Laboratory Exercise 5

Clocks and Counters

The purpose of this exercise is to learn how to create counters and to be able to control when operations occur when the actual clock rate is much faster.

Preparation Before the Lab

You are required to complete Parts I to III of the lab by writing and testing Verilog code and compiling it with Quartus II. Show your schematics, Verilog, and simulations for Parts I to III to the teaching assistants. You must simulate your circuit with ModelSim (using reasonable test vectors you can justify).

In-lab Work

You are required to implement and test all of Parts I to III of the lab. You need to demonstrate all parts to the teaching assistants.

Part I

Consider the circuit in Figure 1. It is a 4-bit synchronous counter that uses four T-type flip-flops. The counter increments its value on each positive edge of the clock if the *Enable* signal is asserted. The counter is reset to 0 by setting the *Clear_b* signal low – it is an active-low asynchronous clear. You are to implement an 8-bit counter of this type.

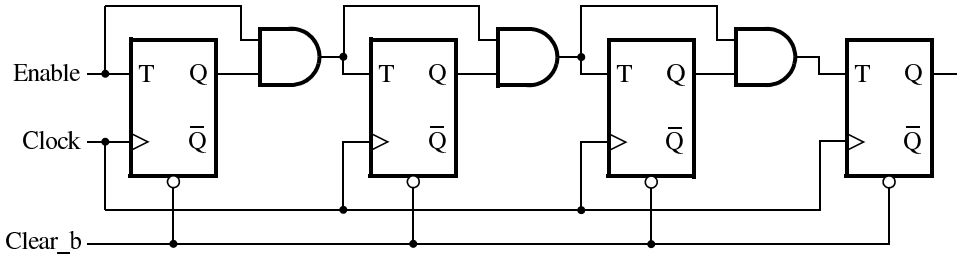

Figure 1: A 4-bit counter.

Perform the following steps:

1. Draw a schematic outlining the hierarchies you will use and explain them to the TA as part of your prelab.
2. Write a Verilog file that defines an 8-bit counter by using the structure depicted in Figure 1. Your code should include a T flip-flop module that is instantiated 8 times to create the counter (i.e. structural Verilog). Compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, F_{max}, at which your circuit can be operated? (Use TimeQuest in Quartus to determine the maximum frequency F_{max} – Refer to the *Using TimeQuest Timing Analyzer* document found on the Altera website.)
3. Simulate your circuit to verify its correctness as part of your prelab.
4. Augment your Verilog file to use the pushbutton KEY_0 as the *Clock* input, switches SW_1 and SW_0 as *Enable* and *Clear_b* inputs, and 7-segment displays *HEX0* and *HEX1* to display the hexadecimal count as your circuit operates. Make the necessary pin assignments needed to implement the circuit on the DE1-SoC

board, and compile the circuit. For this part, you should re-use the hexadecimal-to-7-segment display decoder that you created for Lab 2.

5. Download your circuit into the FPGA chip and test its functionality by operating the switches.

6. Use the Quartus II RTL Viewer to see how the Quartus II software synthesized your circuit. What are the differences in comparison with Figure 1?

Part II

Another way to specify a counter is by using a register and adding 1 to its value. This can be accomplished using the following Verilog statement:

$$Q <= Q + 1;$$

An example code is shown below of a counter that counts from hexadecimal values 0 to F

```
reg [3:0] q;                    // declare q

always @(posedge clock)         // triggered every time clock rises
begin
   if (Clear_b == 1'b0)         // when Clear_b is 0
      q <= 0;                   // q is set to 0
   else if (q == 4'b1111)       // when q is maximum value for the counter
      q <= 0;                   // q reset to 0
   else if (Enable == 1'b1)     // increment q only when Enable is 1
      q <= q + 1;               // increment q
end
```

Observe that q is declared as a 4-bit value making this a 4-bit counter. The check for the maximum value is not necessary in the example above. Why? If you wanted this 4-bit counter to count from 0-9, what would you do?

Design and implement a circuit using counters that successively flashes the hexadecimal digits 0 through F on the 7-segment display *HEX0*. You will use two switches, SW_1 and SW_0 to determine the speed of flashing according to the following table:

SW[1]	SW[0]	Speed
0	0	Full
0	1	1 Hz
1	0	0.5 Hz
1	1	0.25 Hz

Full speed should use the 50 MHz clock signal provided on the DE1-SoC board. You must design a fully synchronous circuit, which means that every flip flop in your circuit should be clocked by the same 50 MHz clock signal. To derive the slower flashing rates you should use a counter, call it RateDivider, that is also clocked with the 50 MHz clock. The output of RateDivider can be used as part of a circuit to create pulses at the required rates. These pulses can be used to drive an *enable* signal on the counter, call it DisplayCounter, that is counting from 0 through F. Recall that an *enable* signal determines whether a flip flop, register, or counter will change on a clock pulse. Figure 2 shows a timing diagram for a 1 Hz enable signal with respect to a 50 MHz clock.

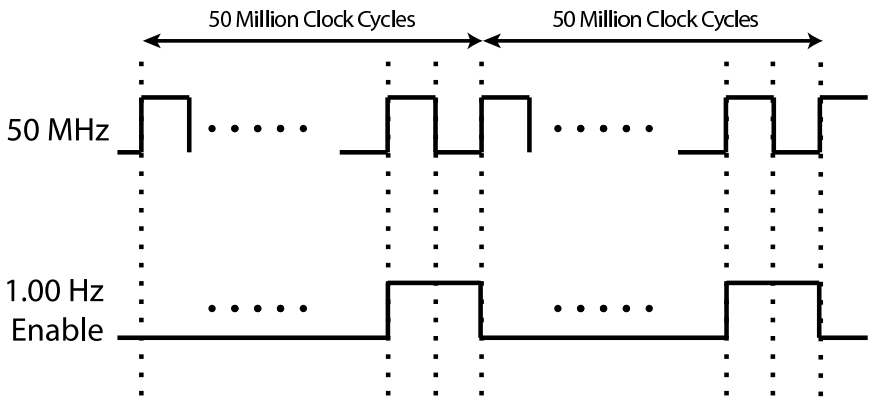

Figure 2: Timing diagram for a 1 Hz enable signal

Perform the following steps.

1. Draw a schematic outlining the hierarchies you will use and explain them to the TA as part of your prelab.
2. Write a Verilog file that realizes the behaviour described above. Your circuit should have the clock and the two switches as inputs.
3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must show this to the TA as part of your prelab. You will also need to think about how to simulate this kind of circuit. For example, how many 50 MHz clock pulses will you need to simulate to show that the RateDivider is properly outputting a 1 Hz pulse?
4. Compile the project.
5. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit.

Part III

In this part of the exercise you are to implement a Morse code encoder using a lookup table (LUT), which can be implemented as a multiplexer with hard-coded inputs, and a rate divider similar to Part II. The Morse code uses patterns of short and long pulses to represent a message. Each letter is represented as a sequence of dots (a short pulse), and dashes (a long pulse). For example, starting from A, eight letters of the alphabet have the following representation:

A	• —
B	— • • •
C	— • — •
D	— • •
E	•
F	• • — •
G	— — •
H	• • • •

You will design and implement a Morse code encoder circuit using a LUT and a rate divider. The LUT stores the patterns for each letter. When a letter is selected, the appropriate pattern is loaded from the LUT into the shift register via parallel load. The pattern is then shifted out of the shift register one bit at a time.

Your circuit should take as input one of the eight letters of the alphabet starting from A (as in the table above) and display the Morse code for it on a red LED, $LEDR_0$. Use switches SW_{2-0} and pushbuttons KEY_{1-0} as inputs. When a user presses KEY_1, the circuit should display the Morse code for a letter specified by SW_{2-0} (000 for A,

001 for B, etc.), using 0.5-second pulses to represent dots, and 1.5-second pulses to represent dashes. The time between pulses is 0.5 seconds. Pushbutton KEY_0 should function as an asynchronous reset.

Hint: Since your minimum time is 0.5 seconds, set each 0 or 1 to be 0.5 seconds. This means that a 0 is a pause or off, a 1 is a dot, and 111 is a dash. Then read each 0 or 1 individually out of a shift register at 0.5 seconds per read. You should have observed that the codes are different lengths. You may assume that all letters can be stored using a single pattern length, i.e., all patterns stored in the LUT use the same number of bits.

Perform the following steps.

1. Draw a schematic outlining the hierarchies you will use and explain them to the TA as part of your prelab.
2. Write a Verilog module that realizes the behaviour described above.
3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must show this to the TA as part of your prelab.
4. Compile the project.
5. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit.