최종 프로젝트 보고서

학　부: 전자전기공학부

제출일: 2023.06.15

과목명: 디지털하드웨어시스템설계

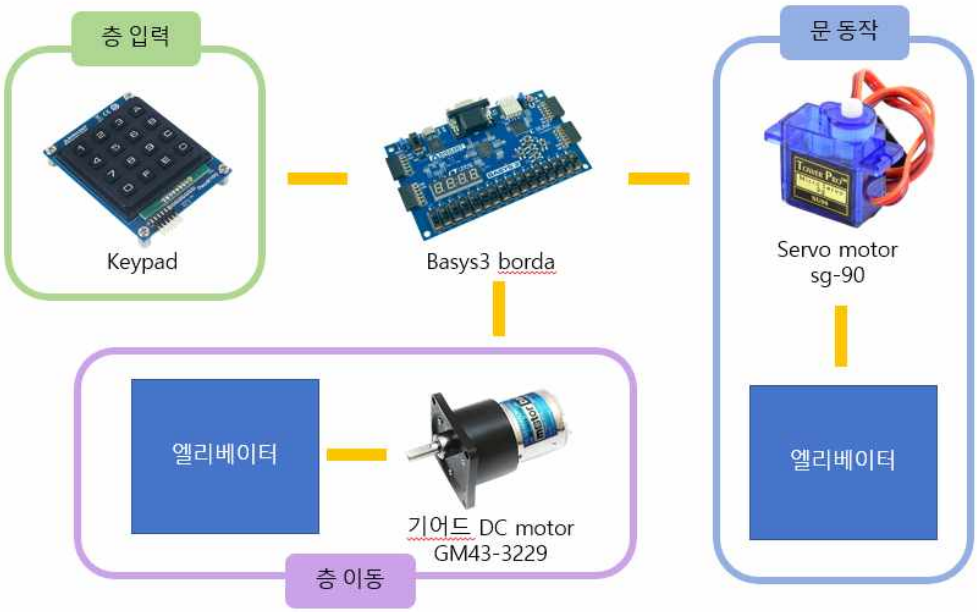
교수명: 김현진 교수님

분　반: 1분반

학　번: 32210776, 32181244,
32183739, 32194431

성　명: 김수민, 김현욱, 이호성, 조혜민

프로젝트 보고서

프로젝트 정보	
프로젝트명	FPGA Elevator
팀원/역할	김현욱(32181244) & 김수민(32210776) / Control Algorithm, Pmod Keypad 및 Segment 제어 이호성(32183739) / main motor(층수 이동 DC모터) 제어 및 H/W 제작 조혜민(32194431) / door motor(문열림 서보모터) 제어 및 H/W 제작
프로젝트 소개	Verilog를 이용하여 엘리베이터 FSM을 설계하고 FPGA를 이용하여 회로 동작을 확인할 수 있다. 본 회로에서는 각 층별로 state를 지정하고 층 정보를 Pmod pack 키패드를 통해 입력받아(순차부 Combinational part) 이동할 층에 맞게 DC모터와 서보모터를 제어하여(출력부) 동작한다.
구성도	
동작 방법	1층부터 4층까지 이동할 수 있는 엘리베이터를 제작하였다. 엘리베이터의 입력은 문을 제어하는 열림 버튼, 닫힘 버튼 두 가지와 엘리베이터 가고자 하는 층을 설정할 수 있는 keypad가 있다. 이 엘리베이터는 열림 버튼을 눌렀을 때만 keypad 입력을 받을 수 있고 닫힘 버튼을 누르면 keypad 입력이 불가능하다. 열림 버튼을 누르면 서보모터가 동작하여 엘리베이터의 문이 열리고 층 설정 후 닫힘 버튼을 누르면 서보모터가 반대로 동작해 엘리베이터 문을 닫은 뒤 DC모터를 동작해 원하는 층으로 이동한다.

I. 프로젝트 개요

1. 프로젝트 소개

- ㅇ 1층부터 4층까지 이동할 수 있는 엘리베이터로 층 이동과 문 개폐를 할 수 있도록 하드웨어를 제작하였다.
- ㅇ Keypad를 사용하여 엘리베이터의 층을 입력받는다.
- ㅇ 열림 버튼을 눌렀을 때 목표 층을 설정할 수 있고 닫힘 버튼을 눌렀을 때는 층 설정을 할 수 없다.

2. 개발목표 및 역할

ㅇ 최종 개발목표

- 목표 층을 keypad로 입력받아 설정할 수 있다. 1 ~ 4 외의 입력은 긴급 상황 버튼으로 설정하여 입력이 들어올 경우 1층으로 이동한다.
- 현재 층과 목표 층의 상태를 비교한 뒤 목표 층으로 이동할 수 있다.
- 목표 층에 도달 시 버튼을 통해 엘리베이터 문을 열고 닫을 수 있다.

ㅇ 역할

- 김수민, 김현욱 / Control Algorithm, Keypad및 Segment 제어

  엘리베이터의 동작을 제어하는 가장 상위 모듈이다. 엘리베이터가 동작할 수 있는 전체적인 Algorithm을 설계하였다. 열림버튼을 눌렀을 때 keypad의 입력을 받고 문이 열릴 수 있도록 하였으며 닫힘버튼을 눌렀을 때는 keypad 입력을 받지 않도록 하고 문이 닫힌 후 목표 층수로 이동할 수 있도록 설계하였다. 문을 닫고 엘리베이터가 동작하는 중에는 문이 열리지 않도록 'block_open' 비트를 이용해 제어하였다. 층수를 구분하기 위한 segment 동작 제어를 설정하였다. Pmod 포트를 통해 Keypad의 입력을 받아 1, 2, 3, 4를 선택했을 때 Basys3 board의 세그먼트에 입력받은 숫자를 나타내었고, 이를 제외한 입력은 X를 나타내었다. 이때 키패드로부터 입력을 받을 때는 버튼의 tickle 현상을 통제하기 위하여 카운터를 사용하였다.

- 이호성 / main motor 제어 및 H/W 제작

  엘리베이터가 층 이동을 할 수 있는 건물과 엘리베이터를 들어 올리는 도르래를 제작하였다. 메인 컨트롤러 모듈로부터 동작을 시작하는 신호와 이동해야하는 층수에 대한 정보, 위 아래의 방향에 대한 정보를 입력 받아 엔코더를 사용하여 DC모터를 정밀하게 제어해 1층부터 4층까지 움직일 수 있도록 하였다. 모터의 동작을 마친 후에는 동작 종료를 알리는 신호를 메인 컨트롤러 모듈로 출력한다.

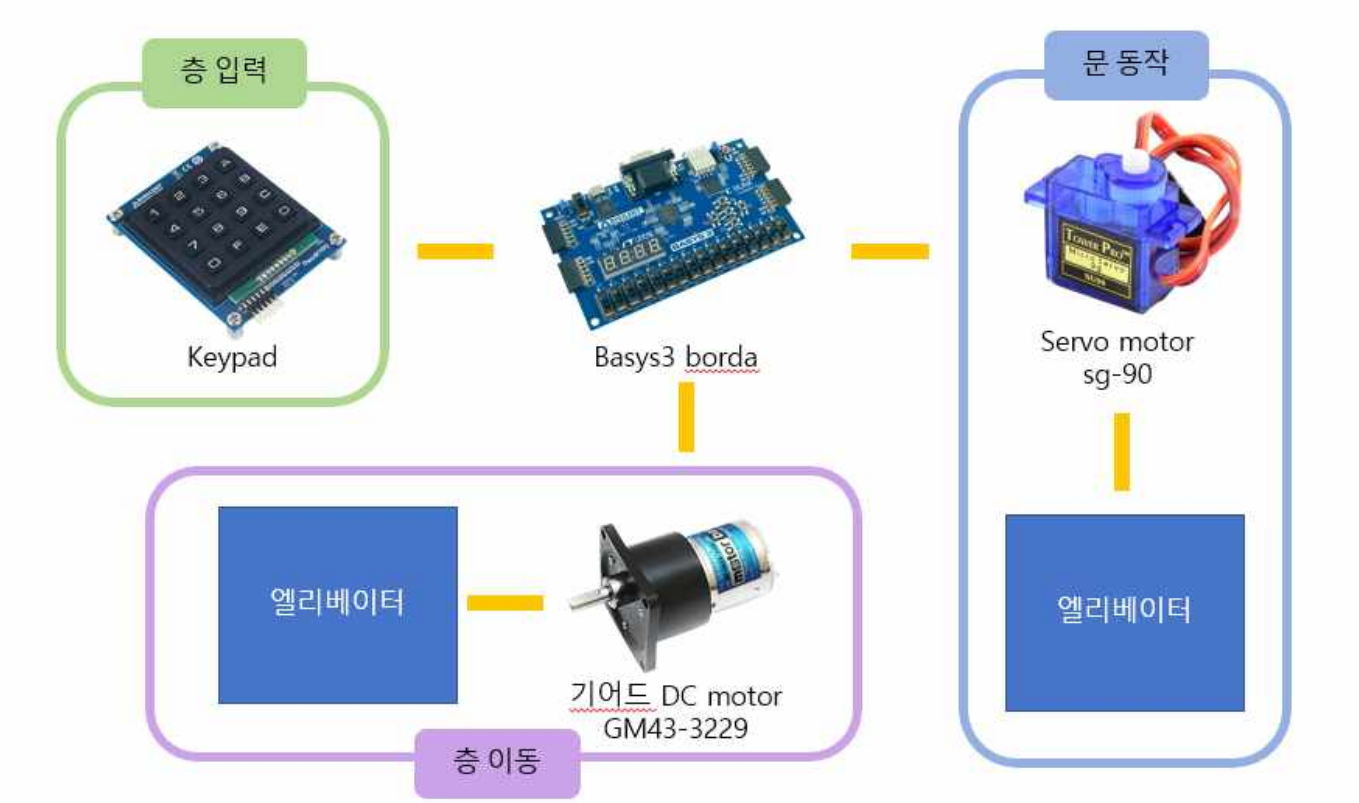
- 조혜민 / door motor 제어 및 H/W 제작

  층을 이동하는 엘리베이터를 제작하였다. 층마다 문을 만들어 열림버튼을 눌렀을 때

엘리베이터가 도착한 층의 문이 열리게 하였고 닫힘버튼을 눌렀을 때 문을 닫을 수 있게 하였다. 엘리베이터의 문은 서보모터를 사용해 열고 닫을 수 있도록 제어하였다. 서보모터는 제어시 카운터를 사용하여 90도 이상 회전하지 않도록 설계하였다.

II. 프로젝트 내용

1. 구성도



2. 소스 코드 및 설명

- elevator.v

```
module elevator(clk, open_btn, close_btn, seg, an, JA, JB, JC);
input clk;                                      //메인 클럭
input [0:0] open_btn, close_btn;                 //열림 버튼과 닫힘 버튼 변수
inout [7:0] JA;                                 //pmod 제어
inout [3:0] JB;                                 //pmod 제어
inout [3:0] JC;                                 //pmod 제어
output reg[6:0] seg;                            //7-segment 제어
output reg[3:0] an;                             //7-segment 제어
reg[0:0] door_open = 1'b0;                      //문열림 상태 판단 변수
reg[3:0] present_floar = 4'b0001;                //현재 층 저장 변수
reg[3:0] set_floar;                             //목표 층 저장 변수
wire[3:0] goal_floar;                           //keypad 출력 값
wire signed [31:0] floor_move;                   //motor 제어의 입력으로 쓰이는 움직여야하는 층 수
```

```
reg[0:0] block_open;                            //motor 동작 중 열림 버튼 제어 변수
assign floor_move = set_floar - present_floar;  //motor 제어의 입력으로 쓰이는 움직여야하는 층 수

pmod_keypad keypad(                             //keypad 모듈 instance
        .clk(clk),
        .col(JA[3:0]),
        .row(JA[7:4]),
        .key(goal_floar)
    );

door_control door(                              //문 제어 모듈 instancd
    .clk(clk),
    .door_open(door_open),
    .present_floar(present_floar),
    .set_floar(set_floar),
    .block_open(block_open),
    .JC(JC)
);

wire moving_cleck_e;                            //모터의 작동 상태 0 : stop
reg move_stop_start_e = 0;                      //모터 시작 제어 변수
main_motor mt(                                  //모터 제어 모듈 instance
        .move_motor_A(JB[0]), .move_motor_B(JB[1]),
        .moving_check(moving_cleck_e),
        .move_encoder(JB[3:2]),
        .floor_move_cnt(floor_move),
        .move_stop_start(move_stop_start_e), .move_clk(clk)
    );

always@(posedge clk)
begin

    if(moving_cleck_e != 1)                     //모터 동작을 멈췄을 때
        block_open <= 1'b0;                     //motor 동작 중 열림 버튼 제어 변수 초기화

    else if(moving_cleck_e)                     //모터 동작 중
        move_stop_start_e <= 0;                 //모터 동작 trriger 초기화


    if(open_btn == 1 && block_open == 0)        //모터 동작을 안할 때 열림 버튼을 눌렀을 때
    begin
        door_open <= 1'b1;                      //문열림 변수
        present_floar <= set_floar;             //현재 층 저장
    end
    else if(close_btn)                          //닫힘 버튼을 눌렀을 때
    begin
        door_open <= 1'b0;                      //문열림 변수
```

```
        move_stop_start_e <= 1'b1;              //모터 동작 trriger
        block_open <= 1'b1;                     //동작 중 열림 버튼 방지를 위한
    end

    if(door_open == 1)                          //문이 열려있을 때
    begin
        an <= 4'b1011;                          //2번째 7-segment에 정보 표시
    end

    else if(door_open == 0)                     //문이 닫혀있을 때
    begin
        an <= 4'b1110;                          //4번째 7-segment에 정보 표시
    end
end

always@(door_open)                                  //문이 열렸을 때
begin
    if(door_open == 1)
    begin
       case(goal_floar)                             //keypad에서 받는 정보
            4'h1:begin
                    seg[6:0] <= 7'b1111001;       //1
                    set_floar <= 4'b0001;         //1층을 변수에 저장
                end

            4'h2:begin
                    seg[6:0] <= 7'b0100100;       //2
                    set_floar <= 4'b0010;         //2층을 변수에 저장
                end
            4'h3:begin
                    seg[6:0] <= 7'b0110000;       //3
                    set_floar <= 4'b0011;         //3층을 변수에 저장
                end

            4'h4:begin
                    seg[6:0] <= 7'b0011001;       //4
                    set_floar <= 4'b0100;         //4층을 변수에 저장
                end

            default:begin
                    seg[6:0] <= 7'b0001001;       //예외 처리. 이외 층수의 버튼을 누르면 X
                    set_floar <= 4'b0001;         //이외 층수의 버튼을 누르면 1층으로 설정
                    end
        endcase
     end
 end
endmodule
```

- door_control.v

```
`timescale 1ns / 1ps
module door_control(
    input clk,                      //클럭
    input door_open,                //문열림 상태 변수
    input [3:0] present_floar,      //현재 층수 변수
    input [3:0] set_floar,          //목표 층수 변수
    input [0:0] block_open,         //문닫힘 상태 변수
    output reg [3:0] JC             //pmod 제어
);
    // Setting Counter
    reg [20:0] counter;             //counter 사용 변수
    reg [1:0] servo_state;          //서보모터 출력 변수
    reg [16:0] control = 0;
    reg toggle = 1;
    always @(posedge clk) begin
        if(door_open == 1 && block_open == 0)              //문 열림 버튼 작동
            begin
                counter <= counter + 1;                    //counter
                if(counter == 'd999999)
                    counter <= 0;
                if(counter < ('d100000 + control)) begin
                    if(present_floar == 4'b0001)           //현재 1층
                        JC <= 4'b0001;                     //1층 문 서보모터 작동

                    else if(present_floar == 4'b0010)      //현재 2층
                        JC <= 4'b0010;                     //2층 문 서보모터 작동
                    else if(present_floar == 4'b0011)      //현재 3층
                        JC <= 4'b0100;                     //3층 문 서보모터 작동
                    else if(present_floar == 4'b0100)      //현재 4층
                        JC <= 4'b1000;                     //4층 문 서보모터 작동
                    end
                else
                    JC <= 0;

                if(control == 'd100000)
                    toggle <= 0;
                if(counter == 0)
                    begin
                        if(toggle == 1)
                            control <= control + 500;
                    end
            end
```

```
        if(block_open == 1)                                 //문 닫힘 버튼 작동
            begin
                counter <= counter + 1;
                if(counter == 'd999999)
                    counter <= 0;
                if(counter < ('d100000 + control))begin
                    if(present_floar == 4'b0001)            //현재 1층
                        JC <= 4'b0001;                      //1층 문 서보모터 작동
                    else if(present_floar == 4'b0010)       //현재 2층
                        JC <= 4'b0010;                      //2층 문 서보모터 작동
                    else if(present_floar == 4'b0011)       //현재 3층
                        JC <= 4'b0100;                      //3층 문 서보모터 작동
                    else if(present_floar == 4'b0100)       //현재 4층
                        JC <= 4'b1000;                      //4층 문 서보모터 작동
                    end
                else
                    JC <= 0;

                if(control == 'd100000)
                    toggle <= 0;
                if(control == 0)
                    toggle <= 1;
                if(counter == 0)
                    begin
                        if(toggle == 0)
                            control <= control - 500;
                    end
            end
        end
endmodule
```

- pmod_keypad.v

```
module pmod_keypad(
    input clk,                  //클럭
    input [3:0] row,            //열
    output reg [3:0] col,       //행
    output reg [3:0] key        //keypad 값
    );
   localparam BITS = 20;        //20비트로 구성된 카운터의 비트수를 나타내는 파라미터

   localparam ONE_MS_TICKS = 100000000 / 1000;     //1ms 클럭틱수

   localparam SETTLE_TIME = 100000000 / 1000000;    //1us 입력신호 안정시간

    wire [BITS - 1 : 0] key_counter;            //keypad 입력신호의 시간 카운트
    reg rst = 1'b0;                             //카운터 초기화 reset
```

```
counter_n #(.BITS(BITS)) counter(           //카운터 모듈
    .clk(clk),
    .rst(rst),
    .q(key_counter)
);
always @ (posedge clk)
begin
  case (key_counter)               //key_counter 값에 따라
        0:                         //0일 때
            rst <= 1'b0;           //rst

        ONE_MS_TICKS:              //ONE_MS_TICKS일 때
            col <= 4'b0111;        //keypad 첫번째 열

        ONE_MS_TICKS + SETTLE_TIME: //안정시간 지나면
        begin
            case (row)
                4'b0111:
                    key <= 4'b0001; // key값 1
                4'b1011:
                    key <= 4'b0100; // key값 4
                4'b1101:
                    key <= 4'b0111; // key값 7
                4'b1110:
                    key <= 4'b0000; // key값 0
            endcase
        end

        2 * ONE_MS_TICKS:          //2ms 지나면
            col <= 4'b1011;        //두번째 열

        2 * ONE_MS_TICKS + SETTLE_TIME:
        begin
            case (row)
                4'b0111:
                    key <= 4'b0010; // key값 2
                4'b1011:
                    key <= 4'b0101; // key값 5
                4'b1101:
                    key <= 4'b1000; // key값 8
                4'b1110:
                    key <= 4'b1111; // key값 F
            endcase
        end
        3 * ONE_MS_TICKS:          //3ms 지나면
            col <= 4'b1101;
```

```
            3 * ONE_MS_TICKS + SETTLE_TIME:

            begin
                case (row)
                    4'b0111:
                        key <= 4'b0011; // key값 3
                    4'b1011:
                        key <= 4'b0110; // key값 6
                    4'b1101:
                        key <= 4'b1001; // key값 9
                    4'b1110:
                        key <= 4'b1110; // key값 E
                endcase
            end

            // 4ms
            4 * ONE_MS_TICKS:           //4ms 지나면
                col <= 4'b1110;         //4번째 열 선택

            4 * ONE_MS_TICKS + SETTLE_TIME:
            begin
                case (row)
                    4'b0111:
                        key <= 4'b1010; // key값 A
                    4'b1011:
                        key <= 4'b1011; // key값 B
                    4'b1101:
                        key <= 4'b1100; // key값 C
                    4'b1110:
                        key <= 4'b1101; // key값 D
                endcase
                // reset the counter
                rst <= 1'b1;
            end
        endcase
    end
endmodule
```

- counter_n.v

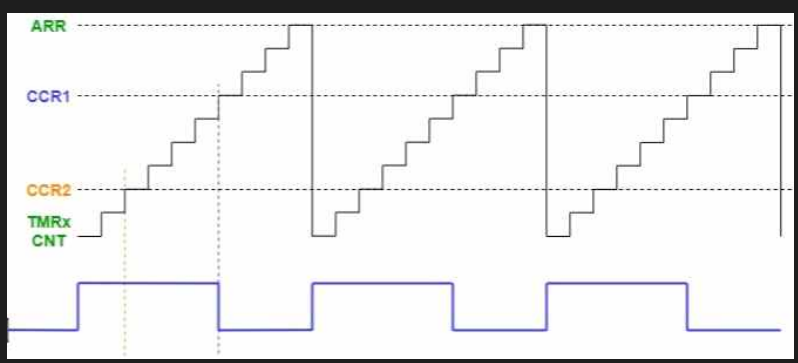
```
`timescale 1ns / 1ps
module counter_n
  # (BITS = 4)                          //매개변수 BITS를 4로 설정
    (
    input clk,
    input rst,
    output tick,
    output [BITS - 1 : 0] q
    );
  reg [BITS - 1 : 0] rCounter = 0;        //카운터 레지스터
    always @ (posedge clk, posedge rst)
      if (rst)                          //rst 활성화되면
            rCounter <= 0;              //rCounter 0으로 초기화
        else                            //그렇지 않으면
            rCounter <= rCounter + 1;   //1씩 증가
    assign q = rCounter;                //q 출력신호를 rCounter값에 연결

  assign tick = (rCounter == 2 ** BITS - 1) ? 1'b1 : 1'b0;  //tick 출력 신호 설정

endmodule
```

- main_motor.v

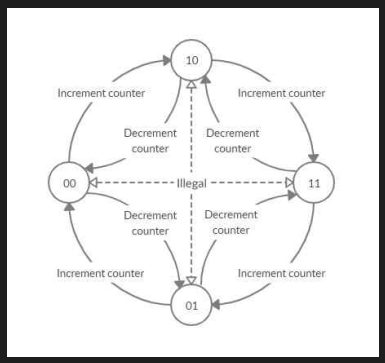
```
`timescale 1ns / 1ps
//엘리베이터 모터의 상승 하강을 관리합니다.
module main_motor(
        output move_motor_A, move_motor_B,  //모터 +, 모터 - 출력
        output reg moving_check = 0,        //동작중 = 1, 비동작 = 0 출력
        input [1:0] move_encoder,           //엔코더 입력
        input signed [31:0] floor_move_cnt, //상승할 칸수, 하강한 칸수
        input move_stop_start, move_clk     //시작 flag, 클럭
    );
    parameter floor_length = 32'd7000;      //한층 올라갈 엔코더 값 입니다. (한칸: 7000)

    reg signed [31:0] goal_encoder_data = 0;    //목표 엔코더 값
    wire [31:0] current_encoder_data;           //현재 엔코더 값
    reg [31:0] move_pwm = 0;                    //모터 PWM 값
    reg move_dir = 0, move_brake = 1;           //모터 방향, 모터 브레이크

    //모터 엔코더 입력
    get_encoder_data motor_encoder( .encoder_data(current_encoder_data),
                                    .encoder_reset(0),
                                    .encoder_clk(move_clk),
                                    .encoder_state(move_encoder));
```

```
//모터 제어
motor_ctrl mt_ctrl( .motor_A(move_motor_A),
                    .motor_B(move_motor_B),
                    .motor_pwm(move_pwm),
                    .motor_dir(move_dir),
                    .motor_brake(move_brake),
                    .motor_clk(move_clk));
reg [31:0] move_distance;   // 움직일 거리

always@(posedge move_clk) begin //always clk
    if(move_stop_start == 1)    //시작명령시
        if(moving_check == 0) begin //엘리베이터가 움직이지 않을 때
            //목표 설정
            goal_encoder_data = goal_encoder_data + (floor_move_cnt * floor_length);
            moving_check <=1;     //움직이기 시작
        end
    move_distance =  current_encoder_data - goal_encoder_data;      //움직일 거리 업데이트
    //움직일 거리양수 //500이상 최고속도 PWM 100
    if(move_distance > 32'hFFFF + 32'd500) begin
        move_pwm <= 32'd1000;
        move_brake <= 0;
        move_dir <= 0;
        moving_check = 1;
    end
    //움직일 거리음수 //-500이하 최고속도 PWM 100
    else if(move_distance < 32'hFFFF - 32'd500) begin
        move_pwm <= 32'd1000;
        move_brake <= 0;
        move_dir <= 1;
        moving_check = 1;
    end
    //움직일 거리양수 //100이상 PWM 600
    else if(move_distance > 32'hFFFF + 32'd100) begin
        move_pwm <= 32'd600;
        move_brake <= 0;
        move_dir <= 0;
        moving_check = 1;
    end
    //움직일 거리음수 //100이하 PWM 600
    else if(move_distance < 32'hFFFF - 32'd100) begin
        move_pwm <= 32'd600;
        move_brake <= 0;
        move_dir <= 1;
        moving_check = 1;
    end
```

```
        //움직일 거리양수 //10이상 최소속도 PWM 380
        else if(move_distance > 32'hFFFF + 10'd10) begin
            move_pwm <= 32'd380;
            move_brake <= 0;
            move_dir <= 0;
            moving_check = 1;
        end

        //움직일 거리음수 //10이하 최소속도 PWM 380
        else if(move_distance < 32'hFFFF - 10'd10) begin
            move_pwm <= 32'd380;
            move_brake <= 0;
            move_dir <= 1;
            moving_check = 1;
        end
        else begin                  //정지, 움직임 flag = 0
            move_pwm <= 32'd0;
            move_brake <= 1;
            moving_check = 0;
        end
    end
endmodule

// PWM 생성기
module pwm_generator(
    output reg      pwm_state,                  //출력
    input  [31:0]   pwm_period, pwm_cc, pwm_clk //주기, count, 클럭
    );
    reg [31:0] count = 0;

    always @(posedge pwm_clk) begin     //pwm 클럭
        if(count >= pwm_period)         //count가 period 이상일대 0
            count <= 32'b0;
        else
            count <= count + 32'b1;     //1
        if(count <= pwm_cc)
            pwm_state <= 1'b1;
        else
            pwm_state <= 1'b0;
    end
endmodule
```

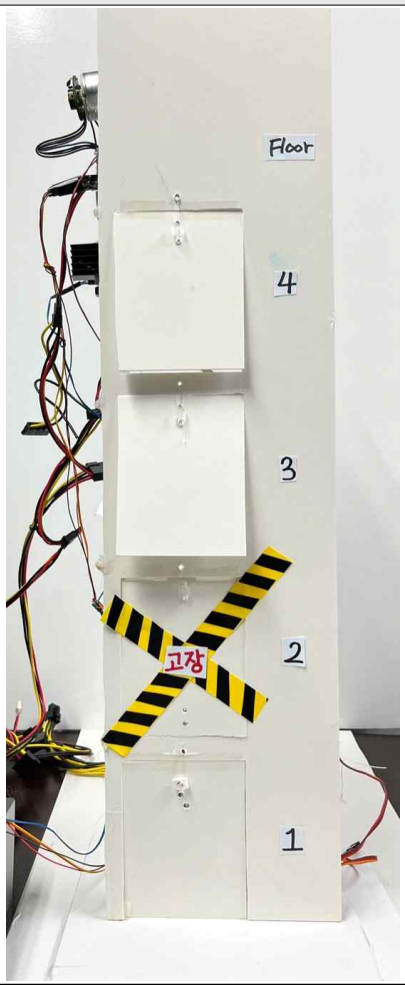
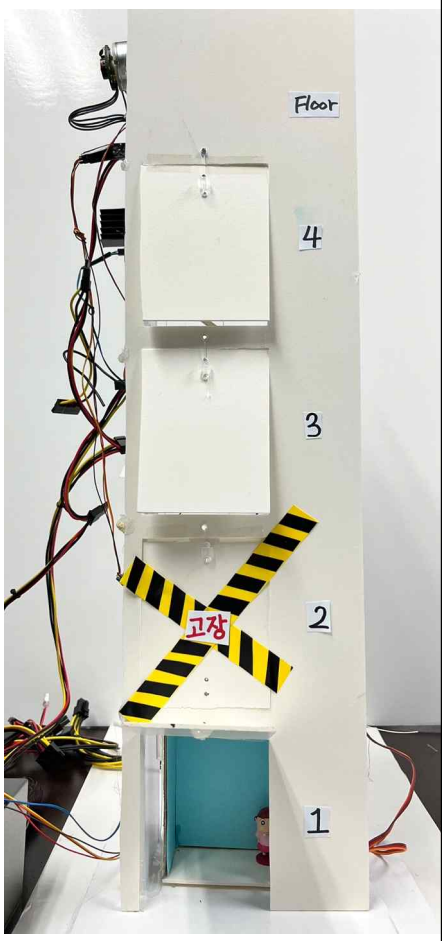


```
module get_encoder_data(
    output reg [31:0] encoder_data = 32'hFFFF,  //초기 엔코데 데이터 FFFF
                                                //정수 이슈로 초기값이 FFFF로 설정함
    input encoder_reset,    //encoder 리셋
    input encoder_clk,      //encoder 클럭
    input [1:0] encoder_state   //encoder 핀 입력
    );
    reg [1:0] n_state;      //상태
    always@(posedge encoder_clk)    //엔코더 클럭
        if(encoder_reset == 1'b1)   //reset 엔코더 0
            encoder_data[15:0] = 0;
        else
            case(encoder_state)
            2'b00 : begin if(n_state == 10)        encoder_data = encoder_data - 1; //감소
                    else if(n_state == 01)   encoder_data = encoder_data + 1;       //증가
                    n_state = 2'b00;end
            2'b10 : begin if(n_state == 11)        encoder_data = encoder_data - 1; //감소
                    else if(n_state == 00)   encoder_data = encoder_data + 1;       //증가
                    n_state = 2'b10;end
            2'b11 : begin if(n_state == 01)        encoder_data = encoder_data - 1; //감소
                    else if(n_state == 10)   encoder_data = encoder_data + 1;       //증가
                    n_state = 2'b11;end
            2'b01 : begin if(n_state == 00)        encoder_data = encoder_data - 1; //감소
                    else if(n_state == 11)   encoder_data = encoder_data + 1;       //증가
                    n_state = 2'b01; end
              endcase
endmodule


//모터 제어
module motor_ctrl(
    output              motor_A, motor_B,   //모터 A핀 B핀
    input  [31:0]       motor_pwm,          //모터 PWM 설정
    input               motor_dir, motor_brake, motor_clk   //모터 방향, brake, 클럭
    );
    reg [31:0] motor_pwm_A, motor_pwm_B;    //PWM A값, PWM B값

    pwm_generator pwmA( .pwm_state(motor_A),    //모터 A핀 설정
                        .pwm_period(32'd1000),
                        .pwm_cc(motor_pwm_A),
                        .pwm_clk(motor_clk));
    pwm_generator pwmB( .pwm_state(motor_B),    //모터 B핀 설정
                        .pwm_period(32'd1000),
                        .pwm_cc(motor_pwm_B),
                        .pwm_clk(motor_clk));
```



```
    always @(motor_pwm, motor_dir, motor_brake) begin   //모터 제어 설정
        if(motor_brake) begin               //brake
            motor_pwm_A <= 0;
            motor_pwm_B <= 0;
        end
        else begin
            if(motor_dir == 1'b0) begin     //정방향
                motor_pwm_A <= motor_pwm;
                motor_pwm_B <= 0;
            end
            else begin                      //역방향
                motor_pwm_A <= 0;
                motor_pwm_B <= motor_pwm;
            end
        end
    end
endmodule
```

3. 결과물

결과물 이미지	
	

Ⅲ. 기타

1. 프로젝트 수행 과정에서의 문제점 및 보완할 점

ㅇ 기존 사용하는 C언어로 코드를 작성하는 것이 아니라 새로운 언어로 코드를 작성하는 게 쉽지 않았다. 기존 C언어 등의 다른 프로그래밍 언어와 다르게 회로를 Description하는 병렬적 처리를 하는 Verilog의 특성상 기존의 프로그래밍을 할 때의 논리적 순서를 따지는 것과 다른 것이 어색하였으며 이로 인해 한 층의 높이를 알고 원하는 층으로 정확히 도착할 수 있게 모터를 제어하는 코드를 작성하는 것이 어려웠다.

ㅇ always의 이해도가 낮아 always문을 사용하여 코드를 작성하였는데 닫힘버튼을 눌렀음에도 keypad의 입력이 받아졌다. always문을 다시 공부하고 정확하게 수행하게 하여 문제를 해결할 수 있었다.

ㅇ 현재 설계 결과로는 엘리베이터 동작 중 키패드 입력 후 문을 열 때 버그가 발생하는 점, 층 도착 후 문 개폐가 수동으로 이루어지는 문제점 등이 있다. 향후 해당 문제도 보완한다면 조금 더 완벽한 엘리베이터를 설계할 수 있을 것이다.

2. 참고 자료

ㅇ Basys3 Reference Manual

ㅇ Vivado Tutorial

ㅇ basys3 pmod keypad 관련 포스팅

ㅇ Pmod KYPD Digilent Reference

3. GitHub Link

- https://github.com/cho-hm02123/Digital_Hardware_System_Design_Challenge