Complex Engineering Problem

System Verilog FSM for an Elevator Control System

Submitted by

Mehmood Ul Haq 2021-CE-35

Submitted to

Sir Afeef Obaid

Course

CMPE-421L Computer Architecture

Semester

7th

Date

2nd October 2023

Department of Computer of Engineering University of Engineering and Technology, Lahore

Introduction

Elevators have become a fundamental part of modern urban living, making it easier for people to move through multi-story buildings with convenience and safety. The smooth and reliable operation of elevators is essential to ensure efficient vertical transportation. In this project, we focus on designing a System Verilog Finite State Machine (FSM) that controls an elevator system, addressing the complex demands of multi-story buildings while balancing user requests and safety measures.

The elevator control system we're developing follows a set of key requirements, such as handling floor selection, managing emergency stops, supervising floors with limit sensors, and using an intuitive user interface. Rigorous testing and simulation will ensure the system functions effectively.

Assumptions

These assumptions guide how the elevator control system operates in different situations:

1. **Direction Persistence**
   The elevator will not change direction until all requests in that direction are completed. If it's moving upwards, it will continue serving upward requests until there are none left, and the same for downward movement. This keeps the elevator's operation efficient.
2. **Idle State**
   Once all requests are served, the elevator stops and enters an idle state at the last floor. This prevents unnecessary movement and ensures the elevator is available for new requests.
3. **Door Control Signal**
   Arrival sensors trigger the elevator doors to open for a set time when reaching a destination floor. This ensures that passengers can safely enter and exit the elevator.

These assumptions provide a clear understanding of how the system works and help in designing states, transitions, and test scenarios to ensure the system runs efficiently and safely.

Improved Priority Algorithm

1. Priority Algorithm Overview

We've enhanced the elevator control system with a modified priority algorithm that improves how requests are handled. This system now prioritizes requests based on proximity and direction, surpassing the limitations of the traditional First-Come-First-Serve (FCFS) method.

2. Proximity-Based Prioritization

- **Directional Priority:**
  The elevator prioritizes requests based on the direction it's moving. If it's going up, it will first serve requests for floors higher than its current position. Similarly, it will prioritize lower floors when moving downwards.
- **Proximity Priority:**
  Within the same direction, the elevator prioritizes requests for the nearest floors. This ensures that the most relevant requests are handled first, reducing unnecessary travel and wait times.

3. Efficient Movement Planning

- **Direction Adjustment:**
  The elevator adjusts its direction based on the highest-priority request. For example, if it's moving upward but a higher-priority request is in the opposite direction, it will change course to serve that request.
- **Reducing Unnecessary Stops:**
  By focusing on requests in the same direction and minimizing the distance between them, the algorithm reduces the number of stops, saving energy and time for passengers.

4. Handling Multiple Requests

- **Optimized Order:**
  When there are multiple requests in the same direction, the system optimizes the order in which they're served. This minimizes travel distance and maximizes efficiency.
- **Energy Efficiency:**
  The optimized order reduces elevator movement and energy consumption, contributing to both passenger convenience and environmental benefits.

5. Emergency System Integration

- **Emergency Stop:**
  In case of an emergency, passengers can press the emergency stop button, halting the elevator. Once the situation is resolved, the elevator resumes operation from where it left off, maintaining both safety and efficiency.
- **Resumption After Emergency:**
  After an emergency stop, the elevator resumes normal operation, continuing to serve requests based on the saved direction and status.

Scalable Elevator Control System

Our elevator control system is currently designed for an 8-floor building, but it can easily scale to work with buildings of different heights, ranging from just 2 floors to high-rise buildings with 64 floors or more. This flexibility showcases the system's scalability and adaptability.

1. Adaptive Scalability

The system is not restricted to any particular number of floors. It can be scaled up or down with minimal changes to the code while keeping the core state machine intact.

2. Unified State Table and Diagram

The elevator's state table and transition logic remain consistent, regardless of the number of floors. This makes the system easy to modify, understand, and maintain.

3. Resource Efficiency

Our design balances flexibility with resource efficiency, ensuring the system remains responsive and resource-friendly, even when scaled to handle more floors.

4. Code Maintainability

Despite being scalable, the code remains well-organized and easy to maintain, ensuring that adding or removing floors doesn't introduce complexity.

5. Thorough Testing and Validation

Extensive testing has been done for the 8-floor configuration, proving the system's reliability and robustness. Future scalability tests will ensure it works well with larger or smaller buildings.

6. Universal Applicability

This system is versatile and can be applied to various buildings. Architects and developers can rely on it to meet different requirements, whether a small building or a high-rise.

Future-Ready Design

By optimizing the system for scalability, we are preparing for the evolving demands of vertical transportation. As buildings grow taller and more complex, this future-proof design will adapt without sacrificing performance or efficiency

State Tables

Table 1: Finite State Machine Transitions

Table 1 provides a comprehensive overview of the finite state machine transitions in our elevator control system. It details the current states, input conditions, next states, and the corresponding output actions. This table serves as a vital reference for understanding how the elevator behaves and responds in various 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: Request Management and Prioritization

Table 2 offers a succinct representation of state transitions related to updating elevator requests, as well as managing max_request and min_request variables. It illustrates how the elevator system processes new floor requests and maintains these critical variables, which are essential for optimizing request prioritization and elevator movement. This table simplifies the understanding of how request management operates 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

This section presents comprehensive state diagrams that illustrate the finite state machine of our elevator control system. Alongside the implementation in System Verilog, these diagrams were generated using the Python library pydot. They offer a clear and intuitive visualization of the elevator's operational states and transitions, facilitating a better understanding of the system's behavior and responses under various scenarios.

Diagram 1

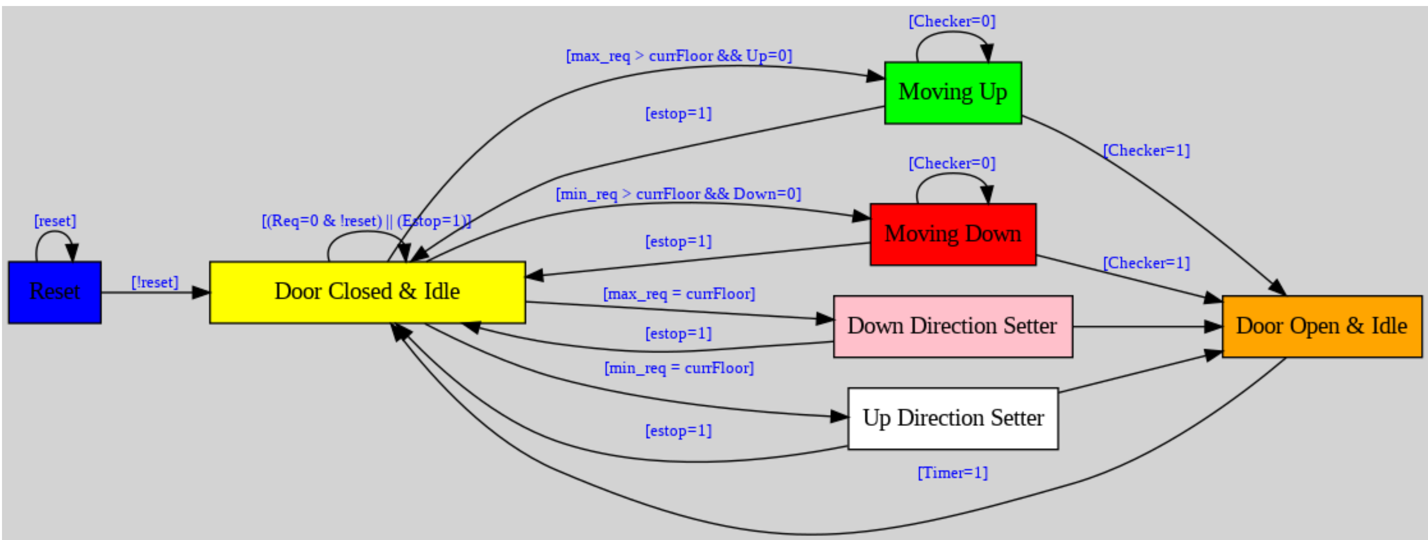
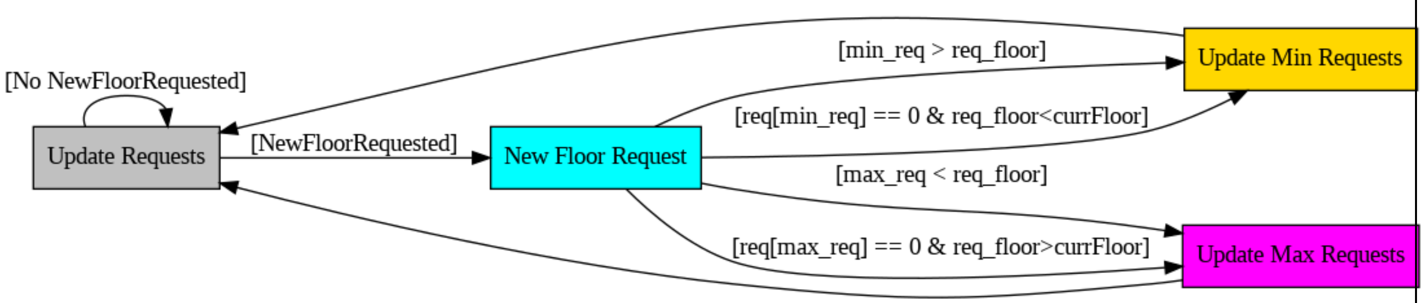


Diagram 2



Design.sv (Part 1)

design.sv

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

Design.sv (Part 2)

design.sv

```
// 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
  // emergency reset
  else if (!emergency_stop && flag)
  begin
    // Emergency stop button is turned off
    emergency_stopped <= 0; // Set emergency stop state
    flag <=0;
  end
  else
  begin
    // Normal operation when not in emergency stop
    if (max_request <= 7)
    begin
      if (min_request < current_floor && Down == 1)
      begin
```

Design.sv (Part 3)

```
        if (min_request < current_floor && Down == 1)
        begin
          // Move down one floor
          current_floor <= current_floor - 1;
          door <= 0;
          idle <= 0;
        end
        else if (max_request > current_floor && Up == 1)
        begin
          // Move up one floor
          current_floor <= current_floor + 1;
          door <= 0;
          idle <= 0;
        end
        else if (req_floor == current_floor)
        begin
             // Open door and handle request
            door <= 1;
            idle <= 1;
        end
        else if (max_request == current_floor)
        begin
          Up <= 0;
          Down <= 1;
        end
        else if (min_request == current_floor)
        begin
          Up <= 1;
          Down <= 0;
        end
      end
    end
  end
endmodule
```

Design.sv Explanation

Module Declaration

- The Verilog module named **"Lift8"** is designed to manage the elevator system. It has several input and output signals that help it function smoothly:
    - **Inputs:**
        - **clk**: The clock signal keeps everything in sync.
        - **reset**: A reset signal to start the elevator system from scratch.
        - **req_floor**: A 3-bit input representing the requested floors (from 0 to 7).
        - **emergency_stop**: A button to immediately stop the elevator in case of an emergency.
    - **Outputs:**
        - **door**: A 2-bit output showing the state of the elevator door (open or closed).
        - **max_request**: A 3-bit output indicating the highest requested floor.
        - **min_request**: A 3-bit output indicating the lowest requested floor.
        - **Up**: A 2-bit output that tells whether the elevator is moving up.
        - **Down**: A 2-bit output that tells whether the elevator is moving down.
        - **idle**: A 2-bit output indicating if the elevator is idle.
        - **current_floor**: A 3-bit output showing which floor the elevator is currently on.
        - **requests**: An 8-bit output representing all floor requests.

Internal Variables

- **door_timer**: This signal manages how long the elevator door stays open or closed.
- **emergency_stopped**: Indicates if the elevator is currently in emergency stop mode.
- **flag**: Helps manage transitions when the elevator is in emergency stop mode.
- **updownflag**: Tracks the previous "Up" and "Down" directions during an emergency reset.

Request Handling

- An **always block** updates the elevator's request list based on what floors are requested.
- It also keeps track of the **max_request** (highest requested floor) and **min_request** (lowest requested floor) to efficiently manage the requests.

Current Floor Handling

- Another **always blocks** takes care of what happens when the elevator reaches a requested floor:
    - When the elevator arrives at a requested floor, it switches to idle mode, opens the door,

clears the request, and starts the door timer.

State Machine for Lift Control

- The main logic of the elevator control system is implemented in an **always @(posedge clk)** block, which represents a synchronous state machine.
- It manages different states and transitions:
  - **Reset**: Initializes the elevator's state.
  - **Emergency Stop**: Activated when the emergency stop button is pressed.
  - **Emergency Stop (During Stopped State)**: The elevator stays stopped until the emergency stop button is released.
  - **Emergency Reset**: Clears the emergency stop state when the button is released.
  - **Normal Operation**: Controls the elevator's movement based on requests and the current floor.
  - **Opening Door**: Opens the door when the elevator arrives at a requested floor.
  - **Other Cases**: Manages the elevator's movement when going up, down, or staying idle based on requests.

Usage of min_request and max_request

- **min_request** tracks the lowest floor with pending requests, while **max_request** tracks the highest.
- These variables help the elevator move efficiently:
  - If the elevator is going up and reaches the **max_request**, it will change direction and start moving down to serve the lower floors.
  - Similarly, if it's going down and hits the **min_request**, it will switch to go back up for higher floors.
- This ensures that the elevator prioritizes requests based on their locations, reducing unnecessary travel and wait times.

Testbench (Part 1)

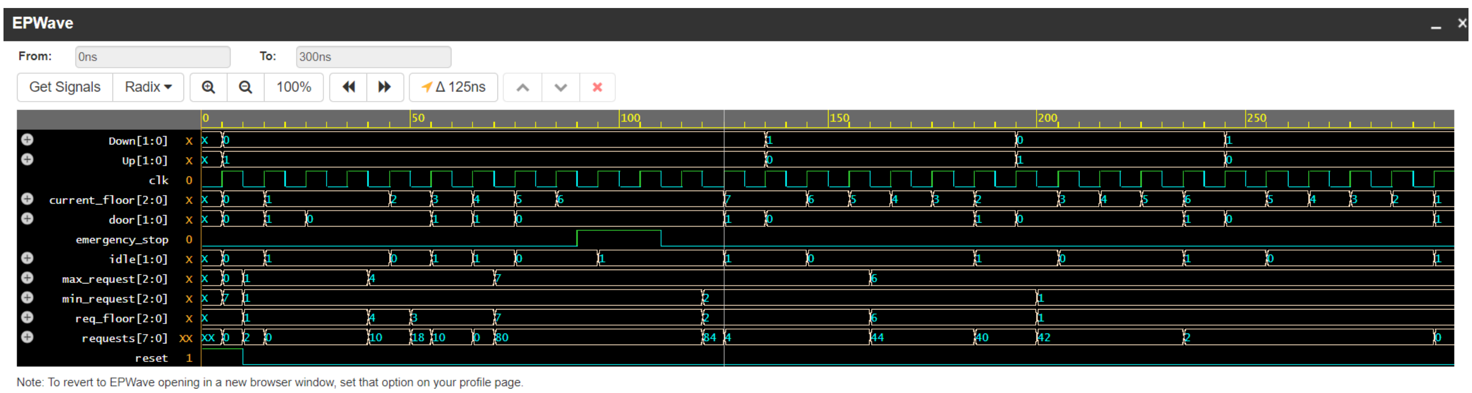
testbench.sv

```
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;
    req_floor = 2; // Request floor 2
    #40;
    req_floor = 6; // Request floor 6
    #20;

    #20;
    req_floor = 1;
  end
```

Testbench (Part 2)

```

  initial begin
  $display("Starting simulation...");

$monitor("Time=%t,clk=%b,reset=%b,req_floor=%h,idle=%h,door=%h,Up=%h,Down=%h,current_floor=%h,max_request=%h,min_request=%h,requests=%h",
    $time, clk, reset, req_floor, idle, door, Up, Down, current_floor, max_request, min_request, requests);
  // Run the simulation for a sufficient duration
  #305; // Adjust the simulation time as needed
  $display("Simulation finished.");
  $finish;
end

// C
  always #5 clk = ~clk;
endmodule

```

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.
```