Complex Engineering Problem

Elevator control system

Submitted by

Ayesha Shafique 2020-CE-39

Submitted to

Sir Afeef Obaid

Course

CMPE-421L Computer Architecture

Semester

7th

Date

1st October 2023

Department of Computer of Engineering

University of Engineering and Technology, Lahore

Table of Contents

Introduction

Elevators, a cornerstone of modern urban infrastructure, have transformed the way we navigate multi-story buildings, offering unparalleled convenience and access. Their operational precision is vital to ensuring efficient vertical transportation while prioritizing passenger safety.

Our focus lies in the creation of a System Verilog Finite State Machine (FSM) that governs an elevator control system. This system caters to the intricate demands of multi-story buildings, balancing passenger requests and security measures.

The elevator control system outlined herein adheres to a comprehensive set of requirements. These include accommodating user floor selections, managing passenger priority requests, providing an emergency stop mechanism, employing limit sensors for floor range supervision, orchestrating a well-structured finite state machine, developing a user-friendly interface, and subjecting the system to rigorous testing and simulation.

Assumptions Made

These assumptions provide context for understanding how the elevator control system functions in various scenarios. Let's break down each assumption:

1. Direction Persistence

The first assumption, about the elevator not changing direction until it completes all requests in that direction, is crucial for maintaining efficiency in elevator operations. This means that if the elevator is moving upward, it will continue to serve requests going upward until there are no more requests in that direction. The same applies when moving downward. This behavior aligns with the concept of the elevator prioritizing requests in the direction it's currently moving.

2. Idle State

The assumption that the elevator stops and becomes idle after serving all requests at the latest destination floor is a practical one. It ensures that the elevator doesn't keep moving unnecessarily once all passenger requests have been fulfilled. The idle state is a crucial part of the system's operation as it signifies that the elevator is available for new requests.

3. Door Control Signal

The assumption that arrival sensors generate a control signal for opening the lift door for a clock timer upon reaching the destination floor is a standard feature in elevator systems. It

ensures that the doors remain open for a sufficient time to allow passengers to enter and exit the elevator safely. This safety feature is essential for preventing accidents and ensuring passenger convenience.

Collectively, these assumptions provide a clear understanding of the elevator control system's behavior and operation. They set the foundation for defining the system's states, transitions, and logic, as well as for designing test scenarios to validate the system's performance. Additionally, they align with common practices in elevator control systems, ensuring that the designed system operates efficiently and safely in a multi-story building.

Modified Priority Algorithm Implementation and Functionality

1. Priority Algorithm Overview

The Elevator Control System has been significantly improved through the implementation of a modified priority algorithm. This algorithm introduces key enhancements to request handling, aiming to optimize elevator operations by intelligently prioritizing user requests based on proximity and direction, thereby overcoming the limitations of the traditional First-Come-First-Serve (FCFS) approach.

2. Proximity-Based Prioritization

- **Directional Priority**

One of the primary enhancements is the introduction of directional priority. This feature ensures that the elevator considers the direction it's currently moving in and prioritizes requests that align with this direction. Specifically:

- When the elevator is moving upwards, it gives higher priority to requests for higher floors in the same upward direction.
- When the elevator is moving downwards, it prioritizes requests for lower floors in the same downward direction.

- **Proximity Priority**

Within requests that are in the same direction as the elevator's movement, the algorithm further refines its decision-making process by selecting the request that is closest in terms of floors to the elevator's current position. This approach guarantees that the elevator serves the most relevant request efficiently, reducing unnecessary travel and wait times for passengers.

3. Efficient Movement Planning

- **Direction Adjustment**

To ensure efficient response to prioritized requests, the elevator dynamically adjusts its movement direction based on the highest-priority request identified by the algorithm. If, for instance, the elevator is currently moving upward but a higher-priority request is in the downward direction, the elevator will alter its course to serve this priority request optimally.

- **Reducing Unnecessary Stops**

By giving precedence to requests that align with the elevator's current direction and minimizing the distance to the next priority request, the modified algorithm significantly diminishes the number of unnecessary stops. This not only conserves energy but also leads to shorter passenger wait times, enhancing overall efficiency and user satisfaction.

4. Handling Multiple Requests

- **Optimized Order**

In scenarios where multiple users request different floors in the same direction, the elevator control system is designed to optimize the order in which these requests are fulfilled. The system prioritizes requests that minimize the distance traveled, ensuring that passengers are picked up and dropped off efficiently, further reducing travel time and energy consumption.

- **Energy Efficiency**

This optimized order not only benefits passengers by reducing their time spent in the elevator but also contributes to energy conservation, as fewer stops and less unnecessary travel result in reduced power consumption and environmental impact.

5. Emergency System Integration

- **Emergency Stop**

In the interest of passenger safety, the elevator system incorporates an emergency system that allows passengers to activate an emergency stop button when necessary. When this button is activated, the elevator comes to a controlled halt, saving the current direction and status. This ensures a safe response to emergencies.

- **Resumption After Emergency**

Following the resolution of the emergency and the rest of the emergency stop button, the elevator seamlessly resumes its normal operation. It automatically reverts to the saved direction, ensuring that the safety features of the emergency system are integrated with the

efficiency of the modified priority algorithm, maintaining both passenger well-being and operational efficiency.

In summary, the implementation of the modified priority algorithm within the Elevator Control System represents a substantial improvement in elevator operations. By intelligently prioritizing requests based on proximity and direction, the system reduces travel times, conserves energy, and enhances the overall passenger experience. The seamless integration of an emergency system ensures that safety is not compromised while maximizing efficiency in elevator operations.

Optimization and Achievement: Scalable Elevator Control System

The Elevator Control System presented in this report is designed for 8 floors, but it has been strategically optimized to seamlessly adapt to a range of floor counts. This achievement is a testament to the system's versatility and scalability, which are essential attributes for addressing diverse building configurations and future needs.

1. Adaptive Scalability

The elevator Control System is not confined to a specific number of floors. It can be efficiently scaled down to accommodate as few as 2 floors or scaled up to handle potentially hundreds of floors, such as 64 floors, with minimal changes to the code and no changes core state table and diagram.

2. Unified State Table and Diagram

What sets our system apart is its unified state table and diagram. Regardless of the number of floors, the fundamental logic and structure of the state transitions remain consistent. This unified approach simplifies development, maintenance, and understanding, ensuring that modifications for different floor counts are straightforward.

3. Resource Efficiency

Our design prioritizes resource efficiency. It strikes a balance between flexibility and resource consumption, ensuring that the system remains responsive and resource-friendly even when scaling to handle a wide range of floor counts.

4. Code Maintainability

Achieving scalability without compromising code readability and maintainability is a key success factor. The codebase is well-structured and organized, making it easy to adapt to varying floor counts while preserving its clarity and ease of maintenance.

5. Thorough Testing and Validation

Rigorous testing and validation have been conducted, primarily focusing on the 8-floor configuration. This extensive testing demonstrates the system's reliability and robustness, laying a solid foundation for scalability testing in the future.

6. Universal Applicability

The adaptability of our Elevator Control System to different floor counts makes it a universal solution. Architects, building planners, and developers can confidently apply this system to a wide range of building structures, ensuring that it meets their specific requirements.

Future-Ready Design

By optimizing our system to handle 8 floors while ensuring scalability, we are future-proofing the technology. This forward-looking approach anticipates the evolving demands of the vertical transportation industry, where building sizes and complexities continue to evolve.

In conclusion, our Elevator Control System's optimization and achievement of adaptability to varying floor counts showcase its versatility, scalability, and resource efficiency. This accomplishment positions our system as a versatile and future-ready solution, ready to meet the diverse needs of the vertical transportation industry, from small-scale buildings to high-rises with numerous floors.

State Tables

Table 1

Table 1 provides a detailed overview of the finite state machine transitions within our elevator control system. It outlines the current states, input conditions, next states, and associated output actions. This table serves as a comprehensive reference for understanding the elevator's behavior and responses under different scenarios, including reset conditions, door operations, and movement directions.

Current State	Input	Next State	Output
Reset	reset = 1	Reset	Idle =1,door=0, Up=1, Down=0,requests=0, estop=0, current_floor=0
	reset = 0	Door Closed & Idle (1)	Idle =1,door=0, Up=1, door_timer=0
Door Closed & Idle (1)	Checker : requests[current_floor] = 1	Door Open & Idle (1)	door = 1; idle = 1; requests[current_floor] = 0; door_timer = 1
	max_request > current_floor && Up = 0	Moving Up	idle = 0; current floor +=1
	min_request > current_floor && Down = 0	Moving Down	idle = 0; current floor -=1
	max_request = current_floor	Down Direction Setter	Up=0; Down=1
	min_request = current_floor	Up Direction Setter	Up=1; Down=0
Door Open & Idle (1)	door_timer = 1	Door Closed & Idle (1)	Idle =1,door=0, Up=1, door_timer=0
Moving Up	Checker = 1	Door Open & Idle (1)	door = 1; idle = 1; requests[current_floor] = 0; door_timer = 1
	Checker = 0	Moving Up	idle = 0; current floor +=1
	estop = 1	Door Closed & Idle (1)	Idle =1,door=0, Up=1, door_timer=0
Moving Down	Checker = 1	Door Open & Idle (1)	door = 1; idle = 1; requests[current_floor] = 0; door_timer = 1
	Checker = 0	Moving Down	idle = 0; current floor -=1
	estop = 1	Door Closed & Idle (1)	Idle =1,door=0, Up=1, door_timer=0
Up Direction Setter	x	Door Open & Idle (1)	door = 1; idle = 1; requests[current_floor] = 0; door_timer = 1
Down Direction Setter	x	Door Open & Idle (1)	door = 1; idle = 1; requests[current_floor] = 0; door_timer = 1

Table 2

Table 2 presents a concise representation of state transitions related to the update of elevator requests, **max_request**, and **min_request**. It defines how the elevator system handles new floor requests and maintains these key variables, which play a crucial role in optimizing request prioritization and elevator movement. This table simplifies the understanding of request management within the elevator control system.

Current State	Input	Next State	Output
UpdateRequests	NewFloorRequested	NewFloorRequested	*x*
	No NewFloorRequested	UpdateRequests	
NewFloorRequested	max_req < req_floor	UpdateMaxReq	max_req = req_floor
	min_req > req_floor	UpdateMinReq	min_req = req_floor
	req[max_req] == 0 & req_floor>currFloor	UpdateMaxReq	max_req = req_floor
	req[min_req] == 0 & req_floor<currFloor	UpdateMinReq	min_req = req_floor
UpdateMaxReq	*x*	UpdateRequests	*x*
UpdateMinReq		UpdateRequests	

State Diagrams

In addition to the implementation of the elevator control system in System Verilog, we have created comprehensive state diagrams to visualize the system's finite state machine. These state diagrams were generated using the Python library **pydot**, providing a clear and intuitive representation of the elevator's operational states and transitions.

Code link:
https://colab.research.google.com/drive/1INVTcT3oR4ZWipnJEBOiINcCHq3BKDPr?usp=sharing

Diagram 1

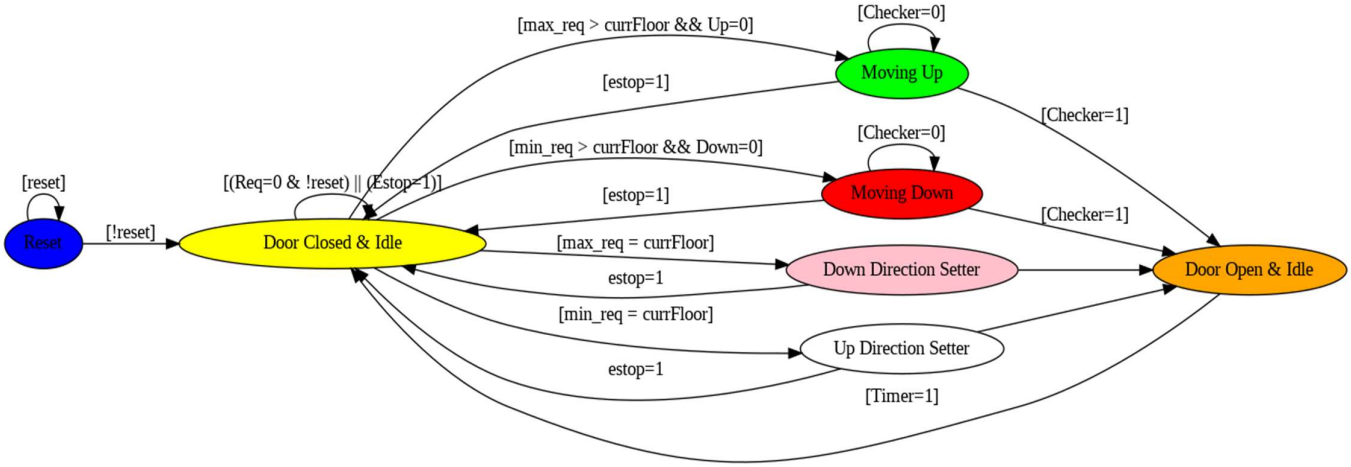
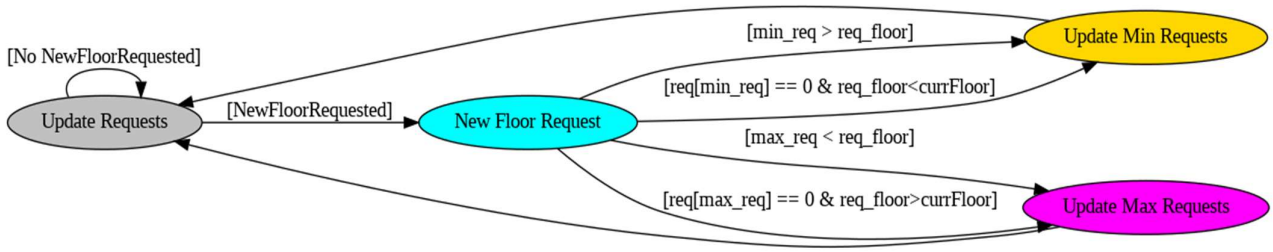


Diagram 2



Design.sv (Part 1)

```
module Lift8(clk, reset, req_floor, idle, door, Up, Down, current_floor,
requests, max_request, min_request, emergency_stop);

  input clk, reset, emergency_stop;
  input logic [2:0] req_floor;      // 3-bit input for 8 floors (0 to 7)
  output logic [1:0] door;
  output logic [2:0] max_request;
  output logic [2:0] min_request;
  output logic [1:0] Up;
  output logic [1:0] Down;
  output logic [1:0] idle;
  output logic [2:0] current_floor;
  output logic [7:0] requests;

  logic door_timer;
  logic emergency_stopped;
  logic flag=0;

  // Update requests when a new floor is requested
  always @(req_floor)
  begin
    requests[req_floor] = 1;
  // Update max_request and min_request based on requested floors
    if (max_request < req_floor)
    begin
      max_request = req_floor;
    end

    if (min_request > req_floor)
    begin
      min_request = req_floor;
    end

      // Update max_request and min_request based on current floor

    if (requests[max_request] == 0 && req_floor > current_floor)
    begin
      max_request = req_floor;
    end

    if (requests[min_request] == 0 && req_floor < current_floor)
    begin
      min_request = req_floor;
    end

  end

  // Check and update lift behavior based on current floor
  always @(current_floor )
  begin
    if (requests[current_floor] == 1)
```

Design.sv (Part 2)

```
    if (requests[current_floor] == 1)
    begin
      idle = 1;
      door = 1;
      requests[current_floor] = 0;
      door_timer = 1; // Start the door timer when opening
    end
  end

  // State machine for lift control
  always @(posedge clk )

  begin
    if (door_timer == 1)
    begin
      door <= 0; // Close the door after the one clock expires
      //$display("%h", current_floor);
    end
    if (reset)
    begin
      // Reset lift to initial state
      flag=0;
      current_floor <= 0;
      idle <= 0;
      door <= 0; // door open
      Up <= 1;   // going up
      Down <= 0; // not going down
      max_request <= 0;
      min_request <= 7;
      requests <= 0;
      emergency_stopped <= 0; // Initialize emergency stop state
    end
    else if (requests == 0 && !reset)
    begin
      // Stay on the current floor if no requests
      current_floor <= current_floor;
      emergency_stopped <= 0; // Clear emergency stop when not moving
    end
    // emergency
    else if (emergency_stop)
    begin
      // Emergency stop button is turned on
      idle <= 1;
      flag <=1;
      emergency_stopped <= 1; // Set emergency stop state
    end
    else if (emergency_stopped && emergency_stop)
    begin
      // Remain stopped until the emergency stop button is reset
      current_floor <= current_floor;
      door <= 0; // Keep the door closed during an emergency stop
    end
```

Design.sv (Part 3)

```
98        // Remain stopped until the emergency stop button is reset
99        current_floor <= current_floor;
100       door <= 0; // Keep the door closed during an emergency stop
101     end
102     // emergency reset
103     else if (!emergency_stop && flag)
104     begin
105       // Emergency stop button is turned off
106       emergency_stopped <= 0; // Set emergency stop state
107       flag <=0;
108     end
109     else
110     begin
111       // Normal operation when not in emergency stop
112       if (max_request <= 7)
113       begin
114         if (min_request < current_floor && Down == 1)
115         begin
116           // Move down one floor
117           current_floor <= current_floor - 1;
118           door <= 0;
119           idle <= 0;
120         end
121         else if (max_request > current_floor && Up == 1)
122         begin
123           // Move up one floor
124           current_floor <= current_floor + 1;
125           door <= 0;
126           idle <= 0;
127         end
128         else if (req_floor == current_floor)
129         begin
130 //              // Open door and handle request
131             door <= 1;
132             idle <= 1;
133         end
134         else if (max_request == current_floor)
135         begin
136           Up <= 0;
137           Down <= 1;
138         end
139         else if (min_request == current_floor)
140         begin
141           Up <= 1;
142           Down <= 0;
143         end
144       end
145     end
146   end
147 endmodule
```

Design.sv Explanation

1. Module Declaration

- The code defines a Verilog module named "Lift8" with various input and output signals.

- **Inputs**
  - **clk**: A clock signal for synchronous operation.
  - **reset**: A reset signal used to initialize the elevator system.
  - **req_floor**: A 3-bit input representing the requested floors (0 to 7).
  - **emergency_stop**: An input signal that serves as an emergency stop button.
- **Outputs**
  - **door**: A 2-bit output representing the state of the elevator door.
  - **max_request**: A 3-bit output representing the highest requested floor.
  - **min_request**: A 3-bit output representing the lowest requested floor.
  - **Up**: A 2-bit output indicating whether the elevator is moving up.
  - **Down**: A 2-bit output indicating whether the elevator is moving down.
  - **idle**: A 2-bit output indicating whether the elevator is idle.
  - **current_floor**: A 3-bit output representing the current floor of the elevator.
  - **requests**: An 8-bit output representing the floor requests.

2. Internal Variables

- **door_timer**: A signal used to control the timing of elevator door opening and closing.
- **emergency_stopped**: A signal to indicate whether the elevator is in an emergency stop state.
- **flag**: A flag to manage emergency stop state transitions.
- **updownflag**: A flag used during emergency reset to keep track of the previous values of "Up" and "Down" directions.

3. Request Handling

- An **always** block updates the elevator's request list (**requests**) based on the requested floors (**req_floor**).
- It also updates **max_request** (Highest Floor no. in requests queue) and **min_request** (Lowest Floor no. in requests queue) based on the requested floors and the current floor.

4. Current Floor Handling

- Another **always** block handles the elevator's behavior when it reaches a requested floor.
- If the elevator arrives at a requested floor, it sets the elevator to idle, opens the door, clears the request, and starts a door timer.

5. State Machine for Lift Control

- The core logic of the elevator control system is implemented within an **always @(posedge clk)** block, representing a synchronous state machine.
- It handles various states and transitions:
    - **Reset:** Initializes elevator state.
    - **Emergency Stop:** Activated when the emergency stop button is pressed.
    - **Emergency Stop (During Stopped State):** Remains stopped until the emergency stop button is released.
    - **Emergency Reset:** Clears the emergency stop state when the button is released.
    - **Normal Operation:** Handles normal elevator movement based on requests and current floor.
    - **Opening Door:** Opens the door when the elevator arrives at a requested floor.
    - **Other Cases:** Handles cases for moving up, moving down, and staying idle based on the requested floors.

6. Usage of min_request and max_request

- **min_request** keeps track of the lowest floor number that has pending requests, while **max_request** tracks the highest floor number with pending requests.
- These variables are used to optimize the elevator's movement strategy:
    - If the elevator is moving upward and reaches **max_request**, it changes direction to move downward to serve lower floors efficiently.
    - Similarly, if the elevator is moving downward and reaches **min_request**, it changes direction to move upward to serve higher floors efficiently.
- **min_request** and **max_request** ensure that the elevator prioritizes requests based on their relative positions, minimizing unnecessary travel and wait times.

In summary, this Verilog module represents an elevator control system that manages floor requests, handles emergencies, and controls the elevator's movement and door operation. The code uses a state machine to control the elevator's behavior under different conditions and states.

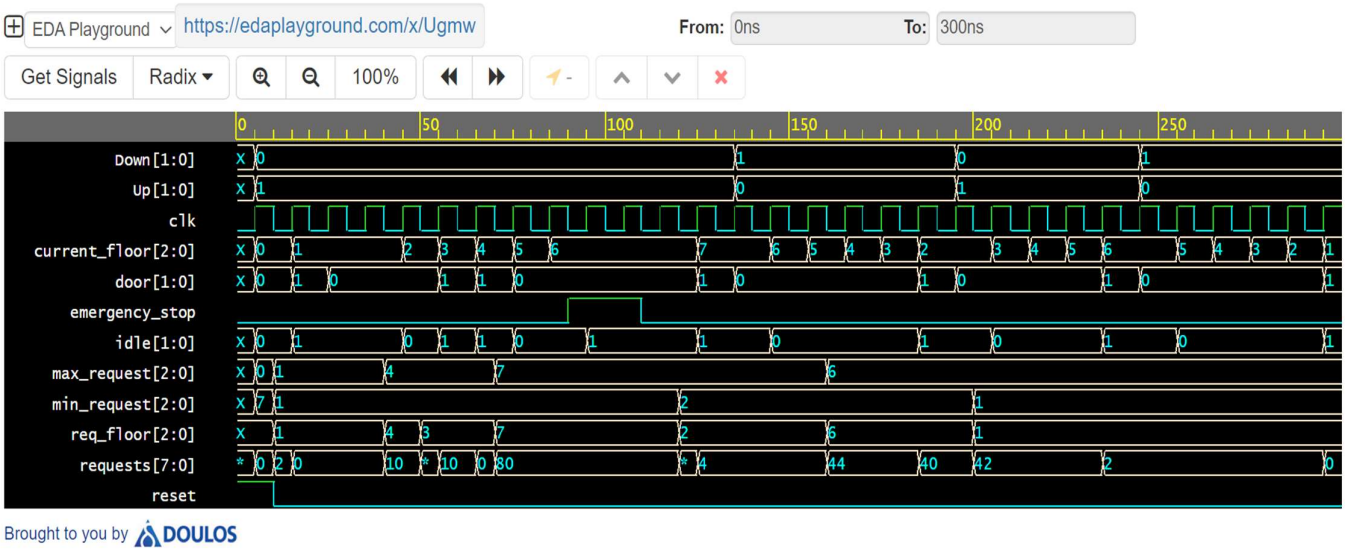
Testbench (Part 1)

```
// Code your testbench here
// or browse Examples
module Lift8_Tb();
  logic clk, reset;
  logic [2:0] req_floor;
  logic [1:0] idle, door, Up, Down;
  logic [2:0] current_floor;
  logic [2:0] max_request, min_request;
  logic [7:0] requests;
  logic emergency_stop;

  Lift8 dut(
    .clk(clk),
    .reset(reset),
    .req_floor(req_floor),
    .idle(idle),
    .door(door),
    .Up(Up),
    .Down(Down),
    .current_floor(current_floor),
    .max_request(max_request),
    .min_request(min_request),
    .requests(requests),
    .emergency_stop(emergency_stop)
  );

  initial begin
    $dumpfile("waveform.vcd"); // Specify the VCD waveform output file
    $dumpvars(0, Lift8_Tb);    // Dump all variables in the module hierarchy

    clk = 1'b0;
    emergency_stop = 0;
    reset = 1;
    #10;
    reset = 0;
    req_floor = 1;
    #30;
    req_floor = 4;

    #10

    // Simulate elevator operation
    req_floor = 3; // Request floor 3
    #20;
    req_floor = 7; // Request floor 5
    #20;
    emergency_stop = 1; // Activate emergency stop
    #20;
    emergency_stop = 0; // Deactivate emergency stop
    #10;
```

Testbench (Part 2)

```
51      req_floor = 2; // Request floor 2
52      #40;
53      req_floor = 6; // Request floor 6
54      #20;
55
56      #20;
57      req_floor = 1;
58    end
59
60    initial begin
61    $display("Starting simulation...");
62    $monitor("Time = %t, clk = %b, reset = %b, req_floor = %h, idle = %b, door = %b,
Up = %b, Down = %b, current_floor = %h, max_request = %h, min_request = %h,
requests = %h",
63        $time, clk, reset, req_floor, idle, door, Up, Down, current_floor,
max_request, min_request, requests);
64    // Run the simulation for a sufficient duration
65    #305; // Adjust the simulation time as needed
66    $display("Simulation finished.");
67    $finish;
68  end
69
70  // C
71    always #5 clk = ~clk;
72  endmodule
73
74
```

Epwave



Terminal Results

```
# Starting simulation...
# Time= 0,clk=0,reset=1,req_floor=x,idle=x,door=x,Up=x,Down=x,current_floor=x,max_request=x,min_request=x,requests=xx
# Time= 5,clk=1,reset=1,req_floor=x,idle=0,door=0,Up=1,Down=0,current_floor=0,max_request=0,min_request=7,requests=00
# Time= 10,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=0,max_request=1,min_request=1,requests=02
# Time= 15,clk=1,reset=0,req_floor=1,idle=1,door=1,Up=1,Down=0,current_floor=1,max_request=1,min_request=1,requests=00
# Time= 20,clk=0,reset=0,req_floor=1,idle=1,door=1,Up=1,Down=0,current_floor=1,max_request=1,min_request=1,requests=00
# Time= 25,clk=1,reset=0,req_floor=1,idle=1,door=0,Up=1,Down=0,current_floor=1,max_request=1,min_request=1,requests=00
# Time= 30,clk=0,reset=0,req_floor=1,idle=1,door=0,Up=1,Down=0,current_floor=1,max_request=1,min_request=1,requests=00
# Time= 35,clk=1,reset=0,req_floor=1,idle=1,door=0,Up=1,Down=0,current_floor=1,max_request=1,min_request=1,requests=00
# Time= 40,clk=0,reset=0,req_floor=4,idle=1,door=0,Up=1,Down=0,current_floor=1,max_request=4,min_request=1,requests=10
# Time= 45,clk=1,reset=0,req_floor=4,idle=0,door=0,Up=1,Down=0,current_floor=2,max_request=4,min_request=1,requests=10
# Time= 50,clk=0,reset=0,req_floor=3,idle=0,door=0,Up=1,Down=0,current_floor=2,max_request=4,min_request=1,requests=18
# Time= 55,clk=1,reset=0,req_floor=3,idle=1,door=1,Up=1,Down=0,current_floor=3,max_request=4,min_request=1,requests=10
# Time= 60,clk=0,reset=0,req_floor=3,idle=1,door=1,Up=1,Down=0,current_floor=3,max_request=4,min_request=1,requests=10
# Time= 65,clk=1,reset=0,req_floor=3,idle=1,door=1,Up=1,Down=0,current_floor=4,max_request=4,min_request=1,requests=00
# Time= 70,clk=0,reset=0,req_floor=7,idle=1,door=1,Up=1,Down=0,current_floor=4,max_request=7,min_request=1,requests=80
# Time= 75,clk=1,reset=0,req_floor=7,idle=0,door=0,Up=1,Down=0,current_floor=5,max_request=7,min_request=1,requests=80
# Time= 80,clk=0,reset=0,req_floor=7,idle=0,door=0,Up=1,Down=0,current_floor=5,max_request=7,min_request=1,requests=80
# Time= 85,clk=1,reset=0,req_floor=7,idle=0,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 90,clk=0,reset=0,req_floor=7,idle=0,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 95,clk=1,reset=0,req_floor=7,idle=1,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 100,clk=0,reset=0,req_floor=7,idle=1,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 105,clk=1,reset=0,req_floor=7,idle=1,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 110,clk=0,reset=0,req_floor=7,idle=1,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 115,clk=1,reset=0,req_floor=7,idle=1,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=1,requests=80
# Time= 120,clk=0,reset=0,req_floor=2,idle=1,door=0,Up=1,Down=0,current_floor=6,max_request=7,min_request=2,requests=84
# Time= 125,clk=1,reset=0,req_floor=2,idle=1,door=1,Up=1,Down=0,current_floor=7,max_request=7,min_request=2,requests=04
# Time= 130,clk=0,reset=0,req_floor=2,idle=1,door=1,Up=1,Down=0,current_floor=7,max_request=7,min_request=2,requests=04
# Time= 135,clk=1,reset=0,req_floor=2,idle=1,door=0,Up=0,Down=1,current_floor=7,max_request=7,min_request=2,requests=04
# Time= 140,clk=0,reset=0,req_floor=2,idle=1,door=0,Up=0,Down=1,current_floor=7,max_request=7,min_request=2,requests=04
# Time= 145,clk=1,reset=0,req_floor=2,idle=0,door=0,Up=0,Down=1,current_floor=6,max_request=7,min_request=2,requests=04
# Time= 150,clk=0,reset=0,req_floor=2,idle=0,door=0,Up=0,Down=1,current_floor=6,max_request=7,min_request=2,requests=04
# Time= 155,clk=1,reset=0,req_floor=2,idle=0,door=0,Up=0,Down=1,current_floor=5,max_request=7,min_request=2,requests=04
# Time= 160,clk=0,reset=0,req_floor=6,idle=0,door=0,Up=0,Down=1,current_floor=5,max_request=6,min_request=2,requests=44
# Time= 165,clk=1,reset=0,req_floor=6,idle=0,door=0,Up=0,Down=1,current_floor=4,max_request=6,min_request=2,requests=44
# Time= 170,clk=0,reset=0,req_floor=6,idle=0,door=0,Up=0,Down=1,current_floor=4,max_request=6,min_request=2,requests=44
# Time= 175,clk=1,reset=0,req_floor=6,idle=0,door=0,Up=0,Down=1,current_floor=3,max_request=6,min_request=2,requests=44
# Time= 180,clk=0,reset=0,req_floor=6,idle=0,door=0,Up=0,Down=1,current_floor=3,max_request=6,min_request=2,requests=44
# Time= 185,clk=1,reset=0,req_floor=6,idle=1,door=1,Up=0,Down=1,current_floor=2,max_request=6,min_request=2,requests=40
# Time= 190,clk=0,reset=0,req_floor=6,idle=1,door=1,Up=0,Down=1,current_floor=2,max_request=6,min_request=2,requests=40
# Time= 195,clk=1,reset=0,req_floor=6,idle=1,door=0,Up=1,Down=0,current_floor=2,max_request=6,min_request=2,requests=40
# Time= 200,clk=0,reset=0,req_floor=1,idle=1,door=0,Up=1,Down=0,current_floor=2,max_request=6,min_request=1,requests=42
# Time= 205,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=3,max_request=6,min_request=1,requests=42
# Time= 210,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=3,max_request=6,min_request=1,requests=42
# Time= 215,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=4,max_request=6,min_request=1,requests=42
# Time= 220,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=4,max_request=6,min_request=1,requests=42
# Time= 225,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=5,max_request=6,min_request=1,requests=42
# Time= 230,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=1,Down=0,current_floor=5,max_request=6,min_request=1,requests=42
```

```
# Time= 235,clk=1,reset=0,req_floor=1,idle=1,door=1,Up=1,Down=0,current_floor=6,max_request=6,min_request=1,requests=02
# Time= 240,clk=0,reset=0,req_floor=1,idle=1,door=1,Up=1,Down=0,current_floor=6,max_request=6,min_request=1,requests=02
# Time= 245,clk=1,reset=0,req_floor=1,idle=1,door=0,Up=0,Down=1,current_floor=6,max_request=6,min_request=1,requests=02
# Time= 250,clk=0,reset=0,req_floor=1,idle=1,door=0,Up=0,Down=1,current_floor=6,max_request=6,min_request=1,requests=02
# Time= 255,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=5,max_request=6,min_request=1,requests=02
# Time= 260,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=5,max_request=6,min_request=1,requests=02
# Time= 265,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=4,max_request=6,min_request=1,requests=02
# Time= 270,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=4,max_request=6,min_request=1,requests=02
# Time= 275,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=3,max_request=6,min_request=1,requests=02
# Time= 280,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=3,max_request=6,min_request=1,requests=02
# Time= 285,clk=1,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=2,max_request=6,min_request=1,requests=02
# Time= 290,clk=0,reset=0,req_floor=1,idle=0,door=0,Up=0,Down=1,current_floor=2,max_request=6,min_request=1,requests=02
# Time= 295,clk=1,reset=0,req_floor=1,idle=1,door=1,Up=0,Down=1,current_floor=1,max_request=6,min_request=1,requests=00
# Time= 300,clk=0,reset=0,req_floor=1,idle=1,door=1,Up=0,Down=1,current_floor=1,max_request=6,min_request=1,requests=00
# Simulation finished.
```