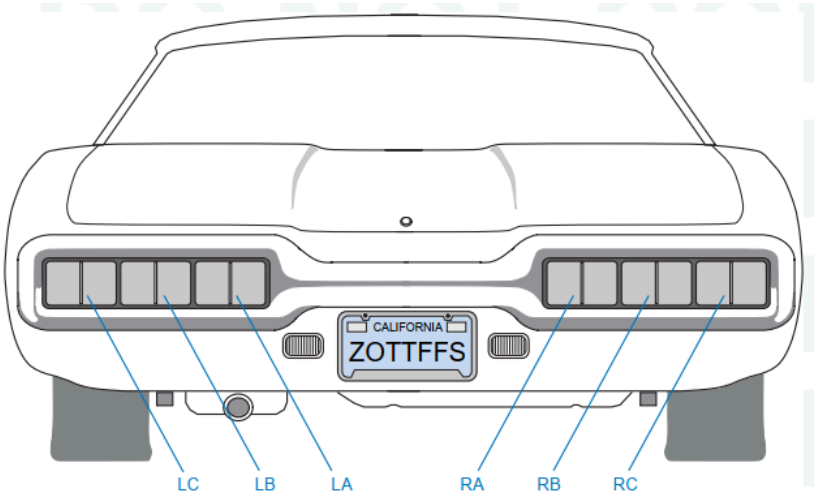
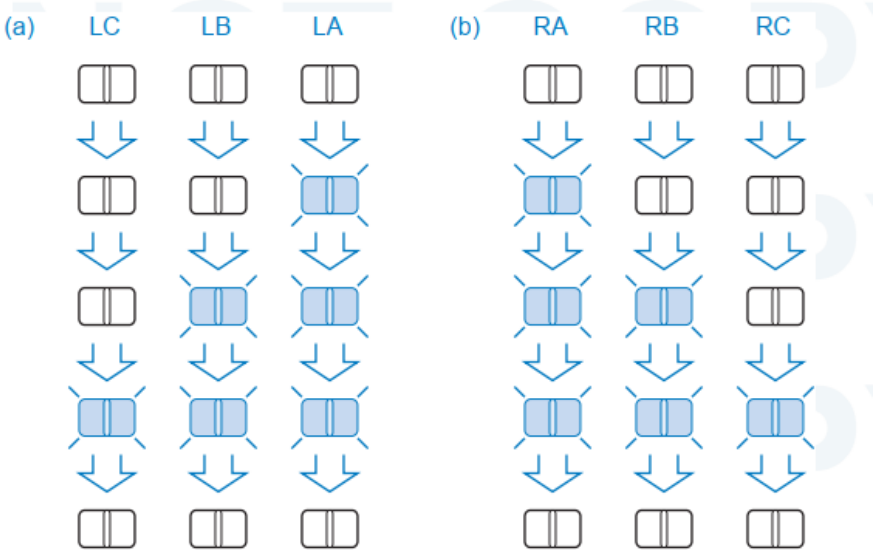
Tail Lights Controller

1: Yêu cầu bài toán



Hình 1

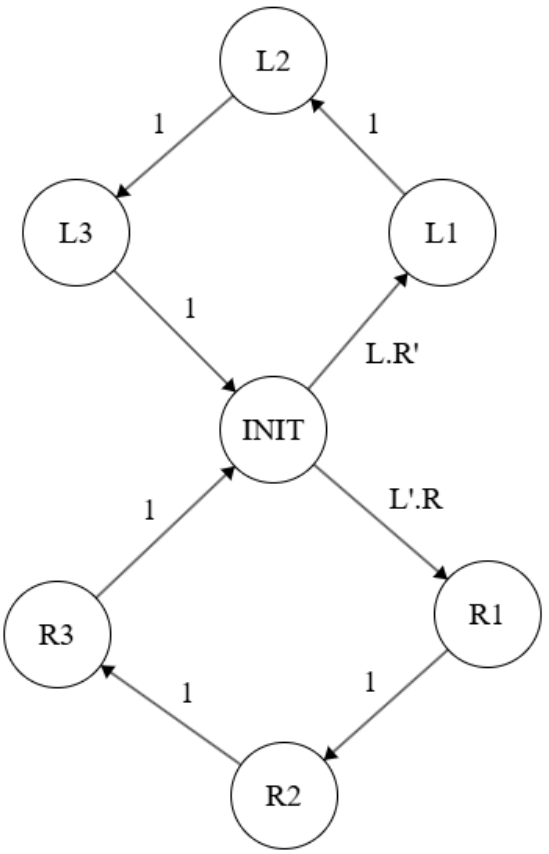
Mục tiêu của bài thực hành này là thiết kế máy trạng thái để điều khiển đèn hậu của xe ô tô. Có 3 đèn ở mỗi bên và lần lượt sáng để chỉ ra hướng rẽ. Hình 1 thể hiện các đèn hậu. Hình 2 minh họa trình tự sáng của các đèn cho rẽ trái (a) và rẽ phải(b)



Hình 2

Máy trạng thái cần có 2 đầu vào là Left và Right để kích hoạt trình tự sáng đèn (flashing sequence) sau khi có tín hiệu. Tại một thời điểm chỉ có một tín hiệu đầu vào. Máy trạng thái có 6 đầu ra là LA, LB, LC, RA, RB, RC. Một khi được kích hoạt, trình tự sáng đèn sẽ diễn ra kể cả khi tín hiệu đầu vào bị hủy. Khi trình tự kết thúc, hệ thống quay lại trạng thái tất cả đèn tắt trong 1 chu kì trước khi một trình tự mới được kích hoạt.

2.Vẽ sơ đồ chuyển trạng thái



Trạng thái	Mô tả
INIT	Trạng thái bình thường, không sử dụng đèn hậu
L1	Xi nhan trái, chỉ đèn LA sáng
L2	Xi nhan trái, đèn LA, LB sáng

L3	Xi nhan trái, đèn LA, LB, LC sáng
R1	Xi nhan phải, chỉ đèn RA sáng
R2	Xi nhan phải, đèn RA, RB sáng
R3	Xi nhan phải, đèn RA, RB, RC sáng

- Từ INIT mạch sẽ chuyển trạng thái nếu nhận được tín hiệu LR' (xi nhan trái) hoặc L'R(xi nhan phải) là đúng (1).
- Khi đã ở trạng thái xi nhan trái hay xi nhan phải ,các trạng thái sẽ chuyển tiếp tuần tự ngay cả khi tín hiệu đầu vào bị hủy (L1->L2->L3->INIT) và (R1->R2->R3->INIT)

3.Thiết lập bảng chuyển đổi trạng thái (state transition table, thể hiện mối liên hệ giữa trạng thái hiện tại và trạng thái kế tiếp) và bảng lối ra (output table, thể hiện mối liên hệ giữa từng trạng thái và lối ra tương ứng)

a, Bảng chuyển đổi trạng thái

Current State	**Inputs**		**Next State**
S	**LEFT - L**	**RIGHT - R**	**S'**
INIT	1	0	L1
L1	X	X	L2
L2	X	X	L3
L3	X	X	INIT
INIT	0	1	R1
R1	X	X	R2
R2	X	X	R3
R3	X	X	INIT

b,Bảng lối ra

State	LC	LB	LA	RC	RB	RA
INIT	0	0	0	0	0	0
L1	0	0	1	0	0	0
L2	0	1	1	0	0	0
L3	1	1	1	0	0	0
R1	0	0	0	0	0	1
R2	0	0	0	0	1	1
R3	0	0	0	1	1	1

4.Xây dựng mạch logic các trạng thái.

Ta sẽ biểu diễn các trạng thái dưới dạng mã nhị phân tương ứng và áp dụng lý thuyết đại số Boolean để biểu diễn trạng thái kế tiếp và các biến đầu ra.

Bảng mã hóa nhị phân

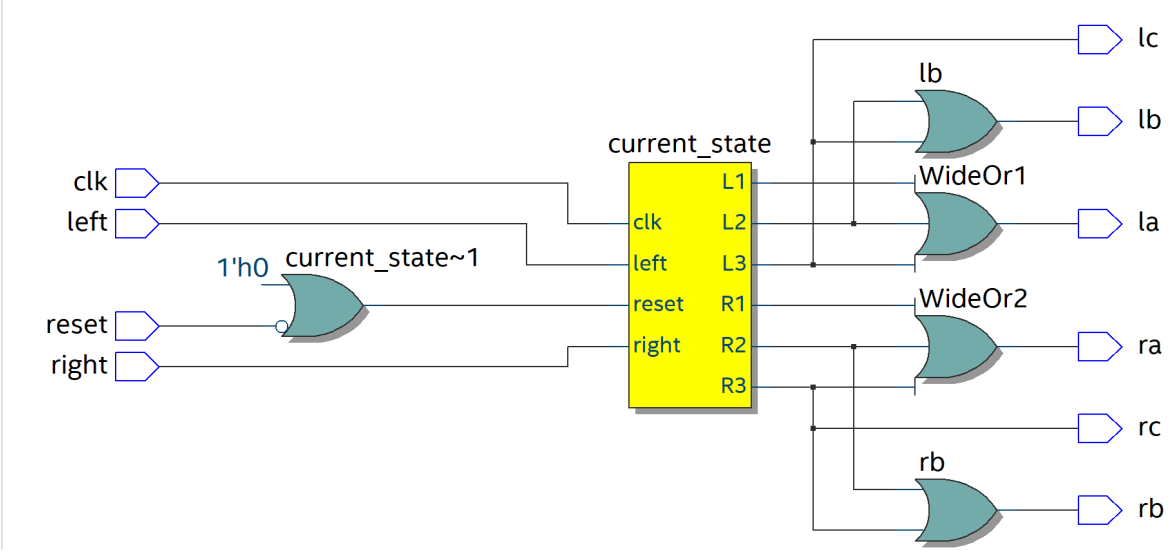
State	$S_{2:0}$
INIT	000
L1	001
L2	010
L3	011
R1	100
R2	101
R3	110

Ta có bảng chân lý:

Current State			Inputs		Next State		
S2	**S1**	**S0**	**L**	**R**	**S′2**	**S′1**	**S′0**
0	0	0	1	0	0	0	1
0	0	1	X	X	0	1	0
0	1	0	X	X	0	1	1
0	1	1	X	X	0	0	0
0	0	0	0	1	1	0	0
1	0	0	X	X	1	0	1
1	0	1	X	X	1	1	0
1	1	0	X	X	0	0	0

Bảng lối ra:

Current State			Outputs					
S2	**S1**	**S0**	**LC**	**LB**	**LA**	**RC**	**RB**	**RA**
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	1	1	0	0	0
0	1	1	1	1	1	0	0	0
b1	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1	1



5:Code HDL và testbench

carFSM.sv

D:/Workspace/carFSM.sv (/carFSM_tb/dut) - Default

```
Ln#
 1  module carFSM(
 2      input logic clk,
 3      input logic reset,
 4      input logic left, right,
 5      output logic la, lb, lc, ra, rb, rc
 6  );
 7      // State encoding
 8      localparam INIT = 3'b000;
 9      localparam L1   = 3'b001;
10      localparam L2   = 3'b010;
11      localparam L3   = 3'b011;
12      localparam R1   = 3'b100;
13      localparam R2   = 3'b101;
14      localparam R3   = 3'b110;
15
16      reg [2:0] current_state, next_state;
17
18      // State register block (State Blocking)
19      always_ff @(posedge clk or negedge reset) begin
20          if (~reset) // Active low reset
21              current_state <= INIT;
22          else
23              current_state <= next_state;
24      end
25
```

```
    // State transition logic
    always_comb begin
        next_state = current_state; // Default to stay in current state
        case (current_state)
            INIT: begin
                if (left)
                    next_state = L1;
                else if (right)
                    next_state = R1;
            end
            L1: next_state = L2;
            L2: next_state = L3;
            L3: next_state = INIT;
            R1: next_state = R2;
            R2: next_state = R3;
            R3: next_state = INIT;
        endcase
    end
    // Output logic
    always_comb begin
        // Default outputs
        {la, lb, lc, ra, rb, rc} = 6'b000000;

        case (current_state)
            L1: {la, lb, lc, ra, rb, rc} = 6'b100000;
            L2: {la, lb, lc, ra, rb, rc} = 6'b110000;
            L3: {la, lb, lc, ra, rb, rc} = 6'b111000;
            R1: {la, lb, lc, ra, rb, rc} = 6'b000100;
            R2: {la, lb, lc, ra, rb, rc} = 6'b000110;
            R3: {la, lb, lc, ra, rb, rc} = 6'b000111;
        endcase
    end
endmodule
```

carFSM_tb.v

```
D:/Workspace/carFSM_tb.sv (/carFSM_tb) - Default *
Ln#
module carFSM_tb;
    logic clk;
    logic reset;
    logic left, right;
    logic la, lb, lc, ra, rb, rc;
    carFSM dut (
        .clk(clk),
        .reset(reset),
        .left(left),
        .right(right),
        .la(la),
        .lb(lb),
        .lc(lc),
        .ra(ra),
        .rb(rb),
        .rc(rc)
    );
    // Clock generation
    always #5 clk = ~clk;
    // Test vectors
    logic [7:0] test_vectors [0:12]; // Adjust size if necessary
    integer i;
    initial begin
        $readmemb("testVectors.txt", test_vectors);
        clk = 0;
        reset = 0;
        #10 reset = 1;
        for (i = 0; i < 13; i = i + 1) begin
            {left, right} = test_vectors[i][7:6];  // Extract inputs
            #10; // Wait for one clock cycle
            if ({la, lb, lc, ra, rb, rc} !== test_vectors[i][5:0]) begin
                $display("Test failed at vector %0d", i);
                $display("Inputs: left=%b, right=%b, Expected: %b, Got: %b",
                        left, right, test_vectors[i][5:0], {la, lb, lc, ra, rb, rc});
            end else begin
                $display("Test passed at vector %0d", i);
                $display("Inputs: left=%b, right=%b, Expected: %b, Got: %b",
                        left, right, test_vectors[i][5:0], {la, lb, lc, ra, rb, rc});
            end
        end
        $stop;
    end
endmodule
```

testVectors.txt

```
// left right la lb lc ra rb rc
```

00_000000

10_100000

10_110000

10_111000

10_000000

10_100000

00_110000

01_111000

01_000000

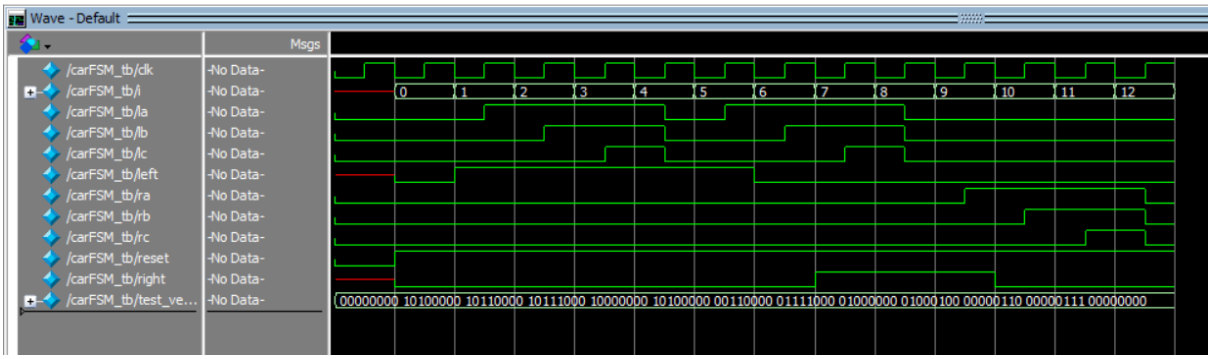
01_000100

00_000110

00_000111

00_000000

6:Kiểm tra đầu ra có phải là đầu ra mong muốn



```
VSIM 100> run
# Test passed at vector 0
# Inputs: left=0, right=0, Expected: 000000, Got: 000000
# Test passed at vector 1
# Inputs: left=1, right=0, Expected: 100000, Got: 100000
# Test passed at vector 2
# Inputs: left=1, right=0, Expected: 110000, Got: 110000
# Test passed at vector 3
# Inputs: left=1, right=0, Expected: 111000, Got: 111000
# Test passed at vector 4
# Inputs: left=1, right=0, Expected: 000000, Got: 000000
# Test passed at vector 5
# Inputs: left=1, right=0, Expected: 100000, Got: 100000
# Test passed at vector 6
# Inputs: left=0, right=0, Expected: 110000, Got: 110000
# Test passed at vector 7
# Inputs: left=0, right=1, Expected: 111000, Got: 111000
# Test passed at vector 8
# Inputs: left=0, right=1, Expected: 000000, Got: 000000
# Test passed at vector 9
# Inputs: left=0, right=1, Expected: 000100, Got: 000100
# Test passed at vector 10
# Inputs: left=0, right=0, Expected: 000110, Got: 000110
# Test passed at vector 11
# Inputs: left=0, right=0, Expected: 000111, Got: 000111
# Test passed at vector 12
# Inputs: left=0, right=0, Expected: 000000, Got: 000000
# ** Note: $stop    : D:/Workspace/carFSM_tb.sv(48)
#    Time: 140 ps  Iteration: 0  Instance: /carFSM_tb
# Break in Module carFSM_tb at D:/Workspace/carFSM_tb.sv line 48
VSIM 101> run

VSIM 101>
```

=> Đầu ra là đầu ra mong muốn.