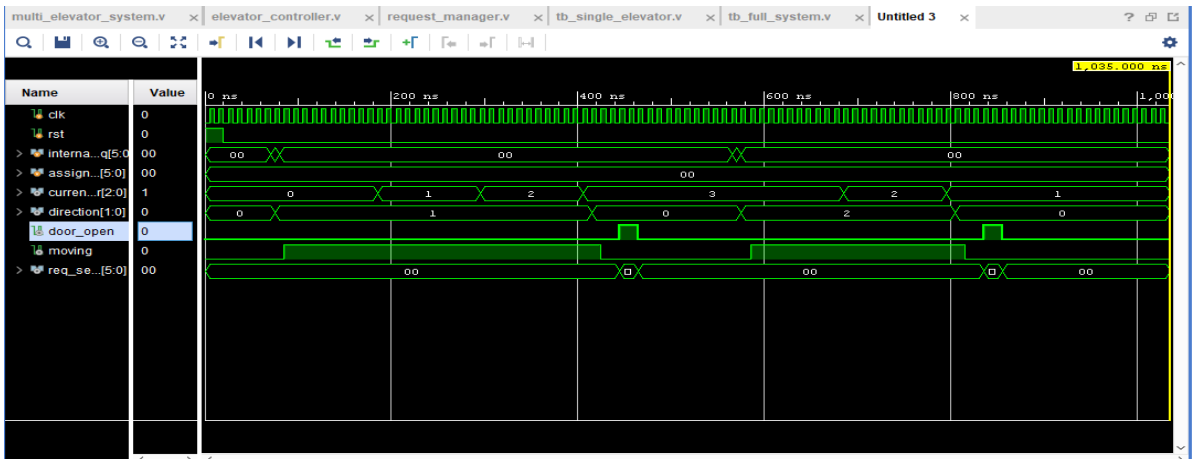
6.5 Testbench for Multi-Elevator System

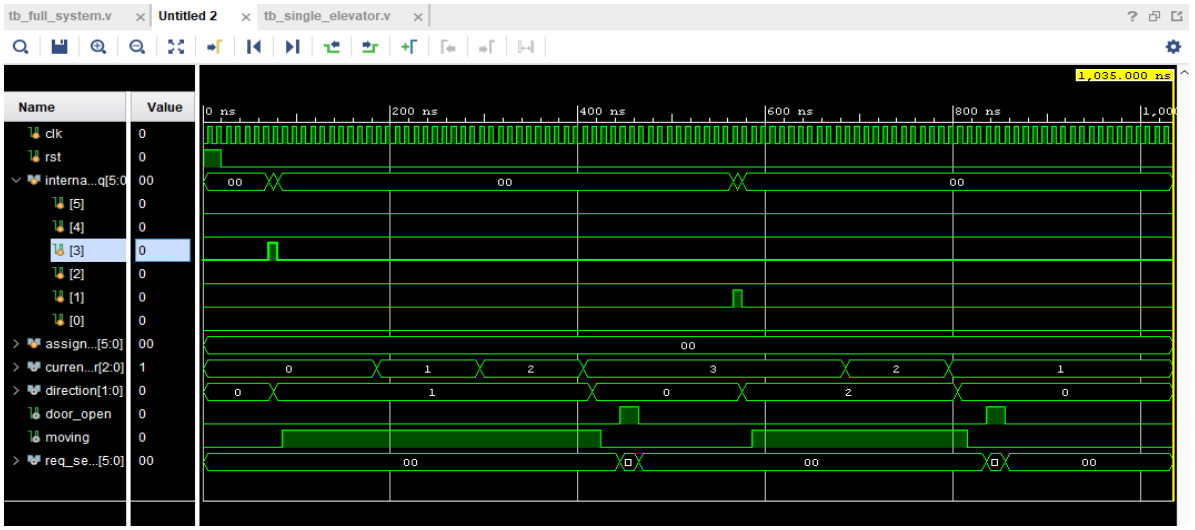
```
`timescale 1ns / 1ps
module tb_full_system;
    reg clk, rst;
    reg [5:0] up_buttons;
    reg [5:0] down_buttons;
    reg [5:0] elev1_internal;
    reg [5:0] elev2_internal;

    wire [2:0] elev1_floor;
    wire [2:0] elev2_floor;
    wire [1:0] elev1_dir;
    wire [1:0] elev2_dir;
    wire elev1_door;
    wire elev2_door;
    wire elev1_moving;
    wire elev2_moving;

    // Instantiate the system
    multi_elevator_system #(.NUM_FLOORS(6)) system (
        .clk(clk),
        .rst(rst),
        .floor_up_buttons(up_buttons),
        .floor_down_buttons(down_buttons),
        .elev1_internal_requests(elev1_internal),
        .elev2_internal_requests(elev2_internal),
        .elev1_current_floor(elev1_floor),
        .elev2_current_floor(elev2_floor),
        .elev1_direction(elev1_dir),
        .elev2_direction(elev2_dir),
        .elev1_door_open(elev1_door),
```

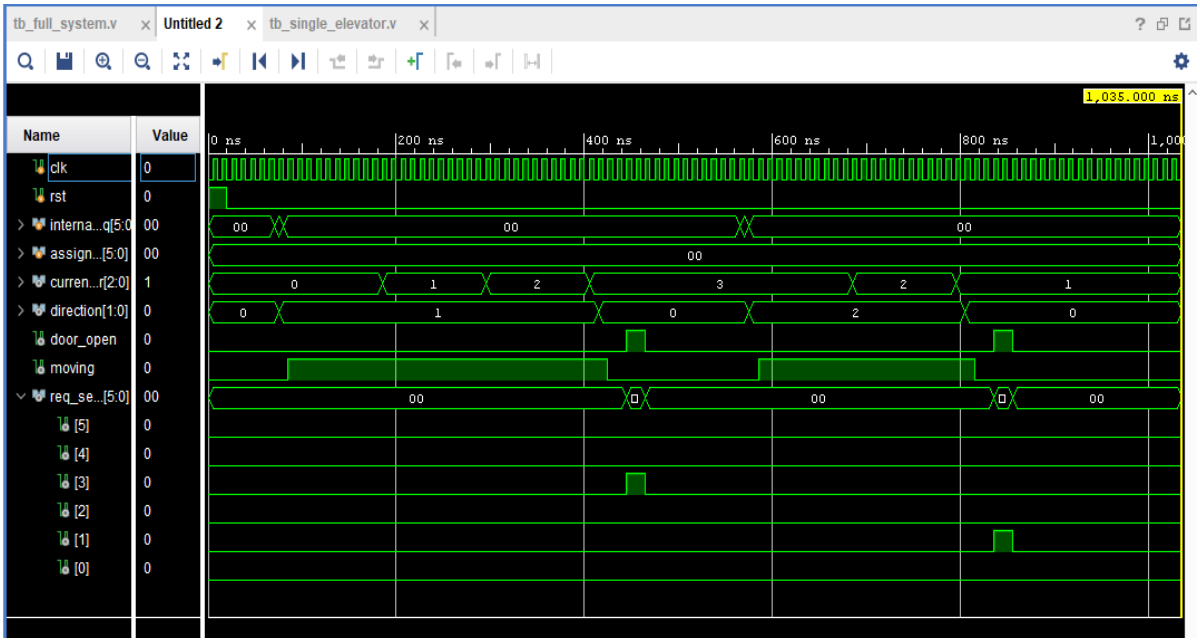
```
        .elev2_door_open(elev2_door),
        .elev1_moving(elev1_moving),
        .elev2_moving(elev2_moving)
    );

    // Clock generation - 10ns period (100MHz)
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Test stimulus
    initial begin
        $display("=======================================");
        $display("Test: Full Multi-Elevator System");
        $display("=======================================");

        // Initialize ALL inputs to zero
        rst = 1;
        up_buttons = 6'b000000;
        down_buttons = 6'b000000;
        elev1_internal = 6'b000000;
        elev2_internal = 6'b000000;

        // Apply reset for sufficient time
        #50;
        rst = 0;
        #50;

        $display("\n[TIME=%0t] System initialized. Both elevators at floor 0.", $time);
        $display("[TIME=%0t] Elev1 Floor=%0d | Elev2 Floor=%0d", $time, elev1_floor, elev2_floor);

        // Test 1: Person at floor 0 presses UP
        #100;
        $display("\n[TEST 1] Time=%0t: Person at floor 0 presses UP button", $time);
        up_buttons[0] = 1;
        #20;
        up_buttons[0] = 0;

        // Wait for assignment
        #50;
        $display("[TEST 1] Time=%0t: Request assigned. Elev1_Floor=%0d, Elev2_Floor=%0d",
                 $time, elev1_floor, elev2_floor);

        // Wait for elevator to arrive (should already be there at floor 0)
        wait(elev1_door == 1 || elev2_door == 1);
        #10;

        if (elev1_door == 1) begin
            $display("[TEST 1] Time=%0t: Elevator 1 door opened at floor %0d", $time, elev1_floor);
            $display("[TEST 1] Time=%0t: Passenger enters Elevator 1 and presses floor 3", $time);
            elev1_internal[3] = 1;
            #20;
            elev1_internal[3] = 0;
        end else if (elev2_door == 1) begin
            $display("[TEST 1] Time=%0t: Elevator 2 door opened at floor %0d", $time, elev2_floor);
            $display("[TEST 1] Time=%0t: Passenger enters Elevator 2 and presses floor 3", $time);
            elev2_internal[3] = 1;
            #20;
            elev2_internal[3] = 0;
        end

        // Wait for elevator to reach floor 3
        #600;
        $display("[TEST 1] Time=%0t: Status - Elev1_Floor=%0d, Elev2_Floor=%0d",
                 $time, elev1_floor, elev2_floor);

        // Test 2: Person at floor 5 presses DOWN
        #200;
        $display("\n[TEST 2] Time=%0t: Person at floor 5 presses DOWN button", $time);
        down_buttons[5] = 1;
        #20;
        down_buttons[5] = 0;

        // Wait for nearest elevator to respond
        #800;
        $display("[TEST 2] Time=%0t: Status - Elev1_Floor=%0d, Elev2_Floor=%0d",
                 $time, elev1_floor, elev2_floor);

        // Test 3: Multiple simultaneous requests
        #200;
        $display("\n[TEST 3] Time=%0t: Multiple people press buttons (floors 1, 2, 4)", $time);
        up_buttons[1] = 1;
        up_buttons[2] = 1;
        down_buttons[4] = 1;
        #20;
        up_buttons[1] = 0;
        up_buttons[2] = 0;
        down_buttons[4] = 0;

        // Let system handle multiple requests
```

```
        #1500;
        $display("[TEST 3] Time=%0t: Status - Elev1_Floor=%0d, Elev2_Floor=%0d",
                 $time, elev1_floor, elev2_floor);

        // Final status
        #500;
        $display("\n=======================================");
        $display("FINAL STATUS:");
        $display("Elevator 1: Floor=%0d, Direction=%0d, Door=%b, Moving=%b",
                 elev1_floor, elev1_dir, elev1_door, elev1_moving);
        $display("Elevator 2: Floor=%0d, Direction=%0d, Door=%b, Moving=%b",
                 elev2_floor, elev2_dir, elev2_door, elev2_moving);
        $display("=======================================");
        $display("SIMULATION COMPLETE - Check waveforms for detailed behavior");
        $display("=======================================");

        $finish;
    end

    // Continuous monitoring
    initial begin
        $monitor("Time=%0t | E1: F=%0d D=%0d Door=%b Mov=%b | E2: F=%0d D=%0d Door=%b Mov=%b",
                 $time, elev1_floor, elev1_dir, elev1_door, elev1_moving,
                 elev2_floor, elev2_dir, elev2_door, elev2_moving);
    end

    // Timeout watchdog
    initial begin
        #5000;
        $display("\n[WARNING] Simulation timeout reached at %0t ns", $time);
        $display("If no activity seen, check module connections.");
        $finish;
    end

endmodule
```

6.6 Simulation Results & Waveforms







7. CONCLUSION & FUTURE WORK

7.1 CONCLUSION

A fully functional **multi-elevator controller** was designed using Verilog HDL. The system successfully:

- Controls elevator movement using FSM logic
- Responds to internal and external requests
- Assigns requests intelligently using Nearest Car algorithm
- Simulates accurately in Vivado
- Maintains clean modularity and synthesizable design

7.2 FUTURE WORK

- Add emergency stop features
- Add load-based elevator selection
- Implement real FPGA deployment
- Add graphical floor display
- Add voice-based floor input
- Add fault-tolerant backup controller

8. REFERENCES

[1] Mano, M. M., & Ciletti, M. D. (2017). *Digital design: With an introduction to the Verilog HDL, VHDL, and SystemVerilog* (6th ed.). Pearson. https://www.amazon.com/dp/0134549899

[2] Chu, P. P. (2008). *FPGA prototyping by Verilog examples: Xilinx Spartan-3 version*. Wiley-Interscience. https://dev.store.wiley.com/en-ca/FPGA-Prototyping-by-Verilog-Examples-Xilinx-Spartan-3-Version/p/x000353095

[3] AMD. (2025). *Vivado Design Suite documentation: User and reference guides* (UG949 & UG896). AMD/Xilinx. https://docs.amd.com/en-US/ug949-vivado-design-methodology