RUTGERS
UNIVERSITY

Course Name: Digital Logic Design Laboratory

Course Number: **14:332:233**

Lab #: 6

Section #: 2

Date Performed: April 23, 2021

Date Submitted: May 7, 2021

Submitted by: Qadis Chaudhry 202001055

Declaration of Authorship

I declare that this lab report is my own and has been generated by me as the result of my own work.

I confirm that I have acknowledged all main sources of help and, where I have consulted with others, this is always clearly attributed, including consultations with my TA or classmates.

I have provided the sources or references of all materials included in this report that were taken from other works, including online materials, such as pictures, graphs, diagrams, screenshots, and text quotes.

Moreover, I affirm that I have not let my classmates or other persons copy materials from this report for their own use.

Sign & Date:  May 7, 2021

Lab 6
Design of Finite State Machines

Qadis Chaudhry

May 7, 2021

This lab deals with the design of finite state machines and in this instance, the machine is a controller responsible for the operation of an elevator. There are two inputs and the two outputs which change based on the state the machine resides in, as well as the inputs given by the user. This allows for the user to control which floor the elevator will end up at based on the inputs they provide.

In order to design any state machine, the equations of operation must be defined and the corresponding tables that are needed must be constructed from which, the machine can then be manifested. In this case there are four states, one for the elevator stopped at each of the three floors, and one for the elevator moving through the second floor. The state table can then be defined as:

State	Q_2Q_1
FLR1	00
FLR2	01
MOV2	10
FLR3	11

The inputs can also be defined as R_1 and R_2 and each of the possible combinations of these inputs corresponds to a specific request from the elevator.

R_1R_2	Meaning
00	No request
01	Move to first floor
10	Move to second floor
11	Move to third floor

With these tables the transition table for the controller can be interpreted. The transition table is what summarizes the main function of the machine and it will allow for the visualization of how the controller operates. This will also lead to the construction of the state diagram of the table and from there, the implementation can be executed within Verilog, where the schematic and the waveform are engendered.

Current State	R_1 R_2 00	01	11	10
FLR1	FLR1	FLR1	MOV2	FLR2
FLR2	FLR2	FLR1	FLR3	FLR2
FLR3	FLR3	MOV2	FLR3	FLR2
MOV2	FLR1	FLR1	FLR3	FLR2

This is the transition table for the elevator. According to this table, it can be seen how the elevator reacts to changes in the input and changes its state based on the new information as well as the current state in which it resides. With this, the elevator can be coded in Verilog using behavioral programming. This will be done by defining the each of the sates and their transitions based on the inputs and from there, the waveform can be analyzed to see whether or not the correct function is achieved.

```
module Elevator(
    input reset,
    input clk,
    input R0,
    input R1,
    output reg Q0,
    output reg Q1
    );

    reg [2:0] state;
    reg [2:0] next_state;

    parameter [2:0] FLR1 = 2'b00;
    parameter [2:0] FLR2 = 2'b01;
    parameter [2:0] MOV2 = 2'b10;
    parameter [2:0] FLR3 = 2'b11;

    always@(posedge clk)
    begin
    if (reset)
    state <= FLR1;
    else
    state <= next_state;
    end

    always@(*)
    begin
    next_state = state;
    case(state)
        FLR1:
        begin
        if (R1 == 1'b0 && R0 == 1'b0) next_state = FLR1;
        else if (R1 == 1'b0 && R0 == 1'b1) next_state = FLR1;
        else if (R1 == 1'b1 && R0 == 1'b1) next_state = MOV2;
        else if (R1 == 1'b1 && R0 == 1'b0) next_state = FLR2;
        end
        FLR2:
        begin
        if (R1 == 1'b0 && R0 == 1'b0) next_state = FLR2;
        else if (R1 == 1'b0 && R0 == 1'b1) next_state = FLR1;
        else if (R1 == 1'b1 && R0 == 1'b1) next_state = FLR3;
        else if (R1 == 1'b1 && R0 == 1'b0) next_state = FLR2;
        end
        FLR3:
        begin
        if (R1 == 1'b0 && R0 == 1'b0) next_state = FLR3;
        else if (R1 == 1'b0 && R0 == 1'b1) next_state = MOV2;
        else if (R1 == 1'b1 && R0 == 1'b1) next_state = FLR3;
        else if (R1 == 1'b1 && R0 == 1'b0) next_state = FLR2;
        end
        MOV2:
        begin
        if (R1 == 1'b0 && R0 == 1'b0) next_state = FLR1;
        else if (R1 == 1'b0 && R0 == 1'b1) next_state = FLR1;
        else if (R1 == 1'b1 && R0 == 1'b1) next_state = FLR3;
        else if (R1 == 1'b1 && R0 == 1'b0) next_state = FLR2;
        end
    endcase
    assign Q0 = (state == MOV2) | (state == FLR1) | (state == FLR3);
    assign Q1 = (state == FLR3) | (state == FLR2) | (state == FLR1);
    end

endmodule
```

Figure 1: State Machine Code

This is the code for the elevator controller within Verilog and a few things can be noticed. The first is that there are four inputs and two outputs. The outputs are designated as Q_0 and Q_1 and the inputs as, R_0, R_1, *reset*, and *clk*. The inputs R_0 and R_1 are self-explanatory and they are the user inputs to the system. The other inputs however, are additional and are required for the functional operation of the machine. Since this is a state machine, the clock input is necessary as all state machines operate on a clock cycle. This particular machine operates on the positive edge of the clock, as seen by the *always* block, argumented at *posedge clk*.

Here, the reset option is invoked and this input provides the sole purpose of "turning off" the machine. With the reset enabled the machine's state will go back to the first possible state, in this case $Q_0Q_1 = 00$ or, FLR1.

From here, the design can be elaborated within Verilog and the schematic can be visualized. The schematic in this case is very unwieldy due to the fact that the elaboration does not seek to use the fewest number of gates to obtain the functionality. Despite this however, the function can be evaluated through the analysis of the waveform as done previously.

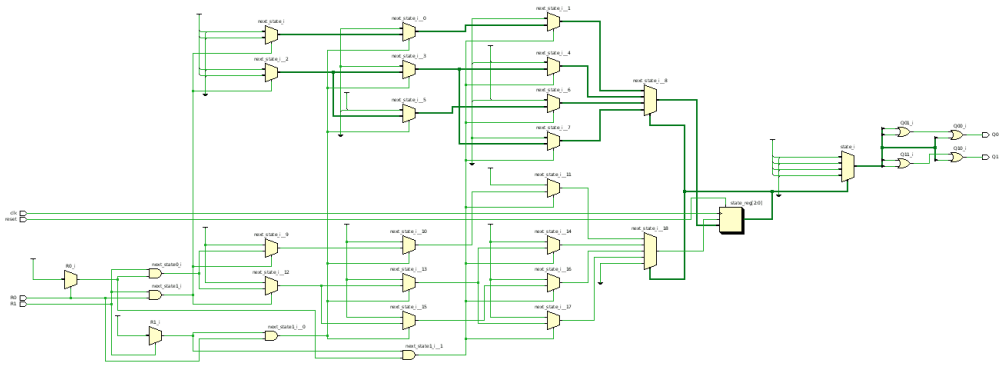

Figure 2: State Machine Schematic

This is the schematic generated and it can be seen that it is not what is really expected from the behavioral programming. Nevertheless, a simulation can be set up and the waveform will tell us if the device is working properly.

```
module Elevator_sim();
reg R0;
reg R1;
reg clk;

wire Q0;
wire Q1;

Elevator Elevator1(R0, R1, clk, Q0, Q1);

initial
begin
R0 = 1'b0;
R1 = 1'b0;
clk = 1'b0;
end

always #5 clk = ~clk;
always #10 R0 = ~R0;
always #20 R1 = ~R1;

always #255 $finish;

endmodule
```

Figure 3: State Machine Simulation Code

Above is the code for the simulation of the created circuit. Here, the inputs and output can be seen as the user inputs and the clock, and the outputs Q_0 and Q_1. The "Elevator" module is instantiated and the given all the inputs and the corresponding outputs. The inputs are initialized as all zeros, and the simulation is ready to be started. In this screenshot of the code, the reset input is not present however, it must be known that the input was added

after the fact and the waveform was constructed from there. The reset was taken as an input of the Elevator module as well and was responsible for proper starting of the machine.

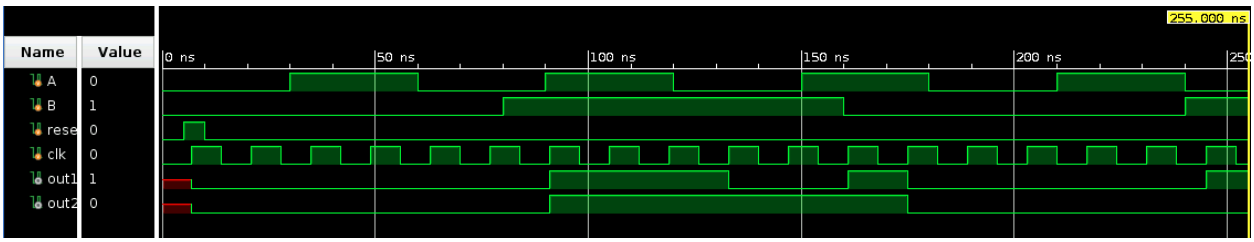

Figure 4: State Machine Waveform

With the visualization of the waveform the reaction of the outputs with respect to the inputs can be seen. This shows how the machine works with the evolution of the clock and the user inputs and from the outputs, it can be seen that they are in correspondence with the transition table. Since the output cycle follows the state cycle defined in the table, the machine is working as intended and the implementation within Verilog is correct.

Conclusion

This lab was fairly straight forward and was able to be done with simplicity as the transition table was given. The only real issue that was encountered was with the creation of the waveform as the outputs were not being represented correctly at first. This was an issue with the reset value as well and was eventually solved to form the proper waveform. This lab was able to teach the design of a state machine from the ground up and exemplified how state machines actually achieve their function. It also discovers the reality of how state machines can be used to achieve a multitude of operations and how there ability to do so might be useful in many aspects of computing.