Lab 4: Finite State Machines

EEL 4712 – Spring 2012

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. 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, ModelSim-Altera Starter Edition, UF-4712 board

Pre-requisite:

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

Pre-lab requirements:

Debounce Logic and Clock Generator

1. To be able to see the output of the counters in real time, you will need to a slow clock signal that is controlled by a button press. 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 25 MHz clock on the board to a 1000 **Hz** 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 between 1000ms and 1001 ms have elapsed, at which point it will output a single clock pulse with the same width as the 1000 Hz clock. Note that the clock generator should only create a pulse *after* 1000 ms have elapsed. In other words, because of the 1 ms resolution of the input clock, the actual time could be anywhere from [1000ms, 1001 ms) depending on when the user presses the button. If the button is pressed right before a rising clock edge, the generator waits for approximately 1000 ms. If the button is pressed right after a clock edge, the generator waits just under 1001 ms.

   Although this timing is not noticeable to someone using the board, timing is often critical in digital circuits. Therefore, the testbench will ensure you are waiting for the requested amount of time. As an example where the timing differences would be more obvious, imagine the input clock being 1 Hz. If the generator doesn't wait at least one second and the user presses the button right before a rising edge, the generator would create a clock pulse instantly, which would obviously not seem like a second. Of course, if the user pressed the button right after a rising edge, the generator would wait almost 2 seconds, but this would still meet the specified requirement.

   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. Note that requiring the button to be held down eliminates bouncing problems, because the bounces will simply reset the time by a negligible amount (1 ms).

   Note that 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 is 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 waveforms for clock ratios of 2 and 4. See the testbench for details on how to change the ratio. For clk_gen, your code will be tested using a testbench 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 Gray code counter

2. Design a 4-bit Gray code counter. Gray code is a numerical system where two successive values differ by only a single bit. Therefore, the binary sequence for the counter should be:

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

Note that after “1000” the counter should go back to the beginning and output “0000”. Create an architecture for the provided 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 an architecture for the provided gray2 entity (gray2.vhd), 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. In addition, the counter should be able to load a count from one of the dipswitches 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, 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.

> 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). 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. 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.