Clay S. Turner

Recursive Discrete-Time Sinusoidal Oscillators

Every few years an article emerges that presents a method for generating sinusoidal functions with a digital signal processor (DSP). While each oscillator structure has been developed pretty much on its own, a simple overlying theory has not been presented that unifies all of the various oscillator structures and can easily allow one to look for other potential oscillator structures. We can find some guidance for our quest by first examining classical oscillators.

The German physicist Heinrich Barkhausen, during the early 1900s, formulated the necessary requirements for oscillation. He modeled an oscillator as an amplifier with its output fed back to its input via a phase-shifting network. From this model, he deduced and stated two necessary conditions for oscillation. The Barkhausen criteria, as they are now known, require the total loop gain to be equal to one and the total loop phase shift needs to be a multiple of 2π radians. So, if we are to unify discrete-time oscillator designs into a single theory, we need to find the discrete-time equivalent of the Barkhausen criteria and use it to develop our theory.

But first, we will look at how some very old trigonometric formulas, viewed in a not so usual way, can be utilized for sinusoidal generation. Several oscillators have been designed via this approach. A sum and difference of angles formula written explicitly in recursive form is

$$\cos(\varphi+\theta) = 2\cos(\theta)\cos(\varphi) - \cos(\varphi-\theta). \tag{1}$$

We will refer to this as the "biquad" form. This is also called the "direct form" implementation. If the value θ is viewed as a step angle, then we can immediately see how this formula can be used to calculate the next sample of a sinusoid given two known samples spaced θ apart and a step factor which is just $2\cos(\theta)$. To see how this can be used for iterative generation of a sinusoid, just view the formula as follows:

$$\text{NextCos} = 2\cos(\theta) \times \text{CurrentCos} - \text{LastCos}. \tag{2}$$

Instead of using a single equation for a recurrence relation, a pair of trigonometric formulas may be used. An example that can be used to recursively generate sinusoids is

$$\begin{aligned}\cos(\varphi+\theta) &= \cos(\varphi)\cos(\theta) - \sin(\varphi)\sin(\theta)\\ \sin(\varphi+\theta) &= \cos(\varphi)\sin(\theta) + \sin(\varphi)\cos(\theta).\end{aligned} \tag{3}$$

We will refer to this as the "coupled" form. The coupling is evident in that each equation uses not only its past value but also the past value produced by the other equation. Again θ is used as the step angle per iteration, and this leads to an oscillator output frequency of $\theta f_s/2\pi$ where f_s is the sample rate. Just like the biquad form, the coupled form requires the use of two past values. This turns out to be the case for all sinusoidal oscillators that are limited to the use of real numbers.

Matrix Derivation of Oscillator Properties

Now we desire to find a general enough process that can be used to express both the biquad and the coupled form equations. First we will denote the two past values (these are actually called state variables) as x_1 and x_2 and their "hatted" versions as the new values. Plus we notice that the update equations are linear for both forms, so we can write them in terms of matrix multiplication.

The matrix form of the update iteration for the biquad is

$$\begin{bmatrix}\hat{x}_1\\ \hat{x}_2\end{bmatrix} = \begin{bmatrix}2\cos(\theta) & -1\\ 1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}. \tag{4}$$

Likewise the coupled form's update iteration is written as

$$\begin{bmatrix}\hat{x}_1\\ \hat{x}_2\end{bmatrix} = \begin{bmatrix}\cos(\theta) & \sin(\theta)\\ -\sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}. \tag{5}$$

The interpretation of the matrix iteration is that the column vector on the right-hand side contains the old state values, and when they are multiplied by the rotation matrix, you get a new set of state values. Then for the next iteration the "new values" from

1053-5888/03/$17.00©2003IEEE

the last iteration are used as the "old values" for the next iteration. Thus each iteration is just performed by a matrix multiplication times the state variables. While the idea of matrix math may seem to unnecessarily complicate things, it actually allows us to go and find new types of oscillators.

The term rotation is used since the matrix multiplication can be viewed from a special vantage point as doing a rotation. This special vantage point will be explained in more detail later. A general form that fits both of these aforementioned oscillators is the use of a 2×2 rotation matrix and two state variables. So a general oscillator iteration is written as

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{6}$$

We will now look at a numerical example of an oscillator iteration. (It is neither the biquad nor the coupled form mentioned earlier.) We will use

$$\begin{bmatrix} 0.95 & -1 \\ 0.0975 & 0.95 \end{bmatrix} \tag{7}$$

for the rotation matrix and use

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{8}$$

for the initial state values. A graph of both state variables' values for the first 100 iterations is shown in Figure 1.

Now that we've seen a numerical example and two analytical examples, naturally the question becomes "what are the constraints on the values of the four elements in the rotation matrix and have the matrix still function for an oscillator iteration?" It turns out that there are two constraints, and these are the discrete time equivalent of the Barkhausen criteria. They are

$$\begin{aligned} ad - bc &= 1 \\ |a + d| &< 2. \end{aligned} \tag{9}$$

The first constraint says the determinant of the rotation matrix must be one. This is analogous to saying the loop gain is unity. The second constraint (assuming the first constraint is met) says the matrix has complex eigenvalues. This means the oscillator output will eventually repeat. This is a discrete-time equivalent to Barkhausen's criterion for periodicity. While not obvious, it can be shown from these two constraints that both matrix elements, b and c, must be nonzero; thus the rotation matrix can't be triangular. What we have done here is basically come up with a set of rules that can be easily applied to any 2×2 matrix to see if it can be used to make an oscillator. And soon we will come up with more rules that will allow you to identify the type of oscillator just by looking at its matrix.

Now we will make a brief sojourn into eigenvalues and eigenvectors. "Eigen" comes from German meaning characteristic. The reason we need to make this side trip is that this theory will allow us to perform an adroit factoring of the rotation matrix. And this will allow us to ascertain the oscillator's properties and determine the initial values for the state variables.

We are used to the idea of identity operations such as adding zero and multiplying by one. An analogous question in matrix theory is "Is there a vector x that, when multiplied by a matrix A, results in a scalar multiple, λ, of the original vector?" Mathematically this is written as

$$Ax = \lambda x. \tag{10}$$

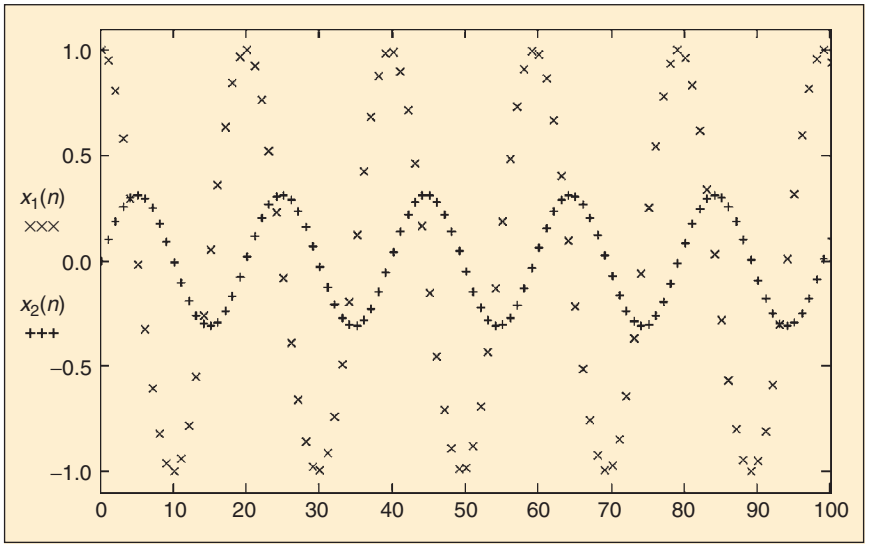
When this is satisfied, λ is an eigenvalue and x is its corresponding eigenvector of the matrix A. For our 2×2 matrices that obey the Barkhausen criteria, there will be two eigenvalues; in fact they are complex conjugates of each other, and each has a magnitude of one. For the rest of this article, the rotation matrix, A, is assumed to be a 2×2 matrix consisting of four real valued elements, and it obeys the aforementioned Barkhausen criteria. Explicitly matrix A is

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}. \tag{11}$$

If we let the vector x contain the initial state values, the nth output, $y(n)$ of the oscillator can be written as

$$y(n) = A^n x. \tag{12}$$

Now we will use a wonderful result of eigenvalue theory and factor the



▲ 1. Output from example numeric oscillator.

general matrix into a product of three matrices. This actually makes raising a matrix to a power much easier to perform

$$A = SDS^{-1}. \tag{13}$$

Thus A^n can be written as $(SDS^{-1})\,(SDS^{-1})\,(SDS^{-1})\cdots(SDS^{-1})$ and after canceling out paired $S^{-1}S$ terms, we get the following simplification for our oscillator output:

$$y(n) = SD^nS^{-1}x. \tag{14}$$

At first blush this doesn't seem to help, but now let's talk about the contents of the S and D matrices. First the D matrix is a diagonal matrix whose only nonzero elements are on the main diagonal (upper left and lower right for our case). These elements are the eigenvalues. Also raising a diagonal matrix to a power is simply the raising of the diagonal elements to the same power. In terms of the original rotation matrix elements, the diagonal matrix is found:

$$D = \begin{bmatrix} e^{j\theta} & 0 \\ 0 & e^{-j\theta} \end{bmatrix} \tag{15}$$

where

$$\theta = \cos^{-1}\left(\frac{\Delta}{2}\right) \tag{16}$$

and

$$\Delta = a + d. \tag{17}$$

It is at this point that we can see how raising the matrix D to a power affects a rotation. In fact, θ, is the step angle of the oscillator per iteration. Now the "change of basis" matrix, S, contains the eigenvectors that correspond to the eigenvalues used in matrix D. Again in terms of the original rotation matrix elements, the matrix S is

$$S = \begin{bmatrix} 1 & 1 \\ \psi e^{j\phi} & \psi e^{-j\phi} \end{bmatrix} \tag{18}$$

where

$$\psi = \sqrt{\frac{-c}{b}} \tag{19}$$

and

$$\phi = \arg(\eta) \tag{20}$$

where

$$\eta = \frac{(d-a) + j\sqrt{4-\Delta^2}}{2b}. \tag{21}$$

It is interesting to comment on the fact that a real-valued rotation matrix is factored into a product of complex valued matrices. However, the implemented oscillators will only use real valued numbers. This brings to mind a saying attributed to the French mathematician Jacques Hadamard: "The shortest path between truths in the real domain passes through the complex domain."

Now the term "change of basis" was mentioned in connection with matrix S. This is used since the matrices S and $S-1$ map between external and internal space. This should be viewed as the state variables undergoing three processes. The first is a transformation to internal space representation. The second process is a pair of rotations performed on them, and lastly, a transformation back to external space. In internal space, the two rotations are in opposite directions; this allows us to combine complex numbers so as to result in only real numbers. Now let the variable z be an internal representation as follows:

$$z = S^{-1}x. \tag{22}$$

Next let x have an initial value so that

$$z = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}. \tag{23}$$

Then our oscillator output is simply written as

$$y(n) = SD^nz \tag{24}$$

that after simplification yields

$$y(n) = \begin{bmatrix} \cos(n\theta) \\ \psi\cos(n\theta + \phi) \end{bmatrix}. \tag{25}$$

Likewise given our initial choice for z, then the initial state values are

$$x = Sz \tag{26}$$

that after simplification yields

$$x = \begin{bmatrix} 1 \\ \psi\cos(\phi) \end{bmatrix}, \tag{27}$$

which is of course just $y(0)$.

Interpreting the Rotation Matrix

The attractive aspect of this analysis method is that we can now evaluate an oscillator's behavior by merely looking at its rotation matrix! We see that our analyses includes two angles. The angle θ is the step angle per iteration, and the angle ϕ is the phase shift between the two state variables. So, we can see that if we desire an oscillator to have quadrature outputs (the two state variables), then ϕ must be $\pm 90°$, which in terms of the matrix elements means that the two values on the main diagonal must be the same! So if $a = d$, we have a quadrature oscillator. Likewise the scaling factor ψ tells us the amplitude of the second state variable relative to the first one. If we desire the two outputs to have equal amplitudes, then the off diagonal elements must be negatives of each other! That is, $b = -c$. So looking back at the matrix used in our numerical example, we can

quickly ascertain the outputs are in quadrature but will have unequal amplitudes as its graph confirms.

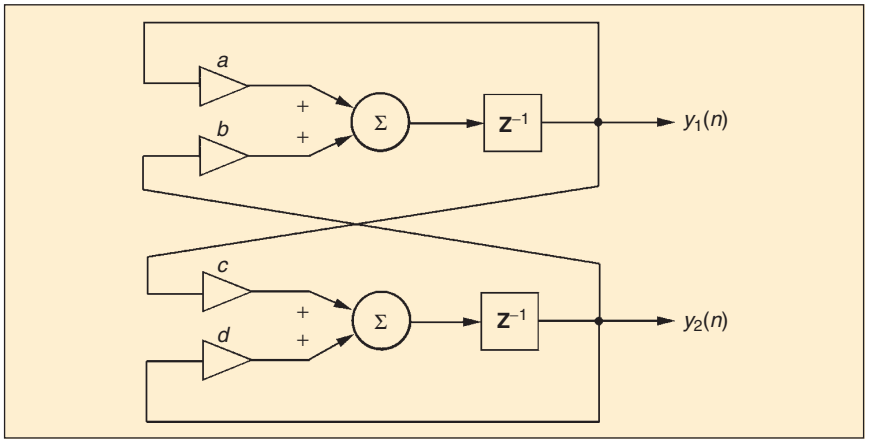
Now when it comes to programming the oscillator in a DSP chip, we would not normally elect to implement the iteration as a matrix multiply; something simpler may be possible! So we will derive the network diagram that represents the general rotation matrix multiplication. In the world of signal processing, it is customary to try to only use addition, multiplication, and delays, and that is all we will use. The centerpiece of the network diagram in Figure 2 is the pair of delay elements that are interpreted to hold the state variables. For each iteration, the outputs of the delay elements are used as the past values and the inputs to the delay elements are the new values. For example, the input to the upper delay element in Figure 2 is calculated:

$$\hat{y}_1 = ay_1 + by_2 .$$

This generic form requires four multiplies and two additions for each iteration. If the rotation matrix has some values in common or some are simply zero or one, then the iteration computations may become simpler.

Catalog of Oscillators

Now that we had gone through the theory of discrete-time recursive oscillators, we will now list some common oscillators along with their attributes. I have chosen five oscillators that span the gamut based on the number of multiplies per iteration and their type of outputs. All oscillators represented by 2×2 matrices will produce two sine waves simultaneously. These two sine waves will always have the same frequency, be out of phase with each other, and may have differing amplitudes. If the outputs are 90° out of phase, then we have a quadrature oscillator. Likewise if the two sine wave outputs have the same amplitude, then we have an equi-amplitude oscillator. Four of the five oscillators in this catalog are in use in industry as they are mentioned in various application notes and trade journals. The quadrature oscillator with staggered update is one I put together using this matrix theory. Often you can make a new oscillator by permuting the matrix elements of a known oscillator. Just apply the Barkhausen criteria to the resulting matrix to see if they still hold. Also you can multiply an oscillator matrix by another matrix and often create another oscillator. So, as you can gather, this catalog hardly exhausts the possible list of oscillators. To show the structure of an oscillator's matrix in its simplest form, the concept of tuning parameter is introduced. This parameter, k, is related to the step angle, θ. The exact relation, which depends on the particular oscillator used, will be provided.



▲ 2. Generic oscillator iteration network diagram.

Biquad

The biquad oscillator was one of the first discrete oscillators to see use in signal processing applications. I recall an application patent issued in the 1980s that used this oscillator for generating call progress tones used in telephony. I found this interesting since François Viète discovered the trigonometric recurrence relation (1) long before; his result was published in the year 1571! The biquad oscillator has equi-amplitude outputs, which turn out to have a relative phase shift of θ

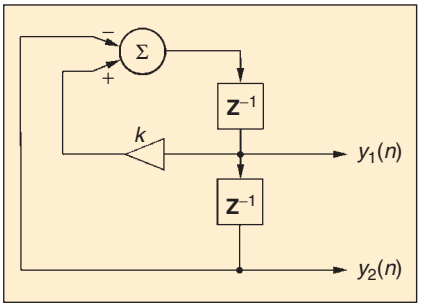
$$k = 2\cos(\theta) \tag{28}$$

$$\begin{bmatrix} k & -1 \\ 1 & 0 \end{bmatrix}. \tag{29}$$

When the rotation matrix (29) elements are substituted in Figure 2, the generic oscillator network becomes the biquad oscillator shown in Figure 3.

Digital Waveguide

The digital waveguide oscillator is the simplest (in terms of the number of multiplies) oscillator with quadrature outputs. For k near zero, the outputs become nearly equal in amplitude. This



▲ 3. Biquad oscillator.

means this oscillator can be effectively used to phase lock a signal near 1/4 the sampling rate. More on dynamic tuning (i.e., changing frequency while in operation) of oscillators later. Figure 4 shows the network form for the digital waveguide oscillator.

$$k = \cos(\theta) \tag{30}$$

$$\begin{bmatrix} k & k-1 \\ k+1 & k \end{bmatrix}. \tag{31}$$

Note that the digital waveguide oscillator is a claimed item under U.S. patent #5701393, so consult the patent's owner before using this oscillator in a product.

Equi-Amplitude, Staggered Update

The staggered update oscillators take their name from the fact that one state variable is first updated and then that new value is used in the update of the remaining variable. This oscillator's outputs are equi-amplitude and quasi-quadrature, with the quadrature relation being reached in the limit of small k. To explicitly show the oscillator iteration as being staggered, its matrix along with a factoring of the matrix is shown. Notice how the matrix factors into a product of two triangular matrices, neither of which can function as an oscillator alone. This oscillator is shown in Figure 5.

$$k = 2\sin\left(\frac{\theta}{2}\right) \tag{32}$$

$$\begin{bmatrix} 1-k^2 & k \\ -k & 1 \end{bmatrix} = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ -k & 1 \end{bmatrix}. \tag{33}$$

Quadrature, Staggered Update

This quadrature oscillator has nearly equi-amplitude outputs when k is small. Again, as before, we show the factoring of its matrix, but here we notice that the right-hand factor is effectively a biquad oscillator. So the left-hand factor is used to change the configuration of the right-hand oscillator. This oscillator is shown in Figure 6.

$$k = \cos(\theta) \tag{34}$$

$$\begin{bmatrix} k & 1-k^2 \\ -1 & k \end{bmatrix} = \begin{bmatrix} 1 & -k \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ -1 & k \end{bmatrix}. \tag{35}$$

Coupled, Standard Quadrature

The coupled standard quadrature oscillator features both quadrature and equi-amplitude outputs. However, there is a cost; this oscillator requires four multiplies per iteration. This oscillator, like the biquad, may be derived directly from trigonometric formulas as shown in (3). Here the two trigonometric functions are written in terms of the single parameter, k. This oscillator is shown in Figure 7.

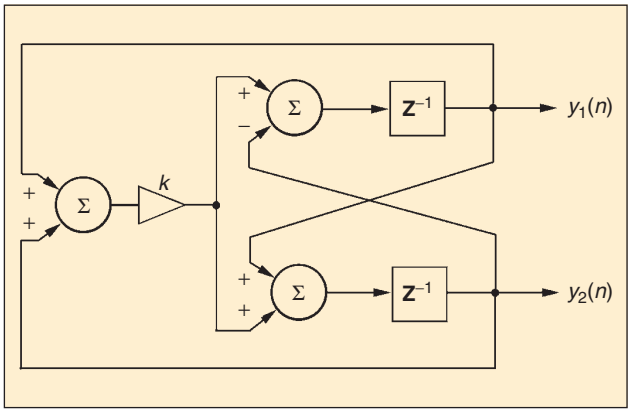
$$k = \sin(\theta) \tag{36}$$

$$\begin{bmatrix} \sqrt{1-k^2} & k \\ -k & \sqrt{1-k^2} \end{bmatrix}. \tag{37}$$

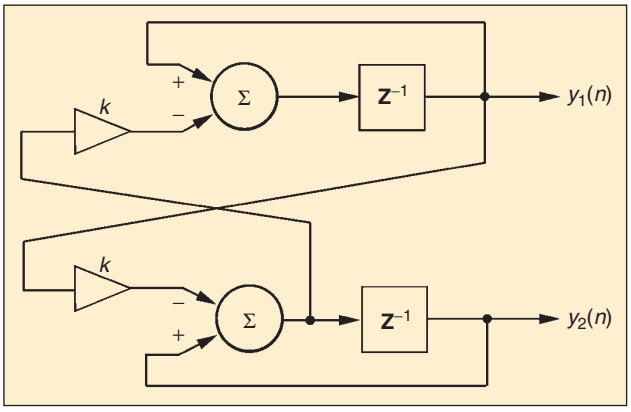
Gathering our catalog of oscillator properties yields the data in Table 1.

Dynamic Amplitude Control

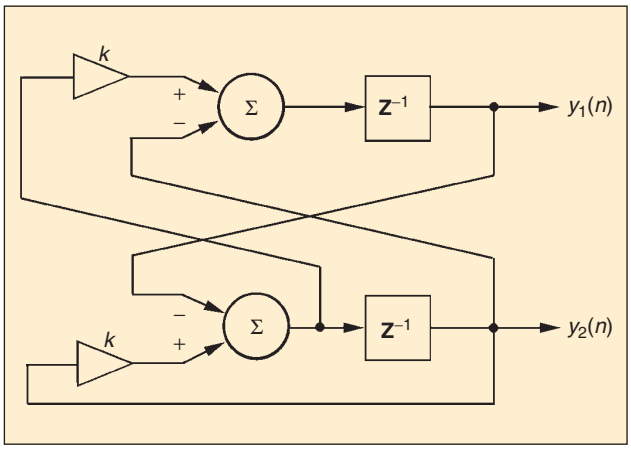
So far the oscillators we have described are ballistic in the sense that they are loaded with some values and allowed to free run. If the run time is "short," then this may be adequate. But errors can and may accumulate to the point where the output no longer meets your requirements. Thus a way of controlling the oscillator's amplitude is needed. A standard approach uses an automatic gain control (AGC). Since we are iterating the oscillator, why don't we just measure the strength of the output and correct it



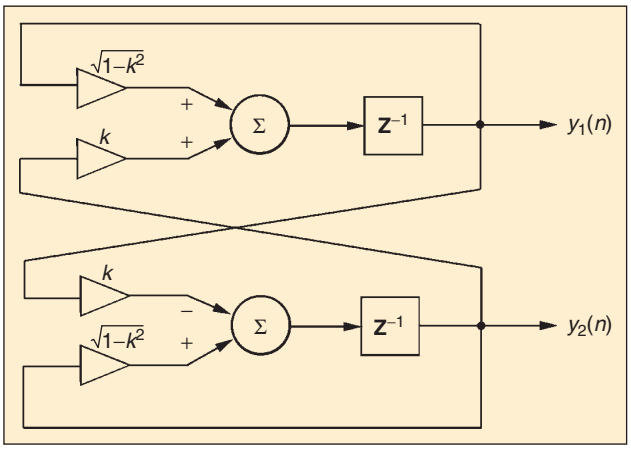
▲ 4. Digital waveguide oscillator.



▲ 5. Equi-amplitude, staggered update oscillator.



▲ 6. Quadrature, staggered update oscillator.



▲ 7. Coupled, standard quadrature oscillator.

after each iteration? Also, if the errors are small, the corrections only need to be approximate.

Now when we are measuring the oscillator's output we will use powers (squares of the amplitude) instead of the amplitudes to avoid square roots. Square root calculations tend to be costly in a DSP. The instantaneous power, P, is given by the following formula (in terms of state variables, matrix elements, and phase shift):

$$P = \frac{x_1^2 - \frac{b}{c}x_2^2 - 2\sqrt{\frac{-b}{c}}\,x_1 x_2\cos(\phi)}{\sin^2(\phi)}. \tag{38}$$

This formula looks painful to implement, but for fixed frequencies, the calculation only involves six multiplies and two subtracts. However, looking ahead to being able to change frequency, we see that if we restrict ourselves to the quadrature oscillators ($\phi \pm 90°$), then the formula for the power becomes much simpler! It is just

$$P = x_1^2 - \frac{b}{c}x_2^2. \tag{39}$$

Along with the power measurement, we need to find the gain needed to properly scale the state variables. A general gain formula is

$$G = \frac{P_0^q}{P^q}. \tag{40}$$

In this formula, P is the measured power, P_0 is the set point power, and q is a convergence factor. Since we rather not perform division, we will use the first-order Taylor's approximation; it has the neat property of turning division into subtraction. It is

$$G \approx 1 + q - q\frac{P}{P_0}. \tag{41}$$

Since we are using G to scale the amplitudes (the state variables), it is best to let $q = 1/2$. Also, when using fixed point DSPs, it becomes convenient to let the set point power also be 1/2 (amplitude≈0.707). Thus our correction formula becomes

$$G = \frac{3}{2} - P. \tag{42}$$

But we still have one more potential problem and that is G will nominally be one, and sometimes we will need to multiply the state variables by a number slightly greater than one. A trick that can be used is to multiply the state variables by 1/2 G and then double the results. So in summary the AGC approach consists of the following steps per iteration.

▲ Perform one oscillator iteration to update the state variables.
▲ Measure the oscillator's output power, P.
▲ Calculate a gain factor, G.
▲ And finally, scale the state variables by this gain factor.

Dynamic Frequency Control

Finally we get to the subject of dynamic frequency control. Since there are some applications where one would like to have a numerically controlled oscillator, we will briefly look at what it takes to do this with these oscillators. Changing an oscillator's frequency merely requires modifying the rotation matrix's k value for any new θ frequency value. The difficult part, however, is maintaining amplitude control during dynamic frequency changes. Since our general power formula (38) shows both a dependence on the frequency and amplitude ratio, this can prove computationally inefficient in the general case. Thus the problem is updating the coefficients in the power formula as the frequency is changed. Plus some oscillators will also change output amplitudes.

If we apply some restrictions to the type of oscillator we can simplify

the situation. If we look at just using a coupled-standard quadrature oscillator, all of these problems go away. In this case the formula for the power becomes independent of the matrix elements altogether! However, the difficulty in this case lies in the matrix coefficients where two of them involve radicals. One can use a first-order binomial expansion for these two terms, but then the determinant is not quite one, so the AGC must make up for the error. The coupled-standard quadrature rotation matrix that uses first-order binomial expansions for the two terms with radicals is

$$\begin{bmatrix} 1-\frac{k^2}{2} & k \\ -k & 1-\frac{k^2}{2} \end{bmatrix}. \tag{43}$$

Equation (43), which can now be used to approximate (37), has a determinant that's nearly one (specifically $1+k^4/4$), so the gain compensation works well for a wide range of k. But even if we use a non-equi-amplitude quadrature oscillator, we can find a simple solution for its amplitude control. In fact we move the approximation in the process from the rotation matrix to the power measurement. An example using the digital waveguide oscillator will now be shown. The power formula for the digital waveguide oscillator in terms of the tuning parameter, k, is

$$P = x_1^2 - \frac{k-1}{k+1} x_2^2 . \tag{44}$$

But knowing that we will be calculating the power on every iteration, and preferring to avoid division, we will use a first-order series expansion of the denominator that gives the following approximation for P:

$$P \approx x_1^2 + (1-k)^2 x_2^2 . \tag{45}$$

This approximation works well when k is small, and k is small for frequencies near one-fourth the sampling rate. If the tuning range needs to be enlarged, a higher-order expansion may be used.

So far we have been making the oscillators easily controllable by using low-order series expansions for the difficult to calculate terms. But these approximations carry a price tag; that is, we either must use the oscillator in a narrow tuning range where k is small, or we must use a higher-order approximation to accommodate a larger range of operation. This results from the limitation that the low-order approximation is only good over a finite range. Sometimes our problem won't really be one of needing a large range of operation, but rather we need to operate over a narrow range where k is centered around some nonzero value. This case is easily handled by the use of a frequency translation matrix. This matrix, in effect, shifts the center frequency of our oscillator to the point of interest. The oscillator's center frequency is taken to be the frequency where k is zero, and thus the approximations are exact at this frequency. The steps for one iteration of a dynamic frequency, amplitude controlled oscillator that uses a frequency translation matrix are as follows:

▲ 1) Perform one oscillator iteration using a tunable oscillator to update the state variables. This can be one of the aforementioned quadrature oscillators.

▲ 2) Update the state variables using the frequency translation matrix. This matrix is simply the same as shown in (37) where k is fixed to represent the translation (shift) frequency. Since the values in this matrix are interpreted to be constant, these values can be calculated prior to runtime.

▲ 3) Measure the oscillator's output power, P. This is just applying (38) or one of its simpler incarnations to the state variables.

▲ 4) Calculate a gain factor, G.

▲ 5) And finally, scale the state variables by this gain factor.

It should be understood that there is only one pair of state variables involved in the previous set of steps. Steps 1, 2, and 5 operate on the one pair of values. And step 3 just uses the same pair to calculate the power. This can be viewed as two oscillators operating on the same state variables. It just happens that one oscillator allows for dynamic frequency control, and the other just shifts the frequency. The networks representing the oscillator iterations are still the same as previously cataloged. Explicitly we can write the translation (shift) matrix as

$$\begin{bmatrix} \cos(\omega) & \sin(\omega) \\ -\sin(\omega) & \cos(\omega) \end{bmatrix}. \tag{46}$$

In a practical implementation, you will set $\omega = 2\pi f_o/f_s$ where f_o is the shift frequency in Hertz, and f_s is the sample rate. The shift frequency may be different than the new desired center frequency, since the natural center frequency for some of the oscillators is one-fourth of the sampling rate.

So what we've learned is that a recipe for recursive oscillator design is the following:

▲ Pick your step angle based on the desired oscillator frequency by using $\theta = 2\pi f/f_s$.

▲ Select the desired oscillator network based on the properties in Table 1.

▲ Define k using its relation to θ in Table 1.

▲ Determine network coefficients from the appropriate rotation matrix in Table 1.

▲ Establish the initial conditions from (27).

▲ Implement the oscillator using the target hardware environment to determine if dynamic amplitude control is necessary.

▲ And if dynamic frequency control is being used, determine if a translation matrix is needed.

FSK Modulator Design Example

For our example, we will highlight the design of a simple modulator for a 1,200 Bd frequency shift-key (FSK) modem. The modem generates either 1,300 or 2,100 Hz depending on whether it is sending a "0" or a "1" bit. We will let our sample rate be 8 kHz, which is common in telephony. Since the data rate does not divide evenly into the sample rate, a sample rate conversion will be needed. This also means the oscillator will generate intermediate frequencies other than 1,300 and 2,100 Hz. This modulator example will benefit from the use of a translation matrix as well.

Looking at our two frequencies, namely 1,300 and 2,100 Hz, we see that we can use a fixed frequency oscillator at 1,700 Hz and let the variable frequency oscillator range between +400 and –400 Hz. So we will use two oscillator matrices in tandem where one has a fixed frequency and the other varies depending on whether we are sending a "0" or a "1." The first oscillator will function as the frequency translation oscillator and hence uses (46) for its work. This oscillator will not change frequency during the modem's operation, so the two trigonometric constants can be calculated prior to use, so we won't be concerned with there being radicals.

The second oscillator that changes frequency while in operation will use the first-order binomial expanded version of the coupled standard quadrature oscillator (43). We are using this oscillator since it allows for easy frequency and amplitude control; remember the amplitude control is required here, somewhat for accumulated numerical errors, but mainly since this matrix (43) does not strictly obey the Barkhausen criteria. Fortunately, the AGC can more than compensate for the oscillator's gain as long as k is small, which means low frequencies with this oscillator, hence the use of the frequency translation in the overall modulator.

Since the input to the modem is a sequence of bits (1,200 b/s), we need to do several things before we let it control the oscillator's frequency. First we need to perform an antipodal mapping of the data, i.e., map "1" bits to a value of +1, and map "0" bits to –1. Next we need to resample these 1,200 values per second to 8,000 per second since this is the modem's sampling rate. Basically a multirate filter is used to interpolate by 20 and decimate by three. The low-pass filter used in this process will be selected to offer intersymbol interference (ISI) rejection. A polyphase-raised cosine filter will suffice. And finally, we will scale the input to the oscillator, so it will emit the correct frequencies. Since we are doing an FM process, the scaling just sets the modulation index. The scaling can be combined into the sample rate conversion process. If a polyphase filter method is used, the coefficients for each of the filters just get scaled to set the modulation index.

The fixed frequency matrix is set to operate at 1,700 Hz, i.e., the average of the two desired frequencies. Numerically, the fixed frequency matrix is just

$$\begin{bmatrix} \cos\left(\frac{2\pi 1{,}700}{8{,}000}\right) & \sin\left(\frac{2\pi 1{,}700}{8{,}000}\right) \\ -\sin\left(\frac{2\pi 1{,}700}{8{,}000}\right) & \cos\left(\frac{2\pi 1{,}700}{8{,}000}\right) \end{bmatrix} \approx \begin{bmatrix} 0.233445 & 0.972370 \\ -0.972370 & 0.233445 \end{bmatrix}. \quad (47)$$

Now for the variable frequency part. Since we desire to deviate between

Table 1. Recursive oscillator properties.

Oscillator	Mult./Iter.	Equi-Amplitude	Quadrature Output	$k =$	Rotation Matrix
Biquad	1	Yes	No	$2\cos(\theta)$	$\begin{bmatrix} k & -1 \\ 1 & 0 \end{bmatrix}$
Digital waveguide	1	No	Yes	$\cos(\theta)$	$\begin{bmatrix} k & k-1 \\ k+1 & k \end{bmatrix}$
Equi-amplitude-staggered update	2	Yes	No	$2\sin(\theta/2)$	$\begin{bmatrix} 1-k^2 & k \\ -k & 1 \end{bmatrix}$
Quadrature-staggered update	2	No	Yes	$\cos(\theta)$	$\begin{bmatrix} k & 1-k^2 \\ -1 & k \end{bmatrix}$
Coupled-standard quadrature	4	Yes	Yes	$\sin(\theta)$	$\begin{bmatrix} \sqrt{1-k^2} & k \\ -k & \sqrt{1-k^2} \end{bmatrix}$

+400 and –400 Hz, we find the scaling for the frequency input to the oscillator by $k = \sin(2\pi 400 / 8{,}000) = 0.309017$. So our ISI filtered antipodal values need to range between ± 0.309017. The frequency input, which is updated 8,000 times per second, becomes the value k used in (43). The two unique values in matrix (43) are calculated 8,000 times per second and then used in the variable frequency oscillator iteration. And of course the initial state values for the oscillator combination are simply

$$\begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix} \approx \begin{bmatrix} 0.707107 \\ 0 \end{bmatrix}. \tag{48}$$

These are found by scaling the result of (27) by $\sqrt{P_0}$, and P_0 is chosen so (42) may be used for amplitude control.

For each iteration then, we just

▲ Antipodal map, resample, ISI rejection filter, and scale the 1,200 b/s data to create "k" for this iteration.

▲ "Matrix multiply" the state variables by the 1,700 Hz fixed frequency matrix.

▲ "Matrix multiply" the state variables by the variable frequency matrix; the value of k was determined above.

▲ Measure the power: $*P = x_1^2 + x_2^2$.

▲ Calculate the gain: $*G = 3 / 2 - P$.

▲ Scalar multiply the state variables by the gain, G.

Summary

We have explored the basic theory of recursive digital oscillators with a bent towards the practical, and from there we have looked at some common oscillators. Then we added some mechanisms for controlling their amplitude and adjusting their frequency. And last we showed a brief example of how these oscillators and their control mechanisms may be used to make FSK modulators. I hope I have piqued the your interests, and I encourage you to go and develop your own oscillators using the rules and techniques presented here.

Acknowledgment

I would like to give credit to those who frequent the USENET group comp.dsp. From time to time questions arise about oscillators, and various group participants have offered information about oscillators that has proved invaluable here for this article.

Clay Turner is vice president and cofounder of Wireless Systems Engineering, Inc. He has over 20 years of experience in digital signal processing, mathematical programming, embedded programming, telephony, and the design of "off-air" protocol analyzers, and he is responsible for mathematical algorithm development at Wireless Systems Engineering, Inc. He is pursuing his Ph.D. and holds a B.S. in mathematics and an M.S. in physics from Georgia State University. He is a member of Pi Mu Epsilon and Sigma Pi Sigma honor societies.

References

[1] J. Diefenderfer, *Principles of Electronic Instrumentation*. Philidelphia, PA: Saunders College Publishing, 1979, pp. 185.

[2] M. Frerking, *Digital Signal Processing in Communication Systems*. Norwell, MA: Kluwer, 1993, pp. 214-217.

[3] S. Friedberg and A. Insel, *Introduction to Linear Algebra with Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1986, pp. 253-276.

[4] R. Higgins, *Digital Signal Processing in VLSI*. Englewood Cliffs, NJ: Prentice-Hall, 1990, pp. 529-532.

[5] S. Leon, *Linear Algebra with Applications*, 2nd ed. New York: MacMillan, 1986, pp. 230-259.

[6] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989, pp. 342-344.

[7] J. Smith and P. Cook, "The second order digital waveguide oscillator," in *Proc. Int. Computer Music Conf.*, San Jose, CA, Oct 1992, pp. 150-153.

[8] J. Smith and P. Cook, "System and method for real time sinusoidal signal generation using waveguide resonance oscillators," U.S. Patent #5701393, Dec 23, 1997.

[9] C. Turner, "A discrete time oscillator for a DSP based radio," in *SouthCon/96 Conf. Rec.*, Orlando, FL, 1996, pp. 60-65.