Homework Exercise 3
Counters

This is Altera's Laboratory Exercise 4.

Part 1 We will start this part in class.

Consider the circuit in Figure 1. It is a 4-bit synchronous counter which uses four T-type flip-flops. The counter increments its count on each positive edge of the clock if the Enable signal is asserted. The counter is reset to 0 by using the Clear (Reset) signal. You are to implement a 16-bit counter of this type.

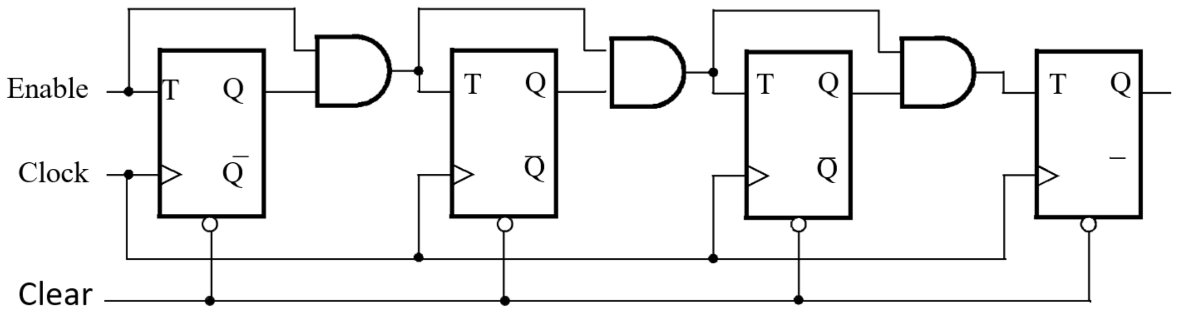

Figure 1. A 4-bit counter.

1. Write a SystemVerilog file that defines a 16-bit counter by using the structure depicted in Figure 1. Your code should include a T flip-flop module that is instantiated 16 times to create the counter.
2. Simulate your circuit to verify its correctness using ModelSim. Figure out some convincing way to demonstrate correct circuit operation without including all 65,536 lines of output! The assert() and/or wait() functions are your friends!
3. Augment your project with a top-level SystemVerilog file called Part1.sv that uses the pushbutton KEY_0 as the *Clock* input, switch SW_1 as an active high *Reset* (Clear) and switch SW_0 as an active high *Enable*, and 7-segment displays *HEX3-0* to display the hexadecimal count as your circuit operates. Red LEDs should show the state of the switches. Make the necessary pin assignments needed to implement the circuit on the DE2 board, and compile the circuit.
4. Compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, *Fmax* (Slow 1200mV 85C Model), at which your circuit can be operated? Write the answers to these questions in the comments at the top of your Part1 module.
5. Download your circuit into the FPGA chip and test its functionality by operating the implemented switches and push-button.
6. Use the Quartus II RTL Viewer to see how Quartus II software synthesized your circuit. What are the differences in comparison with Figure 1?

7. The runlab.do you turn in should run a testbench for your 16-bit counter module (item #1, above). It should contain a line that calls the appropriate *wave.do for this module.

Part 2

Simplify your SystemVerilog code so that the counter specification is based on the SystemVerilog statement

```
Q <= Q + 1'b1;
```

1. Write a Verilog module that defines a 16-bit counter by using this approach.
2. Simulate your circuit to verify its correctness using ModelSim. Note: you can use the same testbench as in Part 1.
3. Augment your project with a top-level Verilog module called Part2 that uses the pushbutton KEY_0 as the *Clock* input, switch SW_1 as an active high *Reset* (Clear) and switch SW_0 as an active high *Enable*, and 7-segment displays *HEX3-0* to display the hexadecimal count as your circuit operates. Red LEDs should show the state of the switches. Make the necessary pin assignments needed to implement the circuit on the DE2 board, and compile the circuit.
4. Compile the circuit. How many logic elements (LEs) are used to implement this circuit? What is the maximum frequency, *Fmax* (Slow 1200mV 85C Model), at which your circuit can be operated? Comment on the differences between this and Part 1. Write the answers to these questions in the comments at the top of your Part1 module.
5. Download your circuit into the FPGA chip and test its functionality by operating the implemented switches.
6. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Part 1.
7. The runlab.do you turn in should run a testbench for your 16-bit counter module (item #1, above). It should contain a line that calls the appropriate *wave.do for this module.

Part 3

Use an LPM from the Library of Parameterized modules to implement a 16-bit counter. Choose the LPM options to be consistent with the above design, i.e. with enable and synchronous clear.

1. Write a SystemVerilog module that implements a 16-bit counter by using this approach.
2. (no ModelSim for this part – we will discuss later how to incorporate LPMs in your ModelSim simulations).
3. Augment your Verilog file to use the pushbutton KEY_0 as the *Clock* input, switches SW_1 and SW_0 as *Enable* and *Reset* inputs, and 7-segment displays *HEX3-0* to display the hexadecimal count as your circuit operates. Red LEDs should show the state of the switches. Make the necessary pin assignments needed to implement the circuit on the DE2 board, and compile the circuit.
4. Compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, *Fmax* (Slow 1200mV 85C Model), at which your circuit can be operated? Comment

on the differences between this and Part 1 and Part 2.

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

6. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Parts 1 and 2.

Part 4

Design and implement a circuit that successively flashes digits 0 through 9 (and then repeat) on the 7-segment display $HEX0$. Each digit should be displayed for about one second. Use a counter to determine the one-second intervals. The counter should be incremented by the 50-MHz clock signal provided on the DE2 board. **Do not derive any other clock signals in your design–make sure that all flip-flops in your circuit are clocked directly by the 50 MHz clock signal. This means you should have no clock warnings in your Quartus compile.** We will discuss in class how to simulate the counter required to implement this circuit.

The runlab.do you turn in should run a testbench for the counter module you used for this circuit. It should contain a line that calls the appropriate *wave.do for this module.

Part 5

Design and implement a circuit that displays the word HELLO, in ticker tape fashion, on the eight 7-segment displays $HEX7 - 0$. Make the letters move from right to left in intervals of about one second. The patterns that should be displayed in successive clock intervals are given in Table 1.

Do not derive any other clock signals in your design–make sure that all flip-flops in your circuit are clocked directly by the 50 MHz clock signal. This means you should have no clock warnings in your Quartus compile.

The runlab.do you turn in should run a testbench for the counter module you used for this circuit. It should contain a line that calls the appropriate *wave.do for this module.

Clock cycle	Displayed pattern							
0				H	E	L	L	O
1			H	E	L	L	O	
2		H	E	L	L	O		
3	H	E	L	L	O			
4	E	L	L	O				H
5	L	L	O				H	E
6	L	O				H	E	L
7	O				H	E	L	L
8				H	E	L	L	O
. . .	and so on							

Table 1. Scrolling the word HELLO in ticker-tape fashion.