Create a swept-sine function in LabView with just one virtual instrument

Sean McPeak - June 25, 2009

Swept sine waves are useful when you want to test a product over a wide frequency range. A large research project included the requirement to determine wave propagation in the open ocean. This application required the generation of a swept sine wave to drive an acoustic transducer. Although many waveform generators have a built-in function for this requirement, you must program it yourself if you want to implement a swept sine with a multifunction data-acquisition card. You can create a swept-sine function in National Instruments’ LabView with just one VI (virtual instrument). Using this function, you can control start and stop frequencies, sample rate, and the overall duration of the sweep (Figure 1).

The LabView software calculates an array of numbers that represent the swept-sine-wave time series at each sample point as the frequency either increases or decreases, depending on the direction of the sweep. You must handle the frequency change of the output on a point-by-point basis. The basic form of the equation is \( Y(I) = V \times \sin\left(\frac{A \times I^2}{2} + B \times I\right) \), where \( Y(I) \) is the amplitude of the swept sine wave as a function of the sample point, \( I \) is the integer that steps through the time series, \( V \) is the peak voltage, and \( A \) and \( B \) are variables. You define \( A \) as \( 2 \times \pi \left( f_{\text{STOP}} - f_{\text{START}} \right) / N \), and you define \( B \) as \( 2 \times \pi f_{\text{START}} \), where \( N \) is the number of samples, \( f_{\text{START}} \) is the normalized start frequency, and \( f_{\text{STOP}} \) is the normalized stop frequency. To normalize the start and stop frequencies, you must change the unit to cycles per sample. You accomplish this task by dividing the \( f_1 \) and \( f_2 \) frequencies in hertz by the sample rate. You determine the sample rate by deciding how smooth of a transition you want to represent your swept sine wave. A good rule of thumb is to have at least 10 samples/cycle at the highest frequency. When setting the sample rate, you need to take into account the overall frequency span you are sweeping and the duration of the sweep itself. It is also helpful to compare the results and performance of the LabView data-acquisition-system implementation of the swept sine wave with those of an AWG (arbitrary-waveform generator).

You use two methods of comparison. First, you compare the output of both the data-acquisition and the AWG swept sine wave on a spectrum analyzer. Second, you run them both through an audio-amplifier/speaker system and simply listen to the output. This method is useful in determining sweep rate, duration, and stop and start frequencies. This type of comparison is valid only if the frequencies involved are in the audible range. The LabView VI employs simple array manipulation and uses a “for” loop. The input duration is in seconds, the sample frequency is in samples per second, and the starting and ending frequencies are in hertz. Dividing the sample rate immediately converts the start and end frequencies to cycles per sample. A maximum/minimum block takes the normalized ending and starting frequencies as its inputs and uses the maximum output of the input pair. You use this method to determine whether your design meets the Nyquist criteria, given the sample rate and highest frequency you require.
This approach drives a simple Boolean variable to alert the user about whether the design meets the Nyquist criteria. You set the “for” loop to run for the total number of samples you want to calculate. You determine this value by multiplying the duration in seconds by the sample rate in samples per second. To guarantee that the loop processes all of the generated samples, you must add one, because the loop stops at N-1.

You implement the output function in the “for” loop with simple algebraic operators and the sine block. The output is an array that reaches the perimeter of the “for” loop. It is important to enable indexing at this node. This action allows the circuit to individually handle each element in the array at the output of the “for” loop. You can also add a simple gain stage to set the peak-to-peak value to whatever point you want. Finally, you use the “rotate-1D-array-block” case structure to flip the array if the ending frequency is lower than the starting frequency. This approach handles cases in which you want a frequency sweep that starts out in a higher frequency and descends to a lower frequency.

You can easily modify and expand this simple program. One idea would be to use the output array, which is nothing more than the time series representing a predescribed frequency sweep, to feed a loop that would drive a data-acquisition module. The output of the module should accurately represent the frequency sweep, as long as the module’s output sample rate is the same as the sample rate you use for generating the frequency-sweep time series. You should then be able to track the output samples and, when they are complete, reverse the frequency-sweep array. You then again feed this new flipped array to the data-acquisition module. Depending on the maximum and minimum frequencies, sweep duration, and sample rate you use, it may be difficult to flip the array and configure the module quickly enough to not miss a sample. In that case, you can prefill a frequency-sweep array for a set number of passes.

These modifications allow the sweep to continue back and forth for a set period. Another improvement would be to add some real-time FFT (fast-Fourier-transform) capability so that the user can see the sweep in the frequency domain. This approach also adds an increased level of insurance that the circuit properly meets the sweep definition.