FSM Design

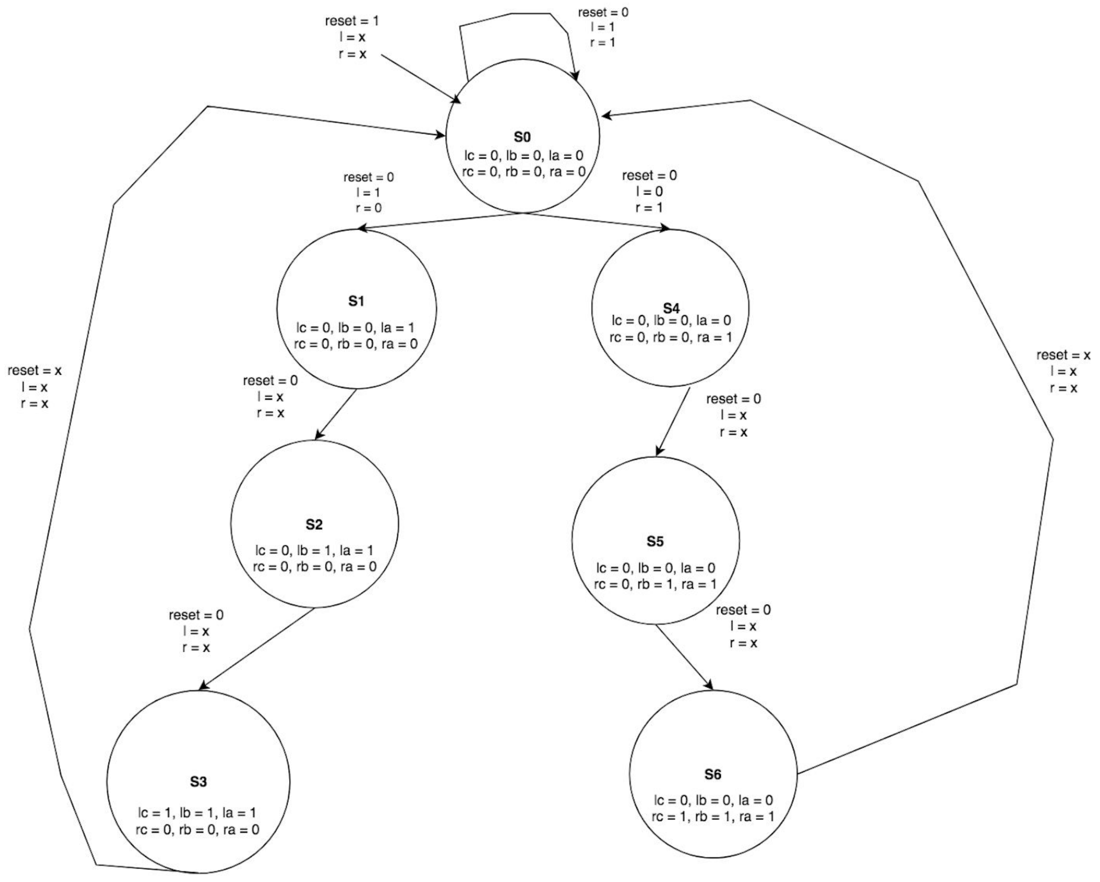
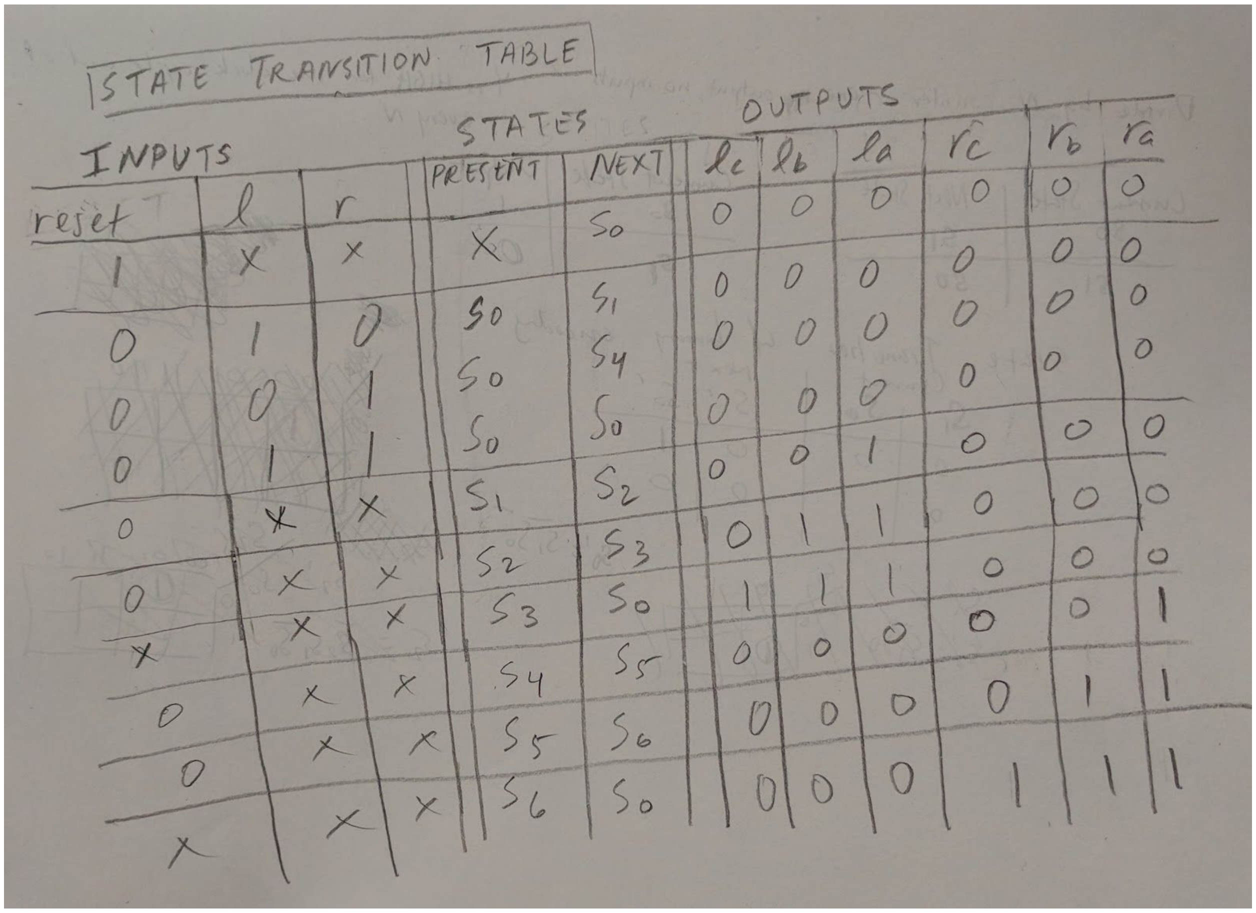


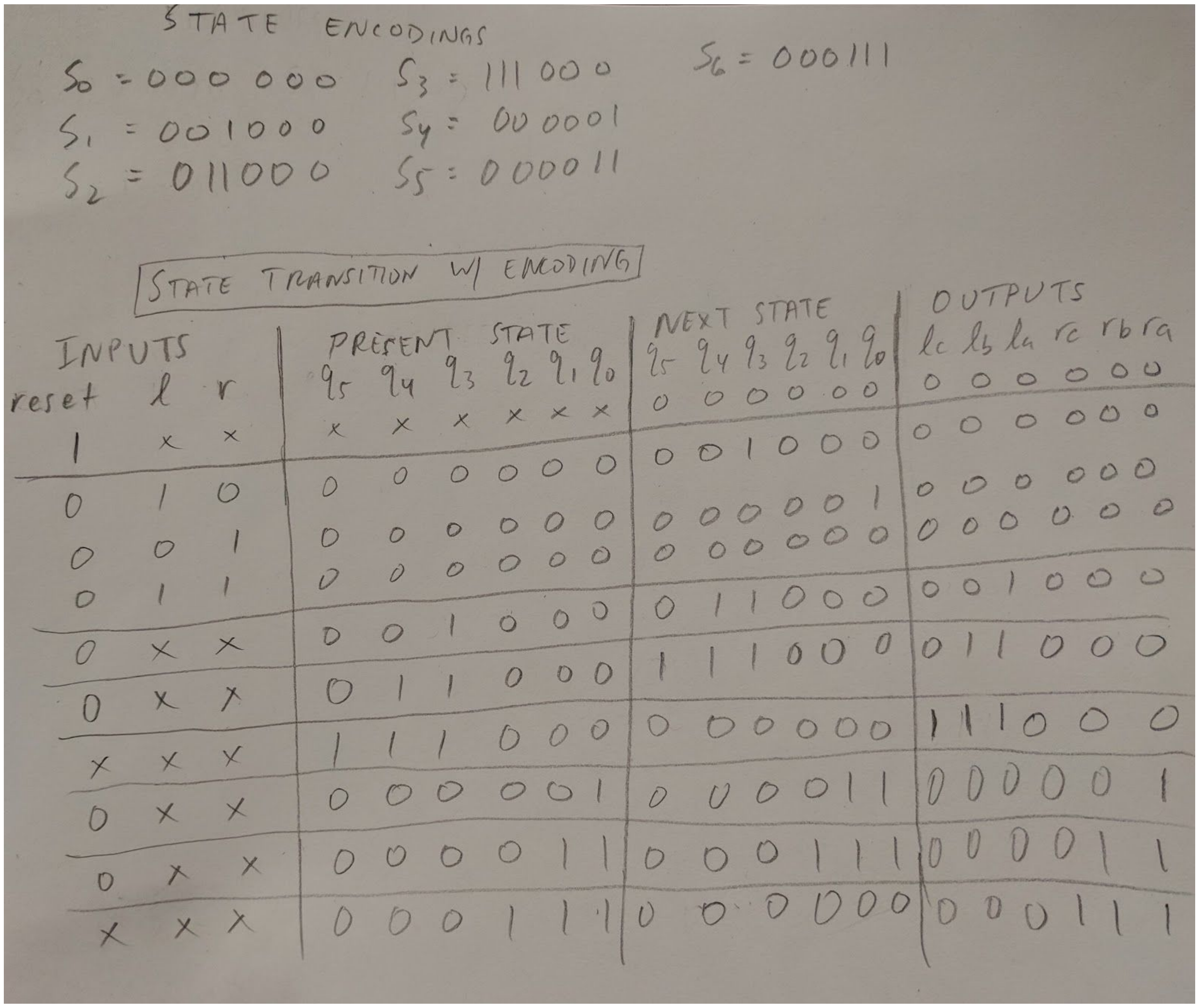
Figure 1: State Diagram



STATE TRANSITION TABLE

INPUTS			STATES		OUTPUTS					
reset	l	r	PRESENT	NEXT	l_c	l_b	l_a	r_c	r_b	r_a
1	X	X	X	S_0	0	0	0	0	0	0
0	1	0	S_0	S_1	0	0	0	0	0	0
0	0	1	S_0	S_4	0	0	0	0	0	0
0	1	1	S_0	S_0	0	0	0	0	0	0
0	X	X	S_1	S_2	0	0	1	0	0	0
0	X	X	S_2	S_3	0	1	1	0	0	0
X	X	X	S_3	S_0	1	1	1	0	0	0
0	X	X	S_4	S_5	0	0	0	0	0	1
0	X	X	S_5	S_6	0	0	0	0	1	1
X	X	X	S_6	S_0	0	0	0	1	1	1

Figure 2: State Transition Table



STATE ENCODINGS

S_0 = 000 000 S_3 = 111 000 S_6 = 000111

S_1 = 001000 S_4 = 00 0001

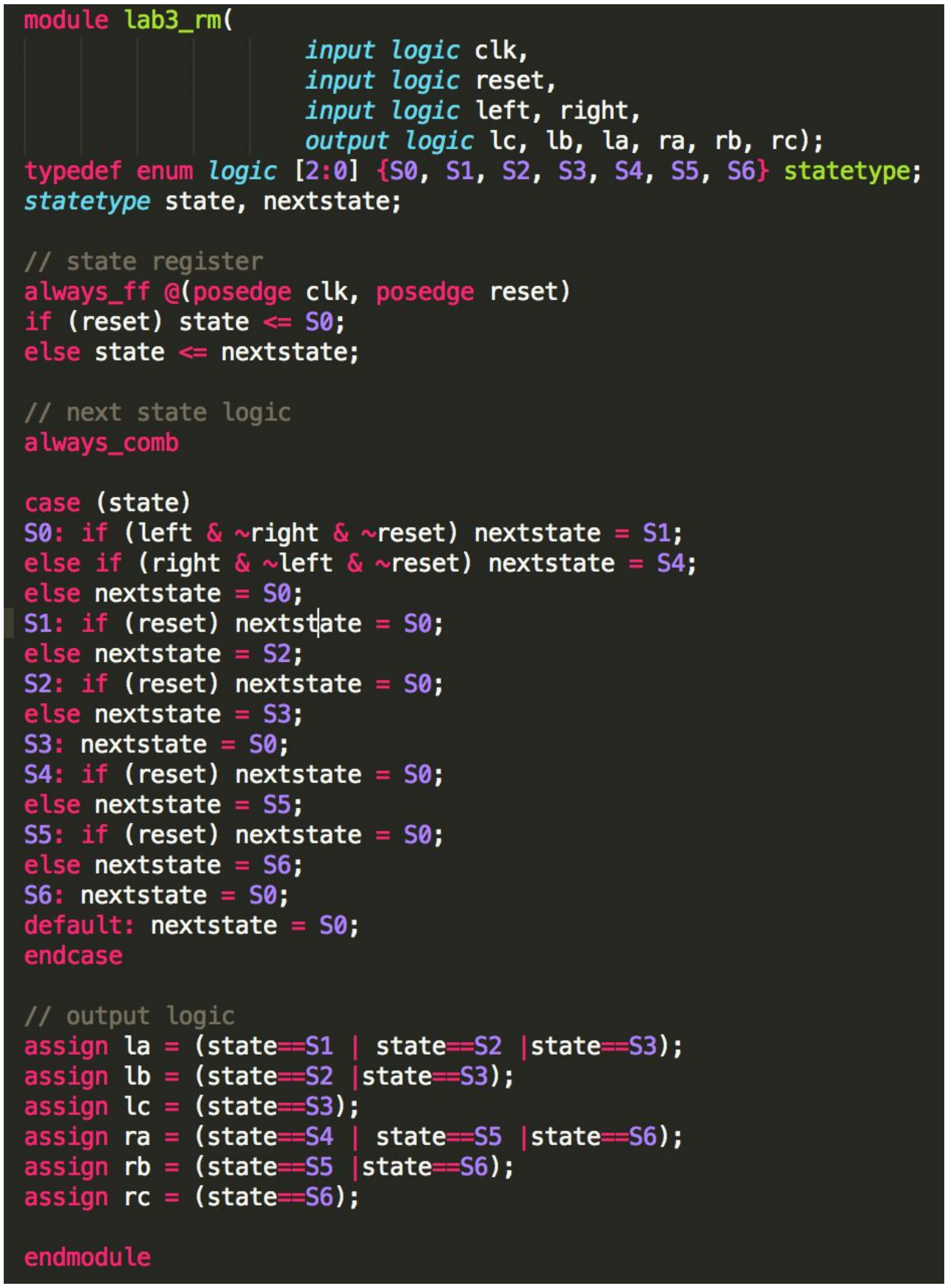
S_2 = 011000 S_5 = 000011

STATE TRANSITION W/ ENCODING

INPUTS			PRESENT STATE						NEXT STATE						OUTPUTS					
reset	l	r	q_5	q_4	q_3	q_2	q_1	q_0	q_5	q_4	q_3	q_2	q_1	q_0	lc	lb	la	rc	rb	ra
1	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	x	x	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0
0	x	x	0	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0
x	x	x	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
0	x	x	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1
0	x	x	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0	0	1	1
x	x	x	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1

Figure 3: State Transition Table with Encodings

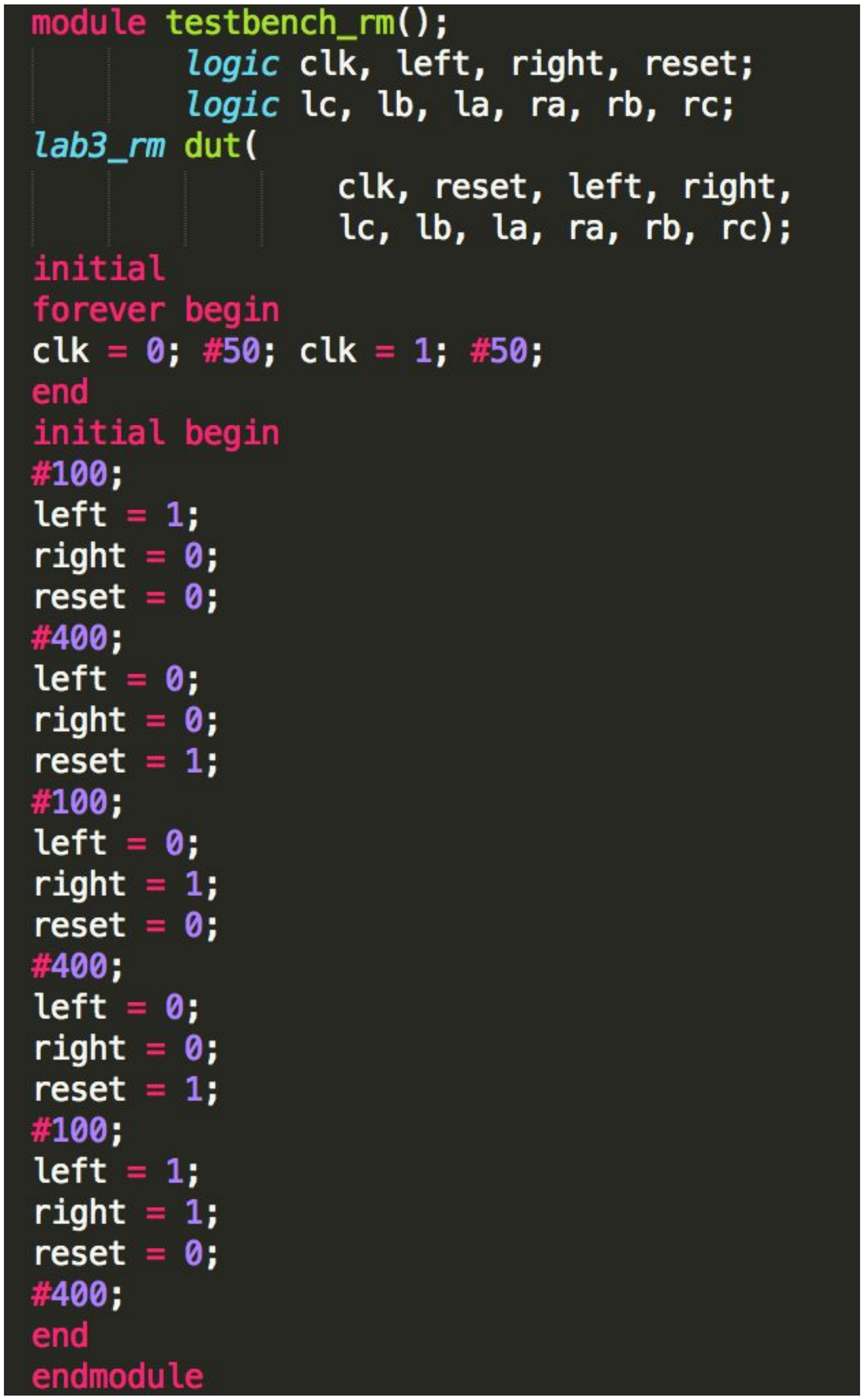
lab_rm.sv Code



```
module lab3_rm(
                    input logic clk,
                    input logic reset,
                    input logic left, right,
                    output logic lc, lb, la, ra, rb, rc);
typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6} statetype;
statetype state, nextstate;

// state register
always_ff @(posedge clk, posedge reset)
if (reset) state <= S0;
else state <= nextstate;

// next state logic
always_comb

case (state)
S0: if (left & ~right & ~reset) nextstate = S1;
else if (right & ~left & ~reset) nextstate = S4;
else nextstate = S0;
S1: if (reset) nextstate = S0;
else nextstate = S2;
S2: if (reset) nextstate = S0;
else nextstate = S3;
S3: nextstate = S0;
S4: if (reset) nextstate = S0;
else nextstate = S5;
S5: if (reset) nextstate = S0;
else nextstate = S6;
S6: nextstate = S0;
default: nextstate = S0;
endcase

// output logic
assign la = (state==S1 | state==S2 |state==S3);
assign lb = (state==S2 |state==S3);
assign lc = (state==S3);
assign ra = (state==S4 | state==S5 |state==S6);
assign rb = (state==S5 |state==S6);
assign rc = (state==S6);

endmodule
```

Figure 4: Design SystemVerilog Code

testbench_rm.sv Code



```
module testbench_rm();
        logic clk, left, right, reset;
        logic lc, lb, la, ra, rb, rc;
lab3_rm dut(
                clk, reset, left, right,
                lc, lb, la, ra, rb, rc);
initial
forever begin
clk = 0; #50; clk = 1; #50;
end
initial begin
#100;
left = 1;
right = 0;
reset = 0;
#400;
left = 0;
right = 0;
reset = 1;
#100;
left = 0;
right = 1;
reset = 0;
#400;
left = 0;
right = 0;
reset = 1;
#100;
left = 1;
right = 1;
reset = 0;
#400;
end
endmodule
```

Figure 5: Testbench SystemVerilog Code

Simulation Waveforms

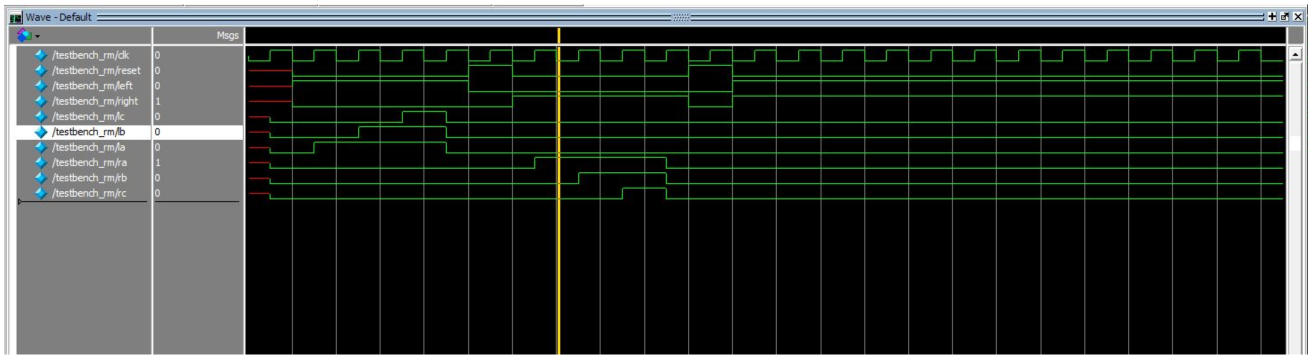


Figure 6: ModelSim Waveforms

Discussion

The design uses 10 I/O pins, which corresponds to the clock, reset, left, right, lc, lb, la, ra, rb, rc input/output signals. This matches my expectations because there are 3 lights on each side of the car, amounting to 6 lights total. These outputs are controlled by two inputs left and right. Furthermore, the flip flop is necessary to hold the states in memory along with the requirement for a clock and reset input pin.