Elevator Controller Design

Final Project

YiWen Meng & CheeKang Tan

4/27/2013
## Content

<table>
<thead>
<tr>
<th>Section</th>
<th>Page number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Design Strategy</td>
<td>2</td>
</tr>
<tr>
<td>Verilog Code:</td>
<td></td>
</tr>
<tr>
<td>1. Input and Output pins</td>
<td>3</td>
</tr>
<tr>
<td>2. Register Parameters and Clock Generator</td>
<td>4</td>
</tr>
<tr>
<td>3. Initialization, Clock Generator Block &amp; Request Floor Loop</td>
<td>6</td>
</tr>
<tr>
<td>4. Set three cases of elevator</td>
<td>9</td>
</tr>
<tr>
<td>5. Test Bench</td>
<td>11</td>
</tr>
<tr>
<td>6. Simulation Result</td>
<td>12</td>
</tr>
<tr>
<td>Schematic and Layout:</td>
<td></td>
</tr>
<tr>
<td>1. Verilog Synthesis with RTL Compiler</td>
<td>14</td>
</tr>
<tr>
<td>2. Power Planning</td>
<td>14</td>
</tr>
<tr>
<td>3. Placement</td>
<td>15</td>
</tr>
<tr>
<td>4. Routing</td>
<td>15</td>
</tr>
<tr>
<td>5. Filler cells</td>
<td>16</td>
</tr>
<tr>
<td>6. Layout</td>
<td>16</td>
</tr>
<tr>
<td>7. DRC Result</td>
<td>17</td>
</tr>
<tr>
<td>8. Elevator Controller design Symbol</td>
<td>17</td>
</tr>
<tr>
<td>Conclusion</td>
<td>17</td>
</tr>
</tbody>
</table>
**Introduction**

This project is designed for an eight floor elevator controller. It is an integrated circuit that can be used as part of an elevator controller. The elevator decides moving direction by comparing request floor with current floor. In a condition that the weight has to be less than 4500 lb and door has to be closed in three minute. If the weight is larger than it, the elevator will alert automatically. The Door Alert signal is normally low but goes high whenever the door has been open for more than three minute. There is a sensor at each floor to sense whether the elevator has passed the current floor. This sensor provides the signal that encodes the floor that has been passed.

The core parts of the design are shift register, three cases of elevator and the while loop when receive Request Floor.

**Design Strategy**

In the coding part, we used several strategies to make the program works.

First, we defined the input and output current floor as In_Current_Floor and Our_Current_Floor to avoid same variable name as output and input.

Second, we add two more input pins - Over_time and Over_Weight in the code. These signals will be output from the mechanical machine to the controller. When the controller receives signal from weight alert or door alert, the complete will become one so that the elevator will stay unmoved at the Out_Current_Floor.

Third, define the Out_Current_Floor, Direction, Complete, Door_Alert and Weight_Alert as reg then assign them equal to the output. Therefore, those variables will run as a register and output.

Next, when the Reset is off the variable Complete, Door Alert and Weight Alert will be initialized to be zero. Similarly, when the Request_Floor is on, the variable In_Current_Floor is set to be equal to Out_Current_Floor only once.

Then, In_Current_Floor stay the same, Out_Current_Floor keep changing (updating) and compare with request floor, until Out_Current_Floor is at the same level as Request_Floor.

Lastly, define three cases of if statement for the elevator. There are cases for normal running cases – (comparing between Request_Floor and Out_Current_Floor to decide the moving direction), door open for more than three minutes - (turn on the Door_Alert) and overweight cases for elevator - (turn on the Weight_Alert).
**Verilog Code - Input and Output pins**

```verilog
module elevator_controller (request_floor, in_current_floor, clk, reset, complete, direction,
                           over_time, over_weight, weigh_alert, door_alert, out_current_floor);

//input pins
//-----------------------------------------------
input [7:0] request_floor; // the 8 bit input request_floor;
input [7:0] in_current_floor; // the 8 bit input request_floor;
input clk; // we generate a low frequency clock
input reset; // the 1 bit input reset
input over_time; //the 1 bit input which indicates the door keep open for 3 minutes
input over_weight; // the 1 bit input which indicates the weight in the elevator is larger than 4500 lbs
//the output pins
//-----------------------------------------------
output direction; // the 1 bit output which indicates the direction of the elevator
output complete; // the 1 bit output which indicates whether the elevator is running or stopped
output door_alert; // the 1 bit output which indicates the door keep open for 3 minutes
output weigh_alert; // the 1 bit output which indicates the weight in the elevator is larger than 4500 lbs
output [7:0] out_current_floor; // the 8 bit output which shows the current floor
```

**Input [7:0] Request_Floor**
Define Request_Floor as 8-bit input variable. The 00000001 to 10000000 represent ground floor to eighth floor. (For example, second floor is 00000010, third floor is 00000100)

**Input [7:0] In_Current_Floor**
Define In_Current_Floor as 8-bit input variable. It represents the initial floor and the next complete floor (or the Out_Current_Floor that has reached the request floor).

**Input clk**
Define clk as input variable. To generate low frequency clock.

**Input reset**
Define reset as input variable. To generate reset.

**Input Over_time**
Define Over_time as input variable. In the Verilog code, “1” represents waiting time is longer than three minutes. Input Over_time will be the signal receives from timer (machinery part).

**Input Over_Weight**
Define Over_Weight as input variable. In the Verilog code, “1” represents elevator is overload. Input Over_Weight will be a signal receives from the weight detector (machinery part).

**Output Direction**
Define Direction as output variable. In the Verilog code, “1” represents moving up and “0” represents moving down.

**Output Complete**
Define Complete as output variable. In the Verilog code, “1” represents the elevator reaches the request floor. It can also represent pause (stay unmoved at the Out_Current_Floor), when the Over_Time or Weight_Alert is on.
**Output Door_Alert**
Define Door_Alert as output variable. In the Verilog code, “1” represents waiting time is longer than three minutes. Output Door_Alert will be on when Over_time is on.

**Output Weight_Alert**
Define Weight_Alert as output variable. In the Verilog code, “1” represents elevator is overload. Input Weight_Alert will on when Over_Weight is on.

**Output [7:0] Out_Current_Floor**
Define Out_Current_Floor as 8-bit output variable. It is a variable that will be used in the shift register.

---

**Verilog Code – Register Parameters and Clock Generator**

```verilog
//register parameters
//-------------------------------
reg r_direction;// 1 bit register connected to the output direction
reg r_complete;// 1 bit register connected to the output complete
reg r_door_alert;// 1 bit register connected to the output door_alert
reg r_weigh_alert;// 1 bit register connected to the output weigh_alert
reg [7:0] r_out_current_floor;// 8 bit register connected to the output out_current_floor;
//-------------------------------
//clock generator register
//-------------------------------
reg [12:0] clk_count;
reg clk_200;
reg clk_trigger;
//-------------------------------
//match pins and registers
//-------------------------------
assign direction=r_direction;
assign complete=r_complete;
assign door_alert=r_door_alert;
assign weigh_alert=r_weigh_alert;
assign out_current_floor=r_out_current_floor;
//-------------------------------
```

**Reg R_Direction**
Define R_Direction as register parameter. In the Verilog code, “1” represents moving up and “0” represents moving down.

**Reg R_Complete**
Define R_Complete as register parameter. In the Verilog code, “1” represents the elevator reaches the request floor. It can also represent pause.

**Reg R_Door_Alert**
Define R_Door_Alert as register parameter. In the Verilog code, “1” represents moving up and “0” represents moving down.
Define R_Door_Alert as register parameter. In the Verilog code, “1” represents elevator is overload.

**Reg [7:0] R_Out_Current_Floor**
Define R_Out_Current_Floor as register parameter. In the Verilog code, “1” represents moving up and “0” represents moving down.

**Reg [12:0] CLK_Count**
Define CLK_Count as 12-bit register parameter. It is a low frequency clock counter.

**Reg CLK_200**
Define CLK_200 as register parameter. It generates a 200 Hz clock.

**Reg CLK_Trigger**
Define CLK_Trigger as register parameter. It is the trigger of generator clock.

**Assign Direction = R_Direction**
To store the data of R_Direction into Direction

**Assign Direction Complete = R_Complete**
To store the data of Direction_Complete into R_Direction

**Assign Door_Alert = R_Door_Alert**
To store the data of Door_Alert into R_Door_Alert

**Assign Weight_Alert = R_Weight_Alert**
To store the data of Weight_Alert into R_Weight_Alert
Verilog Code – Initialization, Clock Generator Block and Request Floor Loop

```verilog
// initialization
// -----------------------------------------
always @ (negedge reset) begin
    clk_200=1'b0;
    clk_count=0;
    clk_trigger=1'b0;
// reset the clock registers
    r_complete=1'b0; // set the default value to 0
    r_door_alert=1'b0; // set the default value to 0
    r_weight_alert=1'b0; // set the default value to 0
end
// -----------------------------------------
```

This part will do the Initialization of the r_complete, r_door_alert and r_weight_alert.

**Always @ (negedge reset)**

It will run only when the reset is zero. This loop will reset \( CLK_{200} \), \( CLK\_Count \) and \( CLK\_Trigger \) to be zero. Also, the \( R\_Complete \), \( R\_Door\_Alert \) and \( R\_Weight\_Alert \) will be set to be zero. If the reset is turn on to be one, the loop will not work.

```verilog
// clock generator block
// ------------------------
always @ (posedge clk) begin
    if(clk_trigger) begin
        clk_count=clk_count+1;
    end
    if(clk_count==5000)begin
        clk_200=¬clk_200;
        clk_count=0;
    end
end
// -----------------------------------------
```

This is a clock generator that will run a loop when the \( CLK \) is on. The If Statement will determine whether the \( CLK\_Trigger \) is on or the \( CLK\_Count \) is equal to five thousand. If they are in the condition, the \( CLK\_Count \) will increment every time the loop runs or the clock generator 200Hz will on. Therefore, the output control will run instantaneously.
This is the core part of the design.

It stores the data of *In_Current_Floor* to *R_Out_Current_Floor* only once.

**Always @ (request_floor)**

The loop will run only when it receives request_floor.

It turn the *CLK_Trigger* on and trigger the 200Hz generator clock.

Also stores the data of *In_Current_Floor* to *R_Out_Current_Floor*.

Therefore, the *R_Out_Current_Floor* will be keep updating (or changing) whenever it reach the next floor, while the *In_Current_Floor* stay at the initial floor data.
Door Alert

Request Floor

Current Floor

Over Weight

Current Floor

Direction
Verilog Code – Set three cases of elevator

```verilog
//the normal running cases of the elevator
always @ (posedge clk) begin
    if(!reset && !over_time && !over_weight) begin
        // case 1: the normal running case of the elevator
        if (request_floor > r_out_current_floor) begin
            r_direction=1'b1;
            r_out_current_floor <= r_out_current_floor << 1;
        end
        else if (request_floor < r_out_current_floor) begin
            r_direction=1'b0;
            r_out_current_floor = r_out_current_floor >> 1;
        end
        else if (request_floor == r_out_current_floor) begin
            r_complete=1;
            r_direction=0;
        end
    end
end
```

This part will do the three cases of elevator. There are cases for normal running, door overtime and over load.

**Always @ (posedge clk)**

This loop will run only when the **CLK** is on.

*If (!reset && !over_time && !over_weight)*

This loop will run only in the condition that **reset**, **over_time** and **over_weight** are off.

**Case 1: the normal running case of elevator**

If the **Request_Floor** is Greater than **R_Out_Current_Floor** the elevator will move up.

If the **Request_Floor** is Smaller than **R_Out_Current_Floor** the elevator will move down.

If the **Request_Floor** is Equal to **R_Out_Current_Floor** (it reach the request floor) the **R_Complete** is on and elevator stop moving

* The “<<1” and “>>1” are shift register. It shifts the data to the pointed direction by one.

For example, “<<1” will move “00000001” to “00000010”
Case 2: the door has been open for more than three minutes

If the Reset is off and Over_Time is on,

The R_Door_Alert and R_Complete are set to be ON

The R_Weight_Alert and R_Direction are set to be OFF

The R_Out_Current_Floor store its data to R_Out_Current_Floor

Therefore the door alert ring and the elevator will stop moving (or pause) when it is over time.

Case 3: the total weight in the elevator is more than 4500lb

If the Reset is off and Over_Weight is on,

The R_Weight_Alert and R_Complete are set to be ON

The R_Door_Alert and R_Direction are set to be OFF

The R_Out_Current_Floor store its data to R_Out_Current_Floor

Therefore the weight alert ring and the elevator will stop moving (or pause) when it is over load.
In the Verilog Test Bench Code, after we defined reg and wire variables, we chose only two cases to test our code, because there are more than fifty possible cases.

The two cases we chose are:

Request floor = 00000001; In_Current_floor = 10000000
The elevator will move down from eighth floor to ground floor.

Request floor = 10000000; In_Current_floor = 00000001
The elevator will move up from ground floor to eighth floor.
Simulation Result

When Request floor = 00000001; In_Current_floor = 10000000

The elevator moves up from eighth floor to ground floor.
When Request floor = 10000001; In_Current_floor = 00000001

<table>
<thead>
<tr>
<th>Messages</th>
<th>10000000</th>
<th>10000001</th>
</tr>
</thead>
<tbody>
<tr>
<td>elevator_controller_to_request_floor</td>
<td>00000001</td>
<td>00000001</td>
</tr>
<tr>
<td>elevator_controller_to_current_floor</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>elevator_controller_to_reset</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>elevator_controller_to_relative_time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>elevator_controller_to_over_weight</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>elevator_controller_to_direction</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>elevator_controller_to_complete</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The elevator moves up from ground floor to eighth floor.
Schematic and Layout

Verilog Synthesis with RTL Compiler

Power Planning
Placement

Routing
Filler Cells

Layout
Elevator Controller Design

DRC Result

For this project, we learned the basic idea of how does the normal elevators run in many cases, even though it is simplified, we still spent lots of time to design and to figure out many problems when combining all the cases, of course the most challenging and time-consuming part is debugging. However, after accomplishing it, we learned many things beyond this project, so it is a very helpful assignment.