FPGA-based design yields low-cost arbitrary-frequency generator

Eugene Palatnik - September 03, 2007

Many systems divide a crystal frequency by an integer number to generate clock signals. Unfortunately, the integer divisor limits the available set of output frequencies. Applications such as arbitrary-waveform and -frequency generators and biomedical-signal simulators require low-jitter signals at odd sets of frequencies. For such applications, you can use a phase-accumulator-based frequency generator.

Look at the math. Consider a sinusoidal waveform, which Equation 1 describes:

\[ Y = \sin(wt) = \sin(2\pi f t) = \sin(\Phi) \tag{1} \]

where \( w \) is frequency in radians per second, \( f \) is the frequency in hertz, \( t \) is time, and \( \Phi \) is the phase in radians.

As time increases by \( \Delta t \), the phase increases by \( \Delta \Phi \):

\[ \Delta \Phi = 2\pi \Delta t. \tag{2} \]

From Equation 2, you can define the frequency as:

\[ f = \frac{\Delta \Phi}{2\pi \Delta t}. \tag{3} \]

In Equation 3, you can consider \( \Delta t \) to be a sample clock, or a clock period, which the crystal oscillator defines. Thus, the meaning of Equation 3 is that, to produce frequency, the phase must increase by \( \Delta \Phi \) radians with every sample-clock period.

You can express \( \Delta t \) in terms of the sampling-clock frequency:

\[ \Delta t = \frac{1}{f_{CLK}}. \tag{4} \]

By combining equations 3 and 4, you obtain the equation for the frequency:

\[ f = \frac{\Delta \Phi f_{CLK}}{2\pi}. \tag{5} \]
You can use **Equation 5** to generate an arbitrary frequency. To do it, you need a device that increases in phase by $\Delta \Phi$ radians with every clock period. You can implement such a device as a phase accumulator—a binary counter, which contains a value that increases by $\Delta N$ counts with each clock pulse.

A binary counter with $k$ stages can generate $2^k$ numbers, corresponding to the full $2\pi$-radian period. Similarly, you can replace the phase increase in radians, $\Delta \Phi$, by the phase increase in counts, $\Delta N$. Switching from radians to counts produces:

$$f = \frac{\Delta N f_{CLK}}{2^k}.$$  \hfill (6)

From **Equation 6**, to obtain frequency, you must increase the value in the phase accumulator by $\Delta N$ counts with every clock:

$$\Delta N = \frac{f 2^k}{f_{CLK}}.$$  \hfill (7)

**Circuit implementation**

You can, for example, design a circuit that uses a 24-bit phase accumulator and a 20-MHz-input-crystal clock. Suppose you need to obtain an output frequency of 501.1 Hz. From **Equation 7**, $\Delta N$ is:

$$\Delta N = \frac{501.1 \times 2^{24}}{20{\tera}} = 420 \text{ COUNTS.}$$  \hfill (8)

With each input-clock pulse, the value in the 24-bit counter increases by 420 counts. When the counter rolls over from the maximum $2^k$ value, the counter generates an output-signal one-to-zero transition. Thus, you can use the phase accumulator’s MSB (most-significant bit) as an output frequency.

**Figure 1** shows an FPGA that connects to a microcontroller. The microcontroller calculates $\Delta N$ from **Equation 8** and clocks it into the FPGA using the 8-bit data bus, the 2-bit address bus, and the wr control signal. Because the phase accumulator’s MSB is an output, you can represent $\Delta N$ in $(k-1)$ bits. In other words, $\Delta N$ cannot exceed $2^{k-1}$. The Verilog source code of **Listing 1** implements the FPGA functions. The following two numerical examples reveal how the circuit operates.

The first example requires an output frequency of exactly 10 MHz. From **Equation 7**, calculate $\Delta N = 8,388,608$, or $2^{24}/2$. **Table 1** shows the phase-accumulator values during each clock period and the frequency-generator output—the MSB, $D_{23}$. This example is a simple divide-by-two stage, but look at the next example.

The second example requires an output frequency of 9,999,998.808 Hz. From **Equation 7**, calculate $\Delta N = 8,388,607$, or exactly one count fewer than the previous example’s output frequency. **Table 2** shows that over $2^{24}$ clock pulses, the phase accumulator’s MSB, $D_{23}$, skips one zero-to-one transition and one one-to-zero transition. Thus, the output frequency is 1.192 Hz less than 10 MHz. **Table 2** shows the skips in red.
Now, estimate the output-frequency resolution—how closely you can approach the required frequency. From Equation 6, two numbers, ΔN₁ and ΔN₂, define two frequencies, f₁ and f₂:

\[ \Delta N_1 = \frac{f_2^2 k}{f_{CLK}}; \quad \Delta N_2 = \frac{f_2^2 k}{f_{CLK}}. \]  (9)

The minimum step between ΔN₁ and ΔN₂ is one count. Thus, combining two equations in Equation 9, for the difference between f₁ and f₂:

\[ 1 = \frac{f_2^2 k}{f_{CLK}} - \frac{f_1^2 k}{f_{CLK}} = (f_2 - f_1) \frac{2^k}{f_{CLK}}, \]  (10)

or

\[ f_2 - f_1 = \frac{f_{CLK}}{2^k}. \]  (11)

From equations 10 and 11, you can conclude that frequency resolution does not depend on the target frequency, but on the input-clock frequency and the phase accumulator’s binary length.

Now, look at the dynamic range of the frequencies you can obtain. The range of ΔN numbers defines the dynamic range of the generated frequencies, which range from 1 to \(2^{k-1}\). Applying the above calculations to the earlier example, you can show that the range of frequencies extends from 0.596 Hz to 10 MHz with a resolution of 0.596 Hz.

**Jitter**

Suppose that you want to construct an arbitrary-waveform frequency generator—for instance, an ECG (electrocardiogram)-waveform simulator (Figure 2). The memory stores a set of samples that represent one full period of the arbitrary waveform. A circular counter updates the memory address with every sample-clock pulse that comes from the arbitrary-frequency generator. Thus, with every sample clock, the DAC receives a new data sample and updates its output voltage.

Sample-clock jitter is the difference between the time that the sample pulse should appear and the time that it actually appears. Excessive sample-clock jitter would cause waveform distortions in the DAC output. What kind of jitter do you get by using a phase-accumulator-based sample-clock generator? Previous examples show two output frequencies generated from a 20-MHz clock. For simplicity, imagine that the 20-MHz-input-crystal clock has no jitter.

The first example generates a 10-MHz output frequency. You can expect every output-clock pulse to arrive exactly 100 nsec after the previous one. From Table 1, with every 20-MHz clock pulse, the value in the 24-bit phase accumulator increases by 8,388,608, and, with every input clock, bit D₁₃ of the phase accumulator transitions from either one to zero or zero to one, generating a precise 10-MHz frequency with no jitter.
The other example produced a 9,999,998.808-Hz frequency for which the phase-accumulator contents increased by 8,388,607. Starting from Step 2 in Table 2, with every clock, bit D_{23} transitions from either one to zero or zero to one, exactly as in the previous example. If so, the transitions occur every 50 nsec and not every 50.00000596 nsec, as required. The transitions are thus a little ahead of schedule. The difference between the two timing intervals—the real and the required—is jitter of 0.596×10^{-14} sec. With the next two clocks, this number doubles, then triples, and so on. As Table 2 shows, in 8,388,608 steps, the accumulated jitter is equal to an entire input-clock period, or 50 nsec. The clock skips one input-clock transition, as the table shows in red, delaying the pulse train by 50 nsec. From this analysis, you can conclude that the input-clock frequency defines the phase-accumulator jitter, which, in essence, is a sampling-frequency error. The higher the clock frequency, the less jitter you can expect.

Now, add frequency modulation to the arbitrary-waveform generator. It might be a good upgrade to the ECG-waveform simulator, because the frequency modulation can simulate some cases of arrhythmia—a condition in which the heart rate gradually or rapidly changes. To produce frequency modulation, the microcontroller changes the output frequency by periodically clocking out a new ΔN value into the FPGA. As the FPGA receives the new ΔN value, it starts adding this new value to the phase accumulator with each clock, thus causing the output-frequency change to occur immediately.

A phase-accumulator-based arbitrary-frequency generator is easy to implement in an FPGA. The frequency is easy to control—statically and dynamically. The output has highly predictable frequency resolution and low and predictable jitter, suiting it to many applications.