Laboratory Exercise 5 – ECE241 Fall 2014

Counters, Arithmetic

Preparation

You are required to write the Verilog code for Parts I to V. For marking by the teaching assistants, you need to bring with you (pasted into your lab book) your Verilog code for Parts III, IV and V.

When doing FPGA designs, it is important to understand what makes circuits go faster and how much logic is being used. How you write your Verilog can have a significant influence on performance and resource usage. Print out and paste into your lab book, for Part I, the portion of the Quartus II report showing the F_{max} of your counter circuit (i.e., the maximum clock frequency at which your counter may be safely operated) and the number of logic elements (LEs) required to implement the counter. Finally, for Part III, print out a simulation that exercises the circuit for several clock cycles.

In-lab Work

You are required to implement and test all of Parts I to V of the lab. But you only need to demonstrate to the teaching assistants Parts III, IV and V. Your mark will be based on these three parts of the lab.

Part I

Consider the circuit in Figure 1. It is a 4-bit synchronous counter which 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* signal low – it is an active-low asynchronous clear. You are to implement an 8-bit counter of this type.

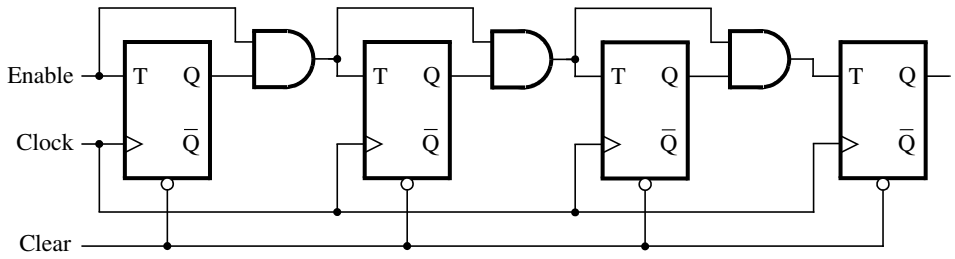

Figure 1: A 4-bit counter.

1. 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}.)
2. Simulate your circuit to verify its correctness.
3. Augment your Verilog file to use the pushbutton *KEY[0]* as the *Clock* input, switches *SW[1]* and *SW[0]* as *Enable* and *Clear* 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 DE2 board, and compile the circuit. For this part, you should re-use the hexadecimal-to-7-segment display decoder that you created for Lab #4.

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

5. Use the Quartus II RTL Viewer to see how 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:

$$Q <= Q + 1;$$

Compile an 8-bit version of this counter and determine the number of LEs needed and the F_{max} that is attainable. Use the same *KEY*, *SW* and 7-segment displays as in Part I above. Use the RTL Viewer to see the structure of this implementation versus the design from Part I. Repeat the steps of Part I above for this counter.

Part III

Design and implement a circuit 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 DE2 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.

1. Write a Verilog file that realizes the behaviour described above. Your circuit should have the clock and the two switches as inputs.

2. Simulate your circuit with QSim to verify its correctness. You will find that it is not realistic to simulate your *final* circuit. Why? This is a case where you need to find a way to make your simulations complete in a realistic time, yet still have confidence that the *final* circuit will be correct. Use your simulation to demonstrate that the circuit should perform correctly.

3. Include the pin constraints for the DE2 board, and synthesize the circuit with Quartus II.

4. Download your circuit into the FPGA chip and test its functionality.

Part IV

Recall the circuit you built in Part I of Lab 3 where you used three of the KEY pushbuttons to rotate a string on six HEX displays. Modify the circuit you just built in Part III of this lab so that it can be used to automatically scroll the characters instead of using the KEY pushbuttons.

1. Write a Verilog file that realizes the behaviour described above. You should use the 50 MHz clock on the DE2 board as the clock input to your circuit. Your circuit should only have the clock and the two switches as inputs. As in Part III, the only clock in your design should be the 50 MHz clock.
2. Simulate your circuit with QSim to verify its correctness.
3. Include the pin constraints for the DE2 board, and synthesize the circuit with Quartus II.
4. Download your circuit into the FPGA chip and test its functionality.

Part V

As you have seen in Lab 3, a ripple-carry adder circuit can be implemented by instantiating full adders. However, an adder circuit can likewise be implemented using a '+' sign in Verilog. For example, the following code fragment adds *n*-bit numbers A and B to produce outputs *sum* and *carry*. Note how the left hand side of the assign statement is one bit wider than the operands on the right hand side to accommodate the generation of the carry bit:

```
wire [n-1:0] sum;
wire carry;
...
assign {carry, sum} = A + B;
```

The curly braces in the example above ({ }) are used to implement *concatenation*. Use this construct to implement the circuit shown in Figure 2. Here the carry bit is used to signify when an overflow has occurred, i.e., that the value of the result cannot be represented as an 8-bit number so that the resulting value at *S* is incorrect.

Design and compile your circuit with the Quartus II software, download it onto a DE2 board, and test its operation as follows:

1. Create a new Quartus II project.
2. Write Verilog code that describes the circuit in Figure 2. You are encouraged to use procedural Verilog.
3. Connect input A to switches *SW[7:0]*, use *KEY[0]* as an active-low asynchronous reset for all flip-flops in the circuit and *KEY[1]* as a manual clock input. The sum output should be displayed on the red *LEDR[7:0]* lights and the *Overflow* should be displayed on the red *LEDR[8]* light.
4. Assign the pins on the FPGA to connect to the switches, keys and 7-segment displays.
5. Compile your design and use timing simulation to verify the correct operation of the circuit. Once the simulation works properly, download the circuit onto the DE2 board and test it by using different values of A. Be sure to check that the *Overflow* output works correctly.
6. Open the Quartus II Compilation Report and examine the results reported by the Timing Analyzer. What is the maximum operation frequency, F_{max}, of your circuit? What is the longest path in the circuit in terms of delay?

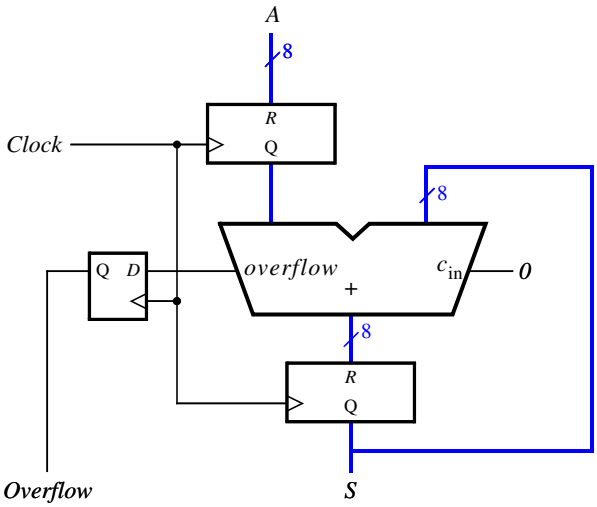

Figure 2: An eight-bit accumulator circuit.