Name: ______________________________

Collaborators: ________________________

Thayer School of Engineering • DARTMOUTH

ENGS 31/CoSc 56 24S

LAB 4 — State Machines

Due on Canvas by Sunday, April 28th at 11:59PM

Contents

Deliverables

1 Overview

1.1 Place in course

In class, we have been learning about state machines, VHDL, and test benches. This lab is the first time you will design and test a state machine from scratch. You will implement your own state machine to meet a pre-defined specification. You will write a testbench to confirm that it performs as expected, and ultimately, you will program an FPGA to verify the design in hardware.

1.2 Learning Objectives

- Gain additional practice converting a textual requirement into a state machine.
- Gain additional practice writing clear concise VHDL.
- Gain exposure to verification through testbenches.

2 Problem Statement

The 1965-67 Ford Thunderbird had three left and three right taillights which flashed in sequence to indicate left and right turns. As depicted in figure 1, they flashed from the center toward the outside.

A video of the lights in action can be found on Youtube[1]

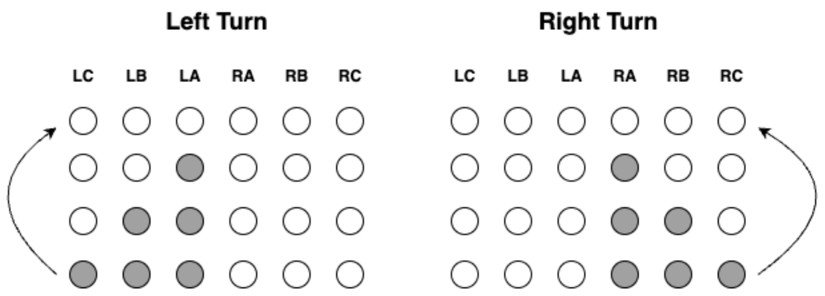

Figure 1: Light Sequence

The original taillight controller was mechanical (a motorized switch). Your task is to design a digital controller for these six lights.

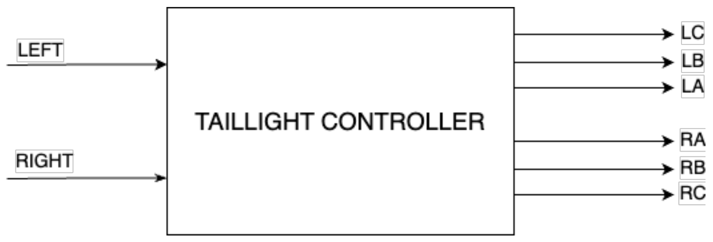

Figure 2: Inputs and Outputs

[1] https://www.youtube.com/watch?v=Qwzxn9ZPW-M

There are two inputs: `LEFT` and `RIGHT`, which come from the driver's turn signal. This is not a perfect representation, because you do not have a three-position turn signal switch available to you. Instead, your controller will work as follows:

- When `LEFT` = 1 the lights flash in the following pattern: all off; `LA` on; `LA` and `LB` on; `LA`, `LB`, and `LC` on; and then the sequence repeats. When `RIGHT` = 1, the sequence is similar for `RA`, `RB`, and `RC`.
- If a switch is turned off in the middle of a turning sequence, the sequence should complete (not go straight to all-off). Thus, in the middle of a `LEFT` (or `RIGHT`) sequence, there is no need to continue checking the `LEFT` (or `RIGHT`) input (until the all-off state is reached).
- If `LEFT` and `RIGHT` are both 1, the lights enter hazard mode, and all six flash off and on in unison.
- If the lights are in hazard mode and one of the switches is turned off, the lights all turn off before starting a turn sequence.

3 Prelab

The problem described in Section 2 is clearly well suited to a state machine. Before coming to lab, complete the following tasks.

1. Compose a state diagram that captures the behavior of the system.
2. Translate the state diagram into a VHDL model (entity plus three-process architecture), following the examples given in class and in the eBook. Use the template[2] to start your model.

Deliverable 1: Prelab *(5 Points)*

Use the space below to draw your state diagram. Submit a link to your code on EDA playground, or submit the VHDL file to Canvas.
Submit your state machine diagram to Canvas.

[2]https://www.edaplayground.com/x/KFT2

4 Procedure

4.1 About this lab

This is the first time you will create an entire VHDL design from scratch. We have provided a largely empty starter file for the state machine, which you filled out during the prelab. We are also providing a top level shell file, and the bulk of the clock divider circuit, but you are wholly responsible for the guts of the FSM and the test bench.

This is no small undertaking and represents a significant step in your digital design experience! By the time you are done with this lab, you will have seen (and created) most of the important parts of all FPGA designs.

4.2 Lab Objectives

- Write an entire state machine in VHDL and ensure that it matches the component defined in the toplevel file.
- Write an entire testbench to exercise your design.
- Gain some exposure to the constraints files used by Vivado.
- Validate your design in hardware and with your testbench.

4.3 Clock Divider

This design will be run using an 8 Hz clock.

Download the Lab 4 project folder from Canvas, extract it to your ENGS31/CoSc56 folder on your O: (home) drive, and then create a new project. Add `lab4_top_level.vhd` and `lab4_clocking.vhd` to your project as design sources. In the design sources hierarchy, expand the system sub-components nested under the lab4_shell.vhd, and open the lab4_clock_generation.vhd model. This VHDL model is very similar to the one that you used in the previous lab.

Deliverable 2: Generate an 8Hz clock *(5 points)*

What value of `CLOCK_DIVIDER_TC` do you need to create an 8Hz clock from a 100MHz clock?
How many bits are needed to store this number?

CLOCK_DIVIDER_TC ____________________

How many bits does this require ______________

Modify the "for synthesis" value of CLOCK_DIVIDER_TC in lab4_clock_generation.vhd to generate an 8 Hz clock from the 100 MHz system clock. Notice that this VHDL will automatically size the register to contain the appropriate number of bits, depending on your selection of CLOCK_DIVIDER_TC.

4.4 Test Bench and Simulation

Before testing any design in hardware, you should simulate it first to verify that all the signals do what you expect them to do. Consider the full system testbench from Lab 3. In this section of the lab, you will construct a testbench for your state machine. As you examine the anatomy of a testbench using Lab 3 as an example, write that section of the testbench for your Lab 4 system. Start with a blank .vhd. Make sure that you include your name and the purpose of this code in a comment at the top.

This testbench should correspond to the highest level of the project. Like many of the VHDL files we will create, testbenches follow a pretty standard pattern. This make them easy to read and, importantly, easy to create.

In general, our testbenches will have the following sections.

1. Library declarations and an empty testbench entity. This is the highest level of the system environment, so nothing will be passed into or out of the testbench.

   After examining the section below, create the corresponding sections for your testbench.

```
--=============================================================================
--ENGS 31/ CoSc 56 22S
--Lab 3 Prelab BCD Counter Test Bench
--B.L. Dobbins, E.W. Hansen, Professor Luke
--=============================================================================

--=============================================================================
--Library Declarations:
--=============================================================================
library IEEE;
use IEEE.std_logic_1164.all;

--=============================================================================
--Entity Declaration:
--=============================================================================
entity bcd_digit_tb is
end entity;
```

2. The architecture and component declarations. Testbenches can test multiple components simultaneously (you will do this in later labs). For now, you will just call the top level of your VHDL model as a component in the testbench. Here, you define the ports into and out of your *device under test* (the **DUT**). This component declaration must match the entity declaration of your top level.

```
--=============================================================================
--Component Declaration:
--=============================================================================
-- Declare the unit that is to be simulated
-- Need to specify what the input ports and output ports are
component bcd_digit is
  port(clk_port    : in  std_logic;
       reset_port     : in  std_logic;         --1 to reset
       enable_port  : in  std_logic;         --1 to count, 0 to hold
     y_port          : out std_logic_vector(3 downto 0);
         tc_port     : out std_logic );
end component;

```

   Pro-tip: The easiest way to create the component definitions is to open your design RTL and copy the entity section. Change *entity* to *component* and you are done.

3. Consider the inputs to the highest level of your design. This will include the external clock and the slide switches LEFT and RIGHT.

   Then consider outputs of the system. These will include LC, LB, LA, and RA, RB, RC.

   Finally, declare constants that will define your clock periods in this design. What clock period do you want to use for simulation? If you use an 8 Hz clock, your system will take a long time to simulate and use up a lot of memory on your computer. Make sure that the system clock period specified in the testbench as a constant matches the period generated by the CLOCK_DIVIDER_TC for SIMULATION in the file lab4_clock_generation.vhd.

Following the example below (from lab 3), create signals for these inputs and outputs, then constants for the clock simulation.

```
--=============================================================================
--Signal Declarations:
--=============================================================================
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Simulated signals from "function generator" (inputs)
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal clk_external         : std_logic := '0';
signal reset_pushbutton         : std_logic := '0';
signal enable_slide_switch      : std_logic := '0';

--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Simulated signals on "digital oscilloscope" (outputs)
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal y_on_scope   : std_logic_vector(3 downto 0) := "0000";
signal tc_on_scope  : std_logic := '0';

constant ext_clk_period: time    := 10ns;   -- simulating a 100 MHz clock
constant system_clk_period: time := 500ns;  -- simulating a 2 MHz clock

```

4. Now use a *port map* to connect the signals that you just made to the ports of your top-level component (the input, output pins of your device). In a port map, the ports are on the left of the assignment operator and the signals in the level of the testbench environment are on the right. This is always the case, independent of if the port is an input or an output.

```
--=============================================================================
--Wire the "Function Generator" and "Digital Oscilloscope" to our design:
--=============================================================================
begin
  dut: lab3_top_level port map (
    clk_ext_port    => clk_external,
    reset_ext_port  => reset_pushbutton,
    enable_ext_port => enable_slide_switch,
    y_ext_port     => y_on_scope,
    tc_ext_port    => tc_on_scope,
    seg_ext_port   => open,                -- "open" means not connected to the digital scope.
    an_ext_port    => open );
```

5. With the scaffolding for doing simulation set up (the signals, ports, and their interconnection defined), you can now write the first of two stimulus processes. This first process mimics the off-chip 100 MHz oscillator.

```
--=============================================================================
--Clock Generation Process:
--=============================================================================
clkgen_proc: process
begin
    clk_external <= not(clk_external);
    wait for ext_clk_period/2;
end process clkgen_proc;
```

This generates a clock with a 50% duty cycle of the correct frequency.

6. Finally, have a look at the stimulus process. This is where you provide the inputs needed to fully exercise your design.

```vhdl
--============================================================================
--Stimulus Process:
--============================================================================
stimulus_proc: process
begin
  --++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  --Test the reset functionality:
  --++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    reset_pushbutton <= '1';    -- hold in reset state for a while
    wait for 5*system_clk_period;

    reset_pushbutton <= '0';    -- let it run, but not all the way to the end
  enable_slide_switch <= '1';
    wait for 5*system_clk_period;

    reset_pushbutton <= '1';    -- reset and start over
    wait for 2*system_clk_period;

    --++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  --Test the roll-over and TC:
  --++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    reset_pushbutton <= '0';    -- let it run all the way to tc and beyond
    wait for 23*system_clk_period;

  enable_slide_switch <= '0';     -- disable the counter
    wait for 5*system_clk_period;

  reset_pushbutton <= '1';    -- reset and start over
    wait for 2*system_clk_period;

  reset_pushbutton <= '0';    -- no inputs
    wait for 2*system_clk_period;
    wait;
end process stimulus_proc;

```

Before writing your stimulus process, write down the behavior you want to test on a sheet of paper. Just as it is valuable to plan out a block diagram on paper before describing that hardware with VHDL code, planning out the tests you want to run before writing the testbench will save you time and stress, particularly as the systems you are designing, building, and validating become increasingly complex.

Then, in the stimulus process, test each of those behaviors methodically, using comments to describe each test in the code. Write your comments so another ES31/CS56 student could easily understand the tests you plan to run.

In your lab4_clock_generation.vhd model, make sure that you modify the CLOCK_DIVIDER_TC appropriately for simulation. Make sure that this matches the system clock period you defined in your test bench. This will have the effect of increasing the slow clock frequency and will make your simulation run faster. After you have completed your verification, set the constant back to the value used for synthesis.

Upload your testbench to the provided project as a simulation source (not a design source, as the testbench does not describe any hardware). Run your simulation and take screenshots of your results. Annotate the screenshots so it is easy to tell that your design is working at a brief glance of the figure. Your annotations should highlight the important features of the design.

Deliverable 3: Testbench and Screenshots *(10 Points)*

Upload your well commented testbench and annotated screenshots to Canvas. Also upload your test plan. Ensure that your testbench stimulus matches your test plan and is evidenced in your screenshots.

4.5 Writing a constraints file

Make a copy of the constraints file titled `ES31_CS56_Constraints_Template.xdc` in the VHDL folder of Lab4_Project. Save the copy in this folder and open the copy. The constraints file (XDC) is the bridge between the ports of your VHDL model and the pins of the FPGA. Each pin on the FPGA is described by a pair of lines in the similar to those below.

```
#set_property PACKAGE_PIN V17 [get_ports {sw_ext_port[0]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {sw_ext_port[0]}]
```

Any line beginning with `#` is a comment; remove the `#` from lines that you intend to use in your design. The first line says that switch `sw[0]` on the BASYS3 board is connected to pin `V17` (row V, column 17) in the FPGA package. The second line says that that pin is internally configured for a 3.3 volt power supply. If you want to use `sw[0]` in your design, uncomment the two lines and replace the name `sw[0]` with the name of the port in your entity block that you want to connect the switch to.

Every port in your entity must be routed to a pin via the XDC file. Otherwise, the implementation tool will either throw an error or assign the port to a pin of its own choosing.

Because every pin of the FPGA is routed somewhere on the BASYS3 board, you cannot assign locations arbitrarily. For example, the MEMS actuator on the BASYS3 that generates the clock signal is wired to pin W5 of the FPGA, which is connected to a clock net inside the FPGA.

```
## Clock signal
#set_property PACKAGE_PIN W5 [get_ports clk_ext_port]
#set_property IOSTANDARD LVCMOS33 [get_ports clk_ext_port]
#create_clock -add -name sys_clk_pin -period 10.0 -waveform {0 5} [get_ports clk_ext_port]

```

You must use these constraint lines to ensure that the actual clock signal is correctly routed to the clk port of your VHDL entity. If you want to give your clock a different name, change it in each line following `get_ports`.

Modify the constraints file to enable the mapping shown in table 1

Entity Name	**Package Pin Name**	**Schematic Name**
`clk_ext_port`	W5	IC9
`left_ext_port`	R2	SW15
`right_ext_port`	V17	SW0
`LC_ext_port`	L1	LD15
`LB_ext_port`	P1	LD14
`LA_ext_port`	N3	LD13
`RA_ext_port`	U19	LD2
`RB_ext_port`	E19	LD1
`RC_ext_port`	U16	LD0

Table 1: IO Mapping for Lab 4

If you are not using a particular set of ports in a bank, delete them from this file. If you need a reference, look at the constraints file from Lab3.

Deliverable 4: Constraint File *(5 Points)*

Upload a copy of your final constraints file to canvas. Be sure to remove any unused lines before uploading.

4.6 Program the FPGA

Double check that your terminal count (`CLOCK_DIVIDER_TC`) is set correctly for synthesis. Take the design through bitstream generation. When a file is elaborated or synthesized, any errors, critical warnings, or warnings are reported. Errors and critical warnings must be fixed. Warnings may or may not be benign. Consult an instructor for guidance on warnings. Program your FPGA and verify that your design is working as intended.

Deliverable 5: Program the FPGA *(10 Points)*

Once your design is programmed and working, proudly demonstrate it to a member of the staff and obtain a signature. It is also acceptable to upload a video. If you choose that route, write the word video on the signature line

Staff Signature: ______________________________

4.7 Final Design

Finally, turn in a neat and accurate copy of your final state machine diagram and VHDL implementation of the FSM.

Deliverable 6: Final Design Documentation *(20 Points)*

Upload the files to Canvas. Be sure that the state machine diagram adhere to our standards and that the code is complete and well commented.