Experiment 6. Counters and Registers

Masud ul Hasan · Alauddin Amin · Ahmad Khayyat – Version 151, 12 October 2015

Table of Contents

1. Objectives

- Learn about sequential circuits in general, and how to use counters and registers in particular.
- Understand and demonstrate the effect of the bouncing problem on counters.
- Use and demonstrate the effect of a debouncing circuit.
- Search through the available modules in the Xilinx library of components.

2. Materials Required

- An FPGA prototyping board.
- Design and simulation software tools.
- Verilog debouncing circuit module.

3. Background

Counters and registers belong to a category of digital circuits known as *sequential circuits*. *Sequential circuits*, as opposed to *combinational circuits* have a notion of time: the behavior of the circuit can depend on its behavior in the past. In this section, you will learn about sequential circuits, counters, registers, and a useful application of counters: how to counter the bouncing effect of mechanical switches.

3.1. Sequential Circuits

- *Sequential logic circuits* have *multiple states*, where the values of circuit output(s) in one state may differ from their values in another state even for the same input values.
- In contrast, a *combinational* logic circuit, e.g. half adder, full adder, or an *n*-bit adder, has a *single state*. Thus, the output(s) of combinational logic circuits depend only on the applied input values.

- States of sequential circuits are stored in memory elements.
- The state of a sequential circuit may change only when it receives a *triggering pulse* (on an input signal commonly called `clock`).
- If no triggering pulse is received, the circuit state can never change.
- Two of the most commonly used sequential elements are:
  1. Counters; and
  2. Registers.

3.2. Binary Counters

- These are digital *counting* circuits that are commonly used in many applications, e.g. stop watches, timers, speedometers, tachometers, alarm clocks, home appliances, mobile phones, etc.
- For example, a *modulo 16* binary counter is characterized by:
  1. Every triggering pulse changes the state of the counter by changing its output incrementally, cycling through the values: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \ldots \rightarrow 14 \rightarrow 15 \rightarrow 0 \rightarrow \ldots$.
  2. After reaching the maximum count of 15, the next pulse will take the counter back to a value of 0, and the sequence repeats every 16 pulses.
  3. The output count is in binary. Since the maximum output count is 15, the output count has 4 bits.
  4. A *down counter* counts in reverse direction as follows: $15 \rightarrow 14 \rightarrow 13 \rightarrow \ldots \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow 15 \rightarrow \ldots$.
- Some counters are *bidirectional*, i.e. they allow counting either *up* or *down*.
- In general, a modulo-n counter counts as follows: $0 \rightarrow 1 \rightarrow \ldots \rightarrow n - 1 \rightarrow 0$.
- The Xilinx library has several types of counters with a variety of capabilities, e.g. the CB4CLED 4-bit mod-16, the CB8CLED 8-bit mod-256, and the CB16CLED 16-bit mod-2^{16} binary counters.

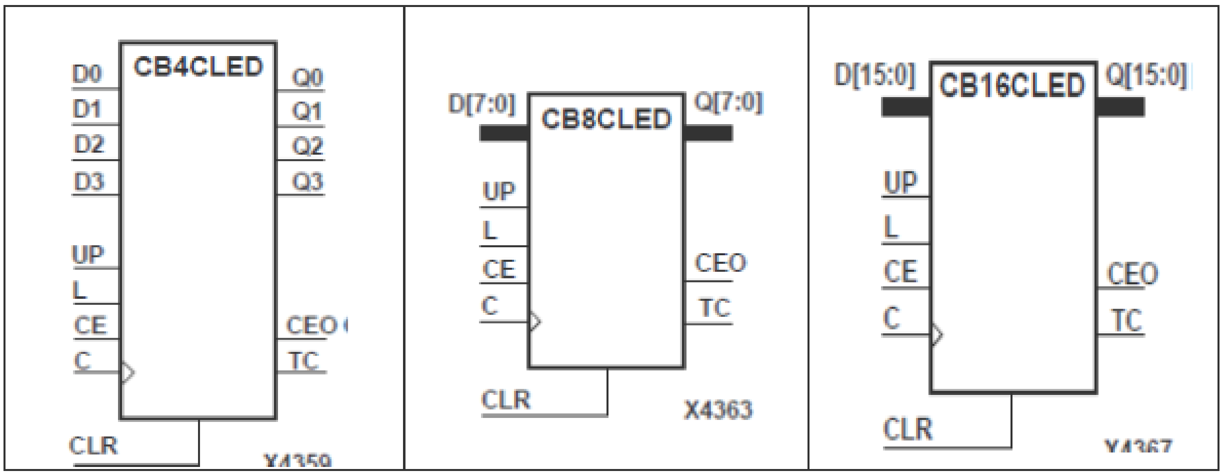

Figure 1. The CB4CLED, CB8CLED and CB16CLED Binary Counters

- The counter control inputs *load* (`L`), *count enable* (`CE`), and *up* are *synchronous* inputs (synchronized to the clock signal), i.e. they do not affect the operation of the counter except after the clock pulse is received.
- The *clear* (`CLR`) control input, however, is *asynchronous*, i.e. it clears the counter contents immediately without waiting for the clock pulse.

The asynchronous nature of the `CLR` input is indicated in the Function Table of the CB4CLED Counter with the clock pulse input (`C`) marked as a don't care, compared to the synchronous inputs which are indicated by a pulse signal at the clock input (`C`). That is, unlike synchronous inputs, how the `CLR` input is not affected by the `clock` signal.

Table 1. Function Table of the CB4CLED Counter

Input						Output
CLR	**L**	**CE**	**C**	**UP**	**$D_3 - D_0$**	**$Q_3 - Q_0$**
1	X	X	X	X	X	0
0	1	X	⎍	X	D_{in} (3:0)	D_{in} (3:0)
0	0	0	X	X	X	No Change
0	0	1	⎍	1	X	Increment
0	0	1	⎍	0	X	Decrement

Exercise

- Study the counter operation and the function of each input using its *Symbol Info* and the Function Table of the CB4CLED Counter.
- Given a CB4CLED 4-bit mod-16 counter, determine the correct counter output *after applying a clock pulse* in the following cases:

Inputs					Outputs (before clock pulse)	Outputs (after clock pulse)
CLR	**L**	**CE**	**UP**	**$D_3 - D_0$**	**$Q_3(t) - Q_0(t)$**	**$Q_3(t+1) - Q_0(t+1)$**
0	0	1	1	1010	0111	
1	0	1	1	1010	0111	
1	0	1	0	1010	0111	
0	0	1	0	1010	0111	
0	1	1	1	1010	0111	
0	0	0	1	1010	0111	
0	0	0	0	1010	0111	
0	1	0	1	1010	0111	
1	1	0	1	1010	0111	

3.3. Registers

- An *n*-bit register is an electronic circuit capable of storing *n* bits of data.

- An n-bit register will, typically, have n input data bits ($D_0 : D_{n-1}$) and n output data bits ($Q_0 : Q_{n-1}$) which show the current contents of the register (current stored values).
- To store a new set of values into the register:
  1. The values of the new n-bit data are applied at the n inputs ($D_0 : D_{n-1}$).
  2. The input `clock` signal is *pulsed* to store the applied input data in the register. If the clock signal is not pulsed, the new applied input data will not be stored and the data already stored in the register will remain unchanged.
- The Xilinx library has several types of registers, among which are the FD4CE, FD8CE, FD16CE 4-bit, 8-bit, and 16-bit registers, respectively.
- Another type of commonly used registers includes the FD4RE, FD8RE, FD16RE 4-bit, 8-bit, and 16-bit registers, respectively.
- The only difference between the two sets of registers is that the *clear* input (`CLR`) is *asynchronous* in the FD4CE, FD8CE, FD16CE family, but is *synchronous* in the FD4RE, FD8RE, FD16RE family of registers.

Table 2. Function Table of the FD4CE Register

Inputs				Outputs
CLR	**CE**	**$D_3 - D_0$**	**C**	**$Q_3 - Q_0$**
1	X	X	X	0
0	0	X	X	No Change
0	1	$D_{in}(3:0)$	↑	$D_{in}(3:0)$

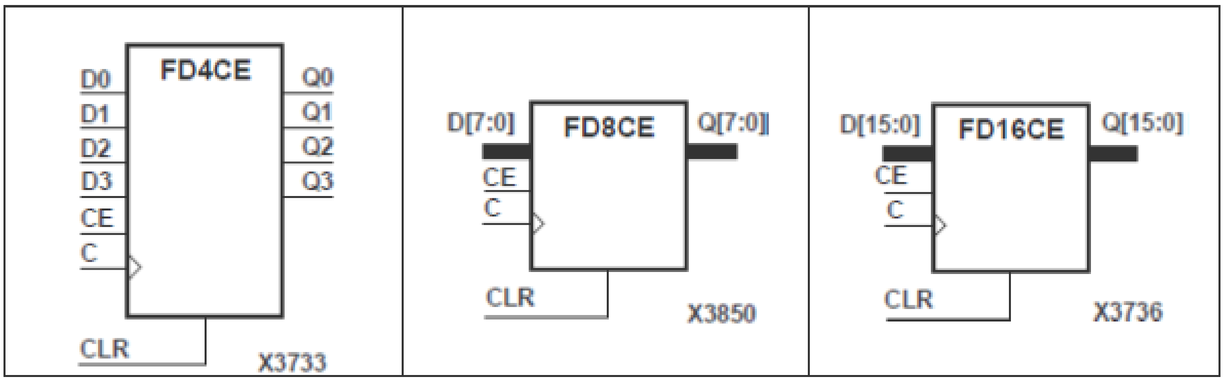

Figure 2. The FD4CE, FD8CE and FD16CE registers

Exercise

- Study the register operation and the function of each input of both the FD4CE register and the FD4RE register.
- In the provided space, write the function table of the FD4RE register.
- What is the difference between the FD4CE and the FD4RE registers? How is this difference indicated in the function tables?

Table 3. Function Table of the FD4RE Register

Inputs				Outputs
CLR	CE	$D_3 - D_0$	C	$Q_3 - Q_0$

3.4. The Bouncing Problem of Mechanical Switches

A big problem with mechanical switches, especially push buttons, is that when the switch is moved to a new position it strikes the metal contact and physically bounces a number of times. This is called contact bounce or simply the *bouncing problem*.

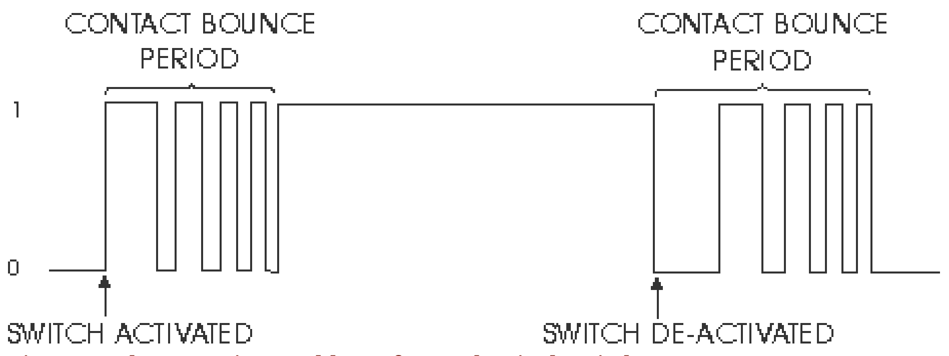

Figure 3. The Bouncing Problem of a Mechanical Switch

If a switch is to turn on a lamp or start a fan or a motor, then contact bounce is not a problem because these devices are slow, and the multiple contacts will effectively appear to the device as a single contact.

If, however, a switch is used as an input to a digital counter, a personal computer, or a microprocessor-based piece of equipment, then contact bounce must be considered. The reason for concern is that the time it takes for the contacts to stop bouncing is measured in *milliseconds* and digital circuits can respond in *nanoseconds*.

Solution

This problem is typically solved by connecting the switch output to a "*debouncing*" circuit which filters out the bouncing pulses, producing a clean single pulse each time the push button is pressed, or the flip switch is flipped.

A Verilog module implementing a debouncing circuit is provided for you in the course website (the `debouncing.v` file). It has the following interface:

```verilog
module debouncer {
    input  wire noisyclk, sysclk,
    output wire cleanclk);
```

The `noisyclk` input should be connected to your bouncy pulse source, whereas the `sysclk` input is the system clock of the FPGA board.

The board’s clock signal is available at pin `V10` on the FPGA. Use it as your input clock signal. Consult the board’s manual for details [nexys3].

The `cleanclk` output is the filtered clean clock generated by the debouncing circuit from the noisy clock.

4. Tasks

In this experiment, you will perform the following tasks:

1. Demonstrate the bouncing problem of mechanical switches (or push buttons) using counters.
2. Use the debouncing circuit with a mechanical push button to demonstrate its effectiveness using the counter.
3. Demonstrate the operation of the register and the function of the `CE` input. Use the same push button as a `clock` pulse input for both the counter and the register. Two independent sets of LEDs should be used to show the outputs of both elements.

4.1. Demonstrate the Bouncing Problem

In this task, a push button is used to manually generate the clock pulses for a 4-bit up counter to observe the bouncing problem of the push button.

1. Use a 4-bit counter component from the Xilinx library.
2. Use a push button on your FPGA board as the `clock` input of the counter, and use four LEDs to observe the counter value.
3. Test your circuit by using the push button several times to pulse the clock for the counter. Observe how the counter reacts to pressing the push button.
   a. What is the expected behavior of the circuit as a result of a single press of the push button?
   b. What is the actual observed behavior?
   c. Explain the difference.

If you get any errors during compilation as a result of using a manual clock signal, add the following command at the end of your UCF file (`.ucf`):

```
NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE
```

The UCF file is a Xilinx-specific way to specify design constraints. It stands for *User Constraints File*. Pin assignments is one type of constraints that can be specified in this file.

4.2. Demonstrate the Debouncing Circuit

1. Modify the counter circuit of the previous task by using the provided debouncing circuit to counter the bouncing problem of the push button.

As you have seen in a previous experiment, you need to create a symbol for the provided Verilog debouncing module in order to be able to use it in your schematic-based design.

2. Modify the UCF file to match the ports of the new design.

The main difference is that the debouncing circuit uses the clock signal provided by the FPGA board (`sysclk`), which is connected to pin `V10` of the FPGA device.

3. Test your circuit by using the push button several times to pulse the clock for the counter. Observe how the counter reacts to pressing the push button. Compare the behavior of this design to the behavior of the design of the previous task (Demonstrate the Bouncing Problem).

4.3. Demonstrate Register Operation

In this task, we will add a 4-bit register to the circuit of the previous task (Demonstrate the Debouncing Circuit). That is, the final design will include a 4-bit counter, a 4-bit register, and the debouncing circuit. We will use the counter to feed data into the register.

The Register and Counter Circuit figure illustrates the end design. The following steps will walk you through creating this design:

1. Create a design that includes the three components.
2. Connect the four count outputs of the counter to the four inputs of the register.
3. Use the same debounced clock signal for both of the counter and the resgister.
4. Use a single switch, or push button, to drive the *clear* inputs of the counter and the register.
5. Use a switch to control the *chip enable* (`CE`) input of the register. The counter should be always enabled.
6. Use a total of eight LEDs to observe the outputs of both of the register and the counter. Connect the outputs according to their significance, i.e. the least significant bit should be connected to the right-most LED.

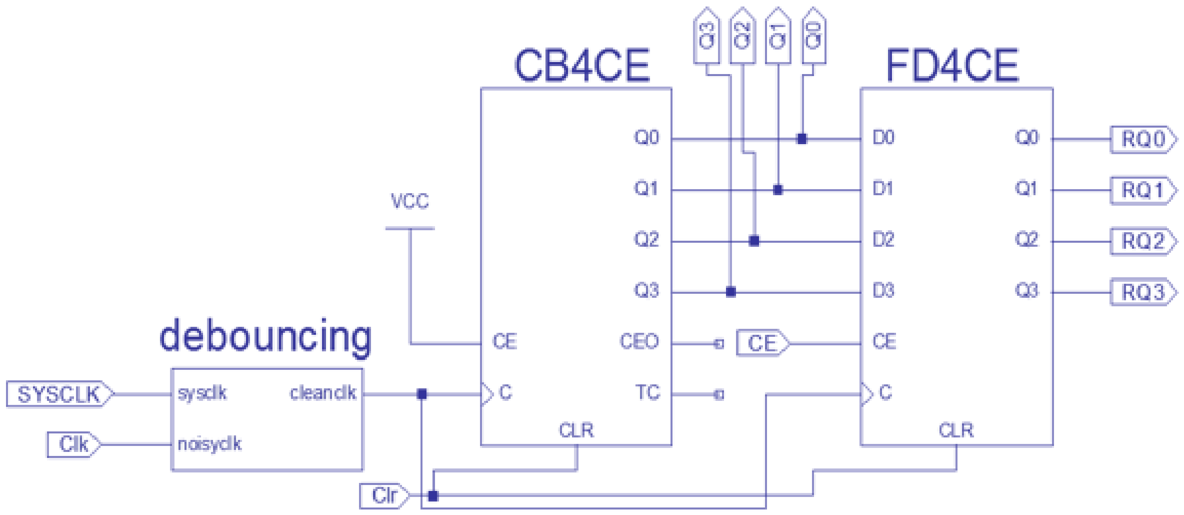

Figure 4. Register and Counter Circuit

Experiment with the inputs of the design: the *clear* switch, the manual *clock* push button, and the *chip* *enable* (`CE`) of the register, and observe the outputs of both the counter and the register for each combination of the inputs.

1. When does the register change its value?
2. What is the relation between the counter current state and the register stored value?
3. Record and justify your observations.

5. Grading Sheet

Task	**Points**
Bouncing problem demonstration	20
Debouncing circuit demonstration	20
Register operation demonstration	35
Lab notebook and discussion	25

6. Resources

[nexys3]

Digilent Inc. *Nexys3 Board Reference Manual.* April 3, 2013. Doc: 502-182. https://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf

Version 151
Last updated 2015-11-06 19:00:27 AST