Designing dual-modulus dividers in an FPGA

Brian Boorman - September 28, 2006

It is often necessary for designers to implement a digital clock divider where the output frequency is not an integer factor of the reference clock. Today's newer FPGA technologies usually contain digital or analog PLLs for frequency synthesis. But because of their specifications, these components are usually applicable only for high clock rates. There is still a need for relatively low-speed clock generation—to connect to peripheral devices, for example.

These peripherals usually connect using either synchronous or asynchronous serial interfaces. An example is the standard asynchronous serial port on virtually all PCs and many embedded products. If your design needs to interface to devices with asynchronous interfaces, synchronous interfaces, or both, then you may need several baud rates available. Often, the synchronous and asynchronous rates are different. Choosing an input clock frequency that can satisfy all of the required output frequencies is frequently impossible. In some cases, you may be able to get close to the desired rate for most frequencies, but the error percentage is too large for other frequencies. Such a scenario is an example of where a dual-modulus frequency-divider circuit may be helpful.

This article presents a method for designing a dual-modulus divider. It includes the math for determining the circuit characteristics and offers an example implementation. Although the use of dual-modulus dividers has been around for a long time, joining the work force every year are new engineers that may be unaware of this technique.

A dual-modulus frequency divider is a counter in which the preloaded value can take on two values depending on the current state of a separate sequence counter. Sometimes the counter counts down from (or up to) one number; other times it counts down from (or up to) a second number. By using the sequence counter to control the ratio of usage of one preload value to another preload value, it performs a fractional division on an average, though not instantaneous, basis.

First, the math

Start by developing the math that will allow you to design dual-modulus frequency dividers. Given the following identifiers: M=real divisor, equivalent to $F_{\text{REF}}/F_{\text{DESIRED}}$; N=integer divisor, which you find by rounding M up to the next integer; and A=integer, A+1 is the total number of division cycles. Then,

\[(A\times N)+(N-1)=(A+1)\times M. \hspace{1cm} (1)\]

In this form, Equation 1 is not very useful, because the term you are trying to find, (A), is on both sides of it. Rearranging this equation in terms of A yields:

\[A=[(M+1)-N]/N-M. \hspace{1cm} (2)\]
However, Equation 2 does not yield an answer for all possible input values. You need a more generalized form of the formula that allows the usage ratio of the two moduli to be other than A-to-1. Thus,

\[(A \times N) + (B \times (N-1)) = (A+B) \times M. \quad (3)\]

If you say that

\[A + B = C, \quad (4)\]

then

\[(A \times N) + (B \times (N-1)) = C \times M. \quad (5)\]

Because A, B, C, and N must all be positive-integer numbers, the value of C must be such that the product C×M is also an integer number. The quickest method for determining the value of C is to examine the fractional portion of the real divisor value M; let's investigate that step further.

If you define P as the greatest integer that is less than the real divisor M (in other words, P=floor(M) in C/C++ vernacular), then you can say that

\[D = \frac{1}{M-P}, \quad (6)\]

where D is the inverse of the fractional portion, and then

\[C = D \times E, \quad (7)\]

where E is the smallest positive integer that makes C an integer also. You find the correct value for E either by inspection (recognizing common fractional values such as 0.25 and 0.333) or by trial and error. A spreadsheet program is useful for this task.

Once you solve the equations for C, you know the sequence length of the dual-modulus counter. The sequence length represents the total number of times that the two moduli will be used before the cycle repeats. The next step is to determine the specific values for A and B. You can accomplish this task by solving:

\[(A/C) = M-P, \quad (8)\]

and, solving for A,

\[A = (M-P) \times C. \quad (9)\]

Combining equations 6 and 7 yields

\[C/E = 1/(M-P), \quad (10)\]

and, solving for E,

\[E = (M-P) \times C. \quad (11)\]

As you can see from equations 9 and 11, A=E. From Equation 4, you know that B=C−A. You now have the necessary information to put the circuit together, at least from a mathematical standpoint. Next, work through a practical example to put this methodology to use.
Figure 1 shows the example circuit for implementation. The entire module is synchronous to the clock input, CLOCK. The other main parameters to the design are the moduli N and P, the sequence length C, and the N-versus-P selection parameter B. The output derives from a divide-by-two D flip-flop circuit.

The sequence counter tracks the total number of divisions that the dual-modulus counter has completed. Because the dual-modulus counter counts from N–1 down to zero A number of times and from P–1 down to zero B number of times, the sequence counter counts the total value C (Equation 4). You implement this counter with a downcounter architecture that counts from C–1 to zero. The sequence counter decrements only when the dual-modulus counter has reached its terminal-count value, indicating that a division cycle has completed. When the sequence counter reaches zero and the next terminal-count pulse is received, the sequence counter reloads to the value of C–1.

The count output of the sequence counter routes to a comparator block that determines whether the sequence-count value is less than parameter B. The output of this block is a 1 when the sequence count is less than B; otherwise, the output is a 0. You use this output to select which modulus the dual-modulus counter will use during the next division cycle using the 2-to-1 bus multiplexer that Figure 1 shows.

The dual-modulus counter is simply a free-running downcounter. When the counter reaches zero, it reloads itself with whatever value is present on the LOAD input. The output of this counter is a simple 1 pulse whenever the counter is at zero.

You generate the output of the circuit with a simple output divider. This divider consists of a D flip-flop that inverts its output state whenever the ENBL input is a 1 (also referred to as a T, or toggle, flip-flop). The purpose of this step is to generate a square-wave output clock signal, OUT, from the module. The impact of it, however, is that the output rate from the dual-modulus counter must be twice the desired final output rate. If your design does not require a square-wave output, then you could replace the T flip-flop with a single D flip-flop output register.

Figure 2 shows the relative timing relationships between the various signals in the example design.

Read more In-Depth Technical Features

Once the design architecture is in hand, you can compute the necessary values for generating a 115.2-kHz output from a 10-MHz input suitable for an asynchronous transmitter. Because the example uses a square-wave output, the output of the dual-modulus counter needs to have a rate of 230.4 kHz.

Start by calculating M, N, P, and D:

\[
M = \frac{F_{\text{req}}}{F_{\text{desired}}} = \frac{10 \text{ MHz}}{230.4 \text{ kHz}} = 43.4027.  
\]

\[
N = 44.  
\]

\[
P = N-1 = 44-1 = 43.  
\]

\[
D = \frac{1}{M-P} = \frac{1}{43.4027-43} = 2.462758511  
\]

You now need to find a value for E that satisfies Equation 7. Inspecting the value you calculated for D yields no useful clues as to the value of E, so the method involves trial and error. Using a spreadsheet, you can simply plug in values for E until C is an integer. Figure 3 shows the output of
the spreadsheet calculation. Through this method, you find that $E=29$ satisfies the criteria:

$$C = D \times E = 2.482758621 \times 29 = 72. \quad (13)$$

Finally, the last parameter you need is $B$:

$$B = C - A = C - E = 72 - 29 = 43. \quad (14)$$

*Figure 4* shows a screenshot of the spreadsheet that aids in performing the necessary calculations. The spreadsheet is in Microsoft Excel format and uses the conditional-formatting feature to help visually identify acceptable values.

Now that the detailed design is complete, take a look at an implementation suitable for FPGA development. This implementation uses VHDL (1993-syntax) coding for hardware description.

**VHDL Code Example 1**

*Listing 1* is the VHDL implementation of the example design. The design has been implemented in a manner that enables reuse, which includes defining constants for all the key design parameters. You could further extend this implementation by passing these values into the module using a VHDL generic interface in the entity declaration. The interface is simple for this example, comprising just a synchronous reset input, a clock input, and the output signal.

You then enter the values for $N$, $P$, $B$, and $C$ into the code in the constant declarations in the architecture body. The code uses these constants to constrain the range of the counters that are also declared, which allows the synthesis tool to infer the correct width of the counter in hardware and verifies during simulation that the circuit does not exceed the range of the counters.

The VHDL code is straightforward, contains comments, and is consistent with the design diagram in *Figure 1*.

**VHDL Code Example 2**

*Listing 2* is the VHDL implementation of the testbench for the example design. The testbench is also a simple design, which comprises an instantiation of the example design, labeled UUT (unit under test); a process to generate the input clock; and a process to generate an initial reset to the design. Further verification is by inspection of the simulation-waveform results.

**Simulation results**

*Figure 4* shows a screen capture of the simulation-wave window. This window shows both the testbench-level signals and the example-design internal signals over a simulation duration of 75 msec. The output waveform is at a 50% duty cycle, and the logic correctly switches between the two modulus-counter values, 42 and 43. By zooming in on the actual simulation, you can verify correct operation with respect to the sequence and modulus counters.

This article explores the basic math and concepts behind the use of dual-modulus frequency-division techniques. Once you understand the concept and mathematics, it becomes easy to use this type of function to solve noninteger-division problems. This article touches on only the basic implementation, but as you may envision, you could use the technique in more complex ways, such as by multiplexing the parameters to create a more versatile baud-rate generation, for example.