VHDL Elevator Controller Report

Spencer Reitman

I. Intro

I started my elevator project on paper. I knew my design would eventually be capable of as many floors as desired, so I had to draft a plan of how all the components would be dynamic enough to allow this. I didn't want to start with a simple elevator where there were only two floors and one button on each floor; that would just be too simple and too much would have to be gutted to get my final design. The main questions I asked myself were

- How will the buttons on each floor work?
- How will the riders choose what floor they want to go to?
- How will the elevator know what floor its on?
- How will the elevator choose what floor to go to given multiple inputs?

By finding answers to these questions, my elevator design would become much more clear.

II. Design

The design of my elevator is structural at the top level entity and behavioral for the lower level entities. You can see the structure of the netlist structure of the top level in figure 2 at the end of this document. The entities are

1. Button registers to hold active floor requests (elevator_buttons_reg.vhdl)
2. Floor shift-register to keep track of the elevator's current floor (elevator_tracker.vhdl)
3. Hybrid Moore/Mealy finite state machine (fsm)(elevator_fsm.vhdl)
4. FSM output module to map states to outputs (elevator_fsm_out.vhdl)
5. Counter to keep track of door opening (elevator_door_counter.vhdl)
6. Counter to keep track of elevator moving (elevator_move_counter.vhdl)
7. Counter to keep track of door holding open (elevator_wait_counter.vhdl)
8. Top level module to connect everything (elevator_controller.vhdl)

The button inputs to the top level module are assumed to be on for a single clock cycle then turn off. The buttons are sent in a one-hot encoding for each floor and stored in that way as well. Whenever a new button input comes in, all the active floor buttons are stored in the button registers, one for external elevator buttons and one for internal elevator buttons. The button registers connect to the fsm.

The fsm has 10 states: idle, opening_door, door_opened, waiting, closing_door, door_closed, start_moving_down, moving_down, start_moving_up, moving_up. See figure 1 for the state machine itself. When the fsm receives a floor request from the button registers, it will first decide what floor to go to based on the current direction priority and whether there are internal vs external button requests. If there are no floor requests the direction priority will alternate between up and down. When a floor request comes from the internal buttons, they are prioritized over the external buttons. The target floor is calculated based on the highest or lowest floor based on the direction priority.

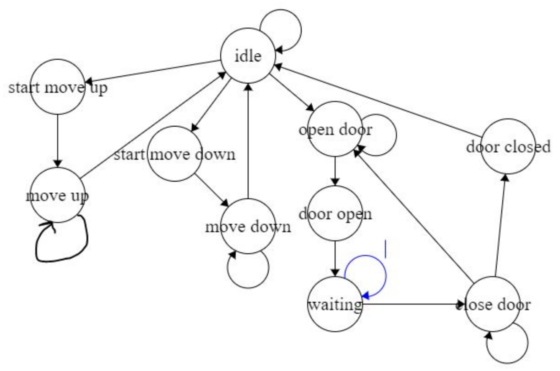

Figure 1.

The elevator starts in the idle state and moves to the start-move-up or start-move-down depending on the target floor. If the current floor is requested, it immediately begins opening the door. Once the door is open, there's a waiting period to hold the door open before closing the door. If the button for the current floor is pressed while the door is closing, it will start reversing the door open again.

If the elevator takes a floor request below the elevator's current position, the next direction priority will be up and vice versa for a floor request above.

Once the target floor is set, the elevator will continue in that direction until it reaches the target floor. The elevator moves using a timer that starts when in the moving state, and the timer will indicate when it reaches a floor. When the elevator starts moving, if it reaches a floor that is also requested besides the target floor, it will open then close the doors before continuing to the target floor. The doors open based on a door timer, and hold open based on a separate timer. Once fully open, the current floor request will be cleared, then the doors will begin closing. The current elevator floor is tracked based on a one-hot encoding, which is shifted right or left to indicate the elevator went up or down a floor respectively.

III. Results

I created a testbench with a semi-rigid test case structure. Each test case continues from the previous one by setting various external and internal buttons and waiting for each floor in the expected order. If there is an infinite loop, then the elevator did not take the expected path, thus an error. I also verified the operation of the elevator by outputting the elevator's current floor, door state, and state of motion to a file and the transcript of modelsim. This, alongside the waveform viewer gave me a clear indication of where things might be going wrong so I could fix them.

There weren't any issues with my design, but there are improvements that could be made, which I will discuss later.

IV. Analysis

My design works for a single elevator with N floors up or down. It will never be caught in a situation where the elevator is moving with the doors open or moving while opening/closing the doors because of the way my state machine operates. It will never lock out a floor because of the direction priority switching, and it will hit the maximum number of floors in a direction because of the way it chooses the target floor.

V. Conclusion

A few improvements that could be made are adding an emergency stop and a second elevator. To add the emergency stop, I would most likely create an extra state and whenever the emergency stop is high, whatever the current state is will go to that new state. The old state will be stored in a register and the counters will also halt until it resumes to the state before it went into emergency mode. To implement multiple elevators, I would add two signals that synchronize the elevators. The first wire would send the other elevator's current floor and the second wire would send the other elevator's target floor. That way if one elevator is going up, for example, and a request comes to a floor below the elevator's current floor, the other elevator can get to it.

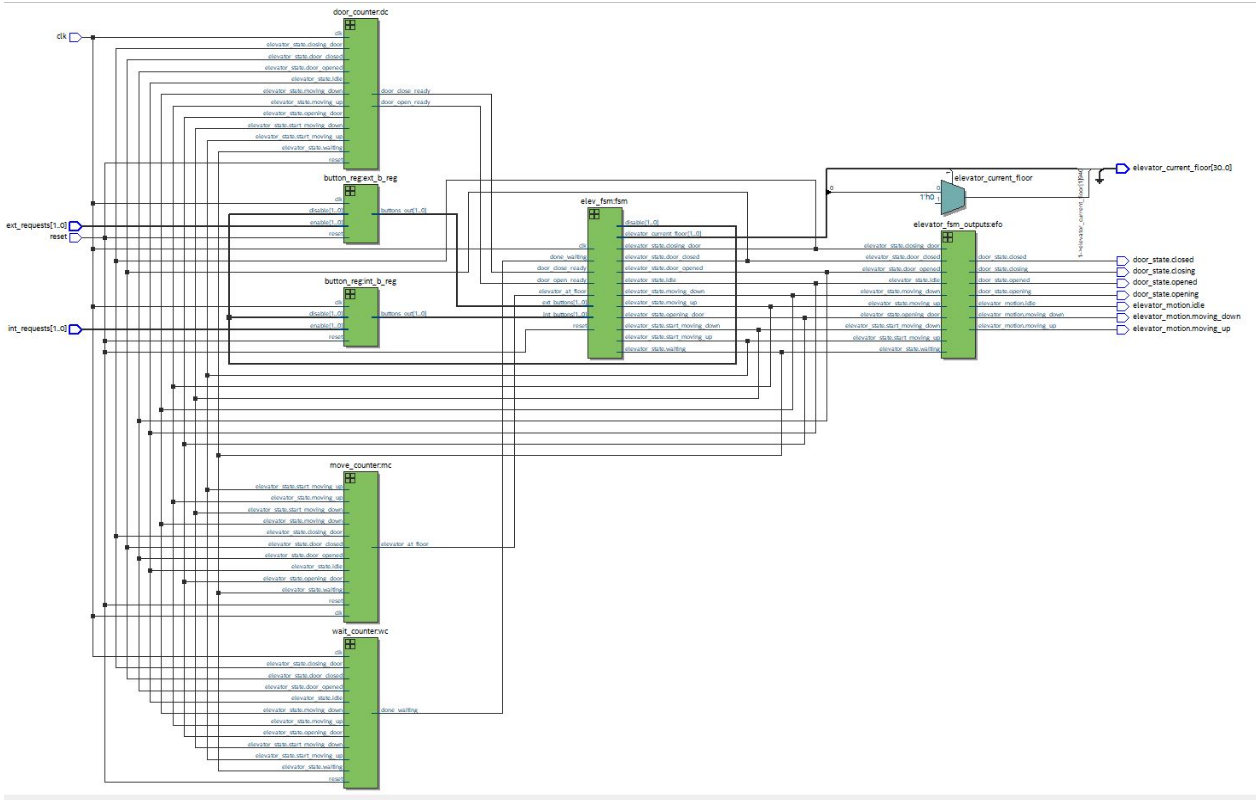

Figure 2.

Netlist view of the top level structure.