American University of Beirut

Maroun Semaan Faculty of Engineering and Architecture

Department of Electrical and Computer Engineering

EECE 320: Digital Systems Design

Verilog Project

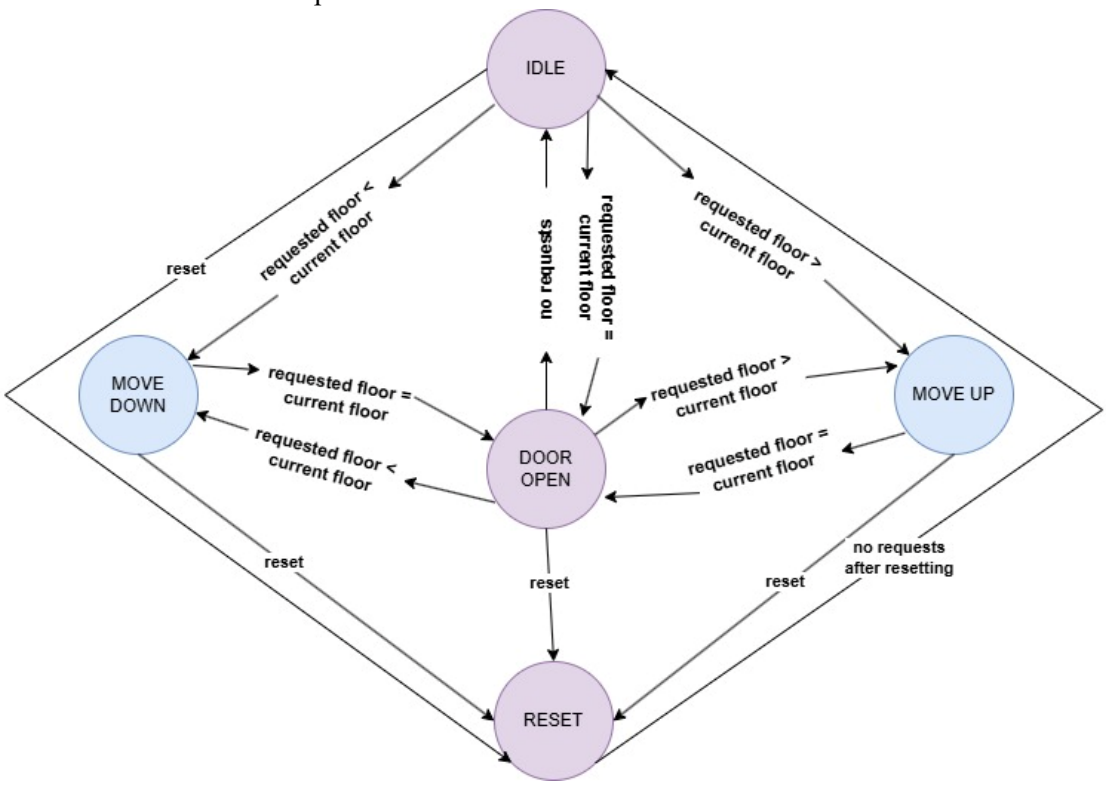
Elevator Controller

Professor: Mazen Saghir

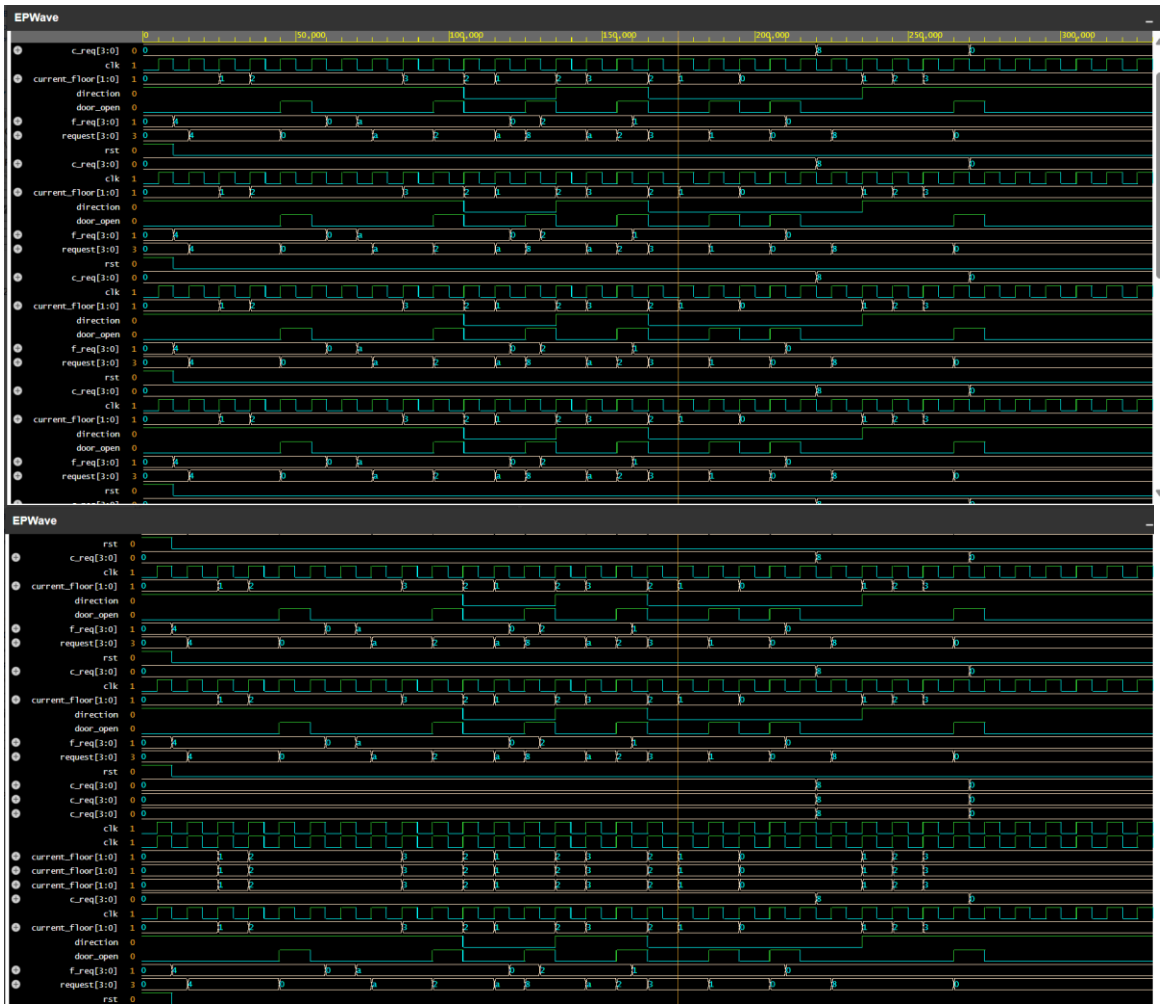
Ali Rida Awad (202401137) and Joud Senan (202420888)

I. State Diagram:

- **Idle State:** If we are in the idle state, it means that there are no requests for the elevator at that time; therefore, the elevator is stopped at the current floor with its door closed. This state changes to Move Up, Move Down, or Door Open depending on the new requests that arise.
  Let us go over cases:
  - Case 1: Requesting a floor above the current floor => Move Up
  - Case 2: Requesting a floor below the current floor => Move Down
  - Case 3: Requesting the current floor => Door Open
- **Move Up:** The direction bit is set to 1. The elevator moves up a floor per clock cycle, and changes to Door Open if the current floor is the requested floor. If no more requests, go back to the idle state.
- **Move Down:** The direction bit is set to 0. The elevator moves down a floor per clock cycle, and changes to Door Open if the current floor is the requested floor. If no more requests, go back to the idle state.
- **Door Open:** If the door is opened, the door_open bit is a 1; otherwise, it is a 0. If the current floor is the same as the requested floor, the bit is set to 1 and the door is opened. The door remains open for a clock cycle and clears that request. This state changes to Move Up, Move Down, or Idle depending on the requests.
- **Resetting:** When rst is asserted, this means that it is set to 1 since it is active-high, the current_floor becomes the ground floor, request are cleared (0000), direction is set to 1 (moving up), and the door is closed (door_open is set to 0). After resetting, the elevator is in the idle state from which we can then change to one of the moving states or door open state.



II. Screenshots of simulation waveforms:



III. Explanation of the Design:

In our Verilog code for the FSM design of the elevator, we implemented it as follows:

- **Two helper methods:** “above” and “below”

  - **“above”:** outputs 1 if there is a pending request for a floor above the current floor through comparing the request vector with a bitmask that excludes floors below the current floor as well as the current floor.
  - **“below”:** outputs 1 if there is a pending request for a floor below the current floor through comparing the request vector with a bitmask that excludes floors above the current floor as well as the current floor.

- **The main module "elevator_fsm":**

  - We join the requests, f_req and c_req into one request vector that contains the combined requests.
  - Overall, this module will track the requests and behave accordingly through checking for current_floor, direction, door's state (open when reaching the requested floor), and prioritizing which requests to handle first in which it uses the helper methods discussed above.

- **Behavior:**

  - **As discussed earlier:** Whether the requests are from inside the cabin or from the outside, it does not matter. The elevator need not know this detail. Therefore, all requests (f_req and c_req) are handled as one vector combining them called "requests". Thus, they are OR-ed together to create the new requests vector handling all requests, and the requests of the current floor are cleared. Note that if there are previous requests available, they are also OR-ed with the new requests. Therefore, the final requests vector has old and new requests or more generally all pending requests.
  - **Case1:** If the current floor is the requested floor, we open the door. Therefore, door_open is set to 1 and the request is cleared (the bit corresponding to this request is now 0).
  - **Case2:** We introduced the helper methods above. Here is where we use them! In this case, we need to use priorities. If all requests are above the current_floor, we move up. **BUT!** If we have both below and above, we check the direction. If we are moving up and there are "above" requests, then serve these first!
  - **Case3:** Similarly, if we have "below" requests only, we serve them directly. Otherwise, check the direction. If we are moving down, then serve these below first, then we flip the direction and serve other requests.
  - **Case4:** If there are no requests, the elevator is now in its idle state.

- **Shedding Light on the Functions Above:**

  Our module ensures that whenever a case is handled, the current_floor request is deleted from the request vector which ensures that inconvenience and redundant servicing won't happen.

- **Real-Life Scenario:**

  Let us say that I entered an elevator, where I am pressing the only request for this elevator. My request would be added to the request vector, then apply the checking scenario. This is the only request, so the elevator will head to it directly, open the door when it reaches the requested floor (one clock cycle) then clears the request. Now, no other requests are available so the door closes, remains on the current floor in an idle state.

  Let us change the scenario a bit, I requested the elevator in a busy building, multiple requests are happening at the same time. The elevator starts handling the requests in the same direction that it is following then switches its direction when done with the requests in that direction.

This is exactly what our project is doing!

- **Test Bench:**
  Below is the output of simulating our code with its test bench on EDA playground. The testbench verifies the correctness of our FSM elevator under various scenarios.
  We set the clock signal to toggle every 5-time units as you can see in the figure below.

```
  0 | Floor: 00 | Requests: 0000 | Direction: 1 | Door Open: 0
 15 | Floor: 00 | Requests: 0100 | Direction: 1 | Door Open: 0
 25 | Floor: 01 | Requests: 0100 | Direction: 1 | Door Open: 0
 35 | Floor: 10 | Requests: 0100 | Direction: 1 | Door Open: 0
 45 | Floor: 10 | Requests: 0000 | Direction: 1 | Door Open: 1
 55 | Floor: 10 | Requests: 0000 | Direction: 1 | Door Open: 0
 75 | Floor: 10 | Requests: 1010 | Direction: 1 | Door Open: 0
 85 | Floor: 11 | Requests: 1010 | Direction: 1 | Door Open: 0
 95 | Floor: 11 | Requests: 0010 | Direction: 1 | Door Open: 1
105 | Floor: 10 | Requests: 0010 | Direction: 0 | Door Open: 0
115 | Floor: 01 | Requests: 1010 | Direction: 0 | Door Open: 0
125 | Floor: 01 | Requests: 1000 | Direction: 0 | Door Open: 1
135 | Floor: 10 | Requests: 1000 | Direction: 1 | Door Open: 0
145 | Floor: 11 | Requests: 1010 | Direction: 1 | Door Open: 0
155 | Floor: 11 | Requests: 0010 | Direction: 1 | Door Open: 1
165 | Floor: 10 | Requests: 0011 | Direction: 0 | Door Open: 0
175 | Floor: 01 | Requests: 0011 | Direction: 0 | Door Open: 0
185 | Floor: 01 | Requests: 0001 | Direction: 0 | Door Open: 1
195 | Floor: 00 | Requests: 0001 | Direction: 0 | Door Open: 0
205 | Floor: 00 | Requests: 0000 | Direction: 0 | Door Open: 1
215 | Floor: 00 | Requests: 0000 | Direction: 0 | Door Open: 0
225 | Floor: 00 | Requests: 1000 | Direction: 0 | Door Open: 0
235 | Floor: 01 | Requests: 1000 | Direction: 1 | Door Open: 0
245 | Floor: 10 | Requests: 1000 | Direction: 1 | Door Open: 0
255 | Floor: 11 | Requests: 1000 | Direction: 1 | Door Open: 0
265 | Floor: 11 | Requests: 0000 | Direction: 1 | Door Open: 1
275 | Floor: 11 | Requests: 0000 | Direction: 1 | Door Open: 0
```

- **Conclusion:** Successfully, the testbench shows that this FSM elevator handles real-life scenario cases effectively including having single requests, multiple requests, requests in both directions, requesting after an idle state, and reaching idle state.