Digital Signal Processing

18

Digital signal processing (DSP) is one of the great technological innovations of the last hundred years. It has found a permanent place not only in radio, but also in the exploration for oil and other fossil fuels, high-definition television (HDTV), compact-disc (CD) recording and many other facets of our lives. Its popularity stems from certain advantages: DSP filters do not need tuning and may be exactly duplicated from unit to unit; temperature variations are virtually non-existent; and DSP represents the ultimate in flexibility, since general-purpose DSP hardware can be programmed to perform many different functions, often eliminating other hardware. This chapter was written by Doug Smith, KF6DX.

DSP FUNDAMENTALS

In this chapter, you will see that DSP is about rapidly measuring analog signals, recording the measurements as a series of numbers, processing those numbers, then converting the new sequence back to analog signals. How we process the numbers depends on which of many possible functions we are performing. We will take a look at some of those functions and explore how real DSP systems are implemented in software and hardware.

Sampling

The process of generating a sequence of numbers that represent periodic measurements of a continuous analog waveform is called *sampling*. Each number in the sequence is a single measurement of the instantaneous amplitude of the waveform at a sampling time. When we make the measurements continually at regular intervals, the result is a sequence of numbers representing the amplitude of the signal at evenly spaced times.

This process is illustrated in **Fig 18.1**. Note that the frequency of the sine wave being sampled is much less than the *sampling frequency*, f_s. In other words, we are taking many samples during each cycle of the sine wave. The sampled waveform does not contain information about what the analog signal did between samples, but it still roughly resembles the sine wave. Were we to feed the analog sine wave into a spectrum analyzer, we would see a single spike at the sine wave's frequency. Pretty obviously, the spectrum of the sampled waveform is not the same, since it is a step-wise representation.

The sampled signal's spectrum can be predicted and interpreted in the following way. The analog sine wave's spectrum is shown in **Fig 18.2A**, above the spectrum of the string of evenly spaced sampling impulses in Fig 18.2B. The sampled signal is just the *product* of the two signals; its spectrum is the *convolution* of the two input spectra, as shown in Fig 18.2C. The sampling process is equivalent to a mixing process: They each perform a multiplication of the two input signals.

Note that the sampled spectrum repeats at intervals of f_s. These repetitions are called *aliases* and are as real as the fundamental in the sampled signal. Each contains all the information necessary to fully describe the original signal. In general, we are only interested in the fundamental, but let's see what happens when the sampling frequency is *less than* that of the analog input.

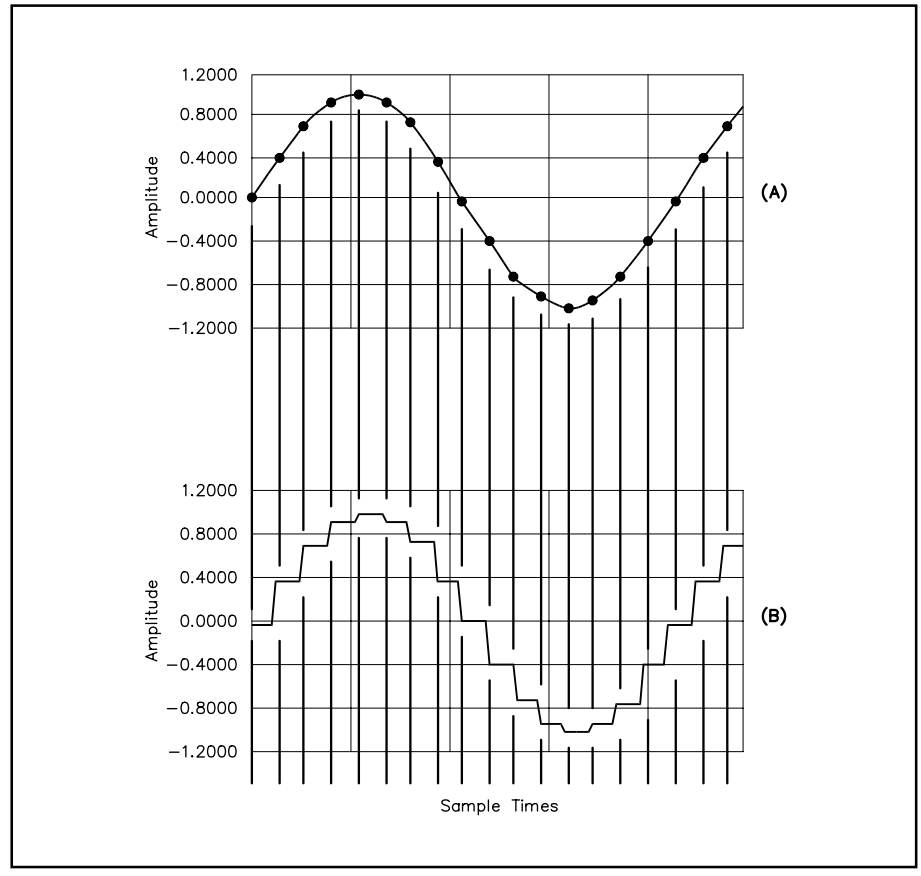

Fig 18.1—Sine wave of frequency much less than the sampling frequency (A). The sampled sine wave (B).

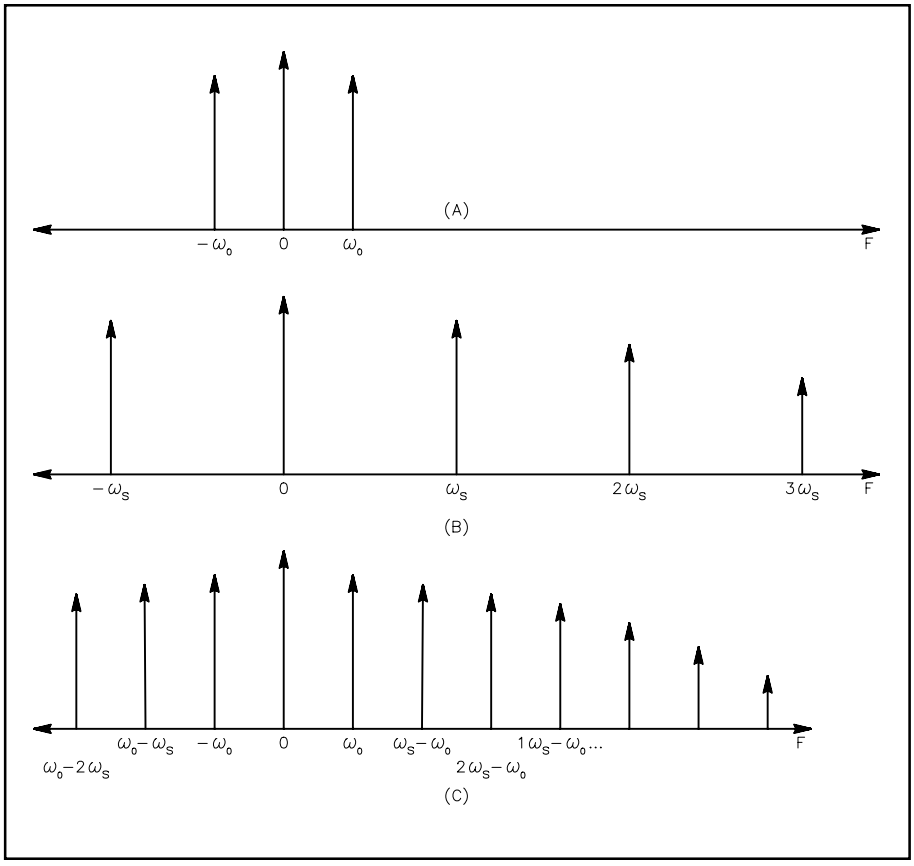

Fig 18.2—Spectrum of an analog sine wave (A). The spectrum of a string of evenly spaced sampling impulses (B). The spectrum of the sampled sine wave (C).

Sine Wave, alias Sine Wave: Harmonic Sampling

Take the case wherein the sampling frequency is less than that of the analog sine wave. See **Fig 18.3**. The sampled output no longer matches the input waveform. Notice that the sampled signal retains the shape of a sine wave at a frequency lower than that of the input. Ordinarily, this would not be a happy situation.

A downward frequency translation is useful, though, in the design of IF-DSP receivers. In addition, lower sampling frequencies are good because they allow more time between samples for signal processing algorithms to do their work; that is, lower sampling rates ease the processing burden. Caution is required, though: An input signal near twice the sampling frequency would produce the same output as that of Fig 18.3. To use this technique, then, we must first limit the BW of the input: A BPF is called for. This is known as *harmonic sampling*. The BPF is referred to as an *anti-aliasing* filter.

Input signals must fall between the fundamental (or some harmonic) of the sampling frequency and the point half way to the next higher harmonic. A frequency translation will take place, but no information about the shape of the input signal will be lost. A spectral representation of harmonic sampling is shown in **Fig 18.4**. It reveals the basis for the often-misquoted *Nyquist sampling theorem*: The sampling frequency must be at least twice the input BW to avoid aliasing. Such aliasing would destroy information; once incurred, nothing can remedy it.

Data Converters and Quantization Noise

The device used to perform sampling is called an *analog-to-digital converter* (ADC). An ADC produces a binary number that is directly proportional to the input amplitude, so only a limited number of amplitude values can be represented. An 8-bit ADC, for example, can only give one of 256 values. This means the amplitude reported is not the exact amplitude of the input, but only the closest value of those available. The difference is called the *quantization error*.

The amplitude reported by the ADC can, therefore, be thought of as the sum of two signals: the desired input and the quantization error. In a perfect ADC, the error cannot exceed ±1/2 of the value of the least-significant bit of the converter—this is the error signal's peak-to-peak amplitude. Assuming the desired input is changing and covers a large range of quantization levels, the error is just as likely to be negative as positive, and just as likely to be small as large. Hence, the error signal is pseudo-random and appears as *quantization noise*.

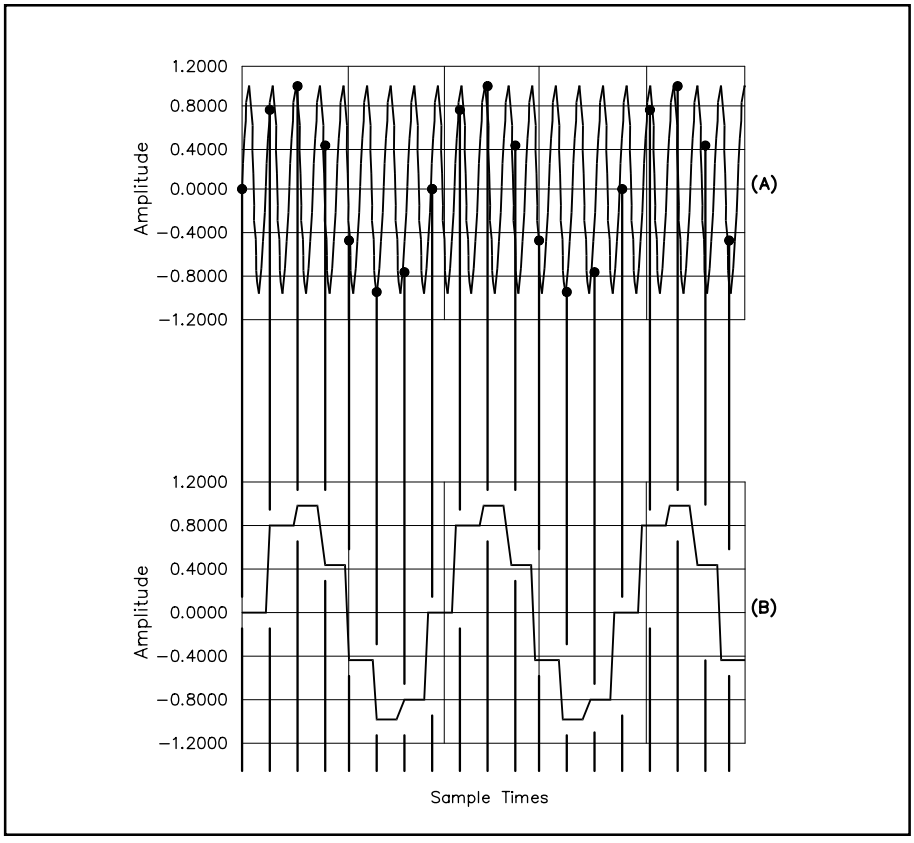

Fig 18.3—Sine wave of frequency greater than the sampling frequency (A). Harmonically sampled sine wave (B).

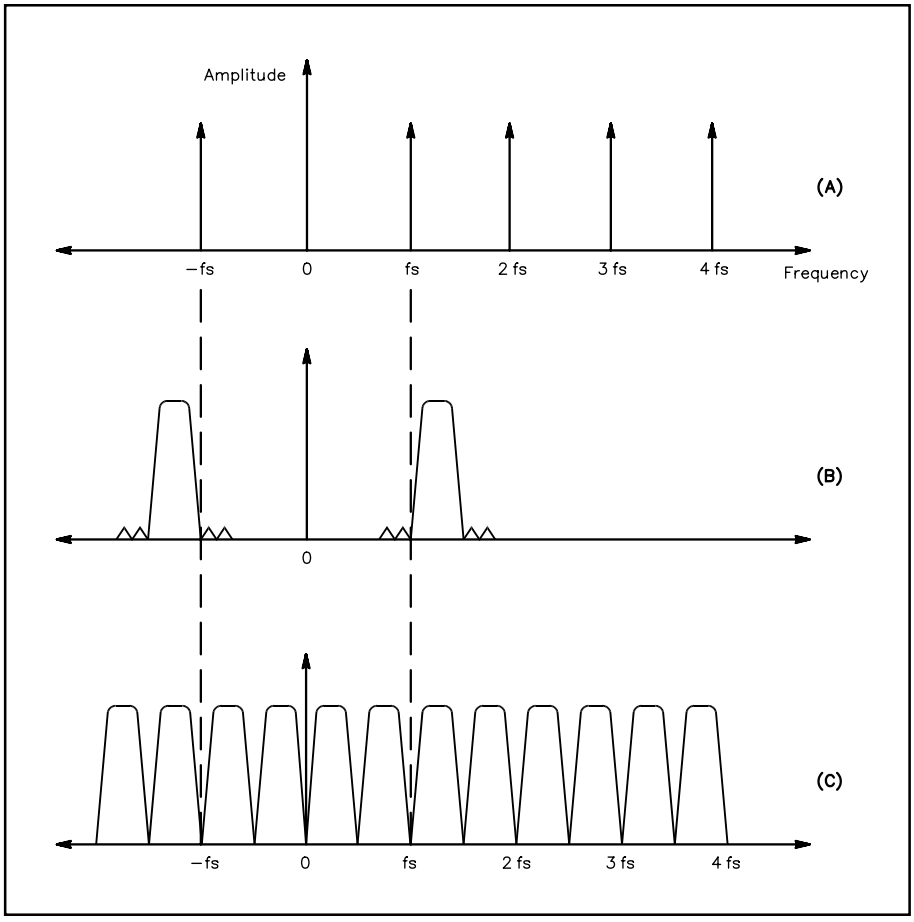

Fig 18.4—Spectrum of sampling impulses (A). Spectrum of a band of real signals (B). Spectrum of a harmonically sampled band of real signals (C).

This noise is spread uniformly over the entire input BW of $f_s/2$. Taking this and the maximum signal the ADC can handle into account, the maximum signal-to-quantization-noise ratio produced by the ADC is:

$$\mathrm{SNR}_{\max} \approx 6.02\,\mathrm{b} + 1.76\,\mathrm{dB} \qquad (1)$$

where b is the number of bits used by the converter.

For a simple 16-bit ADC, the SNR cannot exceed about 98 dB. The reason we wrote that the quantization noise was pseudo-random and not truly random is the following: If there were a harmonic relationship between the input signal and the sampling frequency, the noise might tend to concentrate itself at discrete frequencies, yielding higher power levels than that indicated by Eq 1.

Aperture Jitter

In addition to quantization noise, noise is introduced in ADCs by slight variations in the exact times of sampling. Phase noise in the ADC's clock source, as well as other inaccuracies in the sampling mechanisms, produce undesired phase modulation of the sampled signal. Again, assuming it is uncorrelated with the input signal, this *aperture jitter noise* will be distributed across the entire input BW. Its amplitude is proportional to the squares of both the desired signal's frequency and the RMS time jitter in the sampling rate, and inversely proportional to the sampling rate itself. With contemporary crystal-derived clock sources, aperture jitter is usually not a significant factor until the sampling frequencies reach VHF; even at those frequencies, the effect may be small compared with quantization noise.

Over-Sampling and Sigma-Delta ADCs

The nature of the above-mentioned noise sources is such that if we could increase the sampling frequency by some factor N, then digitally filter the output back down to a lower rate, we could improve the SNR by almost the factor N. This is because the noise would be spread over a larger BW; much of the high-frequency noise would be eliminated by the digital filter. This technique is called *over-sampling*.

So-called *sigma-delta* converters use this method to achieve the best possible dynamic range. They employ one-bit quantizers at very high speed and digital decimation filters (described later) to reduce the sampling frequency, thus improving SNR. They represent the state of the art in ADC technology. Other factors, such as the noise figure of analog stages inside an ADC, tend to limit the SNR of real converters to within a few dB of that calculated by Eq 1.

Non-Linearity in ADCs

The quantization steps of a real converter are not perfectly spaced; conversion results are contaminated by the inaccuracy. In general, two types of non-linearity are characterized by manufacturers: *differential non-linearity* (DNL) and *integral non-linearity* (INL).

DNL is the measure of the output non-uniformity from one input step to the next. It is expressed as the maximum error in the output between adjacent input steps as measured over the entire input range of the device. The worst errors usually occur near the middle of the scale. Since we are talking about the accuracy of the smallest steps the converter can resolve, noisy low-order distortion products caused by this effect limit dynamic range. Current technology uses correction systems to compensate for temperature variations that would otherwise further degrade performance.

An ADC is considered *monotonic* if a steady increase in the input signal always results in an increase in the output. Device manufacturers hold DNL to ± 0.5 bits or better so that monotonicity is maintained.

INL is a measure of an ADC's large-signal handling capability. To measure it, we first inject a signal of amplitude A and measure the output; when we inject a signal of amplitude 100A, we expect the output

to grow in exact proportion. INL represents the maximum error in the output between *any two* input levels. Another way to think about this is to plot the input against the output and see how straight the line is. INL produces harmonic distortion and IMD; values for typical converters are ± 1 or 2 bits over the entire range.

Spurious-Free Dynamic Range and Dithering

Spurious-free dynamic range (SFDR) is defined as the ratio of the largest signal the converter can accurately handle to the largest source of noise and distortion caused by effects mentioned above. Quite often, undesired components may appear in unexpected parts of the input spectrum; spurious responses may be found without apparent explanation. It turns out there are explanations, of course, but we will defer that discussion. Suffice it to write here that manufacturers test for SFDR and usually specify it on their data sheets, especially for high-speed devices.

Sometimes noise and distortion effects conspire to add at discrete frequencies. It is found that the addition of random noise at the clock input helps dissipate these spurious responses. This technique is known as *dithering*. It may seem strange, but artificial noise—usually several bits in amplitude and high enough in frequency to be eliminated by the decimation filter—actually reduces quantization noise and improves performance rather than degrading it. This is discussed in more detail later.

Digital-to-Analog Converters: Additional Distortion Sources

Digital-to-analog converters (DACs) perform the conversion of binary numbers back into analog voltages—the inverse operation of ADCs. They suffer from all the inadequacies described earlier, as well as a few of their own. The first unique distortion of DACs is one of frequency response: *zero-order sample-and-hold distortion*.

Typical converters are sample-and-hold devices: They continue to output the last sampled value throughout the sample period. This effect acts as a low-pass filter having a frequency response:

$$H_f = \frac{\sin\left(\frac{\pi f}{f_s}\right)}{\left(\frac{\pi f}{f_s}\right)} \tag{2}$$

The high-frequency roll-off is quite undesirable in many circumstances. For example, if the output frequency is one quarter the sampling frequency, an attenuation of about 1 dB will occur. Correction can be made for this, but an increase in sampling frequency reduces the attenuation. Interpolation of the sampled output signal (described later) is called for in many cases.

Settling Time and Glitch Energy

When the output of a DAC changes from one voltage to another, it obviously cannot do so instantaneously; a finite time is required for the voltage to reach its new value. This is known as the *settling time*. It is usually defined as the time required to settle to within some number of voltage-equivalent bits of the final value.

Glitch energy or *glitch area* is defined as the product of the voltage error during the settling time and the settling time itself. While volt-seconds are not units of energy, it is assumed the DAC is driving some kind of load; thus, these units can be translated into units of energy (watt-seconds), performing work on that load. The settling mechanism is an important factor in the production of spurious outputs in DACs. Manufacturers usually specify the glitch energy for their high-speed devices. It is an especially important number for direct-digital-synthesis (DDS) applications.

Note also that DACs produce aliases, again repeating at intervals of f_s. These must usually be removed

using an analog LPF. Occasionally, a BPF may be used, and one of the aliases taken as the desired output. This can be a clever way of getting an upward frequency translation under certain conditions.

Reducing the Sampling Frequency: Decimation

As we have seen, sampling at high rates is beneficial because it eases the design of the analog filters we must use to avoid aliasing. It also reduces quantization noise and aperture jitter. We have also noted that lower sampling rates help reduce the computational burden in DSP systems. In addition, we will discover that when it is time to digitally filter some signals, making the filter's BW a large fraction of the sampling frequency makes it easier to build sharp-skirted filters—exactly what DSP is famous for.

Reduction of the sampling frequency is usually called *decimation*. Decimation is normally done by integer factors (although it does not have to be) and is equivalent to resampling the already-sampled signal at a lower rate. The resampled signal has a family of aliases, repeating at intervals of the lower sampling frequency; we have to reduce the BW to less than half this lower sampling frequency to avoid the aliasing that would destroy information.

The process of decimation is simple: Just throw away the unwanted samples. To decimate by two, for example, only every other sample is retained. A *decimation filter*, operating at the higher sampling rate, f_s, reduces signal BW to less than $f_s/4$ prior to discarding the samples to avoid aliasing. But why spend time computing filter outputs that we are only going to discard? We may compute only those we intend to keep. This is exactly the same as running the decimation filter at the lower rate. This method is typical of those used by DSP designers to save time and effort. See the chapter Appendix for a software project (Project A) that demonstrates decimation using Alkin's *PC-DSP* program. This program is included with the book listed in the Bibliography.

Increasing the Sampling Frequency: Interpolation

We learned that when it is time to convert back to analog, an artificial increase in sampling rate may be advantageous. It will push aliases higher in frequency where they are easier to remove by analog filtering, and it will relieve some of the sample-and-hold distortion. So, even having decimated the data at some earlier stage in our designs, we may later employ the process of *interpolation*.

Decimation was performed by deleting samples. Interpolation is performed by inserting them. The inserted samples have a value of zero and are placed between the existing samples. While this increases the sampling frequency, the information in the original samples is not destroyed; however, new information is added in the form of aliases, and an *interpolation filter* is usually required. This filter, most often a low-pass, operates at the higher sampling frequency, f_s, and eliminates components in the interpolated data above $f_s/4$.

The way numbers are represented in DSPs is a major consideration. Let's take a look at this before moving on to filtering algorithms.

Representation of Numbers: Floating-Point vs Fixed-Point

One of the things that makes general-purpose computers so useful is their ability to perform *floating-point* calculations. In this form of numeric representation, numbers are stored in two pieces: a fractional part, or *mantissa*, and an exponent. The mantissa is assumed to be a binary number representing an absolute value less than unity, and the exponent, a binary integer. This approach allows the computer to handle a large range of numbers, from very small to very large. Some DSP chips support floating-point calculations, but it is not as great an advantage in signal processing as it is in general-purpose computing because the range of values we are dealing with in DSP is limited anyway. For this reason, *fixed-point* processors are common in DSP.

A fixed-point processor treats numbers as just the mantissa and does away with the exponent. The radix point—the separation between the integer and fractional parts of a number—is usually assumed

to reside to the left of the most-significant bit. This is convenient, since the product of two fractions less than unity is always another fraction less than unity. The *sum* of two fractions, though, may be greater than unity: *overflow* would be the result. Overflow is a constant concern for fixed-point DSP programmers and leads to considerations for *scaling* of data, as discussed further below, which may limit system dynamic range to less than the data converters' capabilities.

DSP ALGORITHMS FOR RADIO

Digital Filters

The ability to construct high-performance filters is probably the most important rationale for using DSP in radio transceivers. An expensive crystal or mechanical filter having a single BW can be replaced by a set of superior digital filters, offering as many BWs as the associated on-board memory can support.

As shape-factor requirements get more stringent, filters get more complex. As a filter gets more complex—with additional inductors and capacitors in the analog case, or additional delay elements in the digital case—the sensitivity of the filter's response to errors in the element values becomes more severe. Thus, for analog filters, precise values of resistance, inductance and capacitance must be maintained if the filter is to operate as designed. Establishing those values is difficult; holding them within tolerances over temperature variations and aging is more so. DSP filters, on the other hand, are unchanging. The "component" values are numbers stored in a computer that are not susceptible to temperature changes or aging. Filters that would be impractical or impossible in the analog realm are easily implemented by DSP algorithms.

We can build digital filters having linear phase responses, which is very difficult in the analog world. This is an advantage mainly for digital communication modes such as FSK and PSK. Also, filters may be combined numerically to yield composite responses without the need for adding hardware. This is useful for passband tuning or graphic-equalizer applications.

DSP filters are usually characterized by their *impulse responses*. The impulse response of a digital filter is the output of the filter when the input is a one-sample, unity-amplitude impulse. Impulse response is directly related to frequency response by a *Fourier transform*, about which we will learn more later. Suffice it to write for now that digital filters may be broadly divided into two classes: finite impulse response (FIR) and infinite impulse response (IIR). The presence or absence of feedback separates the two.

FIR FILTERS

Take a look at the block diagram of the FIR filter shown in **Fig 18.5**. The string of boxes labeled z^{-1} is simply a delay line, with each box representing a one-sample delay. Programmers will note that

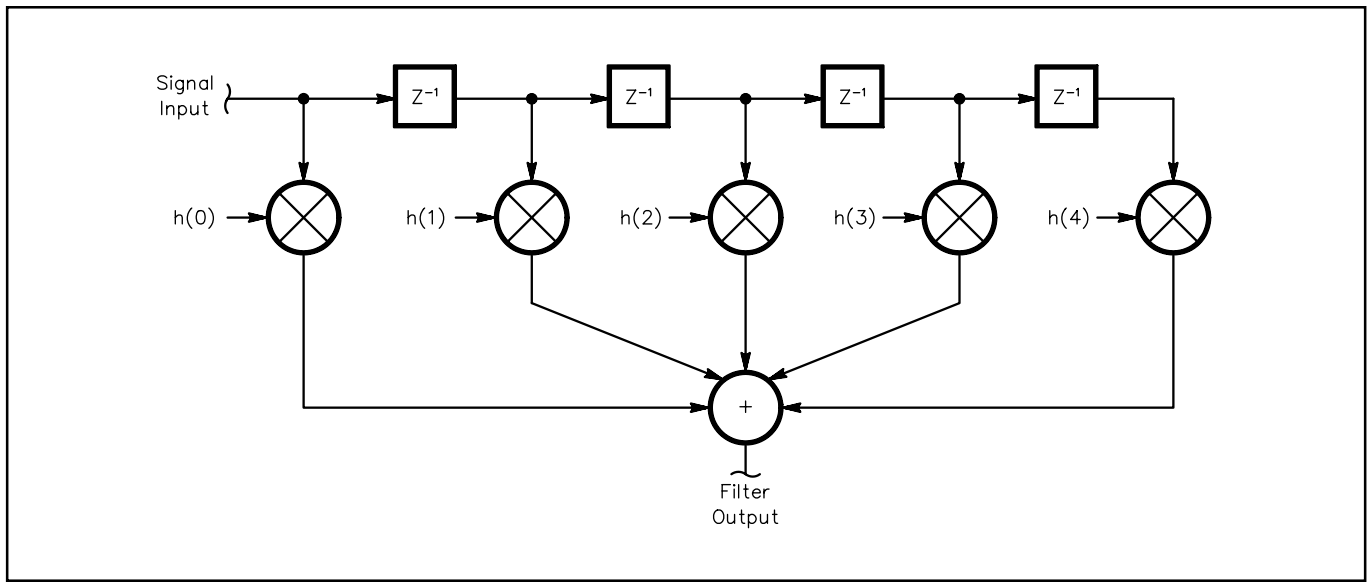

Fig 18.5—Block diagram of an FIR filter for L = 5.

with one input sample in each position, this is just a buffer of length five. Each buffer location may be referred to as a *tap* in the delay line. The datum at each tap, x(n), is multiplied by one of the filter *coefficients*, h(n). All the products are summed at each sample time to produce the filter output. At the next sample time, samples are shifted down the delay line by one position and the *multiply-and-accumulate* (MAC) operation is performed again. Coefficients remain in place and do not shift. The mathematical expression describing this repetitive MAC operation is also called a *convolution sum*:

$$y(k) = \sum_{n=0}^{L-1} h(n)x(k-n) \tag{3}$$

where x(k-n) represents the input data in the buffer.

Since the output depends only on past input values, the filter is said to be a *causal process*. Since no feedback is employed, it is unconditionally stable.

In an FIR filter, the set of coefficients, h(n), is identical to the impulse response of the filter. The trick, then, is to find the impulse response that gives us the frequency response we want. Almost any frequency response can be generated if we use enough taps. In general, low shape factors (steeper roll-offs) require more taps. Most filter-design methods begin with an estimate of the number of taps needed. Rabiner and Gold indicate the estimate may be taken as:

$$L = 1 - \frac{10\log(\delta_1\,\delta_2) - 15}{14\left(\frac{f_T}{f_S}\right)} \tag{4}$$

where δ_1 is the passband ripple, δ_2 is the stopband attenuation, f_T is the transition BW, f_s is the sampling frequency, and L, the number of taps, is called the length of the filter. This equation assumes that enough bits of resolution are used to achieve the required accuracy. In practice, filters of over 100 taps are used to realize shape factors of less than 1.15.

Normally, an FIR filter's impulse response has a symmetry about center; that is, h(0)=h(L–1), h(1)=h(L–2), and so forth. It turns out this is sufficient to ensure a linear phase response and flat group-delay characteristics. The total delay through an FIR filter of length L is:

$$t = \frac{L}{2f_S} \tag{5}$$

As noted, this delay is *independent* of frequency. Remember that longer filters demand more processing than shorter filters.

When personal computers are used to design FIR filters, coefficients are usually represented in floating-point format to the full accuracy of the computer—often with 12 or more significant figures in the mantissa. Embedded, fixed-point DSP implementations ordinarily achieve only 16-bit accuracy. The *truncation* of coefficients and data to this accuracy affects the frequency response and ultimate attenuation of filters, and may be the factor determining dynamic range. Also notice that when we multiply a 16-bit coefficient by a 16-bit datum, the product is a 32-bit number; we are then adding several 32-bit numbers in the final accumulator of an FIR filter. The result may grow by several more bits to 35 or so by the time we are done. At some stage, the result may overflow the accumulator, especially in FIR filters with small transition BWs (sharp skirts). The worst-case output can grow as large as the sum of the absolute value of all the coefficients:

$$y_{max} = \pm \sum_{n=0}^{L-1} |h(n)| \tag{6}$$

We might have to scale the data, the coefficients, or both by the reciprocal of this number to avoid overflow.

The filter output at each sample time is usually *rounded* back down to the bit-resolution of the DAC; say, to 16 bits. The rounding operation introduces a small error in the result. This rounding error is directly analogous to quantization noise; it is computed in almost exactly the same way. A trade-off exists between the possibility of overflow, which is catastrophic, and loss of accuracy because of rounding. It is interesting to note that truncation of filter coefficients affects the frequency response of the filter, but not the amount of noise in the output. On the other hand, truncation and rounding of data do not affect the frequency response but add quantization noise to the output.

One FIR filter-design approach takes advantage of the fact that a filter's frequency response is the Fourier transform of its impulse response. Thus, we may start with a sampled version of the frequency response and apply an *inverse* Fourier transform to obtain the impulse response. All filter-design software is capable of using this method. Better designs may be obtained in many cases by using an algorithm developed by Parks and McClellan. This approach produces an *equi-ripple* design in which all of the passband ripples are the same amplitude, as are all the stopband ripples. Another popular algorithm is the *least-squares* method. Its claim to fame is that it minimizes the error in the desired frequency response.

Since finding coefficient sets for a given filter design is so computationally intensive, it is a good job for a computer program. DSP filter-design programs are readily available at low cost. Refer to the System Software section below for further discussion of filter design and the Bibliography at the end of this chapter for a list of software design tools. The article by Kossor has a practical circuit example of a commutating BPF that employs principles of DSP. Also see Project B in the chapter Appendix for examples of FIR filter designs.

IIR Filters

While FIR filters have a lot going for them, they tend to require a large number of taps and a proportional amount of processing power. As opposed to that, an IIR filter can provide sharp transition BWs with relatively few calculations. What it will not provide, in general, is a linear phase response. In circumstances where the computational burden is of more concern than the phase response, IIR filters may be desirable.

Unlike FIR filters, IIR filters employ feedback: That is what makes their impulse responses infinite. For this reason, IIR filters are usually designed by converting traditional analog filter designs, such as Chebyshev and elliptical types. See the Filters and Projects chapter of this book for a description of those designs. The transfer function of an analog Chebyshev low-pass filter can be written as the ratio of a constant to an n^{th}-order polynomial:

$$H_S = \frac{K}{a_0 s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n s^0} \qquad (7)$$

Tables in the literature, such as in Zverev, list the values of the coefficients, a_n, related to the cutoff frequency; these are used to derive actual component values for the filter. The low-pass design can be transformed to band-pass or band-stop responses. Two popular methods exist for deriving the digital transfer function from the analog: These are known as the *impulse-invariant* and *bilinear transform* methods.

The impulse-invariant method assures that the digital filter will have an impulse response equivalent to its analog counterpart, and thus the same phase response. Problems arise, though, if the bands of interest are near half the sampling frequency; the digital filter's response can develop serious errors in this case. Because of this problem, the impulse-invariant method is not as good as the bilinear transform

method. The bilinear transform method makes a convenient substitution for s in Eq 7 above. The filter output comes out as:

$$y(k)=\sum_{n=0}^{L-1}\alpha(n)x(k-n)-\sum_{n=1}^{L-1}\beta(n)y(k-n) \qquad (8)$$

This filter has L zeros and L–1 poles. The block diagram of such a filter for L = 5 is shown in **Fig 18.6**. Feedback is evident in the diagram: The paths involving coefficients ß loop back and are added to the signal path.

The *direct form* of Eq 8 may be factored into 2-pole sections and implemented in cascaded form. The output of each section serves as the input to the next. See **Fig 18.7** This configuration requires a few more multiplications than the direct form, but suffers less from instability problems that may plague IIR filters. Since feedback is being used, IIR filters are not necessarily unconditionally stable. They also tend to be prone to *limit cycles*, low-level oscillations that arise near the lower end of the dynamic range. For these and other reasons, data and coefficient storage should be cleared or set to zero before processing begins.

A Simple Digital Notch Filter

Along with common LPFs, HPFs and BPFs, radio designers are interested in one other type of filter:

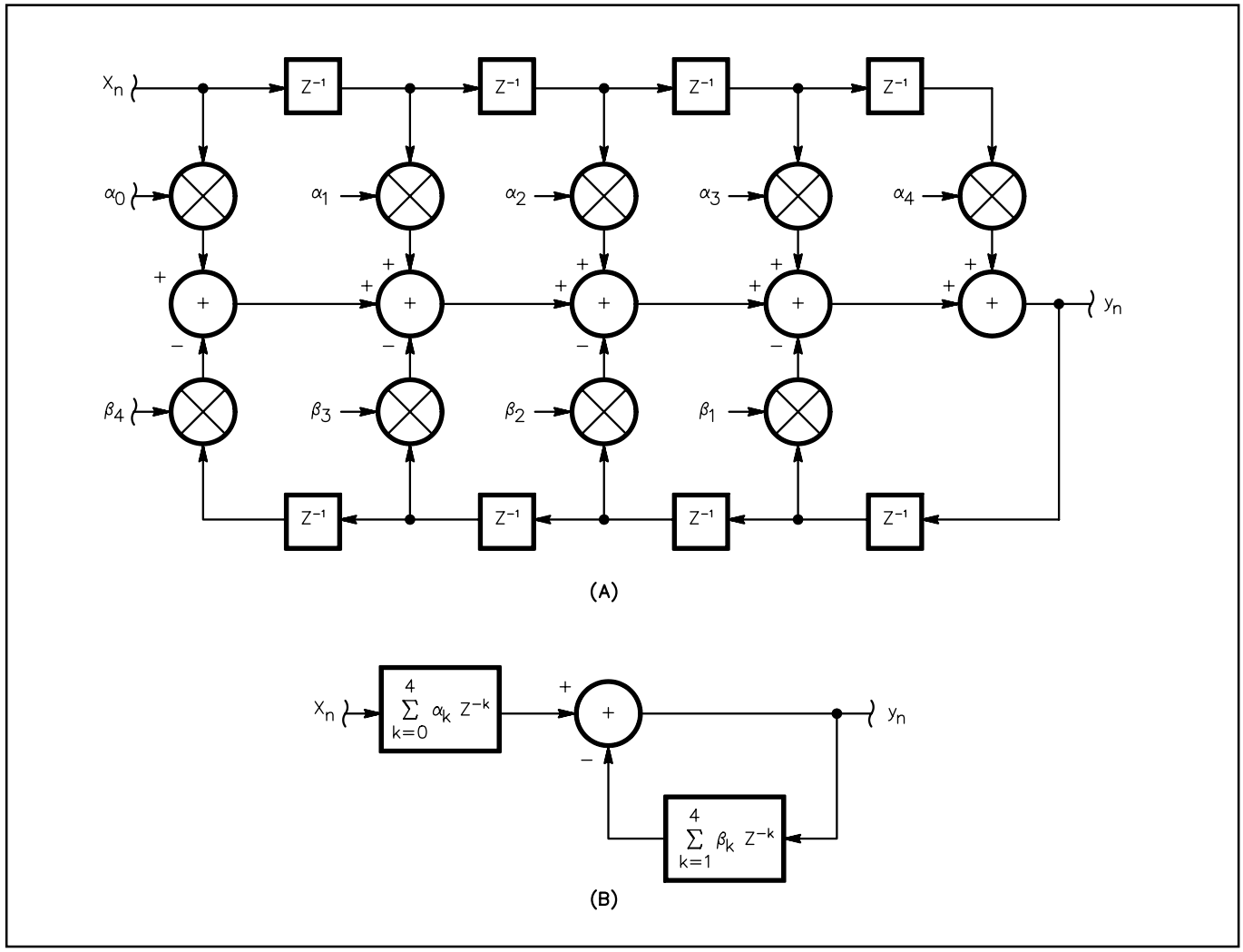

Fig 18.6—Block diagram of an IIR filter for L=5.

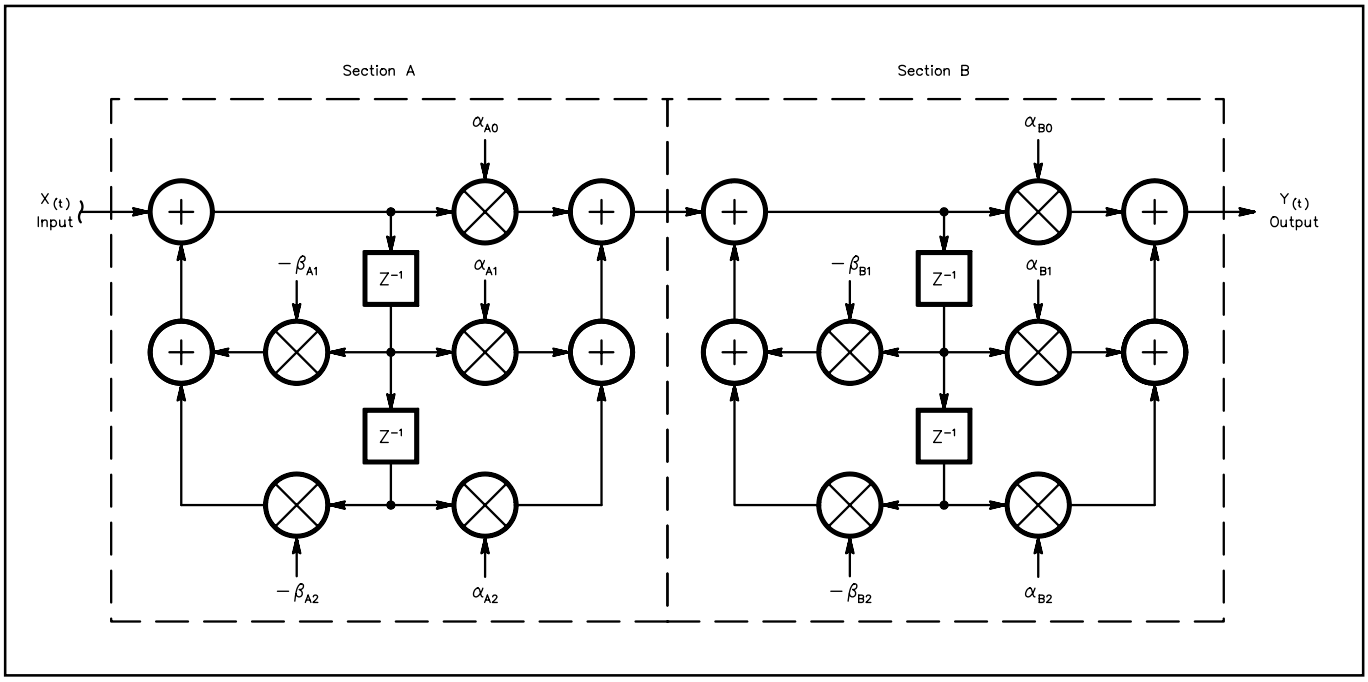

Fig 18.7—Block diagram of a cascade-form IIR filter.

the notch. While most filter-design software can generate notch filters using FIR methods discussed above, Widrow and Stearns have described an unusual type in which the number of taps is minimized. In fact, they were able to prove that only two taps are needed for each frequency to be notched. This is great, since it reduces computation to almost nil. We will take a look at it here and touch briefly on some of the theory of *adaptive signal processing*, treated in depth later.

The situation is this: We want to copy a broadband signal, such as an SSB phone signal, and suddenly a dreadful carrier appears in the passband. Our notch filter will remove it and we will have complete control over the notch width, as well as a notch depth limited only by the bit resolution of our system. Dr. Widrow found that one can build a filtering system that minimizes repetitive signal energy by altering the filter coefficients "on the fly" using a certain algorithm. Known as the *least-mean-squares* (LMS) method, it describes a way to adjust filter coefficients over time to remove undesired, steady tones in the input. A complex reference signal is used at the exact frequency of the offending tone. The algorithm then forms a BPF centered at the tone frequency whose output is subtracted from the input to create the notch. The block diagram of a two-tap system is shown in **Fig 18.8**.

The broadband input is called d(t). The reference input consists of two signals, $A\cos(\omega_0 t)$ and $A\sin(\omega_0 t)$. These signals feed multipliers having coefficients h(1) and h(2), which in turn feed an accumulator just as in a normal FIR filter. This is the BPF output; it is subtracted from the input to form the notch output, e(t). Note that the BPF output is also available at no additional overhead. While the initial values of the coefficients are unimportant to the steady state, the procedure for updating them with the LMS algorithm is:

$$h_{t+1}(1) = h_t(1) + 2\mu e(t) x_t(1)$$
$$h_{t+1}(2) = h_t(2) + 2\mu e(t) x_t(2) \qquad (9)$$

where $0 < m < 1$. Analysis shows that as the reference inputs are sinusoidal, the system is linear and time-invariant for output e(t), although the coefficient values do not necessarily approach any fixed value. The 3-dB BW of the notch is:

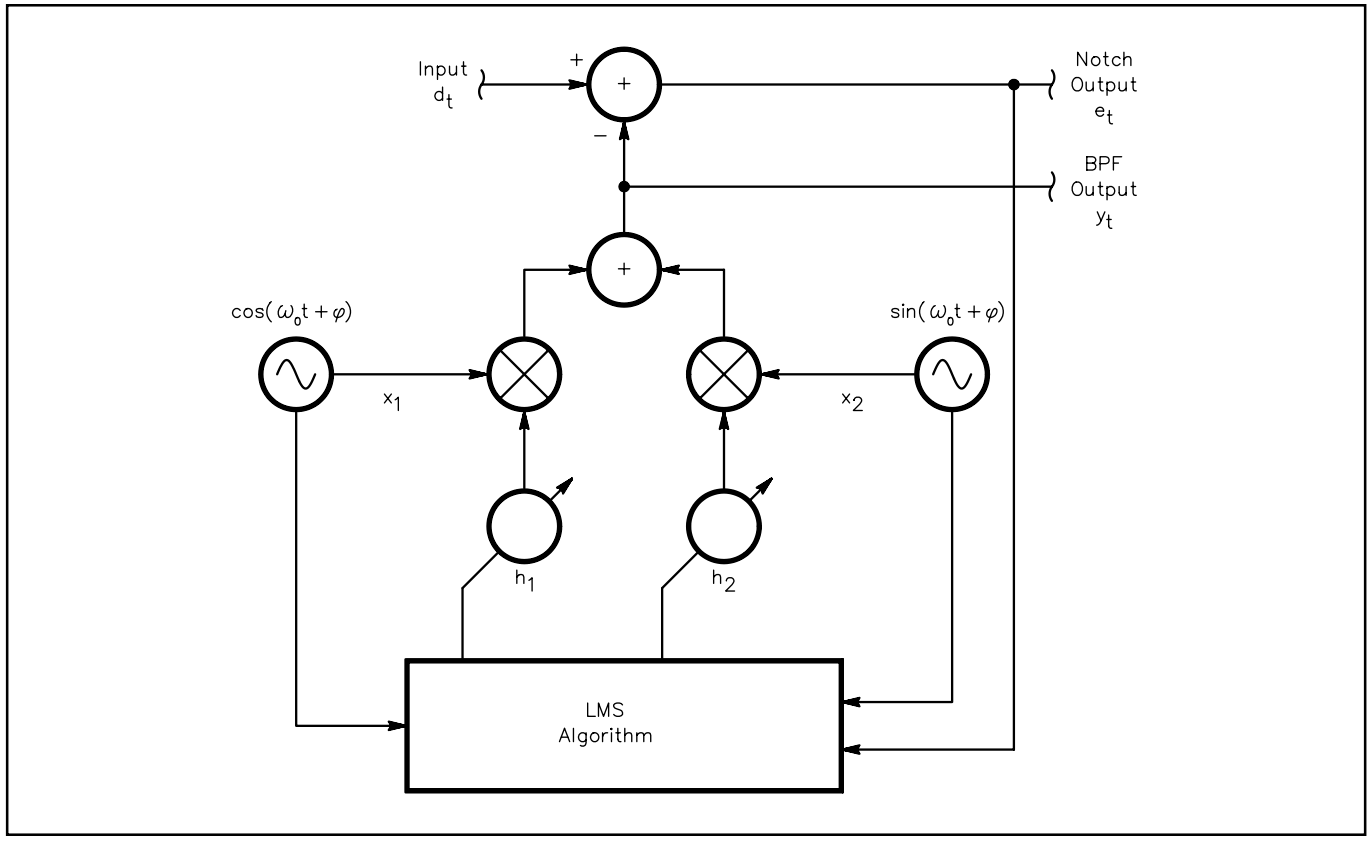

Fig 18.8—Block diagram of a two-tap, adaptive notch filter.

$$BW = \frac{2\mu A^2}{t_s} \text{ rad/s} \tag{10}$$

The Q of the filter may be readily computed. Thus, we have control over the BW by varying the factor μ and the amplitude of the reference signal. The depth of the null is, in general, superior to that of a fixed filter because the algorithm tracks the correct phase relationship for ideal cancellation, even if the reference frequency is changing slowly with the offending tone. Each additional tone to be notched demands two additional taps in the filter. Noise in the input may cause us to have to add more taps to maintain sufficient accuracy. Additional detail of adaptive signal processing will be found below and in material shown in the Bibliography.

Lattice and Other Structures

While many filter-design software packages do not have the capability to work with them, *lattice structures* and other types of digital filters have seen use, especially in adaptive signal processing. Crystal and mechanical lattice filters are common elements of many transceivers. A digital lattice or ladder filter is a lot like its analog brother. The design of digital lattice filters is similar, as well. Digital lattice filters may be either FIR or IIR. Also note that from the IIR cascade form above, we can derive a *parallel form* that may be computationally beneficial in some cases. The design of this kind of filter is a very complicated session in partial fraction expansion. Widrow and Stearns provide more information on these and other exotic concepts.

ANALYTIC SIGNALS AND MODULATION

DSP implementations of transceiver functions, such as modulation and demodulation, compel designers to examine the mathematics behind them. Computers are good at crunching numbers, but they do exactly what they are told! If we expect a DSP system to generate an SSB signal, for example, we had better know which calculations to perform and which to avoid.

Mathematics of Complex Signals

Because DSP makes it easy to build frequency-independent phase shifters—a fantasy in the analog world—the phasing or "I/Q" method has dominated other modulation techniques. Complex signals are not generally well understood and quite often form a stumbling block to those wishing to grasp DSP concepts. The idea of negative frequency is especially troublesome. The key to understanding these concepts lies in the theory of complex numbers. A real signal, such as a cosine wave, is normally thought of as a positive frequency. It can be transmitted and detected normally; however, we shall see that such a signal actually consists of positive *and* negative frequencies when examined in the complex domain.

A real cosine wave embodies the relation:

$$x(t) = \cos(\omega t) \tag{11}$$

where $\omega = 2\pi f$, and t is time. In the complex domain, the cosine wave is really the sum of two complex signals:

$$x(t)\frac{1}{2}\{[\cos(\omega t)+ j\sin(\omega t)]+[\cos(\omega t)- j\sin(\omega t)]\} \tag{12}$$

This signal has both positive and negative frequency components. The real parts add and the imaginary parts cancel to make the equation true. In the complex plane, where the real part is one axis and the imaginary part the other, this signal can be represented as two vectors rotating in opposite directions. See **Fig 18.9**.

While this depiction is beautiful and elegant to the mathematician, what does it really mean to you and me? Well, it means that signals represented in complex form can have a one-sided spectrum—that is, only a positive or a negative frequency component. This is useful as we mix our signals upward to their final frequency positions in a modulator.

As our first example, let's select the task of taking a real input signal, such as the audio from a microphone, and converting it to an SSB signal that can be transmitted. We obviously have to translate the audio signal upward in frequency and preserve its spectral content within the band we want the transmitted signal to occupy. If we wish to produce an upper-sideband (USB) signal, we want the carrier and lower sideband to be suppressed as much as possible. Were we able to translate the spectrum of our cosine wave—with its symmetrical positive- and negative-frequency components—upward in frequency far enough, we would have two positive frequencies separated by twice the original signal's frequency. For a real signal, this is exactly what happens when it is applied to an analog mixer: Both sum and difference frequencies are generated. See the **Receivers, Transmitters, Transceivers and Projects** chapter for more detail of the operation of mixers as multipliers.

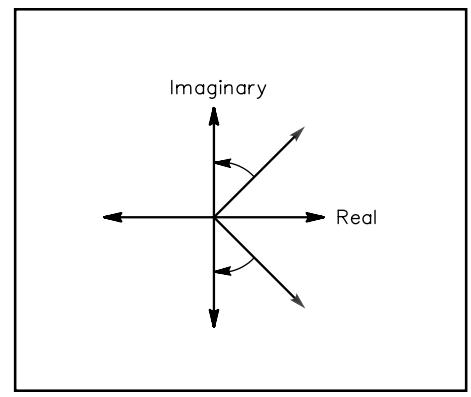

Fig 18.9—Vector representation of a real cosine wave.

To move our sampled audio signal upward in frequency, we must multiply it by (mix it with) a local oscillator. The local-oscillator function can be implemented in DSP software using

direct digital synthesis (DDS) techniques. In this case, though, the local oscillator must be complex; that is, it must have two outputs with a 90× phase relationship between them. This is the same as saying there must be both a sine and a cosine output from it. This will enable us to mix signals having a one-sided spectrum.

When we implement a complex mixer in DSP, we are multiplying complex numbers by complex numbers. Note that the calculations for the real and imaginary parts are carried out separately; each part is treated as if it were a single, real multiplication. Two complex numbers a + *j*b and c + *j*d, when multiplied, produce:

$$(a + jb)(c + jd) = (ac\text{-}bd) + j(ad + bc) \tag{13}$$

Four real multiplications and two real additions are required.

Hilbert Transformers and an SSB Modulator

If we want to create a signal having a one-sided spectrum from a real input signal, such as from the microphone, we need to shift all the frequency components in the sampled signal by 90º. Fortunately, in DSP, we have a way to do that: the *Hilbert transformer*. Recall that an FIR filter with a symmetrical impulse response exhibits a constant, frequency-independent delay. It turns out a filter with an *anti-symmetrical* impulse response—that is, with h(0) =–h(L –1), h(1) = –h(L –2), and so forth—produces a linear phase response, too, but with a phase response exactly 90° different from the symmetrical-impulse-response filter. This is exactly the type of filter we need to generate the components of an *analytic signal*.

Fig 18.10 shows a system using a Hilbert transformer to create an analytic signal from the microphone audio. Since the Hilbert transformer includes not only a 90° phase shift, but also a fixed delay of L/2 sample periods, we need an L/2 delay in the leg that does not contain a phase shift. The delay through the two paths is then equal and the only difference between the two signals produced is the 90° phase shift. The non-phase-shifted signal is called I, the phase-shifted signal is called Q. Together, these signals form our analytic signal I + *j*Q. Now let's see what it looks like when we multiply this signal by a complex local oscillator.

In this case, we are performing the multiplication:

$$\begin{aligned}&[\cos(\omega t) + j\sin(\omega t)][I(t) + jQ(t)] = \\ &\quad [I(t)\cos(\omega t) - Q(t)\sin(\omega t)] + j[I(t)\sin(\omega t) + Q(t)\cos(\omega t)]\end{aligned} \tag{14}$$

This is the equation for a USB signal. We are only interested in the real part of the result, since we only have one real channel on which to transmit. For this reason, the system is really a *half-complex mixer*.

A block diagram of such a mixer is shown in **Fig 18.11**. This is, in fact, the phasing method. Output signals are translated upward by the frequency of the local oscillator, w radians per second, or $\omega/2\pi$ hertz. Most transmitter designs will translate signals to an IF significantly higher in frequency than audio, so it is wise to include an increase in the sampling rate prior to mixing. An interpolation filter is naturally needed. It is particularly convenient to choose an interpolation factor of 4, because the cosine LO produces values of 1, 0, –1 and 0 during a full cycle; the sine LO produces values of 0, 1, 0 and –1. No actual multiplications need take place, saving time and accuracy. The Hilbert transformer can operate at the lower, original sampling rate, but we would like to

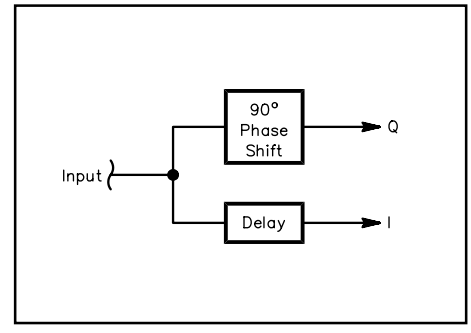

Fig 18.10—Hilbert transformer producing an analytic signal.

include band-pass filtering to limit the spectrum to about 3 kHz BW. In fact, we can build a pair of DSP filters that provide the BPF response and the 90° phase relationship, as described below. Our SSB modulator then matches that shown in **Fig 18.12**.

Before discussing how to generate analytic filter pairs, it is worth noting a few properties of SSB signals created in this way. First, were we to add the I and Q signals instead of subtract them in the summation block of Fig 18.11, we would have an LSB signal instead of USB. It is not too hard to see that we could easily both add and subtract to produce a DSB, suppressed-carrier signal. We can even pre-add and subtract *two* audio signals to produce an independent-sideband (ISB) signal, as shown in **Fig 18.13**. More than two channels can be combined in this way. Second, since the amplitude of the carrier, cos (ω_0 t) ± *j*sin (ω_0 t), is constant, the amplitude of an SSB signal can be specified as some function of the modulating signal. If we think of the analytic audio signal as a vector in the complex plane, its length is equal to the signal's instantaneous amplitude:

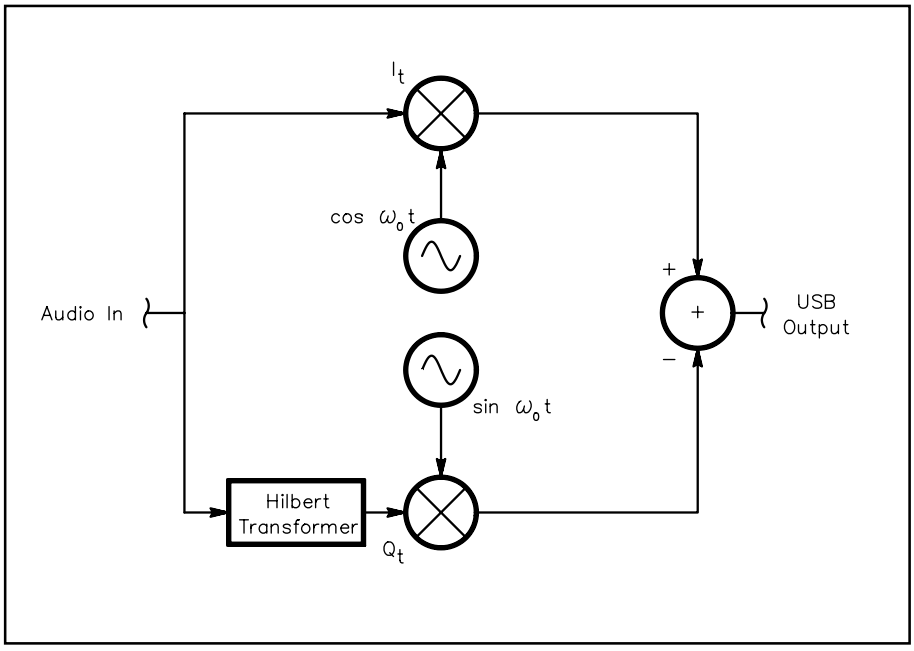

Fig 18.11—Block diagram of a half-complex mixer.

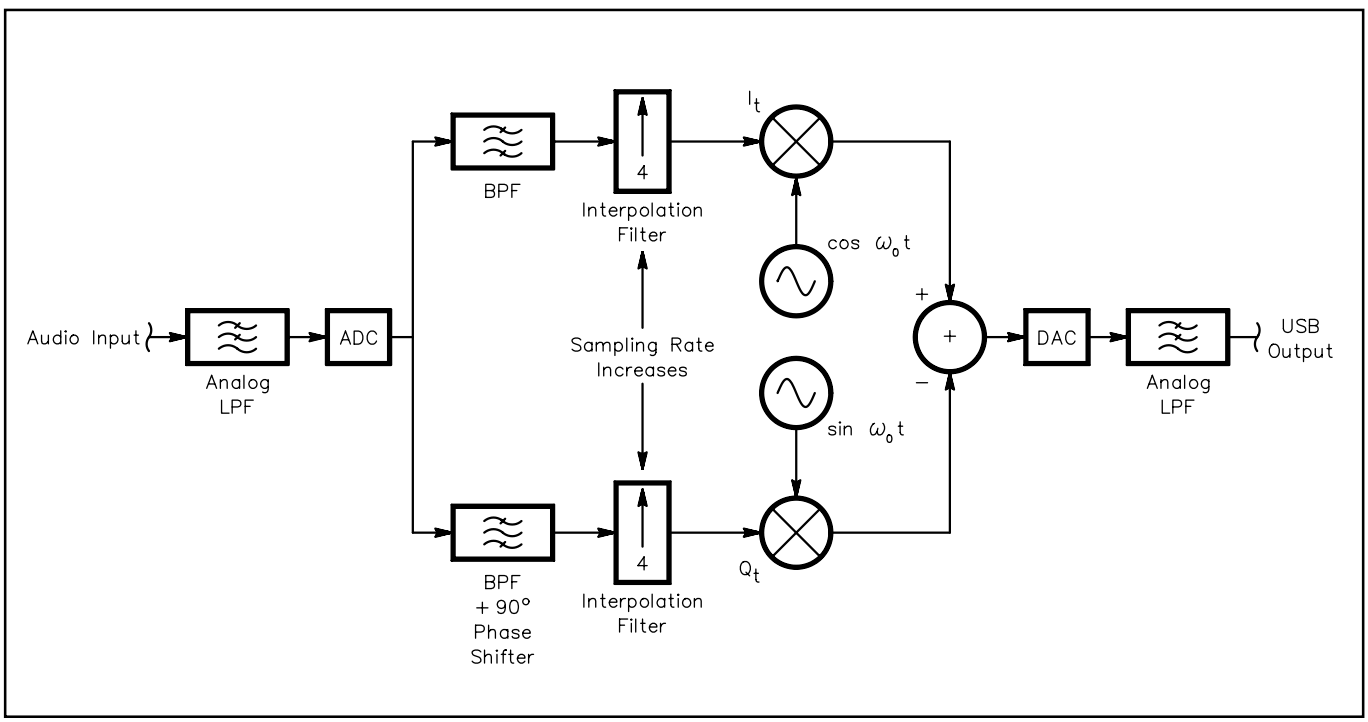

Fig 18.12—Block diagram of a digital SSB modulator.

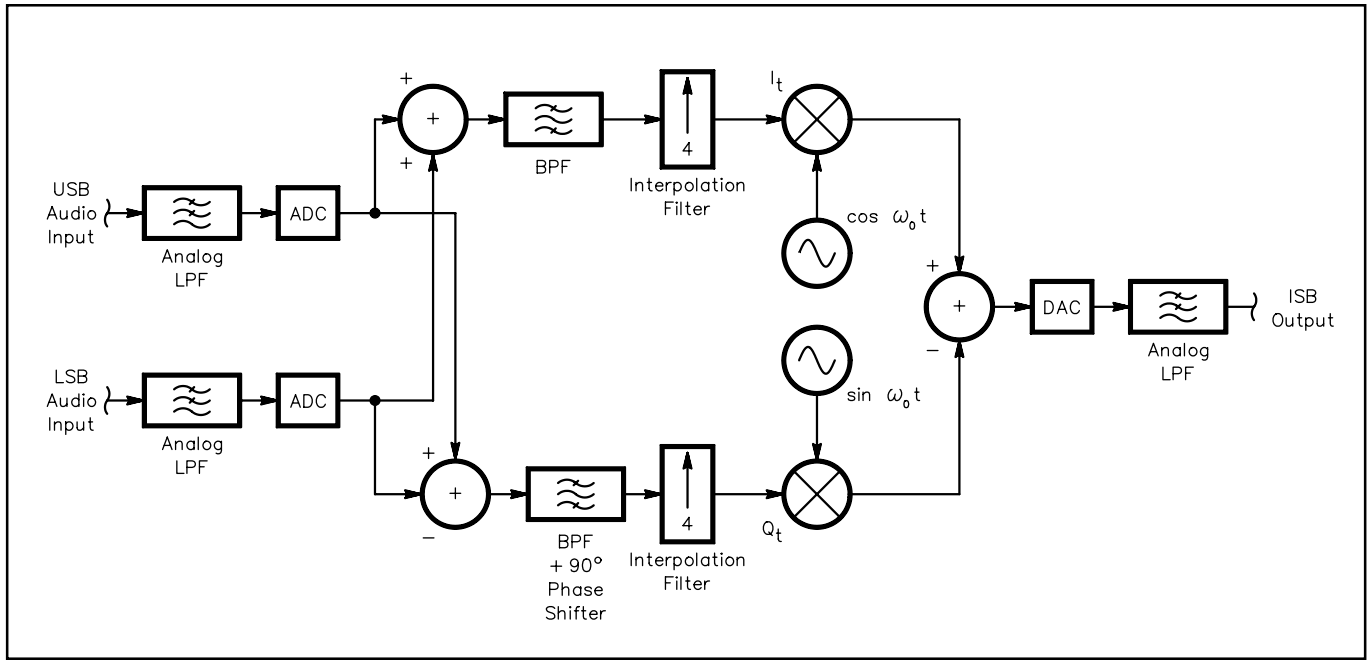

Fig 18.13—Block diagram of a digital ISB modulator.

$$A(t)=\left[I^2(t)+Q^2(t)\right]^{\frac{1}{2}} \tag{15}$$

Finally, the phase of the signal is the instantaneous angle of this rotating vector:

$$\phi(t)=\arctan\left[\frac{Q(t)}{I(t)}\right] \tag{16}$$

Now we can rewrite the real part of Eq 14 as:

$$A(t)\cos[\omega t+\phi(t)] \tag{17}$$

This expression clearly shows SSB to be a hybrid of amplitude and phase modulation. Also, we can write the equation:

$$[I(t)+jQ(t)]=A(t)\{\cos[\phi(t)]+j\sin[\phi(t)]\} \tag{18}$$

directly relating the envelope and phase to the analytic *baseband* signal. The amplitude and phase of the analytic signal are identical to those of the SSB wave so produced.

Analytic Filter-Pair Synthesis

We have seen how complex mixing translates signals in frequency with a one-sided spectrum. We will use this fact to our advantage in creating an analytic filter pair. Each filter will have the same frequency response as the other; they will differ only in their phase responses.

We begin by designing a low-pass filter having the desired transition-band characteristic, H(ω); we obtain its impulse response, h(t). Multiplying the impulse response by a complex sinusoid of angular frequency ω_0 results in two sets of coefficients—one for the real part, and one for the imaginary part:

$$\begin{aligned} h_I(t)&=h(t)\cos(\omega_0 t)\\ h_Q(t)&=h(t)\sin(\omega_0 t)\end{aligned} \tag{19}$$

The frequency response of either one of these filters is given by:

$$H_\omega = \frac{H_{(\omega-\omega_0)} + H_{(\omega+\omega_0)}}{2} \qquad (20)$$

which is a BPF centered at w_0. The I filter has a phase response differing 90° at every frequency from the Q filter. The frequency translation theorem works just as well on the responses of filters as it does on real signals. To perform this transformation of the L coefficients of the prototype LPF, we calculate new coefficients according to:

For $0 \le k \le L-1$,

$$h_I(k) = h(k)\cos\left[\omega_0\left(k - \frac{L}{2} + \frac{1}{2}\right)t_s\right] \quad \text{OR} \quad h_Q(k) = h(k)\sin\left[\omega_0\left(k - \frac{L}{2} + \frac{1}{2}\right)t_s\right] \qquad (21)$$

where t_s is the sampling period. When the low-frequency transition band is placed near zero frequency, as we would like for SSB, the BW of each BPF is approximately twice that of the prototype LPF. A very interesting thing ordinarily happens when the number of taps is odd: The odd-numbered coefficients are zero. This allows reduction in computation by a factor of two. Refer to Project C in the Appendix for a practical example of how analytic filter pairs are generated.

We can alter the exciter's frequency response by convolving the impulse response of our analytic filter pair with that of a filter having the desired characteristic. New coefficients are calculated using the same convolution sum as in Eq 3. Graphic or parametric equalizers may be implemented in this way.

Demodulation: SSB

As in digital exciters, phasing methods prevail in receivers; the process is almost exactly the reverse of the modulator's. **Fig 18.14** presents the block diagram of a digital SSB receiver. After the IF signal is digitized, we wish to reduce the sampling rate and the filtered BW as soon as possible. This is because we need as much time as possible between input samples for the intense filtering and other computations we must perform. As noted above, reduced sampling rates also ease the design of the digital filters that

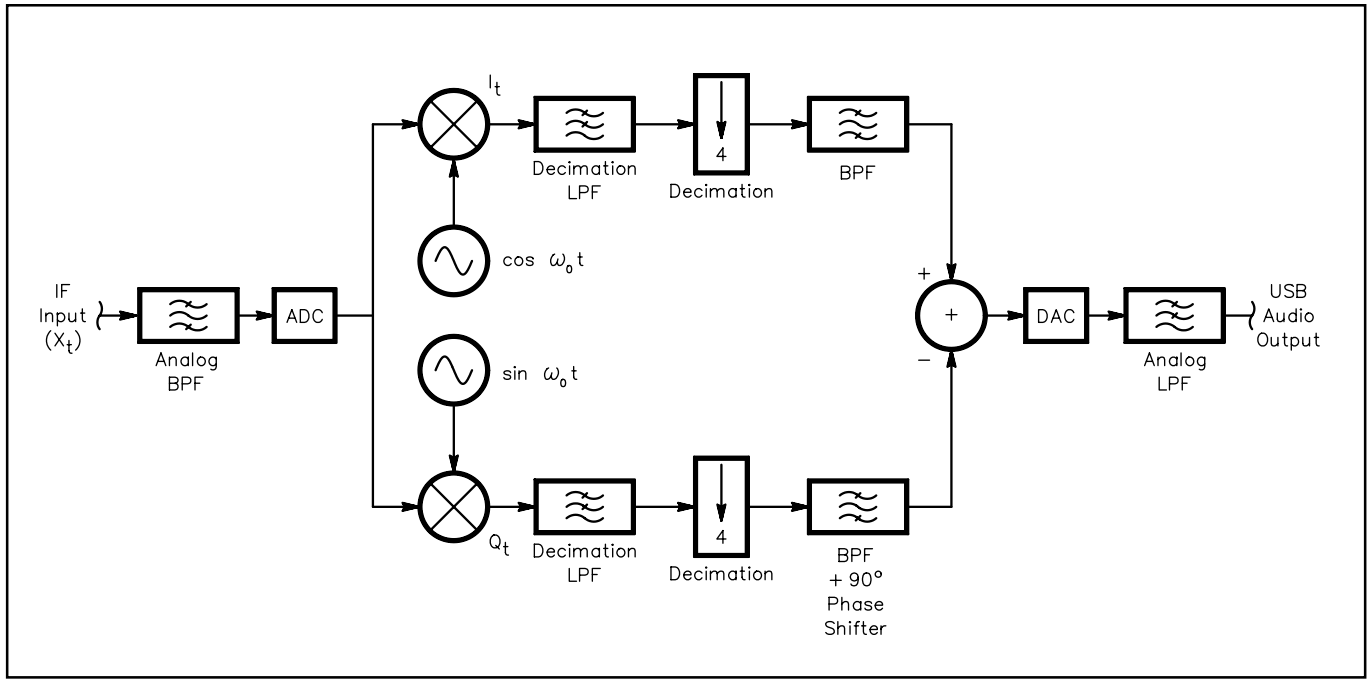

Fig 18.14—Block diagram of a digital SSB demodulator.

provide the final selectivity. We therefore include a decimation filter and decimate by a factor of 4. Again, the LO signals take on only values of 1, 0, –1 and 0, eliminating multiplications. Digitized signals are translated to baseband using the complex mixing algorithms outlined above. Since the input signal, x(t), is real, only two multiplications are necessary:

$$I(t) = x(t)\cos(\omega t)$$
$$Q(t) = x(t)\sin(\omega t) \qquad (22)$$

Now we have an analytic signal as before; the frequency of the BFO, ω rad/s, is chosen to beat the carrier frequency to zero hertz. An analytic filter pair precedes the summation in which we select the sideband we want. The equations work precisely in reverse: That is why they are Hilbert *transforms*.

AM Demodulation

One's first inclination is to demodulate an AM signal by rectifying it. A better way is to use the I and Q signals we have already developed: Eq 15. Now we are stuck with computing square roots. Lucky for us, a fellow named Isaac Newton figured out a slick way almost 400 years ago. In the 17th century, these calculations were quite a burden—everything had to be done by hand! Because this is such a common problem in computing, a lot of additional effort has gone into finding faster algorithms since that time. A new, very fast look-up-table method is also presented here that may be more attractive where enough memory is available.

An I/Q AM demodulator dodges problems associated with rectification methods. It also can use the decimation filters for final selectivity, obviating much of the computations found in the SSB demodulator. **Fig 18.15** shows the circuit. Newton's method for square roots goes like this: Take a crude guess at the square root of the number in question. Divide the number by the crude guess. Add the crude guess to this ratio and divide it all by 2. Use this result as the new crude guess and repeat the process until the desired accuracy is obtained:

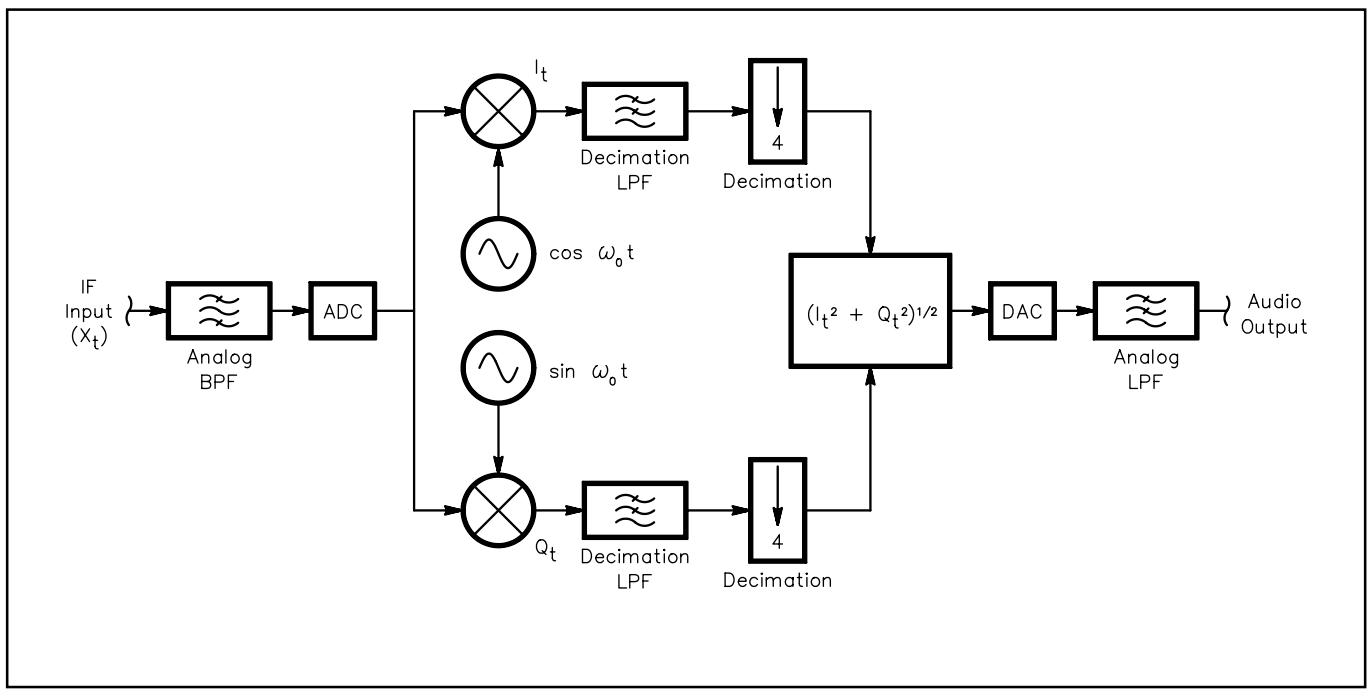

Fig 18.15—Block diagram of a digital AM demodulator.

$$\text{letGUESS}_{new} = \left(\frac{\frac{\text{Number}}{\text{GUESS}_{old}} + \text{GUESS}_{old}}{2} \right)$$

$$\text{letGUESS}_{old} = \text{GUESS}_{new} \qquad (23)$$

REPEAT

In practice, the accuracy of the result reaches the limit of 16-bit representations in five or six iterations when the first guess is good. It is about half an order of magnitude slower than the following look-up table method, but is still among the best where memory is at a premium. Project D in the Appendix describes a *QuickBasic 4.5* example of Newton's method.

A new, very fast look-up-table method for computing integer square roots has been discovered. It employs a short (512-entry) table and first-order interpolation between table entries. First-order interpolation is described in detail in the DDS section below. To preserve accuracy, the algorithm also uses the process of argument normalization. The algorithm serves as our fifth software project in DSP.

The argument of this function—the number of which we must find the square root—is a 32-bit integer. The result is a 16-bit integer. Refer to **Fig 18.16**, a flow chart of the process. In the first step, the argument is normalized to within the range 2^{22}-2^{24}. Arguments greater than 2^{24} are divided by an integral power of two, 2^k, where:

$$k = \Im[\log_2(\text{arg}) - 23] \qquad (24)$$

The script I indicates the integer part, and k—which takes on values of 0, 2, 4 or 6—is saved for later processing. Now the normalized argument is split into integer and fractional parts, with the radix point residing to the left of bit 15:

$$a = \Im\left(\frac{\text{arg}}{2^{k+8}} \right)$$

$$b = \mathcal{F}\left(\frac{\text{arg}}{2^{k+8}} \right) \qquad (25)$$

where a is the integer part and b is the fractional part. In other words, a comprises bits 16-23 of the normalized argument, and b is bits 0-15, as shown in the flow chart. Next, we use a as the address into the look-up table, fetching a 16-bit value, x_a. This value is the nearest table entry lower than the actual root. Fractional part b is used to interpolate between this value and the next higher table entry, x_{a+1}:

$$\text{root} = b(x_{a+1} - x_a) + x_a \qquad (26)$$

This is the square root of the normalized argument.

Finally, this result must be multiplied by the square root of 2^k, which is of course $2^{k/2}$. The result is then "de-normalized" and ready for use. Restricting k to an even integer (as we did) makes this a simple bit-shifting operation, as in the normalization process above. The 16-bit result produced by this algorithm is accu-

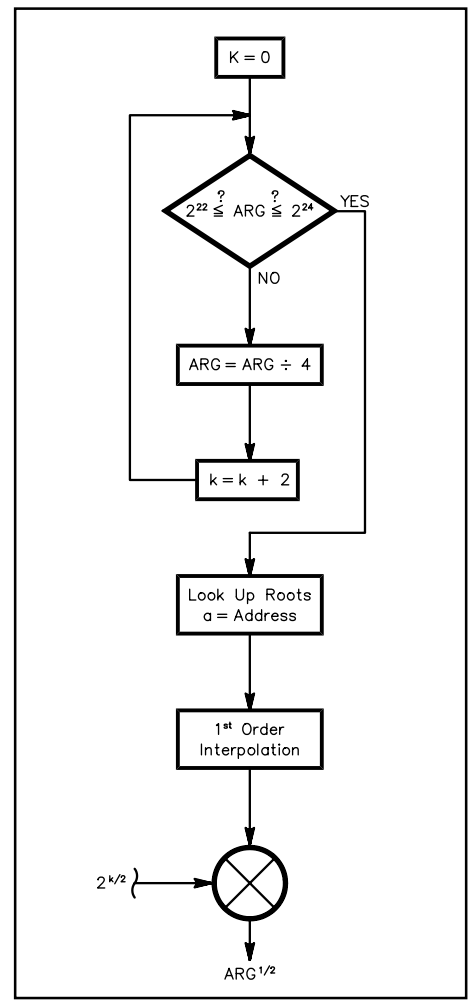

Fig 18.16—Flow chart of a fast square-root algorithm.

rate to within several least-significant bits over the entire range of 32-bit arguments. It is quite a bit faster than the 5 or 6 iterations of Newton's method required for the same accuracy; this is because it avoids the divisions that Newton's method employs. Most DSPs take 3 or 4 times the processing time for a fractional division as they take for multiplication or look-up table indexing. Project E in the Appendix describes an assembly-language implementation of this square-root algorithm.

Additional threshold extension and distortion-avoidance procedures may be employed in an AM demodulator. Of particular interest is the *synchronous*, *exalted-carrier* demodulator. Synchronous, in this case, means that the demodulator's frequency standard is phase-locked to the received carrier. This forces the phases of modulation components into their correct relationships and therefore minimizes phase distortion. A small advantage in SNR performance of up to 3 dB is also gained. DSP makes it relatively easy to build a narrow BPF, centered on the carrier, that strips the modulation prior to application to the PLL used to achieve lock. The exalted-carrier technique is a way of avoiding distortion caused by selective fading of the carrier. Ordinarily, when received carrier amplitude drops, the signal becomes over-modulated, even though it was not transmitted that way. Distortion can be severe. Exalted carrier strips the carrier from the signal using the narrow BPF; it is used only to drive the PLL. A copy of the limited carrier is then added back to the carrier-stripped signal prior to demodulation, at an amplitude that avoids over-modulation. See **Fig 18.17** for a block diagram of this type of demodulator. Refer to the chapter on Modulation Sources for more discussion of AM waveforms.

FM and PM Demodulation

Traditional FM and PM demodulators, such as discriminators (filters) and PLLs may be implemented in DSP; but again, the I/Q method carries distinct advantages as it exploits mathematical relationships. We already defined the phase of an analytic signal in Eq 16, and so we can build a PM demodulator directly by finding arctangents. Possibilities include look-up tables and Taylor series. For an FM demodulator, we would then differentiate the string of phase samples using the technique of *first differencing*. We simply take the difference between adjacent samples by subtracting them:

$$f(t) = \phi(t) - \phi(t-1) \qquad (27)$$

and this is the FM demodulator's output.

One common analog technique that stands out among DSP implementations is the quadrature detector. It is certainly simple and convenient to generate delays and multipliers, such as are required. The input signal is multiplied by a time-delayed copy of itself to produce a voltage proportional to its phase excursions away from the center frequency. This voltage is also proportional to the amount of delay inserted. See **Fig 18.18**. When the delay is an odd integral multiple of one

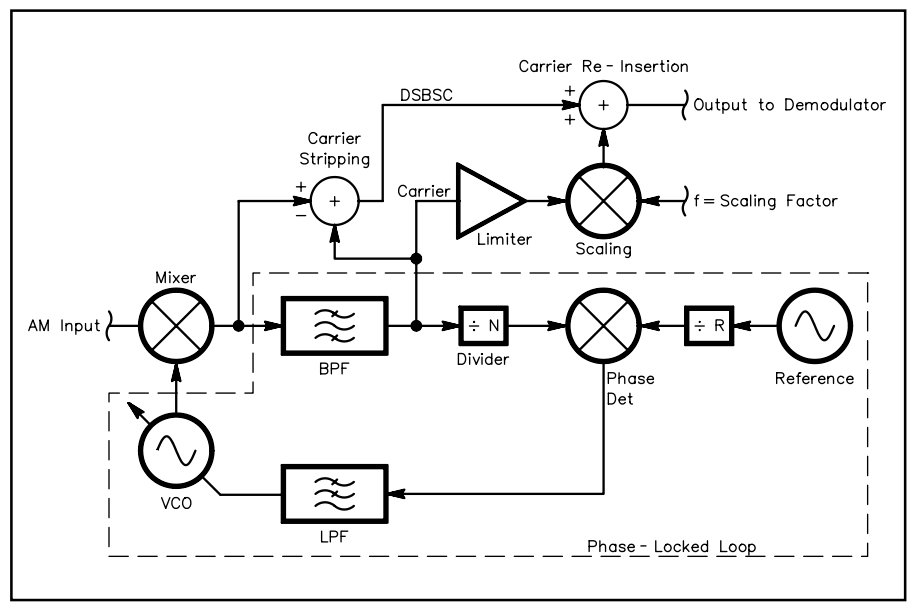

Fig 18.17—Block diagram of a synchronous, exalted-carrier demodulator.

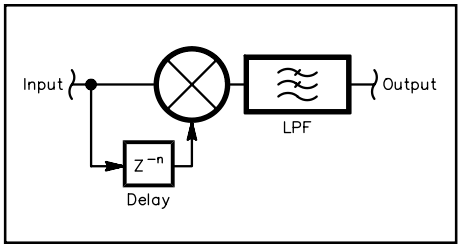

Fig 18.18—Block diagram of a digital quadrature detector.

quarter the input period, the output is zero. Longer delays produce greater output-voltage sensitivities; that is, dV/dϕ increases.

Digital BFO Generation: Direct Digital Synthesis

Synthesizers have come a long way since first becoming popular in HF transceivers of the 1970s. Availability of components then lagged well behind the development of theory. Now, hardware capabilities have nearly caught up—which is the case for DSP in general—and are driving the very rapid advancement of equipment we are now experiencing. Paralleling breakthroughs in the microprocessor and data acquisition fields, progress in *direct digital synthesis* (DDS) has enabled performance levels only dreamed of a decade ago. Virtually all new designs may profit from this technology. Below, we will cover quite a few issues impacting transceiver performance: phase noise, spectral purity, frequency stability, lock times and tuning resolution. A DDS circuit using dedicated hardware is described; discussion of BFO and LO generation in software follows.

Synthesizer performance affects receiver dynamic range. Phase-noise and spectral-purity issues are in play. Phase noise is the unwanted phase modulation of transceiver frequency-control elements by circuit noise. It appears at and near the transmitter's output frequency and may cause interference to stations on adjacent frequencies. In addition, it may cause interference in one's own receiver—even if the signals received are phase-noise free—through the process of *reciprocal mixing*. See the **AC/RF Sources (Oscillators and Synthesizers)** chapter for a discussion of this effect. The spectral purity of a synthesizer may also affect receiver dynamic range by introducing spurious responses where spurs exist on the synthesizer's output. This may be true especially for the first LO in a receiver across the entire range of frequencies present. It is extremely important, then, that this LO be clean.

Radio amateurs are free to operate anywhere within large frequency bands, so it might seem that frequency accuracy is not very critical. Prevalent narrow-band communication modes require it, though, and operators have come to expect excellent stability from their rigs. It is reasonable to expect ±20-Hz stability over a range of –10 to +50° C. Digital compensation techniques currently achieve this. We wish to attain a tuning speed that does not impose limitations on typical use. "Cross-band," or split-frequency operation ought to be considered. For a frequency step of ±600 kHz, an upper limit of 25 ms on the lock time of a synthesizer is a reasonable goal. Lock time is defined as the time required to settle within the stability limits we already set. The smallest frequency steps should be such that they do not impede performance. 10 Hz used to be good enough, but now certain digital modes benefit from finer tuning. In addition, the digital notch filter described before is so sharp that it occasionally needs to be within 1 Hz!

A DDS system generates digital samples of a sine wave and converts them to an analog signal using a DAC. See **Fig 18.19**. In a DDS chip, a phase accumulator is incremented at each clock time; the phase information is used to look up a sine-wave amplitude from a table. This value is passed to the DAC, which outputs a step-

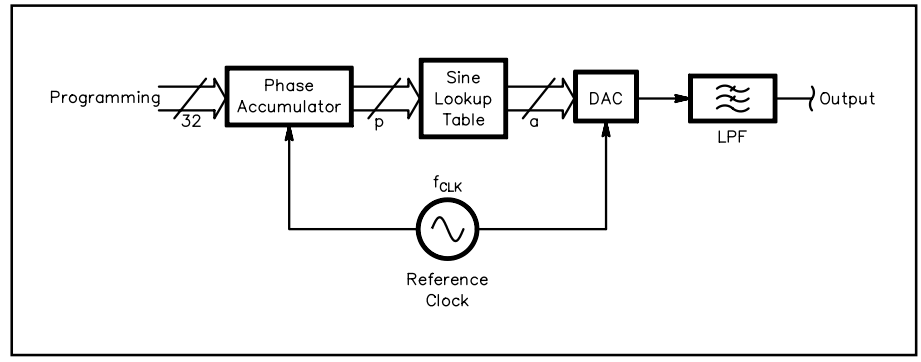

Fig 18.19—DDS block diagram.

wise sine wave. As we saw before, the spectrum of this sine wave is seasoned with aliases and contains other minor pollutants. Since the phase is represented by a binary number with a fixed number of bits, p, errors develop because the number is truncated to that number of bits. Truncation generates PM spurs in the DDS output. This occurs prior to the DAC. Further errors are related to the output resolution of the look-up table. Table values representing the amplitudes are truncated to some number of bits, a. This mechanism produces AM spurs in the output. According to Cercas *et al*, the largest PM spurs have amplitude:

$$P_{spur} = -(6.02p - 5.17)\text{dBc} \quad (28)$$

and maximum AM spurs can rise to:

$$P_{spur} = -(6.02a + 1.75)\text{dBc} \quad (29)$$

Phase noise at the output is that of the DDS clock source times the ratio of the output frequency to the clock frequency, as limited by divider noise. Spurious levels also tend to grow as the DDS output frequency approaches the Nyquist limit. Strange spurs at the output are usually related to IMD and harmonics of the desired signal and their aliases. Remember that frequencies exceeding half the sampling frequency "fold back" into the signal spectrum at a position determined by their frequency, modulo $f_s/2$. High-order harmonics are liable to find their way into one's band of interest. Traps at the DAC output have been known to suppress these responses. See Project F in the Appendix for the schematic and parts list of a DDS project.

In the analog signal we generate, the DAC introduces more AM spurs, harmonics and IMD because of its inherent non-linearity, as discussed above. Spurs are also likely at the clock frequency, its harmonics and sub-harmonics. A higher-order LPF will take care of these, but we must see what we can do about the others. It turns out we may eliminate *all* the AM spurs by squaring the DDS output. We can do nothing about the remaining PM spurs. Cranking through Eq 28 will show that they can be made very low: –113 dBc for a 20-bit-address sine look-up table and 32-bit phase accumulator. This parameter is critical in case we want to use the DDS as the reference to a high-frequency PLL circuit: The PLL will multiply the phase noise and PM spurs by the ratio of the PLL output frequency to the PLL reference frequency within the PLL loop BW. Outside the loop BW, the VCO itself is responsible for establishing spectral purity. So while dividing the DDS to the PLL reference frequency lowers phase noise and PM spurs, the PLL multiplies them back upward. A trade-off exists between spur levels and reference frequency, hence lock time.

A PLL reference frequency of near 100 kHz has been found to be sufficient for the desired lock times, with an output-to-reference ratio of 1000. Such a loop should achieve very fast lock times, as it can be expected to lock within 500 cycles of the reference input. The DDS tuning time is at least three orders of magnitude faster than this.

In the example, the VCO output is near 100 MHz. DDS energy is injected at the reference input to the PLL chip, squaring it and dividing it by 10; the DDS runs near 1000 kHz. The block diagram of a PLL using a DDS as its reference is shown in **Fig 18.20**. Spurs and phase noise inside a loop BW of, say, 1 kHz are amplified by the PLL by the factor:

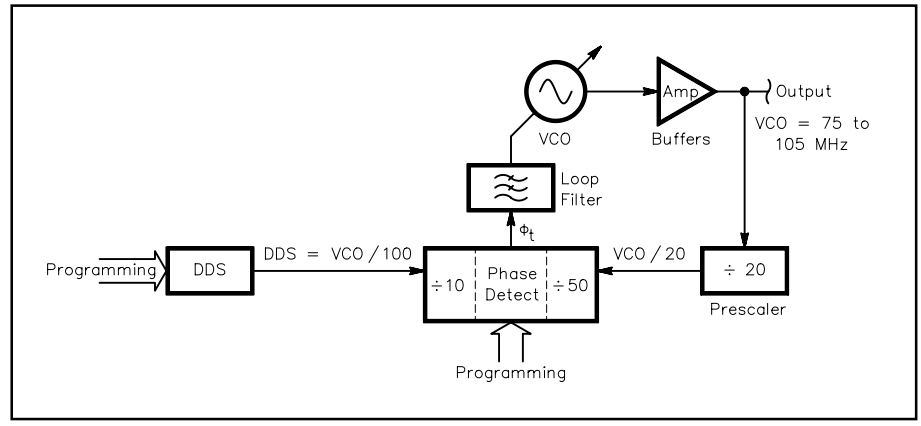

Fig 18.20—Block diagram of a DDS/PLL hybrid synthesizer.

$$N = 20\log\left(\frac{f_{VCO}}{f_{REF}}\right) = 40\,dB \tag{30}$$

Of course, we tune the hybrid synthesizer by programming the DDS; the PLL programming is fixed. Let's say we want 1-Hz tuning resolution at the VCO output. As the DDS frequency is 1/100 of the output, we must tune the DDS in 10 *millihertz* steps! Tuning resolution in a DDS circuit is determined by the phase accumulator's bit resolution, p, and the DDS clock's frequency, f_{clk}:

$$df_{DDS} = \frac{f_{CLK}}{2^P} \tag{31}$$

A clock frequency around 10 MHz and p = 32 easily satisfy our conditions, producing a step size of 2.3 millihertz. As noted above, making the DDS output frequency a small fraction of the clock frequency makes it easier to get a clean output. A range of about half an octave eases the design of the LPF or BPF used at the DDS output to limit spurs, aliases and clock feed-through.

The phase-accumulator/look-up-table approach is equally useful in generating numeric BFOs in software. One of the first things to emerge when considering this scheme is the potentially large size of the look-up table. To maintain the full dynamic range of a DSP system requires BFO phase and amplitude performance, as limited by Eqs 28 and 29, at least as good as the rest of the system. In 16-bit systems, we are shooting for about 90-100 dB of dynamic range. A table with 2^{16} = 65,536 entries is not much of a problem for DDS chip manufacturers to include on-board, but it may tax available memory space in embedded systems.

Fortunately, a couple of ways around the problem have been uncovered. The first involves the process of interpolation, very much like the artificial increase of sampling frequencies we examined above. In this method, we restrict the number of table entries to some arbitrary number, $M<<2^{16}$, while keeping the bit-resolution of the entries themselves, a, high enough to satisfy the limits of Eq 29 for the spur levels we can tolerate. Take the case where $M = 2^8 = 256$ and a = 16. The phase accumulator, incremented at each sample time by an amount df that is directly proportional to the output frequency, forms the address into the look-up table. Let this address have bit-resolution p = 16. According to Eq 28, PM spurs will not exceed –91 dBc. Since there are only 256 table entries, we may use the most-significant byte (MSB) of the address to find the table entries that straddle the correct output value. We then use the least-significant byte (LSB) as an unsigned fraction to find out how far between the two table entries we must go to reach the correct output value. If, in order of increasing address in the table, our two adjacent table entries are d_1 and d_2, we may perform a first-order interpolation between the entries using:

$$d_{int} = d_1\left(\frac{256 - LSB}{256}\right) + d_2\left(\frac{LSB}{256}\right) \tag{32}$$

This results in a linear, piece-wise representation of the data, as shown in **Fig 18.21**. The worst-case amplitude errors caused by this straight-line approximation place total harmonic distortion (THD) at the output at around 0.03% or –70 dBc. Much of this harmonic distortion is concentrated near half the sampling frequency, though, and may not be of much concern in actual systems. Doubling M would reduce THD to around 0.01%. Second and higher-order interpolation algorithms are available that outperform the first-order approximations by a long way.

In systems where an even smaller look-up table must be used, computation of sines and cosines using Taylor series might be attractive. THD is less than 0.008% when using four or five terms from the polynomials:

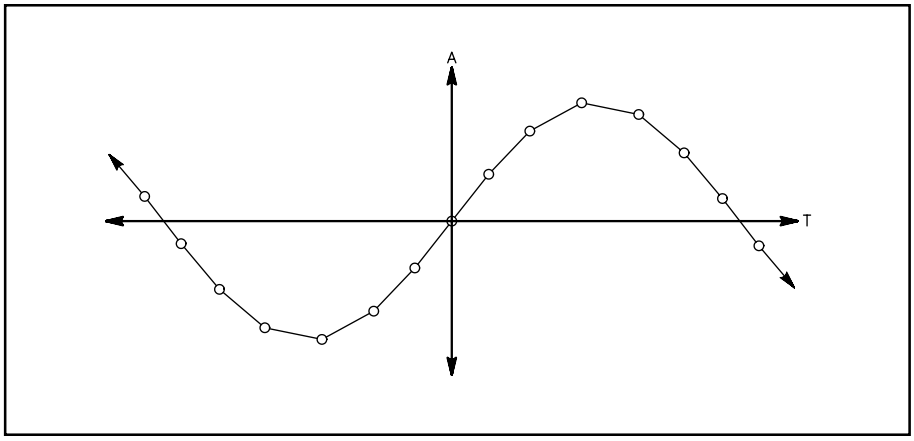

Fig 18.21—Linear piece-wise representation of data resulting from first-order interpolation.

$$\sin(x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 \cdots \tag{33}$$

and

$$\cos(x) = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \frac{1}{8!}x^8 \cdots \tag{34}$$

DIGITAL SPEECH PROCESSING

Virtually all modern transmitters employ fast-attack, slow-decay RF compression: It is called automatic level control (ALC). Because transmitters are usually peak-power limited, some form of gain control is necessary to prevent overdrive of the final RF power amplifier.

RF Compression

A typical ALC system detects the transmitter's envelope with a rectifier and filter, applying this control signal to some gain-controlled stage or stages in the exciter. An increasing level from the envelope-detector results in decreasing gain such that the peak envelope power (PEP) is regulated. ALC is a servo loop employing negative feedback, usually developed only on voice peaks. As the decay time of the detector is decreased, some amplification of parts of speech falling between peaks is achieved. Enhancement cannot exceed the total gain reduction occurring at the voice peaks and usually falls in the range of 3-6 dB. The increase in the transmitter's average output power (talk power) may be quite a bit less than this depending on the characteristics of the voice, especially the *peak-to-average ratio*. In a digital exciter, we may eliminate the need for an analog gain-controlled stage by employing a numeric gain control factor in software and simply regulating the modulator's output level.

Human voices have peak-to-average ratios as high as 15 dB. This does not utilize a peak-limited transmitter very well in SSB mode: At the 100-W PEP level, the average output power might be as little as 3 W! RF compression raises the average output power and tends to further improve intelligibility by bringing out subtle parts of speech. In a digital I/Q modulator, we have a distinct advantage in designing an RF compressor: The RF envelope can be calculated before the modulation is performed. Once the microphone audio has been sampled and converted to an analytic signal, Eq 15 may be used to compute the envelope. To avoid the time-consuming square-root calculation, we may use an approximation:

$$\left\{\begin{array}{l} \text{For: } |I| > |Q|, \left(I^2 + Q^2\right)^{\frac{1}{2}} \approx |I| + 0.4|Q| \\ \qquad |Q| \geq |I|, \left(I^2 + Q^2\right)^{\frac{1}{2}} \approx |Q| + 0.4|I| \end{array}\right\} \tag{35}$$

The envelope signal is used to compress the range of baseband levels prior to modulation so that the peak-to-average ratio is reduced. A block diagram of this system is shown in **Fig 18.22**. The net effect of the system can be shown to be identical to that of a direct, RF compressor. This naturally involves distortion, since the transmitter is no longer linear; however, the distortion produced enhances the syllabic and formant energy in speech without introducing the "mushy" sound caused by heavy audio compression or clipping. As the attack and decay times of an RF compressor are made faster, it approaches the performance of an RF clipper, known to be the most effective form of processing. Because the baseband audio is processed prior to filtering and modulation, occupied BW does not increase much; low-order IMD products will be created, though, that fall within the desired transmit BW. These products ultimately limit the effectiveness of the compressor. This technique may also be applied to receivers.

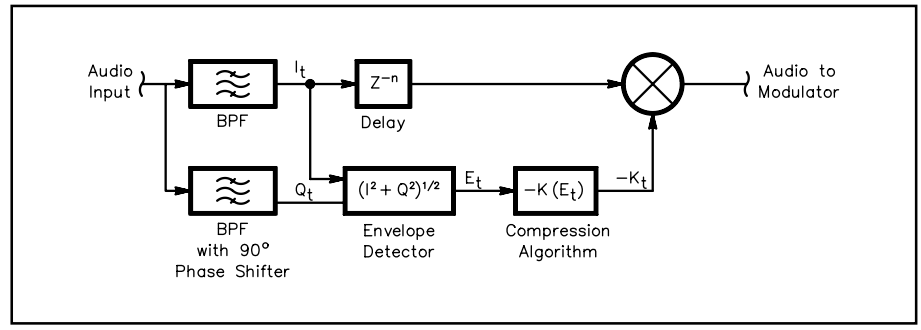

Fig 18.22—Digital RF compressor block diagram.

Audio Compression: Building an AM Transmitter

It has long been a problem to hold the carrier and modulation levels constant in AM transmitters covering several octaves of frequency, such as at HF. Because a baseband signal may not have symmetrical positive and negative amplitudes about its average value, a suitable analog ALC system would be incredibly complex.

In DSP, we may prevent *carrier shift* using adaptive techniques; we prevent over-modulation using an audio compressor. (Refer to **Fig 18.23**.) First, the ratio of drive level to output level, d(t) / y(t), is easily computed by a DSP when the transmitter is on. From this, we can calculate what drive level is required to reach exactly 25% of the peak-power setting. We want the carrier to have this amplitude, regardless of modulation (or lack of it). Second, the baseband signal applied to the modulator must have a maximum peak level equal to the carrier's drive level established above. When the carrier and compressed baseband levels are added, the result is a 100%-modulated AM signal.

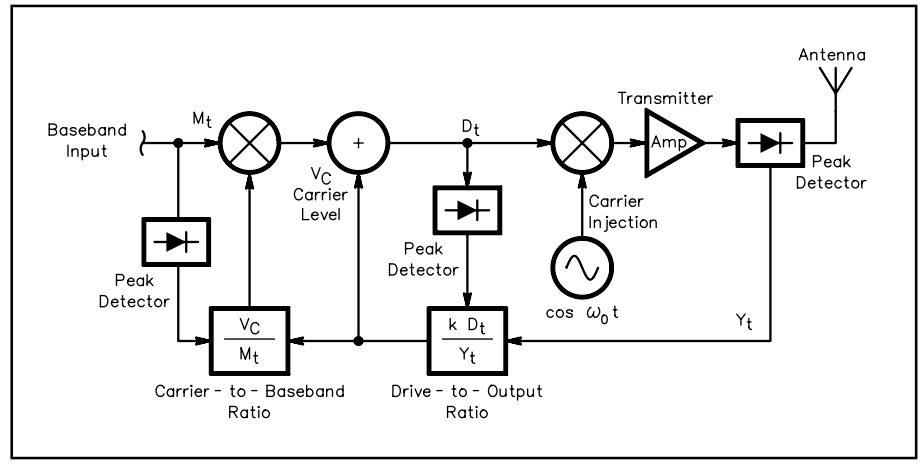

Fig 18.23—AM ALC block diagram.

Fig 18.24 shows this situation, using a baseband signal whose negative excursions are greater than its positive excursions about the average value. Now two servomechanisms are operating in our AM ALC: One continually computes the drive-to-output ratio and sets the carrier level; the other compresses the peak baseband signal to that same peak level. Since the baseband peak detector has to find either the highest negative or highest positive peak, asymmetrical audio inputs may produce an unexpected result: Either the upward or downward modulation may reach 100% before the other can do so. If the downward modulation limits baseband amplitude first, the upward modulation would not cause the transmitter to reach its set PEP level without introducing a carrier shift.

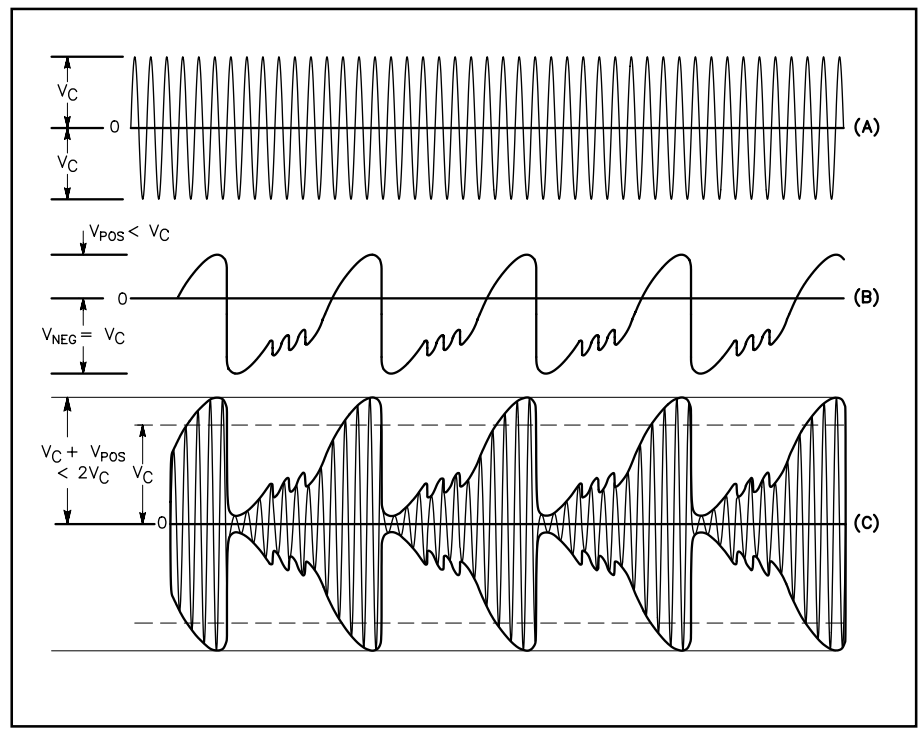

Fig 18.24—AM carrier (A). Baseband input with asymmetrical amplitudes (B). AM modulator output (C).

INTERFERENCE-REDUCTION TECHNIQUES

We touched on the idea of a manually tuned adaptive notch filter using the LMS algorithm. These principles are explored in more detail here, especially as they apply to interference- and noise-reduction systems. The nature of information-bearing signals is that they are in some way coherent; that is, they have some features that distinguish them from noise. For example, voice signals have attributes related to the pitch, syllabic content and impulse response of a person's voice.

Adaptive Filtering

We will find it possible to build an adaptive filter that accentuates those repetitive components and suppresses the non-repetitive (noise). Much research has been done about detection of a sinusoidal signal buried in noise. Adaptive filtering methods are based on the exploitation of the statistical properties of the sampled input signal, specifically, *autocorrelation*. Simply put, autocorrelation refers to how recent samples of a waveform resemble past input samples. We will build an *adaptive predictor*, which actually makes a reasonable guess at what the next sample will be based on past samples. This leads directly to an adaptive noise-reduction system.

An Adaptive Interference Canceler

Imagine we have some sampled input signal, x(t), that we want to adaptively filter to enhance its repetitive content. In the case of a CW signal, all that is required is a BPF centered on the desired frequency. We know that this signal takes the form of a sine wave and that its amplitude will change markedly. Its frequency may not be absolutely constant, either, but we will assume it is fixed for now. We set up an FIR filter structure and an error-measurement system to compare a reference sine wave, d(t), with the output of the filter, y(t). See **Fig 18.25**. Sine wave d(t) is the same frequency we expect the CW tone to be. The difference output, e(t), is known as the *error signal*.

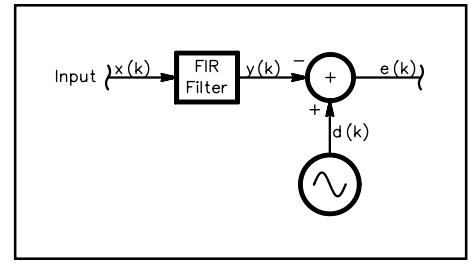

Fig 18.25—An adaptive modeling system.

Now imagine some person is watching the error signal and has their hands on the controls that change the filter coefficients. (See **Fig 18.26**.) Minimizing the error signal by tweaking the coefficients forces the filter to converge to a BPF centered at the frequency of d(t). The speed and accuracy of that convergence is going to depend on how well the person analyzes and reacts to the error data. If it is difficult to tell that a sine wave is present, then adjusting the filter will also be difficult. Further, if the sampling rate is high enough, a person will not be able to keep up; they can check the error only so often or can generate long-term averages of the error.

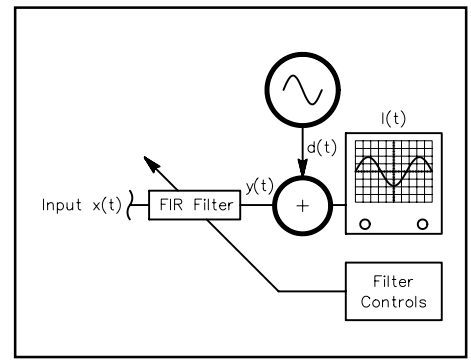

Fig 18.26—An adaptive modeling system, which requires a person at the filter controls.

Using the typical processes of the human mind, the person will soon discover that if they turn the controls the wrong way, the error increases. This information is used to reverse the direction of adjustment; the person then turns the controls the other way. It soon emerges that the person is on a *performance surface*, with an "uphill" and a "downhill," and they know the goal is to go only downhill. So they trash about with the controls, sometimes making mistakes, but ultimately making headway overall down the hill. At some point, the error gets rather small: They know they are near the "bottom of the bowl." Once at the bottom, it is uphill no matter which way they go. The goal of minimizing the error e(t)

has been achieved. They continue gently flailing about with the controls, but always staying near the bottom. This situation is analogous to aligning an analog BPF with an adjustment tool.

After doing this whole thing several times, the person finds that certain rules help speed up the process. First, there is a relationship between the magnitude of the error and the amount they must tweak the controls. If the total error is large, a lot of tweaking must be done; if small, then it is better to make small adjustments to stay near the bottom of the performance surface. Second, there is a correlation between the error, e(t); the input samples, x(t); and the coefficient set, h(t) they need to adjust. Derivation of algorithms providing for steepest descent down the hill is a long and tedious exercise in linear algebra. Let's just say the person goes to school, becomes an expert in matrix mathematics and discovers that one of the fastest and most accurate ways down the hill is to adjust coefficients at sample time t according to:

$$h_{t+1}(k) = h_t(k) + 2\mu e(t)x(t) \tag{36}$$

This is the LMS algorithm. It was developed by Widrow and Hoff in the late 1950s.

Replacing the person with the LMS algorithm, as shown in **Fig 18.27**, we have our manually tuned adaptive interference canceler. Note that both the desired output, y(t), and the undesired, e(t), are available. This is nice in case we want to take only the broadband component and reject the tone. Performance issues of interest include the adjustment error near the bottom of the performance surface and the speed of adaptation. One of the first things we notice about the LMS algorithm is that each of these factors is directly proportional to μ. We select its value, which ranges from 0 to 1, to set the desired properties. A trade-off exists between speed and misadjustment. Large values of m result in fast convergence, but large misadjustment in the steady state. Total misadjustment is also proportional to the number of filter taps, L, and this may place a limitation on the complexity of the filter that may be used. The total delay through the filter also grows with its length; it may become unacceptably large under certain conditions. As in Eq 10, the 3-dB BW of the adaptive BPF is:

$$BW = \frac{2\mu A^2}{t_s} \text{ rad/s} \tag{37}$$

Small values of m result in narrower filters that take longer to adapt. Attempts may be made to adjust m on the fly by using a value that changes is proportion to the error, e(t). A large value is selected initially for rapid convergence, then it is decreased to minimize the long-term misadjustment. This works fine as long as the characteristics of the input signal are not rapidly changing.

An Adaptive Interference Canceler Without An External Reference: An Adaptive Predictor

In the above example, we knew pretty much what to expect at the output: a sine wave of known frequency. What happens when we do not know much about the nature of the input signal, except that it contains coherent components? Quite a few circumstances like this arise in practice. It might seem at first that adaptive processing could not be applied; however, if a delay, z^{-n} is inserted in the *primary input*, x(t), to create the *reference input*, d(t), periodic signals may be detected and thereby enhanced (or elimi-

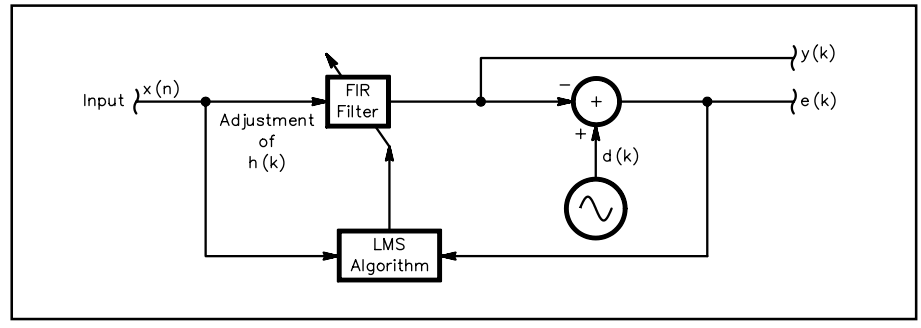

Fig 18.27— An adaptive interference canceler.

nated). See **Fig 18.28**. This delay forms an *auto-correlation offset*, representing the time difference used to compare past input samples with present samples. The amount of delay must be chosen so that the desired components in the input signal correlate with themselves, and the undesired components do not. This is an *adaptive predictor*: Predictable components are enhanced, while the unpredictable parts are removed. Experiments show that for any given value of m, the filter converges quickest when the delay, z^{-n}, is set between one half and one times the filter's total delay.

We may predict this circuit's noise-reduction performance using the ratio of the pre-filtered BW to that of the converged filter:

$$\Delta SNR = 10\log\left(\frac{BW_{input}}{BW_{filter}}\right) = 10\log\left(\frac{BW_{input}}{2\mu A^2 f_s}\right) \tag{38}$$

As an example, for $\mu = 0.005$, $A = 1$, $BW_{input} = 3$ kHz and $f_s = 15$ kHz, the SNR improvement is about 13 dB. When adaptive filters with many taps are used, multiple tones may be either enhanced or notched. Under most conditions, the undesired components are large compared to the desired; enhancement of signals is needed most when the input SNR is low. This situation may not give us enough thrashing about to find our way down the performance surface to convergence. Adding artificial noise to satisfy this condition is tempting, but it turns out we can alter the algorithm slightly to improve our lot without actually adding such noise. These additional terms in the algorithm are known as *leakage terms*.

The unique feature of *leaky LMS algorithms* is a continual "nudging" of the filter coefficients toward zero. The effect of a leakage term can be striking, especially when applied to noise-reduction of voice signals. The SNR increases because the filter coefficients tend toward a lower throughput gain in the absence of coherent input signals. More significantly, leakage helps the filter adapt under low-SNR conditions—exactly when we need noise-reduction the most. One way to implement leakage is to add a small constant of the appropriate sign to each coefficient at every sample time:

$$h_{t+1}(k) = h_t(k) + 2\mu e(t)x(t) - \lambda\{\text{sign}[h_t(k)]\} \tag{39}$$

The value of l may be altered to vary the amount of leakage. Large values prevent the filter from converging on *any* input components, and things get very quiet indeed. Small values are useful in extending the noise floor of the system. In the absence of coherent input signals, the coefficients move geometrically toward zero; during convergent conditions, the total misadjustment is increased to at least l, but this is not usually serious enough to affect signal quality.

An alternate way to implement leakage is to scale the coefficients at each sample time by some factor, g, thus also nudging them toward zero:

$$h_{t+1}(k) = \gamma h_t(k) + 2\mu e(t)x(t) \tag{40}$$

For values of γ just less than unity, leakage is small; values near zero represent large leakage and again prevent the filter from converging. It can be shown that the leaky LMS algorithm is equivalent to adding normalized noise power to the input x(t) equal to:

$$\sigma^2 = \frac{1-\gamma}{2\mu} \tag{41}$$

The leaky LMS algorithm must adapt to survive, much as

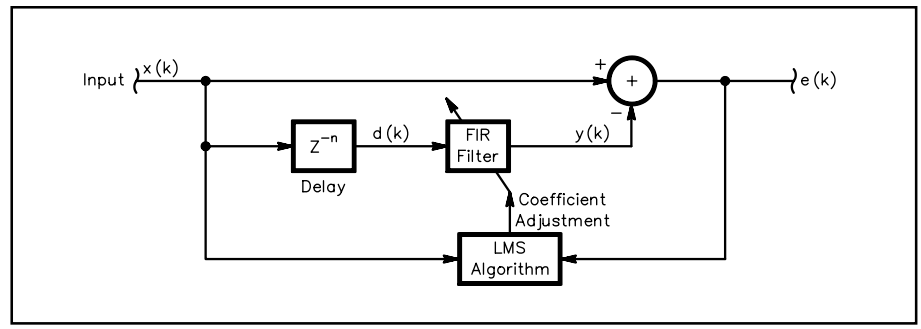

Fig 18.28—An adaptive predictor.

a hummingbird must flap its wings. Were the factor μ suddenly set to zero, the coefficients would all die away, never to recover. Therefore, it is perhaps unwise to use these algorithms with adaptive values of μ. Although values for γ and μ of greater than unity have been tried, the inventors refer to these procedures as "the dangerous LMS algorithm." Enough said.

FOURIER TRANSFORMS

While Fourier transforms are not used exclusively for interference reduction, we present them under that heading here because they are generally superior to adaptive-filtering algorithms in that application. The penalty for this greater effectiveness is an increased computational burden. The relationship Joseph Fourier (pronounced **foor**-ee-ay, 1768-1830) formulated between the application of heat to a solid body and its propagation has direct analogy to the behavior of electrical signals as they pass though filters and other networks. The laws he wrote define the connection between time- and frequency-domain descriptions of signals. They form the basis for DSP spectral analysis, which makes them extremely valuable tools for many functions, including digging signals out of the noise, as we will see.

A Fourier transform is a mathematical technique for determining the frequency content of a signal. Applied to a signal over some finite period of time, it produces an output that describes frequency content by assuming that the section of the signal being analyzed repeats itself indefinitely. Of course, when we analyze a real signal, such as a couple of seconds of speech, we know that those few seconds do not, in fact, repeat endlessly. So at best, the Fourier transform can give us only an approximation of the frequency content. If we looked at a large enough chunk of the signal, that approximation would be pretty good. Certain mathematical conditioning of the input data will help us control the error, even for relatively short analysis intervals.

Originally, the Fourier transform was developed for continuous signals. In DSP, we use a variant of it called the *discrete Fourier transform* (DFT). It is the discrete version because it operates on sampled signals. It is a *block transform* because it converts a block of N input samples into a block of N output *bins*. The input block may be any N contiguous samples. A DFT makes use of complex sinusoids and produces a complex result. When the input data are real, meaning they lack an imaginary part, half the output block consists of the *complex conjugates* of the other half, and so is redundant. When a complex input is used, none of the output bins is redundant.

We learned before that a complex sinusoid is just a pair of waves: a cosine wave and sine wave of the same frequency. Since we will be dragging around a lot of these in the equations below, we introduce a little mathematical shorthand for them called the *Euler identity*:

$$e^{j\omega t} = \cos(\omega t) + j\sin(\omega t) \tag{42}$$

where e is base of natural logarithms. We will shorten this even more later. For each output bin k, where $0 \le k \le N-1$, the DFT is computed as:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}} \tag{43}$$

Expanding Eq 43 using the Euler identity yields:

$$X(k) = \sum_{n=0}^{N-1} x(n)\cos\left(\frac{2\pi nk}{N}\right) - j\sum_{n=0}^{N-1} x(n)\sin\left(\frac{2\pi nk}{N}\right) \tag{44}$$

So each bin has a real part and an imaginary part. Note that each part is calculated using the same convolution sum we saw in Eq 3. Eq 44 is in normal complex-number form: a + *j*b. These coefficients a and b yield the amplitude and phase of the signal x(t) at frequency $\omega = (k\, f_s)/N$:

$$A_k = \left(a_k^2 + b_k^2\right)^{\frac{1}{2}} \tag{45}$$

$$\phi_k = \arctan\left(\frac{b}{a}\right) \tag{46}$$

k is directly proportional to the frequency of its bin according to:

$$f_k = \frac{kf_s}{N}, \text{ for } k < \frac{N}{2} \tag{47}$$

The bins are evenly spaced in frequency by the amount $f_1 = f_s/N$, but there are actually only N/2 real frequencies represented. As mentioned above, half the DFT bins produced from a real input are redundant. Complex inputs may analyze positive and negative frequencies separately.

Working in reverse, we may reconstruct time-domain signal x(t) by summing X(k) for all values of k:

$$x(t) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi kn}{N}} \tag{48}$$

This is the *inverse discrete Fourier transform* (IDFT or DFT^{-1}). It is important to note the duality of the DFT/IDFT relationship. The transforms are not really altering the signal in any way, they are only different ways of representing it mathematically. The strength of the DFT in noise-reduction systems is that it evaluates the amplitude and phase of each frequency component to the exclusion of others.

As far as we can reduce the *resolution BW*, f_s/N, we can eliminate additional noise by artificially zeroing frequency bins not meeting a pre-defined amplitude threshold. Finer resolution BW is obtained by increasing the number of bins, N, decreasing the sampling frequency, or both. Increasing the number of bins, N, involves taking a larger block of N input samples; the larger block represents a longer time span. Obviously we have to wait for N samples to be taken before we can Fourier transform a complete block: A delay of N samples is the result.

Since the DFT assumes the input block repeats indefinitely, we have discontin-uities at the beginning and end of the block where the data have been chopped out of the continuous string of input samples. These abrupt discontinuities cause unexpected spectral components to appear, just as fast on-off keying of a CW transmitter does. This phenomenon is known as *spectral leakage*. Discrete signal components in the input "leak" some of their energy into adjacent frequency bins, smearing the spectrum slightly. Increasing the number of bins, N, helps alleviate this problem. Increasing N moves the bins closer together; a signal that falls between two bins will still cause leakage into adjacent bins, but since the bins are closer together, the spread in frequency will be less. Even so, input components are still spreading their energy over several bins and this overlap makes it difficult to determine their exact amplitudes and phases.

To minimize that problem, we use a technique known as *windowing* on the input data prior to transformation. The data block is multiplied by a *window function*, then used as input to the DFT normally. Window functions are chosen to shape the block of data by removing the sharp transitions in its envelope. Examples of window functions and their DFTs are shown in **Fig 18.29**.

The rectangular window is equivalent to not using a window at all, as all the samples are multiplied by unity. The other window functions achieve various amounts of side-lobe reduction. These window functions are also used to design filters using the Fourier transform method. In fact, these sequences can be used as the impulse responses of prototype LPFs, as should be evident from their frequency responses. Notice that they each involve a trade-off between transition BW and ultimate attenuation. Also note that in the figure, values of ultimate attenuation are plotted without regard to dynamic-range limitations which may be imposed by the bit-resolution of actual systems.

FAST FOURIER TRANSFORMS

In the years before computers, reduction of computational burden was extremely desirable. Many

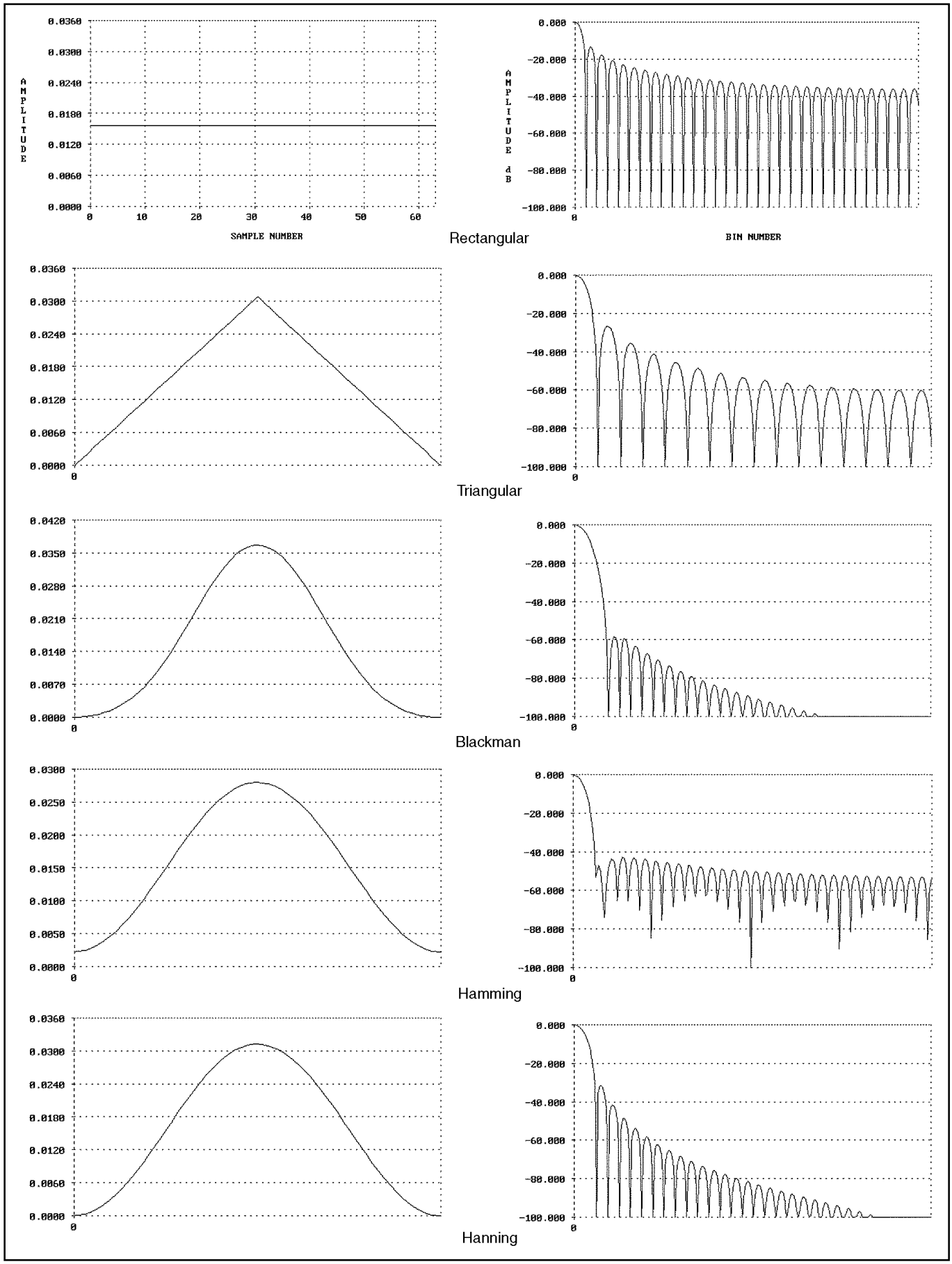

Fig 18.29—Various window functions and their Fourier transforms.

excellent mathematicians, including Runge, applied their wits to the problem of calculating DFTs more rapidly than the direct form of Eq 43. They recognized that the direct form requires N complex multiplications and additions per bin and that N bins are to be calculated, for a total computational burden proportional to N^2. The first breakthrough was achieved when they realized that the complex sinusoid $e^{-j2pkn/N}$ is periodic with period N, so a reduction in computations is possible through the symmetry property:

$$e^{\frac{-j2\pi k(N-n)}{N}} = e^{\frac{j2\pi kn}{N}} \tag{49}$$

This led to the construction of algorithms that effectively break any N DFT computations of length N, into N computations of length $\log_2 N$. Thus, the computational burden is reduced to be proportional to $N\log_2 N$. Because even this much calculation was not practical by hand, the usefulness of the faster algorithms was overlooked until Cooley and Tukey revived it in the 1960s.

To exploit the symmetry referred to, we have to break the DFT computations of length N into successively smaller calculations. This is done by *decomposing* either the input or output sequence. Algorithms wherein the input sequence, x(t), is decomposed into smaller sub-sequences are called *decimation-in-time* FFT algorithms; output decompositions result in *decimation-in-frequency* FFTs. The decomposition is based on the fact that for some convenient number of samples, N, many of the sine and cosine values are the same and products can be combined prior to computing the convolution sums. In addition, other products have factors that are other sine and cosine values. It turns out that electing to decompose by successive factors of 2 produces a very compact and efficient algorithm: a *radix-2* FFT algorithm.

Now for that additional bit of complex-sinusoidal shorthand mentioned earlier. Lots of complex sinusoids will appear in the diagrams to follow, so it sure would be nice to reduce the clutter a bit more. Let's follow the popular DSP text of Oppenheim and Schafer and select the notation:

$$e^{\frac{-j2\pi kn}{N}} = W_N^{kn} \tag{50}$$

This is used in **Fig 18.30** in a flow chart for a complete FFT calculation, for N=8. Multiplication symbols represent complex multiplications, addition symbols represent complex additions. Note that each complex multiplication requires 4 real multiplications and 2 real additions. Complex additions need 2 real additions.

We have 8 input points and 8 output points. Observe that the diagram could not be drawn without crossing many signal paths—there is a lot of calculation going on! Computations progress from left to right in $\log_2 N = 3$ stages; each stage requires N complex multiplications and additions, so the total burden is proportional to $N\log_2 N$. Further, each stage transforms N complex numbers into another set of N complex numbers. This suggests we should use a complex array of size N to store the inputs and outputs of each stage as we go along.

An examination of the branching of terms in the diagram reveals that pairs of intermediate results are linked by pairs of calculations like that shown in **Fig 18.31**. Because of the appearance of this diagram, it is known as a *butterfly computation*.

Making use of another symmetry of complex sinusoids, we can reduce the total multiplications of the butterfly by another factor of 2. A modified butterfly flow diagram is shown in **Fig 18.32**. This calculation can be performed *in place* because of the one-to-one correspondence between the inputs and outputs of each butterfly. The nodes are connected horizontally on the diagram. The data from locations a and b are required to compute the new data to be stored in those same locations, hence only one array is needed during calculation. A complete *8-point* FFT with the modified butterflies is shown in **Fig 18.33**.

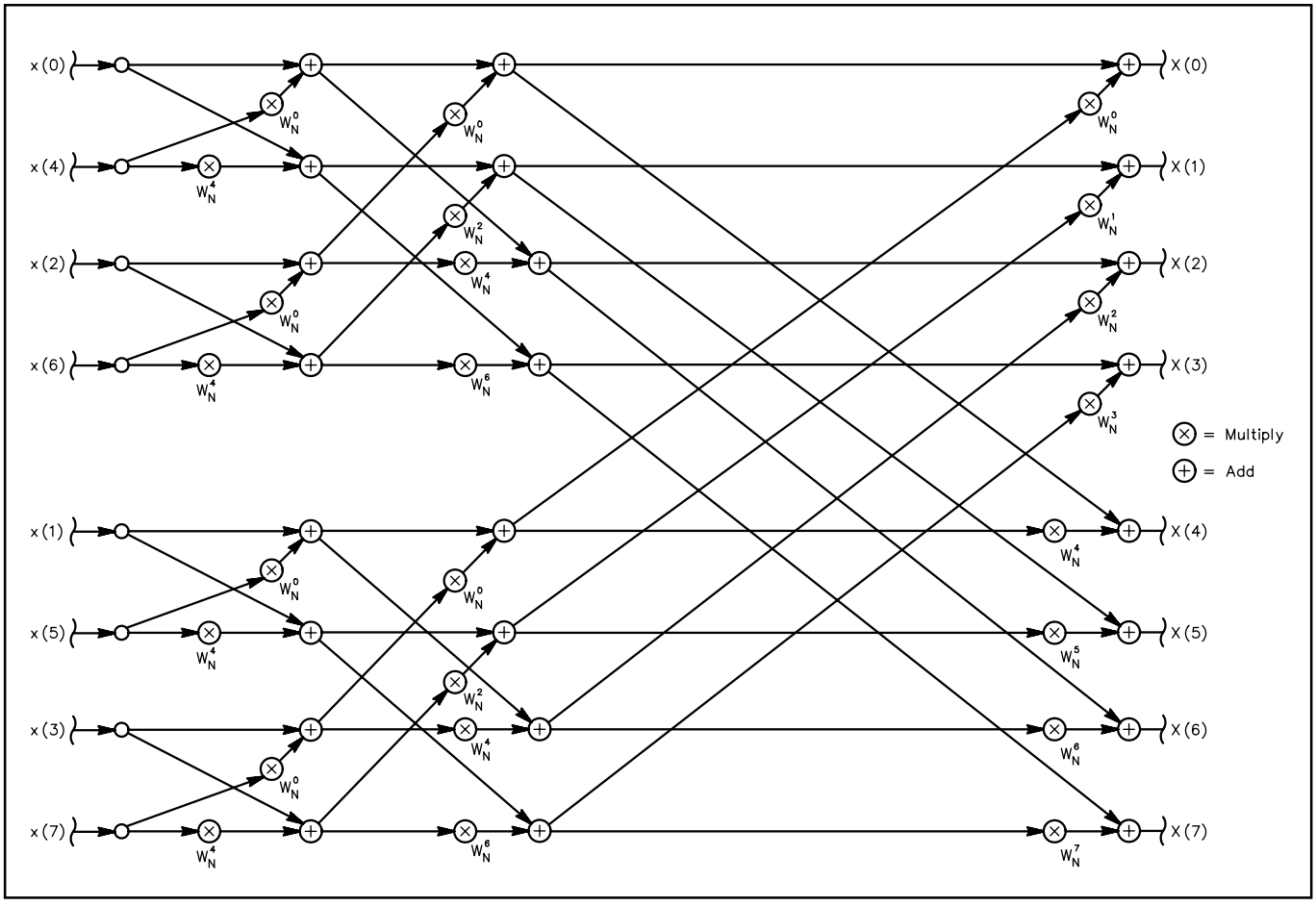

Fig 18.30—Flow chart of an 8-sample FFT.

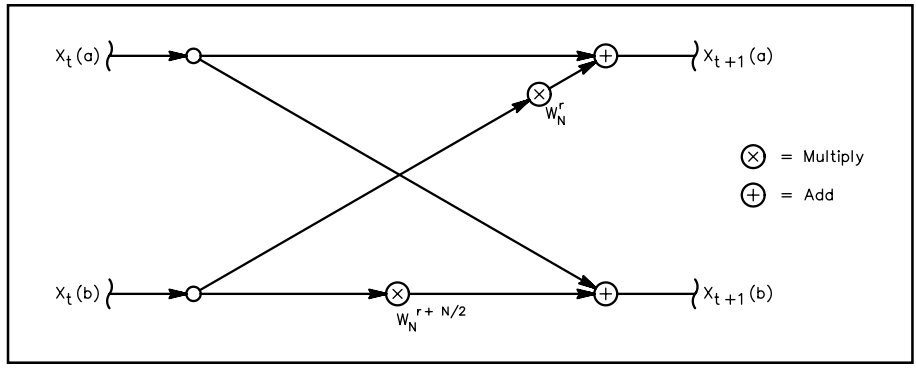

Fig 18.31—Butterfly calculation in a decimation-in-time FFT.

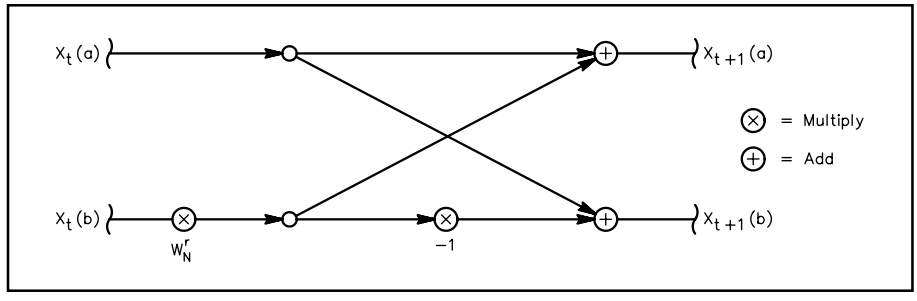

Fig 18.32—Modified butterfly calculation.

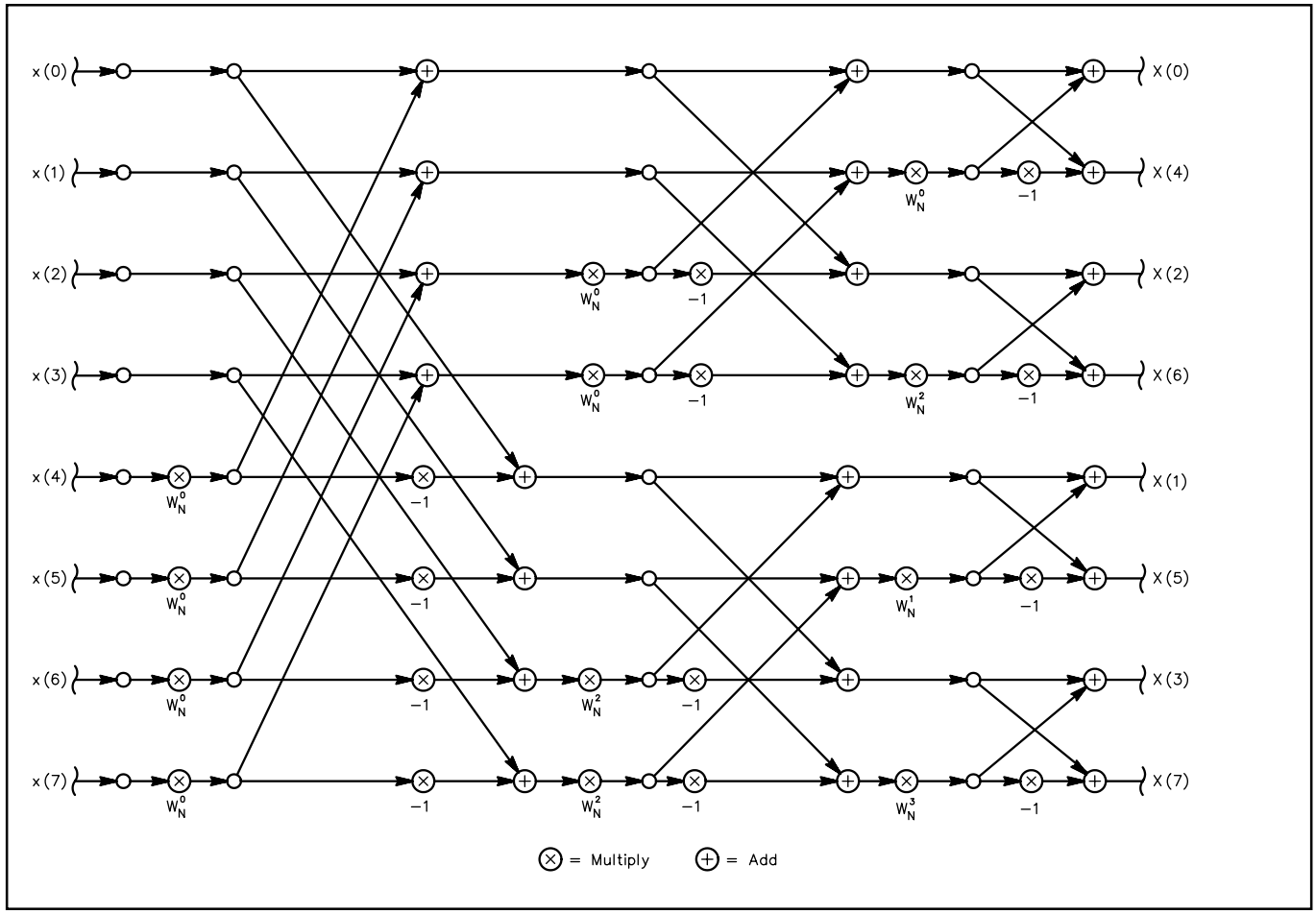

Fig 18.33—Decimation-in-time FFT with different input/output order and modified butterflies.

An interesting result of our decomposition of the input sequence is that in the diagram, the input samples are no longer in ascending order; in fact, they are in *bit-reversed* order. It turns out this is a necessity for doing the calculation in place. To see why this is so, let's review briefly what happens in the decomposition process. We first separate the input samples into even- and odd-numbered samples. Naturally, all the even-numbered samples appear in the top half of the diagram, the odds in the bottom. Next, we separated each of these sets into their even- and odd-numbered parts. This process was repeated until we had N sub-sequences of length one. It resulted in the sorting of the input data in a bit-reversed way. This is not very convenient for us in setting up the calculation, but at least the output arrives in the correct order.

GENERAL FFT COMPUTATIONAL CONSIDERATIONS

While we are on the subject, this business of bit-reversed indexing is the first thing that ties one's brain in knots during coding of these algorithms, so let's have at it. Several approaches are feasible to translate a normally ordered index to a bit-reversed one: a look-up table, the bit-polling method, reverse bit-shifting and the reverse counter approach.

The look-up table is perhaps the most straightforward approach. The table may be calculated ahead of time and the index used as an address into the table. Most systems do not require very large values of N, so the space taken by the table is not objectionable.

For more space-sensitive applications, the bit-polling method may be attractive. Since the bit-reversed indices were generated through successive divisions by two and determination of odd or even, a tree

structure can be devised that leads us to the correct translation, based on bit-polling. See **Fig 18.34**. The algorithm examines the least-significant bit, then branches either upward or downward based on the state of the bit. Then the second least-significant bit is examined and another branch taken, and so forth, until all bits have been examined.

The bit-shifting method requires about the same computation time as bit-polling. Two registers are used: one for the input index shifting right through the carry bit, the other shifting left through carry. After all the bits have been shifted, the left-shifting register contains the result. See **Fig 18.35**.

Finally, Gold and Rader have described a flow diagram for a bit-reversal counter than can be "decremented" each time the index is to change. If data are actually to be moved during sorting, the exchange is made between data at input index n and bit-reversed index m, but only once. That is, only N/2 exchanges need be performed.

During the actual calculations, indexing of data and coefficients requires attention to many details. In particular, several symmetries about offsets of the index may be exploited. At the first stage of Fig 18.32, all the multipliers are equal to $W_N^0 = 1$, so no actual multiplications need take place; all the butterfly inputs are adjacent elements of the input array x(t). At the second stage, all the multipliers are either W_N^0 or integral powers of $W_N^{N/4}$ and the butterfly inputs are two samples apart, and so forth.

The coefficients are indexed in ascending order. These are normally calculated ahead of time and stored in a table. Another way is to use a *recursion formula* to generate them on the fly, but this is discouraged because of numerical-accuracy effects that destroy the efficiency of the technique.

All those multiplications and additions take their toll on the numerical accuracy of our final result. Quantization noise is multiplied and added as well, and at the output of a DFT, the noise power grows by N times.

In an FFT calculation, the situation is roughly the same; however, the requirement to avoid overflow at intermediate stages may force us to scale the data, the coefficients, or both. This further reduces the dynamic range of any FFT. Results have been offered indicating noise increases in the vicinity of 12N. In addition, the quantization-noise contribution of the coefficients increases in inverse proportion to p, the number of bits used to represent them. This, in turn, means that the noise increase with respect to N is slow.

In FFT-based noise-reduction systems, we perform some modification of the frequency-domain

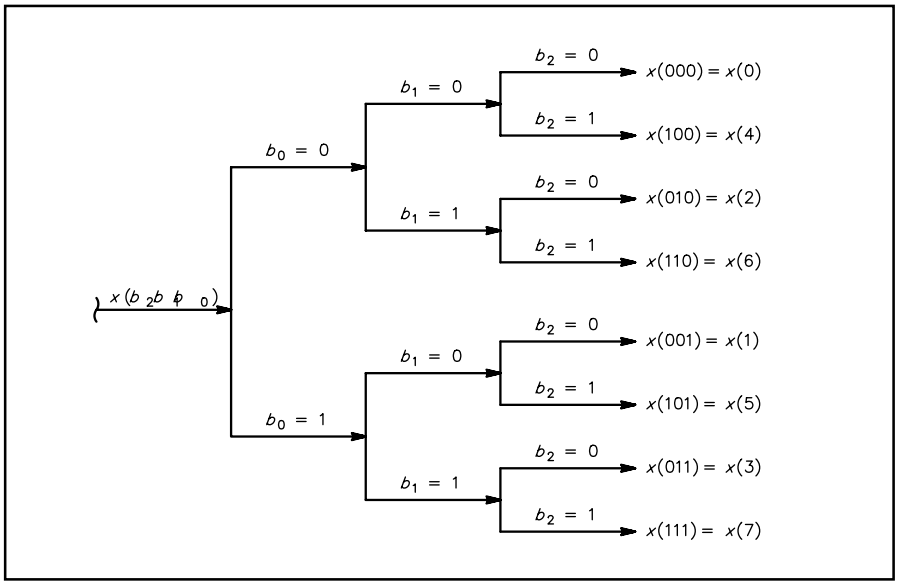

Fig 18.34—Polling tree for bit reversal.

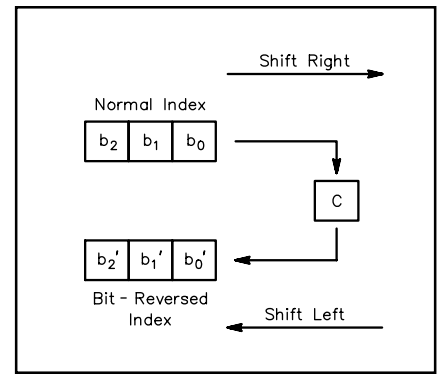

Fig 18.35—Register arrangement for bit-reversal shifting.

data, such as zeroing bins not meeting a pre-defined amplitude threshold. Then we transform the modified data back to the time domain. The duality of the Fourier transform and its inverse can be shown in the flow diagram of a FFT^{-1} as in **Fig 18.36**. This diagram was produced from Fig 18.33 by simply substituting $^1/_2W_N^{-kn}$ for W_N^{kn} at each stage and, of course, using X(k) as the input to obtain x(t) as the output.

Alternatively, we may compute the FFT^{-1} by using the FFT flow diagram and swapping the inputs and outputs and reversing the direction of signal flow. It is important to note that this is a consequence of that fact that we can rearrange the nodes of the flow diagrams however we want, so long as we do not alter the result. The transforms work just as well in reverse as they do in the forward direction.

DAMN-FAST FOURIER TRANSFORMS

When it is necessary to compute Fourier transforms on a sample-by-sample basis, or where frequency resolution must be non-uniform across the sampling BW, even traditional FFTs may be too computationally intensive for the processing horsepower available. A class of algorithms that computes the next transform output very rapidly—based solely on current transform output and the next input sample—has been discovered. A method is included here for controlling its inherent divergence problem by brute force.

The derivation begins by looking at how the Fourier transform results change for each bin at each sample time. Say we start with some discrete Fourier transform output bins $X_r(k)$ at sample time r. Then we compute the DFT for the next sample time r + 1 and examine the sequences to see what has changed.

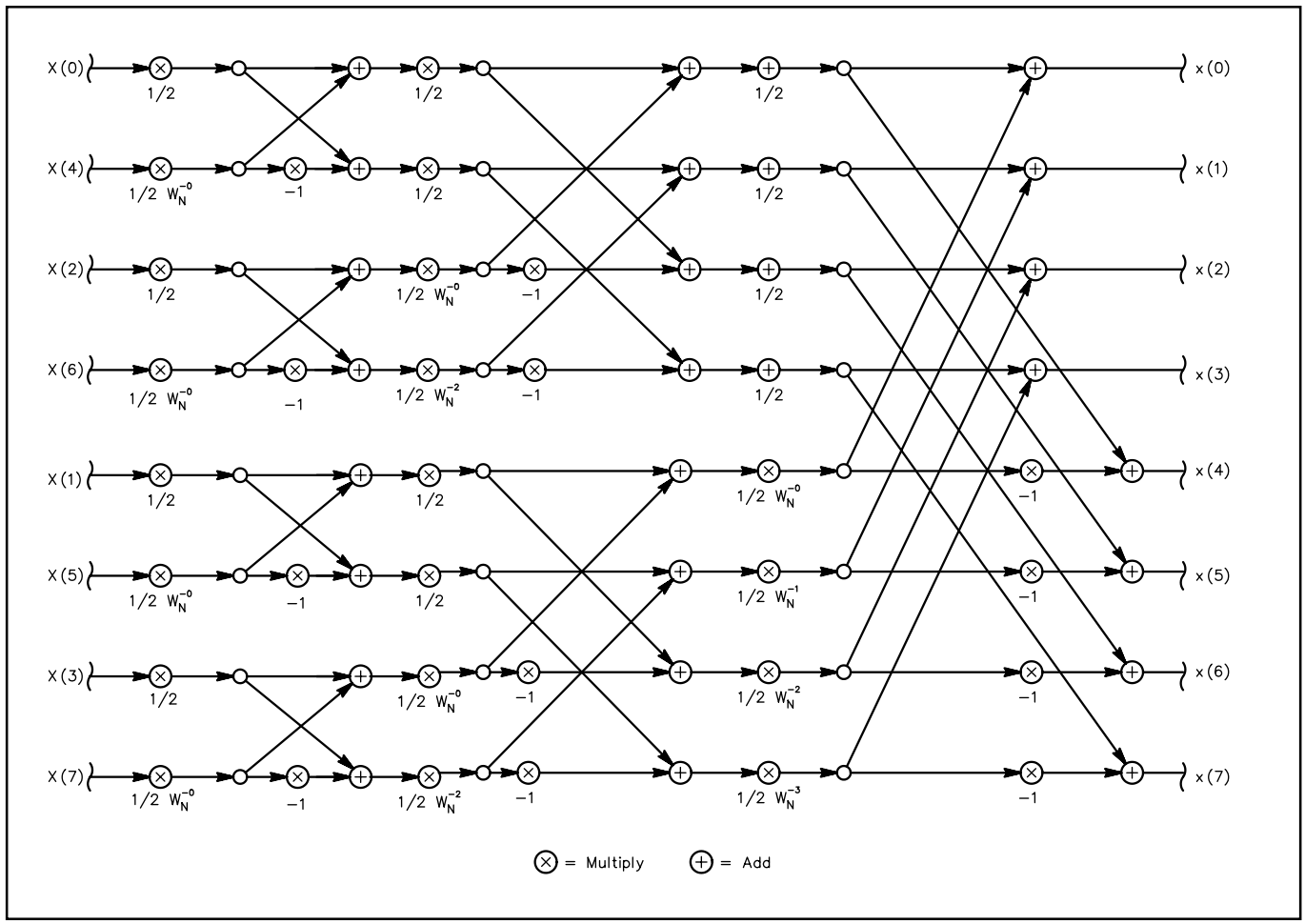

Fig 18.36—FFT^{-1} implemented by interchange of inputs, outputs, and coefficients.

For r = 0, each DFT sequence expands to:

$$X_0(k) = W_N^{0k}\,x(0) + W_N^{1k}\,x(1) + W_N^{2k}\,x(2) + \ldots + W_N^{(N-1)k}\,x(N-1)$$
$$X_1(k) = W_N^{0k}\,x(1) + W_N^{1k}\,x(2) + W_N^{2k}\,x(3) + \ldots + W_N^{(N-1)k}\,x(N) \quad (51)$$

What is evident is that each input sample x(n) that was multiplied by W_N^{nk} in the summation for $X_0(k)$ is now multiplied by $W_N^{(n-1)k}$ in the summation for $X_1(k)$. The *ratio* of the two sequences is nearly:

$$\frac{X_1(k)}{X_0(k)} \approx \frac{W_N^{(n-1)k}}{W_N^{nk}} = W_N^{-k} \quad (52)$$

We still have two terms "hanging out" of the relationship, namely the first and the last:

$$W_N^{0k}\,x(0) = x(0) \text{ and } W_N^{(N-1)k}\,x(N) \quad (53)$$

that have not been accounted for in the ratio. If we first subtract x(0) from $X_0(k)$ before taking the ratio, then add the new term $W_N^{(N-1)k}x(N)$ after, we have the correct result:

$$X_1(k) = W_N^{-k}\left[X_0(k) - x(0)\right] + W_N^{(N-1)}\,x(N) \quad (54)$$

Now this may be simplified a little, since:

$$\begin{aligned} W_N^{(N-1)k} &= e^{\frac{-2\pi j(N-1)k}{N}} \\ &= e^{\frac{-2\pi jN}{N}} \bullet e^{\frac{2\pi jk}{N}} \\ &= W_N^{-k} \end{aligned} \quad (55)$$

and substituting:

$$X_1(k) = W_N^{-k}\left[X_0(k) - x(0) + x(N)\right] \quad (56)$$

This is the damn-fast Fourier transform (DFFT). It means: For N values of k, we can compute the new DFT from the old with N complex multiplications and 2N complex additions, or a computational burden proportional to N. If we begin with $X_0(k)=0$ and take the first N value of x(n)=0, we can start the thing rolling. It saves computation over the FFT by a factor of:

$$\frac{N\log_2 N}{2N} = \frac{\log_2 N}{2} \quad (57)$$

which for large values of N is very significant indeed. For example, if N = 1024, the improvement is by a factor of five. Over the direct-form DFT, it is a factor of N^2/N faster. But there is a catch: An error term will grow in the output because the truncation and rounding noise discussed previously is cumulative. The error will continue to grow unless we do something about it.

The simplest way to handle the situation is to compute two DFFTs for all the output bins k, resetting every other block of N input samples to zero. In other words, one DFFT begins at some time with an input buffer that has been zeroed, the other continues to operate on the continuous stream of real input samples. As sample-taking continues, DFFT output is taken from the second calculation. As the buffer of the first DFFT gradually fills with real samples, the block of zeroes it originally held disappears. At this point,

each DFFT produces the same result except for the greater error in the second DFFT because of truncation and rounding effects. Output is then taken from the first DFFT and the buffer of the second is zeroed; the calculations continue for another N iterations, at which time the exchange and reset are again done, and so forth, continually. This places an upper bound on the cumulative error to that associated with 2N iterations and increases the computational burden by a factor of two. Now the savings over the FFT is only:

$$\frac{\log_2 N}{4} \tag{58}$$

which for N > 16 still represents an improvement. DFFT output quantization noise is at least twice that of the DFT.

Frequency resolution of DFFTs is controlled by the block length, N, used in the calculations, just as in DFTs or FFTs. Resolution may be set differently, though, for each bin; further, not all bins need be computed to compute any particular bin, unlike the Cooley-Tukey FFT. Is there an inverse DFFT? Well, because inverse Fourier transforms map into the time domain, it is simple enough to just compute the next output sample rather than the next N output samples. The easiest output term to compute is x(0), since all coefficients are $W_N^0=1$. The output is then just:

$$x(0)=\frac{1}{N}\sum_{k=0}^{N-1}X(k) \tag{59}$$

and only one multiplication is involved.

RADIO ARCHITECTURES FOR DSP

SUPERHETERODYNE WITH BASEBAND DSP

It is common these days to apply DSP techniques at audio or baseband, especially by using outboard processing units with older, analog receivers. A drawback to this approach is that while a receiver's selectivity may be improved this way, special gain-control settings must be employed. To see why this is so, let's look at a typical arrangement, shown in **Fig 18.37**. The receiver's BW is 3 kHz and we wish to use the outboard DSP unit to implement an RTTY filter with BW = 500 Hz. It follows that some of the signals we digitize are undesired and this raises a problem. When the desired signal is strong relative to the undesired, everything is fine; however, when a strong undesired signal appears within the receiver's BW but outside our DSP filter's BW, the receiver's AGC acts on the combination, reducing the level of our desired signal, as well as that of the undesired signal. We may elect to solve this in several ways; each involves digitally adjusting the gain applied to the desired signal to keep its level constant—the goal of any AGC system.

A block diagram of one such *digital AGC system* is shown in **Fig 18.38**. It consists of a gain-control block (multiplier) and a power-ratio detector. The detector computes the ratio of the sum of the undesired signal's peak amplitude, m, and the filtered signal's amplitude, n, to n:

$$k = \frac{m + n}{n} \qquad (60)$$

k is the factor by which the filtered output must be digitally boosted to keep its peak level constant. This detector includes a fast-attack, slow-decay filter. The decay rate is chosen to match that of the receiver's analog AGC. Analog AGCs usually have a decay that—when plotted as dB *vs* seconds—looks fairly close to a straight line. This exponential decay is achieved in DSP by multiplying the stored detector value by a constant near unity at each sample time. When the ratio k suddenly jumps upward, along with m + n, the stored value is updated immediately to get a fast attack.

The gain-controlled stage is simply a multiplier; its inputs are k and the filtered signal. Note that k • 1 always, hence the multiplication is not the simple fractional type described above. We may now have a need to extend our fixed-point math to values greater than one. This is tedious but not too difficult. We just handle the integer and fractional parts separately. Additions and subtractions are straightforward,

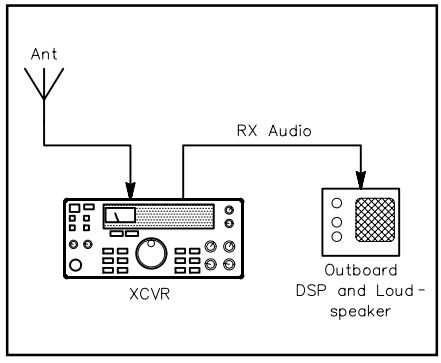

Fig 18.37—A typical use of an outboard, baseband DSP processor.

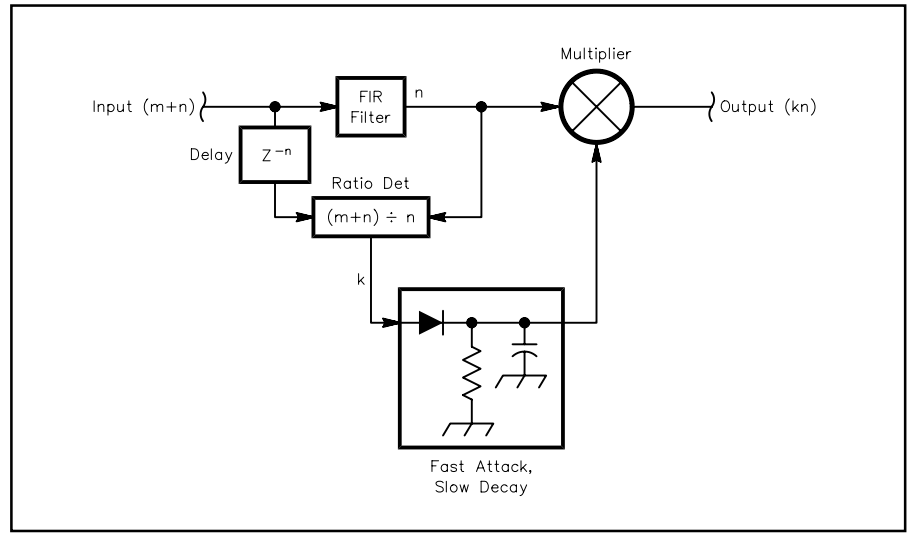

Fig 18.38—Digital AGC system block diagram.

but we have to multiply k by a fractional decay factor, d, then multiply k by another fraction—the filtered signal—at each sample time. Separating the integer and fractional parts by a radix point, we adopt the notation k = (a.b) where:

$$a \in \mathfrak{I}.b \in \mathcal{F} \tag{61}$$

meaning that we treat a as an integer, and b as a fraction. Numbers a and b are ordinarily represented in binary. A number whose absolute value is less than unity has a zero integer part: d = (0.d). The result of the multiplication (a.b)(0.d) = e.f is:

$$(a.b)(0.d) = ([\mathfrak{I}ad + \mathfrak{I}bd] \times [\mathcal{F}ad + \mathcal{F}bd]) \tag{62}$$

requiring four real multiplications, just as in complex math.

A delay is inserted in the path of input signal m to compensate the delay through the DSP filter. Scaling might be necessary to prevent overflow in the gain-controlled stage. Special attention must be paid to what happens during the attack time. Some receivers exhibit *AGC overshoot*, which may cause spikes on incoming signals, resulting in rapid gain excursions. A good approach seems to be to allow gain adjustment in proportion to the attack time of input signal m, but only if m persists at that level for several milliseconds, to avoid triggering on noise pulses.

In practice, baseband DSP filtering may be limited by *in-band IMD* and synthesizer *phase-noise* effects that plague the analog transceiver itself. These cause unwanted signals to appear in the passband, masking the desired signal. With a perfect receiver, performance is limited by the available SNR and SFDR of the ADC in use and by the phase noise of its clock. Noise-reduction algorithms may be very effective, though, even in the face of these margins. As we move the digitization point closer to the antenna, converter noise and phase-noise issues become more critical; other factors actually aid in the resolution of some of the problems outlined above as we go to IF-DSP.

Of course, a receiver's AGC may be disabled to avoid this kind of arrangement, but only with some degradation of dynamic range.

IF-DSP AT A LOW IF

The primary reason for wanting to digitize signals closer to the antenna is to eliminate expensive filters and other hardware whose functions can be performed in DSP. By going to a low IF, we can get rid of balanced modulators and multiple crystal or mechanical filters; demodulation, squelch, and digital AGC duties are also done in software; many things judged quite difficult or impossible in the analog world may be included, as well.

To do it in a receiver, we apply harmonic sampling and a fast, 16-bit sigma-delta ADC at an IF just above the audio range. This IF is selected to be comparable with the ultimate BW of the *roofing filters* used in the receiver's front end. Recall that in harmonic sampling, the sampling frequency may be as low as the IF minus half its BW. An IF BW of 15 kHz, for example, requires a sampling frequency of at least 30 kHz. The center IF itself may be almost anything greater than 7.5 kHz at this minimum sampling frequency. We ought to consider what *image rejection* we are going to get based on such a low IF, though. Roofing and other analog filters will determine it by their attenuation at an offset from center equal to twice the IF. If we intend to use the same IF in transmit mode, the 2nd LO will appear at an offset equal to the IF. Quite a few poles of filtering in the analog sections are still required around this arrangement. See **Fig 18.39**.

From the antenna, signals are low-pass filtered to remove first-mixing image responses and to eliminate LO leakage. Then, they are mixed to a VHF first IF to dodge as many spurious responses as possible. A VHF first IF may be selected above twice or even three times the highest RF to get away from second and third-order mixing products. Six to eight poles of crystal BPF may be used in the strip, with several gain-controlled stages interspersed. A traditional IF analog AGC is employed. In any receiver design,

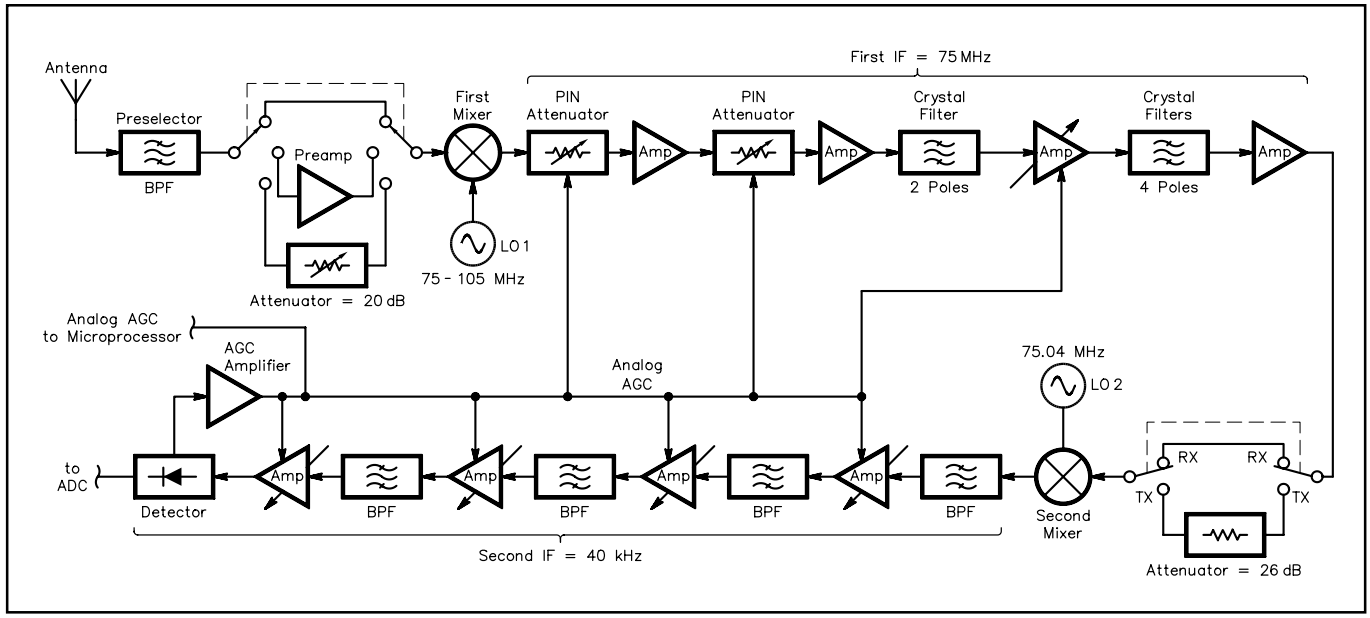

Fig 18.39—IF-DSP receiver block diagram.

it is best to distribute gain and loss evenly to avoid degradation of the SNR under reduced-gain conditions. We would like SNR to continue increasing as the input signal increases, as far as possible. Gain reduction, therefore, is usually made to occur at the stages closest to the antenna first, followed by subsequent stages.

First-IF signals are converted directly to the low IF, then amplified and possibly filtered further. Enough amplification is needed to raise the second-IF signal to within about 10 dB of the maximum ADC input level. This maximizes the SNR available from the ADC and the dynamic range available for digital AGC operation, as described in the Baseband section above. A 10-dB margin leaves the *headroom* needed to accommodate analog AGC overshoot and noise spikes. Overload of the ADC is catastrophic and must never be allowed to occur. IF-DSP architectures that include analog AGC usually must also have digital AGC loops. Note that it is possible to build an analog front end that has the required dynamic range (≥115 dB) and be rid of analog AGC, but this would entail some more-expensive hardware and tighter tolerances. Analog AGC makes it easier to keep the front end linear over the range of input signals expected. These days, receivers may be called on to handle input signals as large as one watt!

One way to handle the situation is to employ a gain-compensation scheme such as that described in the Baseband-DSP section above. We may, in addition, arrange for the DSP to monitor the analog AGC voltage to find out what it is doing. Knowledge of the receiver's AGC voltage-*vs.*-gain curve leads to a new algorithm: When the analog gain is changing rapidly, rapid changes in gain-control variable k are allowed, as in the baseband case. When time constants are set correctly, we do not even have to know the exact amount of analog gain reduction. If we want the digital AGC's threshold to reside above the receiver's noise floor, though, we *do* have to know the gain reduction accurately.

Traditionally, HF receivers have been designed with analog AGCs that have thresholds or "knees" around 3 μV. This means that signals below that threshold are not gain-controlled; when input signals are low, the receiver gets quiet. Note that this parameter is unrelated to the SNR as determined by the noise floor of the receiver. Digital AGCs may be designed with any threshold—it may even be made variable via a front-panel control. The effect is much the same as that produced by an IF-gain control. A *knee-less* digital AGC system may result in operator fatigue, because receiver and atmospheric noise are boosted in the absence of strong input signals.

Another approach to analog AGC in a digital receiver involves generating the gain-control voltage

using a detector in the digital portion. AGC voltage comes from a DAC controlled by the DSP. See **Fig 18.40**. The delay between detection of IF signals and the application of gain control causes the same problem as in traditional analog AGCs: The loop filter must be optimized with regard to amplitude and phase response so it can minimize overshoot. Delays encountered in DSP filters may require the use of a secondary ADC and detector, solely for AGC purposes. Some designers have even gone as far as to build separate IF strips for detection and application of analog AGC, all the way up to and including the receiver's first IF. Such architectures tend to rapidly become complicated: One of the other schemes detailed above should suffice for Amateur Radio applications.

Conversion plans used in IF-DSP receivers may also be used in the transmitter by simply swapping the LOs, inputs and outputs. A switching arrangement for this is shown in **Fig 18.41**. Isolation between the ports of the LO's DPDT switch must be set to the desired level of spectral purity. An example of such a switch is shown in **Fig 18.42** using PIN diodes at VHF. Switch control voltages swing between +5 V and –5 V; when the series diodes are on, the shunt

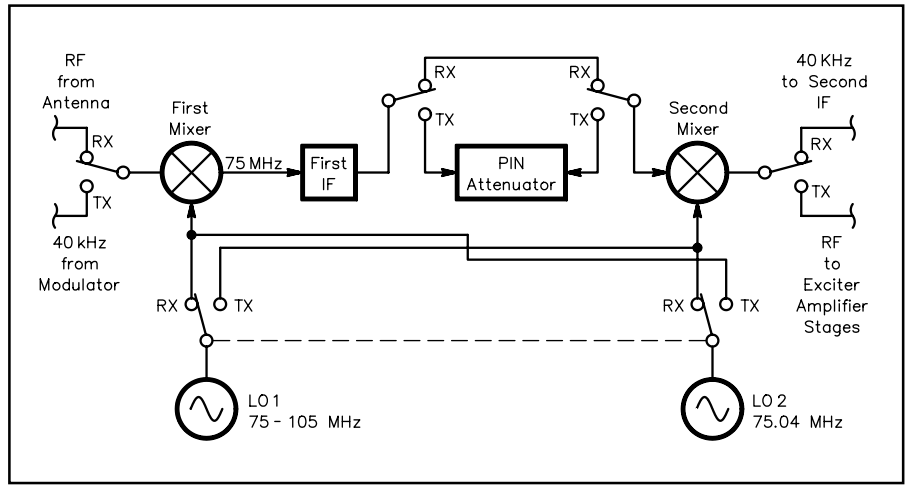

Fig 18.41—Block diagram of IF-DSP conversion scheme with T/R switching added.

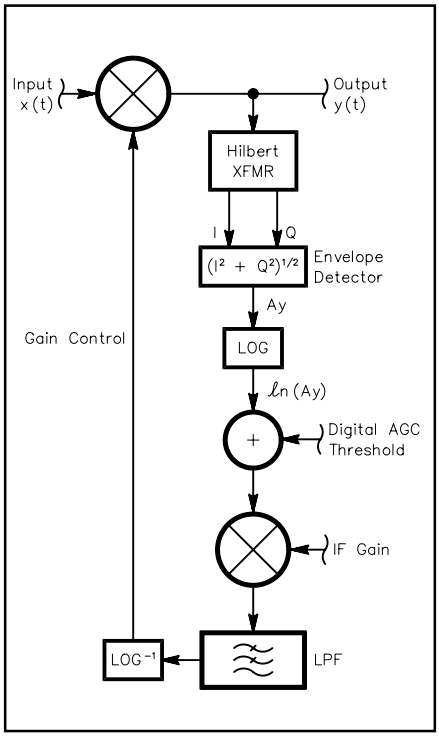

Fig 18.40—IF-DSP receiver with digitally derived analog AGC.

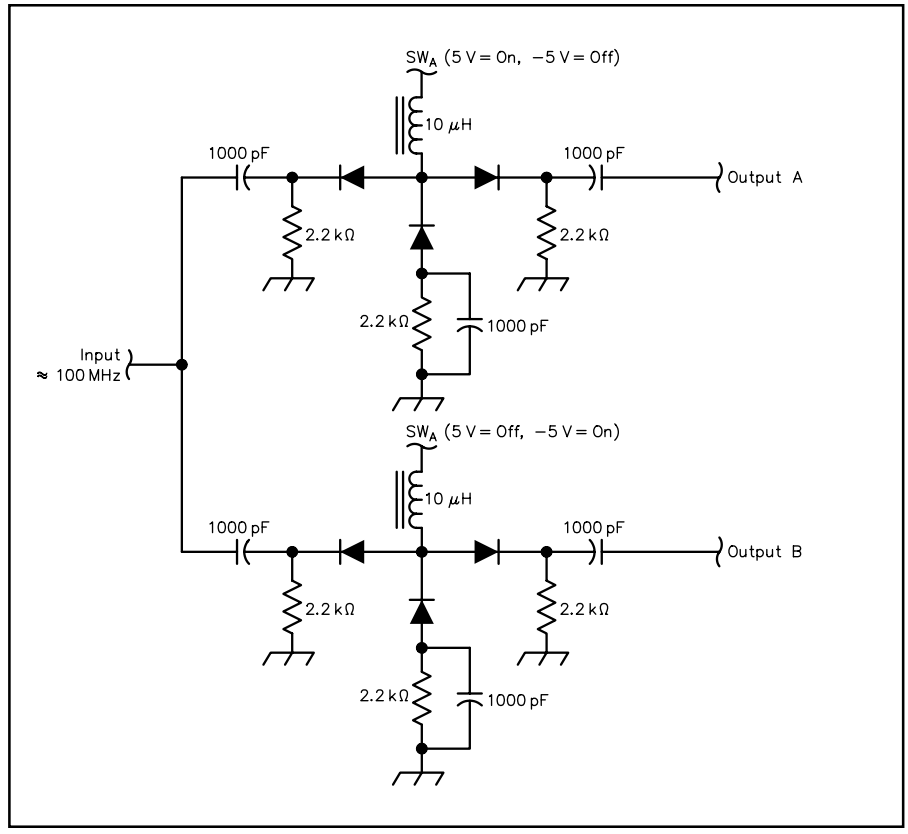

Fig 18.42—SPDT LO switch using PIN diodes. Diodes are Philips BA682 or equiv.

diodes are off, and vice versa. This particular circuit was designed for a 75-105 MHz LO and achieves better than 80 dB of isolation between the ports. Switching of the first mixer's input is best achieved by a relay for HF circuits, PIN diodes for VHF and above. The second mixer's output may be switched using various commercial ICs, such as the Signetics NE630. Isolation in these switches is important because it reduces spurious responses in both receive and transmit modes.

Gain-controlled stages or step attenuators may have to be employed to provide for a difference in IF-strip gain between receive and transmit. To see why this might be necessary, examine the difference in gain between a receiver and a transmitter in typical service. A receiver takes as little as –132 dBm from the antenna and amplifies it to around 1 W (+30 dBm) at the loudspeaker. The power gain is:

$$\mathrm{GAIN_{RX}} = 30 - (-132)\,\mathrm{dBm} = 162\,\mathrm{dB} \tag{63}$$

In a transmitter, a typical dynamic microphone might produce 5 mV RMS into 600 Ω, or –44 dBm. To get to 100 W or +50 dBm, the gain is:

$$\mathrm{GAIN_{TX}} = 50 - (-44)\,\mathrm{dBm} = 94\,\mathrm{dB} \tag{64}$$

The receiver has the far more difficult task, but the transmitter is still doing yeoman's duty. Considering a maximum path loss of:

$$\mathrm{LOSS_{PATH}} = 50 - (-132)\,\mathrm{dBm} = 182\,\mathrm{dB} \tag{65}$$

it is a wondrously large amount of enhancement we get from our electronics, since the total power gain from the microphone on one end to the loudspeaker on the other must be:

$$\mathrm{GAIN_{TOTAL}} = 162 + 94 = 256\,\mathrm{dB} \tag{66}$$

or a factor of 4×10^{25}!

DIRECT DIGITAL CONVERSION

In the ultimate digital receiver, signals are sampled directly from the antenna without any intervening stages. In practice, though, some gain is required ahead of the ADC because of the present limitations of data-conversion technology. As far as gain stages can be made with high dynamic range and good large-signal-handling capability, direct digital conversion (DDC) comes within reach. In this technique, signals are converted directly to baseband without any intermediate analog mixing stages. Refer to **Fig 18.43**. The LO is, in effect, placed very close to the desired signal and through harmonic sampling and decimation, translates it to baseband. The closeness of the LO to the signal accentuates phase-noise effects such as reciprocal mixing (discussed above) and makes short-term clock stability a major issue. Fortunately, low-noise, crystal-derived clock designs are becoming available. RMS clock jitter is usually specified in units of time (picoseconds rms), but a clock's phase-noise-*versus*-frequency characteristic tells the entire tale.

The Nyquist criterion compactly determines the sampling rate required for any given signal or group of signals. If the digitized BW is 50 kHz, the minimum sampling rate is 100 kHz, even if the signal's frequency is in the VHF range or beyond. Ancillary sample-and-hold devices may be employed in a DDC receiver to ease the BW requirements for the ADC. The digitized BW must remain within half the final sampling frequency to avoid aliasing; for this reason, interest in narrow pre-selector filters has been renewed.

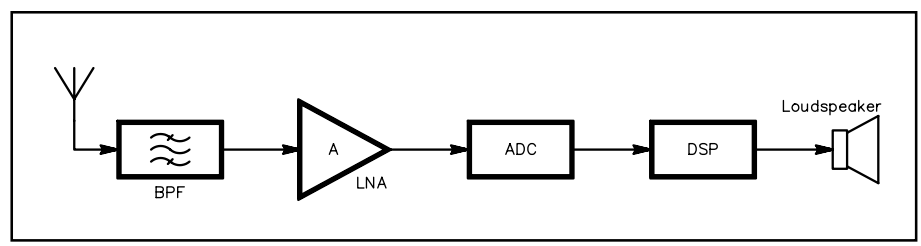

Fig 18.43—DDC receiver block diagram.

In the example of a 50-kHz

received BW, any increase in sampling rate above 100 kHz is called *over-sampling*. Over-sampling is important because it allows for an SNR gain by spreading quantization noise over a larger BW, then filtering it, as discussed above. We are using harmonic sampling, though, so we are also *under-sampling* our signal. We can be both over-sampling and under-sampling at the same time since one is defined with respect to BW and the other by the frequencies of interest.

Frequency planning is of special concern in DDC architectures. Quite often, spurious responses appear in high-speed ADC and DAC outputs that we must plan to avoid. Those are largely responsible for establishing the SFDR of high-speed converters. Problems are also created in supposedly linear stages that generate significant harmonic content, since those harmonics show up as aliases in the digitized spectrum and may mix with other products. Careful selection of sampling frequency and IF can place these spurs where they are harmless: outside the band of interest. Over-sampling only moves us toward the goal by providing more spectrum into which spurs may harmlessly fall.

The technique known as dithering further improves SFDR by spreading the energy contained in spurs—those caused by the DNL of data converters—over greater BWs. Dithering artificially adds noise to the clock input of the ADC or DAC to achieve this spectral spreading. Typical values used are in the range of 10-30 bits peak-to-peak. Spurious reduction of over 20 dB has been attained with high-speed (40-Msample/s) converters.

ADVANCED DSP TOPICS IN COMMUNICATIONS

A few advanced subjects are worth briefly mentioning here. They represent exciting trends in DSP communications science that hold significant promise for the future.

ADAPTIVE BEAMFORMING

Thus far, we have considered the application of DSP techniques to processing of signals in the time and frequency domains. *Adaptive beamforming*—the creation of antenna systems with automatically varying patterns—extends the concept of adaptive DSP to the spatial domain. The main goal of "smart-antenna" methods is to condition the radiation pattern of an array so as to maximize received SNR of the signal of interest, and to minimize interference and noise on an adaptive basis. In this sense, adaptive means that the antenna array's pattern changes in response to changes in the strength and direction of desired and undesired signals.

An adaptive antenna array consists of two or more antennas separated in space, connected to a multi-channel, adaptive signal processor. At least one of the antennas in the array feeds the DSP through an adaptive filter. See **Fig 18.44** for an illustration of a two-antenna system. Such a system was developed by Howells and Applebaum around 1960. Two omnidirectional antennas are used: One is called the primary antenna, the other the reference antenna, in direct analogy to the adaptive-interference-canceling arrangement described before. Let's say we have one desired signal and one undesired signal present, originating from different directions of the compass. The output from each antenna contains both signals, but with different amplitudes and times of arrival. The LMS algorithm may use direction of arrival, strength, energy integrated over time, or one of many other criteria as the basis for interference cancellation. Discussion below is limited to the first case of directional adaptation.

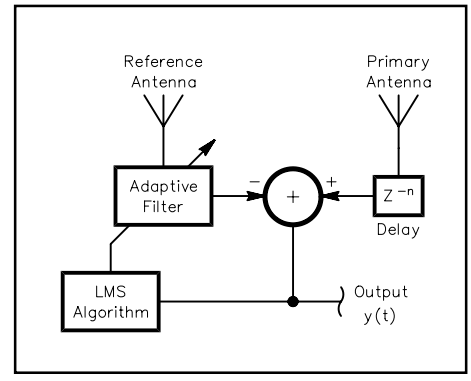

Fig 18.44—Block diagram of a two-antenna, spatial-diversity adaptive system.

Consider what happens when the interference is much stronger than the desired signal: The adaptive filter coefficients are determined almost exclusively by the interference. In the steady state, the adaptive filter's output contains a replica of the interference appearing at the primary antenna, canceling it in the summation. In real systems, each antenna feeds a separate RF receiver where signals are amplified and detected. Receiver and atmospheric noise have a significant effect on performance. The system just described works fine as long as the interference is strong compared with the desired signal. When it is not, leakage terms may bring remarkable improvements to performance, as in the case of adaptive noise reduction above.

Another type of adaptive beamformer developed by Widrow, Griffiths, Mantey and Goode uses a *pilot* signal to steer the array. The pilot signal forces the antenna array to *look* in a specified direction while still retaining the ability to form nulls where interference is present. This has the effect of removing the restriction that the interference be stronger than the desired signal. Others have developed simpler algorithms that accomplish the same thing, but the pilot-signal algorithm remains viable for certain applications. We can dream of many possible configurations for phased arrays. Each must be evaluated for the relationship between angle of arrival (or other criteria) and amplitude/phase of signals appearing at various elements in the array.

PERCEPTUAL SPEECH CODING FOR BW REDUCTION

Driven by the desire to digitally code audio at relatively low bit rates, a lot of effort has been expended on coding algorithms over the years. Much research has been done on the modeling of human speech and

hearing toward that desire. A myriad of schemes have evolved for both modeling and coding—they are too numerous to list here. Let it suffice that these schemes may be divided into two camps: time-domain and frequency-domain. The former type of coding involves the audio waveform directly; the second implies analysis and manipulation of the spectral content of the signal over time.

Time-domain coders use the sample-by-sample characteristics of an analog signal to produce a coded digital signal, which is usually then stored (as on a CD) or transmitted (as through the telephone network). The public switched telephone network (PSTN)—the universal communications medium to which the most people have access—uses a form of audio coding called *m-law* coding. In it, signals are digitized according to a logarithmic (rather than linear) transfer function so as to extend dynamic range. 8-bit quantization is used at a sampling rate of 8 kHz, producing a 64-kbit/s bit stream. Other systems such as *delta modulation* also produce *toll-quality* audio at similar bit rates.

Frequency-domain coders manipulate audio signals by analyzing their spectral content in discrete time intervals. To achieve coding efficiency, these coders exploit the fact that the spectral content of speech signals does not change significantly over short intervals, on the order of 20-30 ms. Spectral signatures may be stored or transmitted using relatively few bits per second. A trade-off may be made between recovered speech quality and transmitted bit rate. To get an idea of what is possible, consider the following example that is a little outside the two classifications above.

It is evident that during normal speech at about 150 words per minute, a textual representation transmitted digitally requires a data rate of only (150 words/min) (5 letters/word) (7 bits/letter) = 5250 bits/minute = 87.5 bits/s. This may be transmitted in a very narrow BW. This system cannot convey any of the characteristics of the speaker's voice, though, such as loudness, timbre, or emotion: It is impossible to tell who the speaker is—unless they tell you! So we must conclude that additional BW is necessary to include such information and to make the recovered speech sound natural.

FFT-based spectral analysis and synthesis may be very effective where it is desired to transmit coded information in the analog domain. The same BW savings may be achieved when transmitting analog signals as when transmitting them digitally, as long as the transmission medium does not introduce deleterious distortions on the coded signals. Media such as HF radio introduce serious multi-path and phase distortion that have proven more damaging to digital modes than analog.

A coded signal may not be the final coder output. It might serve as the input to another algorithm that determines whether speech is present or not, that decides who is talking, or that determines what is being said. Speech-recognition algorithms use many of the same principles as speech-compression algorithms. This technology is being put to use in communications and elsewhere for voice control of equipment, computer dictation, and voice-print verification.

HARDWARE FOR EMBEDDED DSP SYSTEMS

What is it about a microprocessor that makes it a DSP? Well, DSPs are special because they include facilities uniquely designed for the type of calculations common in signal-processing algorithms. They are almost all 16-bit machines, or better, and so are very powerful even without their special facilities. DSPs may be classified primarily by their representation of numbers (fixed-point *vs.* floating-point), also by their data-path width (16-bit, 32-bit), by their programmability (general-purpose *vs.* dedicated co-processor), and their speed.

FIXED-POINT DSPS

Fixed-point DSPs are generally simpler than floating-point units, so they are typically less expensive. Fixed-point processors are common in embedded systems, especially for radio. Special software instructions and separate high-speed computational units are included to accelerate the processing of those common DSP calculations already mentioned. Perhaps the most-used operation is the convolution sum, performed as a series of MAC instructions (see the section on FIR Filters). Designers are interested in how many MACs per second a DSP can execute, because for anything beyond simple audio processing, only a small amount of time is available between samples for filtering and other functions.

A typical 16-bit, fixed-point DSP is shown in the block diagram of **Fig 18.45**. It employs what is called the *Harvard architecture*: It has separate program and data memory paths and also includes a *pipeline* for holding instructions waiting to be executed. This arrangement speeds things along because the CPU can fetch future instructions even when it is executing the current instruction or fetching data from another path.

Consider how this affects an FIR filter algorithm, for example. For each tap in the filter, the processor must multiply a constant (a filter coefficient) by a data value (a stored sample). When the processor can fetch both values simultaneously, an entire cycle time is saved. The subsequent addition of the product to the accumulator and the incrementing of indices for the next MAC instruction may also be executed in a single cycle. When large filters are being implemented, time savings quickly mount. Contrast this with the many cycles needed to perform the same operations in a general-purpose computer and you will see why specialized processors are so much more capable of handling sampled signals.

This business of execution speed is a large factor in the selection of a DSP for any particular use. System planning must begin by reckoning how many instructions can be executed between sample times. In a system with a 30-kHz sampling rate, only 33 µs are available, so a fixed-point DSP that can execute two million MACs per second (2 M-MACs/s) can only get 66 of these in the space between samples. For all but the simplest of systems, this is generally insufficient power for good filtering and other requirements and a separate filter *co-processor* must be employed. This is discussed further below. DSPs are now available having over 200 M-MACs/s performance.

Many fixed-point DSPs are available that also have undedicated parallel and serial input/output (I/O) on board. These may be very useful for embedded applications by obviating the need for other hardware. Processors embedded in radios have traditionally been shut off during times when no user input is present, stopping their clocks. This is done to eliminate the digital-circuit

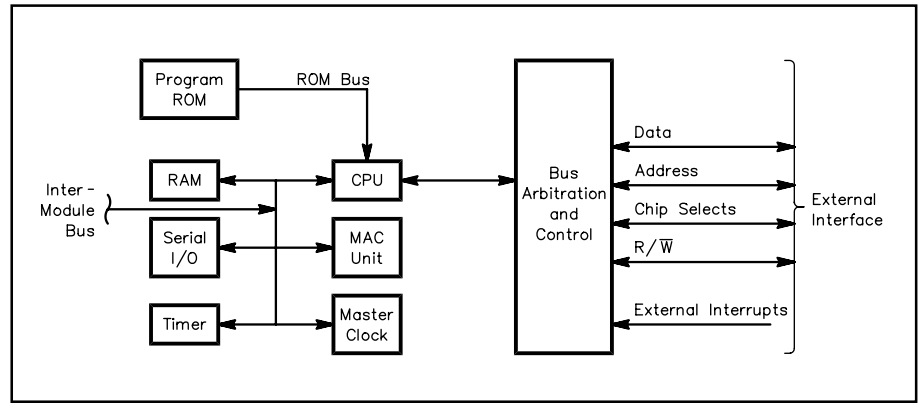

Fig 18.45—Fixed-point DSP block diagram.

noise that otherwise would be difficult to remove. With a DSP in critical signal paths, this luxury is not possible. Careful attention to shielding, grounding and bypassing must therefore be paid. A DSP and associated support components humming along at 25 MHz—or more—tend to generate lots of noise and discrete spectral elements. They also tend to draw significant current, although dissipations in the one-watt range are typical; for base-station equipment, this is not usually a big concern.

Fixed-point math brings with it a limitation on the range of numbers that can be represented, notwithstanding the extended integer/fractional representation demonstrated above. This limitation may form an obstacle to achieving the highest possible dynamic range. For this reason, floating-point DSPs are also widely available for use where greater boundaries must be set on the range of numbers handled. **Table 18.1** shows a listing of popular fixed-point DSPs, along with their floating-point cousins. Manufacturers supply evaluation boards, some of which include data converters and other support circuitry. Control software running on a desk-top computer is available for downloading *object code*—the DSP instructions that make up the program—as well as for debugging by use of tools such as break-points and register dumps.

FLOATING-POINT DSPS

Representation of numbers is a critical decision to be made early in the system design process. A decision to use a floating-point DSP, at generally higher cost than fixed-point, is usually made either to remove dynamic-range barriers or to grant greater flexibility to algorithms that require scaling of data and coefficients, such as the FFT algorithms discussed above. We saw that each floating-point number requires two storage locations: one for the mantissa and one for the exponent. One would expect the processing of these numbers to be slowed by having to handle twice the data, but floating-point architectures are devised in such a way as to minimize or even eliminate this apparent handicap.

Multiplying two floating-point numbers involves multiplying the mantissas, then adding the exponents and any carry (or borrow) from the multiplication. Since multiplications generally require more time than additions, summing the exponents does not really slow the machine very much. Adding two floating-point numbers, though, requires the addition of the mantissas and a possible adjustment to the exponent, and this is always a bit slower than can be done on fixed-point numbers. With an optimized MAC unit, even this restriction can be removed for the bulk of calculations in typical DSP applications. Other than for these points, the block diagram of a floating-point DSP does not look very different from that of the fixed-point unit in Fig 18.45.

SELECTING DATA CONVERTERS

Complete DSP systems almost always include data converters in the form of one or more ADCs and DACs. Selection of these devices for any particular application is made with regard to cost, bit-resolution, speed, SFDR, and digital interface. Manufacturers characterize devices on these bases and obviously, we must choose them so they will handle the highest sampling rate at our analog interface. In general, bit-resolution and speed determine SFDR. Dual 16-bit ADCs and DACs are now very common because they are used in compact-disc (CD) recorders and playback units at a sampling rate of 44.1 kHz. Note

Table 18.1
Fixed and Floating-Point DSPs

Part Number	*Manufacturer*	*# of bits*	*Fixed/Floating*
TMS320Cxx	Texas Instr.	16	Fixed
DSP320Cxx	Microchip	16	"
DSP16	ATT	16	"
ADSP21xx	Analog Dev.	16	"
MC68HC16	Motorola	16	"
MC5600x	Motorola	24	"
MB862xx	Fujitsu	24	Floating
MC9600x	Motorola	32	"
DSP32x	ATT	32	"
TMS320Cxx	Texas Instr.	32	"

that 44.1 kilosamples/s of two channels in a stereo system is equal to $(2) \times (44{,}100) \times (16) \approx 1.41$ megabits per second. Devices with 20 and even 24-bit capability are catching on. This is a lot of data and the bit-resolution of data converters is most often chosen to match that of the DSP, although there may be advantages in having slightly more bit-resolution in the DSP to mitigate round-off errors, as noted in the FIR Filters section above.

We noted before that over-sampling of input signals brings significant advantages for the DSP designer. For this reason, sigma-delta ADCs are the "top of the crop" for use in IF-DSP and DDC receivers. As sampling frequencies increase, over-sampling becomes more difficult to achieve. Engineers working in cellular radio and similar technologies deal with much wider BWs than most of those found in Amateur Radio, and so must grapple with reduced dynamic range; fortunately, they also require less. ADCs that handle 12 to 16 bits at speeds exceeding 65 MHz are available. Viable DDC designs are finding their ways into many commercial services worldwide.

Converters must interface with DSPs through a high-speed digital connection of some kind. Parallel transfer—all 16 bits at once, for example—is more common among DACs than ADCs. High-speed, three-line serial interfaces are popular among converter manufacturers and several standards have evolved. Some of these are compatible with one another. Bearing in mind the amount of data being transferred, realize that these serial links may run at clock speeds in excess of 100 MHz. ADC/DAC evaluation boards may be connected to DSP evaluation boards to form a prototype DSP system. Some data converters are listed in **Table 18.2**.

EXTRA PROCESSING POWER: DSP CO-PROCESSORS

Quite often, a single, general-purpose DSP by itself is not sufficient to handle the computational load in a project. This may be determined early in the system design by evaluating the number of MACs required by filters and other algorithms. Several solutions present themselves: adding one or more general-purpose DSPs, adding specialized co-processor chips, or designing a custom co-processor using programmable-logic chips.

More than one general-purpose DSP may be used to augment net data capacity. The trend these days, though, is to use dedicated co-processor chips that are optimized for the function they are to perform. This is especially true of FFT and other operations that do not lend themselves well to the MAC procedures for which general-purpose chips are optimized. Whatever the algorithm, it seems that multiplication of numbers takes the most time, so a co-processor that incorporates a fast-multiplication algorithm is desirable. A lot of effort has gone into fast multipliers since the 1980s and for the IF-DSP or DDC designer, a knowledge of how it is done may bring plentiful results.

The multiplication of two binary numbers may be decomposed into an addition of several binary numbers. We know that fast binary addition is readily achieved by relatively simple logic. Let's take a look at this, since it forms the basis for most fast multipliers. Shown in **Fig 18.46** is the long multiplication of two 4-bit bi-

Table 18.2
Data Converters

Part Number	*Manufacturer*	*# bits*	*Speed*	*ADC/DAC*
HI1276	Harris/Intersil	8	500 Ms/s	ADC
AD7722	Analog Dev.	16	200 ks/s	"
ADC76	Burr Brown	16	50 ks/s	"
PCM1750	"	18	44 ks/s	dual ADC
CS5322	Crystal	24	2 ks/s	ADC
BT254	Brooktree	24	30 Ms/s	"
Note: Also see Maxim, National, Sipex, Analogic				
CA3338A	Harris/Int.	8	50 Ms/s	DAC
HI1171	"	8	40 Ms/s	"
HI5780	"	10	40 Ms/s	"
HI20201	"	10	160 Ms/s	"
PCM56	Burr Brown	16	93 ks/s	DAC
PCM66	"	16	44 ks/s	dual DAC
Note: See also National, Analog Dev., Maxim, etc.				

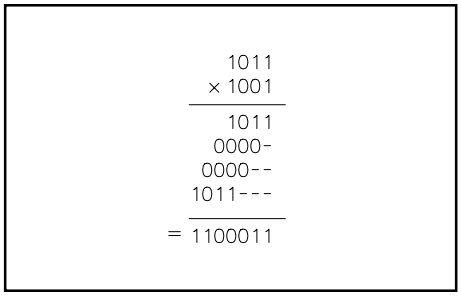

Fig 18.46—Long multiplication of two 4-bit binary numbers.

nary numbers. It is performed in base two the same way as it is in base ten: First, take the least-significant digit of the lower multiplicand and multiply it by the other multiplier. Since in binary, this digit is either one or zero, the digits we write under the line is either a copy of the top multiplicand, or all zeros. Then, the next-significant digit of the lower multiplicand is used, with the result written below the first and shifted one digit to the left. This process continues until all bits of the lower multiplicand have been used. Finally, all the interim results are added. This last result is the product of the two numbers. Note that the result may contain a number of bits as high as the sum of the number of bits in both the multiplicands. Project G in the Appendix shows how simple logic is used to implement a fast multiplier. Pipe-lining and latency issues are discussed there.

Refinements of this technique that use look-up tables and combinatorial methods yield speed increases. Field-programmable gate array (FPGA) manufacturers have worked out the details of these algorithms and routinely provide them to users. FPGAs are available now in very-high-speed versions ($f_{clk} \geq 200$ MHz) that may be used for DSP co-processing. FPGA designs may also employ the Harvard architecture using external, *dual-port memory* to provide a register-based interface to host DSPs. Normally, one sample is passed to the co-processor and one retrieved at each sample time. Filters exceeding 100 taps may be implemented this way, saving processing time in the host DSP for other housekeeping tasks.

Entire down-conversion and I/Q modulation sub-systems have been incorporated on a single chip. These chip sets may be advantageous where FPGA-based designs either do not meet requirements or are too expensive. A sampling of ready-to-use co-processors and DDC chips is given in **Table 18.3**. Also read some of the reference material listed at the end of this chapter for more information on dedicated DSP co-processors.

Table 18.3
Co-processors and DDC Chips

Part Number	*Manufacturer*	*# bits*	*Speed*	*Function*
HSP50016	Harris/Int.	16	52 Ms/s	DSP down-conv.
HSP50110	"	10	60 Ms/s	Quadr. tuner
HSP50210	"	10	52 Ms/s	DSP Costas loop
HSP50306	"	6	2 Mbit/s	QPSK demod
HSP43xxx	"	10-24	var.	DSP filters
510	Harris et al	16	10 Ms/s	Mult/Acc
LMA2010	Logic Dev, IDT	16	40 Ms/s	Mult/Acc
HSP4510x	Harris/Int.	20/32	33 Ms/s	DDS
Various	Xylinx, Altera, Atmel, etc.	8-32	>100 Ms/s	FPGAs

DSP SYSTEM SOFTWARE

ASSEMBLY LANGUAGE AND TIMING REQUIREMENTS

Embedded-DSP application software is most often written in *assembly language*, the native language of the DSP in use. Instructions to be executed are arranged in order, according to the *von Neumann model*, and entered as lines in a text file, using the mnemonics provided by the DSP manufacturer. When this *source code* is ready, an *assembler* program is invoked that translates the source code into object code—the numbers that the DSP understands as instructions. The object code is then transferred to the program memory of the target system for execution.

The reason assembly language is so prevalent in embedded applications is the critical timing involved. Programs compiled in high-level languages do not always handle interrupt-driven events well (the input or output samples) and may bog down. To minimize the required hardware speed, processing of some second-line tasks such as squelch and ALC must have reduced sampling rates to fit into the whole picture. Only a part of their processing burden may be performed at each sample time. This is a form of *time-distributed processing* and is just one in the DSP designer's bag of tricks.

Someone will always think of something more for a transceiver to do and it is better to err on the side of higher speed and more memory at the start than to run out later. Even so, DSP designers must carefully evaluate all the functions included at the outset. Other shortcuts—like the assumption of only integer values by a BFO at one-fourth the sampling frequency—may present themselves, but one cannot always count on it; one must plan diligently to avoid roadblocks. In addition, *unexpected things can occur* if due thought is not given to quantization and scaling effects, especially where adaptive processing is applied, no matter the representation of numbers used. DSP-chip manufacturers provide assemblers and instruction details free of charge. Their applications engineers are ordinarily ready to assist. A plethora of information is available on the Web.

FILTER-DESIGN SOFTWARE

Several software packages for DSP filter design are listed at the end of this chapter. Many more are available. You can expect to find reasonably priced software that will design FIR and IIR filters, as well as let you perform convolution, multiplication, addition, logarithms and other calculations on numeric sequences.

FIR filters usually may be designed with a choice of method (Fourier, Parks-McClellan, least-squares), length, frequency response, and ripple magnitude; they may use various window functions to achieve different shape factors and passband/stopband attenuations. Some are able to take coefficient and data quantization into account and some are not. Large filters may deviate significantly from their theoretical responses because of these effects, so if you are contemplating reasonably long filters, check into this capability.

IIR filter design usually includes a choice of various analog-filter prototypes. Software packages may vary in their ability to display, print, or plot responses and write coefficient files to disk. Filter coefficients are generally part of system firmware and must be transferred from the host DSP to a filter co-processor on demand. It must be possible to translate the filter-design software's output to a format the compiler software understands. A translation program may have to be written to accomplish this.

Longer and more-complex FIR filters may be implemented by convolving the impulse responses of several different filters. This allows the alteration of the frequency response of standard filters to include graphic or parametric equalization and IF shift. Such filtering systems are already being employed in Amateur Radio and commercial transceivers.

OTHER DSP DESIGN TOOLS

FPGA design software is generally available from chip manufacturers. In addition, many schematic-

capture and PCB-layout software vendors provide interfaces to popular FPGAs and other programmable devices. Hardware Design Language (HDL) and Verilog Hardware Design Language (VHDL) have become popular for translating user requirements into programming code for FPGAs. Most FPGA programmers understand HDL or VHDL.

A rich variety of flow-chart software exists in both the public and private domains. It may be especially useful for time-sensitive applications in DSP.

BIBLIOGRAPHY

(Key: **D** = disk included, **A** = disk available, **F** = filter design software)

DSP Software Tools

Alkin, O., *PC-DSP*, Prentice Hall, Englewood Cliffs, NJ, 1990 (**DF**).

Kamas, A. and Lee, E., *Digital Signal Processing Experiments*, Prentice Hall, Englewood Cliffs, NJ, 1989 (**DF**).

Momentum Data Systems, Inc., *QEDesign*, Costa Mesa, CA, 1990 (**DF**).

Stearns, S. D. and David, R. A., *Signal Processing Algorithms in FORTRAN and C*, Prentice Hall, Englewood Cliffs, NJ, 1993 (**DF**).

DSP Textbooks

Frerking, M. E., *Digital Signal Processing in Communication Systems*, Van Nostrand Reinhold, New York, NY, 1994.

Ifeachor, E. and Jervis, B., *Digital Signal Processing: A Practical Approach*, Addison-Wesley, 1993 (**AF**).

Madisetti, V. K. and Williams, D. B., Editors, *The Digital Signal Processing Handbook*, CRC Press, Boca Raton, FL, 1998 (**D**).

Oppenheim, A. V. and Schafer, R. W., *Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975.

Proakis, J. G. and Manolakis, D., *Digital Signal Processing*, Macmillan, New York, NY, 1988.

Proakis, J. G., Rader, C. M., et. al., *Advanced Digital Signal Processing*, Macmillan, New York, NY, 1992.

Rabiner, L. R. and Schafer, R. W., *Digital Processing of Speech Signals*, Prentice Hall, Englewood Cliffs, NJ, 1978.

Widrow, B. and Stearns, S. D., *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1985.

Articles

Albert, J. and Torgrim, W., "Developing Software for DSP," *QEX*, March, 1994, pp 3-6.

Anderson, P. T., "A Simple SSB Receiver Using a Digital Down-Converter," *QEX*, March, 1994, pp 17-23.

Anderson, P. T., "A Faster and Better ADC for the DDC-Based Receiver," *QEX*, Sep/Oct 1998, pp 30-32.

Applebaum, S. P., "Adaptive arrays," *IEEE Transactions Antennas and Propagation*, Vol. PGAP-24, PP. 585-598, September, 1976

Ash, J. et al., "DSP Voice Frequency Compandor for Use in RF Communications," *QEX*, July, 1994, pp 5-10.

Beals, K., "A 10-GHz Remote-Control System for HF Transceivers," *QEX*, Mar/Apr, 1999, pp 9-15.

Bloom, J., "Measuring SINAD Using DSP," *QEX*, June, 1993, pp 9-18.

Bloom, J., "Negative Frequencies and Complex Signals," *QEX*, September, 1994.

Brannon, B., "Basics of Digital Receiver Design," *QEX*, Sep/Oct, 1999, pp 36-44.

Cahn, H., "Direct Digital Synthesis—An Intuitive Introduction," *QST*, August, 1994, pp 30-32.

Cercas, F. A. B., Tomlinson, M. and Albuquerque, A. A., "Designing With Digital Frequency Synthesizers," *Proceedings of RF Expo East*, 1990.

de Carle, B., "A Receiver Spectral Display Using DSP," *QST*, January, 1992, pp 23-29.
Dick, R., "Tune SSB Automatically," *QEX*, Jan/Feb, 1999, pp 9-18.
Emerson, D., "Digital Processing of Weak Signals Buried in Noise," *QEX*, January, 1994, pp 17-25.
Forrer, J., "Programming a DSP Sound Card for Amateur Radio," *QEX*, August, 1994.
Green, R., "The Bedford Receiver: A New Approach," *QEX*, Sep/Oct, 1999, pp 9-23.
Hale, B., "An Introduction to Digital Signal Processing," *QST*, September, 1992, pp 43-51.
Kossor, M., "A Digital Commutating Filter," *QEX*, May/Jun, 1999, pp 3-8.
Morrison, F., "The Magic of Digital Filters," *QEX*, February, 1993, pp 3-8.
Olsen, R., "Digital Signal Processing for the Experimenter," *QST*, November, 1994, pp 22-27.
Reyer, S. and Herschberger, D., "Using the LMS Algorithm for QRM and QRN Reduction," *QEX*, September, 1992, pp 3-8.
Rohde, D., "A Low-Distortion Receiver Front End for Direct-Conversion and DSP Receivers," *QEX*, Mar/Apr, 1999, pp 30-33.
Runge, C., *Z. Math. Physik*, Vol 48, 1903; also Vol 53, 1905.
Smith, D., "Signals, Samples and Stuff: A DSP Tutorial, Parts 1-4," *QEX*, Mar/Apr-Sep/Oct, 1998.
Smith, D., "PTC: Perceptual Transform Coding for Bandwidth Reduction of Speech in the Analog Domain, Part 1," *QEX*, May/Jun, 2000.
Ulbing, Sam, "Surface-Mount Technology—You Can Work With It! Parts 1-4," *QST*, April-July, 1999
Ward, R., "Basic Digital Filters," *QEX*, August, 1993, pp 7-8.

APPENDIX: DSP PROJECTS

PROJECT A: DECIMATION

This project illustrates the concept of decimation using Alkin's *PC-DSP* program, included with the book of that name listed in the Bibliography. First, generate 40 samples of the sinusoid y(n) = sin(n/4), where $0 \le n \le 39$. This sequence may be generated using the "Sine" function of the "Generate" submenu under the "Data" menu, with parameters Var1 = SIN, A = 1, B = 0.25, C = 0 and #Samples = 40. Press F2 to display the data, which should match **Fig 18.A1**.

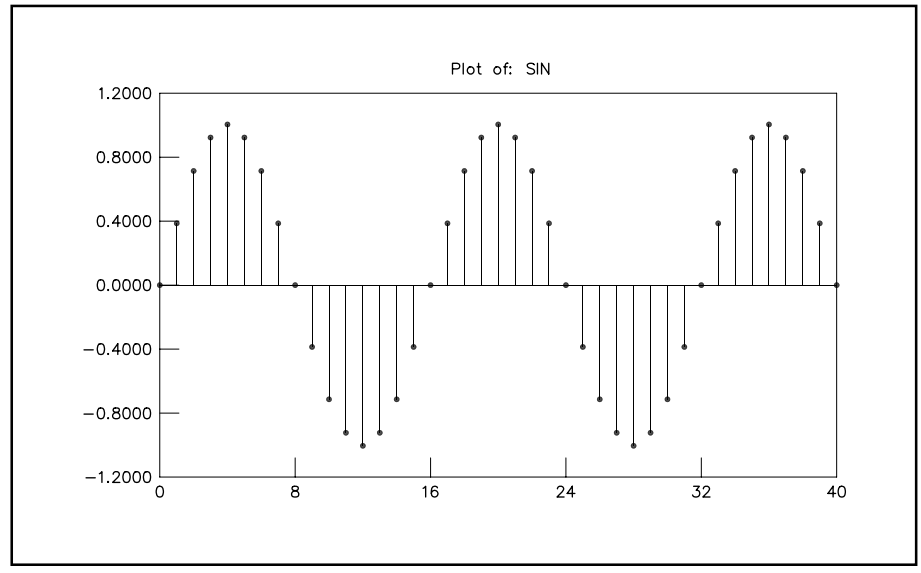

Fig 18.A1—A 40-sample sine wave.

Next, decimate the sequence by a factor of 2 using the "Decimate" function found in the "Process" submenu under the "Data" menu. Use parameters Var1 = SIN2, Var2 = SIN, Factor = 2. Display the new sequence by pressing F2. It should match **Fig 18.A2**.

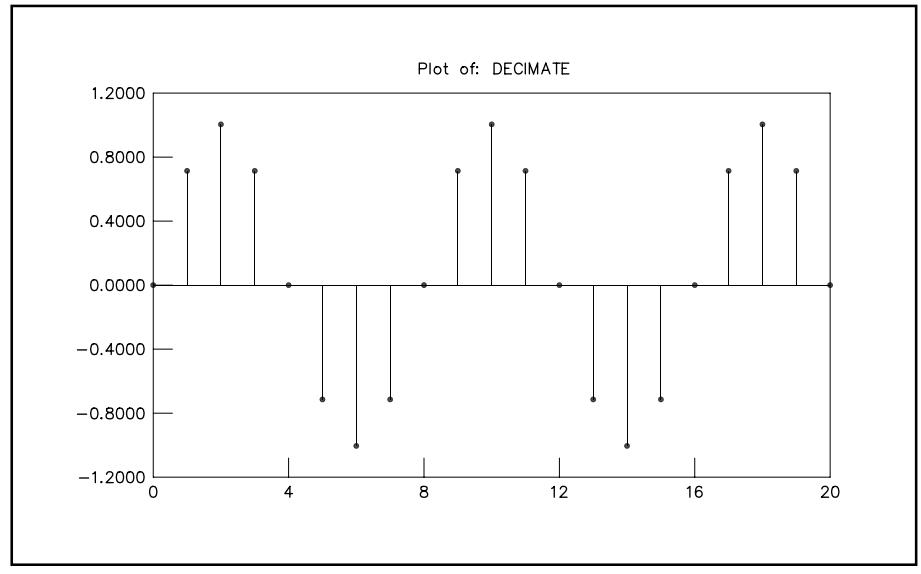

Fig 18.A2—Decimated, 20-sample sine wave.

PROJECT B: FIR FILTER DESIGN VARIATIONS

An FIR filter's ultimate attenuation and its transition BW are largely determined by the filter's length: the number of taps used in its design. Fourier and other design methods do not always readily optimize the trade-off among transition BW, ultimate attenuation and ripple. One way to achieve better ultimate attenuation at the expense of passband ripple is

to convolve the impulse responses of two short filters to obtain a longer filter. The two impulse-response sequences are processed by precisely the same convolution sum that is used to compute FIR filter outputs (Eq 3 in the main text).

A filter obtained by convolving two filters of length L has length 2L –1. In one example, two LPFs of length 31 may be convolved to produce a filter of length 61. The resulting frequency response, plotted against that of a LPF designed with Fourier methods for an identical length of 61 taps, would show that the ultimate attenuation of the convolved filter is 20 dB or 10 times greater than that of the plain, Fourier-designed filter. Also, the convolved filter would have a greater passband ripple and a narrower transition region. Quite often, filters that were designed using different window functions may be convolved to get some of the benefits of each in the final filter.

A look back at Fig 18.29 reveals that different window functions achieve different transition BWs and values of ultimate attenuation. The rectangular window attains a narrow transition BW, but a poor ultimate attenuation; the Blackman window, on the other hand, has nearly optimal ultimate attenuation and a moderate transition BW. Let's see what happens when we convolve the impulse responses of filters designed using each method. We will constrain ourselves to filters with odd numbers of taps so that the convolved impulse response will also have an odd number of taps.

Using your favorite filter-design software, first design a LPF by the Fourier method with a length of 31, using a rectangular window, and a cut-off frequency (–6 dB point) of $0.25f_s$. Its frequency response is shown in **Fig 18.B1A**. We produce a second filter having the same cut-off frequency of $0.25f_s$ using a Blackman window, whose response is shown in **Fig 18.B1B**. The response of the filter formed by the convolution of the two filters is shown in **Fig 18.B1C**, along with that of a standard Fourier-designed LPF. The final filter has length 61 taps. Notice that the filter obtains the benefits of the rectangular window's sharp transition region and those of the Blackman window's good ultimate attenuation.

A second advantage may be garnered by convolving two different filters in that their responses may be governed separately, while producing desired changes in frequency (or phase) response. A good example of this arises when it is desired to alter the audio response of an SSB transmitter (or receiver), but keep the ultimate attenuation characteristics the same. A long BPF with excellent transition properties may be convolved with a much shorter filter that is manipulated to provide the desired passband response.

FIR filters used in Amateur Radio transceivers must usually have at least 60 dB ultimate attenuation. This generally requires at least 63 taps. As our second FIR filter variation, let's consider a case wherein we want to customize an IF-DSP transmitter's frequency response without impacting opposite-sideband rejection. We will use a 99-tap BPF in each leg of a Hilbert transformer (as part of an SSB modulator) whose response is convolved with that of a 31-tap filter describing the variation in frequency response we want. The 99-tap fixed filter has the frequency response shown in **Fig 18.B2A**. The 31-tap filter has been designed using Fourier methods to have a 6 dB/octave rise in its frequency response, as shown in **Fig 18.B2B**.

The frequency response of the convolution of the two filters' impulse responses is shown in **Fig 18.B2C**. It is important to note that the net response is that of the *product* of the two filters' frequency responses; that is, if $H_1(\omega)$ and $H_2(\omega)$ are the two frequency response functions, the final response is simply:

$$H_{composite}(\omega) = H_1(\omega)H_2(\omega) \qquad \text{(B1)}$$

PROJECT C: ANALYTIC FILTER PAIR GENERATION

Frequency-translation properties of complex multiplication work just as well on the responses of filters as they do on real signals. In this project, we will explore just how these properties are applied

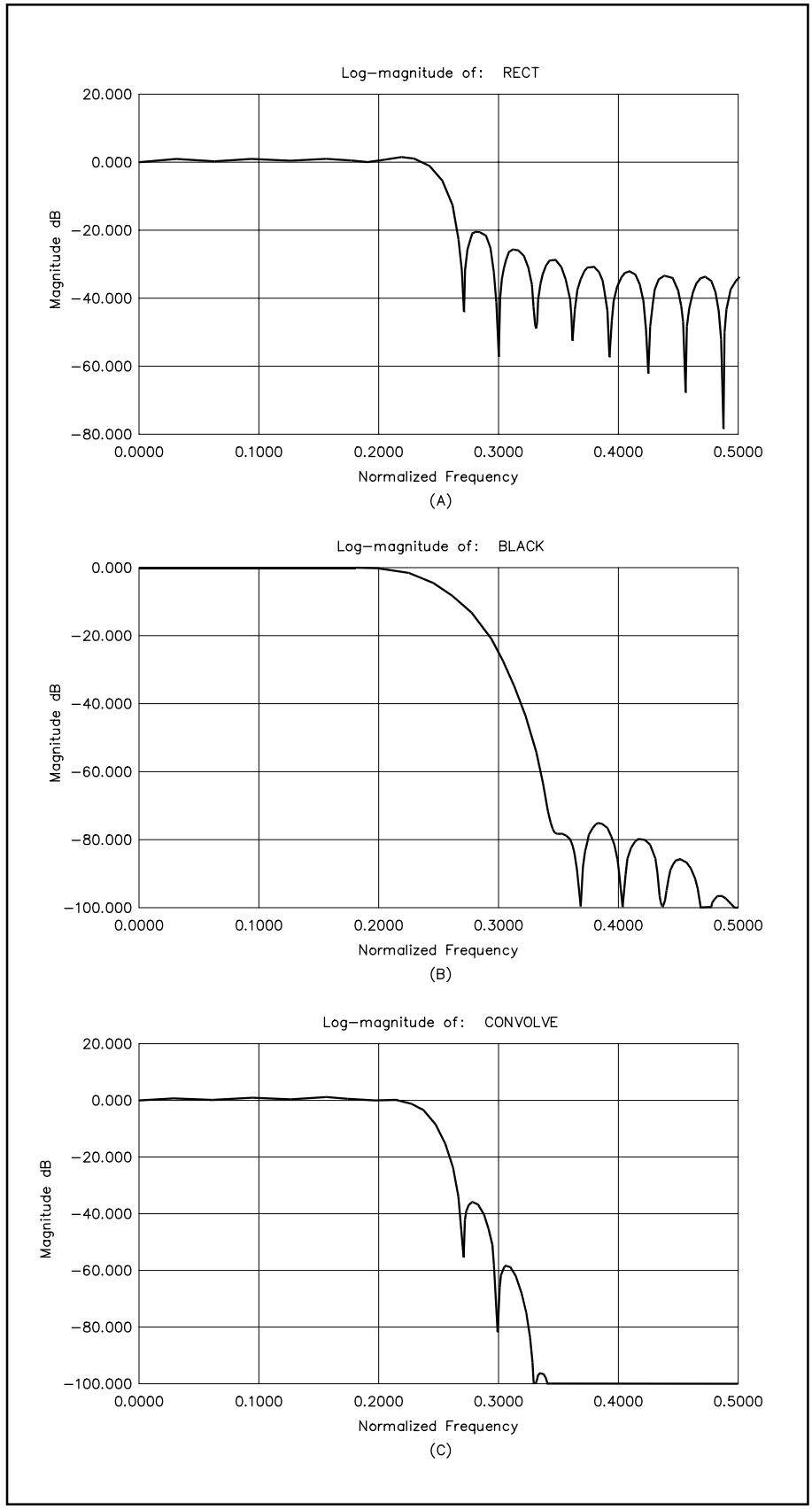

Fig 18.B1—LPF frequency response, rectangular window (A). LPF frequency response, Blackman window (B). LPF frequency response, convolution of filters shown in A and B (C).

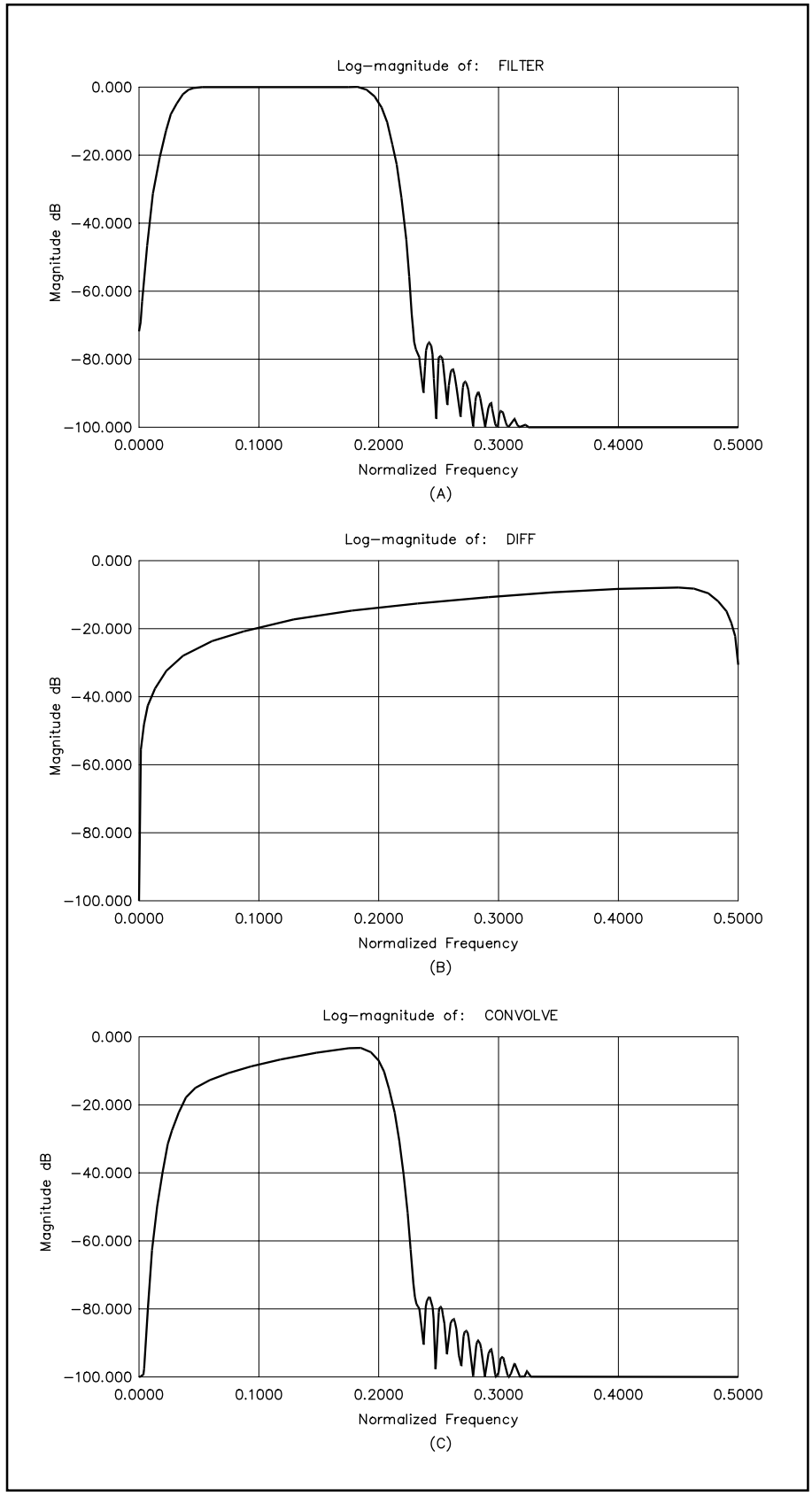

Fig 18.B2—BPF for SSB use, L=99 (A). LPF having rising frequency response, L=31 (B). Frequency response of convolution of filters shown in A and B (C).

to the generation of analytic filter pairs. Analytic filter pairs are used to produce complex signals from real signals for the purposes of modulation, demodulation, and other processing algorithms.

An analytic filter pair consists of two filters (usually BPFs) whose frequency responses are identical, but whose phase responses differ at every frequency by 90·. These filters are used in legs of a Hilbert transformer, as shown in **Fig 18.C1**. The creation of these filters begins with the design of a LPF prototype having the desired passband, transition-band, and stopband characteristics. Such a prototype filter, as might suffice for an SSB receiver, would have a frequency response such as that shown in **Fig 18.C2A** (page 39).

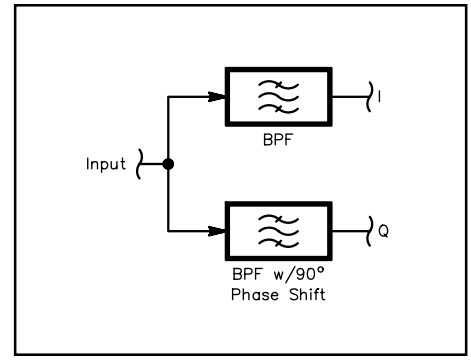

Fig 18.C1—Hilbert transformer using an analytic filter pair.

The filter's impulse response (L = 63) is then multiplied by a sine-wave sequence (also L = 63) whose frequency represents the amount of upward translation applied to the LPF's frequency response. If the sine wave is high enough in frequency, the resulting impulse response is a BPF filter centered on ω_0, the sine wave's frequency. See **Fig 18.C2B**. Likewise, the prototype LPF's impulse response is multiplied by a cosine-wave sequence to produce a filter having the same frequency response as that of the sine-wave filter, but with a phase response differing by 90°. Sample-by-sample multiplication occurs according to Eq 21 in the main text.

When an analytic filter pair is used in a demodulator, IF shift may be included by varying the frequency of ω_0. In combination with various filter BWs, IF shift is useful in avoiding interference by modifying a receiver's frequency response. Further modification may be obtained by convolving each filter in the analytic pair with a filter having the desired characteristic. The phase relation between the filters in the pair will not be altered by the convolution.

PROJECT D: NEWTON'S METHOD FOR SQUARE ROOTS IN QUICKBASIC 4.5

In this example of Newton's method, a generic *BASIC* program is given that computes the root of a 32-bit integer to within an error margin, DERROR. The root of a 32-bit integer is naturally a 16-bit integer. Emphasis is placed in what follows on speed of execution and accuracy as influenced by truncation and rounding. 32-bit integer variables are defined DEFLONG, 16-bit integers are DEFINT. Integer math in *QuickBasic* is much faster than floating-point math.

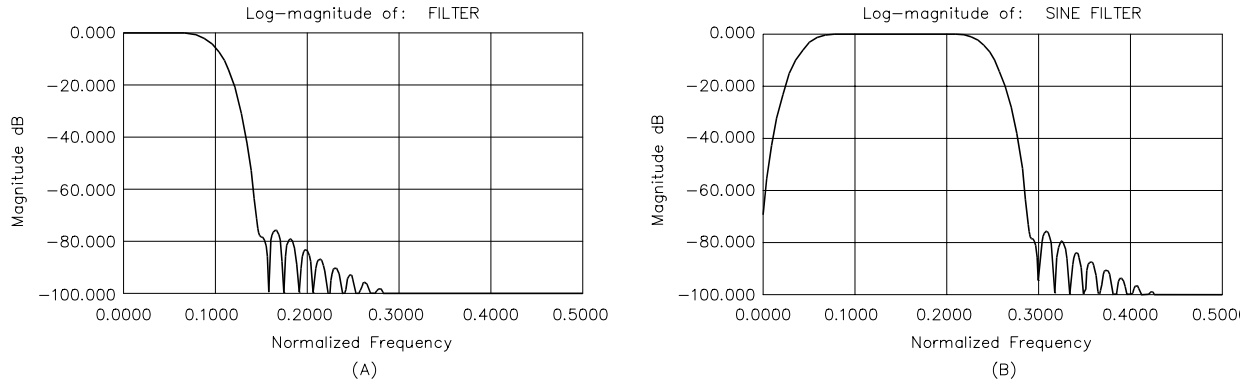

Fig 18.C2—LPF prototype frequency response (A). BPF frequency response of processed impulse response (B).

As described in the AM Demodulation section in the main text, Newton's method iteratively converges on a result. Experience has shown that three to six iterations are necessary to obtain best accuracy for a 16-bit result, but here we execute as many iterations as necessary to obtain accuracy DERROR, initially defined to be one least-significant bit or $1/(2^{15}) \approx 30 \times 10^{-6}$. Note that if DERROR is small or zero, convergence may never be reached because of quantization noise. A loop counter, K, is established to count iterations. The program displays on the computer screen the argument, its root and the iteration count. Users may readily modify the program to use random numbers as arguments to time the number of roots per second it calculates.

The program is included in the *2001 ARRL Handbook* companion software. This software is available for free download from *ARRLWeb* at: **http://www.arrl.org/notes**.

PROJECT E: A FAST SQUARE-ROOT ALGORITHM USING A SMALL LOOK-UP TABLE

This project is a machine-language example of a fast square-root algorithm. The target processor in this case is the Motorola MC68HC16Z1, a 16-bit, fixed-point DSP. The method is depicted in **Fig 18.16** in the main text. Like the previous project, this is included in the *2001 ARRL Handbook* companion software, which is available for free download from *ARRLWeb* at **http://www.arrl.org/notes**.

PROJECT F: A HIGH-PERFORMANCE DDS

A DDS is described below that is used as a reference for a PLL. See **Fig 18.F1**. This DDS is designed to cover a small range of frequencies near 1 MHz. A crystal-oscillator clock at 19.2 MHz is applied to both the DDS, a Harris/Intersil HSP45106, and the DAC, a Harris/Intersil HI5780. Making the DDS output frequency a small fraction of the clock frequency makes it relatively easy to obtain excellent spurious performance. PM spurs are limited to –90 dBc and AM spurs to about –60 dBc. If the output is not squared at the input to a PLL chip, an external Schmitt-trigger squaring stage may be added, eliminating virtually all the AM spurs prior to the LPF.

The LPF at the output of the circuit is a 4-section elliptical type. Design impedance is 100 ohms. This filter cuts out many high-frequency spurs and stops clock feed-through. The DAC's 10 input lines are fed from the 10 most-significant bits of one of the DDS's outputs. The HSP45106 has two 16-bit outputs (sine and cosine) to accommodate the needs of complex-mixer designs, but only one is being used here.

The DDS chip itself is programmed using a 16-bit parallel interface. This is transformed into a serial interface by shift registers U5 and U6, divider U3 and counter U4. Each time the frequency is changed, an internal 32-bit phase-increment accumulator must be updated. The phase increment is just f_{out}/f_{clk}, expressed as a 32-bit, unsigned fraction. This value is written into the chip in two 16-bit segments, most-significant bit of the most-significant word first.

During serial programming, a data bit is placed on the DATA line by the host microprocessor; the clock line is toggled high, then low to shift the bit into the shift registers. After the first 16 bits have been shifted, they are written into the DDS by toggling the ENABLE line. Counter U4 supplies the necessary write pulse with appropriate timing. The remaining 16 bits are then shifted and written to the chip, completing the operation.

An example of the output spectrum of this circuit is shown in **Fig 18.F2**. Components are surface-mount types and care must be exercised during construction. See Ulbing's article in the Bibliography for information on surface-mount soldering techniques.

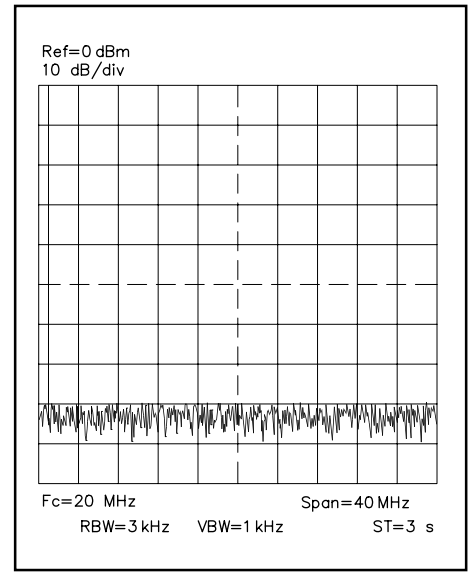

Fig 18.F2—Typical output spectrum of DDS.

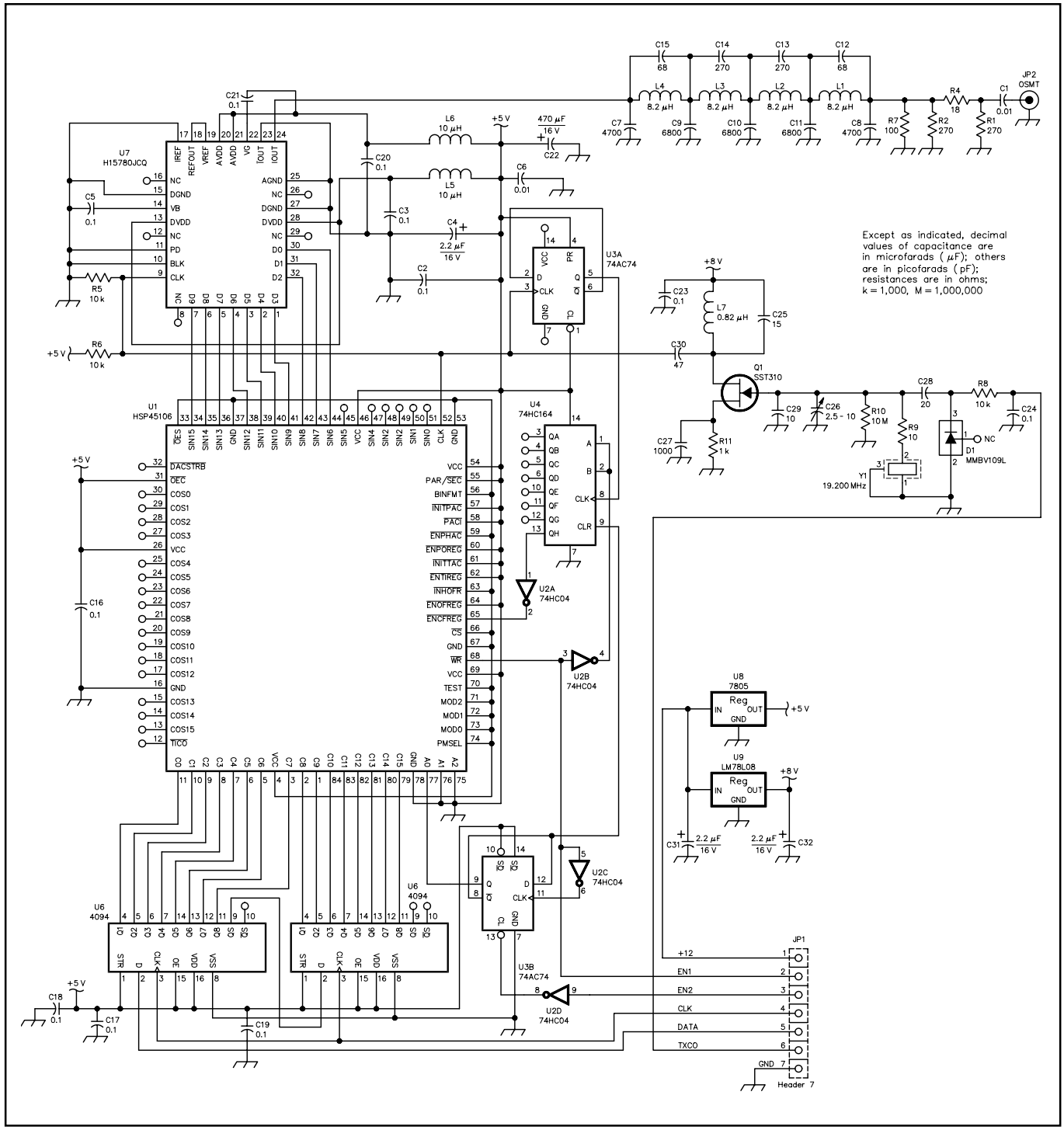

Fig 18.F1—High-performance DDS schematic diagram.

PROJECT G: A FAST BINARY MULTIPLIER IN HIGH-SPEED CMOS LOGIC

In this project, a fast 4-bit binary multiplier is described that may be constructed from 'HC-series logic gates or programmed into an FPGA. Two variations are explored: one without pipelining, and one with pipelining. Pipelining is employed where the propagation delays of gates limit throughput.

As seen in Fig 18.45 in the main text, a 4-bit multiplication may be broken into several 4-bit additions. In our circuit, 4-bit adders are used to add rows of bits in the summation, each one producing a single output bit. The diagram of a fast, 4-bit adder with look-ahead carry is shown in **Fig 18.G1**.

In this multiplier, 4-bit adders are used to add adjacent rows of bits in the traditional way. A multiplier connected this way is shown in **Fig 18.G2**. Not all bits in each addend have mates in the other, so 4-bit adders suffice. In the case where execution speed exceeds the reciprocal of the total propagation delay, pipelining must be employed to avoid error.

To use pipelining, we place storage registers between the stages of addition and one interim result is held by each stage at each clock time. See **Fig 18.G3**. The result is the same, but appears only after a latency of three clock times. When maximum gate delays are well known, this approach also yields more predictable performance because the latency is independent of the input data.

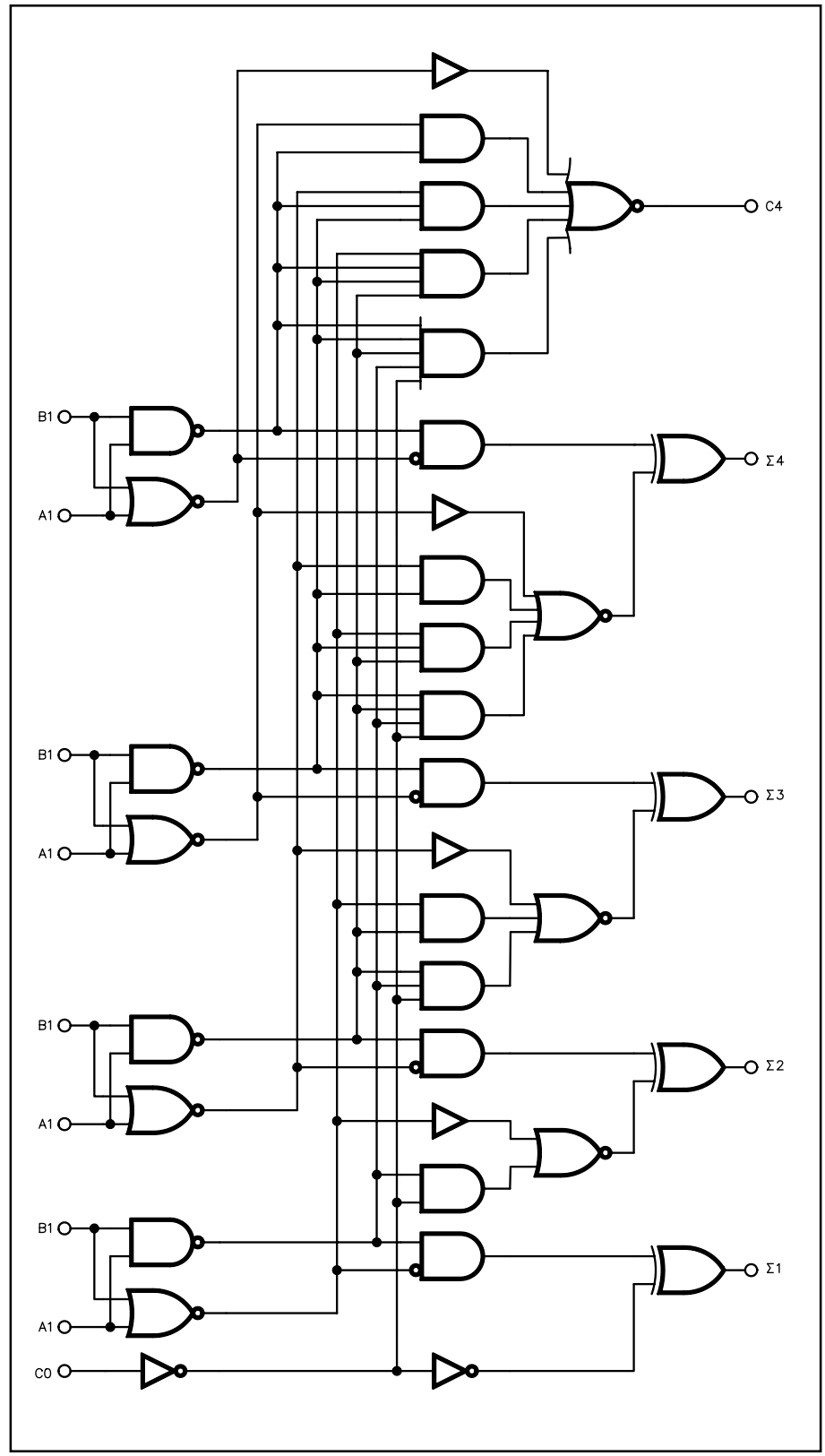

Fig 18.G1—A 4-bit adder schematic diagram.

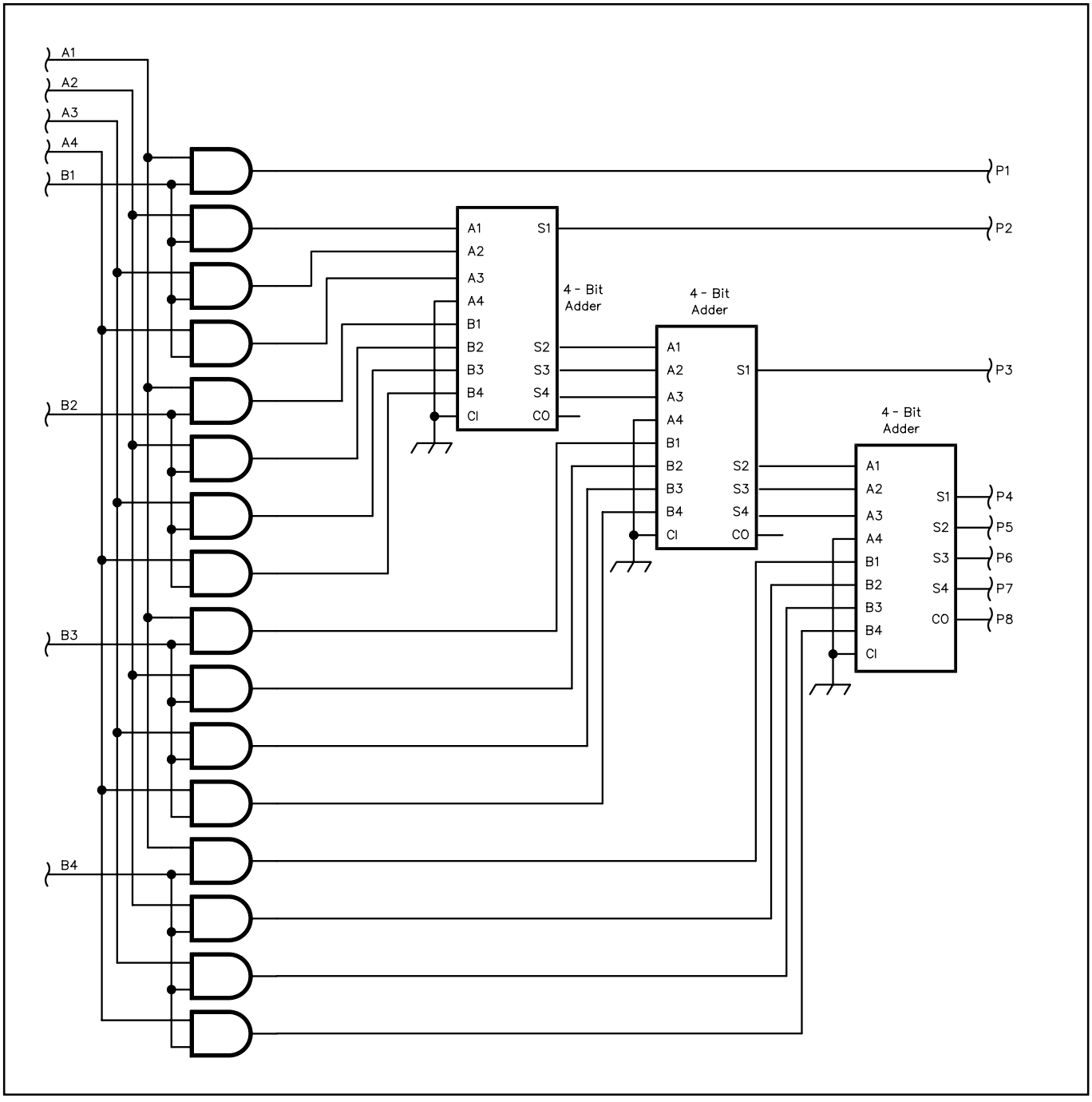

Fig 18.G2—Complete 4-bit multiplier, no pipelining.

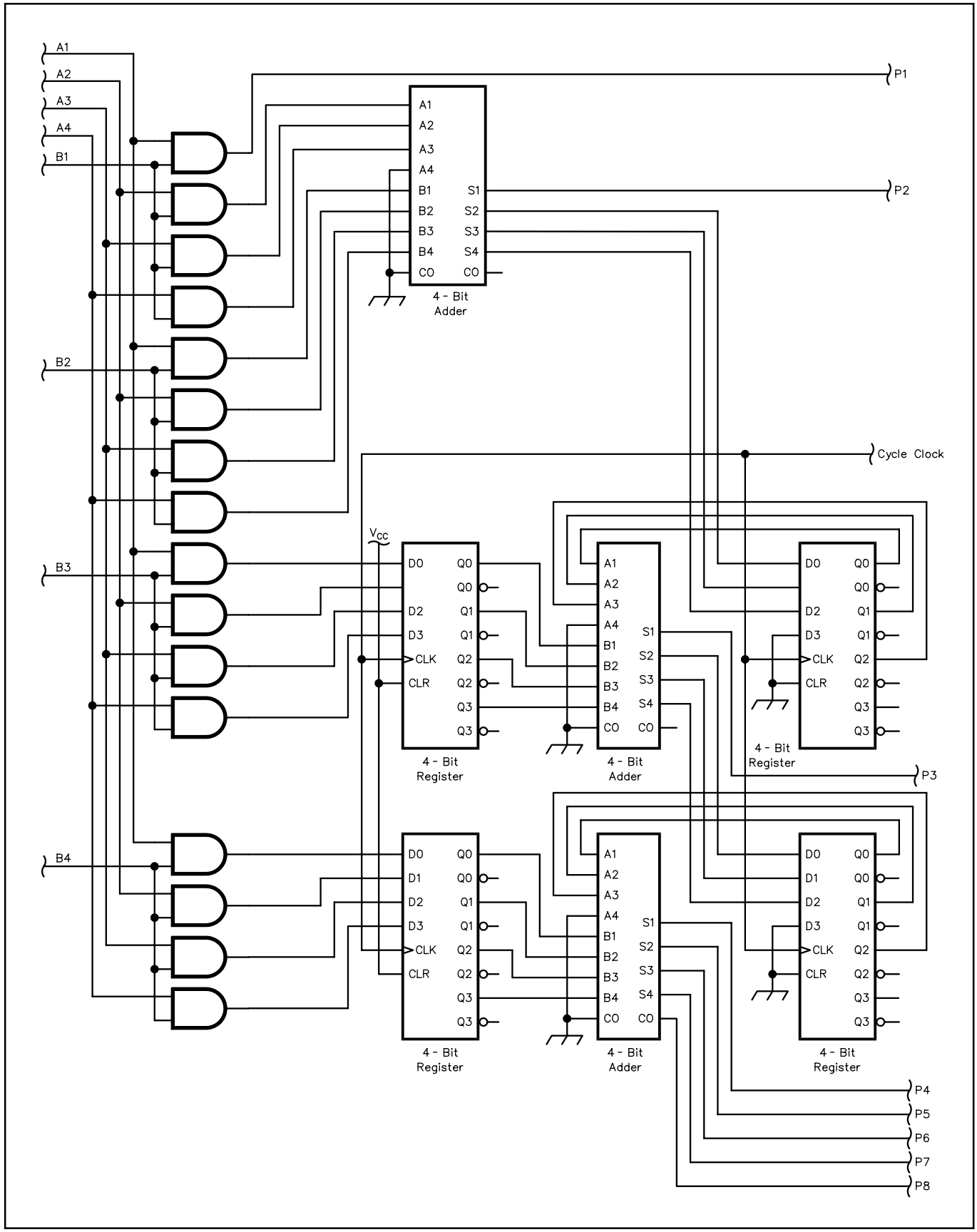

Fig 18.G3—Complete 4-bit multiplier with pipelining.