Elevator Controller

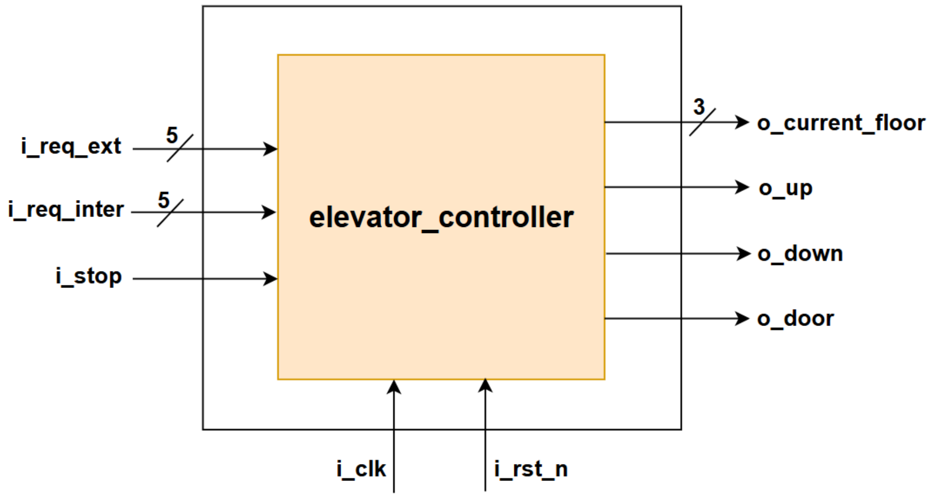
System Description

This assignment involves designing the architecture and writing RTL for an elevator controller in a building with five floors (a ground floor and four upper floors).

The elevator should manage multiple requests simultaneously by assigning priority. The elevator should prioritize requests in its current direction of movement, servicing requests in the upward direction while moving up, and downward while moving down. If the elevator is idle (not moving), priority is given to the first floor requested. If multiple requests are received at the same time while the elevator is idle, priority is given to the highest floor among the requests.

The elevator includes a stop feature: a push button inside the elevator allows passengers to stop it immediately. When pressed, the elevator halts and retains all input, output, and control states. Pressing the button again releases the stop, allowing the elevator to resume from where it left off, with all signals and states preserved for a seamless continuation.

Floor requests can be made via push buttons located on each floor or from buttons inside the elevator. The elevator takes 2 sec. to travel between two consecutive floors. The door stays open for 2 sec. on the floor.



For simplicity, we assume both the external and internal request signals are 5-bit vectors, indexed from 0 to 4 to represent each floor. Each bit corresponds to a specific floor: bit 0 represents the ground floor, bit 1 represents the first floor, and so on. For example, if external requests come from the first and third floors, i_req_ext would be 5'b01010. Similarly, if internal requests come from the second and ground floors, i_req_inter would be 5'b00101. The combined request vector would then be 5'b01111. Requests are granted in the prioritized order specified above.

Signals

Table 1

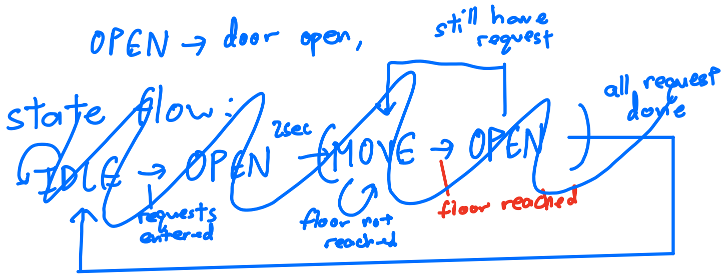
Signal	Width	Direction	Description
i_clk	1	Input	Positive edge system clock (50 MHz)
i_rst_n	1	Input	Negative edge asynchronous system reset
i_req_ext	5	Input	External input requests where each bit represents the push button outside the elevator for its corresponding floor. When a bit is set to 1, it indicates a request from that floor.
i_req_inter	5	Input	Internal elevator requests where each bit represents a push button inside the elevator for a specific floor. When a bit is set to 1, it indicates a request for that floor.
i_stop	1	Input	A stop button inside the elevator allows People to immediately halt the elevator.
o_current_floor	3	output	An output indicating the current floor number. For example, if the elevator is on the third floor, o_current_floor will be set to 3'd3.
o_up	1	output	1: elevator is moving up 0: elevator is moving down or not moving
o_down	1	output	1: elevator is moving down 0: elevator is moving up or not moving
o_door	1	output	1: elevator door opens 0: elevator door closed

Deliverables

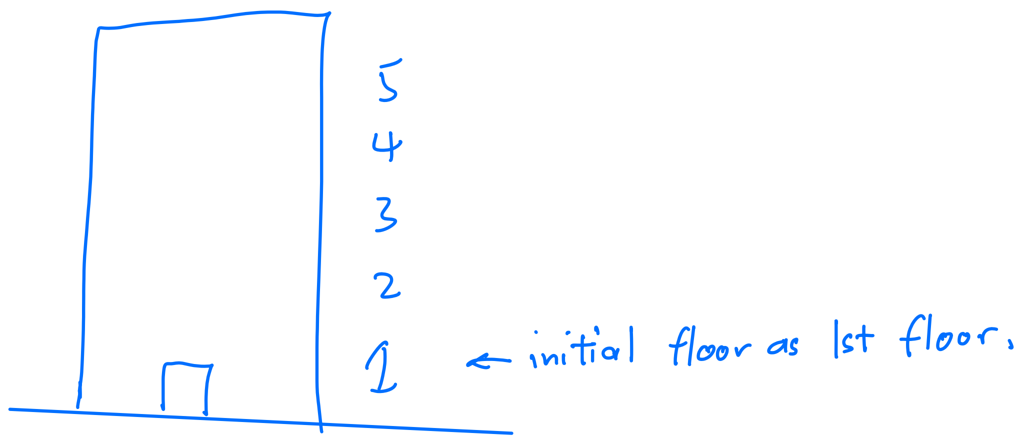
You have to submit a pdf or zip file that contains:

- A PDF file that contains:
  - Architecture of the design which includes how sub-blocks communicate with each other.
  - Description of the operation between blocks and any existing FSM
  - Simulation waveform that shows the module is working correctly.
- Project folder contains (zip file or google drive link):
  - RTL codes of the system.
  - Testbench code of the system.

for draft used, details at next page

Module list	Function (Brief)
↳ 2 seconds prescaler	convert 50MHz clk to 2second pulse for up down controller module. draft pin : clk, i_rst_n, i_stop, out
↳ up down controller	an ASM that controls elevator status. always track its current floor (initial = 1), draft state : IDLE → when no request, waiting for request, door closed MOVE → decide go up or down by comparing . OPEN → door open, state flow:  IDLE OPEN MOVE OPEN 2sec requests entered floor not reached floor reached still have request all request done
↳ priority controller	~~draf state:~~ ~~IDLE: when no request~~ ~~UP_Priority:~~ ~~DOWN_Priority:~~

Remark: floor assigned



Module explaination and variables

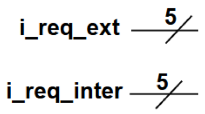
Priority controller: It is a combinational circuit with an ASM and a floor_reached program.

a) State variable:

- IDLE : 00 bit
- UP_PRIORITY : 01 bit
- DOWN_PRIORITY : 10 bit

b) Important variables

- [4:0] request_reg → the OR of the



. (updating rate = clk)

  → store the request

  → keep updated synchronously.

- [4:0] priority_value → determined based on the request_reg [5:0]. (updating rate = clk)

- [2:0] floor_next → the value which define as the target floor (updating rate = clk)
  → eg: if priority_value = 0b00100, // 3rd floor
  floor_next = 0b11. //value of 3
  → If there is no request, floor_next = 0.

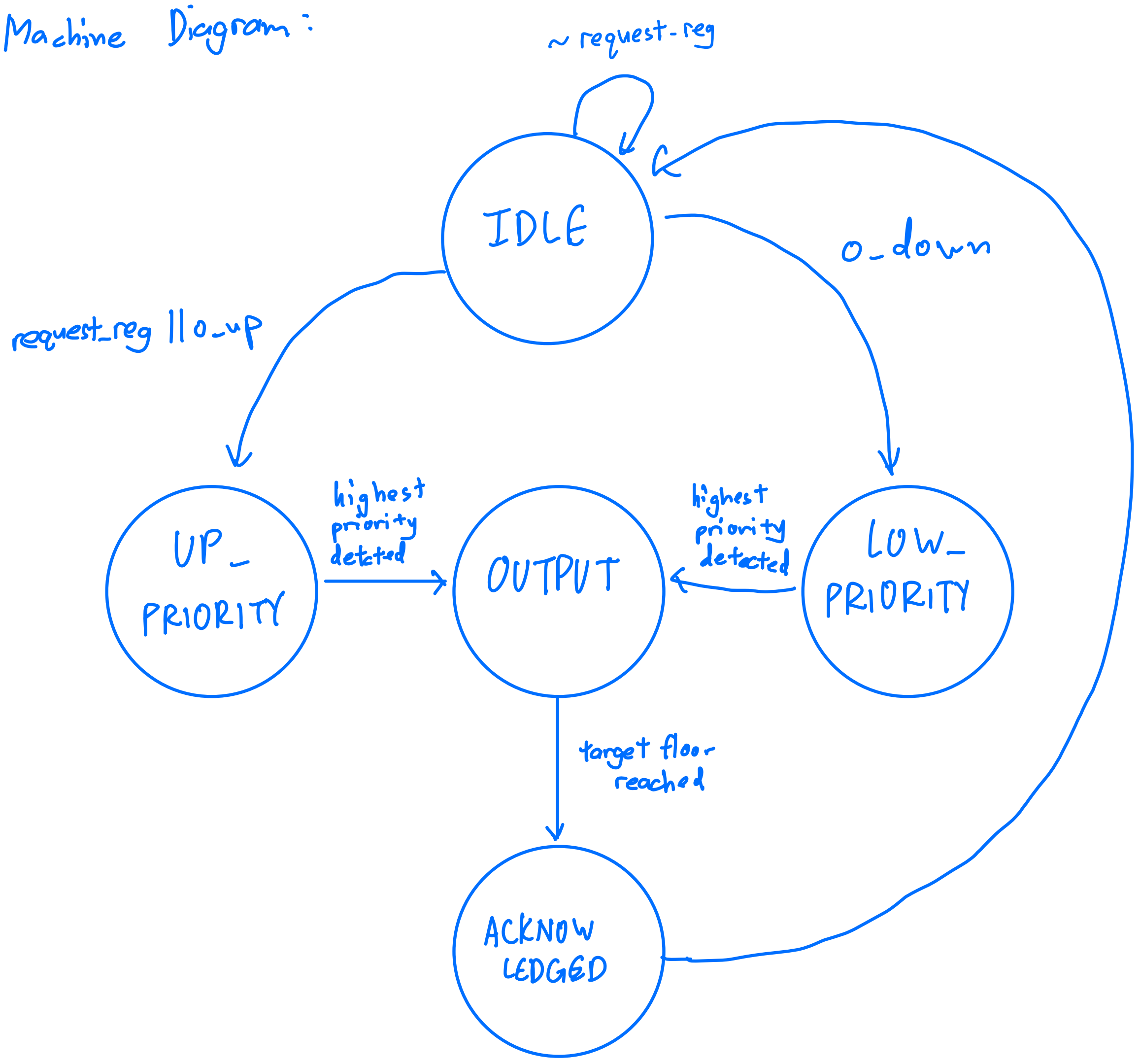
- halt → the boolean which controlled by i_stop.
  → when high, the request_reg is unable to be updated. (and states)

- [2:0] current_floor_reg

- priority_cnt → the cnt which counted the number of shift progressed
  (plz refer to "priority algorithm")

- rotate_shift_flag → consists only one high bit in the 5 bit size of it.
  (plz refer to "priority algorithm")

c) State Machine Diagram:



state description:

IDLE: tracks for requests, assign which priority (UP or DOWN) is used.

UP_PRIORITY: perform up priority algorithm. } (plz refer to "priority algorithm")

DOWN_PRIORITY: perform down priority algorithm. }

OUTPUT: convert priority floor to integer (in binary form) and send to "up-down" module.

ACKNOWLEDGED: remove the target floor request when target floor reached.

d) Priority Algorithm

↳ The algorithm is determined based on the elevator direction, and the distance between current floor to target floor.

↳ from project requirement, the floor assignment :

$$\text{request\_reg} = 5'b\ \underset{\text{5th floor}}{x_4}\, x_3\, x_2\, x_1\, \underset{\text{1st floor}}{x_0}$$

5th floor . . . 1st floor

↳ The priority algorithm will conducted using rotate shifting.

↳ Eg: ① UP_PRIORITY by using left rotate shifting.

```
rotate_shift_flag = (rotate_shift_flag << 1)| (rotate_shift_flag >> 4) ;
        //if rotate shift flag = 5'b10000,
        // the new value = 5'b10000 << 1 | 5'b10000 >> 4
                       = 5'b00001 #
```

② track with request-reg using bitwise AND (&).

```
if (rotate_shift_flag & request_reg)
    state_next = OUTPUT;
// If rotate_shift_flag and request_reg has high at one of the same bits, the condition
// will not equal to 0.
```

↳ The DOWN_PRIORITY is performed using right rotate shifting.

e) ACKNOWLEDGED state

↳ To ensure only the current floor bit is cleared, the following logic equation is used:

```
request_reg = (request_reg & ~(current_floor_bit));
```

↳ eg: If there's full request (request_reg = 5'b11111), after it reaches 3rd floor:

```
request_reg = ( 5'b11111 & ~(5'b00100))
            = (5'b11111 & 5'b11011)
            = 5'b11011 #   // target floor is cleared, and ready to be requested again.
```

up_down

a) States variable :

↳ IDLE : 00 bit

↳ MOVE : 01 bit

↳ OPEN : 10 bit

b) important variables

↳ [2:0] current_floor_reg : indicates the elevator floor, with initially value = 1.

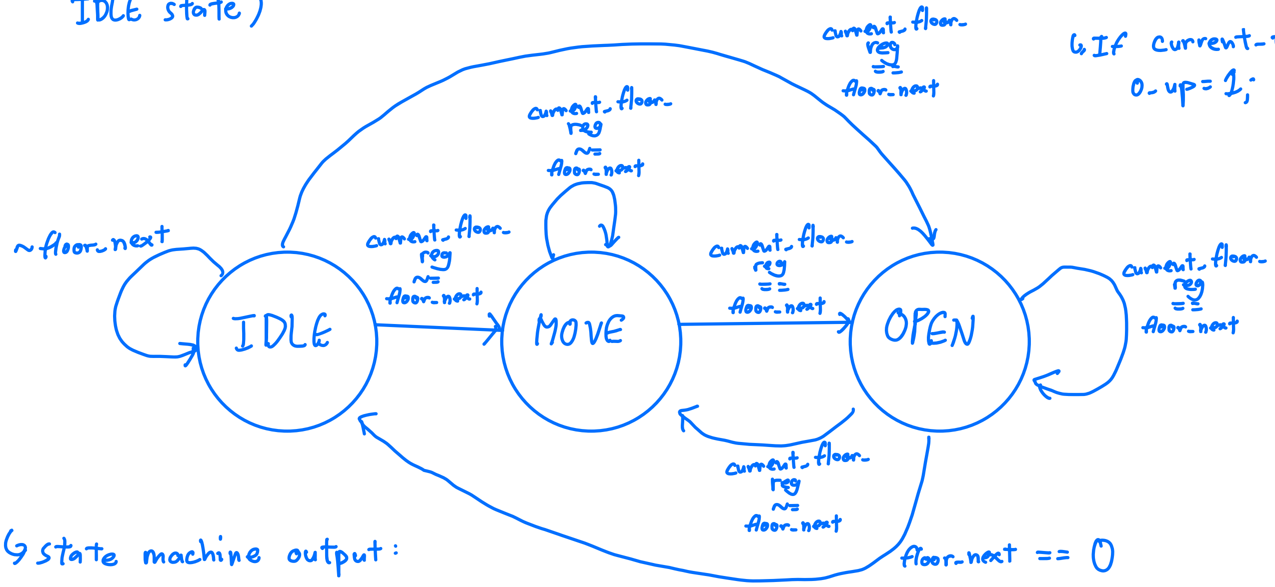
↳ [2:0] floor_next : output target floor value from prority controller.

↳ halt : - the boolean which controlled by i_stop.

- stop update the states, hence the output should also stop updating.

c) State machine diagram :

(Note : the state machine below is updated every 2 seconds, except IDLE state)



~floor_next
IDLE
current_floor_reg ~= floor_next
MOVE
current_floor_reg ~= floor_next
current_floor_reg == floor_next
OPEN
current_floor_reg == floor_next
current_floor_reg == floor_next
current_floor_reg ~= floor_next
floor_next == 0

↳ state machine output :

↳ IDLE : o_door = 0 , floor_reached = 0

↳ MOVE : o_door = 0 , floor_reached = 0

↳ OPEN : o_door = 1 , floor_reached = 1

d) assigning o_up and o_down

↳ If current_floor_reg > floor_next
o_down = 1 ; //going down

↳ If current_floor_reg < floor_next
o_up = 1; //going up