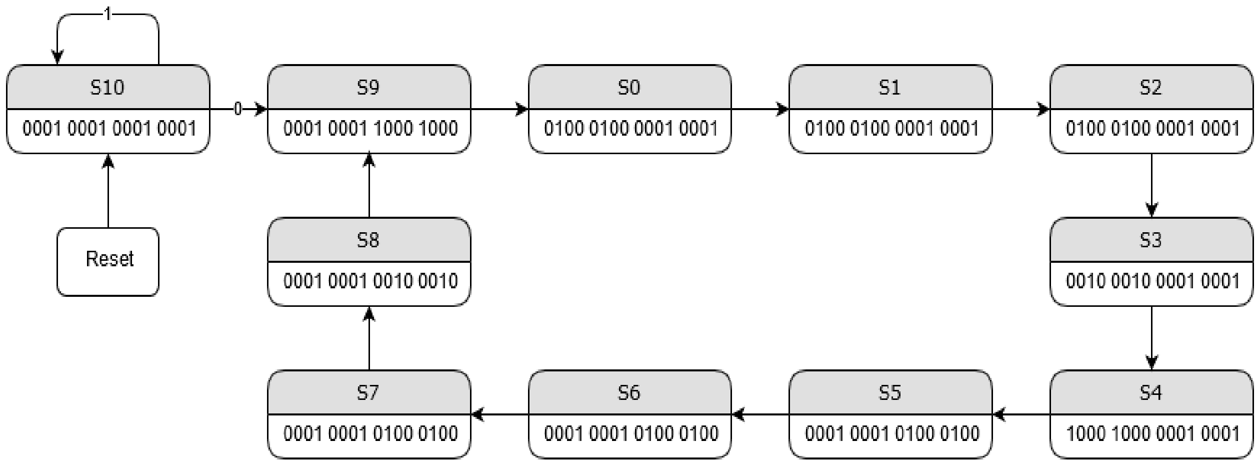
CSE 3381 Digital Logic Design Lab 9 Report

Finite State Machine Diagram



In this system, the traffic lights of a four-way intersection describe the following traffic light cycle:

- For fifteen seconds, northbound and southboundbound roads have a green light while eastbound and westbound roads have a red light.
- For five seconds, northbound and southbound roads change to a yellow light while eastbound and west roads remain red.
- For five seconds, northbound and southbound roads only allow left turns while eastbound and west roads stay red.
- For fifteen seconds, northbound and southbound roads have red lights while eastbound and west roads are green.
- For five seconds, northbound and southbound lights stay red while eastbound and west roads change to yellow.
- For five seconds, northbound and southbound lights are still red while eastbound and west roads change to a light that allows only left turns.

This cycle is described by states S0 through S9, each of which lasts for one clock cycle before advancing to the next state. The clock runs at 0.2 Hz, or lasts for five seconds per cycle. The traffic lights can also be asynchronously reset by a reset switch, which changes all intersection lights to red. The lights will remain red for as long as the reset switch is held. The system only re-enters the cycle on the after the reset switch is released on the next rising edge of the clock.

Each state is associated with a one-hot code that describes which lights in the intersection are turned on or off. Reading the bits left to right: the first four bits describe the northbound lights, the next four bits describe the southbound lights, the next for after that describe the eastbound lights, and the last four bits describe the westbound lights. In each set of four bits, the first bit controls the left-turn light, the second bit controls the green light, the third bit controls the yellow light, and the fourth bit controls the red light.

Finite State Machine Code

The divider module describes a clock divider that outputs a clock with a period of 5 seconds and updates the machine's state on each positive edge.

```
module divider(
    input clk,
    input wire [3:0] next_state,
    output reg clk_divider,
    output reg [3:0] state
    );
reg [31:0] count;
always @ (posedge(clk))
begin
    if (count == 250000000 - 1)
    begin
        count <= 32'b0;
        clk_divider <= ~clk_divider;
        state <= next_state;
    end
    else
        count <= count + 1;
end
endmodule
```

Above is the code for the clock divider used to implement the traffic light system on a BASYS 3 board. It uses a counter to count up to 250,000,000 before flipping the output signal. From a 100 Mhz clock, this creates divided clock with a period of 5 seconds.

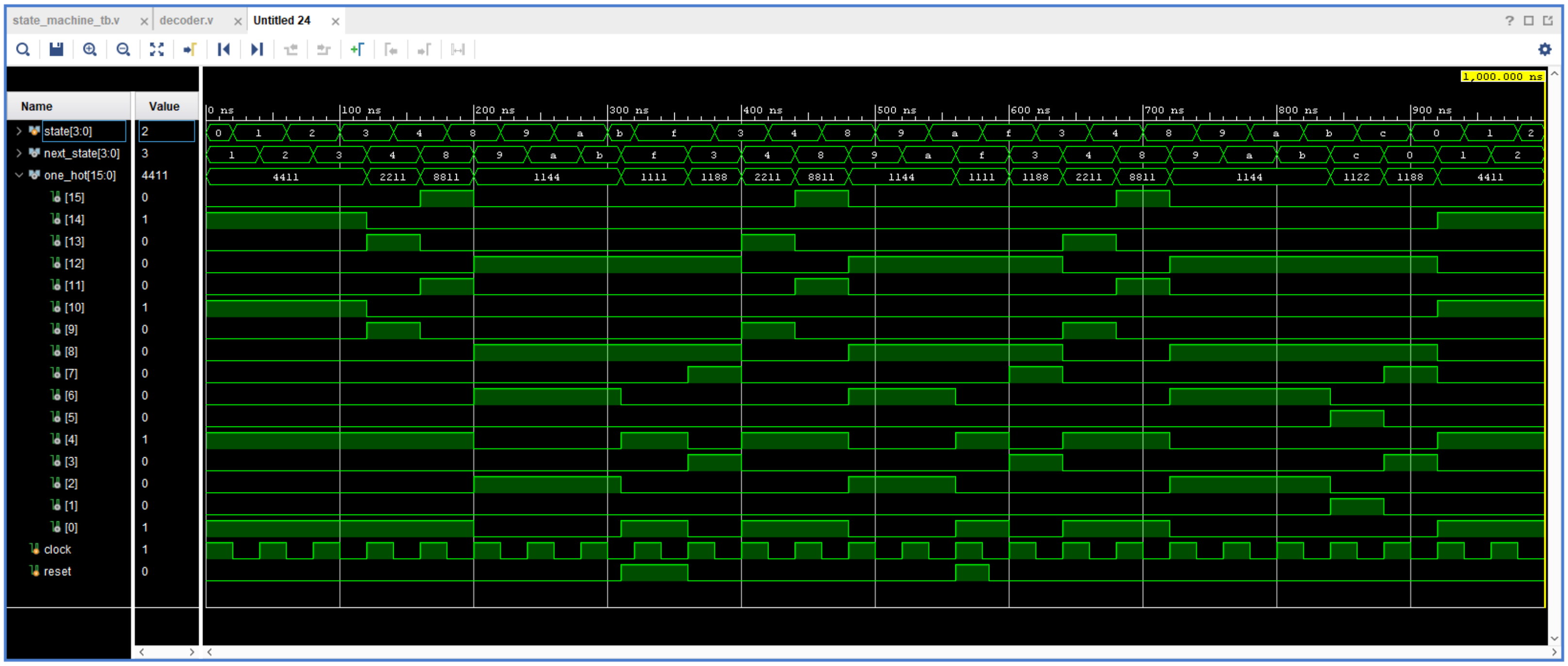
This next module interprets the correct one-hot code from a state described by four bits. On each rising edge of the divided clock or reset switch, the decoder informs the board what lights should be on or off based on the current state.

```
module decoder(
    input clk,
    input rst,
    input wire [3:0] state,
    output reg [15:0] one_hot,
    output reg [3:0] next_state
    );

always @(posedge clk or posedge rst)
begin
    if (state[3] == 0)
    begin
        one_hot[12] <= 0;
        one_hot[8] <= 0;
        one_hot[7:4] <= 4'b0001;
        one_hot[3:0] <= 4'b0001;
        if (state[2] == 0)
        begin
            one_hot[15] <= 0;
            one_hot[14] <= ~&state[1:0];
            one_hot[13] <= &state[1:0];
            one_hot[11] <= 0;
            one_hot[10] <= ~&state[1:0];
            one_hot[9] <= &state[1:0];
        end
        else
        begin
            one_hot[15:12] <= 4'b1000;
            one_hot[11:8] <= 4'b1000;
        end
    end
    else
    begin
        one_hot[15:12] <= 4'b0001;
        one_hot[11:8] <= 4'b0001;
        one_hot[4] <= 0;
        one_hot[0] <= 0;
        if (state[2] == 0)
        begin
            one_hot[7] <= 0;
            one_hot[6] <= ~&state[1:0];
            one_hot[5] <= &state[1:0];
            one_hot[3] <= 0;
            one_hot[2] <= ~&state[1:0];
            one_hot[1] <= &state[1:0];
        end
        else
        begin
            one_hot[7:4] <= 4'b1000;
            one_hot[3:0] <= 4'b1000;
        end
    end

    if(rst)
    begin
        next_state <= 4'b1111;
        one_hot <= 16'b0001000100010001;
    end
    else
    begin
```

```
        if(state[2] == 1)
            next_state = state + 4;
        else
            next_state = state + 1;
    end
end
endmodule
```

The series of if statements form the logic that decodes a four-bit state into a traffic light pattern. The always block responds to the positive edge of both the clock and reset switch, allowing for resetting that is asynchronous from the clock signal. If the reset switch is on, the machine enters S10 with a one-hot code of 0001000100010001 which corresponds to red lights in every direction.

The testbench image below shows how the machine cycles through the states, and can be asynchronously reset to S10 before reentering at S9 on the next positive clock edge.



Post Lab Questions

1. A Mealy state machine is a FSM whose output is determined by its current inputs and its current state. A Moore state machine is a FSM that only uses its current state to generate an output.
2. A vending machine is a Mealy machine, as the food it outputs depends on the selection that is input and the current inventory state. The mechanism for changing channels on a TV is also a Mealy machine, the next channel depends on the current channel and whether the user is switching to the previous or the next channel. An elevator is a Mealy machine, as a user needs to input a floor to change elevator state and move to another floor.
   Clocks used to keep time are Moore machines. A radio is a Moore machine, the audio is outputs depends only on what wavelength it is set to receive. A lamp is a Moore machine, the light produced depends on whether its current state is on or off.
3. This traffic light system is a Moore machine. The output of the lights is only tied to the current state of the system. The reset switch can determine what the next state will be, but it is not considered when generating a state’s output.