

# **Project 85: Elevator Control System**

## **A Comprehensive Study of Advanced Digital Circuits**

**By: Abhishek Sharma , Ayush Jain , Gati Goyal, Nikunj Agrawal**

**Documentation Specialist: Dhruv Patel & Nandini Maheshwari**

Created By Team Alpha

# Contents

|   |          |
|---|----------|
| <b>1 Project Overview</b>                               | <b>3</b> |
| <b>2 Elevator Control System</b>                        | <b>3</b> |
| 2.1 Key Components of Elevator Control System . . . . . | 3        |
| 2.2 Working of Elevator Control System . . . . .        | 3        |
| 2.3 RTL Code . . . . .                                  | 4        |
| 2.4 Testbench . . . . .                                 | 6        |
| <b>3 Results</b>  | <b>7</b> |
| 3.1 Simulation . . . . .                                | 7        |
| 3.2 Schematic . . . . .                                 | 7        |
| 3.3 Synthesis Design . . . . .                          | 8        |
| <b>4 Advantages of Elevator Control System</b>          | <b>8</b> |
| <b>5 Disadvantages of Elevator Control System</b>       | <b>9</b> |
| <b>6 Applications of Elevator Control System</b>        | <b>9</b> |
| <b>7 Conclusion</b>                                     | <b>9</b> |
| <b>8 FAQs</b>   | <b>9</b> |

Created By Team Alpha

# 1 Project Overview

An Elevator Control System is a digital control system designed to manage the operation of an elevator, ensuring smooth, efficient, and safe movement between floors. The system accepts input requests from users and determines the appropriate actions, such as moving up or down, stopping at specific floors, and opening or closing the doors. Modern elevator systems aim to optimize energy usage and minimize wait times for users.

In this project, the control system is implemented using SystemVerilog and executed on Vivado software, producing simulation, synthesis, and schematic design outputs. The provided code represents a simplified version of an elevator system for a building with four floors, showcasing state-based transitions and logic.

## 2 Elevator Control System

### 2.1 Key Components of Elevator Control System

**State Machine Design:** The system operates using a finite state machine (FSM) with states such as IDLE, MOVING\_UP, MOVING\_DOWN, and OPEN\_DOOR.

**Synchronous and Combinational Logic:** Synchronous logic is used for state transitions based on the clock, while combinational logic determines the next state and control signals.

**Target Floor Determination:** The requested floor is compared with the current floor to decide the elevator's direction.

**Direction Indicators:** Separate signals indicate whether the elevator is moving up, moving down, or stationary.

**Reset Logic:** A reset signal ensures the system starts in a defined state.

### 2.2 Working of Elevator Control System

#### Initialization:

The system starts in the IDLE state when powered on or after a reset. The current\_floor is initialized to 0, the door remains closed (door\_open = 0), and the elevator is stationary.

#### Floor Request:

Users request a target floor by providing input through the floor\_request signal. The system checks if the floor\_request differs from the current\_floor.

#### Direction Determination:

If the floor\_request is greater than the current\_floor, the system sets the next state to MOVING\_UP and starts moving upwards. If the floor\_request is less than the current\_floor, the system sets the next state to MOVING\_DOWN and starts moving downwards.

#### Movement:

In the MOVING\_UP state, the current\_floor increments by 1 on each clock cycle until it matches the target\_floor. In the MOVING\_DOWN state, the current\_floor decrements by 1 on each clock cycle until it matches the target\_floor.

#### Door Operation:

Upon reaching the target\_floor, the system transitions to the OPEN\_DOOR state. The door opens (door\_open = 1), and the elevator remains stationary momentarily before transitioning back to the IDLE state.

**Reset Handling:**

If the reset signal is activated, the system immediately reverts to its initial state, disregarding ongoing operations.

**State Transition:**

The system continuously evaluates the inputs (floor\_request, current\_floor, and reset) to determine the appropriate state transitions using combinational logic in the always\_comb block.

**Direction Indicators:**

The moving\_up signal is active during the MOVING\_UP state. The moving\_down signal is active during the MOVING\_DOWN state. Both signals are inactive in the IDLE and OPEN\_DOOR states.

## 2.3 RTL Code

Listing 1: Elevator Control System

```
1
2 module elevator_control_system (
3     input  logic clk, reset,
4     input  logic [3:0] floor_request, // Requested floor (4 floors:
        0-3)
5     output logic [3:0] current_floor, // Current floor indicator
6     output logic door_open,           // Door status (1 = open)
7     output logic moving_up, moving_down // Direction indicators
8 );
9
10 typedef enum logic [1:0] {IDLE, MOVING_UP, MOVING_DOWN, OPEN_DOOR}
    state_t;
11 state_t state, next_state;
12
13 logic [3:0] target_floor; // Target floor for the elevator
14
15 // State transition logic
16 always_ff @(posedge clk or posedge reset) begin
17     if (reset) begin
18         state <= IDLE;
19         current_floor <= 4'd0;
20         target_floor <= 4'd0;
21     end else begin
22         state <= next_state;
23     end
24 end
25
26 // Determine the next state
27 always_comb begin
28     next_state = state;
29     case (state)
30         IDLE: begin
31             if (floor_request != current_floor) begin
32                 if (floor_request > current_floor) next_state =
                    MOVING_UP;
33                 else next_state = MOVING_DOWN;
34                 target_floor = floor_request;
35             end
36         end
37         MOVING_UP: begin
```

```

38         if (current_floor == target_floor) next_state =
           OPEN_DOOR;
39     end
40     MOVING_DOWN: begin
41         if (current_floor == target_floor) next_state =
           OPEN_DOOR;
42     end
43     OPEN_DOOR: begin
44         next_state = IDLE;
45     end
46 endcase
47 end
48
49 // Elevator operation logic
50 always_ff @(posedge clk or posedge reset) begin
51     if (reset) begin
52         moving_up <= 0;
53         moving_down <= 0;
54         door_open <= 0;
55     end else begin
56         case (next_state)
57             MOVING_UP: begin
58                 moving_up <= 1;
59                 moving_down <= 0;
60                 door_open <= 0;
61                 current_floor <= current_floor + 1;
62             end
63             MOVING_DOWN: begin
64                 moving_down <= 1;
65                 moving_up <= 0;
66                 door_open <= 0;
67                 current_floor <= current_floor - 1;
68             end
69             OPEN_DOOR: begin
70                 moving_up <= 0;
71                 moving_down <= 0;
72                 door_open <= 1;
73             end
74             default: begin
75                 moving_up <= 0;
76                 moving_down <= 0;
77                 door_open <= 0;
78             end
79         endcase
80     end
81 end
82
83 endmodule

```

## 2.4 Testbench

Listing 2: Elevator Control System

```
1
2 module tb_elevator_control_system();
3     logic clk, reset;
4     logic [3:0] floor_request;
5     logic [3:0] current_floor;
6     logic door_open, moving_up, moving_down;
7
8     elevator_control_system uut (
9         .clk(clk),
10        .reset(reset),
11        .floor_request(floor_request),
12        .current_floor(current_floor),
13        .door_open(door_open),
14        .moving_up(moving_up),
15        .moving_down(moving_down)
16    );
17
18    // Clock generation
19    initial begin
20        clk = 0;
21        forever #5 clk = ~clk; // 10ns clock period
22    end
23
24    // Test scenario
25    initial begin
26        reset = 1;
27        floor_request = 4'd0;
28        #10 reset = 0;
29
30        // Request Floor 3
31        #20 floor_request = 4'd3;
32        #100;
33
34        // Request Floor 1
35        floor_request = 4'd1;
36        #100;
37
38        // Reset system
39        reset = 1;
40        #10 reset = 0;
41
42        // Stop simulation
43        #100 $stop;
44    end
45
46    // Monitor outputs
47    initial begin
48        $monitor("Time: %0t | Floor Request: %d | Current Floor: %d |
49                Door: %b | Up: %b | Down: %b",
50                $time, floor_request, current_floor, door_open,
51                moving_up, moving_down);
52    end
53 endmodule
```

## 3 Results

### 3.1 Simulation

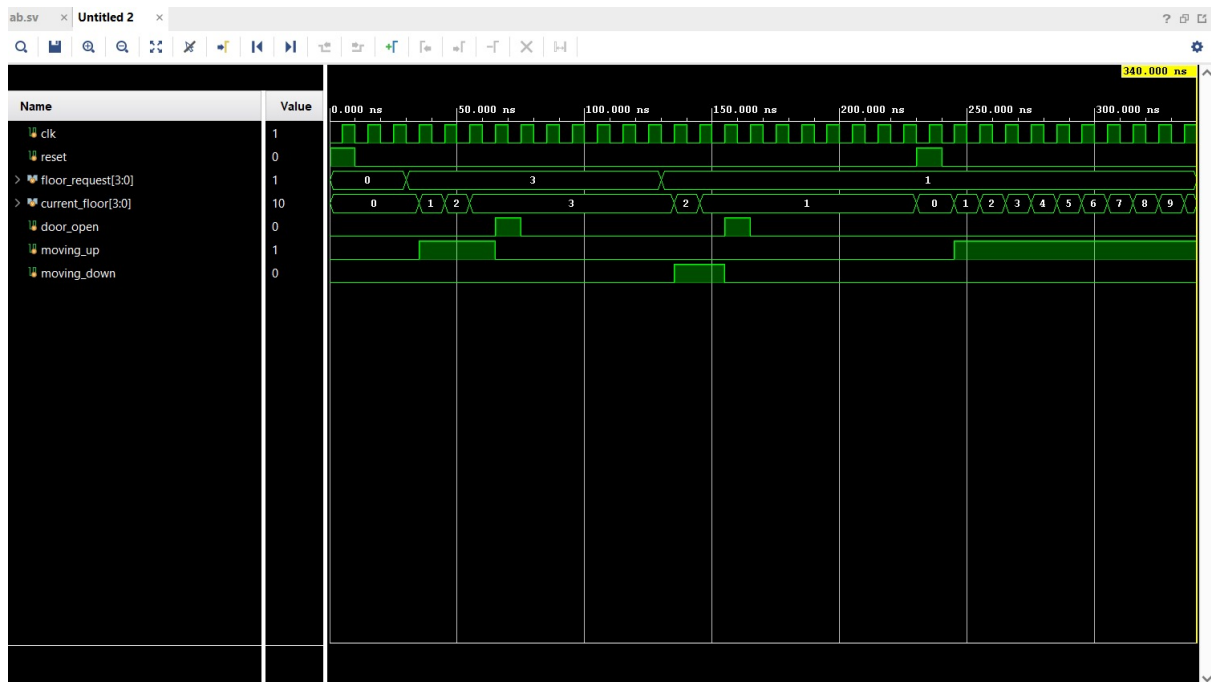


Figure 1: Simulation of Elevator Control System

### 3.2 Schematic

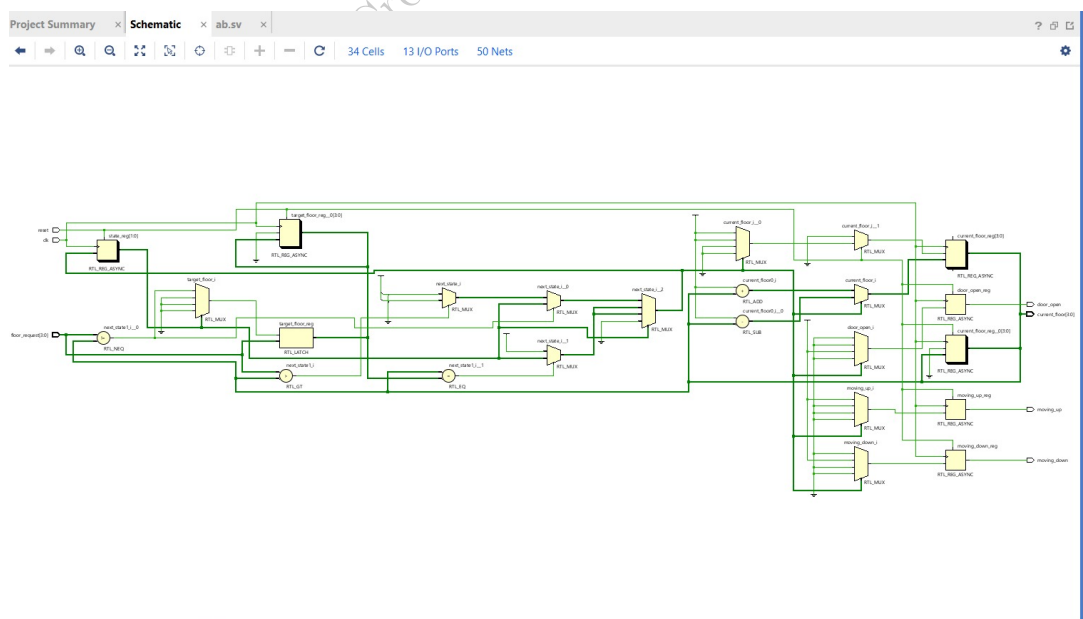


Figure 2: Schematic of Elevator Control System

### 3.3 Synthesis Design

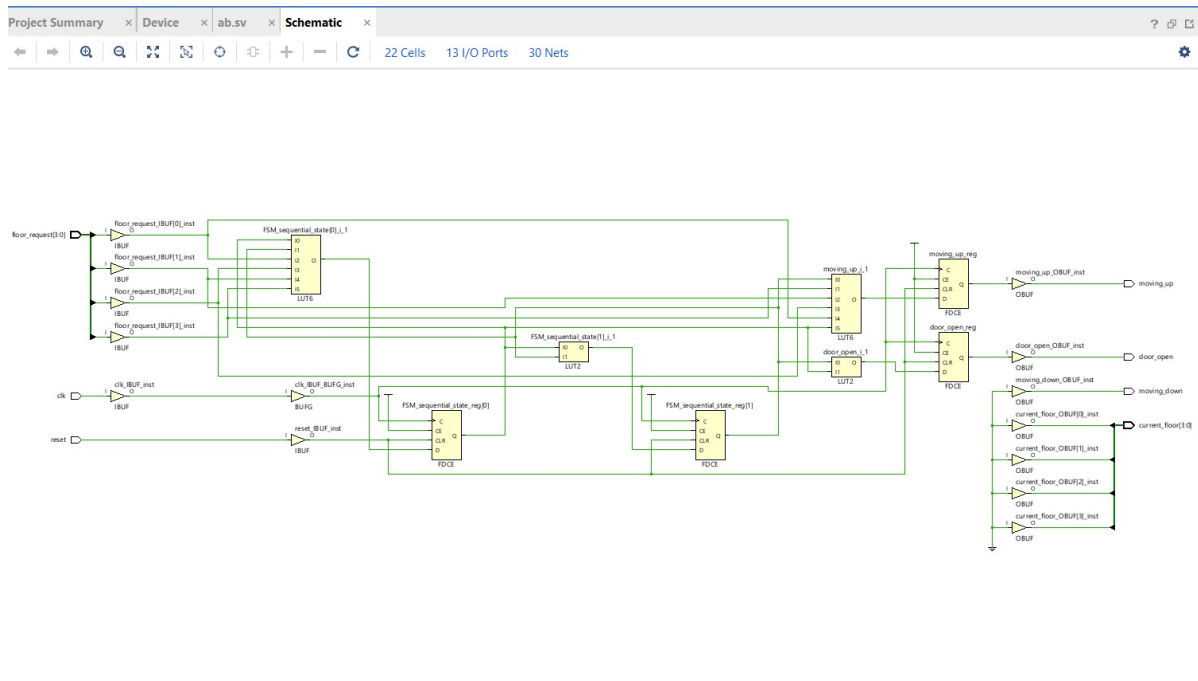


Figure 3: Synthesis Design of Elevator Control System

## 4 Advantages of Elevator Control System

**Efficient Management:** Optimizes elevator operations, reducing wait times and energy consumption.

**Safety:** Ensures smooth transitions and predictable behavior.

**Scalability:** Can be expanded to accommodate more floors or advanced features like prioritization of requests.



## 5 Disadvantages of Elevator Control System

**Limited Functionality:** The basic design does not handle multiple simultaneous requests or emergency conditions.

**Hardware Constraints:** Larger designs may require more resources, increasing costs.

**Complexity for Larger Systems:** Expanding to multiple elevators or advanced algorithms increases design complexity.

**Timing Issues:** Improper clock synchronization may lead to erratic behavior.

**Failsafe Handling:** The current implementation lacks robust mechanisms for handling unexpected hardware or logic failures.

## 6 Applications of Elevator Control System

**Residential Buildings:** Managing elevators in apartments or small office spaces.

**Commercial Complexes:** Optimizing elevator operations in malls or high-rise offices.

**Industrial Facilities:** Controlling lifts for material transport in factories.

**Healthcare:** Ensuring quick and reliable movement in hospitals.

**Smart Buildings:** Integration with IoT for advanced scheduling and predictive maintenance.

## 7 Conclusion

The Elevator Control System designed in SystemVerilog is a practical demonstration of digital design principles, utilizing finite state machines, synchronous logic, and combinational logic. It efficiently manages elevator operations such as responding to floor requests, moving between floors, and operating doors. While the current implementation is basic, it provides a solid foundation for extending functionality to include multi-floor optimization, simultaneous request handling, and advanced safety features.

## 8 FAQs

1. What is an Elevator Control System, and why is it implemented using SystemVerilog?

**Answer:** An Elevator Control System is a digital system for managing elevator operations. SystemVerilog is used for its robust hardware modeling and verification capabilities, making it suitable for designing complex state machines like elevators.

2. Explain the role of FSM in your design.

**Answer:** The finite state machine (FSM) governs the elevator's behavior by defining specific states (IDLE, MOVING\_UP, MOVING\_DOWN, OPEN\_DOOR) and transitions based on inputs such as floor requests and the current floor.

3. How does the system decide the direction of movement?

**Answer:** The system compares the `floor_request` with the `current_floor`. If the requested floor is higher, the elevator moves up; if lower, it moves down.

#### 4. What are the inputs and outputs of your system?

**Answer:** Inputs include `clk`, `reset`, and `floor_request`. Outputs include `current_floor`, `door_open`, `moving_up`, and `moving_down`.

#### 5. What happens during a reset condition?

**Answer:** The system resets to the initial state, setting `current_floor` to 0, closing the door, and entering the IDLE state.

#### 6. Why is the next state logic written as a combinational block?

**Answer:** The combinational block allows immediate calculation of the next state based on current inputs and the state, enabling smooth transitions without waiting for the clock edge.

#### 7. How does the system handle multiple requests?

**Answer:** The provided design handles one request at a time. Extending it to queue multiple requests would involve additional logic, such as request buffers or priority handling.

#### 8. What improvements can be made to this design?

**Answer:** Enhancements could include:

- Handling simultaneous requests.
- Adding emergency stop functionality.
- Optimizing for minimal energy consumption.

#### 9. What is the significance of simulation in this project?

**Answer:** Simulation helps verify the correctness of the design before implementation, ensuring all transitions and outputs behave as expected.

#### 10. How would you scale this design for a high-rise building?

**Answer:** Scaling involves increasing the range of `floor_request` and `current_floor`, adding logic to prioritize requests, and possibly integrating machine learning for demand prediction.

#### 11. What are potential hazards in an elevator control system?

**Answer:** Hazards include abrupt stops, incorrect floor alignment, and failure to respond to requests, which can be mitigated with thorough testing and robust error-handling mechanisms.

#### 12. Explain the role of `always_ff` and `always_comb` blocks in your code.

**Answer:** `always_ff` is used for sequential logic, updating states on the clock edge. `always_comb` handles combinational logic, determining the next state and output signals.

#### 13. What challenges did you face while implementing this project?

**Answer:** Challenges include ensuring correct state transitions, synchronizing the clock, and debugging combinational and sequential logic interactions.

**14. What are some real-world constraints to consider in elevator systems?**

**Answer:** Constraints include weight limits, power availability, and handling multiple user inputs simultaneously.

**15. How can this project contribute to smart building technologies?**

**Answer:** The design can be integrated with IoT for remote monitoring, predictive maintenance, and user-specific optimizations, enhancing overall efficiency.

Created By Team Alpha