Lab 2 : Elevator

EECE5117C

Team 3

**Purpose**

The purpose of this lab was to learn how to correctly utilize finite state machines by creating a program simulating the function of an elevator. This state machine is a Moore state machine, since the output of the machine depends only on the state.

**Procedure**

We wrote each function separately and tested it on the board, recursively editing them until our tests passed. Then once each function worked as expected we moved on to the next function. The following procedure was used for this lab. We used this method for all functions.

**Steps**

1. Design pseudo code for how an elevator should work
2. Draw State Diagrams
3. Write test cases
4. Label the inputs and wires with the corresponding variables
5. Write the code from the diagram
6. Simulate in Vivado
7. Test on board

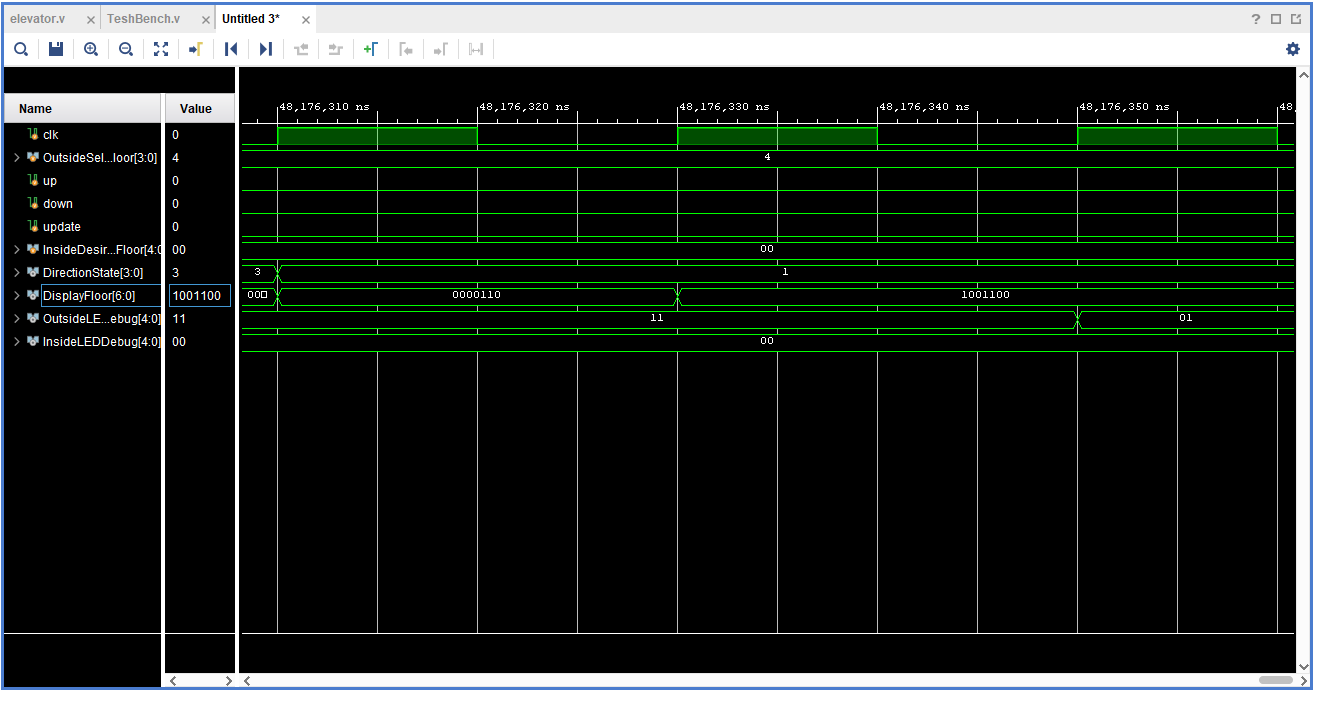


Figure 1: Simulation

Above in Figure 1 shows one of our test cases simulated. The test moves up to floor 4, which is shown by looking at the “DisplayFloor[6:0]” register which corresponds to the binary codes which display on the 7 segment display. Shown in the picture, the elevator reaches 1001100 which is the binary for “4” in the 7 segment display.

**Test Bench Code**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

//

// Elevator Project Testbench

// Contains Test methods for testing the various possibilities of the elevator program in elevator.v

//

//////////////////////////////////////////////////////////////////////////////////

module TestBench;

//Inputs

reg clk = 0;

reg [3:0] OutsideSelectedFloor;

reg up;

reg down;

reg update;

reg [4:0] InsideDesiredFloor;

//Outputs

wire [3:0] DirectionState;

wire [6:0] DisplayFloor;

//wire [3:0] Anode\_Activate; //Used for 3 elevator testing

wire [4:0] OutsideLEDDebug;

wire [4:0] InsideLEDDebug;

//UUT

Lab2Elevator uut(

.clk(clk),

.OutsideSelectedFloor(OutsideSelectedFloor),

.up(up),

.down(down),

.update(update),

.InsideDesiredFloor(InsideDesiredFloor),

//Outputs

.DirectionState(DirectionState),

.DisplayFloor(DisplayFloor),

//.Anode\_Activate(Anode\_Activate),

.OutsideLEDDebug(OutsideLEDDebug),

.InsideLEDDebug(InsideLEDDebug)

);

always begin

//Simulate the clock

#10 clk = 1;

#10 clk = 0;

end

// Add stimulus here

initial begin

update = 1'b0;

OutsideSelectedFloor = 0;

up = 0;

down = 0;

InsideDesiredFloor = 0;

//Just one sample test case shown below, more are defined later in the “Test Vectors Section”

// Request the elevator to pick someone up on the 1st floor going up

OutsideSelectedFloor = 4'b0000;

up = 1'b1;

update = 1'b1;

#200 update = 1'b0;

#200

//Request the elevator to pick someone up on the 5th floor going down

OutsideSelectedFloor = 4'b0100;

up = 1'b0;

down = 1'b1;

update = 1'b1;

#200 down = 1'b0;

update = 1'b0;

end

endmodule

**Test Vectors**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Vectors | OutsideSelectedFloor | UP | Down | DirectionState (output) | DisplayFloor  (output) |
| 1 | 0000 | 1 | X | 11= stop | 0000 |
| 2 | 0001 | 1 | 0 | 11= stop | 0001 |
| 3 | 0010 | 1 | 0 | 11= stop | 0010 |
| 4 | 0011 | 1 | 0 | 11= stop | 0011 |
| 5 | 0001 | 0 | 1 | 11= stop | 0001 |
| 6 | 0010 | 0 | 1 | 11= stop | 0010 |
| 7 | 0011 | 0 | 1 | 11= stop | 0011 |
| 8 | 0100 | X | 1 | 11= stop | 0100 |

Figure 2: Test Vectors for Elevator moving to calling floor (Testing outside call Buttons)

|  |  |  |  |
| --- | --- | --- | --- |
| Test Vectors | FloorInsideDesiredFloor | DirectionState (output) | DisplayFloor  (output) |
| 1 | 0000 | 11= stop | 0000 |
| 2 | 0001 | 11= stop | 0001 |
| 3 | 0010 | 11= stop | 0010 |
| 4 | 0011 | 11= stop | 0011 |
| 5 | 0100 | 11= stop | 0100 |

Figure 3: Test Vectors for Elevator moving to Desired floor (Testing inside Buttons)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Vectors | OutsideSelectedFloor | FloorInsideDesiredFloor | UP | Down | DirectionState (output) | DisplayFloor  (output) |
| 1 | 0010 | 0000 | 1 | 0 | 11= stop | 0010 then 0000 |
| 2 | 0010 | 0001 | 1 | 0 | 11= stop | 0010 then 0001 |
| 3 | 0010 | 0010 | 1 | 0 | 11= stop | 0010 then 0010 |
| 4 | 0010 | 0011 | 1 | 0 | 11= stop | 0010 then 0011 |
| 5 | 0010 | 0100 | 1 | 0 | 11= stop | 0010 then 0100 |
| 6 | 0010 | 0000 | 0 | 1 | 11= stop | 0010 then 0000 |
| 7 | 0010 | 0001 | 0 | 1 | 11= stop | 0010 then 0001 |
| 8 | 0010 | 0010 | 0 | 1 | 11= stop | 0010 then 0010 |
| 9 | 0010 | 0011 | 0 | 1 | 11= stop | 0010 then 0011 |
| 10 | 0010 | 0100 | 0 | 1 | 11= stop | 0010 then 0100 |

Figure 4: Test Vectors for Outside and Inside call Buttons if outside pressed briefly before inside

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Vectors | DisplayFloor before inputs | DirectionState before input | FloorInsideDesiredFloor | DirectionState (output) | DisplayFloor  (output) |  |
| 1 | 0001 | 01=up | 0010 | 11= stop | 0010 |  |
| 2 | 0011 | 00=down | 0010 | 11= stop | 0010 |  |
| 3 | 0001 | 01=up | 0000 | 01= up | 0010 |  |
| 4 | 0011 | 00=down | 0011 | 00= down | 0010 |  |

Figure 5: Test Vectors for pressing a call button while elevator is moving

**Problems**

There were numerous problems during this lab. Originally, the lab requested 3 elevators in the system. We had a lot of problems with this area in particular. After getting a single elevator to work, we tried to utilize similar logic for multiple elevators. This involved having three case statements instead of just one for checking the current state of each elevator. When we tried this, all three elevators kept moving to the same floor that was requested. This is not intended, as only one elevator should pick someone up at a time.

After debugging the three-elevator code, we realized that there was nothing preventing elevators from moving if another elevator was already going that direction. To solve this, we tried adding multiple expressions checking the current state of all the elevators in the system. However, when debugging these statements, we found that either the elevators did not move at all, or all 3 elevators still continued to move to the requested floor. We continuously checked the logic of these expressions, but after numerous attempts, we could not get a correct result.

In addition, when implementing the single elevator, we struggled with occasions where registers were not getting updated when they should be. We realized that when you use the <= assignment symbol, everything is asynchronous, and the registers do not have to be set right at that line of code. Another problem we encountered was that we were trying to set registers in two different always statements in order to implement the storage of the requested floors that users wanted to have the elevator come to. This created multi-driven nets in the code which caused the program not to be implementable.

**Takeaways**

One of the takeaways from this lab is that a machine with a finite state machine is always in a state. It does not just run from top to bottom of a program. We also realized from this lab just how different Verilog is from C programming. Many of our problems arose from thinking about the logic as programmable logic rather than trying to create circuits with the code.

**Team Breakdown**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Cases/Program Design | Behavioral Code | Diagrams/Reports |
| Ben Cohen | x | x | X |
| Moaz Abougabal | X | x |  |
| Quinn Nye | x |  |  |