Project Members: Can Baltalı, Uğurhan Altınbaş

Date of Submission: 22/01/2023 – Fall 2022-23

CS303 – Term Project Report

Elevator Project

Introduction

For our CS303 term project, it was asked of us to implement a 5-floor elevator design on an FPGA board. The “Seven-Segment Display”, “Debouncer”, “Top”, “CLK_Divider”, and “Elevator” modules were given to us with a constraint file, and it was asked of us to modify the elevator module with our code. We have decided on using 4 states in total: “IDLE”, “WAIT”, “BUSY – UP”, “BUSY – DOWN”. The elevator is in its “IDLE” state when there are no inputs given to it (when no buttons are pressed neither from the outside in any of the floors or from inside the elevator) and the elevator rests. The elevator is in its “BUSY” state when there is an input (request) from a floor and the elevator is moving, the “UP” and “DOWN” states indicate which direction the elevator is moving. The elevator is in its “WAIT” state when the elevator stops by a floor in which a request is given and the request’s direction is the same as the elevator’s direction. After the “WAIT” state, the elevator either changes to “BUSY” state (if the floor the elevator is on is not the floor the original request was given from a.k.a. the final floor) or the “IDLE” state (if the floor is the final floor).

The Input and “Reset” Check

```
//****Your code goes here*****//
always@(posedge floor_0_p or posedge floor_1_p or posedge floor_2_p or posedge floor_3_p or posedge floor_4_p or posedge floor_0_d or posedge floor_1_d or posedge floor_2_d or posedge floor_3_d or posedge f
    command <= 1;
end

always@(posedge clk_50hz or negedge rst) begin
    if(~rst) begin
        current_floor <= 0;
        state <= "IDLE";
    end
```

Here, the code checks whether a request has been given to the elevator from any of the floors or from inside the elevator. If a request has been given, the “command” variable becomes 1 which is used to switch the elevator’s “IDLE” state to “BUSY” (see below for implementation).

The second “always” block is the bulk of our code. Whenever the “reset” button is pressed, the elevator is returned to its default condition (the elevator is at the first floor and in the “IDLE” state) instantly; but if the reset button is not pressed, at every clock signal the module runs. The 5 second waiting time is implemented with a “count” variable (see below).

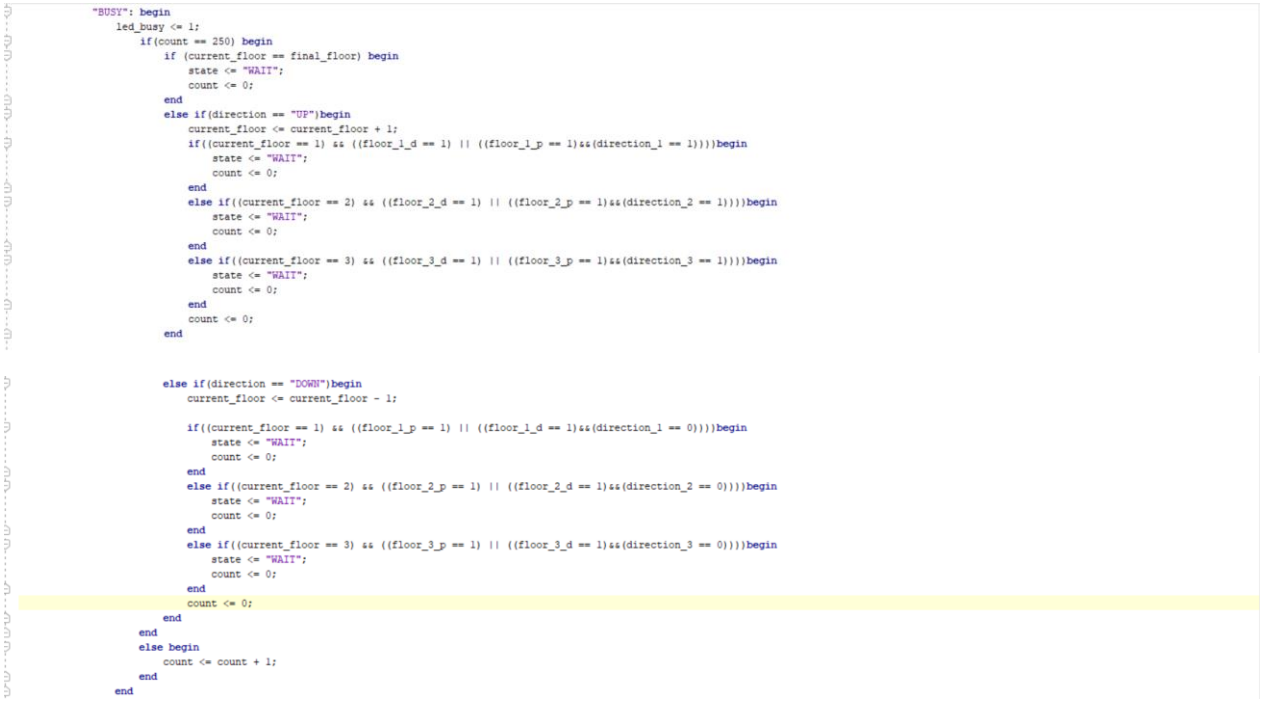
The “IDLE” State

```
    else begin
        case(state)
            "IDLE": begin
                    led_busy <= 0;
                    if(command == 0) begin
                        state <= "IDLE";
                    end
                    else begin
                        if((floor_0_p == 1) || (floor_0_d == 1)) begin
                            state <= "BUSY";
                            final_floor <= 0;
                            count <= 0;
                            direction <= "DOWN";
                        end

                        else if((floor_1_p == 1) || (floor_1_d == 1)) begin
                            state <= "BUSY";
                            final_floor <= 1;
                            count <= 0;
                            if(current_floor < 1)begin
                                direction <= "UP";
                            end
                            else begin
                                direction <= "DOWN";
                            end
                        end

                        else if((floor_2_p == 1) || (floor_2_d == 1)) begin
                            state <= "BUSY";
                            final_floor <= 2;
                            if(current_floor < 2)begin
                                direction <= "UP";
                            end
                            else begin
                                direction <= "DOWN";
                            end
                        end


                        else if((floor_3_p == 1) || (floor_3_d == 1)) begin
                            state <= "BUSY";
                            final_floor <= 3;
                            if(current_floor < 3)begin
                                direction <= "UP";
                            end
                            else begin
                                direction <= "DOWN";
                            end
                        end

                        else if((floor_4_p == 1) || (floor_4_d == 1)) begin
                            state <= "BUSY";
                            final_floor <= 4;
                            if(current_floor < 1)begin
                                direction <= "UP";
                            end
                            else begin
                                direction <= "DOWN";
                            end
                        end
                    end
                end
```

In this state, the code checks whether there are any inputs given to the elevator by checking the “command” variable. As long as there are no requests, the elevator stays in its “IDLE” state and “led_busy” (the LED that lights up when the elevator is moving) is 0.

If a request is given, depending on the context of the request, the final floor information (the first request of the elevator) and the direction the request giver wants the elevator to go is stored. If the final floor is located at a lower position than the current floor, then the elevator’s direction is changed to “DOWN”, and if the final floor is located at a higher position than the current floor, then the elevator’s direction is changed to “UP”, indicating which way the elevator moves.

The ”BUSY” State



```
"BUSY": begin
    led_busy <= 1;
        if(count == 250) begin
            if (current_floor == final_floor) begin
                state <= "WAIT";
                count <= 0;
            end
            else if(direction == "UP")begin
                current_floor <= current_floor + 1;
                if((current_floor == 1) && ((floor_1_d == 1) || ((floor_1_p == 1)&&(direction_1 == 1))))begin
                    state <= "WAIT";
                    count <= 0;
                end
                else if((current_floor == 2) && ((floor_2_d == 1) || ((floor_2_p == 1)&&(direction_2 == 1))))begin
                    state <= "WAIT";
                    count <= 0;
                end
                else if((current_floor == 3) && ((floor_3_d == 1) || ((floor_3_p == 1)&&(direction_3 == 1))))begin
                    state <= "WAIT";
                    count <= 0;
                end
                count <= 0;
            end

            else if(direction == "DOWN")begin
                current_floor <= current_floor - 1;

                if((current_floor == 1) && ((floor_1_p == 1) || ((floor_1_d == 1)&&(direction_1 == 0))))begin
                    state <= "WAIT";
                    count <= 0;
                end
                else if((current_floor == 2) && ((floor_2_p == 1) || ((floor_2_d == 1)&&(direction_2 == 0))))begin
                    state <= "WAIT";
                    count <= 0;
                end
                else if((current_floor == 3) && ((floor_3_p == 1) || ((floor_3_d == 1)&&(direction_3 == 0))))begin
                    state <= "WAIT";
                    count <= 0;
                end
                count <= 0;
            end
        end
        else begin
            count <= count + 1;
        end
    end
```

In this state, since there is a new clock signal at every 20 milliseconds, we have created a temporary variable called “count” to increment by 1 at every 20 milliseconds, until it reaches 250, so that at every 5 seconds the elevator moves to a different floor. If the elevator is at the floor the original request was given from, the elevator changes its state to “WAIT”.

If the elevator is not at the floor in which the original request was given from, then at every floor it checks whether another input was given from the floor the elevator is currently located at. If a request was given from a floor from the outside, if the request giver’s desired direction is the same as the elevator’s direction, then the elevator stops by and changes to “WAIT” state. If a request was given from inside the elevator to stop at that floor, then the elevator stops without regarding the direction. If no inputs were given or the direction given from the outside

did not match the elevator's direction, then the elevator disregards the request and continues to the next floor. During this state, the LED that shows whether the elevator is moving or not lights up (a.k.a. "led_busy").

The "WAIT" State

```
            "WAIT": begin
                led_busy <= 1;
                    if(count == 250)begin
                        if(current_floor == final_floor)begin
                            state <= "IDLE";
                        end
                        else begin
                            state <= "BUSY";
                            count <= 0;
                        end
                    end
                    else begin
                        count <= count + 1;
                    end
                end
        endcase
    end
end
```

In this state, the elevator checks whether it has stopped on the floor in which the original request was given from. If the elevator is on the final floor, then it changes its state to "IDLE" after 5 seconds. If the elevator stopped on a floor in which an additional request was given from, it starts moving again after 5 seconds, changing its state to "BUSY".

The 5 second waiting time is implemented again using the "count" variable.

The Seven-Segment Display Outputs

```
always @(posedge clk_50hz or negedge rst) begin
    if(~rst) begin
        a <= 8'b11110110;
        b <= 8'b11001110;
        c <= 8'b11001110;
        d <= 8'b11101010;
        e <= 8'b11100010;
        f <= 8'b11110010;
        g <= 8'b00100101;
        p <= 8'b11111111;
    end
    else begin
        if(current_floor == 0) begin
            a[0] <= 0;
            b[0] <= 0;
            c[0] <= 0;
            d[0] <= 0;
            e[0] <= 0;
            f[0] <= 0;

            g[0] <= 1;
            p[0] <= 1;
        end
        else if(current_floor == 1) begin
            b[0] <= 0;
            c[0] <= 0;

            a[0] <= 1;
            d[0] <= 1;
            e[0] <= 1;
            f[0] <= 1;
            g[0] <= 1;
            p[0] <= 1;
        end
        else if(current_floor == 2) begin
            a[0] <= 0;
            b[0] <= 0;
            g[0] <= 0;
            d[0] <= 0;
            e[0] <= 0;
```

```
            c[0] <= 1;
            p[0] <= 1;
            f[0] <= 1;
        end
        else if(current_floor == 3) begin
            a[0] <= 0;
            b[0] <= 0;
            g[0] <= 0;
            d[0] <= 0;
            c[0] <= 0;

            e[0] <= 1;
            f[0] <= 1;
            p[0] <= 1;
        end
        else begin
          f[0] <= 0;
          g[0] <= 0;
          b[0] <= 0;
          c[0] <= 0;

          a[0] <= 1;
          d[0] <= 1;
          e[0] <= 1;
          p[0] <= 1;
        end

        if(state == "IDLE")  begin
            a[7:4] <= 4'b1111;
            b[7:4] <= 4'b1100;
            c[7:4] <= 4'b1100;
            d[7:4] <= 4'b1110;
            e[7:4] <= 4'b1110;
            f[7:4] <= 4'b1111;
            g[7:4] <= 4'b0010;
            p[7:4] <= 4'b1111;
         end
         else if((state == "BUSY")&&(direction == "UP")) begin
             a[5] <= 1;
             f[5] <= 0;
             e[5] <= 0;

             d[5] <= 0;
             b[5] <= 0;
             c[5] <= 0;
             g[5] <= 1;
             p[5] <= 1;

             a[4] <= 0;
             b[4] <= 0;
             c[4] <= 1;
             d[4] <= 1;
             p[4] <= 1;
             g[4] <= 0;
             f[4] <= 0;
             e[4] <= 0;
         end
         else if((state == "BUSY")&&(direction == "DOWN")) begin
             b[5] <= 0;
             g[5] <= 0;
             e[5] <= 0;
             d[5] <= 0;
             c[5] <= 0;
             a[5] <= 1;
             f[5] <= 1;
             p[5] <= 1;


             a[4] <= 0;
             b[4] <= 0;
             c[4] <= 0;
             d[4] <= 0;
             e[4] <= 0;
             f[4] <= 0;
             g[4] <= 1;
             p[4] <= 1;

         end
     end
  end
endmodule
//****Your code goes here*****//
```

Depending on the elevator's floor and the direction it is moving, the two seven-segment display's outputs are implemented here.