Traffic Light Controller

Jonathan Frey
Mohammad Raashid Ansari
William Nitsch
Group 4
ECE811

Introduction

A Finite State Machine (FSM) is a mathematical model used to represent a sequential logic circuit. The term state machine comes from that the logic circuit usually has a certain number of states at which it will have different outputs for different states. The term finite in FSM says that the state machine has a finite number of states. FSMs are lumped into two categories, Mealy and Moore machines. For a Mealy machine, typically the outputs of the FSM exist based on the current state. For a Moore machine, outputs will change when the machine changes states.

For this final project, we were to design a FSM with delays to model a four-way traffic light. The FSM was designed to control the colors of the traffic lights. A separate counter module was used to create variable minimum times that the respective lights stay as that color. The machine we designed was a Mealy machine, therefore the outputs (light colors) change depending on the current state.

Design

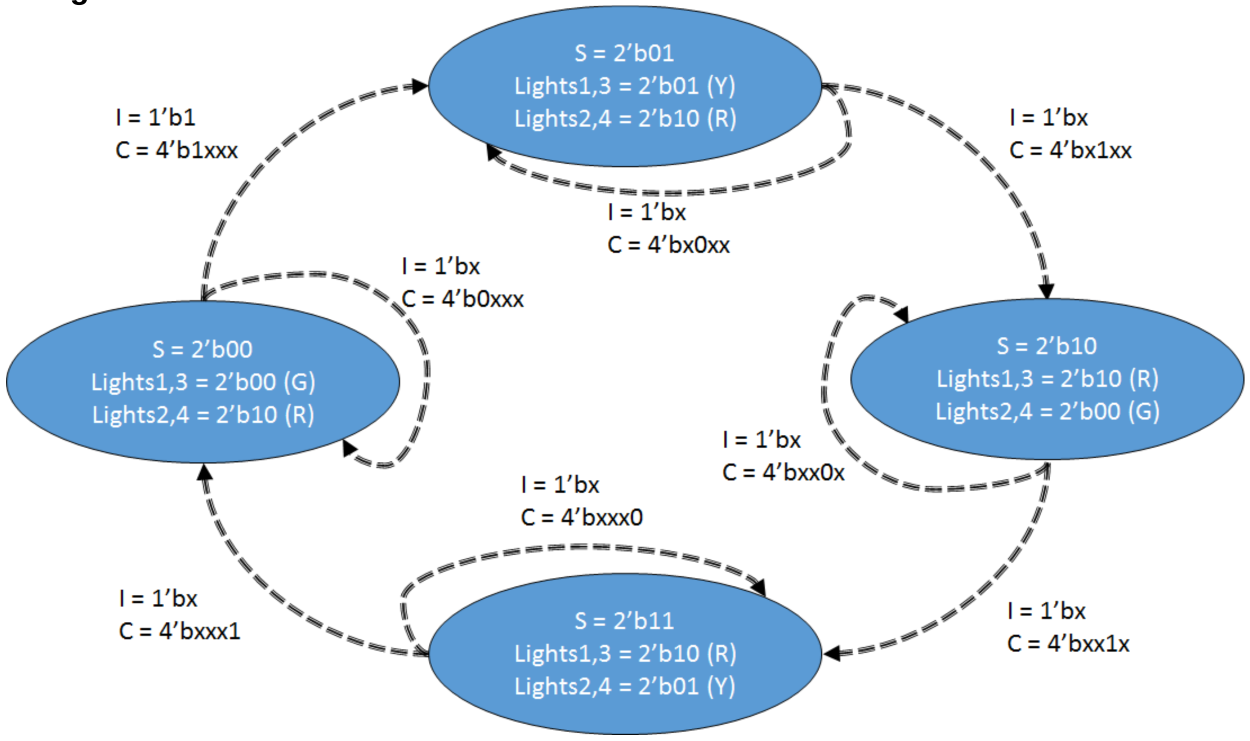

Figure 1. Understanding of FSM design

The FSM seen in Figure 1 is based around two sets of inputs, and three sets of outputs. The two inputs are the car sensor and the output of the timers, or the setbit. For the two car sensors, we use two signals, that we call car2 and car4. We perform the OR function with these two signals as inputs, and get a single output we call “I,” or our interrupt. When there is no car at light 2 or 4, both signals are logic 0, this would lead to “I” being 0. For the cases when there is a car at either or both sensors, this leads to “I” being logic 1. The other input, setbit, or the output of the timers, basically says if the respective delay has been reached or not. If the delay has not been reached yet for the respective state, the respective timer output would be logic 0. If the delay has been reached for the current state, then the respective output would be logic 1.

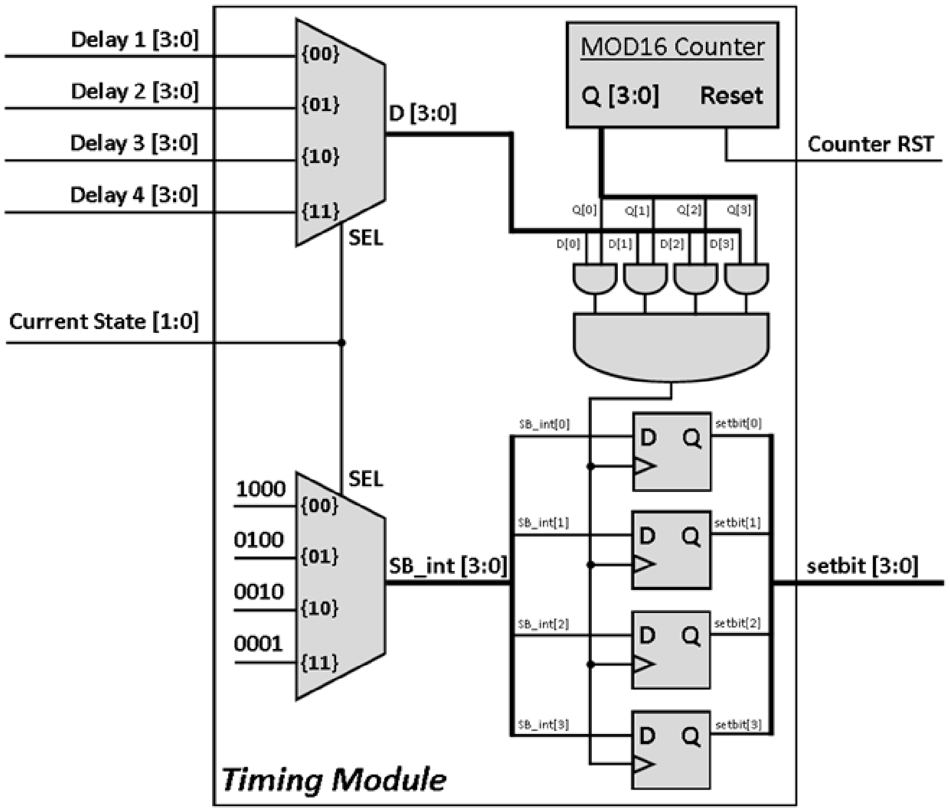

Figure 2. Schematic of Timing Module

In order to allow for multiple delay times using the smallest amount of counters while allowing the delay times to be variable as external inputs, multiplexers were used to automatically select the correct delay input and output (setbit) based on the current state. If the current state is {00}, delay 1 is selected and a value of {1000} for SB_int is selected. SB_int is an internal register for the future output of setbit. The same situation occurs for current states of {01}, {10} and {11}, where the associated delay and SB_int value is selected and used in the other parts of the circuit. After a delay is selected, the value of that delay is compared to the current value of the counter. When the counter value is equal to that of the selected delay, the output of the 4 port AND gate is high for that clock cycle, causing SB_int to be latched to setbit. Setbit is a condition used in the FSM for state transitions. The counter is reset externally when the state changes. In situations where the state does not change for a long time (green on main street with no cars on side streets for some duration), the counter is allowed to roll over, since the delay minimum will have been met and the setbit output will be latched on the first instance of delay equalling the value of the counter. All subsequent instances where delay and the counter output are equal to one another are insignificant, as the same value of setbit will be latched to replace it, meaning nothing changes after the first transition. The FSM is not allowed to change states until the correct value of setbit is output for a given current state.

The FSM was designed by Jonathan, I and William worked on the Timing Module and the Top Module. We needed to integrate the FSM with the Timing module for the whole project to work as required. The presence of a car on light 2 or 4 is represented by "*i*". A car at any of the lights 2 or 4 would trigger the Timing module. Signals *car2* and *car4* represent a car at light 2 and 4 respectively. We assigned "*i"* as *car2* or *car4* so that a car on either side would be able to trigger the FSM to enable the Timing module. Figure below shows the schematic of the top module.

Testbench Justifications

The FSM testbench was designed to check that the FSM module functions as the design objectives were specified. Since the delay/timer module was not built yet when the FSM was designed and tested, dummy signals for the timer were used. The testbench first initializes the car, count, and state signals to logic 0. These signals are if a car is present, the current timer output, and the current state, respectively. The testbench updates the current state based off of the nextstate, where it sets state equal to the nextstate, or what the new state is being computed to be. The testbench now checks that the state doesn't change from state 00 when the delay has been reached, but there is no car present. Next, the testbench makes sure, with the introduction of a car onto the sensor when the delay has already been reached, that the state changes from 00 to 01. This causes the next states to change whenever the delay has been reached. The testbench now checks that the other states change appropriately with the introduction of the respectively timer control bit. Overall, this covered all the test cases needed to verify that the FSM works.

The different verification tests for the Timer testbench include the testing of different delay times up to 15 (the max value of the MOD16 counter). This particular testbench demonstrates the correct delay implantation for delays of 10, 5, 2 and 15 (times the 20 nanoseconds for true delay duration since Tclk = 20ns). Each state change in the test bench (cs, "current state") triggers the correct selection of setbit and input delay. Comparing the selected delay time in the snapshots seen in Figure 6 to the time at which the value of setbit changes relative to the state change shows that the delay is effectively implemented. Additionally, long times were placed between state changes to show that rollover of the counter does not affect the output of the timing module. In order to see that rollover does not affect it again, a second instance of delay being equal to the counter output is included for each delay time.

For testing purposes our first test-case was to check if our design keeps the main road lights green (represented by 00 in *l13*) until a car arrives.

The second test-case was to check the response of our design when a car arrives at light 2 (*car2)*, the lights change as expected. Light 1&3 (*l13)* change from green (00) to yellow (01), then after a delay (defined by *d2*), *l13* changes from yellow (01) to red (10), after delay defined by *d3*. At the same instant lights 2 & 4 change from red (10) to green (00). After a delay (defined by *d4*) the lights 1 & 3 change to green (00) and lights 2 & 4 change to red (10).

The third test-case checks if a car is at light 4 (*car4*). The design responds same as it does in the second test-case. This validates that a car could be at any of the lights 2 or 4 and our design would work as expected.

In the fourth test-case, *car2* and *car4* arrive at the same time, *car2* leaves before *car4*, and later *car4* leaves, the waveform show that the lights change after *car4* leaves. This validates our design.

In conclusion, we have tested our design for all the cases possible in a real scenario and our design works as expected.

We had to connect the nextstate pins and current state pins to update the FSM whenever there's a change in the presence of a car. Whenever there is a state change, the testbench resets the timing module. When there is no change in the state the timing module is not reset which keeps the state change consistent, it also keeps the Timing module in sync with the FSM.

Post-Synthesis

Post-synthesis was done using Synopsys' Design Vision to verify that our design was synthesizable and worked properly. After some changes were made to the code, the design was successfully synthesized and reported and area of 166. Figure 3 shows a close-up view of the schematic of the FSM module. Figure 4 shows the schematic of the timer module. Figure 5 shows the schematic of the top module. Figure 6 shows four examples of the timer module working correctly, as well as the variable delay working. Figure 7 shows the post-synthesis simulation of the design. Figures 8-12 show five snapshots of light transitions.

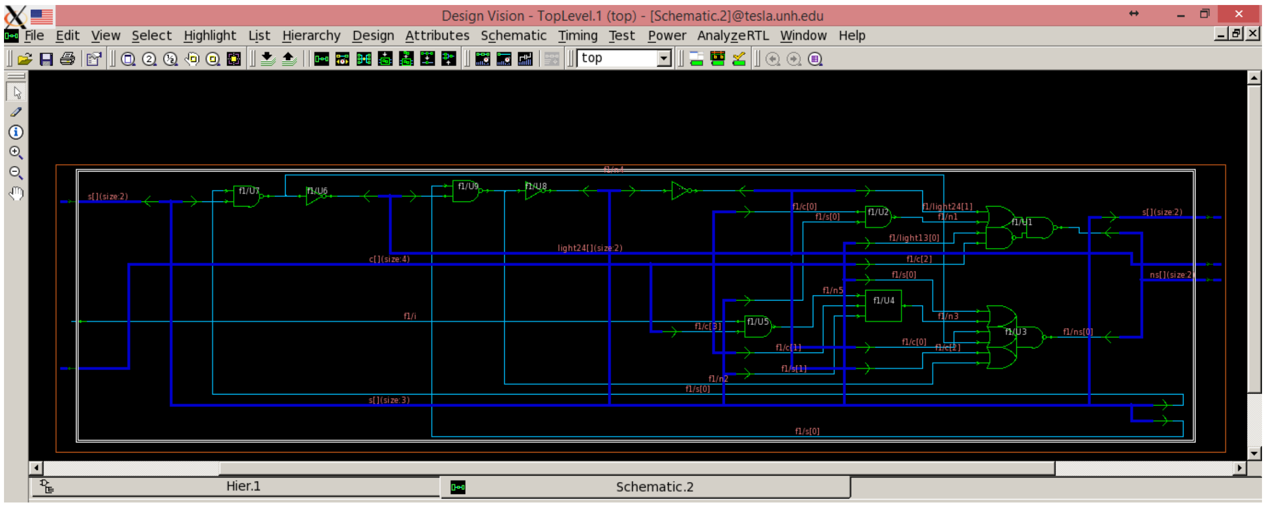

Figure 3. Post-synthesis schematic of the FSM module

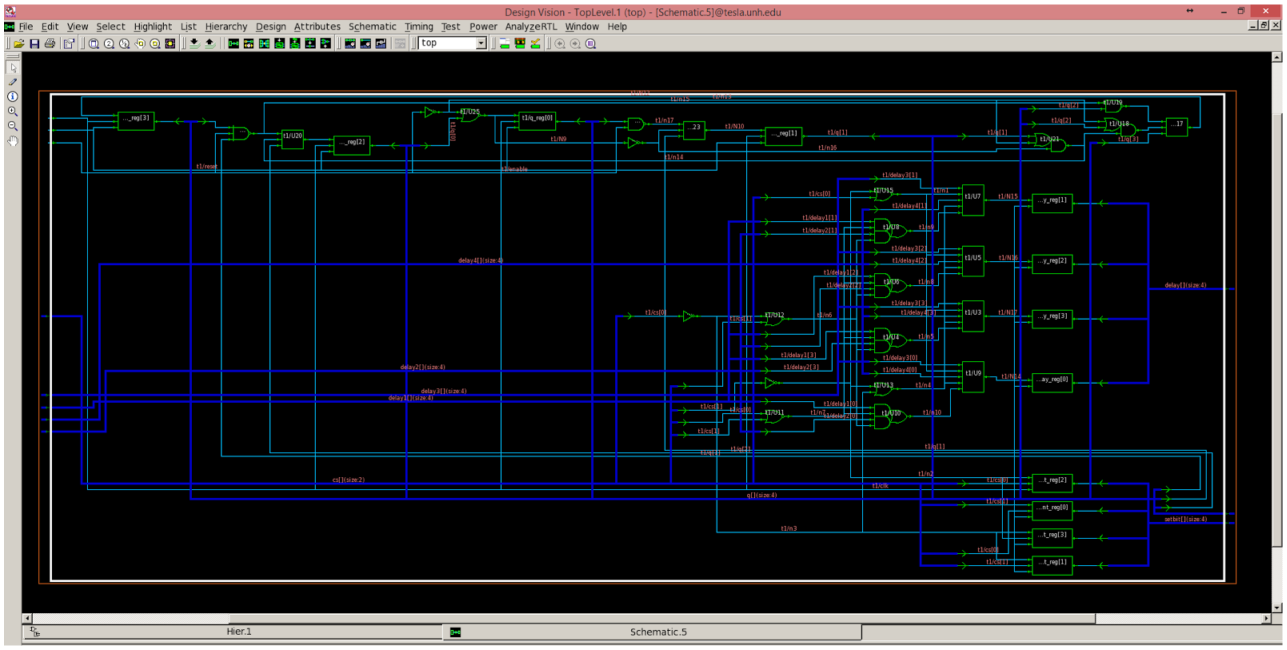
Design Vision - TopLevel.1 (top) - [Schematic.5]@tesla.unh.edu
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
top
Hier.1
Schematic.5

Figure 4. Post-synthesis schematic of Timer Module

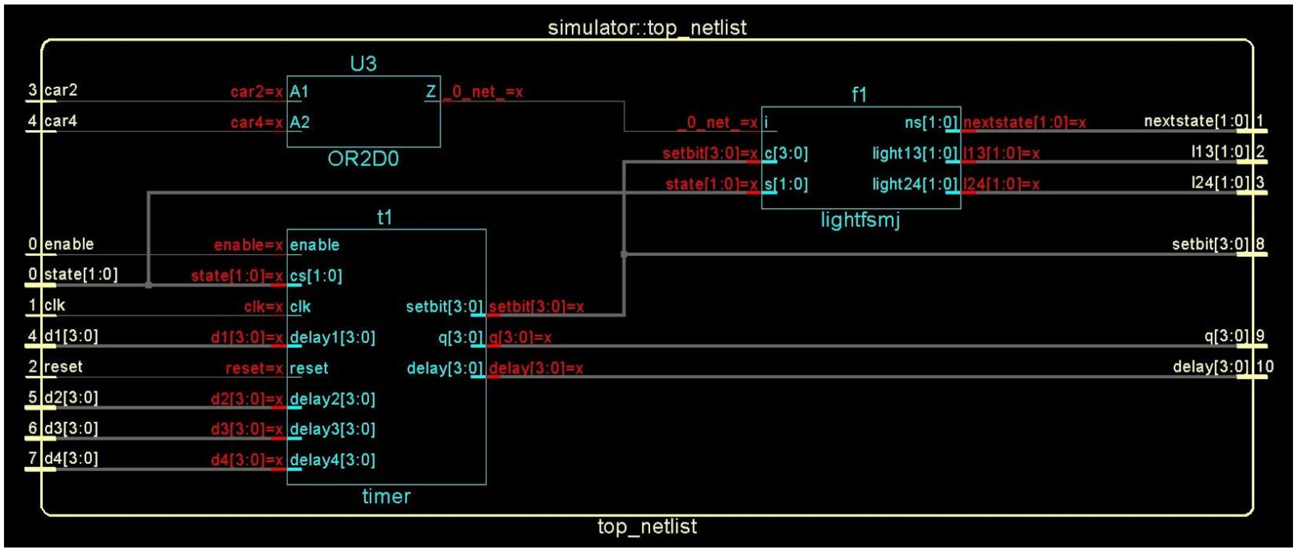

Figure 5. Post-synthesis of Top Module

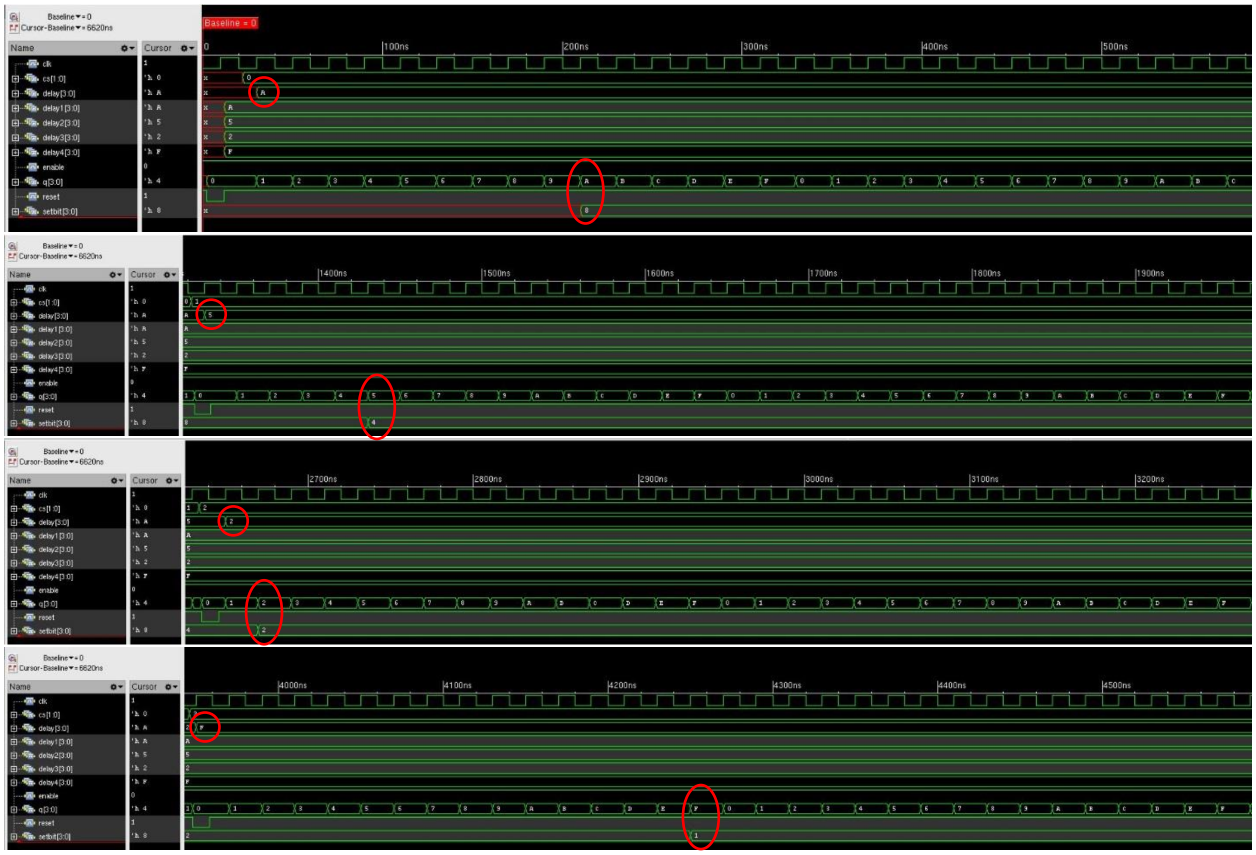

Figure 6. Examples of delay timer working properly

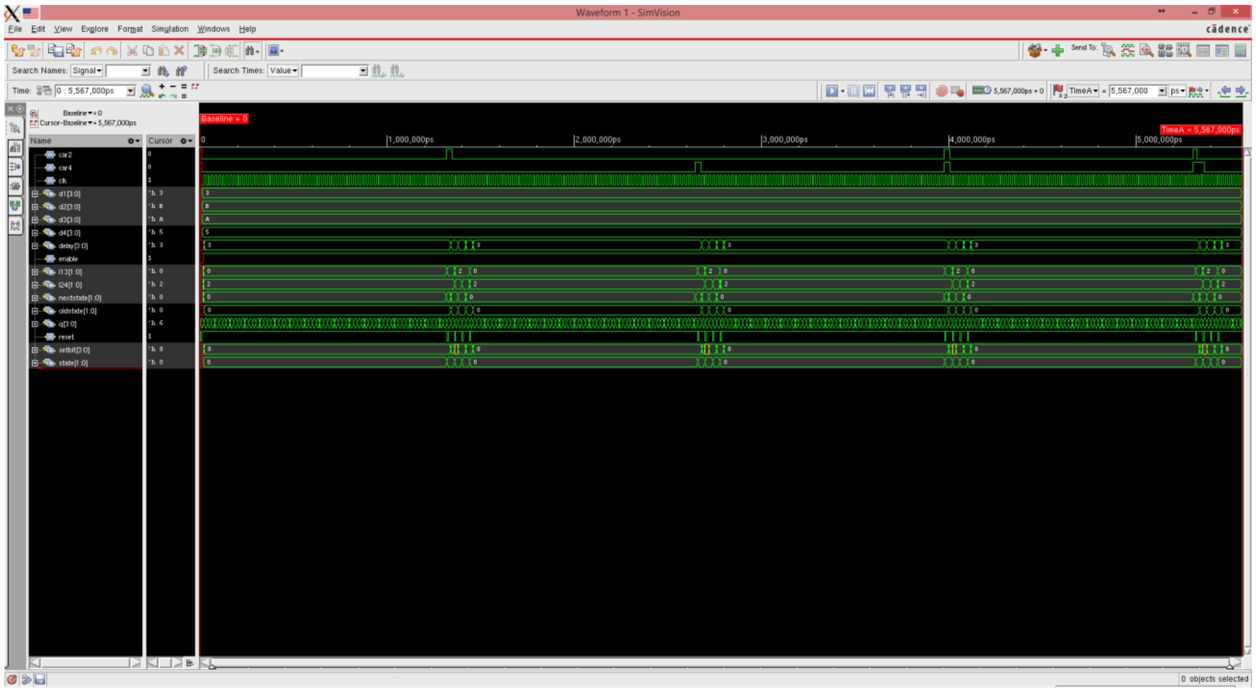

Figure 7. Post-Synthesis Simulation

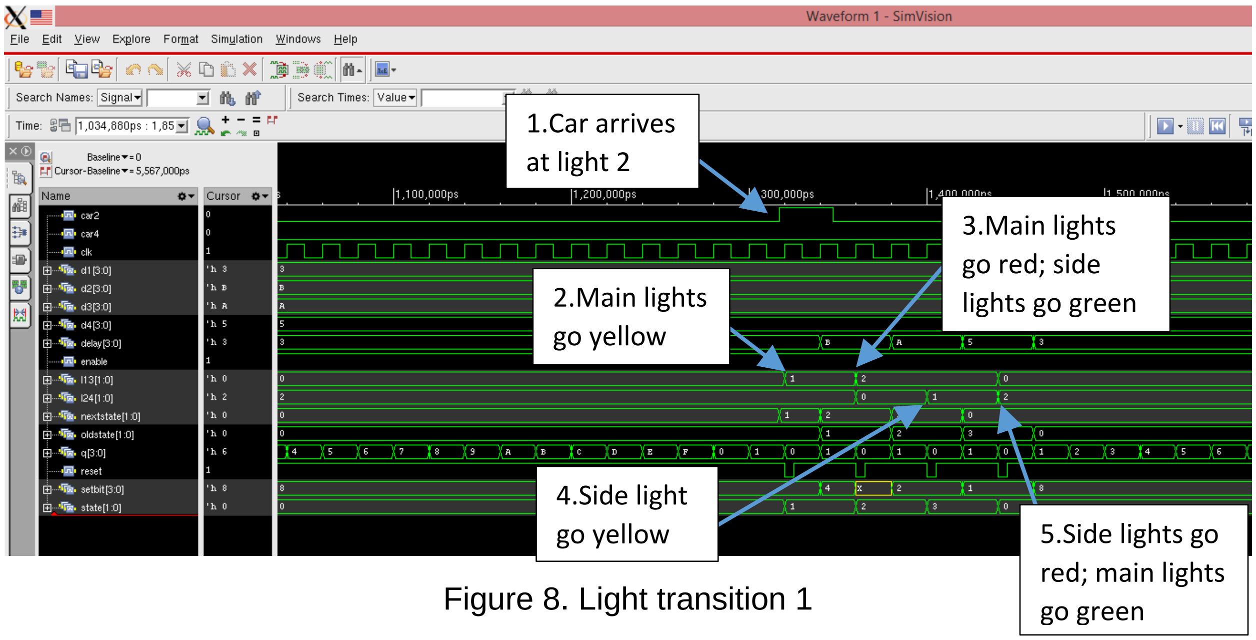

Figure 8. Light transition 1

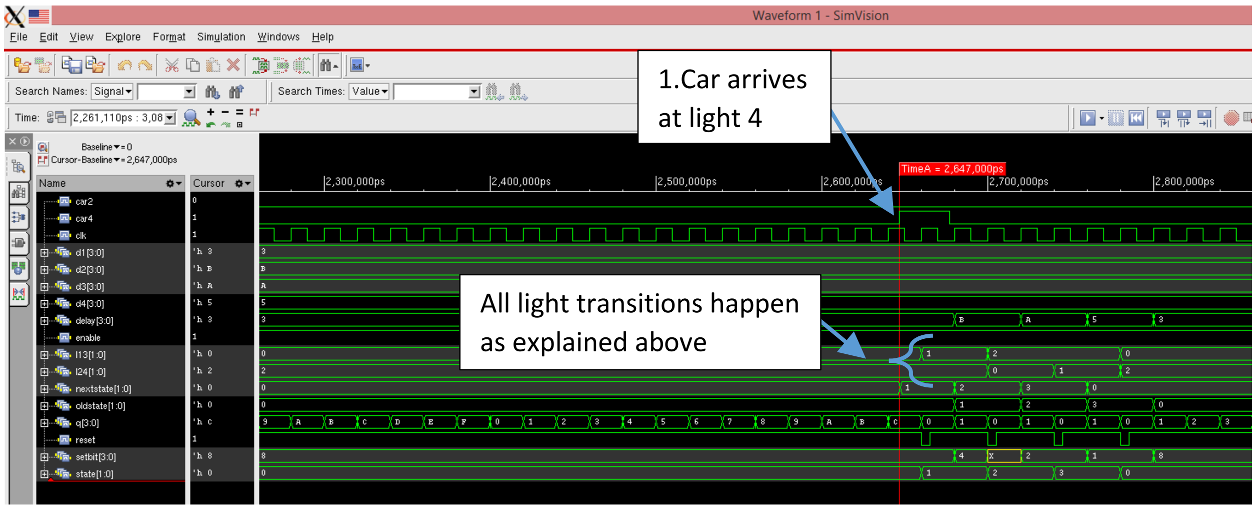

Figure 9. Light transition 2

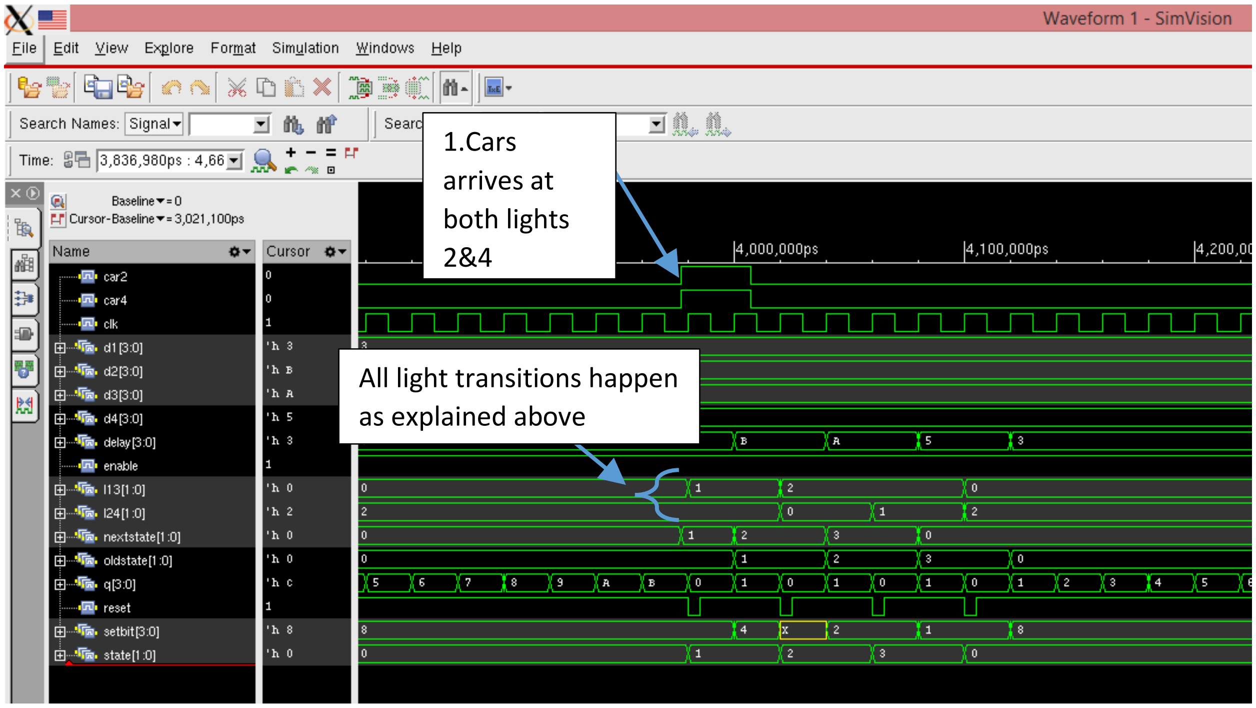

Figure 10. Light transition 3

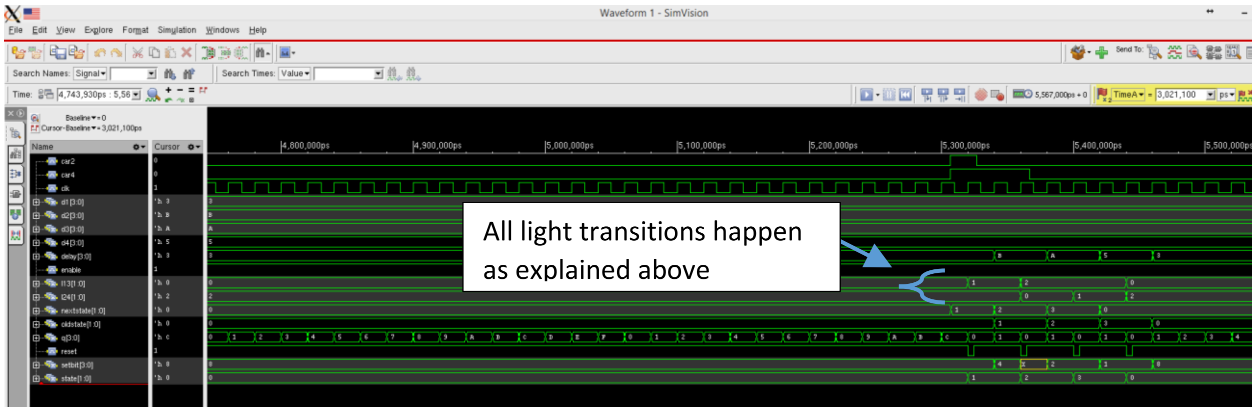

Figure 11. Light transition 4

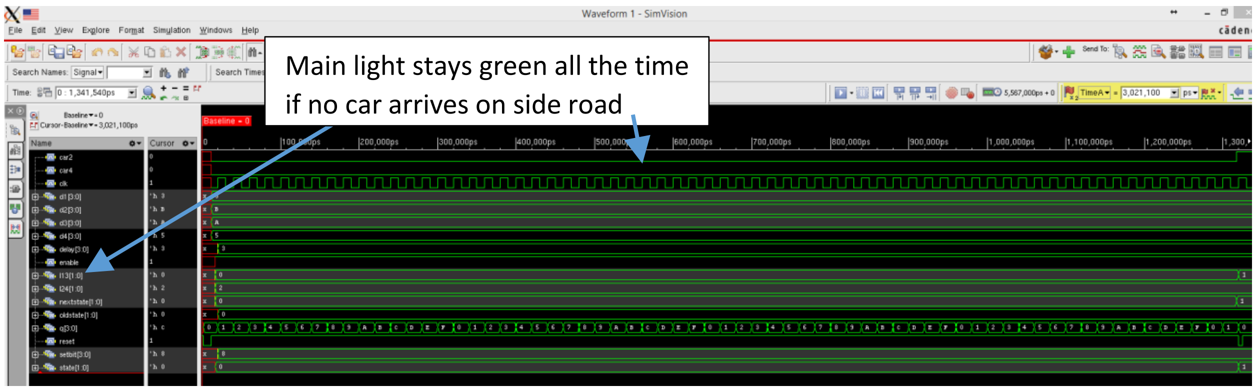

Figure 12. Light transition 5

Revision History

FSM Module:

1. Designed FSM module.
2. Fixed outputs in always block.

Timer Module:

1. Four MOD4 counters with a single internal and non-variable delay times.
2. Changed to single MOD16 counter. Added two multiplexers to allow for current-state dependent delay and setbit value selection to make module more consolidated. Added external reset for counter and current state input. Used AND gates at output to create a pulse when delay was complete.
3. Changed output AND gates to D-Latches so that value was sustained. This was easier to use in the design than a pulse.

Top Module:

1. Connected Timer and FSM modules in Top module by instantiation of sub-modules.
2. Fixed module by using testbench to get it to simulate.
3. Implemented always loop to change states in Top Module rather than the testbench.
4. Verified all scenarios using testbench.
5. Put design through Post-synthesis.
6. Fixed errors in Post-synthesis.
7. Tested all scenarios again using testbench and verified design.

Code

FSM Source Code:

```
`timescale 1ns/1ns
module lightfsmj(i,c,s,ns,light13,light24);
	input i;
	input[3:0] c; //counter
	input [1:0] s; //state
	output[1:0] ns; //next state
	output reg[1:0] light13; //controls lights 1 and 3
	output reg[1:0] light24; //controls lights 2 and 4
	assign ns[1] =
((~s[1])&(s[0])&(c[2]))|(s[1]&(~s[0])&(~c[1]))|(s[1]&(~s[0])&(c[1]))|(s[0]&s[1]&(~c[0]));
//expression for MSB of next state
	assign ns[0] =
((~s[1])&(~s[0])&c[3]&i)|((~s[1])&(s[0])&(~c[2]))|((s[1])&(~s[0])&c[1])|((s[1])&(s[0])&(~
c[0])); expression for LSB of next state
	always@(s) begin
	light13[0] <= ~s[1]&s[0]; expression for LSB of lights 1 and 3
	light13[1] <= s[1]; expression for MSB of lights 1 and 3
	light24[0] <= s[1]&s[0]; expression for LSB of lights 2 and 4
	light24[1] <= ~s[1]; expression for MSB of lights 2 and 4
	end
endmodule
```

FSM Testbench Code:

```
`timescale 1ns/1ns
module lightfsmtbj;
	wire[1:0] light13, light24, nextstate;
	reg car;
	reg[3:0] count;
	reg[1:0] state;
	lightfsmj T0(.i(car), .c(count), .s(state), .ns(nextstate), .light13(light13),
.light24(light24)); //instantiation of FSM
	initial begin
	car = 1'b0; //no car is there
	count = 4'h0; //initialize dummy counters
	state = 2'h0; //initialize state to state 00
	#40 state = nextstate; //update
	#20 count = 4'h8; //set dummy counters
	#20 state = nextstate; //update
	#10 car = 1'b1; //car is there
	#30 state = nextstate; //update
	#10 count = 4'h4; //set dummy counters
	#10 state = nextstate; //update
```

```
        #10 car = 1'b0; //no car is there
        #10 state = nextstate; //update
        #10 count = 4'h2; //set dummy counters
        #10 state = nextstate; //update
        #10 car = 1'b1; //car is there
        #10 state = nextstate; //update
        #10 count = 4'h1; //set dummy counters
        #10 state = nextstate; //update
        #10 car = 1'b0; //no car is there
        #10 state = nextstate; //update
        #20 $finish;
        end
endmodule
```

Timer Code:

```
`timescale 1ns/1ns
module timer(enable,clk,reset,setbit, cs, delay1, delay2, delay3, delay4, q, delay);
        input [3:0] delay1, delay2, delay3, delay4;
        output reg[3:0] q, delay, setbit;
        input enable, clk, reset;
        reg [3:0] sb_int;
        input [1:0] cs;
        wire d0, d1, d2, d3;
        //combinational logic for MOD-16 counter
        assign d0 = (~q[0]&~q[1])|(~q[0]&q[1]);
        assign d1 = (q[0]&~q[1])|(~q[0]&q[1]);
        assign d2 = (q[0]&q[1]&~q[2])|(~q[0]&q[2])|(~q[1]&q[2]);
        assign d3 =
(q[0]&q[1]&q[2]&~q[3])|(~q[0]&q[2]&q[3])|(~q[1]&q[2]&q[3])|(~q[2]&q[3]);

        //sequential logic for MOD-16 counter
        always@(posedge clk or negedge reset) begin
        if(~reset) begin
        q <= 4'b0000;
        end
        else begin
        if(enable) begin
                q[0] <= d0;
                q[1] <= d1;
                q[2] <= d2;
                q[3] <= d3;
        end
        else begin
                q[0] <= 1'b0;
                q[1] <= 1'b0;
                q[2] <= 1'b0;
                q[3] <= 1'b0;
        end
        end
        end

        //logic that decides which delay to choose according to the state of the FSM
        always@(posedge clk) begin
        if(cs == 2'b00) begin
           sb_int <= 4'b1000;
                delay <= delay1;
        end
        else if(cs == 2'b01) begin
                sb_int <= 4'b0100;
                delay <= delay2;
                end
        else if(cs == 2'b10) begin
              sb_int <= 4'b0010;
                delay <= delay3;
                end
        else begin
                sb_int <= 4'b0001;
                delay <= delay4;
        end
        end

        always@(delay == q) begin
        if (delay == q) begin
        setbit = sb_int;
        end
        else begin
        setbit = sb_int;
        end
        end
endmodule
```

Timer Testbench Code:

```
`timescale 1ns/1ns
module timer_tb;
 reg enable, clk, reset;
 wire [3:0] setbit, q, delay;
 reg [3:0] delay1, delay2, delay3, delay4;
 reg [1:0] cs;
 timer t1(enable,clk,reset,setbit, cs, delay1, delay2, delay3, delay4, q, delay);
 always
```

```
        #10 clk = ~clk;
 initial begin
        delay1 = 4'd3;
        delay2 = 4'd4;
        delay3 = 4'd2;
        delay4 = 4'd3;
        clk = 1'b0;

        #340 cs = 2'b00;

        reset = 1'b1;
        #2 reset = 1'b0;
        #5 reset = 1'b1;
        #5 enable = 1'b1;

        #340 cs = 2'b01;
        #2 reset = 1'b0;
        #5 reset = 1'b1;
        #340 cs = 2'b10;
        #2 reset = 1'b0;
        #5 reset = 1'b1;
        #340 cs = 2'b11;
        #2 reset = 1'b0;
        #5 reset = 1'b1;
        #340 cs = 2'b00;
        #2 reset = 1'b0;
        #5 reset = 1'b1;

        #340

        #50 enable = 1'b0;

        #30 $finish;
 end
endmodule
```

Top Level Source Code:

```
`timescale 1ns/1ns
module top(enable, clk, reset, car2, car4, state, nextstate, l13, l24, d1, d2, d3, d4, setbit,
q, delay);
 input car2, car4, enable, clk, reset;
 output wire [1:0] l13, l24;
 output wire [1:0] nextstate;
 input wire [1:0] state;
 output wire [3:0] setbit, q, delay;
 input wire [3:0] d1, d2, d3, d4;
 lightfsmj f1(.i(car2|car4),.c(setbit),.s(state),.ns(nextstate), .light13(l13), .light24(l24));
 timer  t1(.enable(enable),  .clk(clk),  .reset(reset),  .setbit(setbit),  .cs(state),  .delay1(d1),
.delay2(d2), .delay3(d3), .delay4(d4), .q(q), .delay(delay));
endmodule
```

Top level Testbench Code:

```
`timescale 1ns/1ns
module testbench;
 reg enable, clk, reset, car2, car4;
 wire [1:0] nextstate, l13, l24;
 wire [3:0] setbit, q, delay;
 reg [3:0] d1, d2, d3, d4;
 reg [1:0] state, oldstate;
 top_netlist   u1(.enable(enable),   .clk(clk),   .reset(reset),   .car2(car2),   .car4(car4),
.state(state), .nextstate(nextstate), .l13(l13), .l24(l24), .d1(d1), .d2(d2), .d3(d3), .d4(d4),
.setbit(setbit), .q(q), .delay(delay));
 always
          #10 clk = ~clk;
 initial begin
          $sdf_annotate("top.sdf",u1,,,"MAXIMUM");
 //initial conditions
          reset = 1'b1;
          #2 reset = 1'b0;
          #10 reset = 1'b1;
          d1 = 4'd3;
          d2 = 4'd11;
          d3 = 4'd10;
          d4 = 4'd5;
          clk = 1'b0;
          car2 = 1'b0;
          car4 = 1'b0;
          #5 enable = 1'b1;
          state = 2'h0;

          //testing condition when car arrives at light 2
          #1300 car2 = 1'b1;
          #30 car2 = 1'b0;

          //testing condition when car arrives at light 4
          #1300 car4 = 1'b1;
          #30 car4 = 1'b0;

          //testing condition when cars are at both lights 2 and 4
          #1300 car2 = 1'b1;
          car4 = 1'b1;
```

```
        #30 car2 = 1'b0;
        car4 = 1'b0;
        #1300 car2 = 1'b1;
        car4 = 1'b1;
        #20 car2 = 1'b0;
        #40 car4 = 1'b0;
        #200 $finish;

 end
 //required external connections
 always@(posedge clk) begin
        oldstate = state;
        if (|setbit) begin
        if(state == 2'b00 && (car2|car4)) begin
        state = nextstate;
        end
        else if (state != 2'b00) begin
        state = nextstate;
        end
        if(oldstate != state) begin
        reset = 1'b0;
        #5 reset = 1'b1;
        end
        else begin
        reset = 1'b1;
        end
        end
  end
endmodule
```

Synthesis Script:

```
        source ~/Synopsys/cshrc/.synopsys_dc.setup
        analyze                    -format                    verilog
        {/net/home/ECE2014_715/ece71504/DVE/finalproject/top.v
        /net/home/ECE2014_715/ece71504/DVE/finalproject/timer.v
        /net/home/ECE2014_715/ece71504/DVE/finalproject/lightfsmj.v}
        elaborate top -architecture verilog -library DEFAULT
        compile -exact_map
        check_design
        write_sdf ~/DVE/finalproject/top.sdf
        write         -hierarchy         -format         verilog         -output
        /net/home/ECE2014_715/ece71504/DVE/finalproject/top_netlist.v
        report_area > ~/DVE/finalproject/area.rpt
        report_power > ~/DVE/finalproject/power.rpt
        report_timing > ~/DVE/finalproject/timing.rpt
```

```
report_cell
```

Netlist:

```
`timescale 1ns/1ns
module lightfsmj ( i, c, s, ns, light13, light24 );
 input [3:0] c;
 input [1:0] s;
 output [1:0] ns;
 output [1:0] light13;
 output [1:0] light24;
 input i;
 wire   \s[1] , n6, n7, n8, n9, n10;
 assign light13[1] = \s[1] ;
 assign \s[1]  = s[1];
 MOAI22D0 U1 ( .A1(light24[1]), .A2(n6), .B1(light13[0]), .B2(c[2]), .ZN(
        ns[1]) );
 AN2D0 U2 ( .A1(c[0]), .A2(s[0]), .Z(n6) );
 OAI222D0 U3 ( .A1(c[2]), .A2(n7), .B1(s[0]), .B2(n8), .C1(c[0]), .C2(n9),
        .ZN(ns[0]) );
 MUX2ND0 U4 ( .I0(n10), .I1(c[1]), .S(\s[1] ), .ZN(n8) );
 AN2D0 U5 ( .A1(i), .A2(c[3]), .Z(n10) );
 CKND0 U6 ( .I(n9), .ZN(light24[0]) );
 CKND2D0 U7 ( .A1(s[0]), .A2(\s[1] ), .ZN(n9) );
 CKND0 U8 ( .I(n7), .ZN(light13[0]) );
 CKND2D0 U9 ( .A1(s[0]), .A2(light24[1]), .ZN(n7) );
 CKND0 U10 ( .I(\s[1] ), .ZN(light24[1]) );
endmodule
module timer ( enable, clk, reset, setbit, cs, delay1, delay2, delay3, delay4,
        q, delay );
 output [3:0] setbit;
 input [1:0] cs;
 input [3:0] delay1;
 input [3:0] delay2;
 input [3:0] delay3;
 input [3:0] delay4;
 output [3:0] q;
 output [3:0] delay;
 input enable, clk, reset;
 wire   N9, N10, N11, N12, N14, N15, N16, N17, n5, n6, n18, n19, n20, n21,
        n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32;
 DFCNQD1 \q_reg[0]  ( .D(N9), .CP(clk), .CDN(reset), .Q(q[0]) );
 DFCNQD1 \q_reg[1]  ( .D(N10), .CP(clk), .CDN(reset), .Q(q[1]) );
 DFCNQD1 \q_reg[2]  ( .D(N11), .CP(clk), .CDN(reset), .Q(q[2]) );
 DFCNQD1 \q_reg[3]  ( .D(N12), .CP(clk), .CDN(reset), .Q(q[3]) );
 DFQD1 \delay_reg[3]  ( .D(N17), .CP(clk), .Q(delay[3]) );
 DFQD1 \delay_reg[2]  ( .D(N16), .CP(clk), .Q(delay[2]) );
 DFQD1 \delay_reg[1]  ( .D(N15), .CP(clk), .Q(delay[1]) );
 DFQD1 \delay_reg[0]  ( .D(N14), .CP(clk), .Q(delay[0]) );
 DFKCNQD1 \sb_int_reg[3]  ( .CN(n6), .D(n5), .CP(clk), .Q(setbit[3]) );
 DFKCNQD1 \sb_int_reg[1]  ( .CN(n6), .D(cs[1]), .CP(clk), .Q(setbit[1]) );
 DFKCNQD1 \sb_int_reg[0]  ( .CN(cs[1]), .D(cs[0]), .CP(clk), .Q(setbit[0]) );
 DFKCNQD1 \sb_int_reg[2]  ( .CN(n5), .D(cs[0]), .CP(clk), .Q(setbit[2]) );
 AO221D0 U3 ( .A1(delay3[3]), .A2(n18), .B1(delay4[3]), .B2(n19), .C(n20),
        .Z(N17) );
 AO22D0 U4 ( .A1(delay2[3]), .A2(n21), .B1(delay1[3]), .B2(n22), .Z(n20) );
 AO221D0 U5 ( .A1(delay3[2]), .A2(n18), .B1(delay4[2]), .B2(n19), .C(n23),
        .Z(N16) );
 AO22D0 U6 ( .A1(delay2[2]), .A2(n21), .B1(delay1[2]), .B2(n22), .Z(n23) );
 AO221D0 U7 ( .A1(delay3[1]), .A2(n18), .B1(delay4[1]), .B2(n19), .C(n24),
        .Z(N15) );
 AO22D0 U8 ( .A1(delay2[1]), .A2(n21), .B1(delay1[1]), .B2(n22), .Z(n24) );
 AO221D0 U9 ( .A1(delay3[0]), .A2(n18), .B1(delay4[0]), .B2(n19), .C(n25),
        .Z(N14) );
 AO22D0 U10 ( .A1(delay2[0]), .A2(n21), .B1(delay1[0]), .B2(n22), .Z(n25) );
 NR2D0 U11 ( .A1(cs[0]), .A2(cs[1]), .ZN(n22) );
 NR2D0 U12 ( .A1(n6), .A2(cs[1]), .ZN(n21) );
 NR2D0 U13 ( .A1(n5), .A2(n6), .ZN(n19) );
 CKND0 U14 ( .I(cs[0]), .ZN(n6) );
 NR2D0 U15 ( .A1(n5), .A2(cs[0]), .ZN(n18) );
 CKND0 U16 ( .I(cs[1]), .ZN(n5) );
 MUX2ND0 U17 ( .I0(n26), .I1(n27), .S(q[3]), .ZN(N12) );
 OA21D0 U18 ( .A1(n28), .A2(q[2]), .B(n29), .Z(n27) );
 IND2D0 U19 ( .A1(n30), .B1(q[2]), .ZN(n26) );
 MUX2ND0 U20 ( .I0(n30), .I1(n29), .S(q[2]), .ZN(N11) );
 OA21D0 U21 ( .A1(q[1]), .A2(n28), .B(n31), .Z(n29) );
 ND3D0 U22 ( .A1(q[0]), .A2(enable), .A3(q[1]), .ZN(n30) );
 MUX2ND0 U23 ( .I0(n32), .I1(n31), .S(q[1]), .ZN(N10) );
 CKND0 U24 ( .I(N9), .ZN(n31) );
 NR2D0 U25 ( .A1(n28), .A2(q[0]), .ZN(N9) );
 CKND0 U26 ( .I(enable), .ZN(n28) );
 CKND2D0 U27 ( .A1(q[0]), .A2(enable), .ZN(n32) );
endmodule
module top_netlist ( enable, clk, reset, car2, car4, state, nextstate, l13, l24, d1,
        d2, d3, d4, setbit, q, delay );
 input [1:0] state;
 output [1:0] nextstate;
 output [1:0] l13;
 output [1:0] l24;
 input [3:0] d1;
 input [3:0] d2;
 input [3:0] d3;
 input [3:0] d4;
 output [3:0] setbit;
 output [3:0] q;
 output [3:0] delay;
 input enable, clk, reset, car2, car4;
 wire   _0_net_;
 lightfsmj f1 ( .i(_0_net_), .c(setbit), .s(state), .ns(nextstate), .light13(
          l13), .light24(l24) );
 timer t1 ( .enable(enable), .clk(clk), .reset(reset), .setbit(setbit), .cs(
          state), .delay1(d1), .delay2(d2), .delay3(d3), .delay4(d4), .q(q),
          .delay(delay) );
 OR2D0 U3 ( .A1(car2), .A2(car4), .Z(_0_net_) );
endmodule
```

Source Files

top_netlist.v	top.v	top.sdf	timers_tb.v	timers.v	timer_tb.v
timer.v	testbench.v	lightfsmtbj.v	lightfsmj.v	final_script.scr	