Overall design flow:[1]

Big Picture: In this lab, you will design, write, test, and implement in hardware a finite state machine to simulate the taillights of a 1965 Ford Thunderbird.

Authorized Resources: For this lab, you may work in teams of two (including the prelab). You may seek help from any cadet or instructor, not limited to this course, and reference any publication in its completion. Online resources are acceptable. However, no resources containing solutions to the course homework, labs, exams, quizzes, or in class/out of class exercises are allowed. Your work (and your code) must **always** be your own. Normal documentation of all resources utilized is required.

Due Dates: Prelab – 1700 T22 via Gradescope
Demo – 1700 T24 in person or via video link
Report – 1700 T25 via Gradescope

Background

The 1965 Ford Thunderbird has three lights on each side that operate in sequence to indicate the direction of a turn. Figure 1 shows the tail lights and Figure 2 shows the flashing sequence for (a) left turns and (b) right turns.

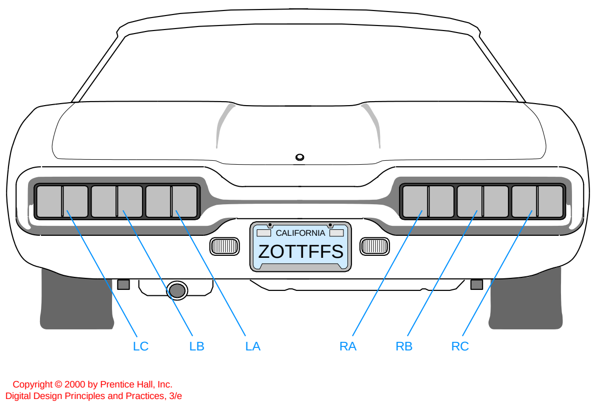

CALIFORNIA
ZOTTFFS
LC
LB
LA
RA
RB
RC
Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

Figure 1 – Thunderbird taillights

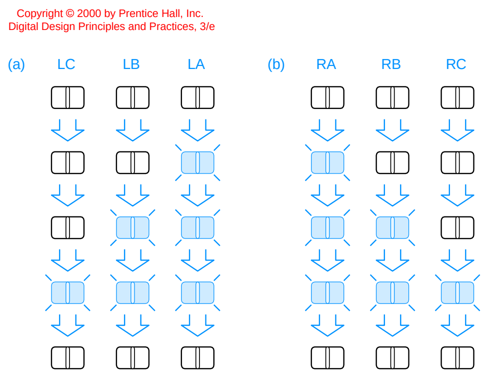

Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e
(a)
LC
LB
LA
(b)
RA
RB
RC

Figure 2 – Flashing sequence for taillights. The lights shaded light blue are illuminated.

[1] Modeled after a lab provided with instructor notes for Digital Design and Computer Architecture, David Money Harris & Sarah L. Harris, 2nd Edition

The objective of this lab is to create a turn signal that will engage the left and right light sets to mimic the Ford Thunderbird. This lab is divided into four parts: design (Prelab), VHDL creation, simulation, and implementation/demonstration. If you follow the steps of FSM design carefully and **ask questions** at the beginning if a part is confusing, you will save yourself a great deal of time.

1. Prelab - FSM Design

Project functionality description

On reset, the FSM should immediately enter a state with all lights off. The outputs for your lights are denoted as LL or RR in your prelab FSM diagram with inputs labeled as a single L or R for the left or right input respectively. When you press left (input L high), you should see LA, then LA and LB, then LA, LB, and LC, then finally all lights off again. This pattern should occur even if you release left during the sequence. If left is still active with the switch held down when you return to the lights off state (L=1), the pattern should repeat. The logic for the right lights is similar. Finally, when both left and right switches are on (L=1 and R=1) your state machine should blink all lights on and off (implementing hazard lights).

Prelab Tasks (20 pts) Due Lesson 22 via Gradescope

You will use a **MOORE FSM** to implement the above functionality. Complete the following tasks to prepare yourself for the lab.

1. Create a new Vivado Project called **Lab3**. This will create your Lab3 folder where you can store your files.
2. Create a state transition diagram using the Lab3_StateTransitionDiagram_Template.pptx found in the Team under Lab 3. Take a screen shot and include it in the Lab3_Prelab_Template.docx found in the Team.
3. **Binary Encoding (TYPE YOUR ANSWERS INTO THE TEMPLATE)**
    a. Complete the encoding table that maps state names to encodings
    b. Complete the state transition table
    c. Complete the output table
4. **One-Hot Encoding (TYPE YOUR ANSWERS INTO THE TEMPLATE)**
    a. Complete the encoding table that maps state names to encodings
    b. Complete the state transition table
    c. Complete the output table
5. Write your next state and output equations using **One-Hot Encoding** **(TYPE YOUR ANSWERS INTO THE TEMPLATE)**

In the remainder of the lab, you will be creating your FSM in VHDL, testing it using a testbench, and then implementing your design on the FPGA prior to demonstrating to the instructor. Since you are designing a synchronous sequential circuit, **we will be making use of a clock signal** to drive the logic. Finally, while not required for the Prelab, you should already be thinking about what **test cases** you will use to verify that your design is correct.

2. Create your Thunderbird FSM VHDL module

Create a new VHDL module file using the course VHDL template. The file should be called `thunderbird_fsm.vhd`. **Include your state encodings in the VHDL comments (i.e. OFF: 10000000)**, and implement your FSM using the following entity interface:

```
entity thunderbird_fsm is
      port(
            i_clk,  i_reset : in  std_logic;
            i_left, i_right : in  std_logic;
            o_lights_L      : out std_logic_vector(2 downto 0);
            o_lights_R      : out std_logic_vector(2 downto 0)
      );
end thunderbird_fsm;
```

3. Verify your FSM module via simulation

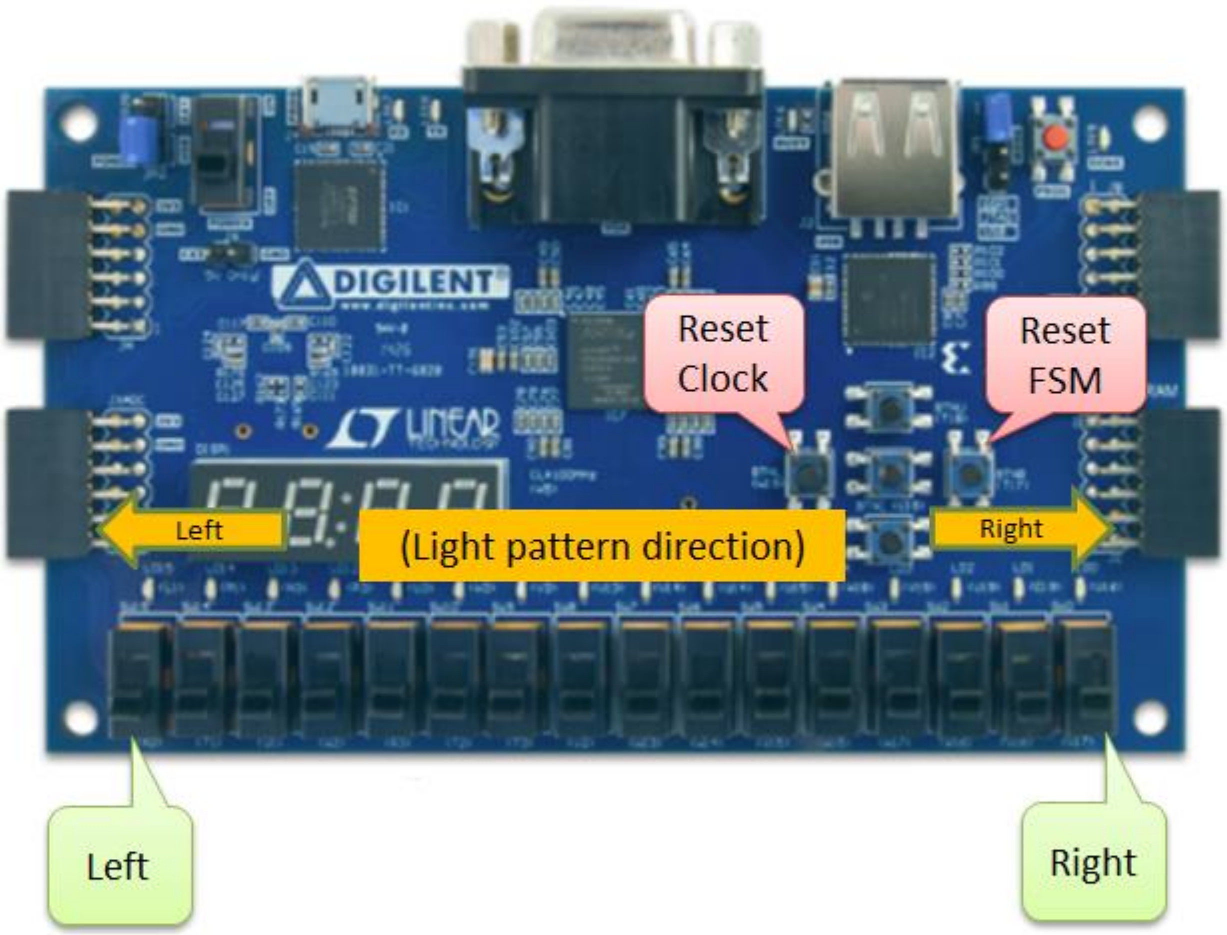
Next, write a testbench that verifies functionality of your FSM. Make sure you carefully consider what test cases you need to run. For instance, **what happens if you change an input in the middle of a blinking pattern? In your test bench comments, *explain* at a high level the various tests you conduct in plain English and why they are important for verifying your design.** It is recommended that you start off with a basic set of test cases first.

- Make sure the first test you run is to apply a **reset**. This should put your FSM in a known good state.
- Do not forget that you must create a clock process to drive the FSM. **Define a *constant* for a clock period of 10ns** and use the constant throughout your testbench file.

For the simulation waveform in your testbench, you should include at a minimum the following signals in order: i_`clk`, i_`reset`, i_`left`, i_`right`, o_`lights_L`, o_`lights_R`, and your current and next state bits. You do not need to change any signal colors, but expand your lights output busses so that the progression of lights can be easily seen. **Have an instructor verify your waveform, and include a picture in your Report.**

4. Create top-level module and implement in hardware

Once you have verified that your FSM works correctly in simulation, you will create a top-level module and implement a hardware interface similar to the one pictured below.



To do this, use structural modeling to connect your FSM signals into a top-level VHDL design file called `top_basys3.vhd`. A top-level VHDL file is provided for you to start with. You will need to ground any unused LEDs, but you can leave unused switches unconnected. In both cases, Vivado will generate warnings, but they may be safely ignored. The hardware interface pictured above is simply an example, so feel free to modify your design as desired.

Using the Basys3 clock

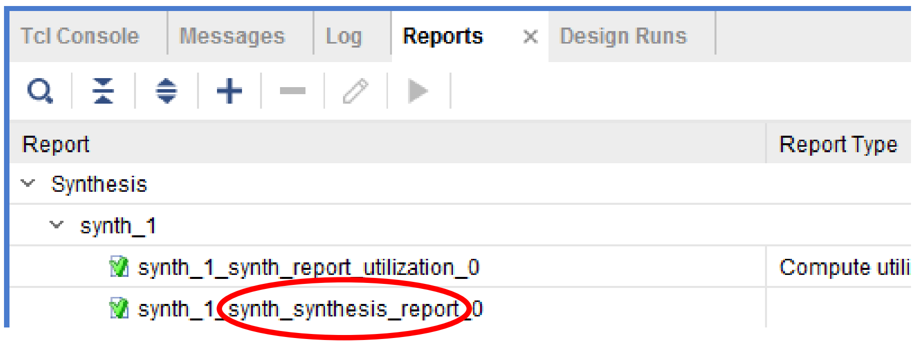
Note, the Basys3 board provides a built-in 100-MHz clock. This is a bit too fast for our application, so you are also provided a VHDL module (`clock_divider.vhd`) that will take in the fast 100 MHz clock, and produce a slower clock that will make your `thunderbird_fsm` transitions visible to the human eye. This is the same clock_divider you used in ICE4, though we are using it to obtain a different clock speed. Your top-level design must incorporate a structural connection of the `thunderbird_fsm` and `clock_divider` to produce a correct design. Hint: you will need to declare a temporary signal to "wire" the connection between the clock divider's output clock port and the FSM's input clock port. An important note, you are not using the `clock_divider` as in input to the top_basys, but just to designate a speed for the `thunderbird_fsm` slow enough to allow human observation of the inputs.

To help you understand how the clock_divider module works, a test bench has been provided. Note how the clock is instantiated. The module uses a generic natural constant called k_DIV to allow you to dynamically set how much you want to slow the clock down. This is done at instantiation using a generic map statement similar to port mapping. For your top level VHDL file, you can copy much of the clock_divider testbench code, but **you need to determine the correct k_DIV amount to slow the clock from 100MHz down to 4Hz.** Reading the VHDL comments and viewing the simulation waveform should help you understand the appropriate value you need. There are multiple ways to implement a clock divider function, but this will suffice for our needs.

Before you write the architecture, neatly create a diagram (draw.io, powerpoint, etc. – do not hand draw it) showing your top-level entity and its internal architecture. **Include this drawing in your Report**. Once you have completed your top-level VHDL design, include a copy of its **RTL schematic in your Report**. Note any discrepancies from your original drawing.

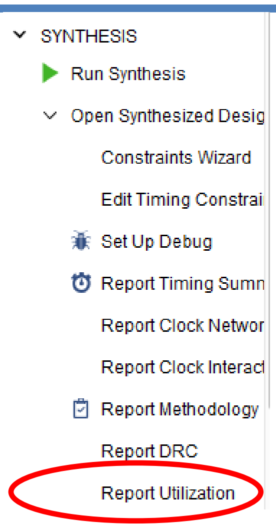
Synthesis Analysis

Now you need to determine how *efficient* your design is. First, synthesize your design. Once synthesis is complete, view the synthesis report by opening the synthesized design and looking under the Reports tab and double-clicking on it:

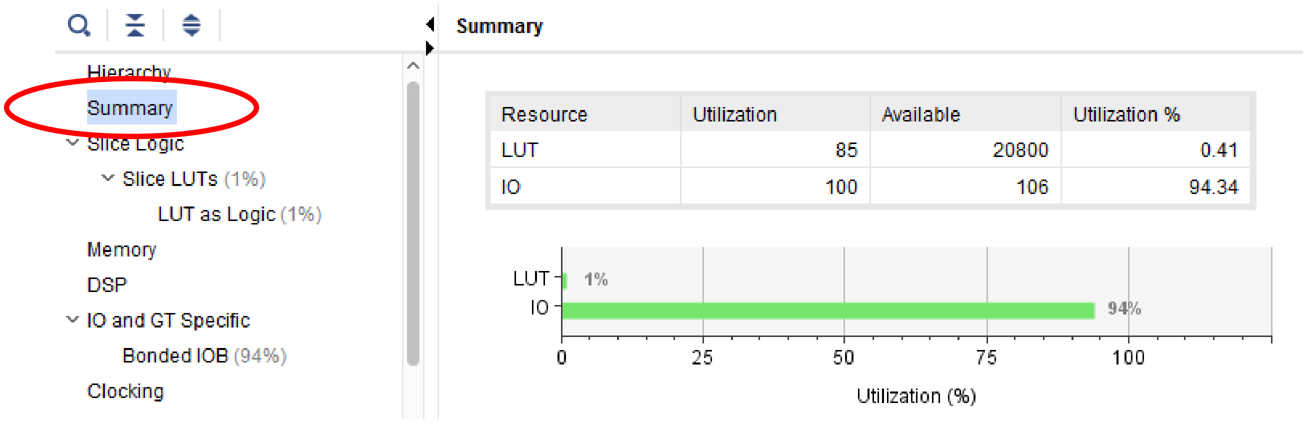


Scroll down or search the report until you find the "**Hierarchical RTL Component report**" section. This should tell you which modules you are using and what the major building blocks contained within them are.

Next, click "Report Utilization" under "Open Synthesized Design" in the Flow Navigator:



Choosing the default name for the report is fine. The utilization report should then automatically open a tab in the lower sub window of Vivado. Click on the Summary to view a table and graph similar to the one below (this picture is for a different VHDL project).



Resource	Utilization	Available	Utilization %
LUT	85	20800	0.41
IO	100	106	94.34

Implement in hardware

Uncomment or modify the required lines in a **Basys3_Master.xdc** file to connect all of the LEDs, switches, and buttons that you are using to your top level design. You will also need to uncomment the lines near the top of the file (see below) to connect the 100 MHz clock.

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
	set_property IOSTANDARD LVCMOS33 [get_ports clk]
	create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

Download your design onto the Basys3 board and test it in hardware. Provide the working product to an instructor for hardware demonstration. **Your simulation must have been verified *first***!

Lab 3 Deliverables

Table 1 below shows how the points will be distributed for this lab. Details for how each item will be scored follows.

Table 1: Point distribution for Lab 3

Item	Out of
Prelab	20
Simulation results waveform	15
Hardware demo	25
VHDL, XDC, and BIT files	15
Git Usage/Repo Structure	5
Report	20
Total	**100**

(20 pts) Prelab

- Use good state names that adequately describe the state (e.g., "OFF")
- State transition diagram needs a key to specify the inputs and outputs. I will assume input order is LR.
- You should specify your encoding in a table.
- Encoding table, state transition, and output tables provided
- Your prelab should always be updated for your final submission
- Documentation statment

(15 pts) Simulation Results Waveform (Submitted within Report)

- Begins with a reset test to start the FSM in a known state
- Clearly shows well thought out test cases with correct results
- Includes all required signals (i_clk, i_reset, i_left, i_right, o_lights_L, o_lights_R, and your current and next state bits).
- Lights output busses expanded to easily see progression of lights
- Does the simulation match your expected results? Show that you actually used the simulation to make sure your design was correct before moving to hardware.
- Ensure the simulation results waveforms have visible values for inputs and outputs.

(25 pts) Hardware Demo (1700 T24 in person or via Teams)

- Simulation results waveform provided prior to hardware demo
- Using left and right switches generates correct LED patterns with **changes occurring at a rate of 4 Hz**
- Holding the clock reset freezes the light patterns
- Pressing the FSM reset *immediately* (i.e. asynchronously) resets the light patterns
- Demo can be performed live with an instructor OR via a video link (Streams Link preferred). For complete proof of correct functionality **you must show good test cases for full credit!**

(15 pts) VHDL, XDC, and BIT files

- thunderbird_fsm.vhd, thunderbird_fsm_tb.vhd, and top_basys3.vhd included in **code** folder
    - Fill out headers as appropriate (include your state encodings in the description)
    - Ensure you update header with your name
    - Remove extraneous code and comments
    - In your testbench file, explain at a high level what test cases your test process performs and why they are needed
    - Include comments throughout code to explain what you are doing.
- Basys3_Master.xdc file included in **code** folder
- Bitstream (.bit) file used for hardware demo in repo (likely at `Lab3\Lab3.runs\impl_1\top_basys3.bit`)
- 15 points will be determined by the quality of your files according to the following scale

Rating	Points	Quality
Excellent	13-15	All required files are included in the "Code" folder of the repository. Each file includes an updated header with accurate filename, author, creation date, description, and documentation. Each entity has a name that matches the file name. Entity inputs and outputs match provided/designed schematics. Variables have descriptive names that meet the naming conventions. Code is commented throughout providing concise descriptions of what is occurring (not every line needs to be commented, but generally blocks of code should be). There are no extraneous comments or code. Spacing/indentation is clear and organized.
Good	10-12	Each file includes an updated header with author. Each entity has a name, inputs, and outputs that match provided/designed schematics. Variables meet the naming conventions. There are no extraneous comments or code.
Acceptable	7-9	Each entity has a name that matches the provided/designed schematics. Variables meet the naming conventions. Portions of headers were updated, some code was cleaned up.
Unacceptable	0-6	Files were not included. Header was not updated. Extraneous comments exist. Variables do not meet naming conventions. No helpful comments.

(5 pts) Git Usage/Repo Structure

- Multiple commits with good messages
- Commit messages begin with "Lab3 –" to easily distinguish between course projects and use some form of "Final" in the name to CLEARLY indicate when you have completed your Lab
- **ALL MEMBERS** of the group will submit **YOUR OWN** code. This does not mean you can't collaborate, but make sure you have done the code yourself to learn the material.
- Repo file structure makes sense and is easy to navigate. If we have to hunt for your files or find hidden in a bunch of other files, you will lose points

(20 pts) Report Quality (1700 T25 via Gradescope)

Utilize the lab 2 template to provide **ONLY** the following sections for the Lab 3 report (report with diagrams should be two pages at most):

- Front Matter:
    - Title, Authors, and Documentation Statment.
- Simulation waveform (with discussion). Explain how you tested your FSM to make sure it was working correctly.
- Design and Results
    - Top level Entity/Schematic and Final Top Level RTL Schematic
        - Pictures provided of planned top-level hardware (use draw.io, PowerPoint, or other design tool) and Vivado RTL schematic
        - You do not need to show the architecture of the FSM in the RTL picture
        - Provide a short discussion on the RTL schematic. Is it what you expected?