Design Document

در این سوال دو ماژول برای طراحی و کنترل کردن آسانسور در نظر گرفته شده است که در ادامه به توضیحات جزئیات هر یک خواهیم پرداخت. پیش از آغاز باید بیان کنیم که در این پروژه از System Verilog استفاده شده است زیرا برای هندل کردن صف حرکت آسانسور به طبقات مختلف نیازمند استفاده از Queue هستیم که در System Verilog موجود است. همچنین علاوه بر توضیحات بیان شده، تمامی قسمت های کد با استفاده از کامنت مناسب توضیح داده شده اند که میتوان برای فهم بهتر کد به آنها نیز مراجعه کرد.

۱- ماژول حالات طبقات (Asansor)

ورودی و خروجی:

این ماژول به عنوان ورودی یک سیگنال کلاک دریافت میکند که کلاک کنترل کننده آسانسور است. یک رجیستر دو بیتی با نام State دریافت میکند که وضعیت آسانسور را در دو بیت ذخیره میکند. نحوه ذخیره سازی به این صورت است که بیت اول آن مربوط به جهت حرکت آسانسور (بالا = ۱ و پایین = ۰) و بیت دوم مربوط به حرکت یا عدم حرکت آسانسور است (حرکت = ۱ و عدم حرکت = ۰). مقدار این دو بیت در ماژول دیگر محاسبه میشود و به عنوان ورودی به این ماژول داده خواهد شد که بر اساس استیت کنونی آسانسور، مقداردهی حالات طبقات که در ادامه توضیحات داده میشود، انجام خواهد شد. یک ثبات ۵ بیتی به نام WhichFloor نیز برای ذخیره سازی مکان کنونی آسانسور در نظر گرفته شده است و همچنین یک ثبات به صورت یک آرایه ۵ بیتی از آرایه های دوبیتی به نام FloorInOut وجود دارد که ورود و خروج به طبقات و ایست در آنها را ذخیره میکند. (۰۰ = ایست و ۱۰ = خروج از طبقه و ۰۱ = ورود به طبقه و ۱۱ = عدم استفاده از این طبقه در حال حاضر)

```
module Asansor (
    input Clk,                // Clock input
    input [1:0] State,        // State input to control elevator movement
    output reg [4:0] WhichFloor, // Output to indicate the current floor
    output reg [1:0] FloorInOut [4:0] // Output array to indicate floor states
);
```

Initial and local params:

در این ماژول پارامترهای ایست، بالا و پایین تعریف شده اند که مطابق توضیحات بیان شده برای State مقداردهی شده اند.

همچنین متغیر i برای شمارنده حلقه ها تعریف شده است.

در ابتدا نیز در یک بلاک initial مقدار حالت همه طبقات به صورت عدم استفاده تعیین شده است و طبقه کنونی آسانسور نیز همکف ست شده است.

```
    // Local parameters for elevator states
    localparam Stop = 2'b00,
               Up = 2'b11,
               Down = 2'b01;

    // Initial block to set initial values
    integer i;
    initial begin
        // Initialize all floor states to notStarting yet
        for (i = 0; i < 5; i = i + 1) begin
            FloorInOut[i] = 2'b11;
        end
        WhichFloor = 5'b00000; // Start at floor 0
    end
```

Always blocks and main logic:

در این ماژول از یک بلاک Always استفاده شده است که وظیفه تعیین حالات آسانسور با توجه به State را برعهده دارد. این بلاک با توجه به لبه پایین رونده کلاک کار میکند زیرا ماژول دیگر که وظیفه تعیین حالات را برعهده دارد با استفاده از لبه بالارونده کار میکند (در ادامه توضیح داده خواهد شد) و به همین سبب به جهت تفکیک عملکرد این دو ماژول، در اینجا از لبه پایین رونده استفاده شده است.

در این بلاک در ابتدا حالت تمامی طبقات reset میشود (به حالت عدم استفاده تبدیل میشود) تا در ادامه مقداردهی مناسب برای طبقات در حال استفاده انجام شود.

در ادامه با توجه به حالت کنونی مدار، مطابق با توضیحات بیان شده برای State و همچنین حالات مورد نظر برای FloorInOut که در ابتدا توضیح داده شد، مقداردهی های لازم انجام میشود (خروج از طبقه کنونی و ورود به طبقه جدید)

در صورتی که در حالت ایست باشیم تاخیر ۱۰۰ واحدی مطابق صورت سوال در نظر گرفته شده است.

در حالت های بالا و پایین رفتن آسانسور نیز پس از مقداردهی مناسب، ۲۰ ثانیه تاخیر در نظر گرفته شده است تا تغییر حالات آسانسور مطابق با واقعیت انجام پذیرد و در واقع حرکت آسانسور میان طبقات در نظر گرفته شده باشد.

```
// Always block triggered on the positive edge of Clk or Reset
always @(negedge Clk) begin
   // Reset all floor states to notStarting Yet
    for (i = 0; i < 5; i = i + 1) begin
        FloorInOut[i] = 2'b11;
    end
    // State machine to control elevator behavior
        case (State)
            Stop: begin
                FloorInOut[WhichFloor] = 2'b00; // Mark current floor as stopped
                #100; // Delay for debounce or timing purposes
            end

            Up: begin
                if (WhichFloor < 5) begin // Check if not already at the top floor
                    FloorInOut[WhichFloor] = 2'b10; // Mark floor as visited
                    WhichFloor = WhichFloor + 1;    // Move up one floor
                    FloorInOut[WhichFloor] = 2'b01; // Mark new floor as current
                end
                #20; // Delay for debounce or timing purposes
```

```
                end

                Down: begin
                    if (WhichFloor > 0) begin // Check if not already at the bottom floor
                        FloorInOut[WhichFloor] = 2'b10; // Mark floor as visited
                        WhichFloor = WhichFloor - 1;    // Move down one floor
                        FloorInOut[WhichFloor] = 2'b01; // Mark new floor as current
                    end
                    #20; // Delay for debounce or timing purposes
                end
                default:
                    ; // No default action
            endcase
        end
endmodule
```

۲- ماژول کنترل کننده آسانسور (Handler)

ورودی و خروجی:

این ماژول به مانند ماژول دیگر یک ورودی کلاک دارد که کنترل کننده کلاک آسانسور است. یک ورودی WhichFloor نیز مطابق ماژول پیشین وجود دارد که طبقه کنونی آسانسور را ذخیره میکند. دو ثبات ۵ بیتی نیز برای فشرده شدن دکمه طبقات درون آسانسور و بیرون از آن تعریف شده اند. در ماژول پیشین ثبات State به گونه ای توضیح داده شد که با استفاده از جهت و حرکت آسانسور مقداردهی میشود؛ در اینجا دو خروجی، یکی برای جهت و دیگری برای حرکت یا عدم حرکت آسانسور در نظر گرفته شده است که با توجه به این دو خروجی، مقداردهی ثبات State در ماژول دیگر انجام میشود. در انتها نیز خروجی PushedButtons در نظر گرفته شده است که از لحاظ منطقی نیازی به قرار دادن آن به عنوان خروجی برای ماژول نیست اما به جهت نمایش بهتر در قسمت تست و درک بهتر، این ثبات در خروجی قرار گرفته است که در واقع طبقات مورد انتظار که باید آسانسور به آنها برسد را در ۵ بیت نشان میدهد. (به ازای انتظار برای طبقه x، بیت xام در این ثبات، مقدار یک را میگیرد و در زمان رسیدن آسانسور به هر طبقه، بیت متناظر با آن، مقدار ۰ را میگیرد.

```
module Handler (
    input clk,                    // Clock signal
    input [4:0] WhichFloor,     // Current floor of the elevator
    input [4:0] InToggle,         // Input signal for internal floor buttons
    input [4:0] OutToggle,        // Input signal for external floor buttons
    output reg Dir,               // Direction of the elevator (Up, Down, or Stop)
    output reg Process,            // Elevator process status (Active or Stopped)
    output reg [4:0] PushedButtons
);
```

Initial and local params:

در این ماژول به مانند ماژول پیشین، سه پارامتر محلی برای حالات ایست، بالا و پایین در نظر گرفته شده است. همچنین در این ماژول یک صف طراحی شده است که شماره طبقه مورد نظر در آن قرار میگیرد و رسیدن آسانسور به طبقات با توجه به مقادیر موجود در این صف انجام میشود.

همچنین یک متغیر برای اینکه آسانسور مقصدی برای حرکت دارد یا خیر تعریف شده است.

دو متغیر انتهایی نیز شمارنده حلقه ها و دیگری، مقصد آسانسور در هر لحظه را نگه داری میکنند.

در بلاک initial نیز در ابتدا آسانسور در حالت ایست قرار میگیرد و آسانسور جهت حرکت و دریافت دستور آماده میشود.

همچنین ثبات متناظر با دکمه های فشرده شده نیز تماما صفر مقداردهی میشود تا از این لحظه آماده دریافت دستور برای حرکت به طبقات باشیم.

```
    bit [2:0] PushedButtons_queue [$:4]; // Queue to store floor requests
    bit CurrentDestination;       // Flag to check if there's a current destination
    integer i, Destination;       // Loop variable and destination floor

    // Local parameters for elevator states
    localparam Stop = 2'b00,
               Up = 2'b11,
               Down = 2'b01;

    // Initial block to set the default states
    initial begin
        {Dir, Process} = Stop;    // Set the elevator to stop initially
        PushedButtons = 5'b00000;   // No buttons are pushed initially
        CurrentDestination <= 1'b1;
    end
```

Always blocks and main logic:

سه بلاک Always در این ماژول مورد استفاده قرار گرفته اند که هر یک را در ادامه توضیح خواهیم داد.

نکته شایان توجه در این قسمت آن است که بلاک ها در این ماژول با توجه به لبه بالارونده کلاک عملکرد خود را انجام خواهند داد که دلیل تفکیک آن نسبت به ماژول پیشین، در قسمت قبل توضیح داده شد.

بلاک ابتدایی مربوط به مقدار دهی PushedButtons است که در آن مقاصد آسانسور با or کردن این ثبات با خودش و ثبات های متناظر با دکمه های درون و بیرون آسانسور بدست می آید.

در بلاک دیگر نیز مقصد آسانسور و جهت و حرکت آن با توجه به حالات متفاوت انجام میگردد که با توجه به تعدد شروط از بیان و شرح آنها خودداری میشود و ضمنا در تمامی خطوط مورد نیاز، کامنت هایی به جهت توضیح هر خط کد در نظر گرفته شده است که شرح هر قسمت را میتوان با توجه به آنها بررسی نمود. در این بلاک در صورتی که آسانسور در حالت رفتن به مقصد جدید باشد، از ابتدای صف، مقصد آسانسور خارج میگردد.

بلاک نهایی نیز وظیفه پر کردن صف را برعهده دارد. در این بلاک نیز با توجه به شروط تعریف شده که به دلیل بیان شده در قسمت قبل، از بیان شرح آنها خودداری میکنیم؛ طبقاتی که فشرده شده اند، در صف با توجه به اولویت منطقی، قرار میگیرند.

```
    // Always block triggered on the negative edge of the clock
    // Updates the state of pushed buttons
    always @(posedge clk) begin
        PushedButtons <= PushedButtons | InToggle | OutToggle;
    end

    // Always block to handle the elevator movement and direction based on the request
queue
    always @(posedge clk) begin
        if ({Dir, Process} == Up) begin
            if (WhichFloor < 5) begin
                if (PushedButtons[WhichFloor] == 1'b1) begin
                    PushedButtons[WhichFloor] <= 1'b0; // Clear the button press for
current floor
                    {Dir, Process} <= Stop; // Stop the elevator
                end
                if (Destination == WhichFloor) begin
                    {Dir, Process} = Stop; // Stop the elevator
                    CurrentDestination = 1; // Mark current destination as reached
                end
            end else begin
                {Dir, Process} <= Stop; // Stop the elevator if at top floor
```

```
            end
        end else if ({Dir, Process} == Down) begin
            if (WhichFloor > 0) begin
                if (PushedButtons[WhichFloor] == 1'b1) begin
                    PushedButtons[WhichFloor] <= 1'b0; // Clear the button press for
current floor
                    {Dir, Process} <= Stop; // Stop the elevator
                end
                if (Destination == WhichFloor) begin
                    {Dir, Process} = Stop; // Stop the elevator
                    CurrentDestination = 1; // Mark current destination as reached
                end
            end else begin
                {Dir, Process} <= Stop; // Stop the elevator if at bottom floor
            end
        end else if (Dir ==0 && Process ==0) begin
            if (PushedButtons != {5{1'b0}}) begin
                if (PushedButtons_queue.size() > 0 && CurrentDestination) begin
                    Destination = PushedButtons_queue.pop_front(); // Get next
destination from queue
                    CurrentDestination = 0;
                end
                if (Destination > WhichFloor)
                    {Dir, Process} <= Up; // Set direction to Up
                else if (Destination < WhichFloor)
                    {Dir, Process} <= Down; // Set direction to Down
            end
            if (Destination == WhichFloor) begin
                PushedButtons[WhichFloor] <= 1'b0; // Clear the button press for current
floor
                CurrentDestination = 1; // Mark current destination as reached
            end
        end
    end

    // Always block to update the request queue based on button presses
    always @(posedge clk) begin
        for (i = 0; i < 5; i = i + 1) begin
            if (InToggle[i] | OutToggle[i] == 1'b1) begin
                 if ({Dir, Process} == Down) begin
                    if (i > WhichFloor || i < Destination)
                        PushedButtons_queue.push_back(i);
                end else if ({Dir, Process} == Stop) begin
                    if (Destination > WhichFloor) begin
                        if (i < WhichFloor)
                            PushedButtons_queue.push_back(i);
                        else if (i > Destination)
                            PushedButtons_queue.push_back(i);
                    end else if (Destination < WhichFloor) begin
```

```
                        if (i > WhichFloor)
                            PushedButtons_queue.push_back(i);
                        else if (i < Destination)
                            PushedButtons_queue.push_back(i);
                    end else
                        PushedButtons_queue.push_back(i);
                end else if ({Dir, Process} == Up) begin
                    if (WhichFloor > i || i > Destination)
                        PushedButtons_queue.push_back(i);
                end
            end
        end
    end
endmodule
```