PROCTECH 3DE3:
Digital Electronics
Lab Project Report

Date Submitted: December 1st , 2024
Instructor: Rameez Ashraf
Submitted By:

Luc Suzuki (400332170)
Yara Idris (400393307)

Table of Contents

I. Project Topic Description

The purpose of this lab project is to simulate a real-life elevator system with 3 floors. With the use of mechanical pushbuttons, a floor will be selected by the user and the elevator will respond accordingly to this selection. For example, if the user selects floor 3 and the elevator is currently at floor 1, the elevator will turn on the forward motor and raise to level 3. To sense which floor the elevator is at, there are reed switches at each floor to send a signal when the elevator is detected.

A 7-segment display will be used in combination with the reed switches to display the current floor. As the elevator moves from floor to floor, the display will update to the floor it is at.

II. Description of the resources used (ICs, port/pin assignment, etc.)

Table 1: Inputs and outputs used in the elevator system.

Inputs	Pin Assignment	Outputs	Pin Assignment
Push Button 1	PIN_B3	Motor Up	PIN_A2
Push Button 2	PIN_B4	Motor Down	PIN_A3
Push Button 3	PIN_A4	Encoder A	PIN_D3
Reed Switch 1	PIN_B5	Encoder B	PIN_C3
Reed Switch 2	PIN_A5	Encoder Voltage	PIN_B6
Reed Switch 3	PIN_D5	CLK_50MHz	PIN_R8

Note: **FPGA –** EP4CE22F17C6 was used

III. The Digital Design

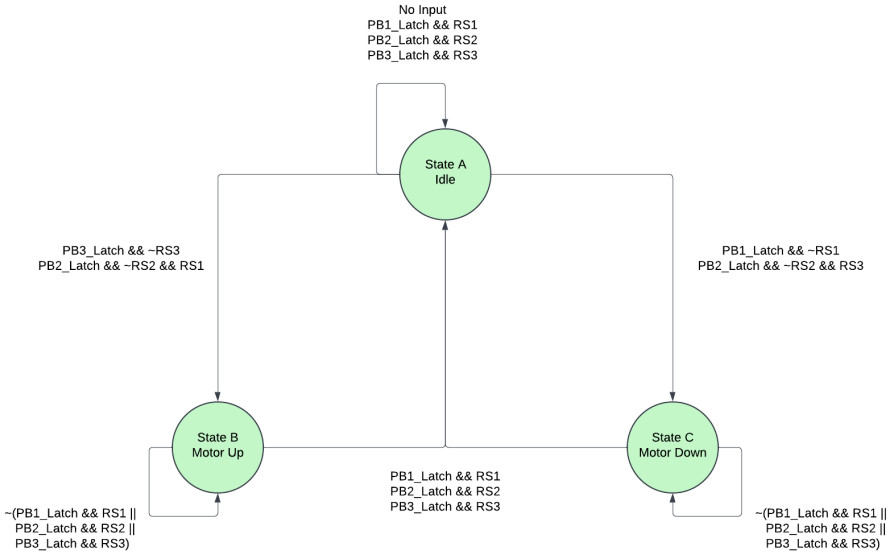

Figure 1: Mealy state diagram of the elevator system.

As seen in figure 1, there are three states that the elevator can be in. The first being an idle state. This state represents when there is no pushbutton active or that the elevator has reached the desired floor. This is detected through the reed switches placed at each floor. The next state will either be motor up or motor down. The criteria for when we want the system to move upwards is:

1. The elevator is currently at floor 1 and the selected push button is either 2 or 3.
2. The elevator is currently at floor 2 and the selected pushbutton is 3.

On the other hand, when we want the motor to move downwards the criteria becomes:

1. The elevator is at floor 3 and pushbuttons 2 or 3.
2. The elevator is at floor 2 and the selected pushbutton is 1.

To implement this logic, we need conditions to describe these scenarios.

Scenario A: Push Button 1 is pressed.

Whenever pushbutton 1 is pressed, it is evident that the current state of the elevator is irrelevant because if the elevator is at floor 2 or 3, it will move into the motor down state – unless the current floor is floor 1. We represent this condition by:

PB1_Latch && ~RS1; **Motor up**

Scenario B: Push Button 3 is pressed.

Whenever pushbutton 3 is pressed, it is evident that the current state of the elevator is irrelevant because if the elevator is at floor 1 or 2, it will move into the motor up state – unless the current floor is floor 3.

We represent this condition by:

PB3_Latch && ~RS3; **Motor down**

Scenario C: Push Button 2 is pressed.

Floor 2 is a unique situation where there are two paths that end up at floor 2 – floor 3 to 2 or floor 1 to 2. To differentiate these two paths, we must add in another condition do determine which of the two we must take. This added condition is the current state of the elevator.

PB2_Latch && ~RS2 && RS3 (Floor 3 to 2); **Motor down**

PB2_Latch && ~RS2 && RS1 (Floor 1 to 3); **Motor up**

Scenario D: Reached desired floor.

When the elevator reached the desired floor, we want to return to the idle state, where the motor is off.

We represent this condition by:

(PB1_Latch && RS1) || (PB2 && RS2) || (PB3_Latch && RS3); **Motor off**

To remain in the **Idle State,** the latched pushbutton and the corresponding reed switch must be both active. Giving us the criteria of:

(PB1_Latch && RS1) || (PB2 && RS2) || (PB3_Latch && RS3); **Motor off**

To remain in a **Motor State** , the none of the latched pushbuttons can be active with its reed switch. Giving us the condition of:

~((PB1_Latch && RS1) || (PB2_Latch && RS2) || (PB3_Latch && RS3)); **Motor No Change**

IV. Flow Chart with Decision Blocks

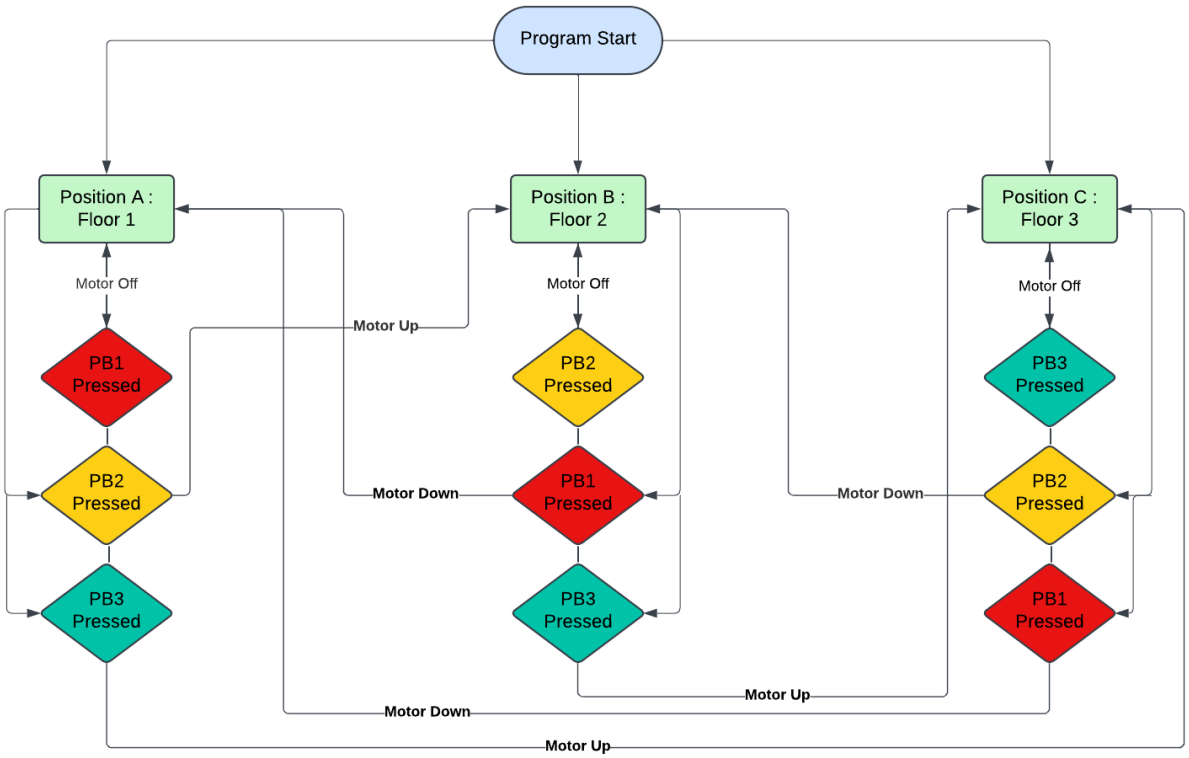

Figure 2: Elevator system flow chart

Action Sequence:

Since this is a continuous program, where we want it to respond to user inputs at different states, there is a cyclic flow to the program. As seen in Figure 2, there are three floors that the elevator can be in idle (1, 2 or 3). At each of these floors the user may press any of the three pushbuttons to start an action.

V. Pictures with Final Setup

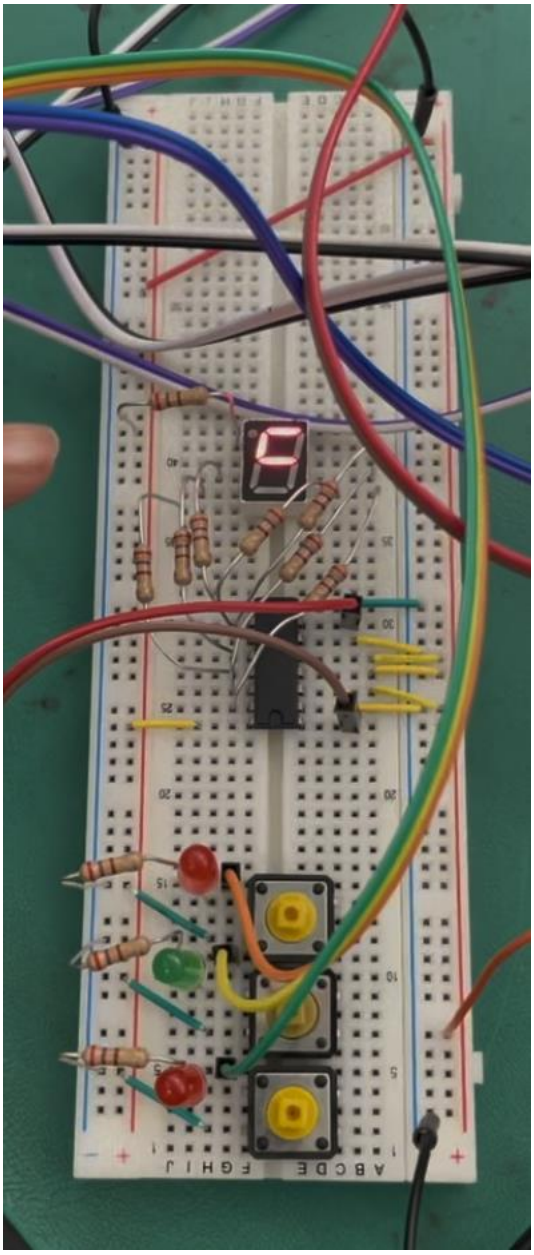

Figure 3.1: Breadboard A - 7-Segment Display + BCD Encoder + PB1-3.

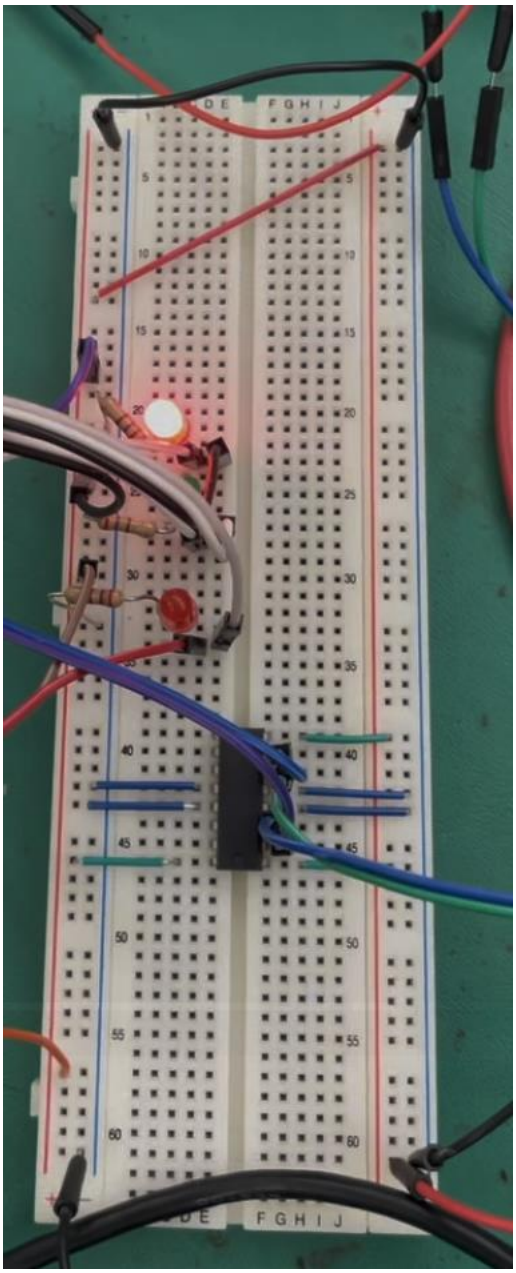

Figure 3.2: Breadboard B - Moto Driver + Reed Switch Display.

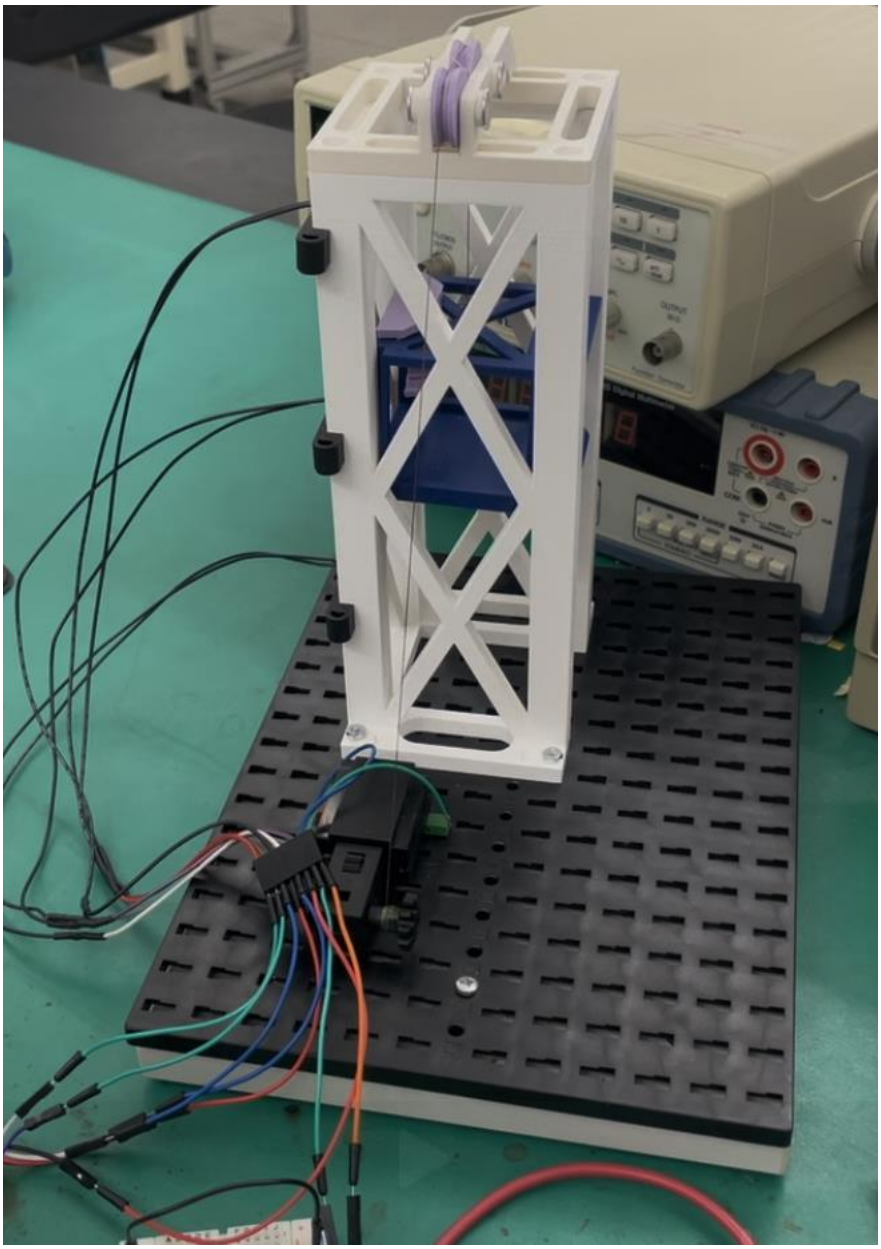

Figure 3.3: Elevator system

VI. Wiring Diagram

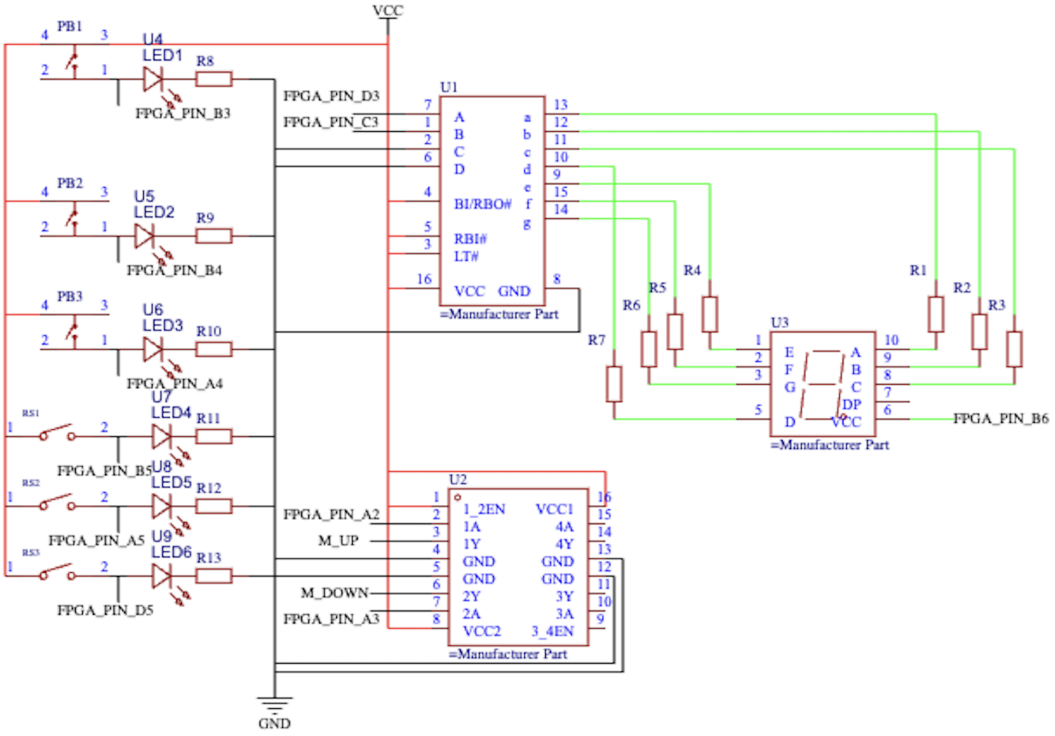

Figure 4: Wiring Diagram

VII. Summary Table with All Operational Features Developed

Table 2: Summary table of the hardware.

Hardware	Trigger	Feature Description
Push Buttons (3)	User presses floor button	Elevator moves to the selected floor
PB LED's (3)	Enables once PB is pressed	Shows the selected floor
Reed Switch's (3)	Elevator reaches a floor	Signal is passed when the elevator is in proximity to the switch
Reed switch LED's (3)	Elevator reaches a floor	Signals that the elevator is at a designated floor
7-Segment Display (1)	Latched reed switch	Displays the most recent floor
SN74LS47 BCD-to-7-segment Display Decoder (1)	BCD code is sent to the device	Converts the BCD code to activate specific segments
SN754410 Motor Driver (1)	Voltage is applied to the device	Moves the motor forwards or backwards

VIII. Discussions of Project

Limitations:

Our elevator is unable to make multiple floor selections, it only takes in a new floor input after we have reached our previous desired floor. We are also unable to remember what floor we were at after a power outage, the elevator must be reset at any floor before continuing with the code.

Design Selection:

In the design stage of the project, we had two thoughts about how to implement the system. The first being floor based. Creating our state diagram around the floors of the elevator. In essence there will be 3 floors / states and in combination with some conditions, the system will travel from floor to floor at the user's demand. This is a valid solution to the problem from what we observed. However, we believed that it runs into problems with scalability. With more and more floors being added it gets increasingly harder to implement. Our second option was a motor-based state diagram, where each of the states dictate what the motor is doing specifically whether the motor is moving up, down or not moving. We believed this would have been the best solution to the project.

Project Achievements:

1. System reaches desired floor from user selection.

Based on the user's floor selection, the elevator system responds correctly to the input. For example, if the elevator is at floor 2 and the user inputs floor 3 the motor will drive upwards until the third reed switch – RS3 – is enabled, which halts any more motion. This operation works the same for any combination of floor and selection pairing.

2. System ignores multiple floor selections.

If the user happens to select more than one floor at one time, the system remains in the idle state.

3. Pushbutton and reed switch detection with LED

In figure 3.1 it is shown that there are 3 LED beside the floor pushbuttons. When the user selects their desired floor, the LED is enabled, visually displaying and confirming their selection. Similarly, the reed switches have a paired LED to identify the current floor the elevator is at. Once the elevator has reached or passed a certain floor, the reed switch will enable and activate the LED.

4. 7-segment display shows current floor; **partial**

This system's 7-segment display uses a BCD encoder – specifically for 7-segment display usage. Using the FPGA, we are able to assign 3 pins to control the display, Encoder A, Encoder B and Encoder Voltage. These three bits are passed to the encoder and are sent directly to the 7-segment display and displaying the desired output. Encoder A controls the LSB and Encoder B controls the adjacent bit to the LSB. Encoder voltage controls the power output, to turn on and off the display. Since there are only 3 floors, two bits can encompass all options. For example, when we want a "3" displayed both encoder A and B will be active.

$$0001 \rightarrow a = 0,\ b = 1,\ c = 1,\ d = 0,\ e = 0,\ f = 0,\ g = 0$$

$$0010 \rightarrow a = 1,\ b = 1,\ c = 0,\ d = 1,\ e = 1,\ f = 0,\ g = 1$$

$$0011 \rightarrow a = 1,\ b = 1,\ c = 1,\ d = 1,\ e = 0,\ f = 0,\ g = 1$$

At each of the floors, the display will show the current floor.

Code Review:

```
module JK_FF(
  input J, K, C       // J, K and clock inputs
  output reg Q, Q_   // Q and Q_ outputs
);
 always @(posedge C) begin
    if (K) begin
      Q <= 0;    // Reset Q to 0
      Q_ <= 1;   // Reset Q_ to 1
     end
   else if (J) begin
         Q <= 1;       // On clock edge, latch J to Q
         Q_ <= 0;     // Q_ is the complement of J
     end
    end
endmodule

module Lab_Project_Final(
    input CLK_50MHz, PB1, PB2, PB3,    //  Pushbuttons (PB1, PB2, PB3)
    input RS1, RS2, RS3,                // Reed switches (RS1, RS2, RS3)
    output reg M_UP = 1'b0, M_DOWN = 1'b0,  // Motor directions
output reg Encoder_A, Encoder_B, Encoder_Voltage
);
    localparam IDLE = 2'b00,           // Elevator is idle (waiting for button press)
               MOVING_UP = 2'b01,     // Moving up to a floor
               MOVING_DOWN = 2'b10,   // Moving down to a floor
               SETUP = 2'b11;       // Elevator has reached the floor

    reg [1:0] state, check; // Registers for current and next state
    reg ARRIVED, Many_In;  // Register to check if elevator has arrived at the floor
    reg [5:0] up_counter; // Counter to track clock cycles for moving up

    // Flip-flop outputs (latch the button presses)
    wire QPB1_latch, QPB2_latch, QPB3_latch, QE1_latch, QE2_latch;
    wire Q_PB1_latch, Q_PB2_latch, Q_PB3_latch, Q_E1_latch, Q_E2_latch;
    // setting the JK flipflops for each PB and Decoder inputs
```

Figure 5.1: Section 1 of FPGA code

The first implemented module is the J-K Flip Flop. This is used to latch the user's selection and keep it in memory. This is necessary because we want the user to be able to press the button and for the system to remember the selection. This is very useful when implementing the state logic later in the code.

The second module that is added in is our main logic module, here we've added in our physical devices as well as our logic components seen in Table 1. The use of a clock allows us to check for changes in conditions of the system.

```
    // Flip-flop outputs (latch the button presses)
    wire QPB1_latch, QPB2_latch, QPB3_latch, QE1_latch, QE2_latch;
    wire Q_PB1_latch, Q_PB2_latch, Q_PB3_latch, Q_E1_latch, Q_E2_latch;
    // setting the JK flipflops for each PB and Decoder inputs
    JK_FF JKFF1 (
        .J(PB1),
        .C(CLK_50MHz),
        .K(ARRIVED),
        .Q(QPB1_latch),
        .Q_(Q_PB1_latch)
    );

    JK_FF JKFF2 (
        .J(PB2),
        .C(CLK_50MHz),
        .K(ARRIVED),
        .Q(QPB2_latch),
        .Q_(Q_PB2_latch)
    );

    JK_FF JKFF3 (
        .J(PB3),
        .C(CLK_50MHz),
        .K(ARRIVED),
        .Q(QPB3_latch),
        .Q_(Q_PB3_latch)
    );
// If on 1st or 3rd floor LSB is set to 1 and unlached when on 2nd floor
   JK_FF JKFF4 (
        .J(RS1 || RS3),
        .C(CLK_50MHz),
        .K(RS2),
        .Q(QE1_latch),
        .Q_(Q_E1_latch)
    );
```

Figure 5.2: Section 2 of FPGA code

Here is our J-K Flip Flop implementation. Where we are latching each of the pushbuttons as well as the encoder values used for the 7-segment display.

```
// If on 2nd or 3rd floor second bit is set to 1 and unlached when on 1st floor
   JK_FF JKFF5 (
        .J(RS2 || RS3),
        .C(CLK_50MHz),
        .K(RS1),
        .Q(QE2_latch),
        .Q_(Q_E2_latch)
    );

// State transition logic (based on button presses and reed switches)
always @(posedge CLK_50MHz) begin
   Encoder_A <= QE1_latch;
   Encoder_B <= QE2_latch;
   check <= ((QPB1_latch && RS1) || (QPB2_latch && RS2) || (QPB3_latch && RS3));

if (moving_down && ~RS1 && ~RS2 && ~RS3) begin
   if (up_counter <= 6'd50) begin
      up_counter <= up_counter + 1;
   end
   else begin
      Encoder_Voltage <= ~Encoder_Voltage;
      up_counter <= 6'd0;
   end
end
else Encoder_Voltage <= 1;

if (ARRIVED) ARRIVED <= 0;
  case(state)
      IDLE: begin
         if ((QPB1_latch && QPB2_latch) || (QPB1_latch && QPB3_latch) || (QPB2_latch && QPB3_latch)) begin
         ARRIVED <= 1;
         state <= IDLE;
         M_UP <= 1'b0;
```

Figure 5.3: Section 3 of FPGA code

This next section of code begins our state logic. The check variable is crucial to operation because it determines if the elevator is at a desired floor / current floor.

This case statement in figure 4.3 and 4.4, involves the three states seen in Figure 1. Using those conditions, we are able to move between states and update the motor function accordingly.

```
        state <= IDLE;
        M_UP <= 1'b0;
        M_DOWN <= 1'b0;
        end
        else if (QPB1_latch && ~RS1) begin
        state <= MOVING_DOWN;
        M_UP <= 1'b1;
        M_DOWN <= 1'b0;
        end
        else if (QPB2_latch && ~RS2 && RS1) begin

              state <= MOVING_UP;
              M_UP <= 1'b0;
              M_DOWN <= 1'b1;

        end
        else if (QPB2_latch && ~RS2 && RS3) begin

              state <= MOVING_DOWN;
              M_UP <= 1'b1;
              M_DOWN <= 1'b0;

        end
        else if (QPB3_latch && ~RS3) begin
        state <= MOVING_UP;
        M_UP <= 1'b0;
        M_DOWN <= 1'b1;
        end
        else if (~RS1 && ~RS2 && ~RS3 && ~moving_down) begin
        state <= SETUP;
        end
        else begin
        state <= IDLE;
        M_UP <= 1'b0;
        M_DOWN <= 1'b0;
        end
```

Figure 5.4: Section 4 of FPGA code

```
            else begin
            state <= IDLE;
            M_UP <= 1'b0;
            M_DOWN <= 1'b0;
            end
        end
        MOVING_UP: begin
            if (check) begin
            ARRIVED <= 1;
            state <= IDLE;
            M_UP <= 1'b0;
            M_DOWN <= 1'b0;
            end
            end
        MOVING_DOWN: begin
            if (check) begin
            ARRIVED <= 1;
            state <= IDLE;
            M_UP <= 1'b0;
            M_DOWN <= 1'b0;
            end
            end

default: begin
state <= IDLE;
ARRIVED <= 0;
moving_up <= 0;
moving_down <= 0;
end
  endcase
      end
endmodule
```

Figure 5.5: Section 5 of FPGA code

As a default, the system will be put into the idle state to ensure that no movement is enabled at activation.

IX. Peer Evaluation

Luc Suzuki - 50%

Throughout this project, both my partner and I worked very closely from design all the way to our final product. In our design stage, we both engaged heavily by contributing two different solutions to the problem, the motor state option and the floor-based option. This brainstorming period really helped us develop both the hardware and software behind its implementation.

On the hardware side, my partner took charge of controlling the wiring of the breadboard and pin assignments for the FGPA. I was able to contribute to this section by looking over the documentation for the motor driver and the BCD-to-7-segment decoder. We both contributed to the troubleshooting of the wiring when results were not what we were expecting and found some errors to help move forward.

Lastly, with the coding of the FPGA I was able to implement the main logic for the FPGA using the motor state option which I was familiar with. My partner helped with the decoding logic for the 7-segment display.

Yara Idris – 50%

We had many ups and downs throughout each step of the process; however, we were able to overcome each obstacle. My lab partner and I had many different ideas on how to work out the logic for the elevator originally.

When it came to actually starting on the project, I took charge of the physical wiring of the components while my partner worked on the logic on how to implement the elevator. We both worked on sections of the code. Such as how we will code each state, or how to make a function for JK flip-flops. Then later we both worked on troubleshooting any issues we faced when putting the hardware and software together.