Thunderbird Turn Signal

Signature and Grading Sheet

Group #:_________ Name(s):____ .

Grading

- Section 4.1(a): FSM state diagram (15 points):______.
  Attach state diagram (can be manually drawn and then photoed)
- Section 4.1(b): ASM chart (15 points):______.
  Attach ASM chart (can be manually drawn and then photoed)
- Section 4.1(c): VHDL code (30 points):______.
  Attach code
- Section 4.2(b) RT-level simulation waveform (20 points):______.
  Attach simulation waveform screen capture.
- Section 4.3(e): post-synthesis simulation waveform (10 points): ______.
  Attach simulation waveform screen capture
- VHDL code file format and comments (10 points):______.

Total points: .

Experiment
Thunderbird Turn Signal

1 Purpose

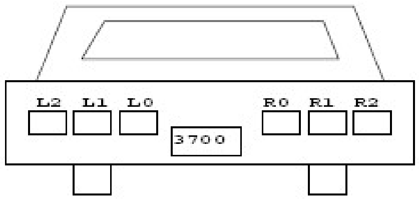
To design an FSM

2 Reading

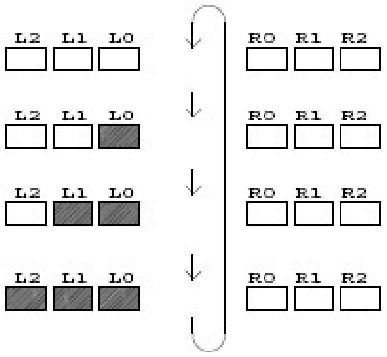
- Chapter 5 of *FPGA Prototyping by VHDL Examples*

3 Project specification

The taillights of a 1965 Ford Thunderbird (a classic car) are shown below:



There are three lights on each side that operate in sequence to indicate the direction of a turn. There are three flashing sequences: left turn, right turn, and hazard. The left-turn sequence is:



The right-turn sequence is similar and represents a "mirror" sequence of the left-turn pattern. In the hazard sequence, the six lights flash on and off alternatively.

A simple FSM can be constructed to control the tail light operation. The input and output are

- input:
  - `clk`: clock signal
  - `reset`: asynchronous reset signal
  - `left`: 1-bit left-turn signal.
  - `right`: 1-bit right-turn signal.
  - `haz`: 1-bit hazard signal.
- output
  - `light`: 6-bit light signal.

The system operates as follows:

- Anytime `haz` is asserted, the FSM enters the hazard sequence immediately. If the FSM currently in the middle of a left- or right-turn sequence, the sequence will be aborted.
- When `haz` is not asserted and `left` is asserted and, the FSM goes through the complete left-turn sequence. This means that the lights should go through a complete left-turn sequence even if `left` is de-asserted sometime in the middle of the sequence or if `right` is asserted in the middle of a sequence. However, the FSM enters the hazard sequence if `haz` is asserted.
- When `haz` is not asserted and `right` is asserted, the FSM goes through the complete right-turn sequence. This means that the lights should go through a complete right-turn sequence even if `right` is de-asserted sometime in the middle of the sequence or if `left` is asserted in the middle of a sequence. However, the FSM enters the hazard sequence if `haz` is asserted.
- We assume that `left` and `right` will never be asserted simultaneously.

The design must be synchronous, or 50% will be deducted.

4 Design Procedures

4.1 FSM

(a) Derive the FSM and draw the complete state diagram.

(b) Convert the state diagram to an ASM chart following the notations used in the lecture.

(c) Design the FSM in VHDL. The entity declaration of this design is

```
entity tbird_fsm is
   port(
         clk, reset: in std_logic;
         left, right, haz: in std_logic;
         light: out std_logic_vector(5 downto 0)
    );
 end tbird_fsm;
```

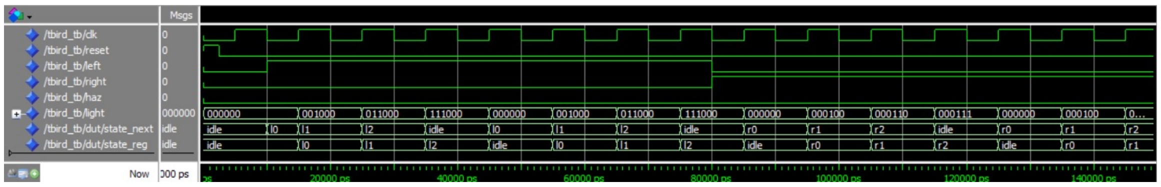
Derive the architecture body.

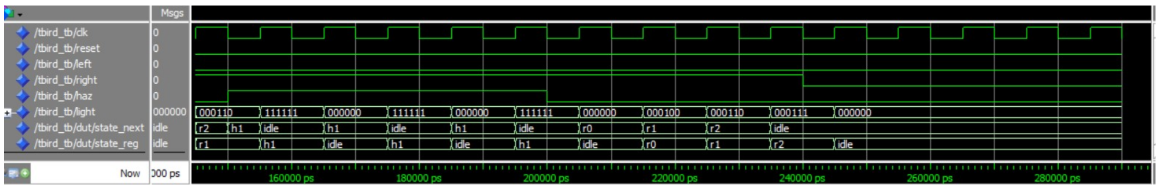
4.2 Simulation

(a) Use the testbench (`tbird_tb_2020.vhd`) to simulate your VHDL code.

(b) Develop a proper "layout" and "format" for the simulated waveform. If needed, use multiple screen captures to make the 6-bit output patterns visible. To get full credits, the input and output signals should be properly arranged and represented in a proper format so that the simulation result can be easily understood.

(c) The expected waveform is:





It is critical to observe the symbolic state transitions in an FSM. To include the internal state signals, expand `tbird_tb` on the left panel, select `dut` instance (i.e., `tbird_fsm` design) and darg the `state_reg` and `state_next` signals to the `Wave` panel. Note that your design may have different state names and even different transitions but the output should be the same.

4.3 Post-synthesis ModelSim simulation

(a) Perform compiling (synthesis/placement and routing) and obtain the `.vho` file.

(b) Follow the format guideline to add a header to the file and verify the "structure" architecture body is generated.

(c) Revise the testbench to use the "structure" architecture for uut.

(d) Perform post-synthesis simulation. Note that the symbolic states are converted to binary representation during synthesis and thus the `state_reg` and `state_next` signals no longer exist.

(e) Do screen capture(s) of the simulated result.

(f) Since the waveform is the same as the RT-level simulation, the snapshot should include the expanded **uut** on the left panel. <u>No point will be given if the **uut** unit is not expanded in the snapshot.</u>