Lab 4: Sequential Logic and Finite State Machines

EEL 4712 – Fall 2019

Objective:

The objective of this lab is to use finite state machines to implement several counters, with the clock being generated from a debounced button press after a specific amount of time. It is up to you to determine how the counters get mapped onto the board by analyzing the provided top_level entity.

Required tools and parts:

Quartus2 software package, ModelSim-Altera Starter Edition, Altera DE10 board.

Pre-requisite:

You must be "up-to-speed" with Quartus, ModelSim, and the board before coming to lab.

Pre-lab requirements:

Clock Generator

1. To be able to see the output of the counters in real time, you will need to control the counters using a button press to generate a clock. To do this, you will create a clock generator that generates a single clock pulse (i.e. low-high-low transition) when the button has been pressed for 1000ms (1 sec). If the button is continually held down, it should generate a pulse every second until the button is released.

   This step requires several entities. First, design a clock divider (clk_div.vhd) that converts the 50 MHz clock on the board to a 1000 Hz (1ms) clock with any duty cycle you want. Next, design the clock generator (clk_gen.vhd), which will count 1000 Hz clock pulses (each is 1 millisecond) while the button is pressed down until 1000ms have elapsed, at which point it will output (i.e. generate) a single clock pulse based on the 1000 Hz clock (i.e., the output clock will be high for one 1000 Hz clock period). You are free to implement the clock generator architecture however you like, as long as the entity doesn't change from the provided file and as long as it uses the clock divider. Feel free to add additional entities that you may require.

   It is not possible to guarantee that a generated clock pulse occurs after exactly 1s because the button may be pressed in the middle of the 1ms clock. Therefore, the actual time for the generated clock after the first button press will be between [1000ms, 1001ms). In other words, the first generated clock will occur anytime being 1000ms and 1001ms after the button was pressed. However, for repeated generations of the clock, it is possible to produce a clock after exactly 1000ms because you know that the button was already pressed at the beginning of the cycle. The provided testbench tests these times, so be aware that being off by a single cycle will cause assertion errors.

   Both the clk_div and clk_gen entity use generics. Clk_div must work using generics that specify the input and output clock frequencies. Clk_gen works by specifying how many milliseconds the button must be pressed to generate a clock pulse.

   A test bench will be provided to help you test clk_div and clk_gen, although you should also test each entity using your own test benches.

Turn in all vhdl files. For clk_div, include in your pre-lab report example waveforms for clock ratios of 2 and 4. See the test bench for details on how to change the ratio. For clk_gen, your code will be tested using a test bench similar to the one provided, so make sure there are no error messages during your simulations. Also, make sure to test different ms times for the button press.

4-bit Course Number Counter

2. Design counter using a finite state machine which outputs the course number as follows. Therefore, the binary sequence for the counter should be:

0000 (0)	1100 (C)
0001 (1)	1101 (D)
0011 (3)	1111 (F)
0010 (2)	1110 (E)
0110 (6)	1010 (A)
0111 (7)	1011 (B)
0101 (5)	1001 (9)
0100 (4)	1000 (8)

The signal "go" in the entity will determine whether the state will change. If "go" is asserted then the output will progress, else it will hold on the current state. (Hint: It may help to draw an FSM diagram)

Note that this table is read top-to-bottom in the left column and then the right column. Also, after "1000" the counter should go back to the beginning and output "0000". Create an entity for the Gray code counter called gray1 (gray1.vhd). Use the 1-process FSM model (i.e., a single process with nothing except clock and reset in the sensitivity list). Use the provided entity.

Create your own test bench and use it to generate a waveform that illustrates the correct functionality. Include the waveform in your pre-lab report. Turn in all vhdl files.

3. Create another entity (gray2 in gray2.vhd) for the Gray code counter, using the 2-process FSM model (i.e., one process for sequential logic and one process for combinational logic). Use the provided entity.

Create your own test bench and use it to generate a waveform that illustrates the correct functionality. Include the waveform in your pre-lab report. Turn in all vhdl files.

4-bit Up/Down Counter with Load

4. Create a 4-bit up/down counter (counter.vhd) that counts upwards when the active-low input "up_n" is asserted (i.e., ='0') and down otherwise. The counter should count from 0 to 15 and overlap to 0 when counting up, and the opposite when counting down, although the counter should start at 0 when reset. Also, configure a "go" signal which performs the same function as the one in the gray code counter (no state change if go = '0'). In addition, the counter should be able to load a count from the switches when load_n = '0'. Load_n should take priority over up_n. Both load_n and up_n are synchronous.
You are free to implement the counter however you want (it does not have to be an FSM), as long as you conform to the provided entity. Be aware that this counter can be implemented with very little code, so if your architecture description is getting long, consider a different way of implementing it. However, do not violate any synthesis coding guidelines.

Create your own test bench and use it to generate a waveform that illustrates the correct functionality. Add annotations to illustrate all input operations (up, down, load, go). Include the waveform in your pre-lab report. Turn in all vhdl files.

Top Level

5. Read the code for the provided top_level entity (top_level.vhd) and describe what it does. Be specific.

Include the description in your pre-lab report.

In-lab procedure (do as much as possible ahead of time):

1. Using Quartus, assign pins to each of the top_level.vhd inputs/outputs such that the signals are connected to the appropriate locations on the board. Note that the exact connections are purposely omitted so that you have to understand the top_level.vhd file. Make sure to add the 7-segment decoder code to your project.
2. Download your design to the board, and test it for different inputs and outputs. Demonstrate the correct functionality for the TA.
3. **Be prepared to answer simple questions or to make simple extensions that your TA may request. There is no need to memorize the different packages. If you have done the pre-lab exercises, these questions should not be difficult.**

Lab report: (In-lab part only)

If you had any problems with portions of the lab that could not be resolved during lab, please discuss them along with possible justifications and solutions. If you had no problems, this report is not necessary.

Turn the lab report in on e-learning, if explanation is needed for partial credit. Make sure to turn it in to the "lab" section and not the "pre-lab" section