

A Low Distortion, High Accuracy Signal Generator Using the ADSP-218x or ADSP-219x Families

Author: Al Clark, President, Danville Signal Processing, Inc.

Signal processing applications often require accurate calculation of a complex sinusoidal source of the form:

$$e^{j nT} = \cos(nT) + j\sin(nT)$$

For example, suppose we would like to create a tunable bandpass filter using a heterodyning technique. We might use a Weaver modulator. This method translates the incoming signal to DC, low-pass filters the signal, and translates the signal back to its original frequency. In this case, we need to calculate $e^{j nT}$, which requires $\cos(nT)$ and $\sin(nT)$. If these functions are not calculated accurately, we create artifacts at the center frequency of our bandpass filter.

Of course, you may just want to create an accurate signal generator at some arbitrary frequency. In this case, you may only need the real part of $e^{j\omega nT}$, that is, $\cos(nT)$.

The methods this article describes use variations of the phase-accumulator method. The phase-accumulator method calculates the angle, $\theta = \omega_0 nT$, required at the next sample time. You can think of the phase accumulator as a modulo counter that counts from 0 to 2π in n steps.

If we have a sampling rate, f_s , and a frequency increment, f_0 , we need a phase accumulator that indexes $n = f_s / f_0$ possible values:

$$n = 2\pi * f_0 * n / f_s$$

For example, suppose we want a generator with 10 Hz resolution using $f_s = 48000$ Hz, then our phase accumulator will point to 4800 sine and cosine coefficients. Our angles are going to be $0, 2\pi/4800, 2 * 2\pi/4800, \dots, 2 * 4799/4800$.

Our next step is to choose the frequency of our generator and determine the necessary phase increment.

$$\theta = f / f_0$$

$$n = (\theta_{n+1} - \theta_n) \text{ MOD } n$$

Using our previous example, if we want to output 310 Hz, we must increment the phase accumulator by 31.

At this point, we have determined the angle, θ , for each sample of our signal generator. Now we are left with the task of calculating sines and cosines.

There are many ways to calculate sines and cosines suitable for signal generation. Taylor series or Chebyshev polynomials are often used where trade-

offs between the number of terms and accuracy must be considered. The Cordic algorithm is another popular method.

Since the ADSP-218x and ADSP-219x DSPs have abundant memory, we are going to discuss various table lookup methods.

All of the methods that follow have these attributes:

1. The method is computationally fast.
2. The method can generate sines and cosines for each with minimal quantization errors.
3. Both sine and cosine outputs are available for each sample period.

Direct-Lookup Method

The direct-lookup method uses an array of cosine values for all possible values of θ . This method is fast and accurate, but may require a considerable amount of memory. For example, a generator with 1 Hz resolution and a 48 kHz sample rate requires a table of 48000 words.

There are many applications where this method is attractive. For example, a very simple generator can be created by using a phase array of four elements:

$$= \{ 1, 0, -1, 0 \} \quad (\text{cosine sequence})$$

This trivial sequence generates a frequency of $f_s/4$ that can be used as a very quick coding trick to test DAC outputs. If you start at the fourth element of this set, you have the sine sequence. More than one quadrature mixer has been built using this approach.

As long as the table length is evenly divisible by four, you can obtain the sine values by indexing the cosine table at an offset of $3n/4$ where n is the size of the cosine array, since $\sin(\theta) = \cos(\theta/2 - \theta)$.

Another good use for the direct-lookup method is when the desired frequency requires a relatively small table. For example, if $f_s = 48000$, a 1000 Hz oscillator requires only a 48 word array.

In principle, we can make the phase accumulator as large as we want. DDS (Direct Digital Synthesis) systems routinely do this, but usually truncate the phase table to a much smaller size than the accumulator. This allows the frequency to be set to very fine increments on the average, at a cost of increased distortion components and phase jitter.

Quarter-Table, Direct-Lookup Method

Assuming that the cosine table is evenly divisible by four, it is apparent that with a little sign and indexing control, only the first quadrant of the cosine table is really needed to generate all the necessary sine and cosine values for all four quadrants. This has the benefit of reducing the table memory requirements by a factor of four with only a modest increase in computation or programming burden.

Multiple Table Lookup Method

The methods discussed previously are easy to implement, but what if you want to generate frequencies that require large tables, for example, calibration oscillators of 1004 or 997 Hz. DTMF tones are also defined as multiples of 1 Hz. If you sample at 48 kHz, the quarter-table, direct-lookup method requires a table of 12,000 words. We can conserve memory by replacing the large lookup table with several smaller tables. We rely on the following trigonometric formulas:

$$\begin{aligned}\cos(\theta + \phi) &= \cos(\theta)\cos(\phi) - \sin(\theta)\sin(\phi) \\ \sin(\theta + \phi) &= \sin(\theta)\cos(\phi) + \cos(\theta)\sin(\phi)\end{aligned}$$

We create a coarse table using phase increments of θ to generate cosine and sine phases that are approximately our desired phase. This table calcu-

lates sines and cosines over 2^n . We generate two fine sine and cosine phase tables each of length f_c / f_0 , where f_c is the coarse frequency step and f_0 is the desired frequency step.

For example, suppose we want to create arbitrary frequencies with 1 Hz resolution using a sampling rate of 48 kHz. We can use a coarse table to generate phases that correspond to frequency generation at multiples of 100 Hz. We then create a sine table of 100 words and a cosine table of 100 words to create fine increments that correspond to frequency offsets in multiples of 1 Hz. This partition of coarse and fine phase angles works, but it is not the best partition choice since we are still dealing with one phase accumulator.

A better partition is created by using 2^n fine cosine and sine steps. We want the coarse table to be evenly divisible by 4 so we can use a quarter-table, direct-lookup method and also generate sines from a cosine table.

For $f_s = 48000$ Hz and $f_0 = 1$ Hz:

$$48000 = 4 * 375 * 2^5$$

Thus, the coarse table has 375 words and each fine table has 32 words.

The beauty of the multiple-table lookup method is that you can repeat the process to create even finer resolutions without sacrificing accuracy. You just need to create a new set of fine tables.

Other Considerations

If you want to use these methods to create very low distortion sinusoids, use double-precision arithmetic and store all of the sine and cosine values into two words each (32 bits).

If we assume 1.15 or 1.31 numbers, $\cos(0)$ or $\sin(\pi/2)$ cannot be represented since our range of numbers are $1-2^{-n}$ to -1 . This means that our sine and cosine tables should have limits of $0x7FFF$ to $0x8001$. This also ensures symmetry of negative and positive values.

If you would like to simplify the multiple-table lookup method with a small cost in accuracy, here is a useful approximation:

If f is small (the fine phase increment), then $\cos(f) \approx 1$ and $\sin(f) \approx f$.

This yields the following approximations:

$$\begin{aligned} \cos(\theta + f) &\approx \cos(\theta) - \sin(\theta) * f \\ \sin(\theta + f) &\approx \sin(\theta) + \cos(\theta) * f \end{aligned}$$

These approximations work well when the ratio of coarse increments to fine increments are not large. This implies that the fine table only defines a small span of angles.

Source Code and Table Calculation

Source code for the ADSP-218x or ADSP-219x family that illustrates these methods may be downloaded at <http://www.danvillesignal.com/applications/generator.htm>

There is also a table generation tool to generate the necessary sine and cosine coefficients.

