VLSI Project: Elevator Control System

Part 2

Name: Prajwal Paul & Khoa Diep

ktd170030

psp170030

CE 6325

Instructor: Prof. Sechen

Date: 9/24/2023

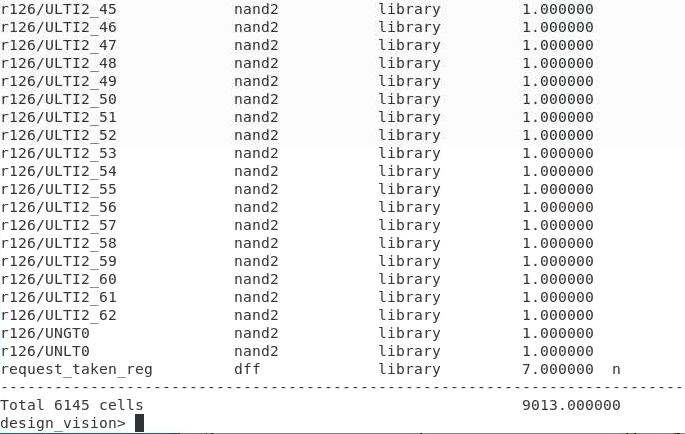
**Background & Description**

Comparing our previous Verilog code to this new upscaled version, more variables and complexity has been added. Previously, our system was a single elevator for 8 floors and an overweight sensor that would allow the elevator to move only when the load was under the set weight. There was a destination floor, current floor and the direction of the elevator and the arrived variable determined if the doors were open, which would occur when the elevator arrived at its destination or was over the weight limit. On the positive edge of the clock, we ran through various switch cases, set up in descending priority, to determine what the appropriate action was, which were a reset function, if the load was overweight, when the targeted floor was above or below the current floor and finally, when the elevator was at or arrives at the targeted floor. In the testbench, we ran through the different cases and manipulated the different inputs to ensure proper function. As the resulting cells from this implementation were not over 3000, we decided that we needed to scale up this design and its complexity. First, we decided to increase the floors to 64. Also, instead of using the sum of the bits for the floor output, we are now shifting the 1 bit among zeros to signify which floor it's on. The main purpose of this was to increase complexity. Next, we implemented a two elevator system that answers the request at a certain floor. During this, we had to make sure that both elevators don’t move to that floor and also make sure that the elevator answering the request is not busy from answering another requested floor or destination floor request.

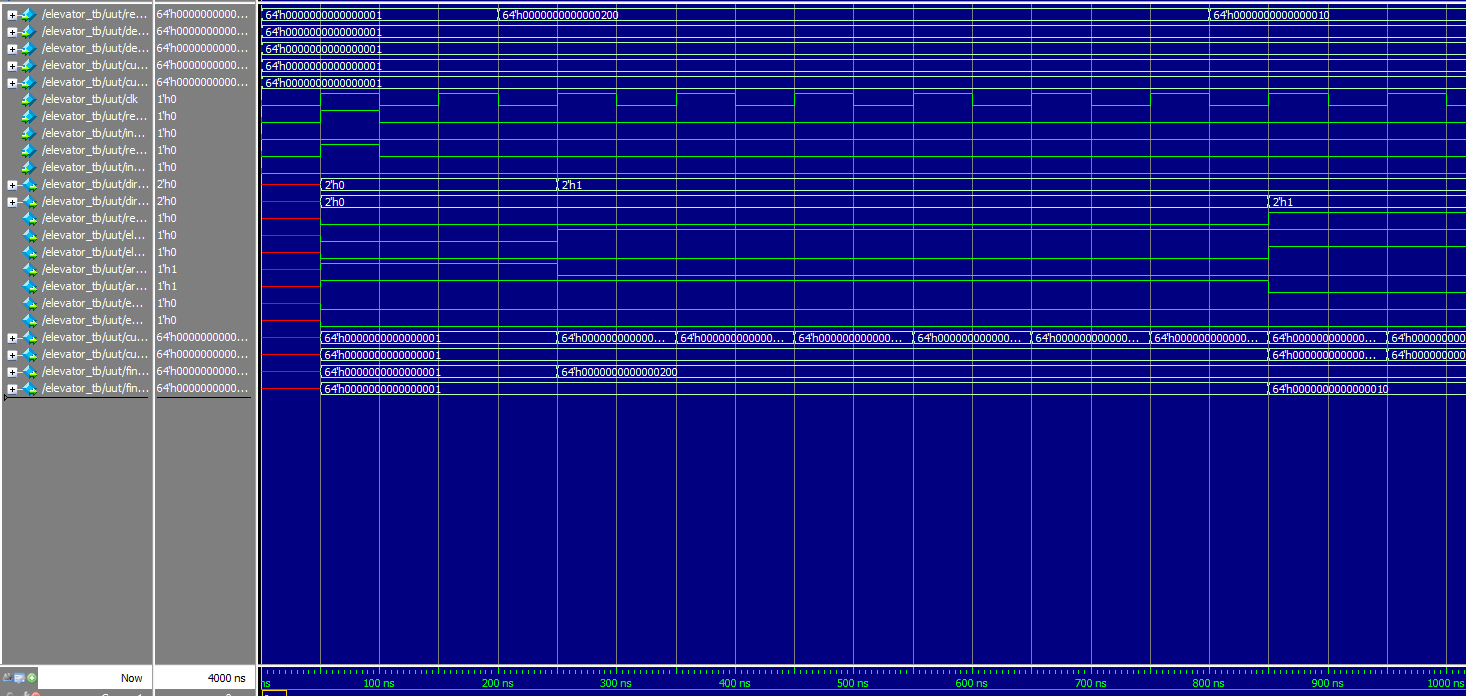
Looking at the behavioral code, all the needed variables for this upscale were defined and initialized. On the posedge of the clock, we once again run through the various cases, set up in descending priority in separate if-else loops, to determine what the appropriate action is. The first situation is the assigning of requested floors to the two elevators. First, we give priority to elevator 1, which takes the request and assigns it as its final floor and flags as request is taken. Next, if elevator 1 is busy, the request is sent to elevator 2 and the same actions as completed for this, which are the request taken flag and the floor is assigned as the final floor for this elevator. The final implicitly is when both elevators are busy, then this request is held on till the first elevator gets free. The next situation is the reset and emergency function. We decided to rename overweight to emergency because they both functioned the same way in the code and testbench and figured that this would better capture the purpose. The reset function case helps reset the elevator system. This command is used as a “new command is given” method and resets the arrived at destination and emergency signals, while also setting the current floor where the elevator is at to the output. The emergency function checks if the emergency input is activated then forces the doors open by the arrived variable, turns on the signal and stops elevator motion. The last situation is the checking and motion of the elevator by checking the relationship between the destination floor and current floor. The first case is when the targeted floor is above the current floor and there is no emergency signal. In this action, the direction is set to 1 meaning going up and the elevator moves up one level per clock cycle, which is done by bit shifting the 1 bit to the left each cycle. The arrived at destination signal is kept off during movement and as the elevator is moving to a targeted floor, the busy status flag is turned on. Also, as a precaution, the emergency signal is turned off as this check has already been completed. The next case is when the targeted floor is below the current elevator floor. Here, the direction is set to 2 meaning going down and the elevator goes down a floor per clock cycle by bit shifting the 1 bit to the right each cycle. The arrived at destination signal is kept off during movement here as well and as the elevator is moving to a targeted floor, the busy status flag is turned on. Also, as a precaution, the emergency signal is turned off as this check has already been completed. The final case is when the elevator has arrived at the targeted floor, where the arriving signal is turned on to open the doors and the direction is set to 0 which means no movement. Here, we free up the busy status for the elevator and show that the requested task flag that was turned off is completed. The only communication needed between the two elevators is done in the first situation. The second and third situations are analogous and are interchangeable. This is done by manipulating each elevator’s input and output in its related function. For example, elevator 2’s movements are done in its own if else loop while elevator 1’s is done another. Also, each elevator’s reset flags are controlled separately.

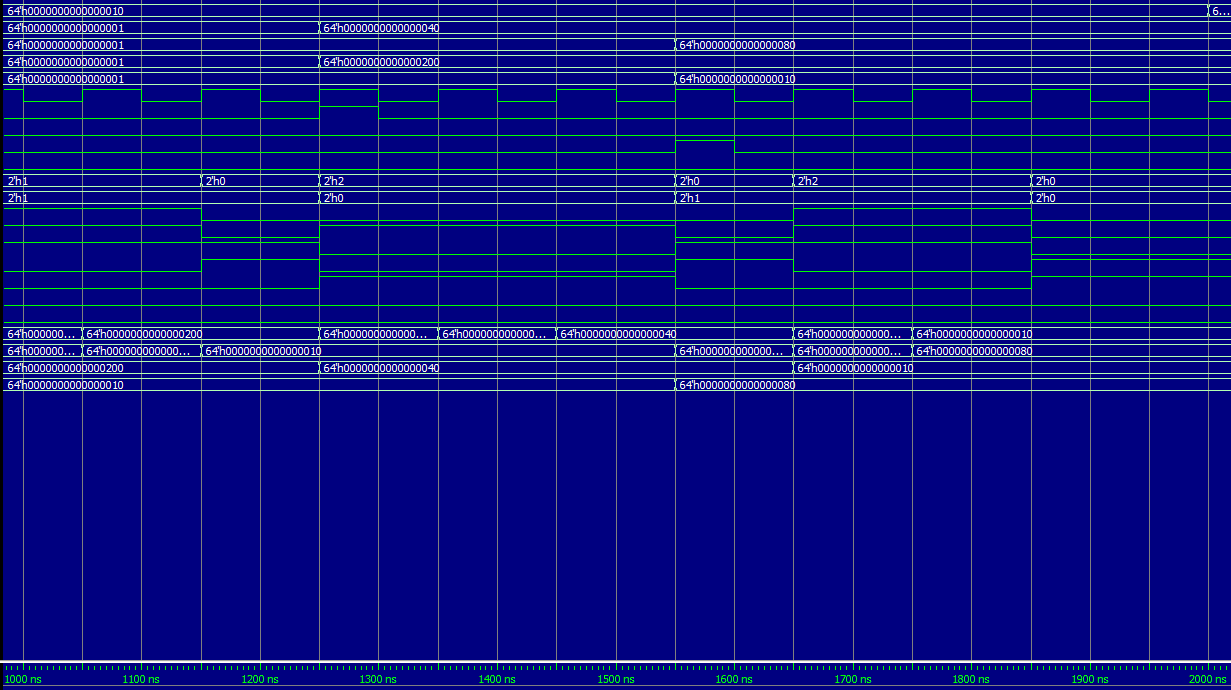
Looking at the testbench and resulting waveforms, we run through the different cases and manipulate the different inputs to ensure proper function. We start with a reset and declaration of all signals to initialize them with both elevators starting at the first floor. To start off, the first request is sent, which is the floor button request being made at floor 10. Based on our priority, elevator 1 should take this request. Then while elevator 1 is busy, a second floor button request is made at elevator 2. After these two complete, once elevator 1 arrives at 10, the destination floor button (meaning the button inside the elevator as opposed to the floor button which is located nearby outside on the wall) is set to floor 7 and the reset is pressed to show that a new command has been given. Also, this tests downward motion for the elevator and as both are same in code, we can assume the other elevator’s downward motion works as well. Next, a new destination has been given by floor 8 from 5 to elevator 2. Next, as intended, due to the fact that the request is still at floor 2 and elevator 2 is busy going up to 8, elevator 1 arrives at floor 2. We decided to implement this as it is better when elevators remain staggered instead of the same floor to allow for quicker response to the requested floor, especially in a 64 floor implementation. During this, we check the emergency stop button, by turning it on for some cycles, forcing elevator 2 to stop before turning this off and resuming motion. This elevator is still in motion to its destination floor of 8. Finally, we check the emergency stop function of elevator 1 by giving it a new request of floor 12 and turning on the signal for some clock cycles. Lastly, we let the system rest to ensure that the elevators remain staggered, with one elevator being at the request floor and the other idle from its last command.

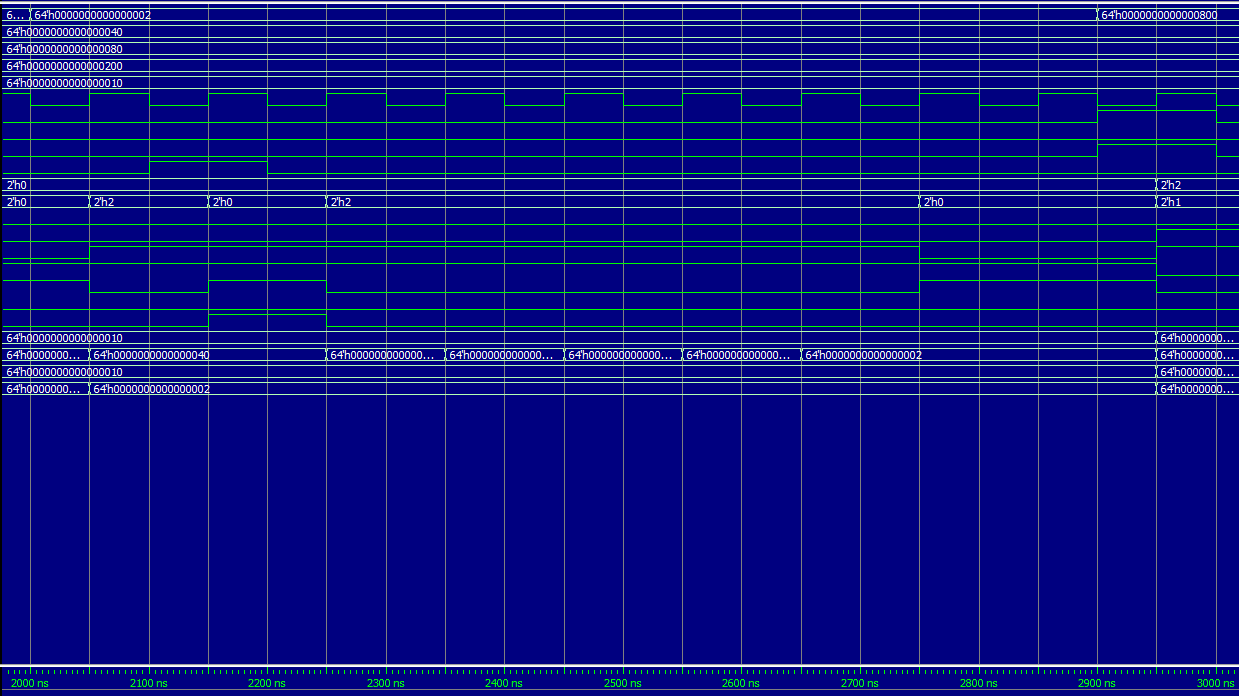
**Cell Report**

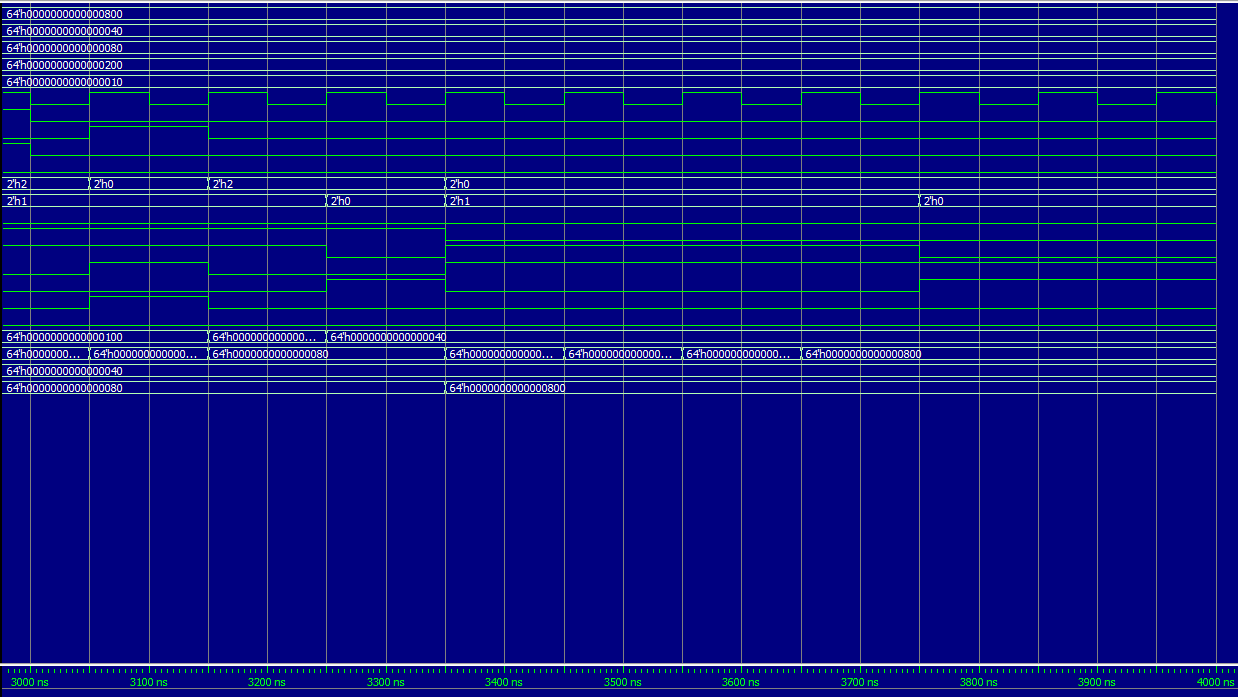
****

**Behavioral Code Waveform**

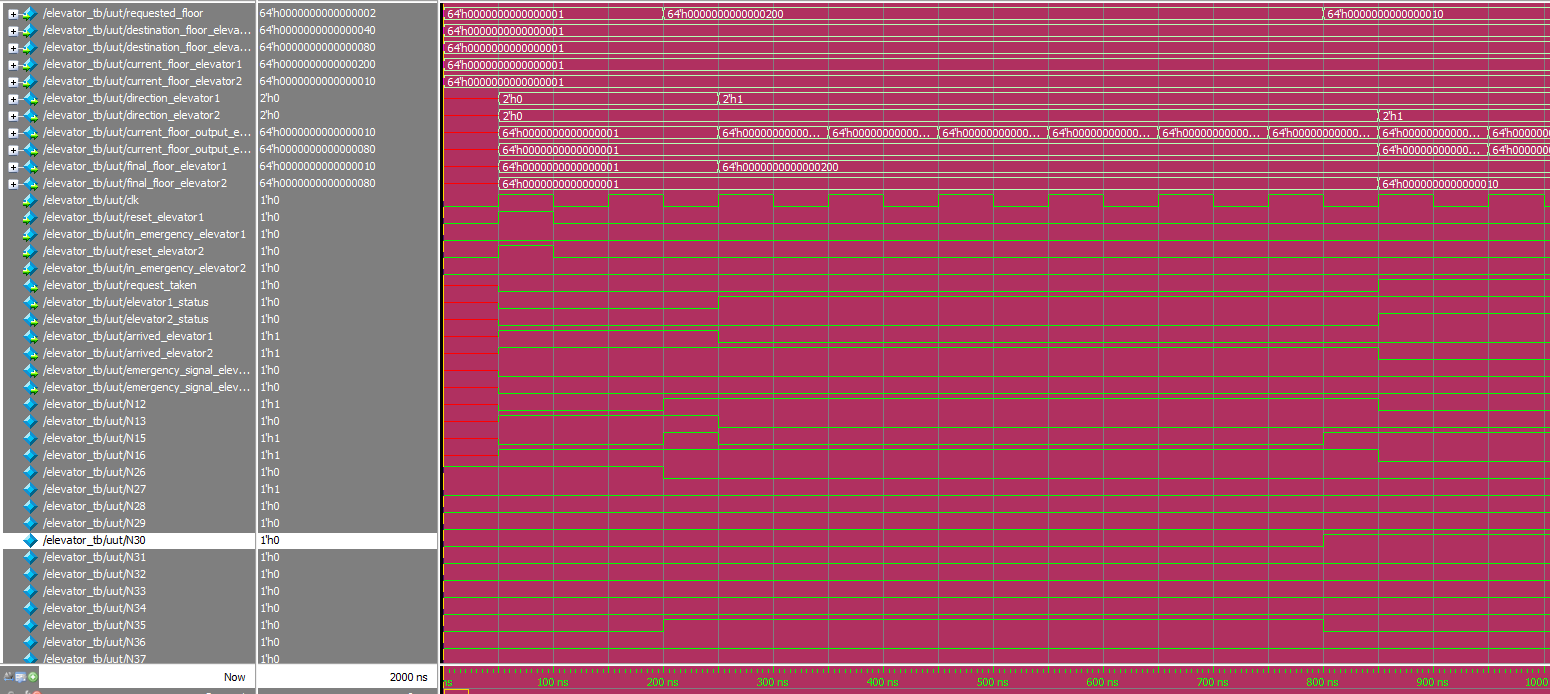
****

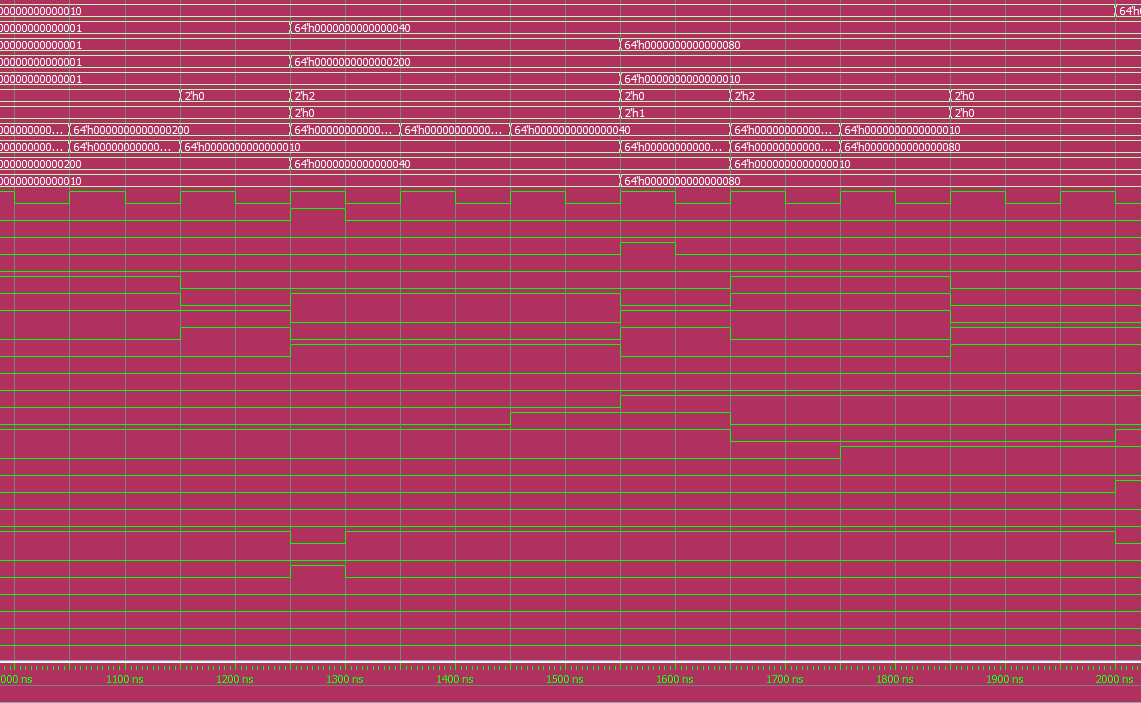
****

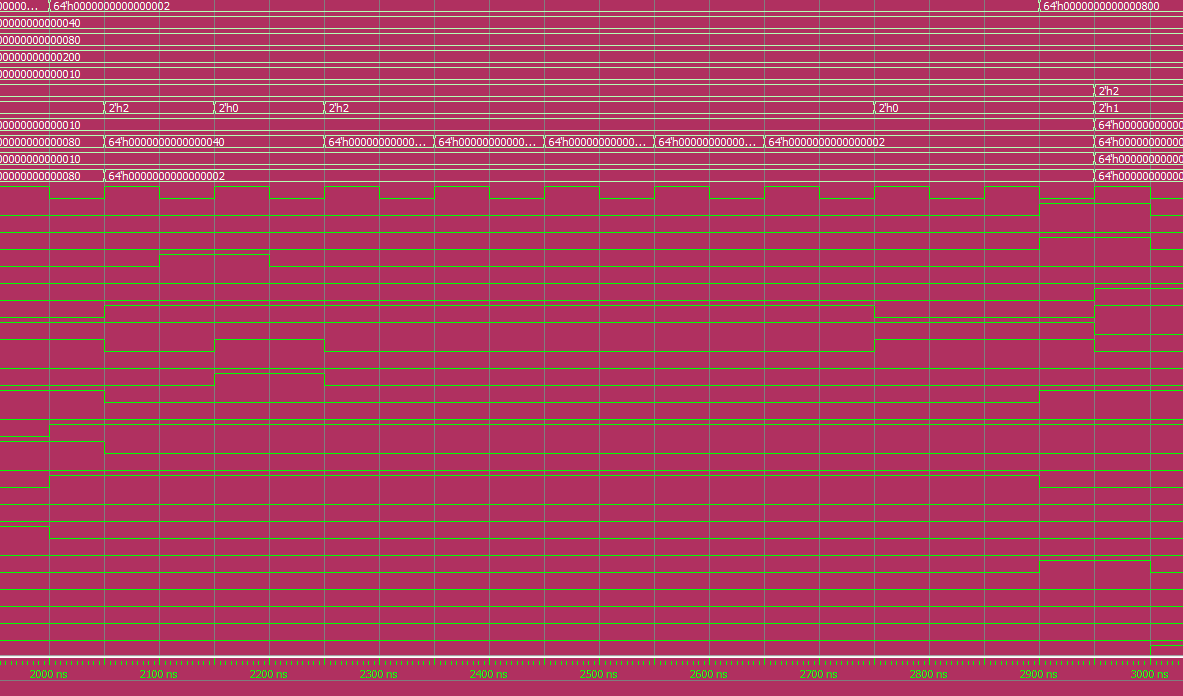
****

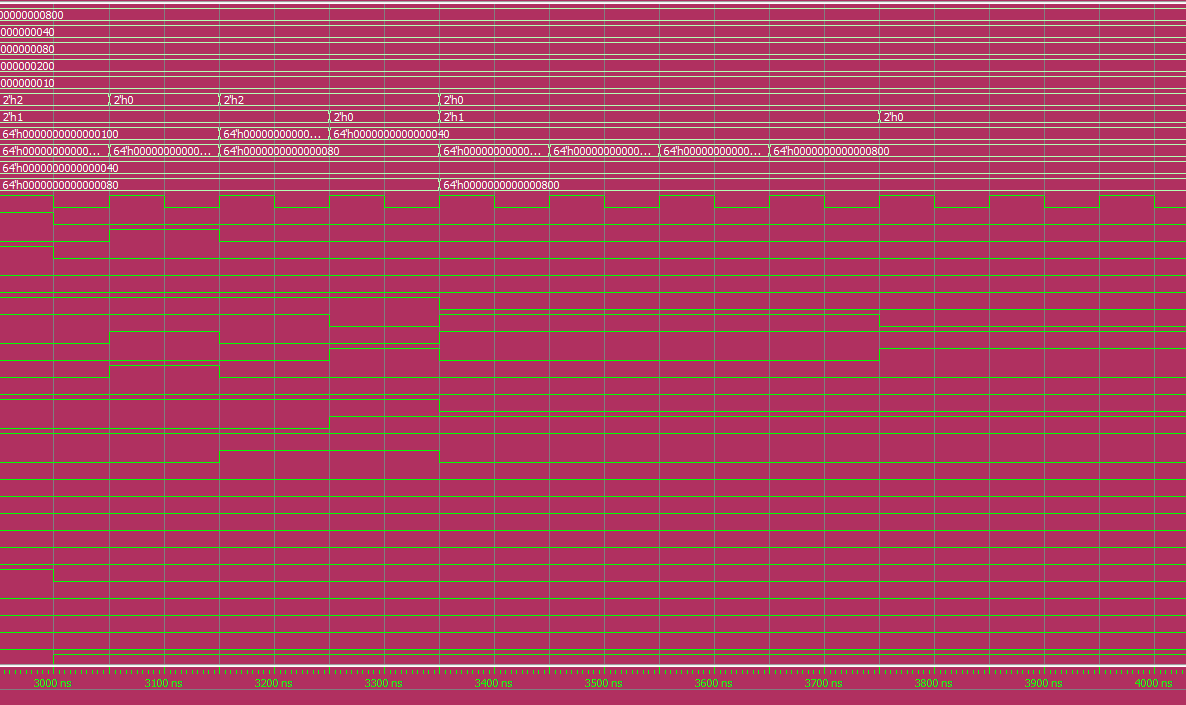
****

**Mapped Code Waveform**

****

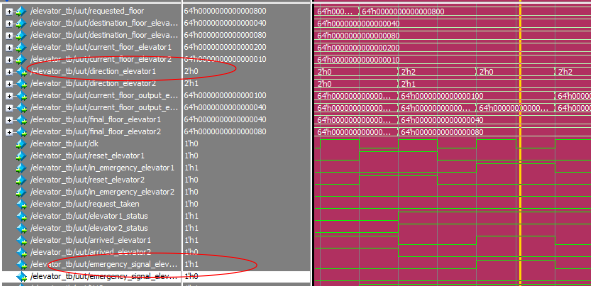
****

****

****

**Explanation of both waveforms**

Comparing both waveforms, (ignoring the gates of the synthesized Verilog waveform) they have the exact same functionality as one another. Specifically after 250 ns, requested\_floor is equal to 10 (using 1b '1000000000 instead of the decimal value) and the direction for elevator\_1 is equal to 1, indicating that it is moving up from floor 1. From 300 ns onwards, current\_floor\_output\_elevator\_1 (indicating the current floor the elevator is actually on) is being shifted by one every 100 ns, which is every time the clock hits a positive edge. This occurs until current\_floor\_output\_elevator\_1 is equal to 10. Looking at another test case, at 850 ns requested\_floor is equal to 5. Since elevator 1 is currently busy, elevator 2 will take the request. Similar to the previous case, elevator 2 direction is up and current\_floor\_output\_elevator\_2 is incrementing by shifting until current\_floor\_output\_elevator\_2 is equal to 5. Finally, both emergency signals for both elevators stop the elevator until the emergency signal is turned off is shown on both waveforms at around 2200 ns and 3100 ns.

****

Looking at before the yellow vertical line, the direction of elevator1 changes from 2 to 0 (down to staying still) since the emergency signal for elevator 1 is turned on. After, the elevator resumes going down as soon as the emergency signal is turned off (1 to 0).

**Upscaled Behavioral Verilog Code**

module elevator(

input [63:0] requested\_floor,

input [63:0] destination\_floor\_elevator1,

input [63:0] destination\_floor\_elevator2,

input [63:0] current\_floor\_elevator1,

input [63:0] current\_floor\_elevator2,

input clk,

input reset\_elevator1,

//input overweight\_elevator1,

input in\_emergency\_elevator1,

input reset\_elevator2,

//input overweight\_elevator2,

input in\_emergency\_elevator2,

output reg [1:0] direction\_elevator1,

output reg [1:0] direction\_elevator2,

output reg request\_taken,

output reg elevator1\_status,

output reg elevator2\_status,

output reg arrived\_elevator1,

output reg arrived\_elevator2,

//output reg overweight\_signal\_elevator1,

output reg emergency\_signal\_elevator1,

//output reg overweight\_signal\_elevator2,

output reg emergency\_signal\_elevator2,

output reg [63:0] current\_floor\_output\_elevator1,

output reg [63:0] current\_floor\_output\_elevator2,

output reg [63:0] final\_floor\_elevator1,

output reg [63:0] final\_floor\_elevator2

);

always @ (posedge clk) begin

// Case: Elevator 1 takes requested\_floor if elevator 1 is not busy

if(elevator1\_status == 1'b0 && request\_taken == 1'b0 && final\_floor\_elevator2 != requested\_floor && current\_floor\_output\_elevator1 == destination\_floor\_elevator1) begin

final\_floor\_elevator1 = requested\_floor;

request\_taken = 1'b1;

end

// Case: Elevator 2 takes requested\_floor if elevator 2 is not busy and elevator 1 is busy

else if (elevator2\_status == 1'b0 && request\_taken == 1'b0 && final\_floor\_elevator1 != requested\_floor && current\_floor\_output\_elevator2 == destination\_floor\_elevator2) begin

final\_floor\_elevator2 = requested\_floor;

request\_taken = 1'b1;

end

// Case: when reset\_elevator1 is pressed, outputs for elevator 1 are initialized/reset

if (reset\_elevator1) begin

direction\_elevator1 = 2'b00;

request\_taken = 1'b0;

arrived\_elevator1 = 1'b0;

elevator1\_status = 1'b0;

//overweight\_signal\_elevator1 = 1'b0;

emergency\_signal\_elevator1 = 1'b0;

current\_floor\_output\_elevator1 = current\_floor\_elevator1;

final\_floor\_elevator1 = destination\_floor\_elevator1;

end

//Case if Emergency stop

/\*else if (in\_emergency\_elevator1 == 1'b1) begin

emergency\_signal\_elevator1 = 1'b1;

arrived\_elevator1 = 1'b0;

direction\_elevator1 = 2'b00;

end\*/

//Case: Emergency button is pressed in elevator 1

if (in\_emergency\_elevator1 == 1'b1) begin

emergency\_signal\_elevator1 = 1'b1;

arrived\_elevator1 = 1'b1;

direction\_elevator1 = 2'b00;

end

// Case: Emergency button is released in elevator 1

else if (in\_emergency\_elevator1 == 1'b0) begin

emergency\_signal\_elevator1 = 1'b0;

end

// Case: Elevator 1 going up

if ((final\_floor\_elevator1 > current\_floor\_output\_elevator1) && !in\_emergency\_elevator1) begin

//overweight\_signal\_elevator1 = 1'b0;

emergency\_signal\_elevator1 = 1'b0;

direction\_elevator1 = 2'b01;

current\_floor\_output\_elevator1 = current\_floor\_output\_elevator1 << 1;

arrived\_elevator1 = 1'b0;

elevator1\_status = 1'b1;

//request\_taken = 1'b0;

end

// Case: Elevator 1 going down

else if ((final\_floor\_elevator1 < current\_floor\_output\_elevator1) && !in\_emergency\_elevator1) begin

//overweight\_signal\_elevator1 = 1'b0;

emergency\_signal\_elevator1 = 1'b0;

direction\_elevator1 = 2'b10;

current\_floor\_output\_elevator1 = current\_floor\_output\_elevator1 >> 1;

arrived\_elevator1 = 1'b0;

elevator1\_status = 1'b1;

//request\_taken = 1'b0;

end

// Case: Elevator 1 staying at current floor

else if (final\_floor\_elevator1 == current\_floor\_output\_elevator1) begin

direction\_elevator1 = 2'b00;

arrived\_elevator1 = 1'b1;

elevator1\_status = 1'b0;

request\_taken = 1'b0;

end

// Case: when reset\_elevator2 is pressed, outputs for elevator 2 are initialized/reset

if (reset\_elevator2) begin

direction\_elevator2 = 2'b00;

request\_taken = 1'b0;

arrived\_elevator2 = 1'b0;

//overweight\_signal\_elevator2 = 1'b0;

elevator2\_status = 1'b0;

emergency\_signal\_elevator2 = 1'b0;

current\_floor\_output\_elevator2 = current\_floor\_elevator2;

final\_floor\_elevator2 = destination\_floor\_elevator2;

end

//Case if Emergency stop

/\*else if (in\_emergency\_elevator2 == 1'b1) begin

emergency\_signal\_elevator2 = 1'b1;

arrived\_elevator2 = 1'b0;

direction\_elevator2 = 2'b00;

end\*/

// Case: Emergency button is pressed in elevator 2

if (in\_emergency\_elevator2 == 1'b1) begin

emergency\_signal\_elevator2 = 1'b1;

arrived\_elevator2 = 1'b1;

direction\_elevator2 = 2'b00;

end

// Case: Emergency button is released in elevator 2

else if (in\_emergency\_elevator2 == 1'b0) begin

emergency\_signal\_elevator2 = 1'b0;

end

// Case: Elevator 2 going up

if ((final\_floor\_elevator2 > current\_floor\_output\_elevator2) && !in\_emergency\_elevator2) begin

//overweight\_signal\_elevator2 = 1'b0;

emergency\_signal\_elevator2 = 1'b0;

direction\_elevator2 = 2'b01;

current\_floor\_output\_elevator2 = current\_floor\_output\_elevator2 << 1;

arrived\_elevator2 = 1'b0;

elevator2\_status = 1'b1;

//request\_taken = 1'b0;

end

// Case: Elevator 2 going down

else if ((final\_floor\_elevator2 < current\_floor\_output\_elevator2) && !in\_emergency\_elevator2) begin

//overweight\_signal\_elevator2 = 1'b0;

emergency\_signal\_elevator2 = 1'b0;

direction\_elevator2 = 2'b10;

current\_floor\_output\_elevator2 = current\_floor\_output\_elevator2 >> 1;

arrived\_elevator2 = 1'b0;

elevator2\_status = 1'b1;

//request\_taken = 1'b0;

end

//Case: Elevator 2 staying still

else if (final\_floor\_elevator2 == current\_floor\_output\_elevator2) begin

direction\_elevator2 = 2'b00;

arrived\_elevator2 = 1'b1;

elevator2\_status = 1'b0;

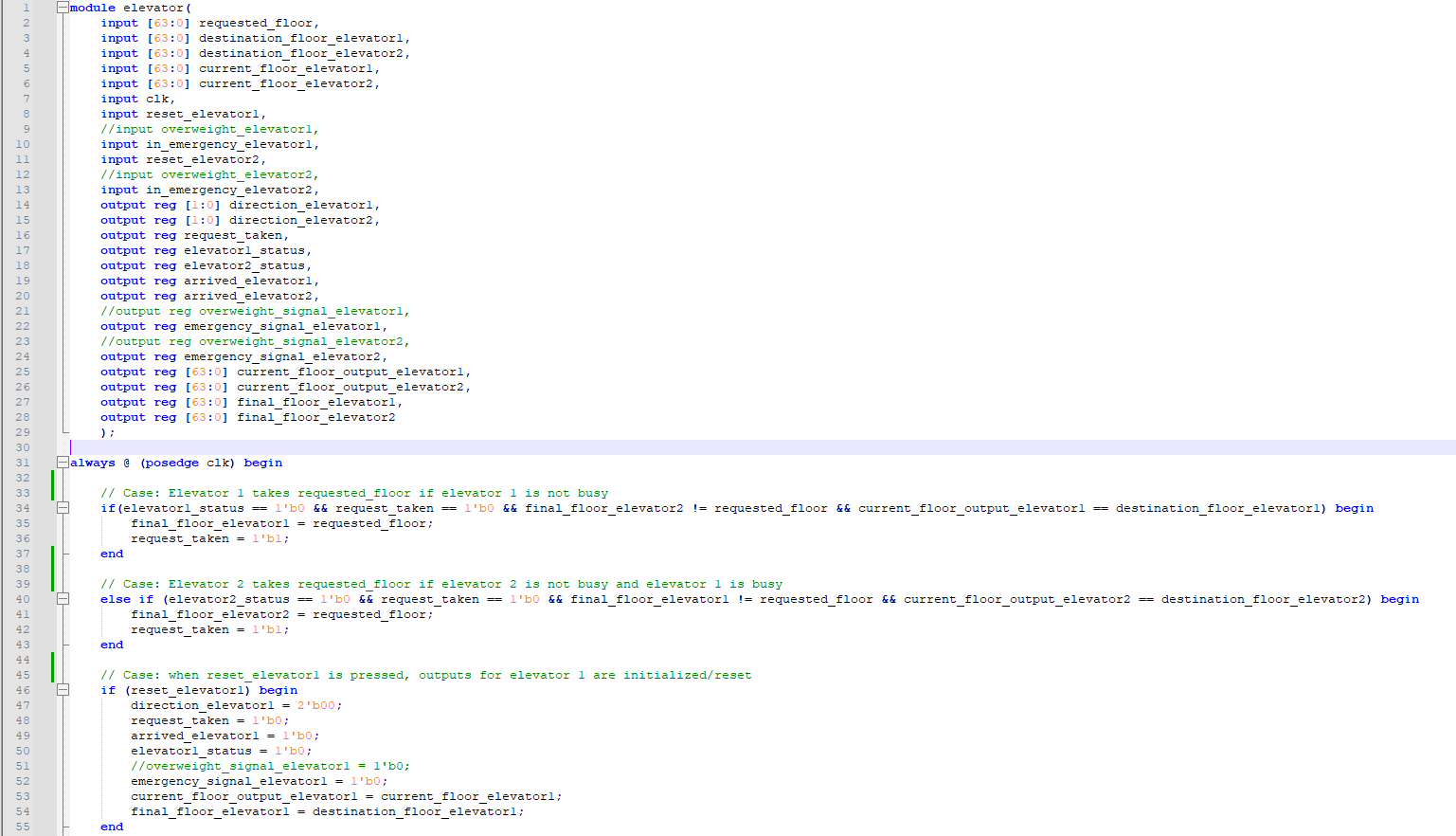
request\_taken = 1'b0;

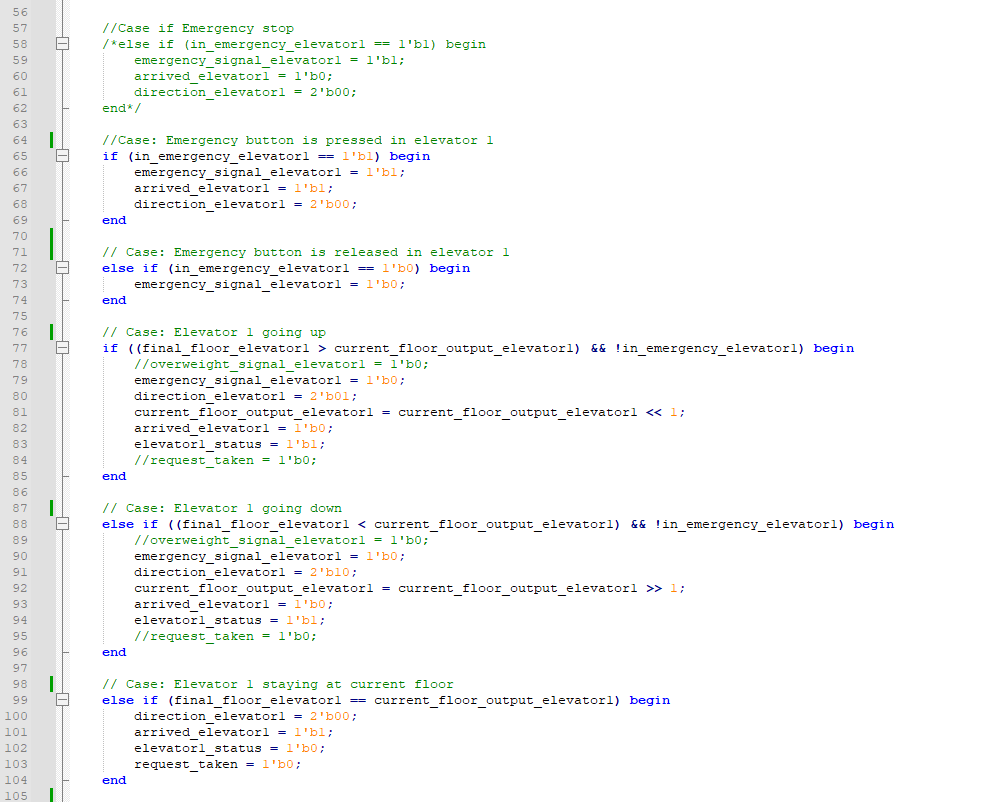
end

end

endmodule

**Upscaled Behavioral Verilog Screenshots**

****

****

****

**Testbench Verilog Code**

`timescale 1 ns/ 1 ns

module elevator\_tb;

// Define constants

reg [63:0] requested\_floor;

reg [63:0] destination\_floor\_elevator1;

reg [63:0] destination\_floor\_elevator2;

reg [63:0] current\_floor\_elevator1;

reg [63:0] current\_floor\_elevator2;

reg clk;

reg reset\_elevator1;

//reg overweight\_elevator1;

reg in\_emergency\_elevator1;

reg reset\_elevator2;

//reg overweight\_elevator2;

reg in\_emergency\_elevator2;

// Instantiate the elevator module

elevator uut (

.requested\_floor(requested\_floor),

.destination\_floor\_elevator1(destination\_floor\_elevator1),

.destination\_floor\_elevator2(destination\_floor\_elevator2),

.current\_floor\_elevator1(current\_floor\_elevator1),

.current\_floor\_elevator2(current\_floor\_elevator2),

.clk(clk),

.reset\_elevator1(reset\_elevator1),

//.overweight\_elevator1(overweight\_elevator1),

.in\_emergency\_elevator1(in\_emergency\_elevator1),

.reset\_elevator2(reset\_elevator2),

//.overweight\_elevator2(overweight\_elevator2),

.in\_emergency\_elevator2(in\_emergency\_elevator2),

.direction\_elevator1(),

.direction\_elevator2(),

.request\_taken(),

.elevator1\_status(),

.elevator2\_status(),

.arrived\_elevator1(),

.arrived\_elevator2(),

//.overweight\_signal\_elevator1(),

.emergency\_signal\_elevator1(),

//.overweight\_signal\_elevator2(),

.emergency\_signal\_elevator2(),

.current\_floor\_output\_elevator1(),

.current\_floor\_output\_elevator2(),

.final\_floor\_elevator1(),

.final\_floor\_elevator2()

);

initial

begin

$display("Running testbench");

end

always

// optional sensitivity list

// @(event1 or event2 or .... eventn)

begin

// Initial declaration of inputs

#0 clk = 1'b0;

// Both elevator start on floor 1

current\_floor\_elevator1 = 64'b0000000000000000000000000000000000000000000000000000000000000001;

current\_floor\_elevator2 = 64'b0000000000000000000000000000000000000000000000000000000000000001;

requested\_floor = 64'b000000000000000000000000000000000000000000000000000000000000001;

destination\_floor\_elevator1 = 64'b0000000000000000000000000000000000000000000000000000000000000001;

destination\_floor\_elevator2 = 64'b0000000000000000000000000000000000000000000000000000000000000001;

reset\_elevator1 = 1'b0; reset\_elevator2 = 1'b0; in\_emergency\_elevator2 = 1'b0; in\_emergency\_elevator1 = 1'b0;

// Reset is pressed and then released for both elevators

#50 reset\_elevator1 = 1'b1; reset\_elevator2 = 1'b1;

#50 reset\_elevator1 = 1'b0; reset\_elevator2 = 1'b0;

// First request from floor 10

#100 requested\_floor = 64'b0000000000000000000000000000000000000000000000000000001000000000;

#200;

// Second request from floor 5 (elevator 2 should take request since elevator 1 is busy)

#400 requested\_floor = 64'b0000000000000000000000000000000000000000000000000000000000010000;

// Destination floor for elevator 1 is 7

#450 destination\_floor\_elevator1 = 64'b0000000000000000000000000000000000000000000000000000000001000000; current\_floor\_elevator1 = 64'b0000000000000000000000000000000000000000000000000000001000000000; reset\_elevator1 = 1'b1;

#50 reset\_elevator1 = 1'b0;

// Destination floor for elevator 2 is 8

#250 destination\_floor\_elevator2 = 64'b0000000000000000000000000000000000000000000000000000000010000000; current\_floor\_elevator2 = 64'b0000000000000000000000000000000000000000000000000000000000010000; reset\_elevator2 = 1'b1;

#50 reset\_elevator2 = 1'b0;

// Third request is to floor 2

#400 requested\_floor = 64'b0000000000000000000000000000000000000000000000000000000000000010;

// Emergency is on for elevator 2

#100 in\_emergency\_elevator2 = 1'b1;

#100 in\_emergency\_elevator2 = 1'b0;

// Fourth request is to floor 12

#700 requested\_floor = 64'b0000000000000000000000000000000000000000000000000000100000000000; reset\_elevator2 = 1'b1; reset\_elevator1 = 1'b1;

#100 reset\_elevator1 = 1'b0; reset\_elevator2 = 1'b0;

// Emergency is on for elevator 1

#50 in\_emergency\_elevator1 = 1'b1;

#100 in\_emergency\_elevator1 = 1'b0;

// Testing rest feature

#3000;

$finish;

end

// Clock function (50ns clock cycle)

always #50 clk = ~clk;

endmodule

**Testbench Verilog Screenshots**

****

****