Computer Engineering -CSC 7011

Prof. Stephen Taylor

Arnika Vishwakarma (@01367603)

LAB REPORT- 07

Title: Building a design a finite state machine in system Verilog to control the taillights of a 1965 Ford Thunderbird 1.

Purpose:

- The purpose of thus lab is to design, simulate and implement a finite state machine.
- Downloading the code onto DE2 Board and analyze the output of the given inputs.

Requirements:

- Altera Quartus II 9.1p2 software
- DE2 board

Description:

Altera Quartus II 9.1p2 – It's a wed edition software which allow the user to analyze and synthesize the HDL designs, which enables the user to design the schematic model and simulate the design. Quartus implement the VHDL and Verilog for hardware description, vector waveform simulation and visual editing of logic circuits.

DE2 board: It's a board to test the circuit design on FPGA chip. FPGA will correspond to the input and outputs of the design.

Transition Design:

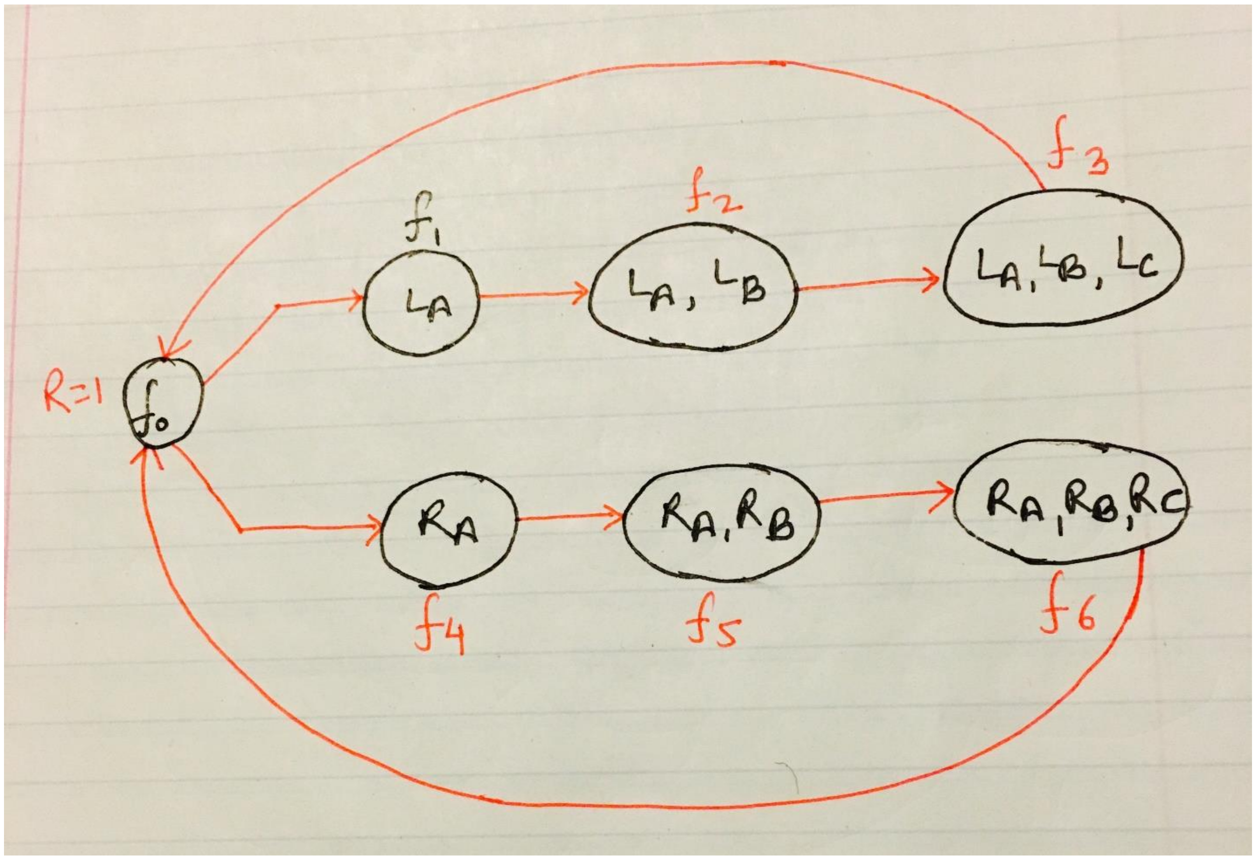

Fig.1 Transition Diagram of thunderbird

Truth Table:

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**

Schematic Design code:

After working for more than 2 hours in lab we developed the schematic design using Altera Quartus II, implemented the design on hardware named DE2 board.

Step 1: Creating a new schematic, click **File/New/VHDL,** after the windows opens click on the Templates icon and choose **VHDL.**

Step 2: Below is the code for Thunderbird:

```
module Thunderbird (input logic clk,
 input logic reset,
 input logic left,right,
 output logic la,lb,lc,  ra,  rb,  rc);
 enum int unsigned {f0, f1, f2, f3, f4, f5, f6} state, next_state;
 always_comb
 begin

 case(state)
            f0:
                        begin
                        la= 1'b0;
                        lb= 1'b0;
                        lc= 1'b0;
                        ra= 1'b0;
                        rb= 1'b0;
                        rc= 1'b0;
                        if ((left == 1'b0 && right == 1'b0)||(left == 1'b1 &&right == 1'b1))
                        next_state <= f0;
                        else if (left==1'b1 && right==1'b0)
                        next_state <= f1;
                        else
                        next_state <= f4;
                        end
            f1:
                        begin
                        la= 1'b1;
                        lb= 1'b0;
                        lc= 1'b0;
```

```
          ra= 1'b0;
          rb= 1'b0;
          rc= 1'b0;
          next_state <= f2;
          end
f2:
          begin
          lb = 1'b1;
          la= 1'b1;
          lc= 1'b0;
          ra= 1'b0;
          rb= 1'b0;
          rc= 1'b0;
          next_state <= f3;
          end
f3:
          begin
          lc = 1'b1;
          la= 1'b1;
          lb= 1'b1;
          ra= 1'b0;
          rb= 1'b0;
          rc= 1'b0;
          next_state <= f0;
          end
f4:
          begin
          ra = 1'b1;
          rb= 1'b0;
```

```
        rc= 1'b0;
        la= 1'b0;
        lb= 1'b0;
        lc= 1'b0;
        next_state <= f5;
        end
f5:
        begin
        rb = 1'b1;
        ra= 1'b1;
        rc= 1'b0;
        la= 1'b0;
        lb= 1'b0;
        lc= 1'b0;
        next_state <= f6;
        end
f6:
        begin
        rc = 1'b1;
        ra= 1'b1;
        rb= 1'b1;
        la= 1'b0;
        lb= 1'b0;
        lc= 1'b0;
        next_state <= f0;
        end
default:
        begin
        la= 1'b0;
```

```
                lb= 1'b0;
                lc= 1'b0;
                ra= 1'b0;
                rb= 1'b0;
                rc= 1'b0;
                next_state <= f0;
                end
 endcase
 end
 always_ff @(posedge clk)
 begin
        if(reset == 1'b1)
        state <= f0;
        else
        state <= next_state;
 end
endmodule
```

Wrapper Code:

```
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
```

DE2 Board Implementation:

After the design has been correctly. Our next goal was to download the circuit onto a DE2 test board to test it on the FPGA chip. The pins on the FPGA will correspond to the inputs and outputs of our design. We need to assign the pins so that we can use switches to control the inputs and LEDs to display the outputs.

Connecting the DE2 board:

- We plugged in the power adapter, attached it to the board
- Connected the DE2 board to the computer USB cable port. The cable must go into the leftmost USB jack on the board labeled BLASTER
- Switch S9 should be in the run position

Testing the DE2 Board:

- Toggle the Switch SW [0], SW [1], SW [2] to different input patterns
- Check the output in LED [0], LED [1], LED [2], LED [5]. LED [6], LED [7].
- For every combination of the inputs the LED will glow and display the output.

Our design worked correctly on the DE2 Board and showed the desired output. Below is the output of the seven segment decoder on DE2 Board.