امیرحسین رضائی
۴۰۱۱۰۵۹۸۹
Elevator_Controller

```
module Elevator_Control (
    input clk,
    input reset,
    input [4:0] sensor,
    input [4:0] call_button,
    input [4:0] dest_button,
    output reg motor_dir,
    output reg motor_move
);
```

در این بخش ورودی و خروجی مشخص شده اند که sensor طبقه مورد نظر را مشخص میکند و call_button تعیین میکند که کدام کلید فشرده شده و dest_button مقصد را مشخص میکند و خروجی motor_dir خروجی مقصد بوده و motor_move مشخص میکند که اسانسور به بالا حرکت میکند یا به پایین

```
parameter IDLE_GROUND = 3'b000,
          IDLE_1      = 3'b001,
          IDLE_2      = 3'b010,
          IDLE_3      = 3'b011,
          IDLE_4      = 3'b100,
          MOVING_UP   = 3'b101,
          MOVING_DOWN = 3'b110,
          STOPPING    = 3'b111;
```

این بخش state را مشخص میکند و به حرکت یک عدد مطابق بالا نسبت میدهد

```
parameter GROUND = 3'b000,
          FLOOR_1 = 3'b001,
          FLOOR_2 = 3'b010,
          FLOOR_3 = 3'b011,
          FLOOR_4 = 3'b100;
```

در این بخش طبقه های متفاوت مانند انچه در بالا مشاهده میشود تعریف میشوند

```
reg [2:0] state, next_state;
reg [4:0] call_requests;
reg [4:0] dest_requests;
reg [2:0] current_floor;
```

در این بخش رجیستر های متفاوت تعریف شده که کارکرد هر کدام با توجه به اسم انها مشخص است

```
initial begin
    state = IDLE_GROUND;
    call_requests = 5'b0;
    dest_requests = 5'b0;
    current_floor = GROUND;
    motor_dir = 0;
    motor_move = 0;
end
```

در این بخش رجیستر ها و state ها مقدار دهی اولیه میشوند

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        call_requests <= 5'b0;
        dest_requests <= 5'b0;
    end else begin
        call_requests <= call_requests | call_button;
        dest_requests <= dest_requests | dest_button;
    end
end
```

در این بخش call_request و dest_request اپدیت میشود و اگر reset یک باشد ریست شده و مقدار صفر میگیرند

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        state <= IDLE_GROUND;
        current_floor <= GROUND;
    end else begin
        state <= next_state;
        case (sensor)
            5'b00001: current_floor <= GROUND;
            5'b00010: current_floor <= FLOOR_1;
            5'b00100: current_floor <= FLOOR_2;
            5'b01000: current_floor <= FLOOR_3;
            5'b10000: current_floor <= FLOOR_4;
            default: current_floor <= current_floor;
        endcase
    end
end
```

این بخش مربوط به logic تغییر state است و در صورت reset شدن state و current_floor به مقدار دیفالت ان تغییر پیدا میکند و در غیر این صورت state فعلی برابر با next_state میشود و با استفاده از sensor طبقه فعلی مشخص میشود

```
always @(*) begin
    next_state = state;
    motor_dir = 0;
    motor_move = 0;

    case (state)
```

در این بخش خروجی مشخص خواهد شد و در بخش بعد حالت مختلف بررسی میشوند

```
IDLE_GROUND: begin
    if (call_requests[FLOOR_1] || dest_requests[FLOOR_1]) begin
        next_state = MOVING_UP;
        motor_dir = 1;
        motor_move = 1;
    end
end
```

این بخش مربوط به طبقه همکف است که همیشه به بالا حرکت میکند و مقصد ان همیشه طبقه یک است

```
IDLE_1: begin
    if (call_requests[GROUND] || dest_requests[GROUND]) begin
        next_state = MOVING_DOWN;
        motor_dir = 0;
        motor_move = 1;
    end else if (call_requests[FLOOR_2] || dest_requests[FLOOR_2]) begin
        next_state = MOVING_UP;
        motor_dir = 1;
        motor_move = 1;
    end
end
```

این بخش مربوط به طبقه اول است که با تشخیص این که به بالا حرکت خواهد کرد یا پایین (از روی call_requests و dest_request) حالات متفاوت را مقدار دهی میکند

```
IDLE_2: begin
    if (call_requests[FLOOR_1] || dest_requests[FLOOR_1]) begin
        next_state = MOVING_DOWN;
        motor_dir = 0;
        motor_move = 1;
    end else if (call_requests[FLOOR_3] || dest_requests[FLOOR_3]) begin
        next_state = MOVING_UP;
        motor_dir = 1;
        motor_move = 1;
    end
end
IDLE_3: begin
    if (call_requests[FLOOR_2] || dest_requests[FLOOR_2]) begin
        next_state = MOVING_DOWN;
        motor_dir = 0;
        motor_move = 1;
    end else if (call_requests[FLOOR_4] || dest_requests[FLOOR_4]) begin
        next_state = MOVING_UP;
        motor_dir = 1;
        motor_move = 1;
    end
end
IDLE_4: begin
    if (call_requests[FLOOR_3] || dest_requests[FLOOR_3]) begin
        next_state = MOVING_DOWN;
        motor_dir = 0;
        motor_move = 1;
    end
end
```

در این بخش همان منطق بخش اول برای طبقه های ۲و ۳و ۴ انجام میشود

```
MOVING_UP: begin
    if (sensor[FLOOR_4]) begin
        next_state = IDLE_4;
        motor_move = 0;
    end else if (sensor[FLOOR_3]) begin
        next_state = IDLE_3;
        motor_move = 0;
    end else if (sensor[FLOOR_2]) begin
        next_state = IDLE_2;
        motor_move = 0;
    end else if (sensor[FLOOR_1]) begin
        next_state = IDLE_1;
        motor_move = 0;
    end else begin
        next_state = MOVING_UP;
        motor_move = 1;
    end
    motor_dir = 1;
end
```

در این بخش حرکت به بالا برسی شده که با حالت بندی روی sensor و state فعلی حرکت به بالا را تشخیص میدهد

```
MOVING_DOWN: begin
    if (sensor[GROUND]) begin
        next_state = IDLE_GROUND;
        motor_move = 0;
    end else if (sensor[FLOOR_1]) begin
        next_state = IDLE_1;
        motor_move = 0;
    end else if (sensor[FLOOR_2]) begin
        next_state = IDLE_2;
        motor_move = 0;
    end else if (sensor[FLOOR_3]) begin
        next_state = IDLE_3;
        motor_move = 0;
    end else begin
        next_state = MOVING_DOWN;
        motor_move = 1;
    end
    motor_dir = 0;
end
```

در این بخش حرکت به پایین برسی شده که با حالت بندی روی sensor و state فعلی حرکت به پایین را تشخیص میدهد

```
STOPPING: begin
    if (sensor[current_floor]) begin
        next_state = state;
        motor_move = 0;
    end
end
```

در این قسمت حالت بدون حرکت را بررسی میکند که در ان next_state برابر state فعلی قرار میگیرد

```
always @(posedge clk) begin
    if (motor_move == 0 && sensor[current_floor]) begin
        call_requests[current_floor] <= 1'b0;
        dest_requests[current_floor] <= 1'b0;
    end
end
```

در این بخش در هر کلاک اگر نیاز به خالی و clear کردن request ها باشد انجام میشود

TB

```
module TB;
    reg clk;
    reg reset;
    wire move;
    wire direction;
    reg [4:0] intenral_buttons;
    reg [4:0] external_buttons;
    wire [4:0] current_floor;
    reg [1:0] prev_floor_indi [4:0];
    wire [1:0] floor_indi [4:0];

    elevator #(5) e_inst (
        .clk(clk),
        .state({move, direction}),
        .current_floor(current_floor),
        .reset(reset),
        .floor_indi(floor_indi)
    );
    elevator_controller #(5) ec_inst (
        .clk(clk),
        .external_buttons(external_buttons),
        .floor_indi(floor_indi),
        .reset(reset),
        .intenral_buttons(intenral_buttons),
        .current_floor(current_floor),
        .direction(direction),
        .move(move)
    );

    always #5 clk = ~clk;
```

در این بخش با سیم اتصالات داخلی را به هم متصل میکند و سیگنال ها را و رجیستر های مورد نیاز را تعریف میکند و کلاک را هم تعریف میکند

```
initial begin
    clk = 0;
    reset = 1;
    intenral_buttons = 0;
    external_buttons = 0;

    #10 reset = 0;
    external_buttons[0] = 1'b1;          // همکف
    #10 external_buttons[0] = 1'b0;      // همکف
    #10 intenral_buttons[3] = 1'b1;      // سوم
    #10 intenral_buttons[3] = 1'b0;      // سوم
    #200 external_buttons[2] = 1'b1;     // دوم
    #10 external_buttons[2] = 1'b0;      // دوم
    #10 external_buttons[4] = 1'b1;      // چهارم
    #10 external_buttons[4] = 1'b0;      // چهارم
    #10 external_buttons[1] = 1'b1;      // اول
    #10 external_buttons[1] = 1'b0;      // اول
    #100 intenral_buttons[2] = 1'b1;     // دوم و چون در مسیر اسانسور قرار دارد پس در ان توقف میکند
    #10 intenral_buttons[2] = 1'b0;      // همکف
    #2500 intenral_buttons[0] = 1'b1;    // اول
    #10 intenral_buttons[0] = 1'b0;      // همکف
    #150 intenral_buttons[4] = 1'b1;     // اول
    #10 intenral_buttons[4] = 1'b0;      // همکف
    #20 external_buttons[0] = 1'b1;      // اول
    #10 external_buttons[0] = 1'b0;      // همکف
    #1000;
    $stop();
end
```

در این بخش تست های متفاوتی انجام شده تا حالات متفاوتی چک شوند که در comment روند ان توضیح داده شده

توجه شود که در حالات دیگر توقفی شکل نمیگیرد چون در مسیر حرکت قرار نداشته و فقط در مقصد اسانسور می ایستد

```
always @(posedge clk) begin
    if (floor_indi !== prev_floor_indi) begin
        $display("at time %3d", $time, " Floor indicators changed:");
        $display("Current value:  %p", floor_indi);
    end
    prev_floor_indi <= floor_indi;
end
```

در این بخش هم دستور display برای مشاهده خروجی و state قرار گرفتن ان انجام شده

در زیر میتوان این خروجی را مشاهده کرد

```
# at time   5 Floor indicators changed:
# Current value:  '{0, 0, 0, 0, 0}
# at time  25 Floor indicators changed:
# Current value:  '{0, 0, 0, 0, 3}
# at time 125 Floor indicators changed:
# Current value:  '{0, 0, 0, 2, 1}
# at time 135 Floor indicators changed:
# Current value:  '{0, 0, 2, 1, 0}
# at time 145 Floor indicators changed:
# Current value:  '{0, 0, 3, 0, 0}
# at time 245 Floor indicators changed:
# Current value:  '{0, 0, 1, 2, 0}
# at time 255 Floor indicators changed:
# Current value:  '{0, 0, 0, 3, 0}
# at time 355 Floor indicators changed:
# Current value:  '{0, 0, 2, 1, 0}
# at time 365 Floor indicators changed:
# Current value:  '{0, 0, 3, 0, 0}
# at time 465 Floor indicators changed:
# Current value:  '{0, 2, 1, 0, 0}
# at time 475 Floor indicators changed:
# Current value:  '{0, 3, 0, 0, 0}
# at time 575 Floor indicators changed:
# Current value:  '{0, 1, 2, 0, 0}
# at time 585 Floor indicators changed:
# Current value:  '{0, 0, 1, 2, 0}
# at time 595 Floor indicators changed:
# Current value:  '{0, 0, 0, 1, 2}
# at time 605 Floor indicators changed:
# Current value:  '{0, 0, 0, 0, 3}
# at time 705 Floor indicators changed:
# Current value:  '{0, 0, 0, 2, 1}
# at time 715 Floor indicators changed:
# Current value:  '{0, 0, 0, 3, 0}
# at time 815 Floor indicators changed:
# Current value:  '{0, 0, 2, 1, 0}
# at time 825 Floor indicators changed:
# Current value:  '{0, 2, 1, 0, 0}
# at time 835 Floor indicators changed:
# Current value:  '{2, 1, 0, 0, 0}
# at time 845 Floor indicators changed:
# Current value:  '{3, 0, 0, 0, 0}
```