Experiment 9. Building a Digital Timer

Masud ul Hasan · Ahmad Khayyat – Version 151, 9 April 2015

Table of Contents

1. Objectives

- Design modulo counters
- Use cascaded modulo counters to build a digital timer
- Generate slower clock signals with exact frequencies

2. Material Required

- An FPGA prototyping board.
- Design and simulation software tools.
- A seven-segment display interface module.

3. Background

3.1. Modulo Counters

A modulo-N counter is a counter that counts from 0 to $N-1$. For example, a modulo-10 counter counts from 0 to 9, then it wraps around back to 0. A modulo-10 counter is also known as a BCD counter, for Binary-Coded Decimal, because it counts through the decimal digits.

A modulo-N counter must be wide enough to represent all of its counter values. For example, a modulo-10 counter must be able to represent all the values between 0 and 9, requiring four bits. Therefore, a modulo-10 counter can be implemented using a 4-bit binary counter.

Unlike a typical binary counter, however, a modulo counter may need to go back to zero before it exhausts all its binary combinations. For example, a 4-bit binary counter counts up to 15 (binary `1111`), then naturally resets to zero. A modulo-10 counter, however, must be coerced to go from 9 to 0, despite it being implemented using a 4-bit counter.

The Waveforms of a Modulo-10 Counter figure shows the behavior of a modulo-10 counter. In the figure,

the `C0` through `C3` signals are the counter value bits. Collectively, they represent the values from 0 to 9. The `CE` signal is the *counter enable* input, and `CEO` is the *counter output enable* signal, which allows the counter to indicate to other circuits that it is going to wrap around by the next clock cycle. You can see in the figure that `CEO` is high while the counter is at its last value, 9.

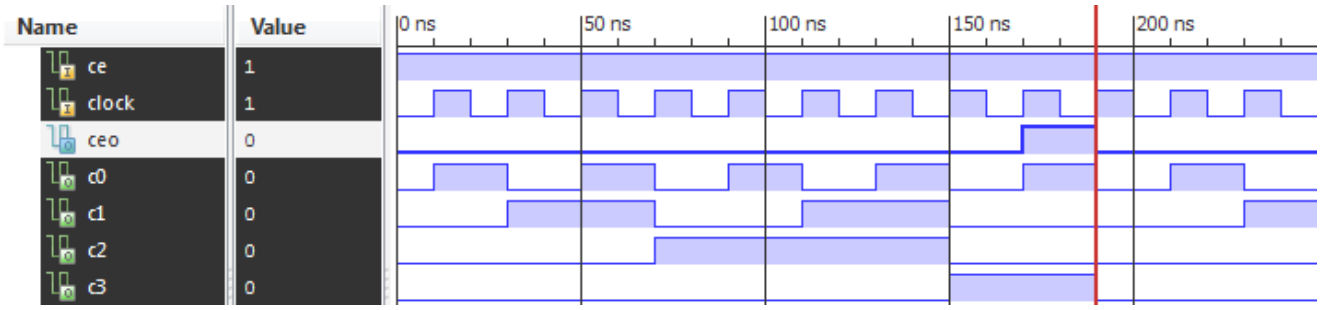


Figure 1. Waveforms of a Modulo-10 Counter

The `CEO` output is also usually known as *carry out*, or `CO`.

To have the modulo-*N* counter reset itself to go from *N* – 1 back to zero, the binary counter needs to be augmented with a circuit that detects the last valid counter value, *N* – 1, and when detected, uses the counter's *clear* input to reset it back to zero.

The input to this circuit would be the current counter value, and the output is a single bit that determines whether to clear the counter or not.

The Schematic of a Modulo-10 Counter figure illustrates how you can build a modulo-10 counter using a 4-bit binary counter. The `CB4RE` component in the figure is a 4-bit binary counter with a reset input (`R`).

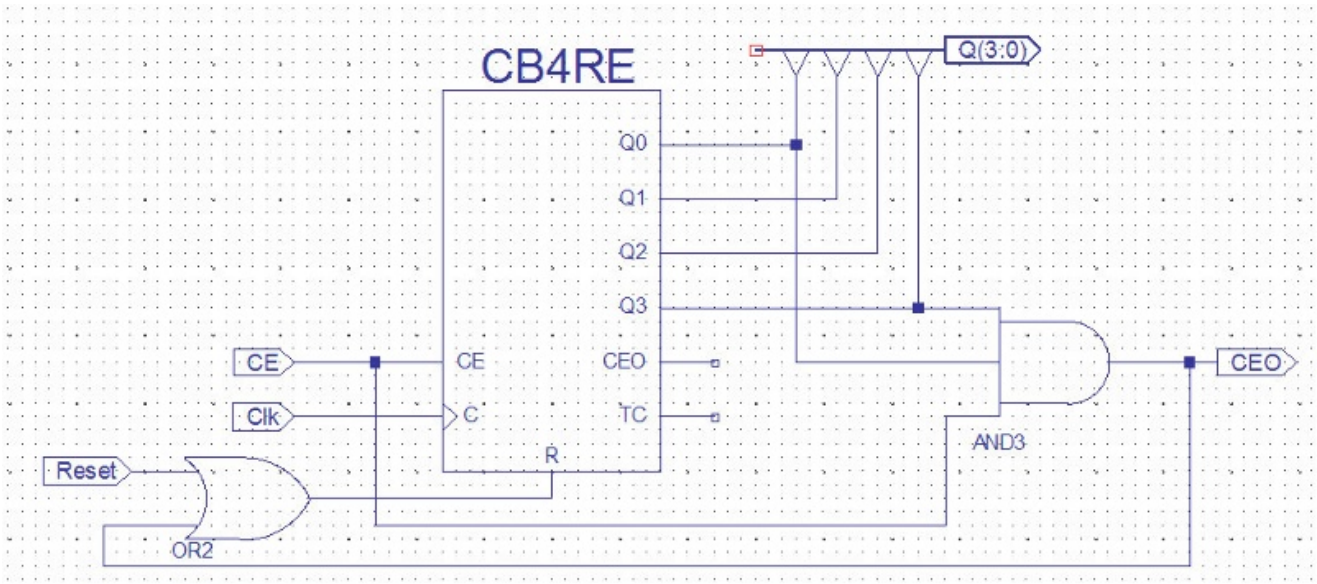


Figure 2. Schematic of a Modulo-10 Counter

Synchronous vs. Asynchronous Clear

The *reset*, or *clear*, input of the counter can be *synchronous* or *asynchronous*. A *synchronous clear* input clears the counter's output, i.e. sets the counter value to zero, on the next edge of the clock cycle. On the other hand, an *asynchronous clear* input clears the counter's output immediately, regardless of all other inputs, including the clock.

The design of the reset circuit in a modulo counter depends on the type of the *clear* input in the used binary counter. Using a *binary counter with synchronous clear* requires a different reset circuit than the one required if a *counter with an asynchronous clear* is used.

In general, synchronous inputs are preferred.

Exercise

From the Schematic of a Modulo-10 Counter figure:

- Extract the Boolean expression for the circuit used to reset the binary counter (the `R` input).
- Explain how that reset circuit transforms the 4-bit binary counter into a modulo-10 counter.

3.2. Pulse Generator with Exact Frequency

In the previous experiment, we used a counter to perform frequency division of a clock signal in order to obtain a slower clock. While frequency division works, it does not allow us to obtain an accurate clock signal for any arbitrary desired frequency, because each generated clock is half as fast as the previous clock.

For example, if we want to generate a 1-kHz clock from a 100-MHz clock using frequency division, then:

$$f_{15} = \frac{100\ \text{MHz}}{2^{15+1}} \approx 1.5\ \text{kHz}, \qquad f_{16} = \frac{100\ \text{MHz}}{2^{16+1}} \approx 763\ \text{Hz}$$

where f_{15} and f_{16} are the frequencies of bits 15 and 16 of the binary counter used for frequency division, respectively. These frequencies correspond to error of 50% and 24%, which is likely to be intolerable for most applications.

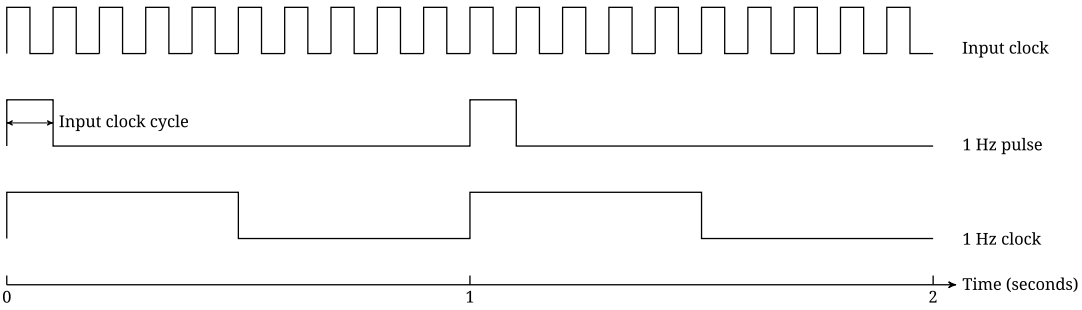
Exercise

What condition must the desired frequency satisfy for it to be possible to obtain accurately using frequency division, given a source clock frequency f?

To generate a signal with a frequency of exactly 1 kHz, or a period of 1 ms, for example, from a 1-MHz clock (period of 1 µs), we need to count the exact number of 1-MHz cycles that make up a single 1-kHz cycle (or the number of 1 µs periods in a single 1 ms period), i.e. 1 MHz / 1 kHz = 1000 cycles. Therefore, we need to build a circuit that counts 1000 cycles, then resets back to zero, and starts counting again, i.e. a modulo-1000 circuit.

Pulse vs. Clock

Using a modulo counter to generate a pulse with an exact frequency will generate a single-cycle pulse with the desired frequency. On the other hand, a frequency division circuit generates a clock signal that remains high for half of its period. That is, it has a 50% duty cycle.



Exercise

What is the duty cycle of a 1-kHz pulse using a 1-MHz input clock?

3.2.1. Building a Modulo-1000 Circuit

Building a modulo counter for a small modulus can be intuitive. With a large modulus, however, it is useful to analyze the problem and solve it systematically.

To count 1000 cycles and then reset, we need:

1. A binary counter that can count from 0 to 999 (1000 values)
2. A circuit that clears the counter after the value 999

Exercise

How wide of a counter do we need to count up to 999?

Because 999 is not a power of 2, we cannot detect its binary representation by simply monitoring a single bit, as is the case with frequency division. In contrast, a *comparator* circuit is required, to compare the counter's value with the binary pattern of the value 999. Once detected, the comparator's output is asserted, and so it can be used to drive the *reset* input of the binary counter.

A comparator can be modeled in Verilog by a single continuous assignment.

Verilog Behavioral Model of a Comparator

```
equal = (count == 999);
```

3.3. Digital Timers

A digital timer is a digital circuit that measures time, like a stopwatch. A timer that counts minutes and hours displays two digits for the minutes, to display values ranging from `00` to `59`, and one or more digits for the hours, depending on the desired range.

Every time the minutes counter wraps around, i.e. goes from `59` back to `00`, the hours counter is incremented. This behavior is called *cascading counters*, where two counters are connected to make up a larger counter.

To cascade a pair of counters, the `CEO` output of the lower-order counter can be used to increment the higher-order counter, by connecting it to the counter's `CE` input.

The Example Cascaded Counters figure shows how two modulo-10 counters can be cascaded to make up a counter that counts from `00` to `99`. Notice that the `CEO` output of the entire circuit is the `CEO` output of the high-order counter only. That is because this modulo-10 counter will assert its `CEO` output only when it reaches its maximum value, `9`, while its `CE` input is also asserted. As a result, the `CEO` output will be asserted only when the overall counter value is `99`.

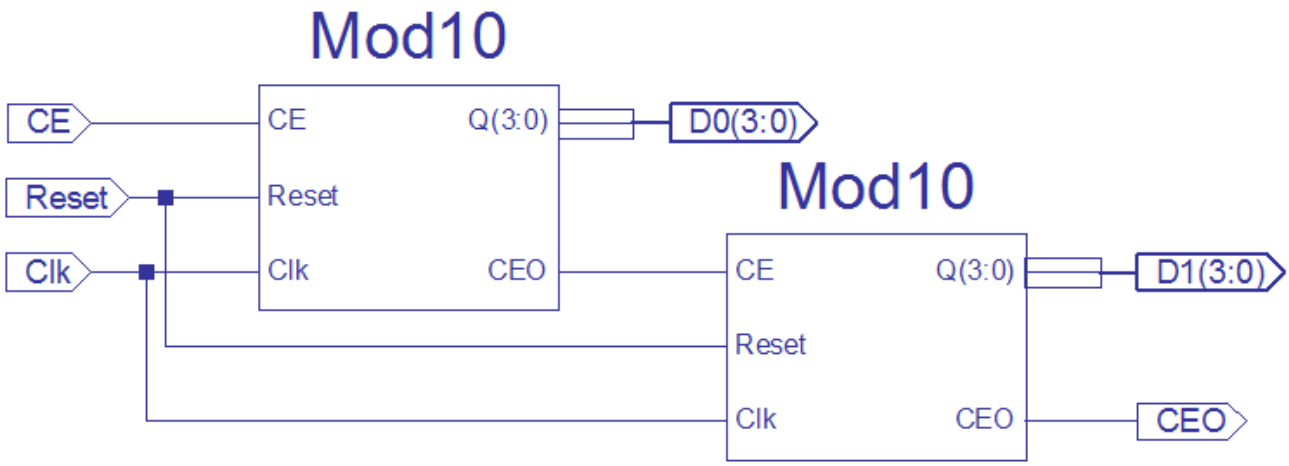


Figure 3. Example Cascaded Counters

Exercise

- How can you count from `00` to `59` using two cascaded modulo counters? What would the modulus for each counter be?
- Would it be easier to use a single modulo-60 counter, or two cascaded modulo counters, to display the counter's decimal value? Explain.
- Suppose that we would like to cascade three counters, when should the `CE` input of the most-significant counter be asserted? Can you formulate your answer as a Boolean expression?

3.3.1. Timer Resolution

The desired resolution of the timer determines the required frequency to drive it. For example, a timer that displays hours, minutes, and seconds, needs a signal with a 1-Hz frequency to drive it, whereas a timer that displays tenths of seconds needs a 10-Hz signal. Such a signal controls the least-significant counter in the timer, which in turn, controls the next counter according to how the counters are cascaded.

Although you may be tempted to use the `clock` input of the individual counters to control their speed, it's generally recommended to connect those `clock` inputs to the main clock signal driving the entire design, without any tampering.

Instead, you can use the `CE` inputs of the counters to control their speeds.

4. Tasks

4.1. Building Modulo Counters

1. Design a modulo-10 counter Verilog module named `mod10`. Simulate it to verify its correct operation.
2. Design a modulo-6 counter Verilog module named `mod6`.
3. Design a modulo-60 counter Verilog module named `mod60` by cascading modulo-10 and modulo-6 counters.

   The module should output two BCD digits representing its current counter value.

4.2. Pulse Generators

1. Design a Verilog module, named `onehz`, that generates a pulse signal with an exact frequency of 1 Hz from a 100-MHz input clock signal.
2. Design a Verilog module, named `tenhz`, that generates a 10-Hz pulse from a 100-MHz input clock signal.

4.3. Building a Digital Timer

1. Design a digital timer Verilog module, named `timer`, that measures minutes and seconds. Minutes should wrap around after `59`:

   Inputs clock, reset, count enable (`CE`)

   Outputs four BCD digits, two for minutes, and two for seconds

   Simulate your module to verify its correct operation.
2. Instantiate your `timer` module in a Verilog module named `timer_nexys3` where:
   a. The `CE` input is driven by a circuit that can generate a 1-Hz or a 10-Hz pulse signal, based on a switch on the FPGA board.
   b. The reset input is connected to a push button on the FPGA board.
   c. The four BCD digits are connected to the four seven-segment displays on the FPGA board.

   Implement your `timer_nexys3` module and test it on the FPGA board.

5. Grading Sheet

Task	Points
Simulate modulo-10 counter	10
Simulate modulo-60 counter, output: two BCD digits	10
Simulate the `timer` module	20
Implement the `timer_nexys3` module on the FPGA board	40
Discussion	20

Version 151
Last updated 2016-04-09 20:30:21 AST