Lab 3: Finite State Machine

March 4th, 2021

1 Aim

- Learn how to model finite state machines using any hardware description language (VHDL).
- Design a simple circuit that emulates the blinking lights of a Ford Thunderbird.

2 Theory

In this lab, we start with circuits with distinct states and design a finite state machine to control the tail lights of a 1965 Ford Thunderbird. There are three lights on each side that operate in sequence to indicate the direction of a turn. Figure 1 shows the taillights and Figure 2 shows the flashing sequence for (a) left turns and (b) right turns.

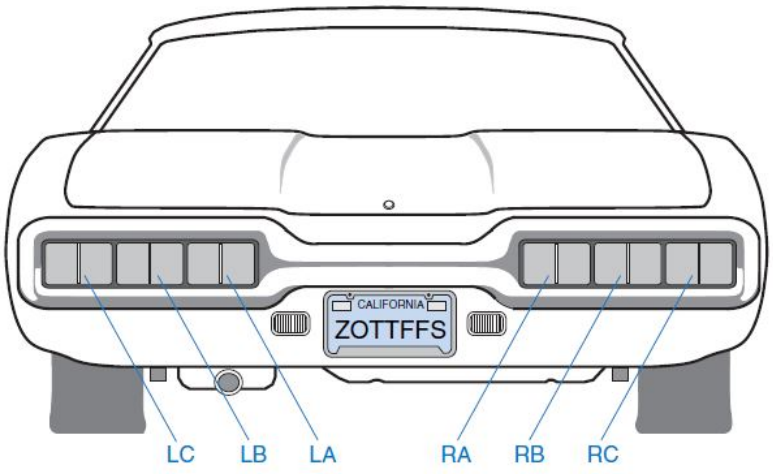

Figure 1: Thunderbird Tail Lights

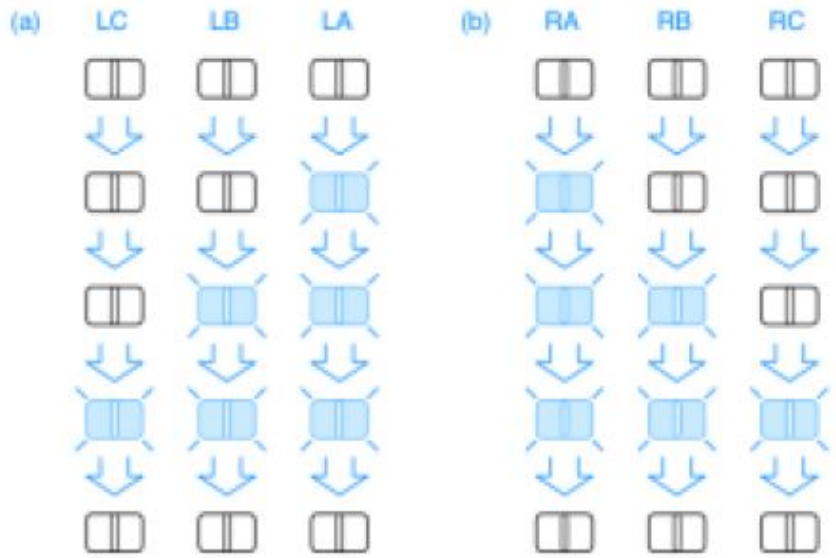

Figure 2: Flashing Sequence (shaded lights are illuminated)

3 General Guidelines for FSM Design

Following are few guidelines on FSM design. Let us start with designing the state transition diagram for this FSM.

A FSM should do the following things:

- Determine the next state from the present state, and the inputs (next state logic).
- Realize the output function based on the present state and the inputs (Mealy machine) or just the present state (Moore machine).
- Advance from the present state to next state when a clock event arrives (state register).

Implementing FSM in HDL:

The syntax of hardware description languages will allow you to write the same FSM in a number of different ways. The essential components of any FSM are the following:

- The state register (where clock moves the next state to the present state)
- The next state logic (where the next state is determined by the present state and inputs)
- The output logic (where the outputs are determined by the present state and inputs)

It is better to separate state register, next state logic and output logic which makes implementing FSM easy in VHDL.

It is also good practice to use a naming style that clearly identifies the signals that are registered. If you want to know how every registered signal is connected, please see Figure 3 as a reference.

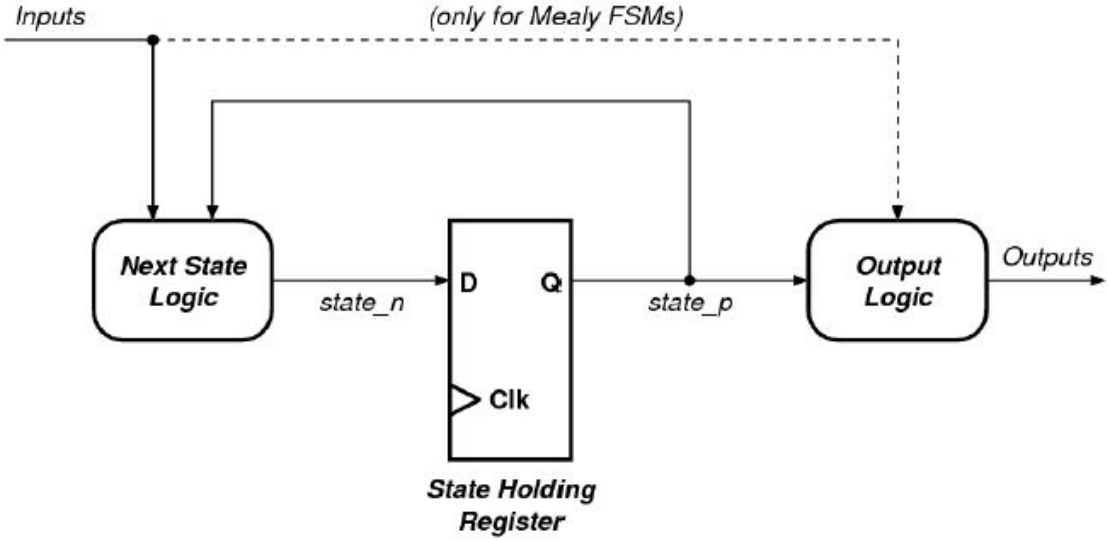

Figure 3: A simple view of an FSM

In this example there are two signals associated with the state. The next state signal is called "state_n" and the present state signal is called "state_p". This is just an example, but it is good practice to use the same "basename" and to add some identifiers as a suffix to differentiate the present and next states. In this way, you are always able to distinguish which signals are registered. Note that in your textbook the examples are not in this fashion, but we still believe it makes life easier.

The important thing is to be consistent in the way you write the code. This allows you to find problems easier, makes code re-use easier, and allows others to understand your code easier.

4 Labwork

Implement the state diagram that emulates the blinking lights of a Ford Thunderbird. (Follow the procedure given below)

- The state machine has two input signals LEFT and RIGHT and six led outputs LC, LB, LA, RA, RB, and RC. It also has an emergency flasher input HAZ which causes all six lights flashing on and off together.

- We assume the existence of a free-running clock signal whose frequency equals the desired flashing rate for the lights. (You can use the 1 Hz clock on the Krypton board as the clock signal).
- We will design a Moore machine, so that the state alone determines which lights are on and which are off.
- A common IDLE state is defined in which all of the lights are off.
- When all the inputs i.e LEFT, RIGHT and HAZ are not asserted, the FSM should enter a state with all lights off.
- When a left turn is requested, the machine goes through three states in which 1, 2, and 3 of the left hand lights are on, and then back to IDLE. Refer Figure 2.
- If LEFT is still down when you return to the IDLE state, the pattern should repeat.
- Right turn works similarly.
- In hazard mode, only two states are required - all lights ON and all lights OFF.
- Lets define a state called LR3. Transition to this state happens for an HAZ input.

Hints:

- Make HAZ input as the top priority input and transition to LR3 state should happen immediately, irrespective of state the FSM is currently present.
- Also we will treat LEFT and RIGHT asserted simultaneously as a hazard request, since the driver is clearly confused and needs help.

4.1 Pre-Lab

1. Write the state diagram implementing the above mentioned specifications.
2. Write the state transition table and transition equations.
3. Write a VHDL code (use the same entity defined below) to implement the corresponding FSM.

4.2 In-Lab

1. Use the state machine viewer in Quartus to verify the FSM inferred by the Quartus compiler.
2. Verify the functionality with the tracefile provided(simulation).
3. Implement it on krypton board and validate the functionality.

Use the following entity:

```
entity  tail_controller is
 port(clk   : in std_logic;
  reset: in std_logic;
  leftt  : in std_logic; --left turn
  rightt : in std_logic;  --right turn
      haz  : in std_logic;   -- hazard
      la: out std_logic; --left leds
      lb    : out std_logic;
      lc    : out std_logic;
      ra    : out std_logic; --right leds
      rb    : out std_logic;
      rc    : out std_logic
  );
end tail_controller
```

5 References

1. John Wakerly, Digital design, 3rd edition

2. A good video about Thunderbird tail light: https://www.youtube.com/watch?v=Qwzxn9ZPW-M&ab_channel=BenandMaria