Báo cáo thực hành buổi 11+12

Họ và tên : Dương Minh Vương

MSV : 22022156

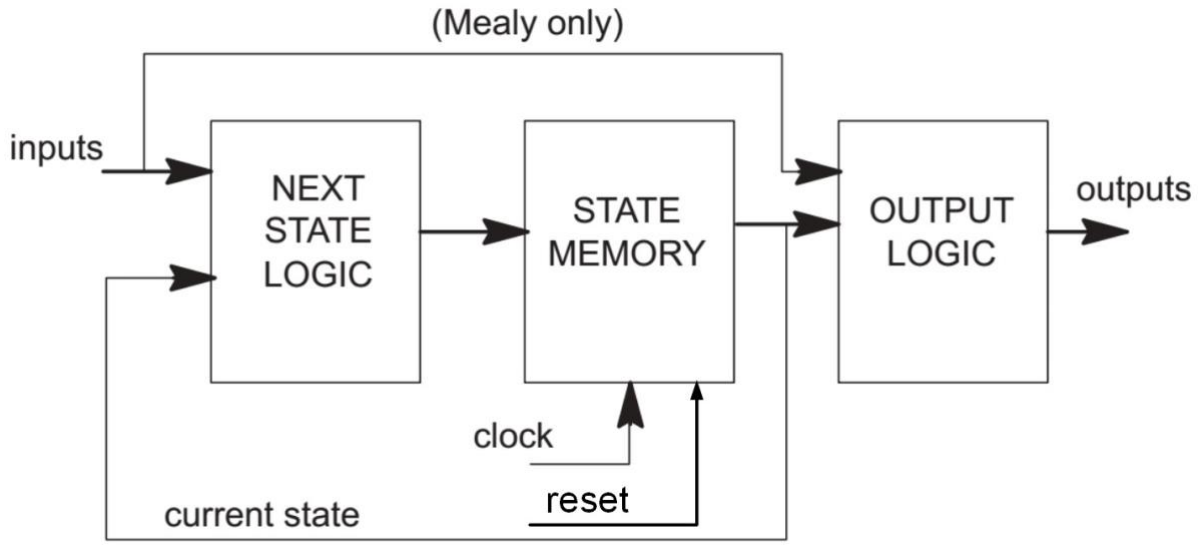
Finite State Machine

I. Định nghĩa về máy trạng thái hữu hạn (finite state machine)

Máy trạng thái hữu hạn, viết tắt là FSM, là một thành phần được sử dụng phổ biến trong thiết kế vi mạch số với ưu điểm là dễ kiểm soát quá trình hoạt động của thiết kế và dễ debug hoạt động của thiết kế.

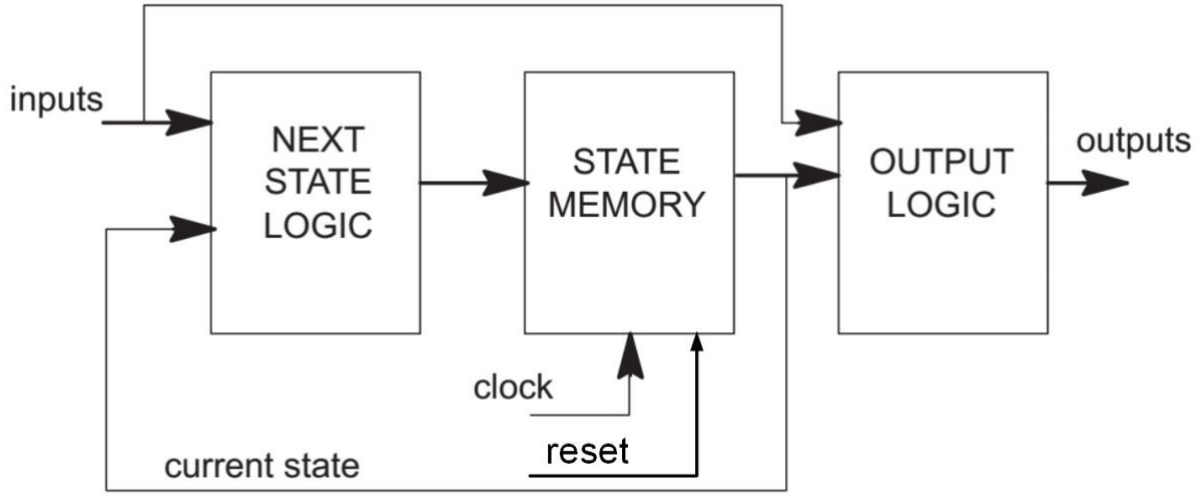
FSM gồm có 3 thành phần cơ bản như sau:

1. Mạch tạo trạng thái kế tiếp (Next state logic) là mạch tổ hợp phụ thuộc vào ngõ vào FSM và giá trị trạng thái hiện tại lấy từ bộ nhớ trạng thái (state memory)
2. Bộ nhớ trạng thái (state memory) là phần tử lưu trạng thái hiện tại của FSM nó có thể là Flip-Flop, Latch, ... lấy ngõ vào từ mạch tạo trạng thái kế tiếp. Bộ nhớ trạng thái thường được sử dụng trong các thiết kế đồng bộ là FF hoạt động theo xung clock. Một tín hiệu reset có thể phải sử dụng để khởi động FSM đến một giá trị ban đầu. Reset không cần sử dụng đối với các FSM luôn hoạt động đúng dù giá trị ban đầu của FF là bao nhiêu.
3. Mạch tạo ngõ ra (output logic) là mạch tổ hợp tạo giá trị ngõ ra tương ứng với trạng thái hiện tại của FSM. Mạch này lấy ngõ vào là giá trị trạng thái hiện tại và có thể tổ hợp thêm ngõ vào của FSM.



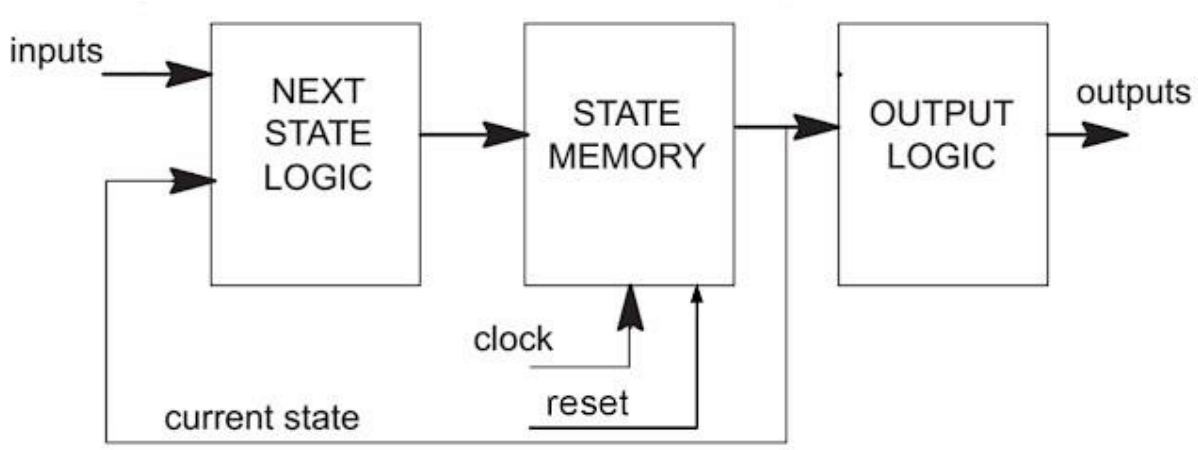
Hình: Mô hình cơ bản của FSM

–Máy trạng thái Mealy (Mealy machine): Máy trạng thái Mealy là một máy trạng thái mà dữ liệu đầu ra được quyết định bởi trạng thái hiện tại và các dữ liệu ngõ vào.



Hình: FSM Mealy

–Máy trạng thái Moore (Moore machine): Máy trạng thái Moore là máy trạng thái mà dữ liệu ngõ ra được quyết định duy nhất bởi trạng thái hiện tại.



Hình: FSM Moore

-Các bước xây dựng một mạch tuần tự

Bước 1: Tìm các biến đầu vào, biến đầu ra và biến trạng thái của các phần tử nhớ.

Bước 2: Xác định số lượng biến trạng thái chính là số lượng flip flop có trong mạch.

Bước 3: Xác định số lượng trạng thái có thể : 2^n, n là số lượng biến trạng thái.

Bước 4: Tìm đầu vào kích thích cho mỗi flip flop.

Bước 5: Dùng bảng đặc trưng và bảng kích thích của mỗi flip flop để xác định trạng thái tiếp theo khi trạng thái hiện tại đã biết.

Bước 6: Chuẩn bị bảng đặc trưng từ phương trình đặc trưng của flip flop và đầu ra. Chuẩn bị bảng chuyển trạng thái từ bảng đặc trưng. Vẽ bảng thể hiện trạng thái hiện tại, trạng thái tiếp theo khi đầu vào là 0, trạng thái tiếp theo khi đầu vào là 1, đầu ra khi đầu vào là 0, đầu ra khi đầu vào là 1.

Bước 7: Gán các kí tự hợp lí cho các biến.

Bước 8: Vẽ sơ đồ trạng thái.

II. Traffic light

Traffic light

Traffic light: Two lane A and B
- **S0:** Wait until there a no people in lane A
- **S1:** Count down before check lane B
- **S2:** Wait until there are no people in lane B
- **S3:** Count down before check A

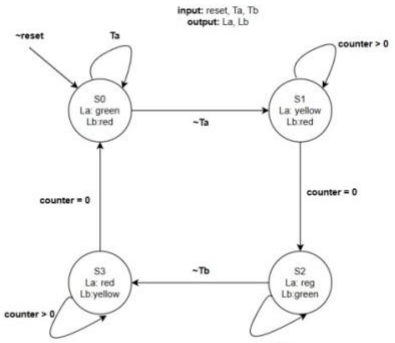


Fig 6. Traffic light FSM

Current State S	Inputs T_A	Inputs T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Fig 7. State transition table

State	Output
S0	LA: green, LB: red
S1	LA: yellow, LB: red
S2	LA: red, LB: green
S3	LA: red, LB: yellow

Output table

1. Các trạng thái (State)

Hệ thống có 4 trạng thái chính:

- S0: Đèn La: xanh, Lb: đỏ.
    - Chờ đến khi làn A không còn người (`~Ta`).
- S1: Đèn La: vàng, Lb: đỏ.
    - Đếm ngược trước khi kiểm tra làn B.
- S2: Đèn La: đỏ, Lb: xanh.
    - Chờ đến khi làn B không còn người (`~Tb`).
- S3: Đèn La: đỏ, Lb: vàng.
    - Đếm ngược trước khi kiểm tra làn A.

2. Cấu trúc FSM

FSM được mô tả bằng hình trạng thái (Fig 6), trong đó:

- Mỗi trạng thái được liên kết với một logic điều kiện để chuyển đổi.
- Các điều kiện bao gồm:
    - `Ta`: Có người trong làn A.
    - `Tb`: Có người trong làn B.
    - Counter > 0: Bộ đếm chưa hết.

Quy trình chuyển đổi trạng thái:

- Từ S0 đến S1: Khi làn A không còn người (`~Ta`).
- Từ S1 đến S2: Khi bộ đếm ở trạng thái S1 bằng 0. ● Từ S2 đến S3: Khi làn B không còn người (`~Tb`).
- Từ S3 đến S0: Khi bộ đếm ở trạng thái S3 bằng 0.

3. Bảng chuyển trạng thái (Fig 7)

Bảng này mô tả cách FSM chuyển đổi giữa các trạng thái:

- Cột Current State S: Trạng thái hiện tại.
- Cột Inputs Ta, Tb: Các tín hiệu đầu vào từ cảm biến.
- Cột Next State S': Trạng thái kế tiếp dựa trên tín hiệu.

Ví dụ:

- Ở trạng thái S0:
    - Nếu `Ta = 0` (không có người ở làn A): Chuyển sang S1.
    - Nếu `Ta = 1` (có người ở làn A): Giữ nguyên S0.

4. Bảng đầu ra (Output table)

Bảng này hiển thị trạng thái của các đèn giao thông:

- La: Đèn giao thông ở làn A (RGB: xanh, vàng, đỏ).
- Lb: Đèn giao thông ở làn B (RGB: xanh, vàng, đỏ).

Ví dụ:

- Ở trạng thái S0:
    - La: Xanh (xe ở làn A được phép đi). ○ Lb: Đỏ (xe ở làn B phải dừng).

5. Ứng dụng

FSM này được sử dụng để:

- Điều khiển đèn giao thông tự động tại ngã tư có hai làn đường.
- Đảm bảo mỗi làn được ưu tiên lần lượt dựa trên trạng thái giao thông.

- Giảm thiểu xung đột giữa các làn và tăng hiệu quả giao thông.

6.Code HDL

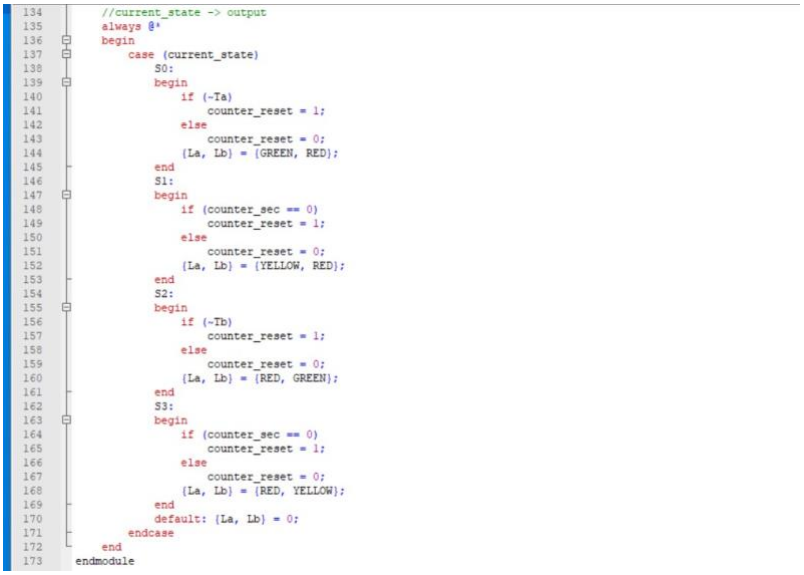
D:/Workspace/traffic_light.sv - Default

```
Ln#
 1  module traffic_light(
 2      input clk,
 3      input reset_n,
 4      input Ta,
 5      input Tb,
 6      output reg [7:0] led_7_segment_1,
 7      output reg [7:0] led_7_segment_2,
 8      output led_7_segment_1_ena,
 9      output led_7_segment_2_ena,
10      output reg [2:0] La,
11      output reg [2:0] Lb
12  );
13      localparam clk_div_period = 12_000_000;
14      reg [24:0] counter_clk;
15      reg [4:0] counter_sec;
16      wire [4:0] counter_sec_wire;
17      reg counter_reset;
18
19      localparam S0 = 0; //La green, Lb red
20      localparam S1 = 1; //La yellow, Lb red
21      localparam S2 = 2; //La red, Lb green
22      localparam S3 = 3; //La red, Lb yellow
23      localparam GREEN = 3'b101; //(rgb: 5)
24      localparam YELLOW = 3'b001;//(rgb: 1)
25      localparam RED = 3'b011;  //(rgb: 3)
26
27      reg [1:0] current_state;
28      reg [1:0] next_state;
29
30      //7 segment digit
31      assign led_7_segment_1_ena = 0;
32      assign led_7_segment_2_ena = 0;
33      assign counter_sec_wire = current_state == S0 || current_state == S2 ? 4 : counter_sec;
```

```
    always @*
    begin
        case (counter_sec_wire)
            0: led_7_segment_1 = 8'h3F;
            1: led_7_segment_1 = 8'h06;
            2: led_7_segment_1 = 8'h5B;
            3: led_7_segment_1 = 8'h4F;
            4: led_7_segment_1 = 8'h66;
            5: led_7_segment_1 = 8'h6D;
            6: led_7_segment_1 = 8'h7D;
            7: led_7_segment_1 = 8'h07;
            8: led_7_segment_1 = 8'h7F;
            9: led_7_segment_1 = 8'h6F;
            default: led_7_segment_1 = 8'hFF;
        endcase
    end

    always @*
    begin
        case (counter_sec_wire)
            0: led_7_segment_2 = 8'h3F;
            1: led_7_segment_2 = 8'h06;
            2: led_7_segment_2 = 8'h5B;
            3: led_7_segment_2 = 8'h4F;
            4: led_7_segment_2 = 8'h66;
            5: led_7_segment_2 = 8'h6D;
            6: led_7_segment_2 = 8'h7D;
            7: led_7_segment_2 = 8'h07;
            8: led_7_segment_2 = 8'h7F;
            9: led_7_segment_2 = 8'h6F;
            default: led_7_segment_2 = 8'hFF;
        endcase
    end
```

```
    always @(posedge clk, negedge reset_n)
    begin
        if (~reset_n)
            counter_clk <= 0;
        else
            if (counter_clk == clk_div_period - 1)
                counter_clk <= 0;
            else
                counter_clk <= counter_clk + 1;
    end

    always @(posedge clk, negedge reset_n)
    begin
        if (~reset_n)
            counter_sec <= 4;
        else
        begin
            if (counter_reset)
                counter_sec <= 4;
            else
                if (counter_clk == clk_div_period - 1)
                    counter_sec <= counter_sec - 1;
                else
                    counter_sec <= counter_sec;
        end
    end
```

```
    //register state
    always @(posedge clk, negedge reset_n)
    begin
        if (~reset_n)
current_state <= S0;
        else
            current_state <= next_state;
    end

    //current_state -> next_state
    always @*
    begin
        case (current_state)
            S0:
                if (~Ta )
                    next_state = S1;
                else
                    next_state = S0;
            S1:
                if (counter_sec == 0)
                    next_state = S2;
                else
                    next_state = S1;
            S2:
                if (~Tb )
                    next_state = S3;
                else
                    next_state = S2;
            S3:
                if (counter_sec == 0)
                    next_state = S0;
                else
                    next_state = S3;
            default:
                next_state = S0;
        endcase
    end
```



```
    //current_state -> output
    always @*
    begin
        case (current_state)
            S0:
            begin
                if (~Ta)
                    counter_reset = 1;
                else
                    counter_reset = 0;
                {La, Lb} = {GREEN, RED};
            end
            S1:
            begin
                if (counter_sec == 0)
                    counter_reset = 1;
                else
                    counter_reset = 0;
                {La, Lb} = {YELLOW, RED};
            end
            S2:
            begin
                if (~Tb)
                    counter_reset = 1;
                else
                    counter_reset = 0;
                {La, Lb} = {RED, GREEN};
            end
            S3:
            begin
                if (counter_sec == 0)
                    counter_reset = 1;
                else
                    counter_reset = 0;
                {La, Lb} = {RED, YELLOW};
            end
            default: {La, Lb} = 0;
        endcase
    end
endmodule
```

Các khối chính trong đoạn mã Khối chia tần số (`clk_div_period`)

```
always @(posedge clk, negedge reset_n)
    begin if (~reset_n) counter_clk <= 0;
    else if (counter_clk == clk_div_period -
        1) counter_clk <= 0;
        else counter_clk <= counter_clk +
            1;
end
```

- Chia xung nhịp thành 1 giây bằng cách sử dụng `counter_clk`.
- Khi đạt đến giá trị `clk_div_period - 1`, bộ đếm được đặt lại.

Khối đếm ngược thời gian

```
always @(posedge clk, negedge reset_n)
    begin if (~reset_n) counter_sec <= 4;
    else begin if
        (counter_reset)
        counter_sec <= 4;
```

```
        else if (counter_clk == clk_div_period - 1)
            counter_sec <= counter_sec - 1;
    end
end
```

- `counter_sec`: Đếm số giây cho mỗi trạng thái đèn.
- Bộ đếm được đặt lại khi tín hiệu `counter_reset` được kích hoạt.

Khối trạng thái (`current_state`)

```
always @(posedge clk, negedge reset_n)
    begin if (~reset_n) current_state <=
    S0;
    else current_state <=
        next_state;
end
```

- `current_state`: Trạng thái hiện tại của hệ thống (S0, S1, S2, S3).
- `next_state`: Trạng thái kế tiếp dựa trên logic chuyển đổi trạng thái.

Chuyển đổi trạng thái (`current_state -> next_state`)

```
always @* begin case (current_state) S0:
            if (~Ta) next_state
                = S1;
            else next_state =
        S0; S1:
```

```
            if (counter_sec == 0)
                next_state = S2;
            else next_state =
        S1; S2:
            if (~Tb) next_state
                = S3;
            else next_state =
        S2; S3:
            if (counter_sec == 0)
                next_state = S0;
            else next_state =
                S3;
        default:
            next_state = S0;
    endcase
end
```

- **S0** → **S1**: Khi `Ta = 0` (xe ở làn A đã hết).
- **S1** → **S2**: Khi thời gian đếm ngược (`counter_sec`) của trạng thái S1 kết thúc.
- **S2** → **S3**: Khi `Tb = 0` (xe ở làn B đã hết).
- **S3** → **S0**: Khi thời gian đếm ngược (`counter_sec`) của trạng thái S3 kết thúc.

Đầu ra đèn giao thông (`current_state -> output`)

```
always @* begin case (current_state) S0:
```

```
            {La, Lb} = {GREEN, RED};
        S1:
            {La, Lb} = {YELLOW, RED};
        S2:
            {La, Lb} = {RED, GREEN};
        S3:
            {La, Lb} = {RED, YELLOW};
        default:
            {La, Lb} = 0;
    endcase
end
```

- Tại mỗi trạng thái, đèn `La` và `Lb` được đặt theo quy định:
  - `S0`: La xanh, Lb đỏ.
  - `S1`: La vàng, Lb đỏ.
  - `S2`: La đỏ, Lb xanh.
  - `S3`: La đỏ, Lb vàng.

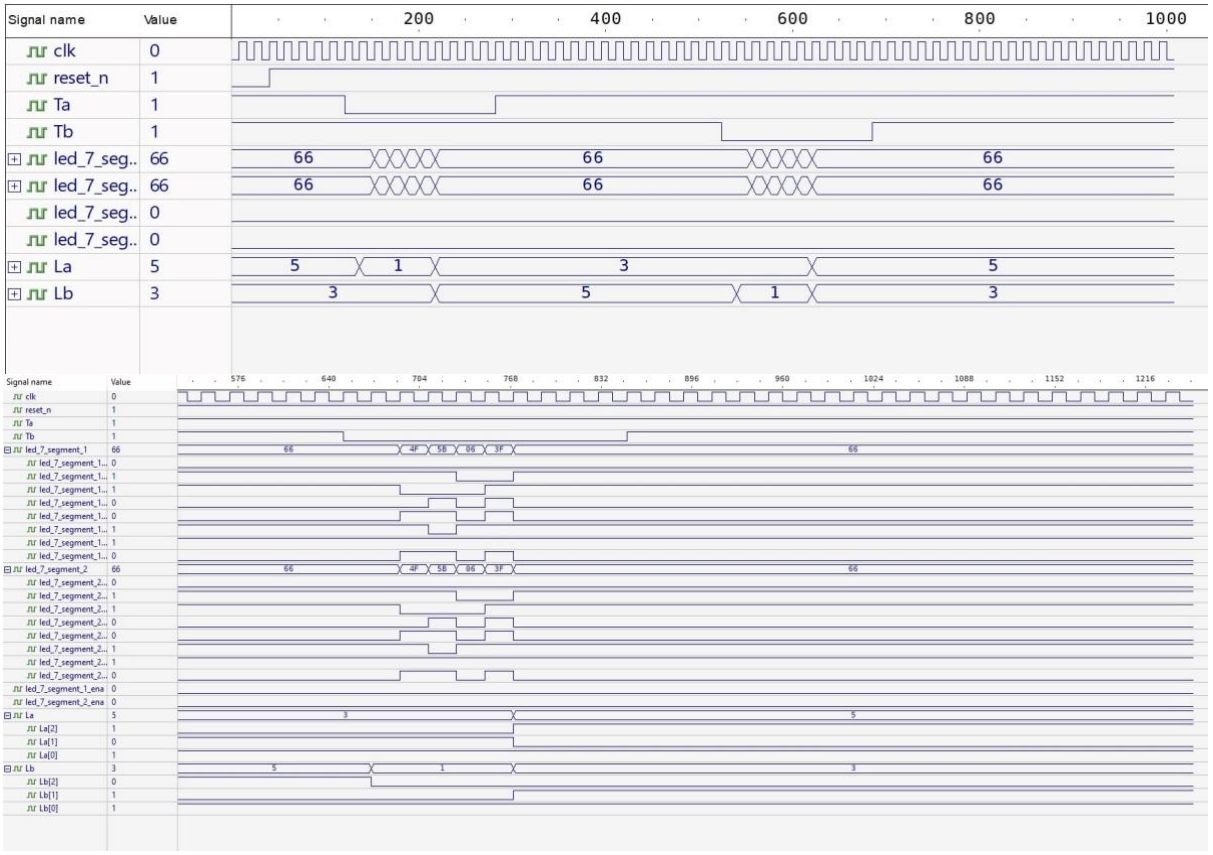
Hiển thị số giây trên LED 7 đoạn

```
always @* begin case (counter_sec_wire)
        0: led_7_segment_1 = 8'h3F;
        1: led_7_segment_1 = 8'h06;
        ...
```

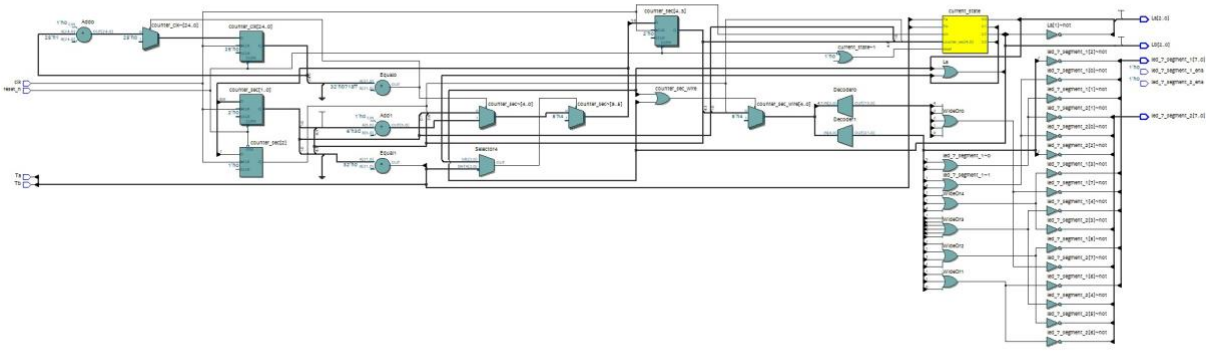
```
            default: led_7_segment_1 = 8'hFF;
        endcase
end
```

- Hiển thị số giây đếm ngược còn lại lên `led_7_segment_1` và `led_7_segment_2`.
- Các giá trị HEX tương ứng với chữ số hiển thị trên LED 7 đoạn.

7. Testbench và mô phỏng mạch trên ModelSim



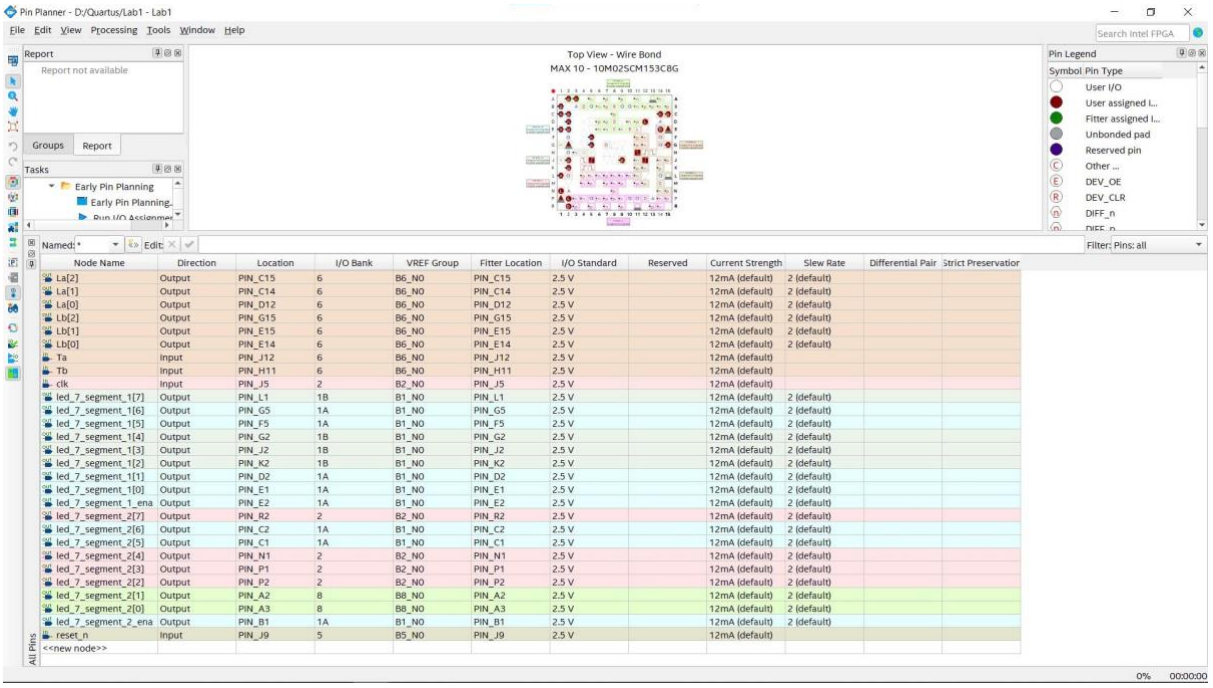
8. Mô phỏng mạch trên Quartus



9.Mô phỏng trên bo mạch FPGA

Video mô phỏng trên KIT:

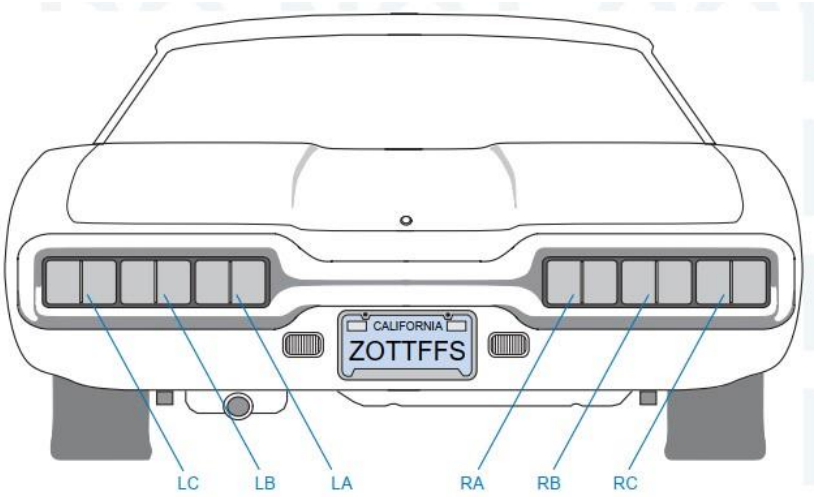
👉 [Video báo cáo thực hành]



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
La[2]	Output	PIN_C15	6	B6_N0	PIN_C15	2.5 V		12mA (default)	2 (default)		
La[1]	Output	PIN_C14	6	B6_N0	PIN_C14	2.5 V		12mA (default)	2 (default)		
La[0]	Output	PIN_D12	6	B6_N0	PIN_D12	2.5 V		12mA (default)	2 (default)		
Lb[2]	Output	PIN_G15	6	B6_N0	PIN_G15	2.5 V		12mA (default)	2 (default)		
Lb[1]	Output	PIN_E15	6	B6_N0	PIN_E15	2.5 V		12mA (default)	2 (default)		
Lb[0]	Output	PIN_E14	6	B6_N0	PIN_E14	2.5 V		12mA (default)	2 (default)		
Ta	Input	PIN_J12	6	B6_N0	PIN_J12	2.5 V		12mA (default)			
Tb	Input	PIN_H11	6	B6_N0	PIN_H11	2.5 V		12mA (default)			
clk	Input	PIN_J5	2	B2_N0	PIN_J5	2.5 V		12mA (default)			
led_7_segment_1[7]	Output	PIN_L1	1B	B1_N0	PIN_L1	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[6]	Output	PIN_G5	1A	B1_N0	PIN_G5	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[5]	Output	PIN_F5	1A	B1_N0	PIN_F5	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[4]	Output	PIN_G2	1B	B1_N0	PIN_G2	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[3]	Output	PIN_J2	1B	B1_N0	PIN_J2	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[2]	Output	PIN_K2	1B	B1_N0	PIN_K2	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[1]	Output	PIN_D2	1A	B1_N0	PIN_D2	2.5 V		12mA (default)	2 (default)		
led_7_segment_1[0]	Output	PIN_E1	1A	B1_N0	PIN_E1	2.5 V		12mA (default)	2 (default)		
led_7_segment_1_ena	Output	PIN_E2	1A	B1_N0	PIN_E2	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[7]	Output	PIN_R2	2	B2_N0	PIN_R2	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[6]	Output	PIN_C2	1A	B1_N0	PIN_C2	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[5]	Output	PIN_C1	1A	B1_N0	PIN_C1	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[4]	Output	PIN_N1	2	B2_N0	PIN_N1	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[3]	Output	PIN_P1	2	B2_N0	PIN_P1	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[2]	Output	PIN_P2	2	B2_N0	PIN_P2	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[1]	Output	PIN_A2	8	B8_N0	PIN_A2	2.5 V		12mA (default)	2 (default)		
led_7_segment_2[0]	Output	PIN_A3	8	B8_N0	PIN_A3	2.5 V		12mA (default)	2 (default)		
led_7_segment_2_ena	Output	PIN_B1	1A	B1_N0	PIN_B1	2.5 V		12mA (default)	2 (default)		
reset_n	Input	PIN_J9	5	B5_N0	PIN_J9	2.5 V		12mA (default)			
<<new node>>											

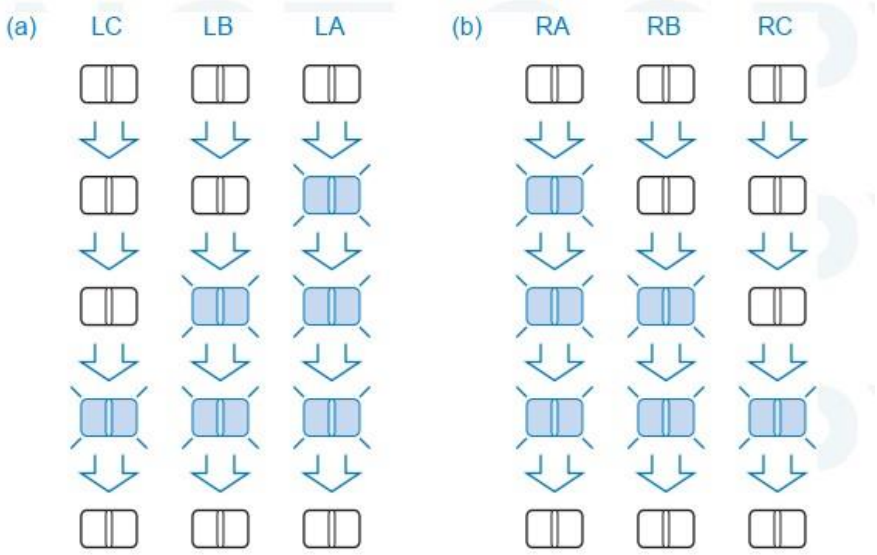
BTVN:Tail Lights Controller

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)

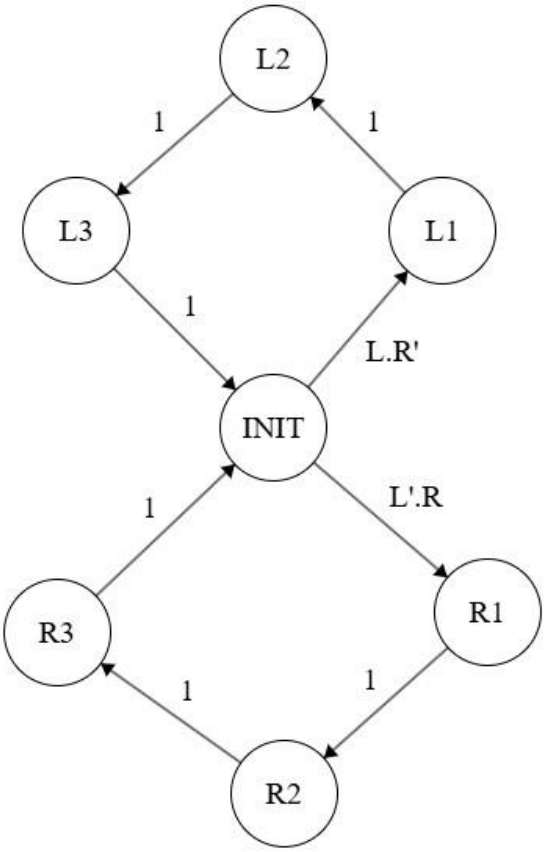


Hình 2

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	**In uts**		**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

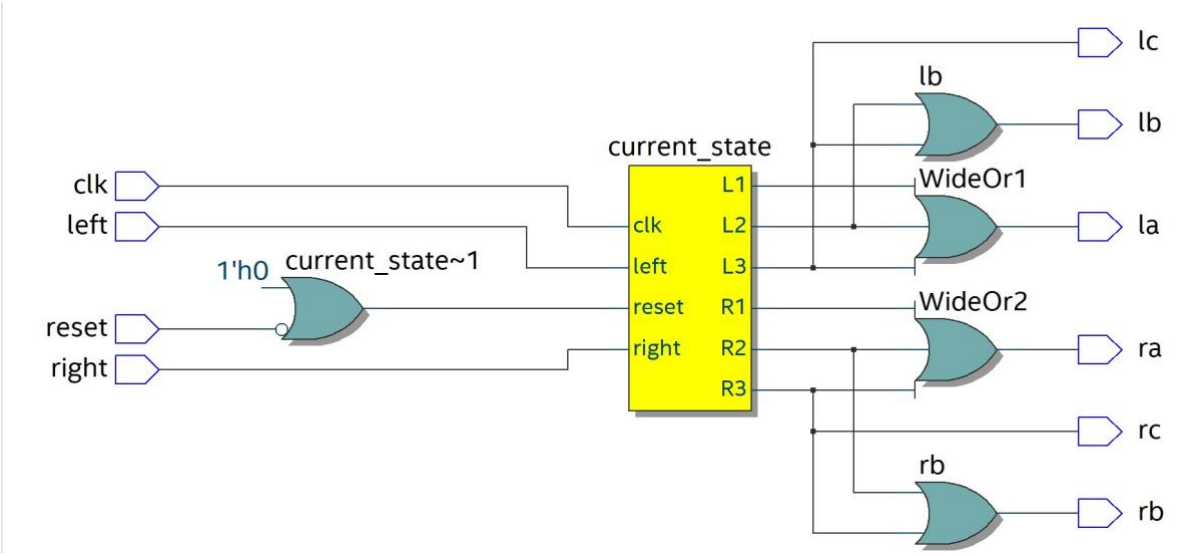
Ta có bảng chân lý:

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

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#
1  module carFSM_tb;
2      logic clk;
3      logic reset;
4      logic left, right;
5      logic la, lb, lc, ra, rb, rc;
6      carFSM dut (
7          .clk(clk),
8          .reset(reset),
9          .left(left),
10         .right(right),
11         .la(la),
12         .lb(lb),
13         .lc(lc),
14         .ra(ra),
15         .rb(rb),
16         .rc(rc)
17     );
18     // Clock generation
19     always #5 clk = ~clk;
20     // Test vectors
21     logic [7:0] test_vectors [0:12]; // Adjust size if necessary
22     integer i;
23     initial begin
24         $readmemb("testVectors.txt", test_vectors);
25         clk = 0;
26         reset = 0;
27         #10 reset = 1;
28         for (i = 0; i < 13; i = i + 1) begin
29             {left, right} = test_vectors[i][7:6];  // Extract inputs
30             #10; // Wait for one clock cycle
31             if ({la, lb, lc, ra, rb, rc} !== test_vectors[i][5:0]) begin
32                 $display("Test failed at vector %0d", i);
33                 $display("Inputs: left=%b, right=%b, Expected: %b, Got: %b",
34                          left, right, test_vectors[i][5:0], {la, lb, lc, ra, rb, rc});
35             end else begin
36                 $display("Test passed at vector %0d", i);
37                 $display("Inputs: left=%b, right=%b, Expected: %b, Got: %b",
38                          left, right, test_vectors[i][5:0], {la, lb, lc, ra, rb, rc});
39             end
40         end
41         $stop;
42     end
43 endmodule
44
45
```

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

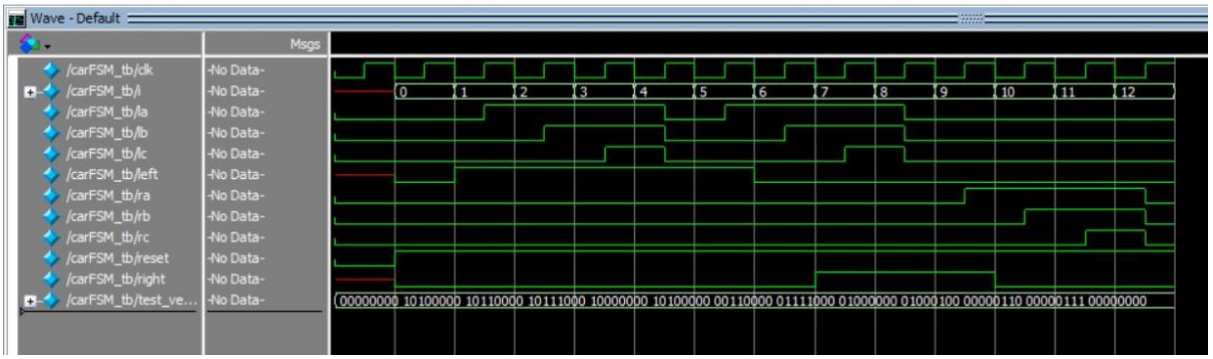
01_000100

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.