EC311 Introduction to Logic Design Spring 2017
Lab 3: Sequential Design – Counters and Debouncers

Goals

- To get acquainted with Verilog implementation of sequential logic.
- To implement clock-driven modules in Verilog.

1. Overview

This lab will introduce you to sequential logic and Verilog implementation of it. Sequential logic, unlike combinational logic, is dependent on an external trigger. A periodic external trigger, called a clock, allows all parts of the system to be synchronized.

In this lab, you will implement an 8-bit counter using behavioral Verilog modeling and utilizing push-buttons on the Nexys-3 FPGA board. Pushing the button may last several clock cycles, thus the counter will count several times for a single push. To resolve this issue, you will also write Verilog code to debounce the push-buttons of the FPGA board. The debouncer should output 1 for a single clock cycle once no matter how long the button lasts.

Tasks

1. Button Triggered 8-bit Counter

Design an 8-bit counter with global clock and reset inputs, and an input trigger. The output of the counter should be the value of the counter, count. At the positive edge of the clock, if the trigger is high then the counter is incremented. At the positive edge of reset, the counter should be reset to zero. Connect the trigger to one of the push-buttons. A code-fragment example for a 2-bit counter is provided below:

```
output reg  [1:0] count;

always @(posedge clock or posedge reset)
begin
        if (reset)
        begin
                count = 2'b0;
        end
        else if (trigger)
        begin
                count = count + 1'b1;
        end
end
```

The internal clock signal for the FPGA is designated as ‘v10’.

Note: If you receive an error during the mapping stage, saying that your push-button cannot be a clock, put the following line of code in your UCF file:

NET ``button'' CLOCK_DEDICATED_ROUTE = TRUE;

2. Debouncer

While testing the first part of your design, you will notice that one push of a button does not always increment the count by 1, but rather by some random value. This happens because of the mechanical “bouncing" of the button. To avoid this, you need to implement a debouncer – a low-pass-filter that cleans the input signal.

As a very basic debouncer, we can implement a second counter that increments every clock cycle after the push-button was pressed. Only once the second counter has reached its maximum value, we determine that the push-button has changed state. At this point we can also reset the second counter. This technique utilizes the fact that our global clock is significantly faster than the time it takes a person to press the bush-button repetitively.

Your debouncer module should have global clock and reset inputs, as well as a push-button input. Its output should be the clean push-button signal. A sample algorithm for implementing the debouncer is shown in Figure 1 below.

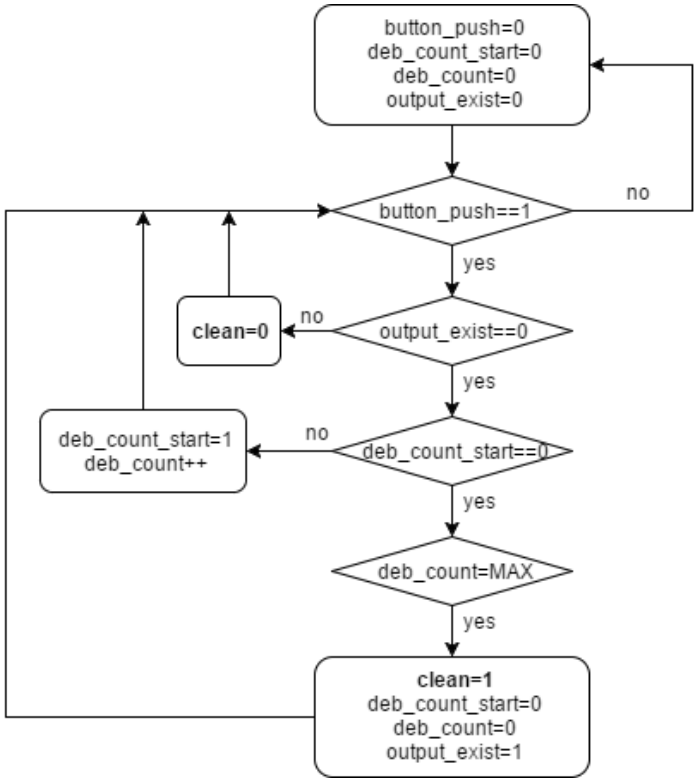

Figure 1: Sample algorithm for implementing the debouncer

Be sure to test your bouncer with different max values until you find one that works well.

For more information on debouncers and possible solutions, see
http://www.fpga4fun.com/Debouncer.html

3. Final Design

Your final design for the counter (i.e. "debounced counter") should instantiate two modules: debouncer and counter. It should have global clock and global reset inputs, a push-button input, and an 8-bit counter output. Refer to figure 2 below.

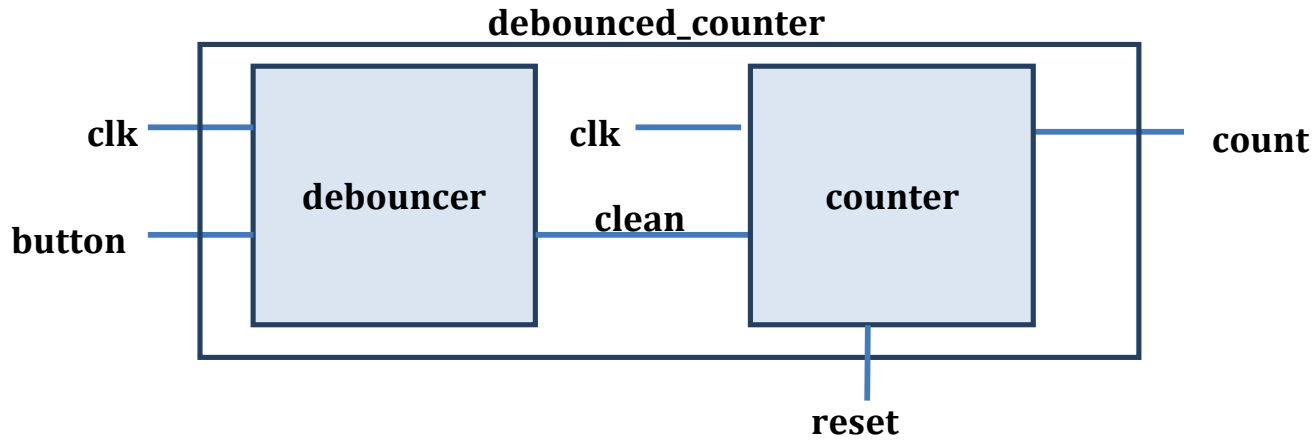

Figure 2: Block diagram for the final design

Deliverables

- Demo your simulation waveforms and final design uploaded to the Xilinx Nexys-3 FPGA board.
- Submit your Verilog code on blackboard. It is recommended to tar or zip all your code together before uploading it.