EMBEDDED HARDWARE DESIGN

LIFT CONTROLLER

GHORI ZEEL JIVRAJBHAI

7984313073

ghorizeeljkvrz@gmail.com

EL-203

LIFT CONTROLLER

PROF. BISWAJIT MISHRA
PROF. YASH AGARWAL

BATCH 4
GROUP 107 -15

INTRODUCTION

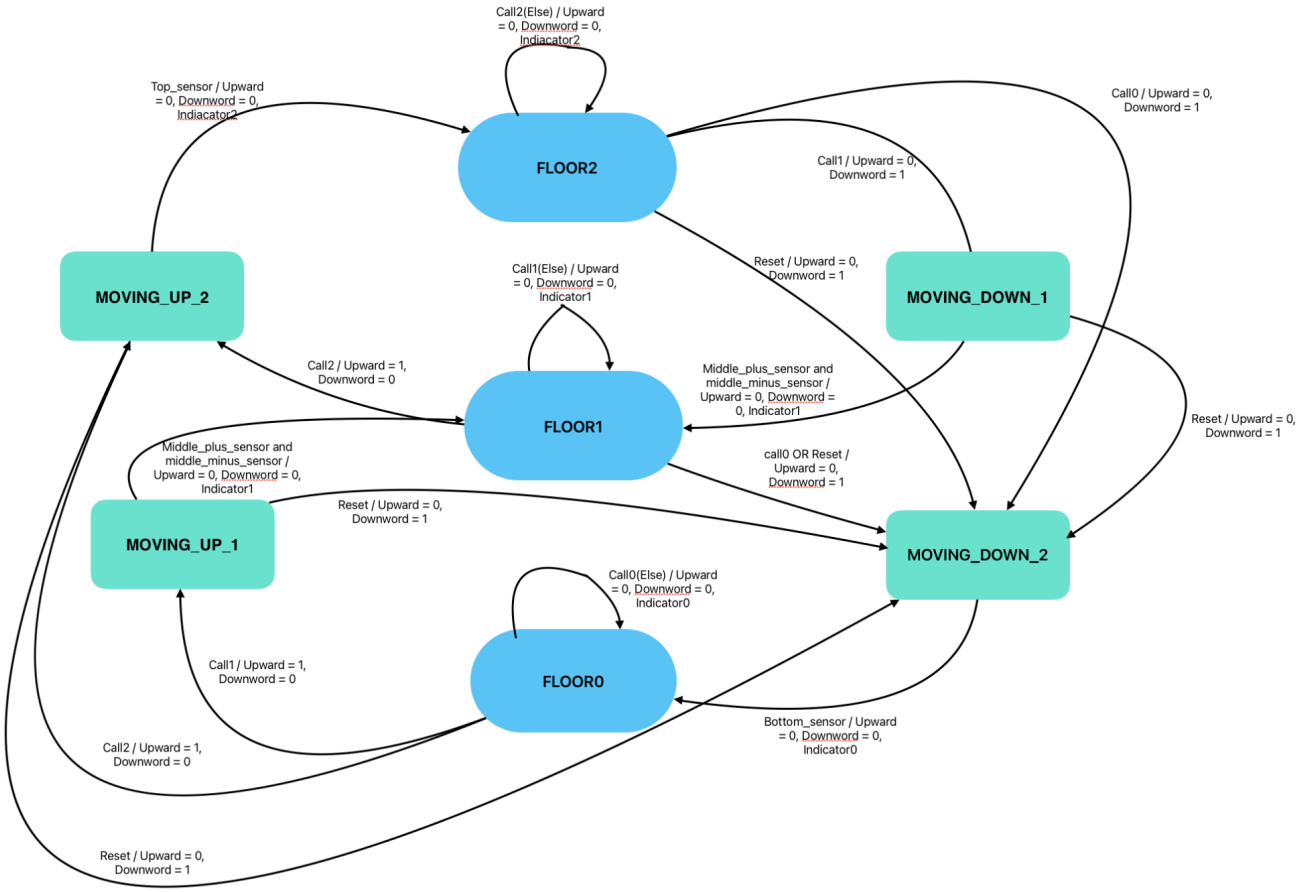
At its core, the elevator control system functions as a finite state machine (FSM). An FSM is characterized by a collection of states, a starting state, and specific inputs that trigger transitions between states. Based on particular inputs, the FSM may shift from one state to another, a process known as a transition.

GUIDELINES FOR ELEVATOR OPERATION SYSTEM

1. Initially, the elevator is at floor 0, referred to as State (S0), and the motor remains inactive.
2. Activation occurs when a floor selection button is pressed, indicating a floor different from the current one. This triggers a change to one of the four predefined motion states in the state diagram, each designed to control motor activity.
3. Once a selection is made, subsequent inputs are disregarded until the elevator arrives at the selected floor. The motor then receives a command to either ascend or descend, indicated by inputs '10' or '01' respectively to the motor controller.
4. A LED indicator lights up to show the door is open at a particular floor. Arrival at the floor is confirmed through the use of four infrared sensors, prompting a transition to the appropriate state.

STATE DIAGRAM

The control system of a three-floor elevator includes seven distinct states. Requests to move the elevator are made using push buttons located on each floor, which determine the response of the system.



DETAILED DESCRIPTION OF EACH STATE

FLOOR STATES :

FLOOR0 (S0):

This state corresponds to the zero floor within the elevator system. Activating call0 will maintain the elevator in its current state. When call1 is pressed to request the first floor, the state transitions to MOVING_UP_1 (S3), which will move the elevator upwards to reach the first floor (S1). Similarly, pressing call2 changes the state to MOVING_UP_2, setting the course for the elevator to ascend to the second floor (S2) in the subsequent state.

FLOOR1 (S1):

This state is associated with the first floor of the elevator system. Pressing call1 will keep the elevator on this floor without any state change. When call0 is pressed for the zero floor, the state moves to MOVING_DOWN_2 (S6), which directs the elevator downwards to reach the zero floor (S0). Conversely, if call2 is pressed to request the second floor, the state changes to MOVING_UP_2, guiding the elevator upwards to the second floor (S2) in the following state.

FLOOR2 (S2):

This state pertains to the second floor in the elevator system. Pressing call2 will maintain the current state. If call0 is pressed, the state shifts to MOVING_DOWN_2(S6), which will move the elevator downwards to eventually reach the zero floor (S0). Similarly, pressing call1 to go to the first floor will change the state to MOVING_DOWN_1(S5), which will also direct the elevator downwards, ultimately arriving at the first floor (S1) in its next state.

MOVING STATES :

MOVING_UP_1 (S3):

This state facilitates the elevator's ascent from FLOOR0 (S0) to FLOOR1(S1). When activated, the motor receives a ‘01’ signal, causing it to rotate counterclockwise and thus propelling the elevator upward. Once infrared sensors, namely middle_plus_sensor and middle_minus_sensor, are both triggered, the system transitions to FLOOR1(S1). At this point, the motor is deactivated by sending a ‘00’ input to the motor controller, effectively turning it off.

MOVING_UP_2 (S4):

This state facilitates the elevator's ascent from FLOOR1 (S1) to FLOOR2 (S2) or FLOOR0 (S0) to FLOOR2 (S2) . When activated, the motor receives a ‘01’ signal, causing it to rotate counterclockwise and thus propelling the elevator upward. Once the infrared sensor, namely top_sensor is triggered, the system transitions to FLOOR2 (S2). At this point, the motor is deactivated by sending a ‘00’ input to the motor controller, effectively turning it off.

MOVING_DOWN_1 (S5):

This state manages the descent of the elevator from FLOOR2(S2) to FLOOR1(S1). In this mode, the motor is signaled with ‘01’, causing it to rotate clockwise and thus lowering the elevator. Once both infrared sensors, designated as middle_plus_sensor and middle_minus_sensor, are activated, the system transitions to FLOOR1(S1). Here, the motor is shut down by delivering a ‘00’ input to the motor controller, turning it off.

MOVING_DOWN_2 (S6):

This state manages the descent of the elevator from FLOOR2(S2) to FLOOR0(S0) or FLOOR1(S1) to FLOOR0(S0). In this mode, the motor is signaled with ‘10’, causing it to rotate clockwise and thus lowering the elevator. Once the infrared sensor, designated as bottom_sensor, is activated, the system transitions to FLOOR0(S0). Here, the motor is shut down by delivering a ‘00’ input to the motor controller, turning it off.

INPUT CALL BUTTONS (call0, call1, call2)

These are the inputs which are obtained from the user. It will drive the elevator from floor states(S0/S1/S2) to moving states (MOVING_UP_1(S3), MOVING_UP_2(S4), MOVING_DOWN_1(S5) OR MOVING_DOWN_2(S6)).

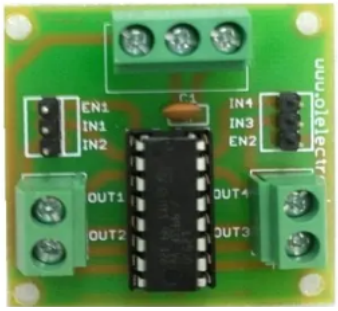
SENSORS

TOP_SENSOR: This IR sensor will provide input for making sure that floor 2 is reached. The input from this sensor is useful when the elevator is in state MOVING_UP_2 and is about to reach Floor 2 (S2). The elevator will transition to S2 only when this sensor will become active, which will happen when the elevator blocks the path of this sensor.

MIDDLE_MINUS_SENSOR AND MIDDLE_PLUS_SENSOR: These IR sensors will provide input for making sure that floor 1 is reached. The inputs from these sensors are useful when the elevator is in state MOVING_UP_1/MOVING_DOWN_1 and is about to reach Floor 1(S1). The elevator will transition to S1 only when these sensors will become active, which will happen when the elevator blocks the path of both of these sensors.

BOTTOM_SENSOR: This IR sensor will provide input for making sure that floor 0 is reached. The input from this sensor is useful when the elevator is in state MOVING_DOWN_2 and is about to reach Floor 0(S0). The elevator will transition to S0 only when this sensor will become active, which will happen when the elevator blocks the path of this sensor.

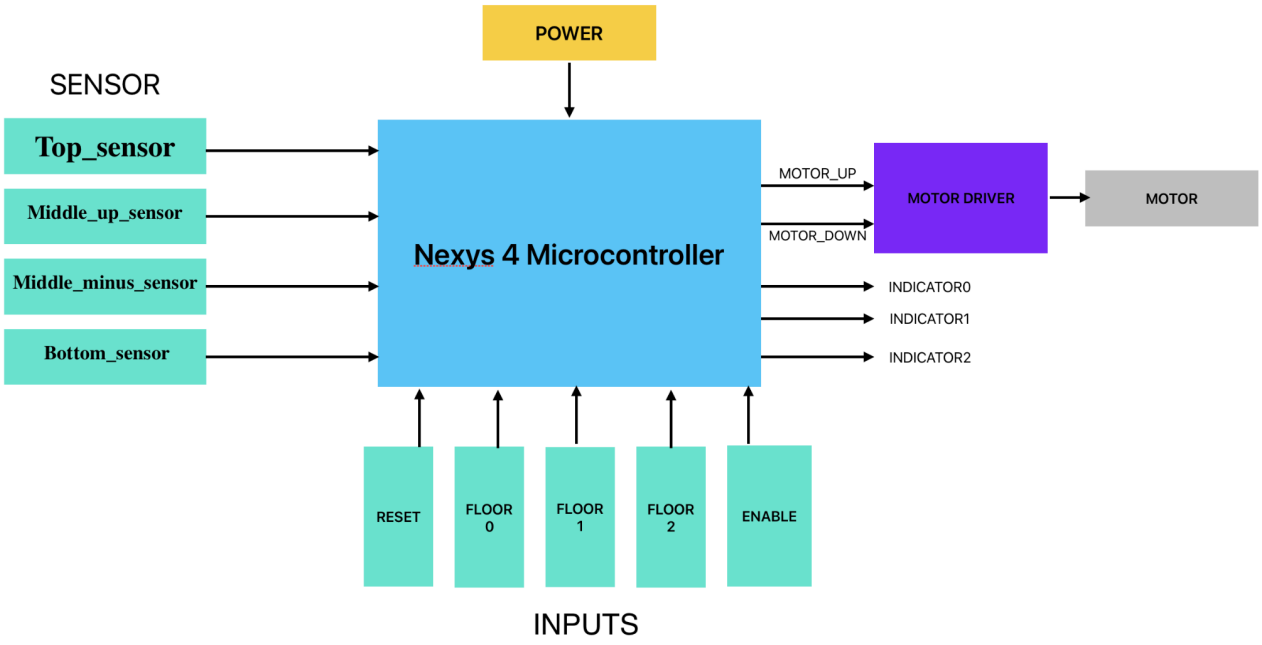
DC MOTOR DRIVER



A DC motor driver is an electronic circuit that controls the direction, speed, and operation of a DC motor, enabling it to interface safely and effectively with low-power control systems like microcontrollers. It modulates input signals (e.g., from a microcontroller) into the necessary output signals to drive the motor, often incorporating features for protection and efficiency. It takes MOTOR_UP AND MOTOR_DOWN as input from the FPGA and when the value of both the pins are 0 then the motor will be in halt. If (MU, MD) = (1,0) then lift will move upward and id (MU,MD) = (0,1) then lift will move downward.

BLOCK DIAGRAM

The following block diagram helps us to understand the flow of inputs and outputs in our elevator system much more clearly than the state diagram. The cyan blocks show us the inputs to the microcontroller, which mainly consists of IR sensors and the call buttons (Floor input) and a power supply.The output from the microcontroller is MOTOR_UP and MOTOR_DOWN sent to the motor driver which then drives the motor according to the bit sequence provided to the motor driver. The other outputs are indicators(LED) at each floor.



IMPLEMENTATION

CODE SNIPPETS :

Changing the frequency of clock and Using Clock for changing states

The clock is mapped to the 'E3' pin of the FPGA Board. The frequency of the inbuilt clock of the FPGA is 100MHZ but by using a counter we reduce the frequency of the clock to 20KHz. The lift control system works at 20KHz frequency. This code snippet shows the code where we are changing the state of our control system from the rising edge of the clock. Also, after changing the state we are driving the motor according to the current_state value. If asynchronous reset is on then lift will go in MOVING_DOWN_2 state and then finally to its initial state which is FLOOR0 in our case otherwise on every positive edge of clk_20KHz the current state will change to next_state.

```
reg clk_20KHz;
integer counter = 1;

always @(posedge clk) begin
    if (counter == 2500) begin
        counter = 1;
        clk_20KHz = ~clk_20KHz;
    end
    else begin
        counter = counter + 1;
    end
end
```

```
always @(posedge clk_20KHz or posedge reset) begin
    if (reset) begin
        current_state <= MOVING_DOWN_2;
    end else begin
        current_state <= next_state;
    end
end
```

Driving the Motor Based on state

We know that the lift is moving upwards when we are in state MOVING_UP_1 and MOVING_UP_2, and the motor is moving downwards when we are in state MOVING_DOWN_1 and MOVING_DOWN_2. When we want to go downwards we want to have (motor_up, motor_down) = (0, 1) and similarly if we want to go upwards we set values as (motor_up, motor_down) = (1,0). When we want to put the motor on halt we make values of motor_up and motor_down as 0.

```
MOVING_UP_1: begin // Moving Up 1
    motor_up = 1;
    motor_down = 0;
    indicator0 = 0;
    indicator1 = 0;
    indicator2 = 0;
    if (middle_minus_sensor && middle_plus_sensor) begin
        next_state = FLOOR1; // Reached Floor 1
        motor_up = 0; // Stop motor
        motor_down = 0;
        indicator1 = 1;
    end else begin
        next_state = MOVING_UP_1;
    end
end
MOVING_UP_2: begin // Moving Up 2
    motor_up = 1;
    motor_down = 0;
    indicator0 = 0;
    indicator1 = 0;
    indicator2 = 0;
    if (top_sensor) begin
        next_state = FLOOR2; // Reached Floor 2
        motor_up = 0; // Stop motor
        motor_down = 0;
        indicator2 = 1;
    end else begin
        next_state = MOVING_UP_2;
    end
end
```

```
MOVING_DOWN_1: begin // Moving Down 1
    motor_up = 0;
    motor_down = 1;
    indicator0 = 0;
    indicator1 = 0;
    indicator2 = 0;
    if (middle_minus_sensor && middle_plus_sensor) begin
        next_state = FLOOR1; // Reached Floor 1
        motor_up = 0; // Stop motor
        motor_down = 0;
        indicator1 = 1;
    end else begin
        next_state = MOVING_DOWN_1;
    end
end
MOVING_DOWN_2: begin // Moving Down 2
    motor_up = 0;
    motor_down = 1;
    indicator0 = 0;
    indicator1 = 0;
    indicator2 = 0;
    if (bottom_sensor) begin
        next_state = FLOOR0; // Reached Floor 0
        motor_up = 0; // Stop motor
        motor_down = 0;
        indicator0 = 1;
    end else begin
        next_state = MOVING_DOWN_2;
    end
end
```

State Management based on call buttons

This code snippet manages state changes triggered by input from call buttons. It ensures that when one button is pressed, the system ignores inputs from the other two buttons, effectively handling one input at a time. Additionally, it updates the LEDs corresponding to each floor, meeting the specified requirement of the elevator controller. Similarly we can write code for every state and for every state we can write according to inputs and outputs.

```
always @(*) begin
    case (current_state)
        FLOOR0: begin // Floor 0
            indicator0 = 1;
            indicator1 = 0;
            indicator2 = 0;
            if (call0) begin // Button for floor 0 pressed
                next_state = FLOOR0;
                motor_up = 0; // Stop motor
                motor_down = 0;
            end else if (call1) begin // Button for floor 1 pressed
                next_state = MOVING_UP_1; // Move up to Floor 1
            end else if (call2) begin // Button for floor 2 pressed
                next_state = MOVING_UP_2; // Move up to Floor 2
            else begin
                next_state = FLOOR0;
                motor_up = 0; // Stop motor
                motor_down = 0;
            end
        end
```

State Management based on IR Sensor Input

This code snippet verifies if the elevator has arrived at any floor by examining data from the relevant IR sensors. It includes a check to ensure that the inputs from the other IR sensors are '0', as failing to do so would compromise the accuracy of the floor state detection.

```
if bottom_sensor = '1' && middle_minus_sensor = '0' && middle_plus_sensor = '0' && top_sensor = '0' begin
    if MOVING_DOWN_2 begin
        current_state = FLOOR0;
    end
end else if bottom_sensor = '0' && middle_minus_sensor = '1' && middle_plus_sensor = '1' && top_sensor = '0' begin
    if MOVING_UP_1 || MOVING_DOWN_1 begin
        current_state = FLOOR1;
    end
end else if bottom_sensor = '0' && middle_minus_sensor = '0' && middle_plus_sensor = '0' && top_sensor = '1' begin
    if MOVING_UP_2 begin
        current_state = FLOOR2;
    end
end
```

CONTRIBUTION

NAME	CONTRIBUTION
GHORI ZEEL JIVRAJBHAI(202201287)	HARDWARE IMPLEMENTATION, VHDL CODE, STATE DIAGRAM, REPORT(code snippet), BLOCK DIAGRAM
ADIT SHAH(202201289)	REPORT
NAISARGI PATEL(202201291)	HARDWARE IMPLEMENTATION, CODE
DEVAMM PATEL(202201292)	STATE DIAGRAM(initial), CODE, HARDWARE IMPLEMENTATION
VIVEK CHAUDHARI(202201294)	REPORT, STATE DIAGRAM,BLOCK DIAGRAM
SHOURYA NAHAR(202201296)	HARDWARE IMPLEMENTATION