Lebanese American University

LAU School of Engineering

Digital Systems Lab

Section 33

Laboratory manual 5

Nour Kachmar: 201902597

Youssef Kourani: 202004265

18/11/2023

Table of Contents

List of Figures

List of Tables

Introduction

In this lab, we will create an elevator by first drawing a state diagram, then implementing it in code. Moreover, we will create the necessary pulse generator according to our state diagram needs. We will also make use of the LCD to display the current floor and the state of the door if opened or closed.

Experiment

State Diagram

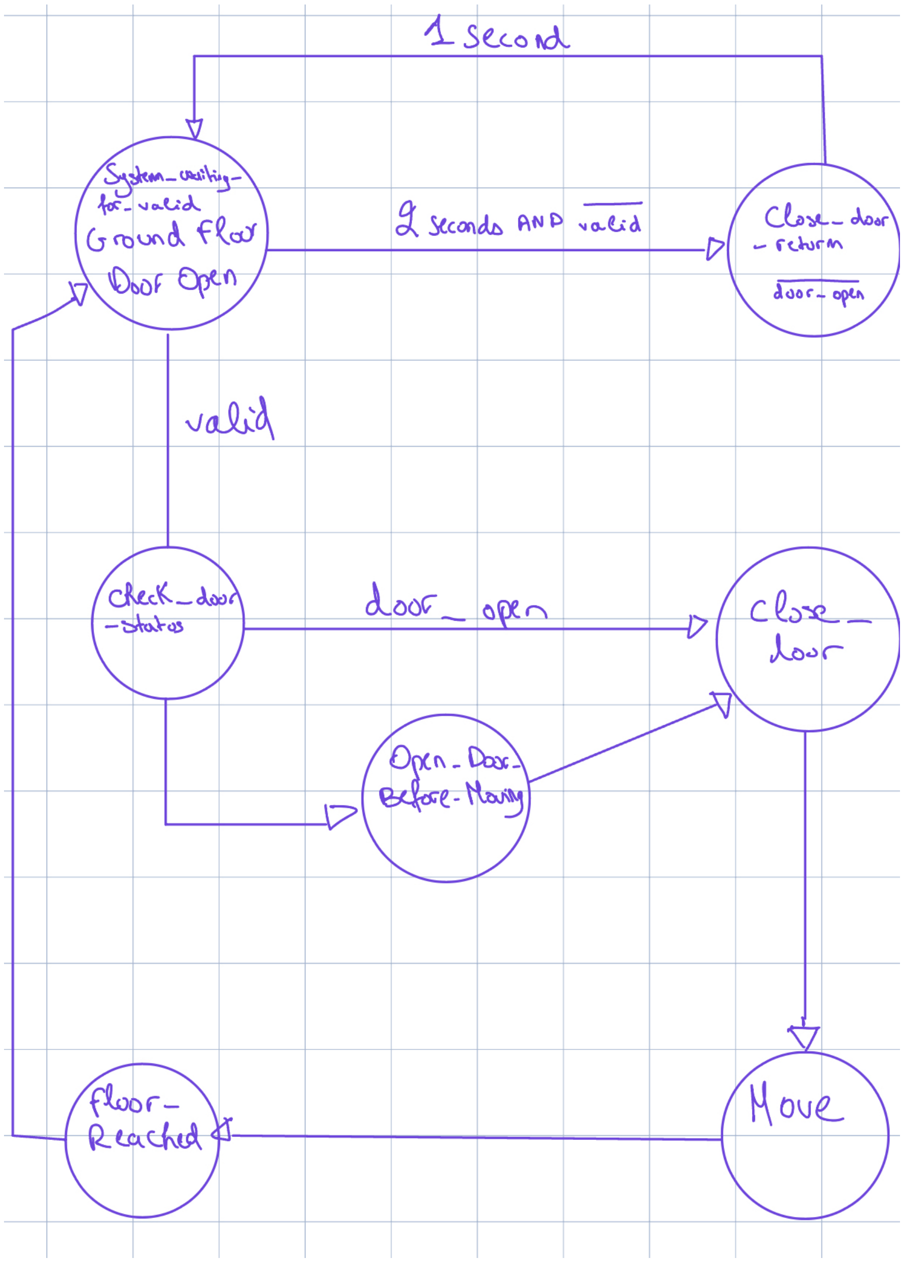

1 second
System_waiting_for_valid Ground floor Door Open
2 seconds AND $\overline{valid}$
Close_door_return $\overline{door\_open}$
valid
Check_door_status
door_open
Close_door
Open_Door_Before_Moving
Move
floor_Reached

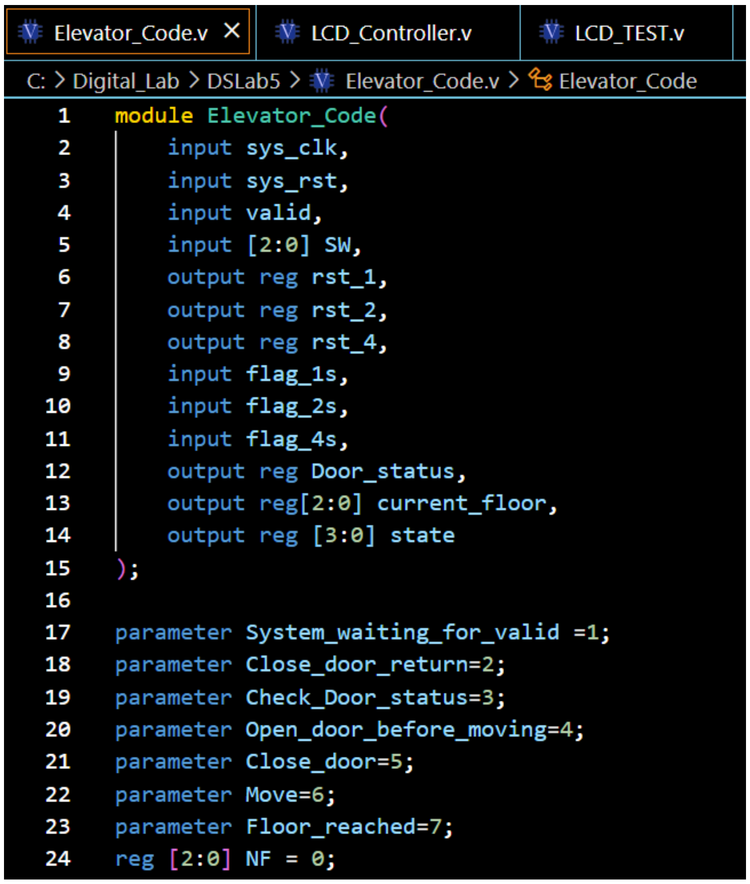
Figure 1: State Diagram of the elevator

In this state diagram represented by figure 1, we can see 7 different states that orchestrate the behavior of our elevator. We will dig dipper into each state when analyzing the code in the following.

Elevator Code

We will dissect the elevator's code into multiple portions for scalability and clarity.

Module Name and Variables used



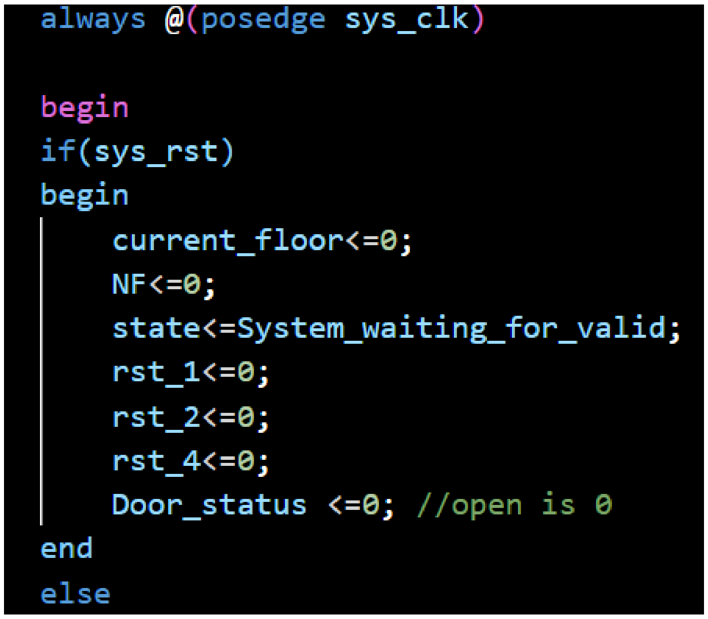
```verilog
module Elevator_Code(
    input sys_clk,
    input sys_rst,
    input valid,
    input [2:0] SW,
    output reg rst_1,
    output reg rst_2,
    output reg rst_4,
    input flag_1s,
    input flag_2s,
    input flag_4s,
    output reg Door_status,
    output reg[2:0] current_floor,
    output reg [3:0] state
);

parameter System_waiting_for_valid =1;
parameter Close_door_return=2;
parameter Check_Door_status=3;
parameter Open_door_before_moving=4;
parameter Close_door=5;
parameter Move=6;
parameter Floor_reached=7;
reg [2:0] NF = 0;
```

Figure 2: I/O and parameters used

Figure 2 represents the input and outputs used and the parameters which represent the state. Note that the parameter (states) are self-explanatory as well as for the inputs and outputs.

Reset Option



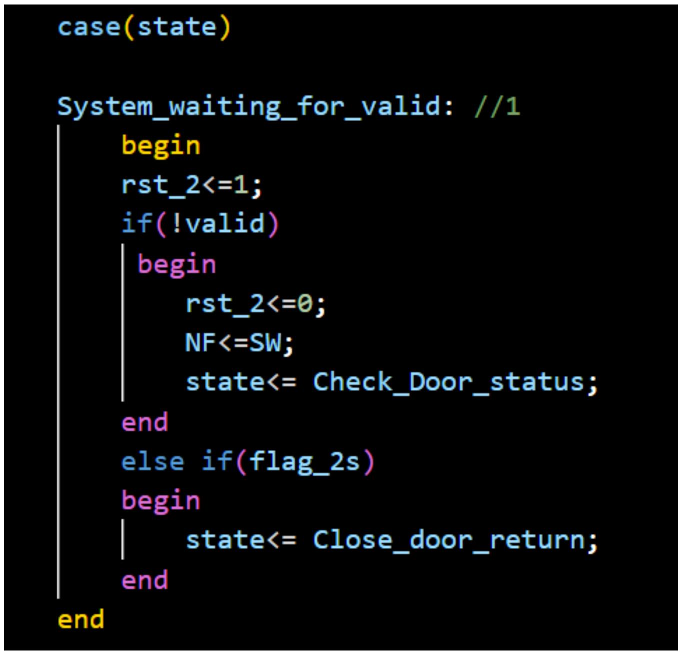
```
always @(posedge sys_clk)

begin
if(sys_rst)
begin
    current_floor<=0;
    NF<=0;
    state<=System_waiting_for_valid;
    rst_1<=0;
    rst_2<=0;
    rst_4<=0;
    Door_status <=0; //open is 0
end
else
begin
```

Figure 3: Reset Option in Code

Figure 3 shows a reset option available when the switch corresponding to the reset is active high. We reset the next floor to go to, the current floor to the ground floor, the door to open and all the resets corresponding to the pulses to 0. Also, the system goes to state 1 which is System_waiting_for_valid.

Case: System_waiting_for_valid



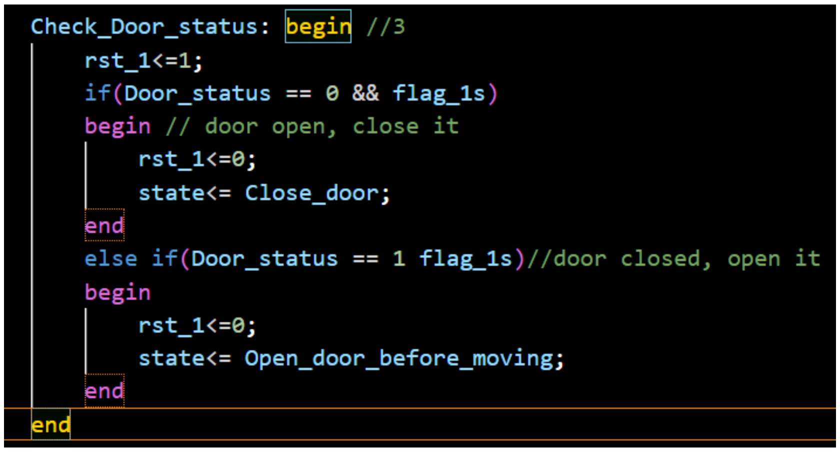
```
case(state)

System_waiting_for_valid: //1
    begin
    rst_2<=1;
    if(!valid)
     begin
        rst_2<=0;
        NF<=SW;
        state<= Check_Door_status;
    end
    else if(flag_2s)
    begin
        state<= Close_door_return;
    end
end
```

Figure 4: Case when System is waiting for valid

In figure 4, we start the 2 seconds counter by giving rst_2 a value of 1. If valid is pressed (works on active low because it's a push button), we go to check the door's status and stop the 2 seconds couter by giving rst_2 a value of 0. NF takes the value of the switch inputs.

Case: Check_door_status

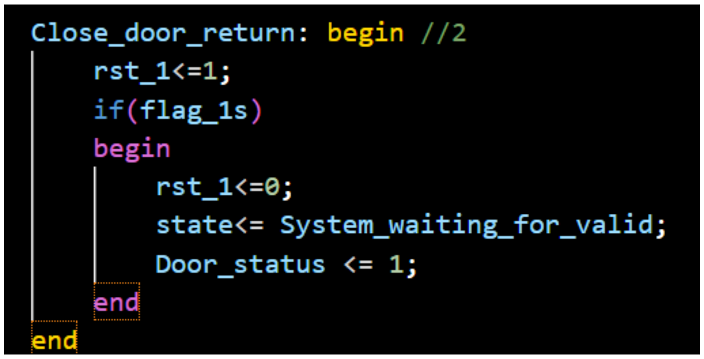


```
Check_Door_status: begin //3
    rst_1<=1;
    if(Door_status == 0 && flag_1s)
    begin // door open, close it
        rst_1<=0;
        state<= Close_door;
    end
    else if(Door_status == 1 flag_1s)//door closed, open it
    begin
        rst_1<=0;
        state<= Open_door_before_moving;
    end
end
```

Figure 5: System in Check_Door_status

Figure 5 shows that we start the counter for 1 second by putting a 1 in rst_1. Then, we check the door status. If it is open and 1 second has passed, we stop the 1 second counter and close the door. We do the opposite in the second else if condition.

Case: Close_door_return

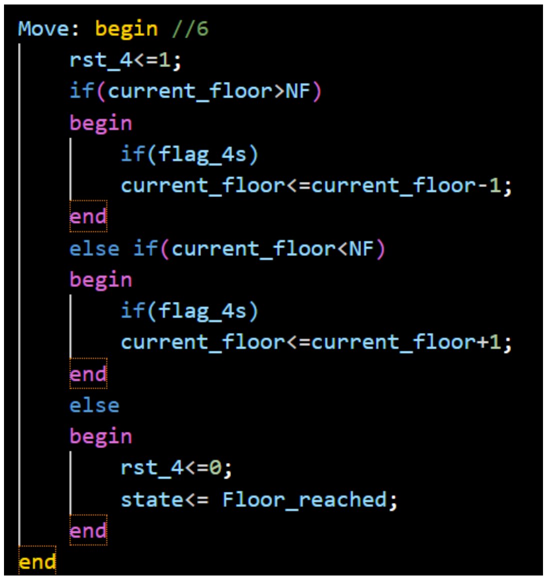


```
Close_door_return: begin //2
    rst_1<=1;
    if(flag_1s)
    begin
        rst_1<=0;
        state<= System_waiting_for_valid;
        Door_status <= 1;
    end
end
```

Figure 6: System in close_door_return

In figure 6, we first start the counter by putting a 1 in rst_1. We then check if 1 second has passed. If it passed, we reset the counter to 0, close the door and go back to the state waiting for the valid button.

Case: Move

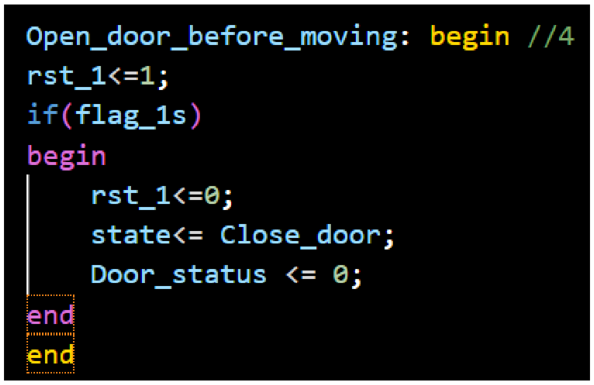


```
Move: begin //6
    rst_4<=1;
    if(current_floor>NF)
    begin
        if(flag_4s)
        current_floor<=current_floor-1;
    end
    else if(current_floor<NF)
    begin
        if(flag_4s)
        current_floor<=current_floor+1;
    end
    else
    begin
        rst_4<=0;
        state<= Floor_reached;
    end
end
```

Figure 7: System in Move state

Figure 7 shows that we are starting the 4 seconds counter and resetting it each time a condition is met. We either increment the current floor till we reach the desired one or decrement the current floor until we reach the desired floor.

Case: Open_door_before_moving

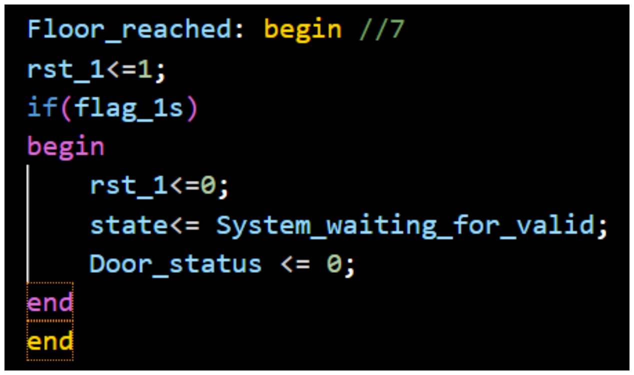


```
Open_door_before_moving: begin //4
rst_1<=1;
if(flag_1s)
begin
    rst_1<=0;
    state<= Close_door;
    Door_status <= 0;
end
end
```

Figure 8: System in Open_door_before_moving

In figure 8, we start the 1 second counter and reset it when 1 second passes. We open the door and then go to the state to close it before moving.

Case: Floor_Reached

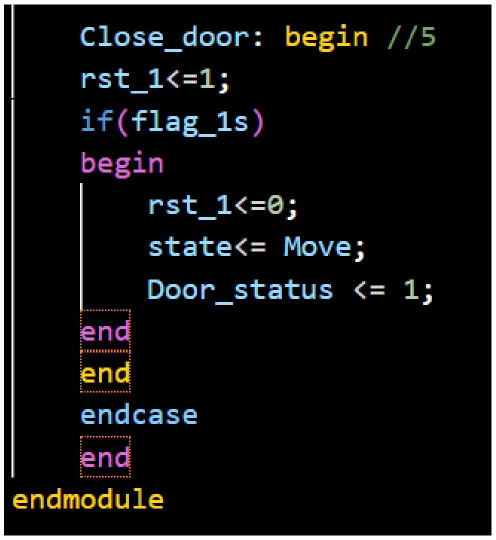


```
Floor_reached: begin //7
rst_1<=1;
if(flag_1s)
begin
    rst_1<=0;
    state<= System_waiting_for_valid;
    Door_status <= 0;
end
end
```

Figure 9: System in floor_reached

Figure 9 shows that we have a 1 second counter that we reset when 1 second passes. We also open the door and send back the system to the waiting for valid state.

Case: Close_Door



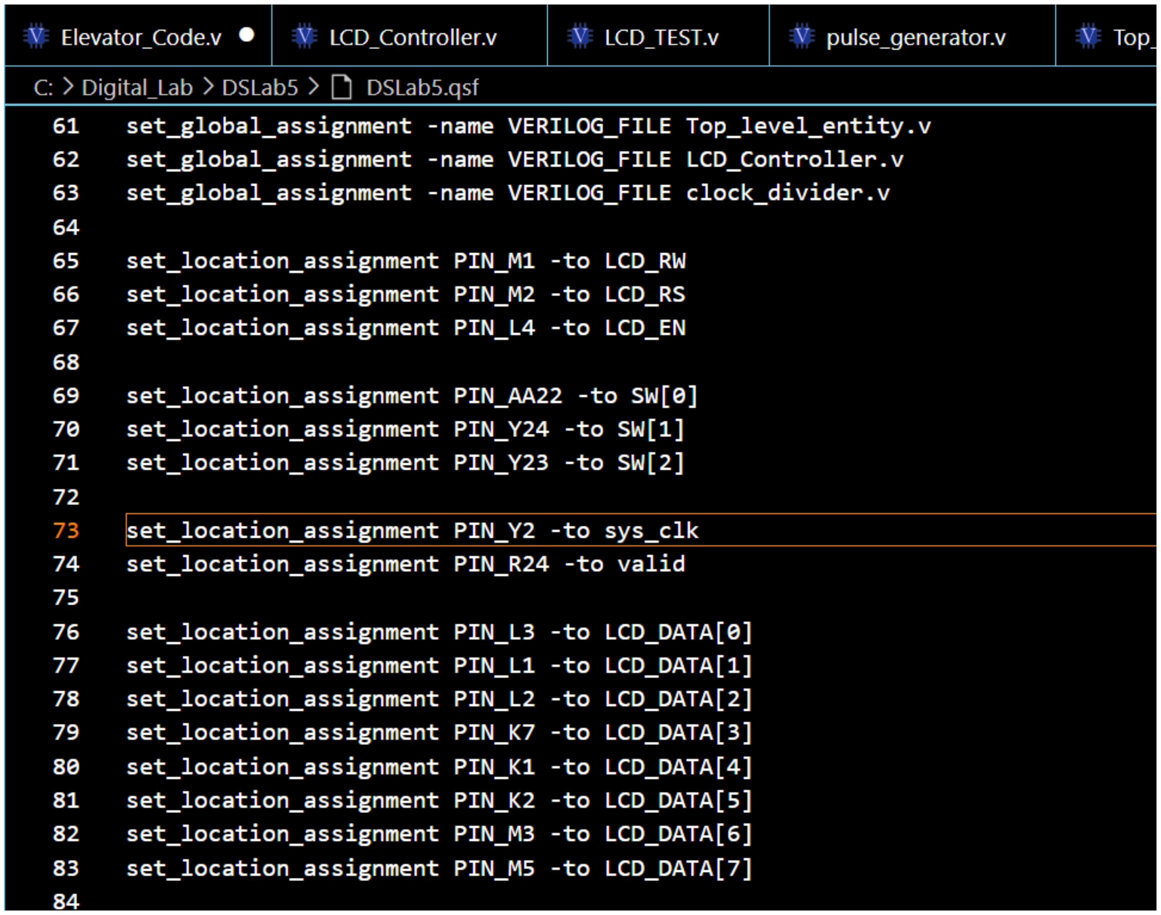
```
    Close_door: begin //5
    rst_1<=1;
    if(flag_1s)
    begin
        rst_1<=0;
        state<= Move;
        Door_status <= 1;
    end
    end
    endcase
    end
endmodule
```

Figure 10: System in Close_door state

In figure 10, we have a 1 second counter that we reset when 1 second passes. We also close the door and send the system to the move state.

We will know include the Pin assignments to show the connections to the LCD and the board.

Pin Assignment



```
set_global_assignment -name VERILOG_FILE Top_level_entity.v
set_global_assignment -name VERILOG_FILE LCD_Controller.v
set_global_assignment -name VERILOG_FILE clock_divider.v

set_location_assignment PIN_M1 -to LCD_RW
set_location_assignment PIN_M2 -to LCD_RS
set_location_assignment PIN_L4 -to LCD_EN

set_location_assignment PIN_AA22 -to SW[0]
set_location_assignment PIN_Y24 -to SW[1]
set_location_assignment PIN_Y23 -to SW[2]

set_location_assignment PIN_Y2 -to sys_clk
set_location_assignment PIN_R24 -to valid

set_location_assignment PIN_L3 -to LCD_DATA[0]
set_location_assignment PIN_L1 -to LCD_DATA[1]
set_location_assignment PIN_L2 -to LCD_DATA[2]
set_location_assignment PIN_K7 -to LCD_DATA[3]
set_location_assignment PIN_K1 -to LCD_DATA[4]
set_location_assignment PIN_K2 -to LCD_DATA[5]
set_location_assignment PIN_M3 -to LCD_DATA[6]
set_location_assignment PIN_M5 -to LCD_DATA[7]
```

Figure 11: Pin Assignment

Other codes used will be included in the submission.

Conclusion

In this lab, we created an elevator by using a state diagram and implementing its code, a pulse generator and displaying the door status as well as the floor on the LCD.