# Thunderbird Turn Signal

Submitted by: Piyusha Jahagirdar

In this lab, we designed a Finite State Machine(FSM) in System Verilog for controlling the taillights of Thunderbird.

Figure 1 shows the State Transition Diagram for the FSM

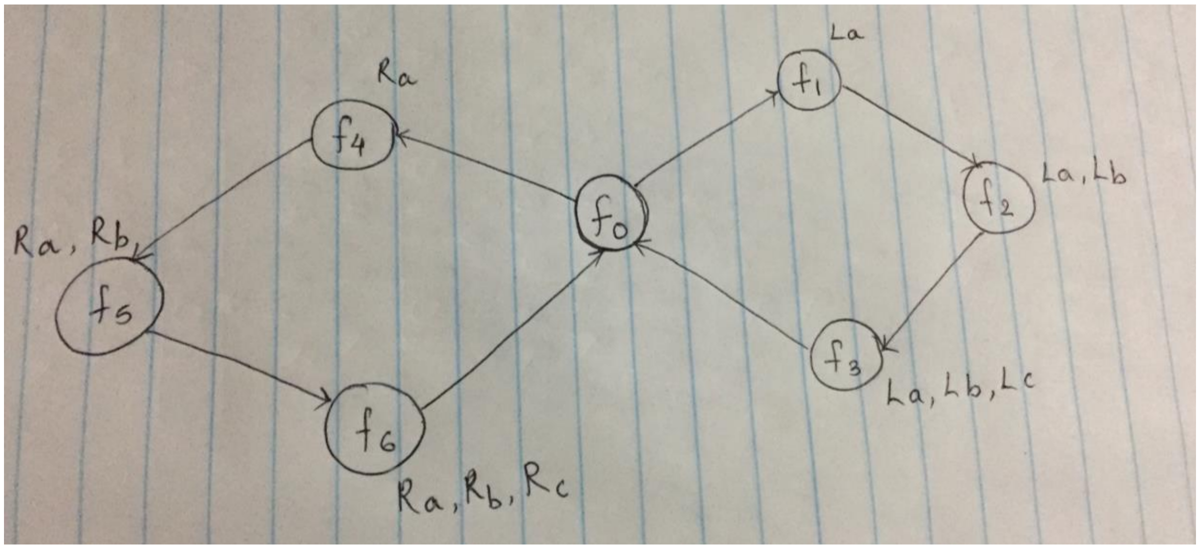

*Figure 1*

Table 1 is the for the FSM

| Current State | F0 | F1 | F2 | F3 | F4 | F5 | F6 | Next State |
|---|---|---|---|---|---|---|---|---|
| F0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | F1 |
| F1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | F2 |
| F2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | F3 |
| F3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | F0 |
| F0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | F4 |
| F4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | F5 |
| F5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | F6 |
| F6 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | F0 |

*Table 1*

**Thunderbird_code.sv**

```
--Code for thunderbird turn signal

module Thunder_bird(
input logic clk,

input logic reset,

input logic left, right,

output logic l1,l2,l3,r1,r2,r3);

enum int unsigned {s0,s1,s2,s3,s4,s5,s6} state,nextstate;

always_comb


begin

case(state)

        s0:

                begin

                l1=1'b0;

                l2=1'b0;

                l3=1'b0;

                r1=1'b0;

                r2=1'b0;

                r3=1'b0;


                if((left == 1'b0 && right == 1'b0)||(left == 1'b1 && right == 1'b1))

                        nextstate <=s0;

                elsif(left == 1'b1 && right == 1'b0)

                        nextstate <=s1;

                else

                        nextstate <=s4;

                end


        s1:

                begin

                l1=1'b1;
```

```
          l2=1'b0;

          l3=1'b0;

          r1=1'b0;

          r2=1'b0;

          r3=1'b0;

          nextstate <= s2

          end


s2:

          begin

          l1=1'b1;

          l2=1'b1;

          l3=1'b0;

          r1=1'b0;

          r2=1'b0;

          r3=1'b0;

          nextstate <= s3

          end


s3:

          begin

          l1=1'b1;

          l2=1'b1;

          l3=1'b1;

          r1=1'b0;

          r2=1'b0;

          r3=1'b0;

          nextstate <= s0

          end
```

```
s4:
        begin
        l1=1'b0;
        l2=1'b0;
        l3=1'b0;
        r1=1'b1;
        r2=1'b0;
        r3=1'b0;
        nextstate <= s5
        end

s5:
        begin
        l1=1'b0;
        l2=1'b0;
        l3=1'b0;
        r1=1'b1;
        r2=1'b1;
        r3=1'b0;
        nextstate <= s6
        end

s6:
        begin
        l1=1'b0;
        l2=1'b0;
        l3=1'b0;
        r1=1'b1;
```

```
                r2=1'b1;
                r3=1'b1;
                nextstate <= s0
                end

        default:
                begin
                l1=1'b0;
                l2=1'b0;
                l3=1'b0;
                r1=1'b0;
                r2=1'b0;
                r3=1'b0;
                nextstate <= s0
                end

        endcase
end
always_ff@(posedge clk)
begin
        if(reset == 1'b1)
        state <= s0;
        else
        state <=nextstate
end
endmodule

--Thunderbird_wrapper.sv
--rename signals to match LEDs and switches on DE2 board
```

```
Module wrapper(input logic[2:0] SW,
        input logic [0:0] KEY,
        output logic [2:0] LEDR,
        output logic [7:5]LEDG);
--use KEY0 for clk
--switches for inputs
--red and green LEDs for output
Thunderbird Thunderbird(.clk(KEY[0], .reset(SW[0]),
.left(SW[2]), .right(SW[1]),
        .la(LEDR[0], .lb(LEDR[1]), .lc(LEDR[2]),
        .ra(LEDG[7], .rb(LEDG[6]), .rc(LEDG[5]));
Endmodule
```

## Testing on the DE2 board

After compiling the code out next goal was to download the code on the DE2 board and test it on the FPGA chip. The pins on the FPGA will correspond to our input and output pins.

We toggled the switches SW[0], SW[1] and SW[2] to different input patterns. The output was displayed on the LED[0], LED[1], LED[2], LED[5], LED[6], LED[7].

## Images of the output