Chapter 25

Radar Digital Signal Processing

James J. Alter
Jeffrey O. Coleman

Naval Research Laboratory

25.1 INTRODUCTION

The exponential growth in digital technology since the 1980s, along with the corresponding decrease in its cost, has had a profound impact on the way radar systems are designed. More and more functions that historically were implemented in analog hardware are now being performed digitally, resulting in increased performance and flexibility and reduced size and cost. Advances in analog-to-digital converter (ADC) and digital-to-analog converter (DAC) technologies are pushing the border between analog and digital processing closer and closer to the antenna.

For example, Figure 25.1 shows a simplified block diagram of the receiver front end of a typical radar system that would have been designed around 1990. Note that this system incorporated analog pulse compression (PC). It also included several stages of analog downconversion, in order to generate baseband in-phase (I) and quadrature (Q) signals with a small enough bandwidth that the ADCs of the day could sample them. The digitized signals were then fed into digital doppler/MTI and detection processors.

By contrast, Figure 25.2 depicts a typical digital receiver for a radar front end. The RF input usually passes through one or two stages of analog downconversion to generate an Intermediate Frequency (IF) signal that is sampled directly by the ADC. A digital downconverter (DDC) converts the digitized signal samples to complex form at a lower rate for passing through a digital pulse compressor to backend processing. Note that the output of the ADC has a slash through the digital signal line with a letter above. The letter depicts the number of bits in the digitized input signal and represents the maximum possible dynamic range of the ADC. As will be described later, the use of digital signal processing (DSP) can often improve the dynamic range, stability, and overall performance of the system, while reducing size and cost, compared to the analog approach.

This chapter will provide a high-level outline of some of the major digital processing techniques for radar systems that have become practical since the Second Edition of this *Handbook* was published, as well as some design tradeoffs that need to be considered.

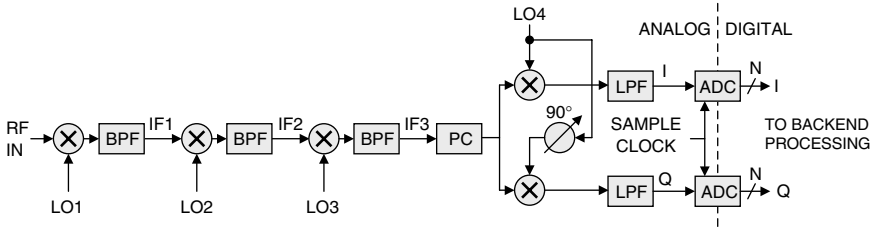

FIGURE 25.1 Typical radar receiver front-end design from 1990

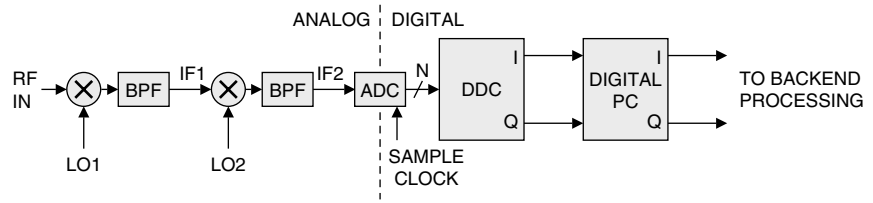

FIGURE 25.2 Typical digital receiver front end

25.2 RECEIVE CHANNEL PROCESSING

Major advances in analog-to-digital converter and digital component technology have transformed the receiver front ends of radar systems, providing higher performance at lower cost. This section will describe how these new technologies are being applied to radar systems and the benefits they bring to system performance.

Signal Sampling Basics. Digital signal processors are sampled signal systems. *Sampling* is the process by which a continuous (analog) signal is measured at regular intervals of time (the *sampling interval*), producing a sequence of discrete numbers (samples) that represents the values of the signal at the sampling instants. The *sampling frequency* is the inverse of the sampling interval and is typically designated f_s. Sampled systems are subject to the Nyquist limit,[1] which lower bounds the sampling rate at which reconstruction of the unsampled signal from its samples is possible without corruption by *aliasing*, the overlapping of spectral components. The bound, termed the *Nyquist frequency* or *Nyquist rate,* is equal to the two-sided signal bandwidth B, the bandwidth considering components at both positive and negative frequencies. Sampling below the Nyquist rate always results in aliasing, but sampling above it does not guarantee alias-free operation. We will see that for bandpass signals a sampling rate higher than Nyquist may be required to avoid aliasing in some situations.

The Nyquist rate is often said to be twice the signal bandwidth, but that refers to a one-sided bandwidth, positive frequencies only, of a real signal. Our definition refers to the two-sided bandwidth, both positive and negative frequencies, of a signal that, in general, is complex with a real signal as a special case.

Is the two-sided bandwidth always twice the one-sided bandwidth? For complex signals in general, no, but for real signals in particular, yes. Here's why: any signal, real or complex, when expressed as a Fourier integral (inverse Fourier transform) is

seen to be a combination of spectral components of the form $A\,e^{j2\pi ft}$. Sampled signals have $t = nT$ with T a sampling interval and n an integer time, but sampled or not, the basic component form is the same. And either way, complex amplitude A is a function of frequency f, but let's write A instead of $A(f)$ for simplicity.

In these terms then, what's special about real signals is that an easily derived Fourier-transform property requires their Fourier components to occur in conjugate pairs, so that if there is a component $A\,e^{j2\pi ft}$ at frequency f with complex amplitude A, there is also a component $A^*\,e^{-j2\pi ft}$ at frequency $-f$ with the complex conjugate A^* of that complex amplitude. If a band of positive frequencies from f_1 to f_2 is occupied by spectral components, the corresponding band of negative frequencies from $-f_2$ to $-f_1$ will be occupied by spectral components also, so the two-sided bandwidth must be twice the one-sided bandwidth.

Real signals have spectral components in conjugate pairs because by using complex amplitude expressed in polar form as $A = re^{j\theta}$,

$$A\,e^{j2\pi ft} + A^*\,e^{-j2\pi ft} = 2\,\mathrm{Re}\{Ae^{j2\pi ft}\} = 2\,\mathrm{Re}\{r\,e^{j\theta}e^{j2\pi ft}\}$$
$$= 2r\,\mathrm{Re}\{e^{j(2\pi ft+\theta)}\} = 2r\cos(2\pi ft+\theta)$$

The imaginary parts of the conjugate spectral components have canceled to reveal that those components together indeed represent a real signal, a sinusoid with amplitude and phase specified by the magnitude and angle of the complex amplitude. The latter relationship is so much a part of the engineering culture that the terms *amplitude* and *phase* are commonly, if imprecisely, used to refer to the magnitude and angle of a complex signal at an instant in time.

The following figures illustrate the origin of the Nyquist rate. Imagine that a real signal with a *lowpass* signal spectrum of two-sided bandwidth B is plotted on a long piece of paper, as shown in Figure 25.3*a*. In the figure, the positive-frequency spectral components of the signal are darkly shaded, and the negative-frequency components are lightly shaded. To see the effect of sampling this signal at Nyquist rate B, the long sheet is cut into smaller sheets, with the first cut at zero frequency and subsequent cuts at sample-rate (B, in this case) intervals in positive and negative frequency. The sheets are stacked one on top of the other as shown on the left side of Figure 25.3*b*, and the resulting portion of the sampled signal spectrum from 0 to the sampling rate of B is generated by adding the spectra of the stacked pages together, as shown on the right.

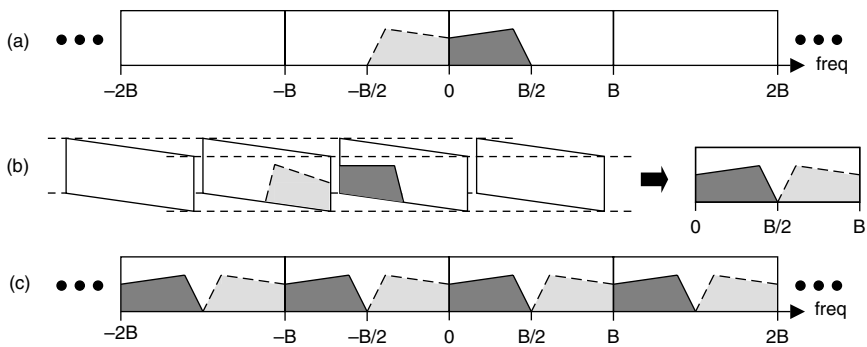

FIGURE 25.3 (*a*) Bandlimited, real signal spectrum before sampling, (*b*) portion of sampled spectrum from 0 to B, and (*c*) full sampled signal spectrum

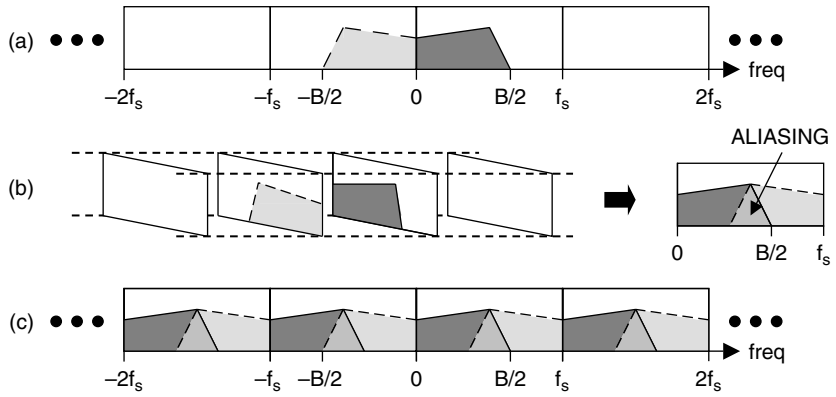

FIGURE 25.4 (*a*) Bandlimited lowpass signal spectrum before sampling, (*b*) aliased lowpass signal spectrum after sampling at rate $f_s < B$, and (*c*) aliased sampled signal spectrum

Note that the lightly shaded negative-frequency portion of the spectrum now appears on the right of the sampled spectrum and doesn't overlap the darker positive-frequency portion. As long as the two portions of the sampled signal don't overlap, the signal is not aliased. The full sampled-signal spectrum is obtained by laying copies of this page end-to-end, as shown in Figure 25.3*c*, producing copies of the 0 to *B* portion of the sampled signal spectrum at *B* intervals.

Figure 25.4 shows the result of sampling below the Nyquist rate. Figure 25.4*a* shows the same bandlimited signal as the previous example, but this time it is sampled at some rate that is less than Nyquist rate *B*. The resulting sampled spectrum, shown in Figure 25.4*b* and *c*, contains overlapped, or aliased, spectral components that add and represent corruption of the signal.

Figure 25.5 repeats this Nyquist analysis for a *bandpass signal*—a signal not containing spectral components at or near 0 Hz. Figure 25.5*a* shows a real bandpass signal with two-sided bandwidth *B* and composed of positive-frequency and negative-frequency spectral components, each of bandwidth *B*/2, that are complex-conjugated mirror images. The Nyquist rate is the signal's two-sided bandwidth irrespective of the particular portion of the spectrum in which the signal resides. Therefore, for this signal the Nyquist frequency is *B* even though the signal contains components at actual frequencies greater than *B*. Figure 25.5*b* shows the result of sampling this signal at

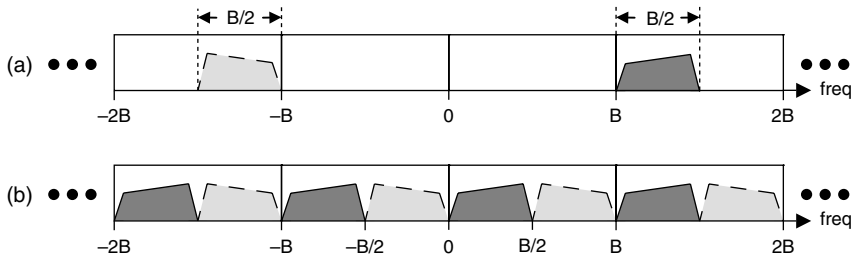

FIGURE 25.5 (*a*) Bandlimited, real passband signal spectrum before sampling and (*b*) signal spectrum after sampling

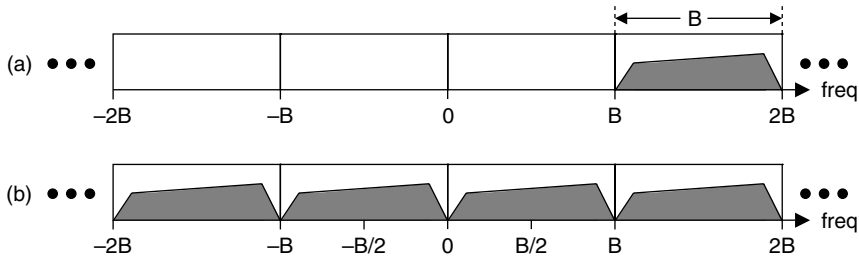

FIGURE 25.6 (*a*) Non-real signal spectrum before sampling by Nyquist frequency, *B*, and (*b*) signal spectrum after sampling

the Nyquist bound. The sampled spectra of the two portions of the signal do not overlap; the sampled signal is not aliased. As will be described in more detail later in the chapter, this technique, *bandpass sampling,* is a powerful tool that allows a relatively high-frequency signal to be sampled by a relatively low-performance digitizer, which can result in considerable cost savings.

Figure 25.6*a* shows the spectrum of a more general complex signal of bandwidth *B* before sampling. Note that this signal does not possess complex-conjugate spectral symmetry. The signal spectrum after sampling by its Nyquist frequency *B* is shown in Figure 25.6*b*. There is no aliasing.

The Nyquist rate is a *minimum* sampling frequency for a signal, a bound, and meeting the bound is necessary, but not sufficient, to ensure that no aliasing occurs. Consider the case presented in Figure 25.7*a*, which is the same bandlimited bandpass signal shown in Figure 25.5, but shifted in frequency so that it doesn't begin exactly at *B*. The sampled signal spectrum in Figure 25.7*b* shows that, although the sampling rate satisfies the Nyquist bound, the sampled signal is still aliased. To solve this problem the signal could be moved to a different center frequency before sampling or the sampling rate could be increased. The system designer must always develop the frequency plan of a sampling system carefully to determine an appropriate sampling frequency and to ensure that aliasing does not occur. A full treatment of this subject is presented by Lyons.[2]

In an actual system, before sampling the signal is typically passed through an *anti-aliasing filter*, which is an analog lowpass or bandpass filter that places an upper limit on the signal bandwidth. The filter needs to provide enough stopband rejection that any aliased components are insignificant. Of course, practical filters do not have

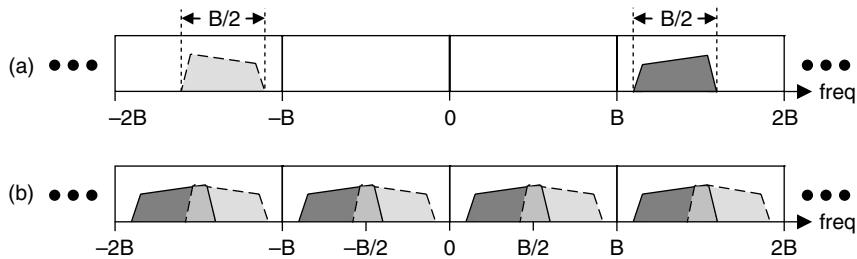

FIGURE 25.7 (*a*) Bandlimited, real passband signal spectrum before sampling and (*b*) signal spectrum after sampling

passbands extending right up to their stopband edges, so the widths of intervening transition bands must be counted as part of two-sided signal bandwidth B for purposes of determining the Nyquist rate, as the filter output may contain components in these transitions bands that could otherwise result in significant aliasing.

Digital Downconversion (DDC). The application of digital technology to IQ demodulation, which is just downconversion of an IF signal to a complex baseband, has greatly improved the performance of coherent systems. Here, we explore two forms of such digital downconversion, a general form that is structurally parallel to traditional analog downconversion and a restricted form, direct digital downconversion, which is more economical when it is applicable.

Analog Downconversion and Sampling. The general approach to digital downconversion derives from analog downconversion and sampling, as illustrated in the frequency domain in Figure 25.8. The spectra on the first and = lines represent signals at various points in the system, and the spectra on the $*$ and $\times$ lines, respectively, represent *spectral-convolution* and point-by-point *spectral multiplication* operations that relate those signals.

The first line in the figure depicts schematically a real IF signal with one- and two-sided bandwidths of 40 MHz and 80 MHz, respectively, and with positive- and negative-frequency components, respectively centered at 75 MHz and −75 MHz. The second line of Figure 25.8 shifts the IF signal by LO frequency −75 MHz using spectral convolution. (We'll see shortly how this is done in hardware.) The result, on the third line, has spectral components centered at 0 MHz and −150 MHz. Multiplication by the lowpass-filter frequency response in line 4 then removes the latter component, leaving only the complex baseband signal of line 5, which has a two-sided bandwidth and a Nyquist frequency of 40 MHz. The spectral convolution on line 6 corresponds

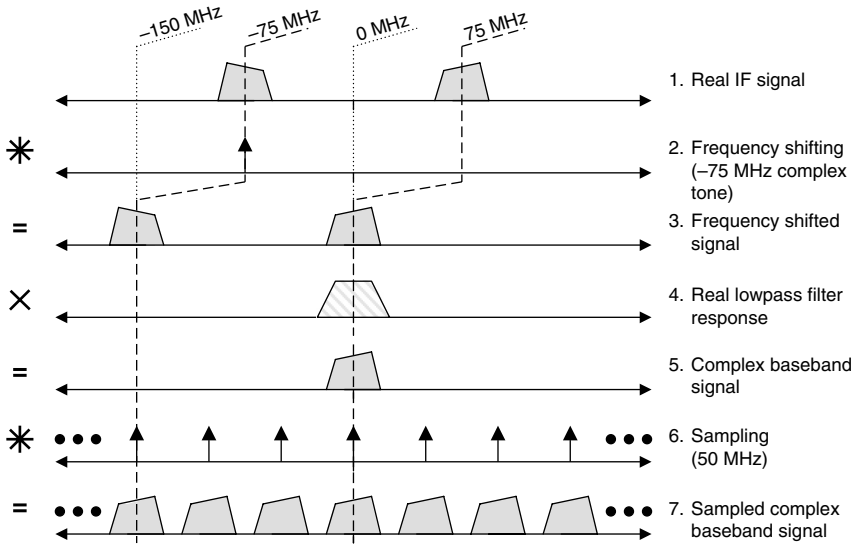

FIGURE 25.8 Analog downconversion in the frequency domain

to time-domain multiplication of a uniform impulse train at the 50 MHz sampling frequency with the signal represented by line 5. In the time domain, the result is a 50 MHz train of sampling impulses with areas that are (give or take a scale factor that we are ignoring) samples of the line 5 signal at the sampling instants. Of course, we will not create the line 7 impulses in hardware but will instead realize the impulse areas digitally as numbers in registers.

A block diagram showing how this process might be implemented in hardware is shown in Figure 25.9. The IF signal is sent to two mixers. In one mixer, the IF signal is beat with the 75 MHz LO with cosine phasing, and in the other mixer, the IF is beat with the same LO but with negative sine phasing, so that the mixers are operated in quadrature, 90° apart. The mixer outputs taken together as a complex pair form a complex signal with the spectrum shown in line 3 of the previous figure. These signals are then passed through lowpass filters (LPF) to remove the spectral component centered at −150 MHz that would otherwise have resulted in aliasing in the sampling step to follow.

Labels I (in-phase) and Q (quadrature) are traditionally used to indicate the real and imaginary parts of complex time-domain signals, like those here, that are realized as pairs of real signals. When a vertical cut through a diagram, such as in Figure 25.9, picks up one I signal and one Q signal, the represented complex signal crossing that cut is I + jQ. In the diagram, cuts just before and after the LPF blocks pick up complex signals with the spectra shown on lines 3 and 5 of Figure 25.8, respectively. The line 3 signal is created in the time domain as

$$\begin{aligned}[\text{line 3}] &= [\text{line 1}]\, e^{-j2\pi f_{\text{LO}}t} \\ &= [\text{line 1}]\cos(2\pi f_{\text{LO}}t) \\ &\quad -j[\text{line 1}]\sin(2\pi f_{\text{LO}}t) \\ &= [I_3 \\ &\quad + jQ_3]\end{aligned}$$

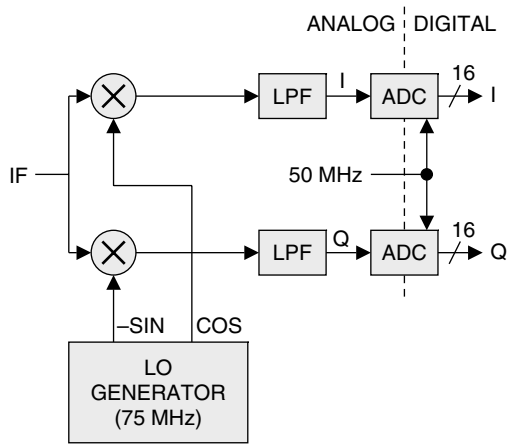

FIGURE 25.9 Typical analog downconversion to baseband and digitizer

where f_{LO} = 75 MHz. Similarly, using $*$ for time-domain convolution (filtering with an impulse response),

$$\begin{aligned}[\text{line 5}] &= [\text{line 3}] * h(t) \\ &= [\mathrm{I}_3 \\ &\quad + j\mathrm{Q}_3] * h(t) \\ &= (\mathrm{I}_3 * h(t)) \\ &\quad + j(\mathrm{Q}_3 * h(t)) \\ &= \mathrm{I}_5 \\ &\quad + j\mathrm{Q}_5\end{aligned}$$

When the filter outputs are viewed as complex signal [line 5] = $\mathrm{I}_5 + j\mathrm{Q}_5 = Ae^{j\theta}$, the complex magnitude A and angle θ give the amplitude (give or take a scale factor) and phase of the IF signal, because the original IF signal could be re-created from the line 5 signal in the time domain (again give or take a scale factor) as [line 1] = Re{[line 5] $e^{j2\pi f_{LO}t}$}, from which it follows that, [line 1] = Re{A $e^{j\theta}$ $e^{j2\pi f_{LO}t}$} = A Re{$e^{j(2\pi f_{LO}t + \theta)}$} = A cos($2\pi f_{LO}t + \theta$).

As a final step the baseband I and Q signals after the filters are digitized by ADCs at a 50 MHz sampling rate, producing I_7 and Q_7 output samples or, equivalently, complex output samples $\mathrm{I}_7 + j\mathrm{Q}_7$.

The slash through the output of the ADC with a "16" above it in Figure 25.9 indicates that our ADC produces 16 bits of digital output. ADCs provide approximately 6 dB of dynamic range per bit, so our 16-bit ADC provides about 96 dB of dynamic range, assuming ADC nonlinearities are negligible.

A General Approach to Digital Downconversion. In digital downconversion, the analog IF signal is first sampled by an ADC, and all of the subsequent processing is then done digitally. Figure 25.10 depicts the digital downconversion process for our previous example, again in the frequency domain. The top line schematically represents the real IF signal with parameters as before. Performing the sampling analysis described previously, we discover that setting the sample rate to the two-sided signal bandwidth of 80 MHz would produce aliasing. However, a 100 MHz sample rate does not cause aliasing and is used on the second line of the figure. Sampling the input signal at 100 MHz replicates the signal spectrum at 100 MHz intervals as shown on line 3. Frequency shifting is accomplished by spectrally convolving this signal with the complex –75 MHz LO tone shown in line 4, producing the frequency-shifted signal on line 5. The latter signal is spectrally multiplied by the filter response shown on line 6 to remove the copies of the negative-frequency signal component, producing the complex baseband signal shown on line 7. This signal, which now has a two-sided bandwidth and Nyquist frequency of 40 MHz, is spectrally convolved in line 8 with impulses at the spectral origin and at 50 MHz to effectively decimate the signal by a factor of two.[3] The final baseband signal on line 9 has a sample rate of 50 MHz.

Figure 25.11 depicts the hardware implementation of this DDC architecture. The IF signal centered at 75 MHz is digitized directly by an ADC. After the ADC, the architecture is very similar to analog downconversion, except that the processing is performed digitally. In our example, we elect to sample the IF signal at a rate of 100 MHz with a 16-bit ADC. This architecture realizes the LO with a numerically controlled oscillator (NCO) that generates digital words to represent the cosine and negative sine signals at the LO frequency, here 75 MHz and sampled at the ADC

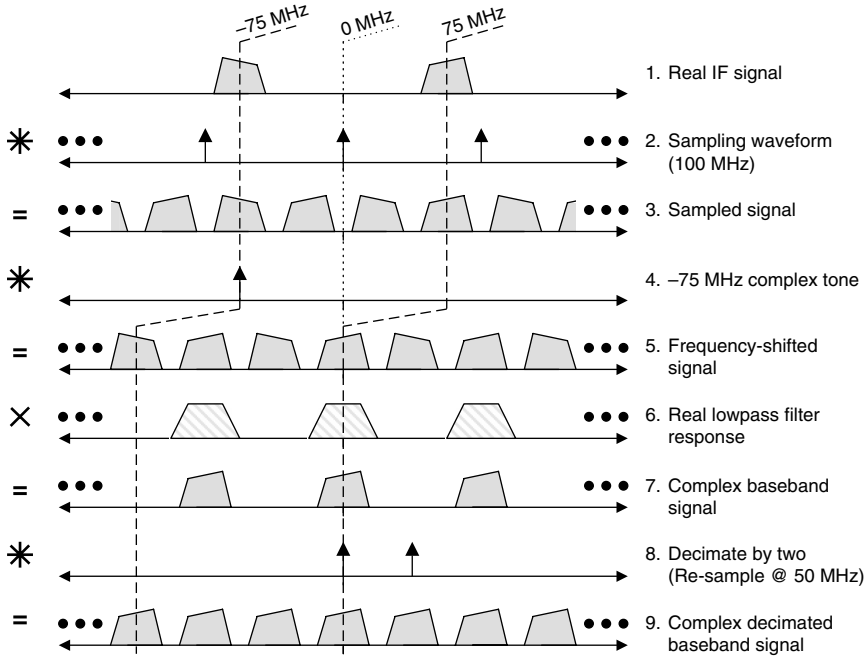

FIGURE 25.10 Digital downconversion in the frequency domain

sample rate. The sine and cosine signals from the NCO are then digitally multiplied by the digitized IF signal. In this particular example, the relationship between the LO frequency and the sampling rate will make the required NCO and the multipliers both rather trivial because each required NCO output value is zero or ±1, and that special case will be addressed shortly. For now, this architecture supposes that no such special situation exists and that a general NCO/multiplier structure is needed. The design of

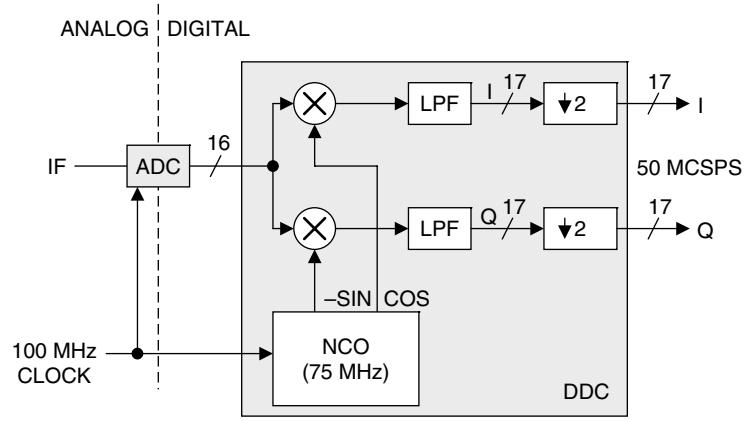

FIGURE 25.11 Digital downconversion architecture

a general NCO will be in Section 25.3. Following the multiplications, digital lowpass filters prevent aliasing when their outputs are decimated by a factor of two to produce complex output samples at a 50 MHz rate. In the figure, MCSPS stands for *million complex samples per second.*

The lowpass filter also reduces out-of-band noise and thus increases signal-to-noise ratio (SNR). In order to preserve this SNR growth, the number of bits used to represent the filter output might need to increase. If the filter reduces the bandwidth of the data by a factor R without affecting the signal of interest, then the SNR increase in dB is given by $10\log_{10}R$. In our example, a 2× reduction in bandwidth results in approximately a 3 dB increase in SNR. With each bit representing about 6 dB of SNR, the minimum number of bits required to represent the filtered signal could grow from 16 to 17.

In an actual application, the system designer needs to analyze the effects of sampling and digital processing and determine how many bits need to be carried through the calculations in order to preserve SNR and prevent overflow. Considerations include front-end system noise, which is typically allowed to toggle two or more bits (four or more quantization levels) of the ADC output. Also, an actual N-bit ADC never provides exactly 6N dB of SNR, due to ADC-induced errors. For example, a typical 16-bit ADC generally provides about 14 bits or about 84 dB of SNR. Carrying 16 bits through the signal processing provides about 96 dB of dynamic range. In this case, the designer may elect to allow the datapath through the lowpass filter to remain at 16 bits, realizing that the filtering process has simply increased the SNR of the signal to 87 dB, which could still be accommodated by the 16-bit datapath.

A DDC provides several benefits compared to analog downconversion. The analog approach is subject to various hardware errors, including mismatch of the mixers, LO signals not exactly 90° apart, and mismatches in the gains, DC offsets, or frequency responses of the I and Q signal paths. A DDC avoids these problems, though it is vulnerable to the phase noise of the ADC sample clock, ADC nonlinearities, and arithmetic round-off noise. Realizing maximum performance requires careful attention to design details.

Direct Digital Downconversion. If the designer has some flexibility in either the IF center frequency or ADC sample rate, a simplified DDC architecture, direct digital downconversion, can be considered.[4,5] If the ADC sample rate is four times the center of the IF band, then the sampling process can also shift the spectrum to baseband, and the NCO and associated multipliers of the general DDC are not needed. In general, direct conversion to baseband is a simple and cost-effective DDC method that can be used when the signal being sampled is always centered at a single frequency. The standard DDC architecture might need to be used when the center frequency of the signal being sampled dynamically changes, which forces the DDC's LO to change accordingly.

Let's look at the direct DDC in the time domain first, for intuition, and then we can carefully derive the architecture in the frequency domain. Suppose the DDC architecture is as sketched in Figure 25.11, with an IF centered at 75 MHz and a 75 MHz LO and suppose the NCO is set to 300 MHz so that it produces the sampled sines and cosines shown in Figure 25.12*a*, where vertical lines and dots indicate sample times and values, respectively. Because the sample rate is four times the LO frequency, the (cos, -sin) LO sample pairs cycle repeatedly through (1, 0), (0,–1), (–1, 0), and (0, 1).

Next, suppose the IF signal is a 75 MHz sinusoid of arbitrary phase as in line (b). The DDC's mixer outputs I and Q, the products of the line (b) IF signal with the two

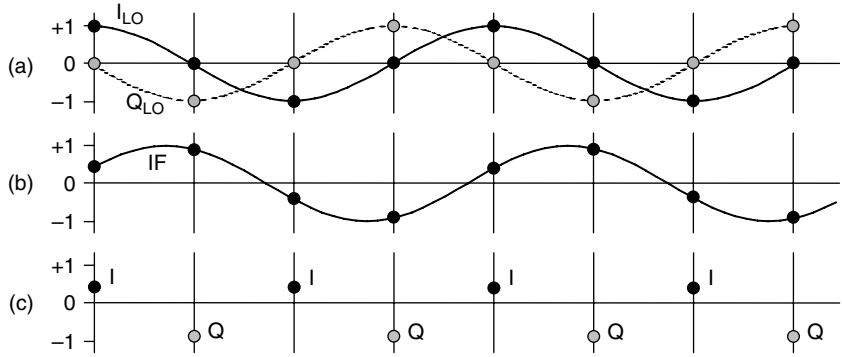

FIGURE 25.12 Various signals sampled at 300 MHz: (*a*) 75 MHz cosine and –sine LO signals, (*b*) 75 MHz IF tone, and (c) result of multiplying (*a*) samples by (*b*) samples

line (a) LO signals, are then as in line (c). Because our hypothetical IF signal on line (b) was exactly at one quarter of the sample rate, both I and Q are constants, the sine and cosine of the IF signal's phase angle.

Figure 25.13 shows the same 75 MHz IF tone, but sampled at 100 MHz and 60 MHz, which are odd integer submultiples (1/3 and 1/5) of the original sample rate of 4× the IF center frequency, 300 MHz. Note that odd samples still cycle between I and –I, and the even samples switch between Q and –Q. Odd integer submultiples of 4× the IF center frequency can, therefore, be viable alternative sample rates. A Nyquist bound applies and requires the two-sided IF bandwidth to be less than the sampling rate.

Now let's derive the direct-DDC architecture carefully in the frequency domain. Suppose a real IF signal is once again centered at 75 MHz and sampled at 100 MHz as in line (a) of Figure 25.13. The first three lines of Figure 25.14 illustrate this in the frequency domain with line 3 showing the sampled IF signal. The bandpass filter response on line 4 removes the unwanted spectral components to produce the complex passband signal of line 5. This signal is then decimated by 2 and shifted by –75 MHz to produce, at a 50 MHz sampling rate, the desired complex baseband signal shown on line 9.

Figure 25.15 shows the corresponding block diagram. The magnitude of the frequency response on line 4 of Figure 25.14 is neither an even nor an odd function, so the corresponding impulse response is neither purely real nor purely imaginary.

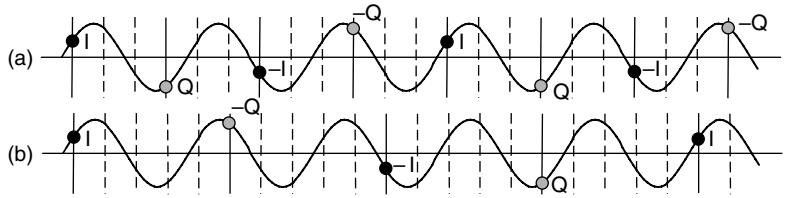

FIGURE 25.13 75 MHz tone sampled at (*a*) 100 MHz (4/3 × IF) and (*b*) 60 MHz (4/5 × IF)

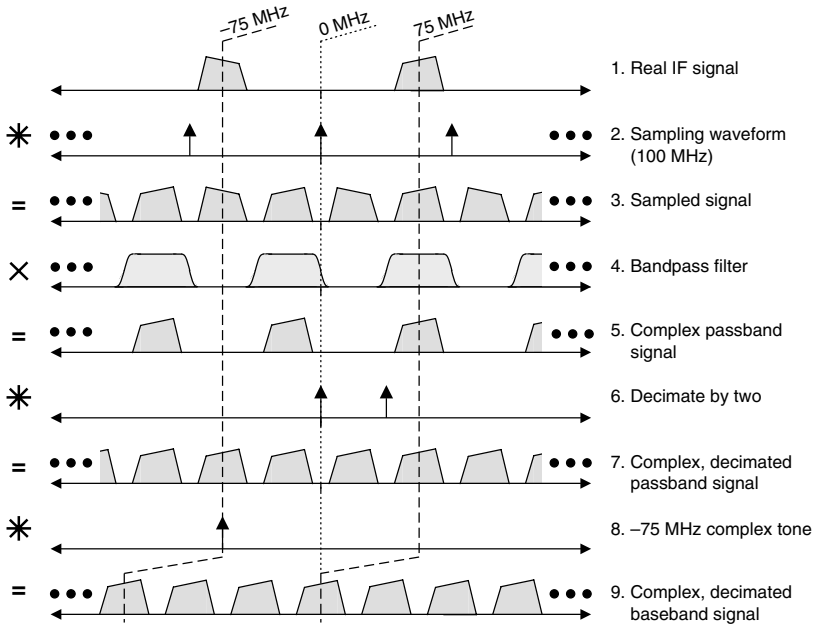

FIGURE 25.14 Direct digital downconversion in the frequency domain

Writing that impulse response as $h(n) = h_I(n) + jh_Q(n)$, using real functions $h_I(n)$ and $h_Q(n)$, the line 4 operation becomes

$$\begin{aligned}[\text{line 5}] &= [\text{line 3}] * h(n) \\ &= [\text{line 3}] * [h_I(n) \\ &\quad + jh_Q(n)] \\ &= ([\text{line 3}] * h_I(n)) \\ &\quad + j([\text{line 3}] * jh_Q(n)) \\ &= I_5 \\ &\quad + jQ_5\end{aligned}$$

where the fact that line 3 is real in the time domain was used in the last step. In Figure 25.15, the sampled IF signal, therefore, passes through different FIR filters

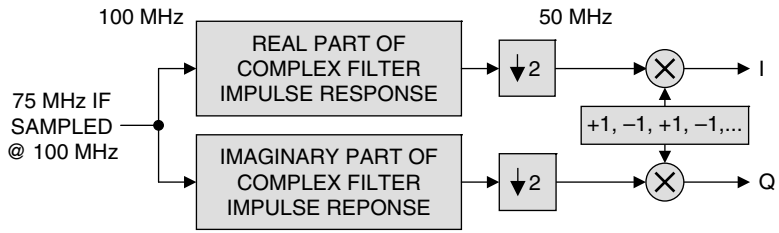

FIGURE 25.15 Time-domain implementation of a direct digital downconverter

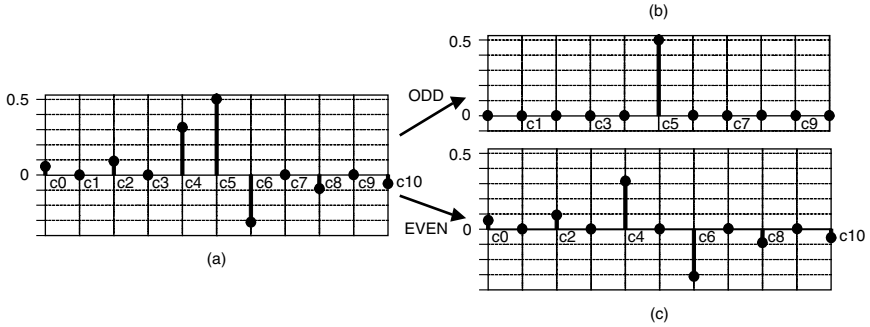

FIGURE 25.16 (*a*) Halfband bandpass filter coefficients for a direct digital downconverter, (*b*) real (odd), and (*c*) imaginary (even) parts of the complex impulse response

(FIR filters are described in Section 25.4), where the top and bottom filters apply the real and imaginary parts of the coefficients, respectively. The equivalent complex-impulse-response filter, with the frequency response shown in line 4 of Figure 25.14, is a halfband filter because that frequency response and a version shifted in frequency by half a period sum to a constant. This property causes almost half of its impulse-response coefficients to be zero. Figure 25.16*a* illustrates the coefficients of a typical filter for this application. All of the odd-numbered coefficients, except for the one in the center, are zero, so the filter is very efficient to implement, as the zero coefficients don't require multipliers. The frequency response's symmetry about 1/4 of the sampling rate causes the even-and odd-numbered coefficients to be purely real and purely imaginary, respectively, so the even-and odd-numbered coefficients are used to, respectively, create I and Q, as shown in Figure 25.16*b* and *c*.

After the filters, the complex signal is decimated by 2 to produce a 50 MHz output sample rate. The final spectral convolution by a –75 MHz tone is accomplished by negating every other sample.

In Figure 25.17, we transform the system of Figure 25.15 to make it more computationally efficient. We begin with the structure in Figure 25.17*a*, which shows the filtering in detail using τ to indicate each clock-interval delay. The location of the one nonzero coefficient in the real part $h_I(n)$ of the Figure 25.16 impulse response corresponds to an odd-numbered delay, so $h_I(n)$ is realized using a single delay and some number of double delays. The imaginary part $h_Q(n)$ of the impulse response, in contrast, has nonzero coefficients only at even numbers of delays, so it is realized with double delays only.

The architecture can then be further simplified by moving the decimation ahead of the 2τ delays, as shown in Figure 25.17*b*. This changes each double delay to a single delay at the lower clock rate at which the filter computations are now more efficiently clocked. Optionally, the negation of alternate samples at the output can now be relocated to the decimation's output. Each delay that the negation crosses as it moves in this transformation causes a net sign change in the signal, so each signal path between the old location and the new that contains odd numbers of delays requires coefficient negation to compensate. The result in the design of Figure 25.17*c* is negation of alternate coefficients in the Q filter, as shown in Figure 25.17*c*.

The optional negation-moving transformation just described yields a simple interpretation of system operation. Figure 25.12 shows that the leading τ delay, decimation, and sign-negation operations of Figure 25.17*c* work together to steer I and Q samples into the upper and lower filter paths, respectively, but the samples that are

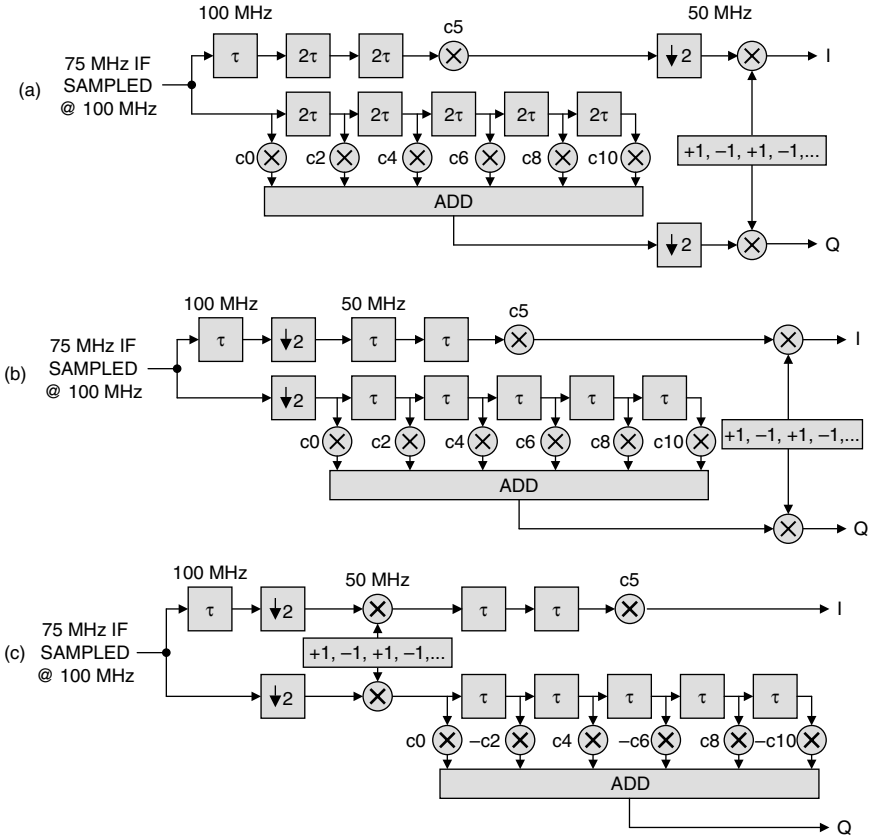

FIGURE 25.17 Direct digital downconverter: (*a*) baseline implementation, (*b*) decimating before filters, and (*c*) inverting every other sample after decimation

now aligned in time as they pass through the remaining processing do not actually correspond to the same points on the IF signal input's time line, since the I and Q samples were derived from alternate ADC samples. However, the Q filter with alternate coefficients negated, shown in Figure 25.18, actually approximates the half-sample delay

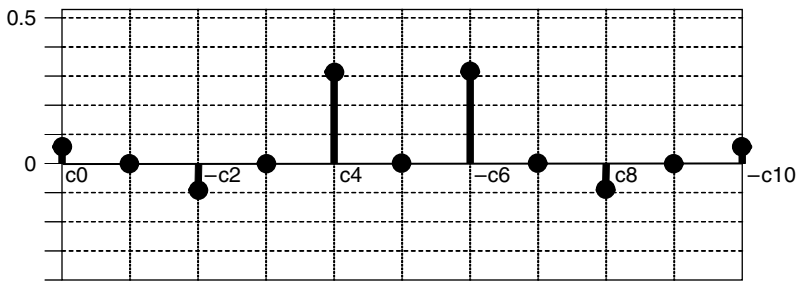

FIGURE 25.18 Negated-alternate-signs version of Q filter coefficients

required to realign the data in the two paths, and this causes the I and Q output values to be effectively sampled at the same instants.

Signal-Sampling Considerations. Actual devices and signals introduce errors. For example, clock jitter results in errors in the sampled output of an ADC, as shown in Figure 25.19. In addition, real ADCs also add internal jitter, or *aperture uncertainty*, which must be taken into account.[6] If the errors in the effective sampling instant introduced by these jitters are uncorrelated, a reasonable approximation, the RMS sample-time jitter they introduce, t_J, is

$$t_J = \sqrt{\left[(t_{J(\mathrm{ADC})})^2 + (t_{J(\mathrm{CLOCK})})^2\right]}$$

where $t_{J(\mathrm{ADC})}$ and $t_{J(\mathrm{CLOCK})}$ are the RMS sample time jitters introduced by the ADC and the clock, respectively.

A sinusoidal input signal of amplitude A and frequency f is expressed as

$$v(t) = A\sin(2\pi ft)$$

which has derivative

$$dv(t)/dt = A2\pi f\cos(2\pi ft)$$

The maximum error due to jitter occurs at $t = 0$, when the derivative of the signal is at its peak, or

$$dv(0)/dt = A2\pi f$$

The RMS error voltage, V_e, produced by an RMS sample time jitter, $t_{J,}$ is given by

$$V_e = A2\pi f\, t_J$$

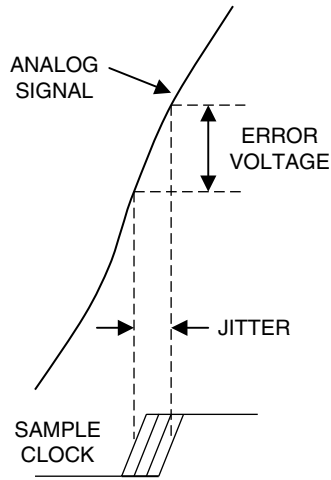

FIGURE 25.19 RMS jitter vs. RMS noise

This error voltage limits the theoretical maximum SNR of an ADC by

$$\mathrm{SNR}_{\max} = 20\log(A/V_e) = -20\log[(2\pi f t_j]$$

This relationship is presented in Figure 25.20, which plots on the left and right axes, respectively, the SNR and the equivalent ADC effective number of bits or ENOB (≈ SNR/6 dB), both versus analog frequency and for different values of RMS sample jitter. Due to a variety of error sources internal to an ADC (aperture uncertainty, nonlinearities, added noise, etc.), the specified ENOB of an ADC is always less than the number of bits it provides. For example, a 14-bit ADC typically has an ENOB of 12.

With the bandpass sampling technique described earlier, where the ADC can sample at a rate that is considerably lower than the analog frequencies being sampled, it might seem attractive to do away with the receiver altogether and sample the RF signal directly. Although this is possible, ADC limitations restrict the performance of such architectures. First, the analog front end of an ADC has a lowpass 3 dB cutoff frequency specified by the manufacturer. ADC input frequencies should be kept well below this cutoff. Second, as shown previously in Figure 25.20, sampling the RF signal directly will dramatically increase the slew rate of the signal presented to the ADC, thus requiring very low levels of RMS clock jitter. Also, the ADC has inherent nonlinearities that produce spurs in the ADC output, a problem which typically worsens with increasing input frequency. ADC datasheets specify the *spur-free dynamic range (SFDR)* of the device, which is typically defined as the dB difference in signal level between the desired signal and the largest spur measured at the ADC output with a single tone applied to the input. The SFDR of a typical ADC is higher than its specified SNR. Unfortunately, there are many definitions of SFDR, so the designer is advised to read manufacturers' datasheets carefully in this regard. As mentioned earlier, the SNR of a sampled signal can be increased by filtering to eliminate noise in parts of the spectrum that are otherwise unused. However, the spurs generated by an ADC may lie in the band of interest, where filtering would be inappropriate. Therefore, spurs lower than the unfiltered noise level can become relatively significant after ADC noise is reduced through filtering.

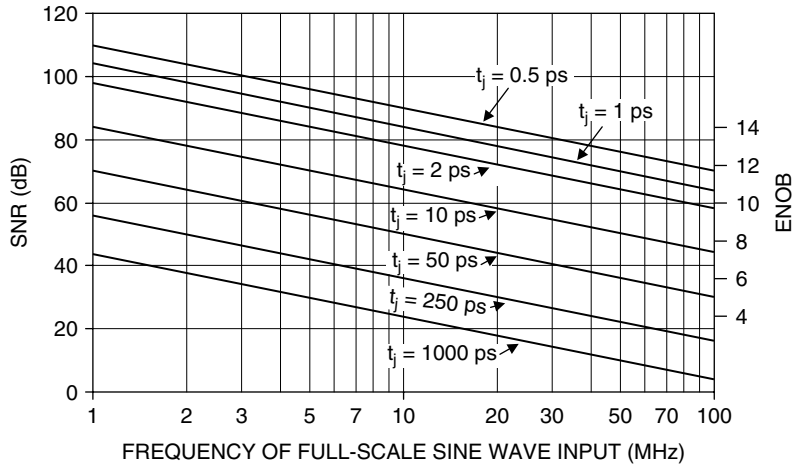

FIGURE 25.20 Signal-to-noise ratio vs. analog frequency for varying sample jitter

Multi-Beam Digital Beamforming. An important application of digital technology is for the beamforming function in a phased array antenna system. Figure 25.21*a* depicts an analog beamforming system. The wavefront shown can be thought of as the return from a target of interest. Note that the wavefront will hit each element of the array at different times. In order to form a beam in that particular direction, each element of the array needs to be followed by a time delay unit that delays the signal received at each element by the appropriate amount, such that when all of the outputs of the time delays are summed, they add up coherently to form a beam in the desired direction. If the system has a narrow bandwidth (bandwidth < ~5% of RF frequency) and the antenna beamwidth is not too narrow (so that the 3 dB beamwidth in degrees is greater than the percent bandwidth), the time delay can be approximated using phase shifters. Wide bandwidth systems require "true" time delays in order to form the beams and preserve the bandwidth. The receiver would follow the analog beamformer, as shown in the figure. Figure 25.21*b* shows an extreme application of digital beamforming, where

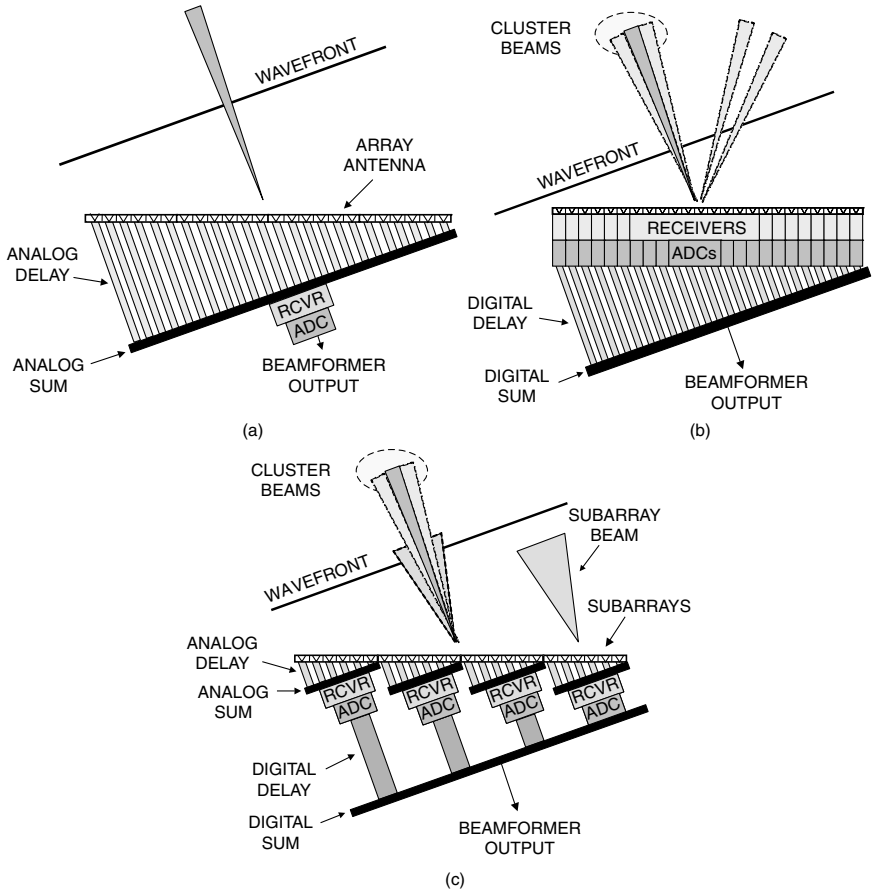

FIGURE 25.21 (*a*) Analog beamformer, (*b*) every-element digital beamformer, and (*c*) subarray digital beamformer

a receiver and ADC are behind every element. In this system, the time delay is implemented either as a digital phase shift or digital time delay, followed by a digital summer. This configuration allows beams to be formed in any direction, and multiple beams can be formed simultaneously, if desired, by using the same sample data and implementing different time delays to form the different beams. However, at this writing, putting a digital receiver behind every element is expensive and is usually not feasible for most large antenna applications (i.e., for systems with thousands of elements). One compromise solution is shown in Figure 25.21*c*, where analog beamforming is used to implement subarrays, which are followed by digital receivers and digital time delays.

Digital beamforming offers several advantages over analog beamforming. With an analog beamformer, usually only one beam is formed at a time. Radars are typically required to perform multiple functions, such as volume surveillance, target confirmation, tracking, etc. With only one beam at a time, there may not be enough time available to perform all of the required functions. A digital beamformer allows the formation of multiple simultaneous beams, allowing the volume surveillance function to be performed much more quickly, allowing more time to do other things. Of course, in order to form multiple simultaneous receive beams, the transmitted beam must be made broader to encompass the receive beams, which might require a more powerful transmitter or more integration on receive to provide the same performance as a single-beam system.

Another advantage has to do with dynamic range. In an analog beamforming system, there is only one receiver and ADC, and the dynamic range performance is limited to the capability of a single channel. In a digital beamforming system, there are multiple receivers and ADCs, and the number of ADCs that are combined determines the system dynamic range. For example, if the outputs of 100 ADCs were combined to form a beam, assuming that each ADC induces noise that is of equal amplitude and uncorrelated with the others, there would be a 20 dB increase in system dynamic range, compared to a single-receiver system using the same ADC.

Figure 25.22 shows a block diagram of a typical digital beamforming system. Each antenna output port, be it from an element or a subarray, is followed by a digital

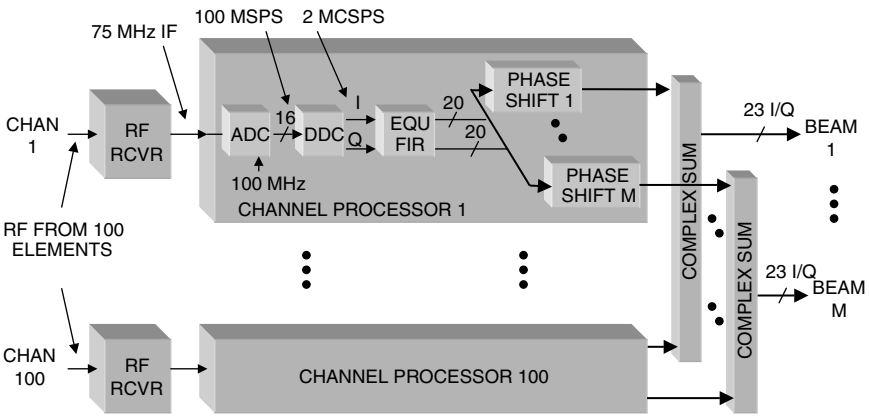

FIGURE 25.22 Typical digital beamformer

downconverter and an equalization filter (EQU FIR). The equalization filter is typically a complex finite impulse response (FIR) filter (described later) that adjusts the frequency response of each channel so that its passband matches the other channels in phase and amplitude before it is summed with the other channels in the beamformer. The coefficients of this filter are determined through a calibration process. During calibration, a test signal is presented to the RF input of all channels. This signal is typically a swept frequency tone or a noise input that covers the channel bandwidth. The ADC samples of all channels are collected simultaneously and complex weights are calculated for the equalization filters that force the frequency response of each channel to be matched. Once the channel is equalized, a unique time delay is implemented for each beam to be formed. As mentioned earlier, this time delay can be realized either as a phase shift for a narrowband system or as a time delay for a wideband system. A phase shift can be implemented with a complex multiply or a CORDIC operation, both of which will be described later. A time delay can be implemented with a FIR filter that imposes a linearly changing phase shift over frequency on the signal. Once the time delay is realized in each channel, the appropriate complex time-delayed signals from all of the channels are summed to form a beam. M complex summers are required to form M beams.

Digital Pulse Compression. Pulse compression is another signal processing function that is predominantly being performed digitally in radar systems. However, at this writing, many systems still exist with analog-delay-line pulse compressors. In these systems, analog pulse compression is performed at an IF, followed by the ADC in the processing chain. Because pulse compression increases the SNR of the signal, performing it before sampling increases the dynamic range requirement of the ADC. In a digital pulse compression system, the ADC precedes the pulse compressor and only has to accommodate the precompression dynamic range of the signal, which can be a significantly lower requirement. The digitized signal is converted to baseband and passed to the digital pulse compressor. The increased dynamic range due to the pulse compression gain is accommodated by increasing the number of bits in the digital computation.

Chapter 8 is devoted totally to pulse compression radar. In summary, there are two basic approaches to implementing digital pulse compression: time-domain and frequency-domain convolution. A generic time-domain convolver consists of a complex FIR filter, where the coefficients are the complex conjugate of the transmitted baseband waveform samples in time-reversed order (which is also the definition of the matched filter for the transmitted signal). This architecture can compress any arbitrary waveform. A simplified version of the architecture results when the modulation is a binary phase code. In this case, the coefficients are either +1 or –1, so the arithmetic performed for each sample is a complex sum or subtraction instead of a full complex multiplication.

Pulse compression may also be accomplished by operating in the frequency domain, where it is referred to as *fast convolution.* In this case, the baseband samples of the receive data and the reference transmit waveform are passed through fast Fourier transforms (FFTs), the data FFT outputs are multiplied point-by-point by the complex conjugate of the reference FFT outputs, and then the result is converted back to the time domain by an inverse FFT. In general, it is more hardware efficient to perform time-domain convolution for a small number of coefficients and frequency domain convolution for a large number (more than 8 or 16) coefficients.

25.3 TRANSMIT CHANNEL PROCESSING

Before digital technology became widely available, analog techniques were employed to generate radar transmit waveforms. Simple pulsed systems used analog RF switches to gate the LO on and off. Frequency modulated signals were generated by surface acoustic wave (SAW) devices. Simple binary phase modulation schemes like pseudo-random noise waveforms were also possible. Digital technology, however, presents the radar system designer with many more options and allows arbitrarily modulated transmit waveforms to be modified pulse-to-pulse if desired. This section describes several of the techniques commonly used to generate the radar transmit signal digitally.

Direct Digital Synthesizer (DDS). Figure 25.23 shows a block diagram of this technique, in which a numerically controlled oscillator (NCO) generates a digitized sinusoid that is converted to an analog signal by a digital-to-analog converter (DAC). Figure 25.24 demonstrates how an NCO operates to produce a sine wave. The n-bit tuning word is actually a phase increment that determines the frequency of the sine wave output. The phase increment is expressed in a format called Binary Angle Measurement (BAM), in which the most significant bit (MSB) of the word represents 180°, the next bit represents 90°, and so on. In the phase accumulator, the tuning word is added to the output of a running sum, implemented as an adder followed by a register (REG). This produces a uniformly increasing phase, incremented at the system clock rate. The m MSBs of the running sum are sent to a phase-to-amplitude converter, which is a lookup table that produces a k-bit value that represents the amplitude of the sine wave at the input phase. If we represent the tuning word by M, the sample frequency by f_s, and the number of bits in the phase accumulator by n, then the frequency of the output sine wave can be expressed as $Mf_s/2^n$.

In this scheme, the phase represented by the running sum will overflow when it crosses over 360°. The advantage of expressing the phase in BAM notation is that it allows modulo-2π arithmetic and overflows are automatically taken care of, since a 360° phase shift is the same as 0°. For example, assume we have a 3-bit BAM notation, which means that the least significant bit (LSB) represents a phase shift of 45°. Let's also assume that the tuning word is represented by 001 for a phase increment of 45° every clock. The running sum phase would steadily increase on every clock edge, becoming 000 (0°), 001 (45°), …, 110 (270°), and 111 (315°). On the next clock edge, the phase should be represented by 1000 for 360°. However, we are only provided a 3-bit adder, so the MSB is simply lost, leaving us with a phase code of 000 (0°), which is the same as 360°. Therefore, the resulting phase waveform takes on a sawtooth pattern, linearly ramping from 0° to not quite 360° and then resetting to 0° and ramping again.

An important feature of an NCO for a radar application is the CLEAR signal shown going to the phase accumulator register. For a coherent radar exciter implementation, the transmit signal must start at the same phase on every pulse. Otherwise, the

FIGURE 25.23 Direct Digital Synthesizer (DDS)

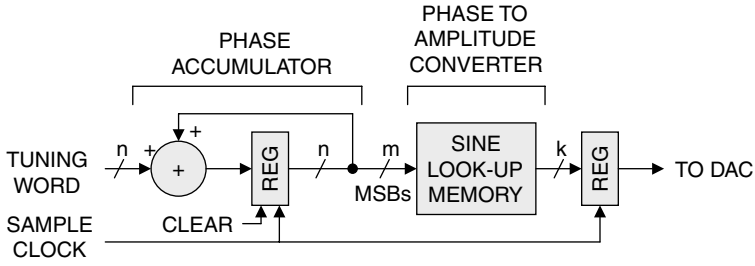

FIGURE 25.24 NCO block diagram

transmitted signal will have arbitrary phase from pulse to pulse, making doppler processing difficult if not impossible. The CLEAR control provides the means to do this. In some applications, like for a transmit beamformer, the starting phase in each channel may need to be different in order to steer the beam. In this case, we could provide a mechanism to set the phase to the desired value at the beginning of a pulse instead of simply clearing it.

The DDS can also be used to generate linear and nonlinear FM "chirp" waveforms. This is accomplished by providing circuitry that changes the tuning word from sample to sample in order to provide the desired frequency (or phase) modulation. For example, a linear FM chirp waveform requires a phase that changes in a square-law fashion with time. This can be accomplished by changing the tuning word (or phase step size) in a linearly increasing or decreasing way on every sample.

Digital Upconverter (DUC). Another popular method to implement a transmit waveform is through digital upconversion, also referred to as *arbitrary waveform generation.* In this technique, a digital complex baseband waveform, usually read from a memory, is first interpolated to a higher sample rate, and then modulated with digitized sine and cosine signals to produce a modulated carrier. Figure 25.25 provides a block diagram of a DUC that translates a complex baseband signal up to a 25 MHz IF. The baseband I and Q signals enter the DUC at a rate of 2 MCSPS and are first up-sampled by a factor of 50. This is accomplished by inserting 49 zeroes between each input sample and increasing the clock rate to 100 MHz. This signal is then passed through a digital lowpass filter that performs the interpolation. These signals are then multiplied

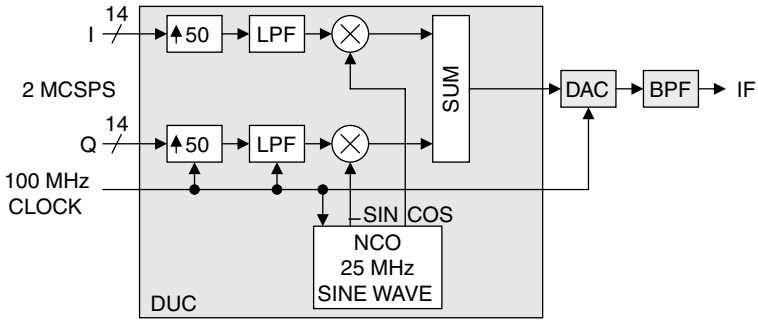

FIGURE 25.25 Digital upconverter (DUC)

by digitized sine and cosine waveforms for the modulation carrier frequency, producing a complex modulated IF as an output. These signals are digitally summed, converted to analog through a DAC, and passed through a bandpass filter to produce an IF output. For large upsampling ratios, a cascaded-integrator comb (CIC) interpolator, described in the next section, provides an efficient implementation.

25.4 DSP TOOLS

This section will describe various processing architectures and techniques that are available to DSP engineers.

Phase Shift. The phase shift is a core element in DSP design, and there are several techniques available to implement one. The most straightforward approach is to simply perform a complex multiply, as shown in Figure 25.26. In this example, the complex input sample is denoted as $A + jB$, which is multiplied by the complex coefficient $C + jD$ to produce $(AC - BD) + j(AD + BC)$ in order to effect the phase shift. This operation requires four multipliers and two adders.

After some manipulation, the following can be shown:

$$I = (AC - BD) = D(A - B) + A(C - D)$$

$$Q = (AD + BC) = C(A + B) - A(C - D)$$

Noting that the final term is the same in both equations, we see that this complex multiplier can be implemented with only three real multipliers and five real adds. This can be important if real multipliers are at a premium. Figure 25.27 shows a block diagram of this architecture.

CORDIC Processor. An efficient and versatile algorithm that can implement a phase shift without using multipliers is the **CO**ordinate **R**otation **DI**gital **C**omputer (CORDIC) function, first described by Volder[7] in 1959. The CORDIC can implement

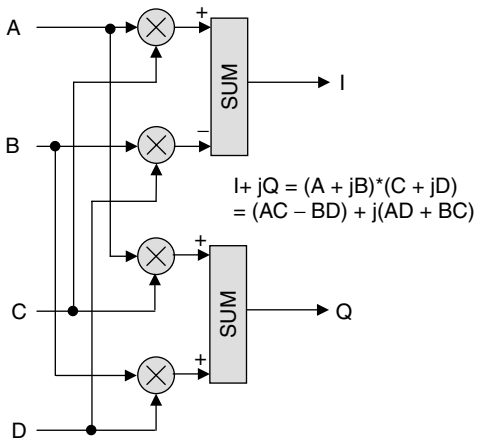

FIGURE 25.26 Standard complex multiply

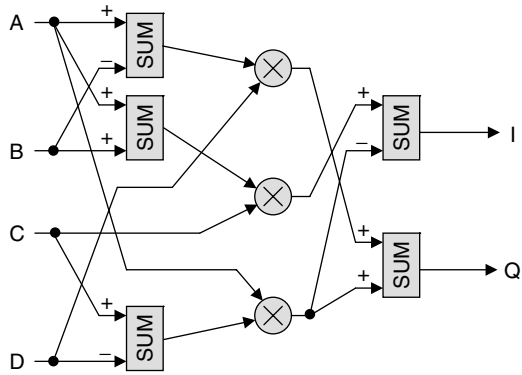

FIGURE 25.27 Complex multiply with three real multipliers

a variety of functions, including sine, cosine, vector rotation (phase shift), polar-to-rectangular and rectangular-to-polar conversions, arctangent, arcsine, arccosine, and vector magnitude, through an iterative process that just uses bit shifts and adds.[8] The following discussion describes the CORDIC algorithm.

The equations that shift the phase of complex number $I_0 + jQ_0$ by an angle θ to produce $I_1 + jQ_1$ are as follows:

$$I_1 = I_0(\cos(\theta)) - Q_0(\sin(\theta))$$

$$Q_1 = I_0(\sin(\theta)) + Q_0(\cos(\theta))$$

These equations can be manipulated to provide

$$I_1 = \cos(\theta)[I_0 - Q_0(\tan(\theta))]$$

$$Q_1 = \cos(\theta)[Q_0 + I_0(\tan(\theta))]$$

The CORDIC algorithm takes advantage of this relationship to approximate an arbitrary phase shift by implementing multiple stages of phase shifts, where the tangent of the phase shift in each successive stage is the next smaller fractional power of 2, and multiplication by this number can be implemented by shifting the input data bits an integer number of places. The first few stages are as follows:

$$I_1 = \cos(\theta_0)[I_0 - Q_0(\tan(\theta_0))] = \cos(\theta_0)[I_0 - Q_0(1)]$$

$$Q_1 = \cos(\theta_0)[Q_0 + I_0(\tan(\theta_0))] = \cos(\theta_0)[Q_0 + I_0(1)]$$

$$I_2 = \cos(\theta_1)[I_1 - Q_1(\tan(\theta_1))] = \cos(\theta_1)[I_1 - Q_1(½)]$$

$$Q_2 = \cos(\theta_1)[Q_1 + I_1(\tan(\theta_1))] = \cos(\theta_1)[Q_1 + I_1(½)]$$

Table 25.1 shows these parameters for an eight-stage CORDIC processor. Each row of the table represents successive iterations of the algorithm. The **tan(θ_i)** column shows the factor by which the I and Q values are multiplied for each iteration. Note that these values are fractional powers of 2, so the multiplication can be realized by shifting the binary I and Q values right by i places. The θ_i column shows the arctangent of this factor, which can also be thought of as the phase shift applied during each iteration.

TABLE 25.1 CORDIC Parameters for First Eight Stages

i	$\tan(\theta_i)$	θ_i (deg)	$\cos(\theta_i)$	P $[\cos(\theta_i)]$
0	1	45.000	0.707107	0.707107
1	1/2	26.565	0.894427	0.632456
2	1/4	14.036	0.970143	0.613572
3	1/8	7.1250	0.992278	0.608834
4	1/16	3.5763	0.998053	0.607648
5	1/32	1.7899	0.999512	0.607352
6	1/64	0.8951	0.999878	0.607278
7	1/128	0.4476	0.999970	0.607259

The $\cos(\theta_i)$ column shows the cosine of this angle, which should be multiplied by the result of each iteration, as shown in the equations above. In actual applications, however, this cosine multiplication is not performed at every iteration. At each stage, the implied factor that needs to be multiplied by the IQ outputs of the stage in order to provide the correct answer is the product of all of the cosines up to that point, as shown in the P**[cos(θ_i)]** column. For a large number of iterations, this product of cosines converges to a value of 0.607253. In most cases, this scaling can be compensated for in later stages of processing. The inverse of this factor, 1.64676 for a large number of iterations, is the processing gain imposed on the IQ results of the CORDIC. If integer arithmetic is performed, an extra bit should be provided at the most significant end of the adders in order to accommodate this increased signal level.

Figure 25.28 is a flow chart that represents the CORDIC algorithm to implement a phase shift. The inputs to the algorithm are the I_{in}, Q_{in}, and ϕ_{in} (the desired phase shift). The variable i will keep track of the processing stage being performed and is initialized to zero. The basic algorithm can perform a phase shift between ±90°. If the desired phase shift is outside of that range, the input I and Q values are first negated, imposing a 180° phase shift, and 180° is subtracted from the desired phase shift. The new phase shift is now within ±90°, and the algorithm proceeds normally.

Next, the algorithm loops through N iterations with the goal of driving the residual phase error, ϕ, to zero. In each iteration, a new ϕ is calculated by subtracting or adding the phase shift for

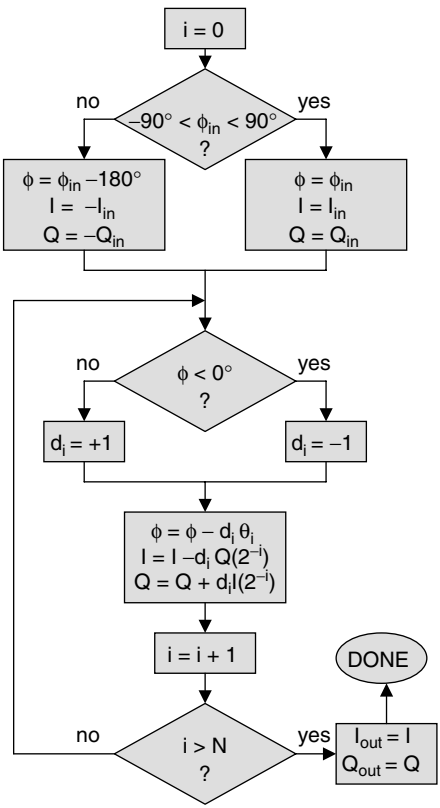

FIGURE. 25.28 CORDIC algorithm flow chart

that stage (θ_i from the table) to the previous value of ϕ. If $\phi < 0$, θ_i is added to ϕ. Otherwise, θ_i is subtracted from ϕ. In each stage, the Q (or I) input is divided by a factor of 2^i by shifting the number to the right by i bits, and then added to or subtracted from the I (or Q) input, depending on the sign of ϕ. The variable i is incremented and the process repeats until $i > N$, at which point the phase-shifted results are available.

Figure 25.29 is a block diagram of an eight-stage CORDIC processor that implements a phase shift, where each stage represents an iteration in the flow chart. An N-stage processor provides a phase shift that is accurate to within $\pm\theta_N$ degrees (from the table), so the more stages in the processor, the more accurate the answer. The input I and Q values change on the rising edge of an assumed sample clock. In the first stage, the I value is either added to or subtracted from the Q value in the ADD/SUB block. The control block on the bottom of the figure determines whether additions or subtractions are performed at each stage, based on the algorithm described previously. If the ADD/SUB block in the Q channel performs an addition, the same block in the I channel will perform a subtraction, and vice-versa. The result of the ADD/SUB blocks is stored in a register (REG) on the next clock edge and passed to the next stage of processing. In this implementation the last block labeled (PASS/INV) performs the required inversion of I and Q if the desired phase shift is beyond the $\pm 90^o$ range of the algorithm. The final multiplication by a constant is optional, as described earlier.

The architecture shown in Figure 25.29 is a good example of a *pipelined* processor, in which a portion of the computation is performed and the result is stored in a register on each rising edge of the sample clock and passed to the next stage of processing. The processor would still function if the registers were removed. However, in that case, when the input I and Q values changed, the final output would not be useable until the results of the new input values rippled through all of the stages of processing, which would generally be an unacceptably long period of time. In a pipelined processor, a small portion of the total calculations is performed at a time, and the result is stored in a register and passed to the next processing stage. This architecture provides a higher *throughput* than the nonpipelined version, which means that the final result can be produced at a much higher sample rate, which is inversely proportional to the delay of a single stage. The *latency* of a pipelined processor refers to the delay experienced between the time a new data sample is entered into the processor and the time that the result based on that input is available on the output. The eight-stage, pipelined CORDIC processor shown in the figure would have a latency equivalent to eight clock periods and a throughput equivalent to the clock rate (i.e., once the pipeline is filled and the first result is available on the output, successive clocks will produce new outputs at the clock rate).

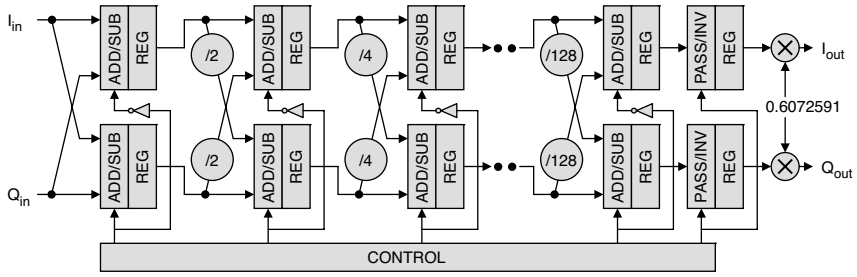

FIGURE 25.29 Eight-stage CORDIC processor

Digital Filters and Applications. This section describes several of the major forms of digital filters and how they are used in radar signal processing.

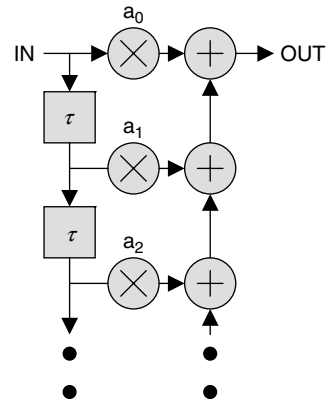

FIGURE 25.30 General direct-form FIR filter block diagram

Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) Filters. Figure 25.30 shows a block diagram of a direct-form digital FIR filter. The input sample feeds a shift register, where each block labeled τ indicates a one-sample delay in the shift register. The input sample and the output of each stage of the shift register are multiplied by unique coefficients, and the multiplier outputs are summed to produce the filtered output. Software tools exist that generate these coefficients and the number required when the user provides the desired filter characteristics, such as filter type (lowpass, highpass, bandpass, etc.), sample rate, cutoff and stopband frequency, desired passband ripple, and stopband attenuation. The filter shown is referred to as a real FIR filter, since the input data and coefficients are real values and real mathematical operations are performed. In a complex FIR filter, the data samples, coefficients, and math are complex.

This type of filter is termed *finite* impulse response because an impulse presented at the input (a single sample of "1" surrounded by samples of zeroes) would produce a finite-length output, consisting of the coefficients of the filter output in order as the "1" propagates down the shift register, as shown in Figure 25.31 for a FIR filter with seven coefficients (commonly referred to as a 7-tap FIR filter). In this example, zero-valued samples are first clocked into the FIR filter shift register, filling the shift register with zeroes and forcing the filter output to be zero. When the sample with a value of "1" is clocked into the filter, the filter output produces the first coefficient, a_0, since the other samples in the filter are still zero. On the next clock, the "1" moves to the second tap of the shift register, and a "0" is clocked into the first tap, forcing the filter output to produce the second filter coefficient, a_1. On successive clocks, the "1" propagates through the shift register, while zeroes are clocked into the shift register input, producing all of the filter coefficients on the output in sequence. The FIR filter uses feed-forward terms only, meaning that the output values only depend on the input values with no feedback terms.

Figure 25.32 depicts the general form for an infinite impulse response (IIR) filter. IIR filters make use of feed-forward and feedback terms. They are referred to as *infinite* impulse response because an impulse presented at the input to the filter will produce an infinite string of nonzero outputs in an ideal situation.

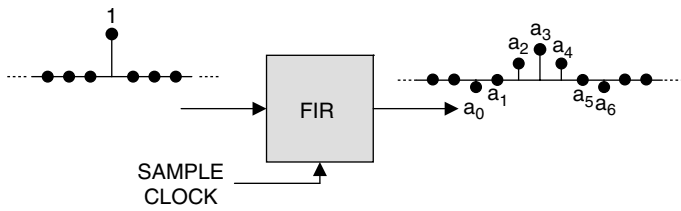

FIGURE 25.31 Impulse response of 7-tap FIR filter

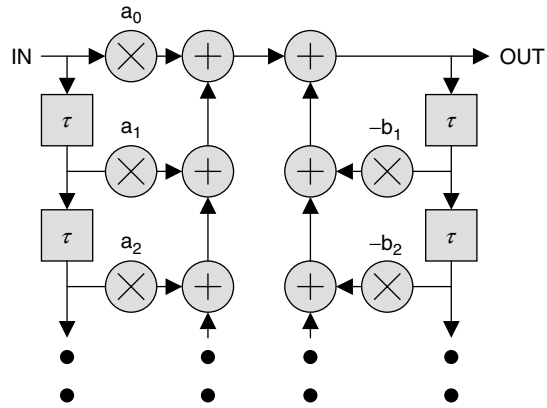

FIGURE 25.32 General IIR filter block diagram

Compared to FIR filters, IIR filters offer several advantages. In general, they require less processing and memory to implement similar functions. It is also easier to implement some filter responses as IIR rather than FIR filters. However, without careful design, IIR filter responses can be very sensitive to coefficient quantization limitations and could exhibit a tendency to *overflow* (i.e., produce an output that exceeds the processor dynamic range, determined by the number of bits in the datapath). Although IIR filters are almost never used in radar systems for these and a variety of historical reasons, a cautious designer might find an application where they can be used to good advantage.

By contrast, FIR filters are inherently stable. Real FIR filters with symmetric coefficients automatically provide a linear phase shift over frequency, introducing little or no phase distortion to the filtered signal, which is highly desirable in many applications. Because FIR filters require no feedback, they are easier to use in very high-speed applications than IIR filters, which typically require the computation of an output sample before the next output sample can be formed. Complex FIR filters, where a complex multiplication is performed at each tap, can be used to implement equalization filters, time delays, and pulse compression filters.

Figure 25.33 shows an alternative form for a FIR filter, called a *transposed form* FIR filter. In this configuration, each input sample is multiplied by all of the coefficients at once, with the sample delays between the summer outputs.

If the coefficients of a FIR filter are symmetric, so that the coefficients on either side of the center of the filter are mirror images of each other (as is the case with linear phase filters), multipliers can be saved by adding the samples that get multiplied by the same coefficient first, thereby requiring about half as many multipliers, as shown in Figure 25.34 for a 7-tap example.

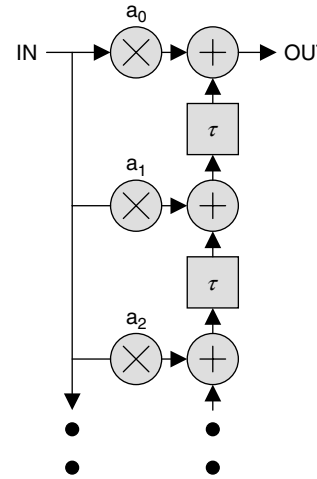

FIGURE 25.33 Transposed form FIR filter

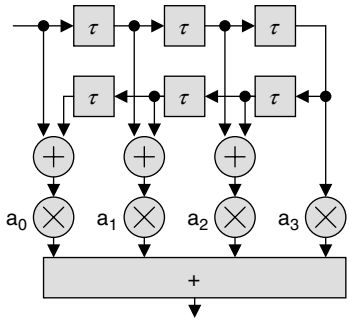

FIGURE 25.34 7-tap FIR filter with symmetric coefficients

Decimation Filters. As mentioned previously, the complexity and cost of a signal processor, in terms of the amount of system resources required to implement it, generally varies linearly with the data sample rate. For this reason, in most system applications, it is cost-effective to reduce the data sample rate to a value that is just adequate to support the bandwidth of the system. In applications where the sample rate of a signal is to be decreased (decimated), the frequency content of the signal must first be reduced so that the Nyquist criterion is satisfied for the new sample rate. This can be accomplished by first passing the signal through a digital FIR filter to restrict the bandwidth of the signal to less than half of the decimated sample rate, and then reducing the sample rate of the filtered signal by a factor of R by selecting every R^{th} sample, as described in the previous discussion of decimation. A designer can take advantage of decimation by realizing that only the filter outputs that are used need to be computed. For example, if the output of a FIR filter is to be decimated by a factor of 4, only every fourth filter output needs to be computed, which reduces the required processing by a factor of 4.

Interpolation Filters. Interpolation is the process by which the sample rate of a signal is increased, for example in preparing the signal to be upconverted to an IF, as shown in Figure 25.25. Interpolators are typically FIR filters with a lowpass filter response. To increase the sample rate by a factor R, R –1 zeroes are first inserted between the low-rate data samples, creating a data stream at a sample rate R times faster than the input rate (upsampling). This data stream is then passed through the lowpass FIR filter to produce the interpolated high-sample-rate output. Of course, the FIR filter must be clocked at the higher data rate. This process is illustrated in Figure 25.35 for a four times increase in sample rate.

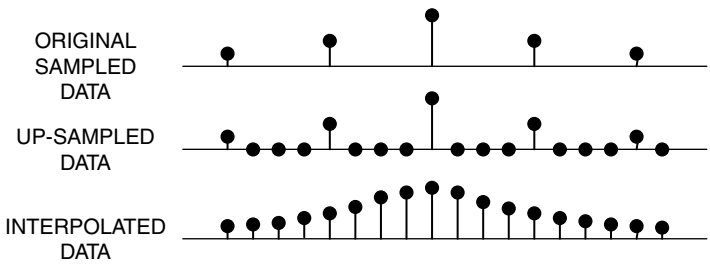

FIGURE 25.35 Illustration of interpolation filtering

Cascaded Integrator-Comb (CIC) Filters. In decimation or interpolation applications where the rate change factor is large (typically 8 or greater), a FIR filter implementation might be prohibitively costly due to the large number of filter taps that would be required. CIC filters are a class of filters introduced by Hogenauer[9] that provide a very efficient means of implementing these filter functions that do not require multipliers. Excellent descriptions of this class of filter are provided by Lyons[2] and Harris,[10] which form the basis for the following discussion.

Figure 25.36*a* shows a single-stage CIC decimator. The filter contains an integrator stage consisting of a single sample delay and an adder, followed by a comb stage with a D-stage shift register (denoted by the $D\tau$ block) and a subtractor. The comb filter gets its name because its frequency response looks like a rectified sine wave and resembles the teeth of a comb. After the comb stage, the signal is decimated by a factor R (denoted by the $\downarrow R$ block) by only passing every R^{th} sample. In most applications, the number of stages in the shift register, D, is equal to the rate change factor, R. Figure 25.36*b* depicts a CIC interpolator, where upsampling by a factor of R (denoted by the $\uparrow R$ block) is followed by a comb section and an integrator. The upsampling is accomplished by zero insertion as described in the previous section, "Interpolation Filters." Note that the processing only consists of delays and adds.

Figure 25.37*a* shows the $\sin(x)/x$ frequency response of a single-stage CIC decimator, where $R = D = 8$. The desired passband is the lightly shaded area centered at 0 Hz with bandwidth BW. The darker shaded areas with bandwidth BW in Figure 25.37*a* indicate signals that will alias into the baseband signal after decimation by 8, as shown in Figure 25.37*b* (after Lyons[2]). Note that unless BW is very small, a significant portion of out-of-band signals would get folded into the decimated baseband signal. The typical method used to improve this filter response is to increase the filter order by adding more stages. Figure 25.38 shows a three-stage CIC decimation filter, and its frequency response before and after decimation by 8 is shown in Figure 25.39*a* and *b*, respectively. Note that the aliased components are significantly reduced in amplitude, compared to the single-stage CIC filter response, and the main passband has more attenuation toward the edges. In typical applications, a CIC decimator is followed by a FIR lowpass filter and a final decimation by 2. That is, a decimate-by-16 filter would be composed of a decimate-by-8 CIC filter followed by a decimate-by-2 FIR filter. The FIR filter can be tailored to remove the undesired residual components before the final decimation. The FIR filter can also be configured to compensate for the droop in the passband response.

Figure 25.40 shows an equivalent form for a CIC decimation filter, where the decimation occurs right after the integrator section and before the comb section. The delay in the comb filter becomes a value $N\tau$, where N is equal to D/R. This allows the comb section to operate at the decimated sample rate, which makes it simpler to implement. Due to this simplification, CIC decimators are generally implemented in this form.

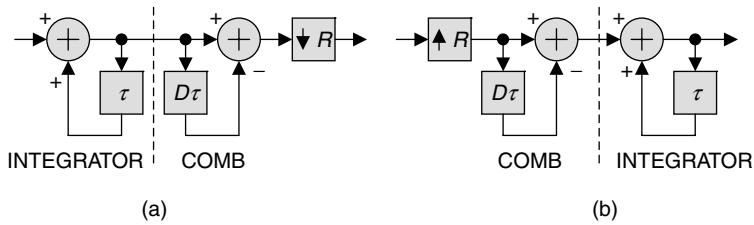

FIGURE 25.36 (*a*) CIC decimation filter and (*b*) CIC interpolation filter

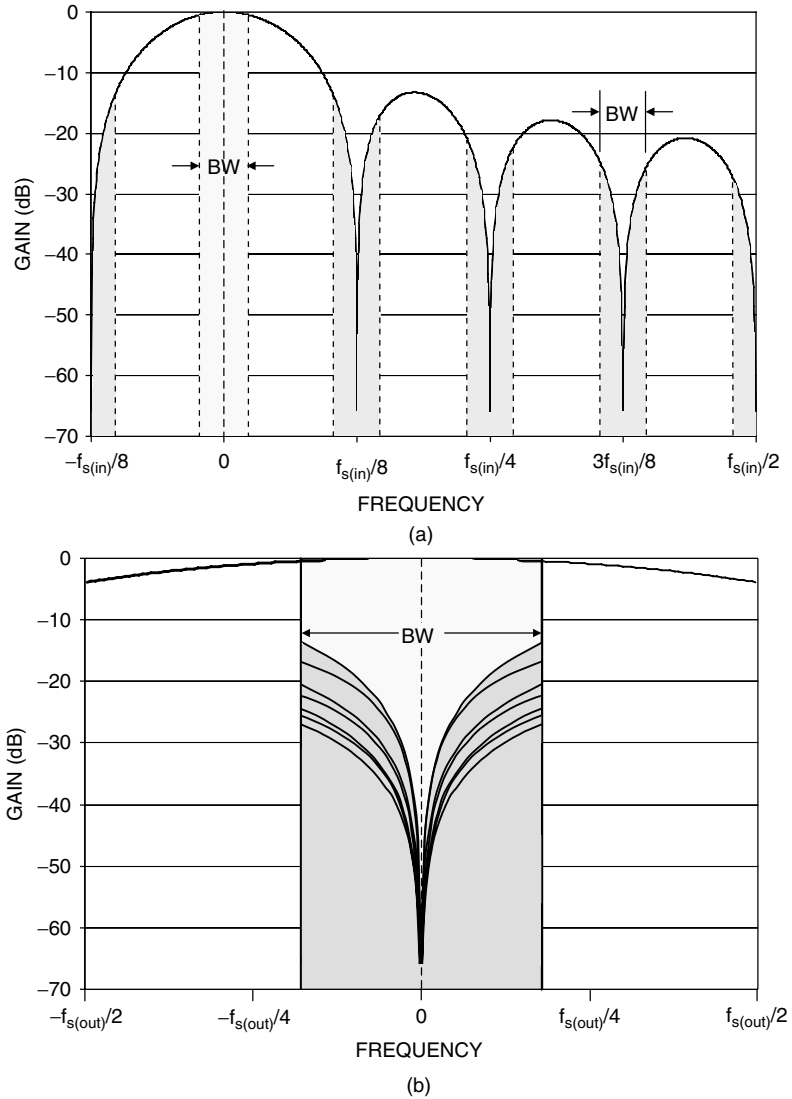

FIGURE 25.37 Frequency response of single-stage CIC decimation filter (*a*) before decimation and (*b*) after decimation

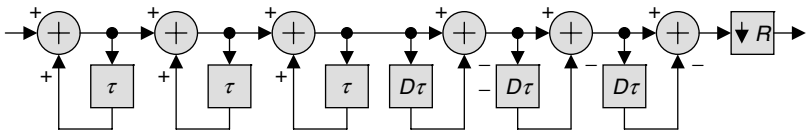

FIGURE 25.38 Three-stage CIC decimation filter

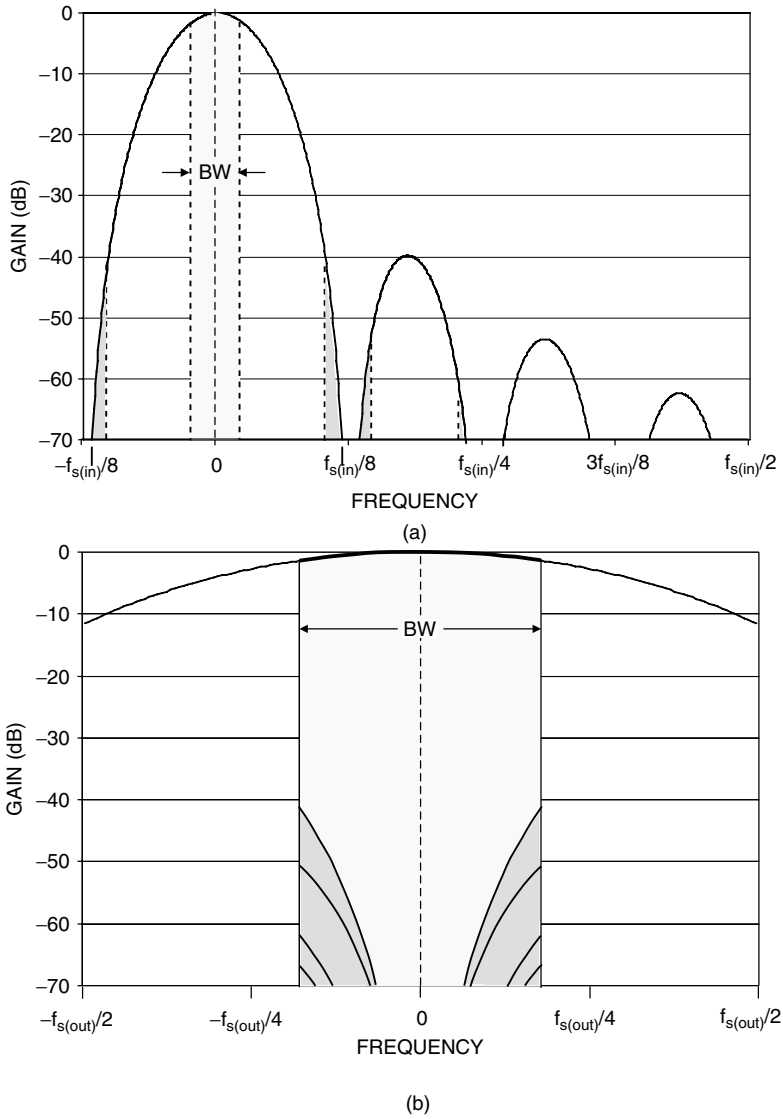

FIGURE 25.39 Frequency response of third-order CIC decimation filter (*a*) before decimation and (*b*) after decimation

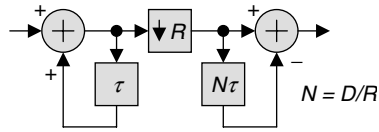

FIGURE 25.40 CIC filter with decimation after integrator

Careful inspection of the decimator architecture reveals a potential problem with the integrator. The input samples continually get added to the running sum, producing a definite overflow condition. The beauty of the architecture is that overflows are allowed and compensated for by the comb section, as long as there are enough bits in the adders to represent the maximum expected output value and the filter is implemented using two's complement arithmetic. As described by Harris,[10] the number of bits required in the adders (b_{ADDER}) is given by

$$b_{\text{ADDER}} = b_{\text{DATA}} + \text{CEIL}[\log_2(\text{GAIN})]$$

where b_{DATA} is the number of bits in the input data and CEIL[ ] indicates rounding the number in the brackets to the next highest integer. GAIN is given by

$$\text{GAIN} = R^K$$

where R is the decimation factor and K is the number of stages in the filter, resulting in

$$b_{\text{ADDER}} = b_{\text{DATA}} + \text{CEIL}[\log_2(R^K)]$$

For example, assume we have 12-bit input data ($b_{\text{DATA}} = 12$) and a 3-stage CIC filter ($K = 3$) that decimates the sample rate by a factor of 10 ($R = 10$). Substituting into this equation produces

$$b_{\text{ADDER}} = 12 + \text{CEIL}[\log_2(10^3)] = 12 + \text{CEIL}[9.966] = 12 + 10 = 22$$

In practice, although the first adder stage must support this number of bits, lower order bits may be pruned from the adders in successive stages, as described by Harris.[10]

A CIC interpolating filter would be preceded by a FIR-filter-based interpolator. CIC interpolators are described in detail in the referenced literature.

The Discrete Fourier Transform (DFT). In many sampled data systems, spectral analysis is implemented by performing the discrete Fourier transform (DFT). The DFT forms the basis for many radar signal processing algorithms, such as doppler processing and fast convolution pulse compression (described in Chapter 8), as well as radar functions such as synthetic aperture radar (SAR) and inverse synthetic aperture radar (ISAR). The DFT takes N data samples (real or complex) as input and generates N complex numbers as output, where the output samples represent the frequency content of the input data sequence. For a sample rate f_s, each output frequency sample (bin) has a width of f_s/N. The m^{th} output sample, $X(m)$, represents the amplitude and phase of the frequency content of the finite-length input sequence centered at the frequency mf_s/N.

If an input signal is exactly centered in one of the DFT frequency bins, the output will have a maximum value for that bin and nulls for all other bins. However, any frequency other than one centered in a bin will bleed into the other bins. The basic DFT bin has a frequency response similar to $\sin(x)/x$, which means that a signal in another bin might bleed into a DFT frequency bin with an attenuation as small as 13 dB. To compensate for this, the input samples can be weighted in amplitude with a wide selection of weights, such as Hanning and Hamming weights, which broaden the main lobe response of the DFT output, but reduce the amplitude of the side lobes. A thorough treatment of DFT weighting functions and their effects is given by Harris.[11]

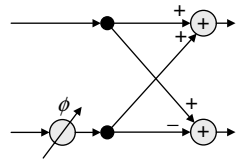

FIGURE 25.41 Radix-2 butterfly

The Fast Fourier Transform (FFT). The implementation of a DFT is computationally intensive, requiring N^2 complex multiplies. The fast Fourier transform (FFT)[12] is a very efficient technique to implement the DFT, if N is a power of 2, which requires only $(N/2)\log_2 N$ complex multiplies.

The basic computational element in an FFT is the *butterfly,* shown in Figure 25.41. In the butterfly operation, one input is phase shifted and then added to and subtracted from a second input to form two outputs. This structure is referred to as a radix-2 butterfly because it has two inputs. For certain FFT configurations, radix-4 and higher-radix butterflies provide some computational savings.

Figure 25.42 shows a radix-2, 8-point FFT. The phase shifts are represented as complex weights W_N^k, where N is the number of points in the FFT and k indicates the particular phase shift applied. W_N^k denotes a phase shift of $2k\pi/N$. These weights are often referred to as *twiddle factors*. Figure 25.43 shows the phase shifts associated with various twiddle factors.[2]

Note that the 8-point FFT consists of three stages. All of the computations in each stage are executed before proceeding to the next stage. Also note that the phase shift in the first stage, W_8^0, is zero, which requires no computation at all.

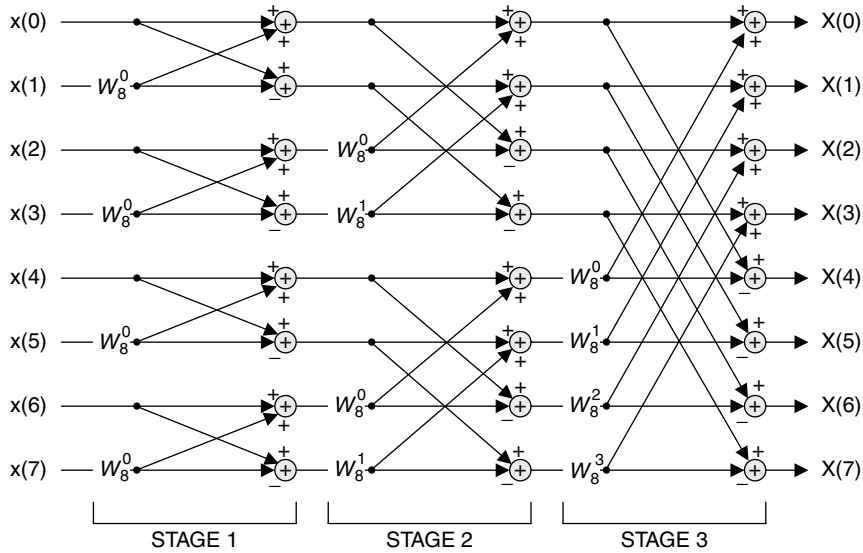

FIGURE 25.42 Eight-point, Radix-2 FFT

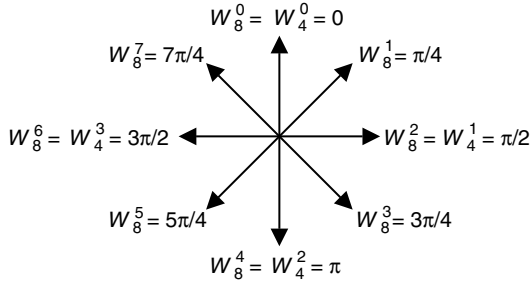

FIGURE 25.43 Phase shifts inferred by various twiddle factors

Since additions are performed in each stage, the magnitude of each output stage sample could be a factor of two or more greater than the input samples. If fixed-point computations are used, then this increased dynamic range results in a growth in the number of bits required to represent the values, and there needs to be a strategy to accommodate it.

There are several techniques generally used to handle this increased dynamic range in fixed-point FFTs. One scheme would be to ensure that the computation stages carry enough bits to accommodate the bit-growth. For example, in our 8-point FFT example, if we assume that the input samples are 12-bit complex numbers, and if we assume that the magnitudes of the complex numbers do not exceed 12 bits, then the final FFT outputs could grow 3 bits compared to the inputs, so the FFT computations could be performed with 15 bit or larger adders. This also means that the multipliers would have to handle the larger number of bits on the inputs. This method could get unwieldy for large FFTs.

Another technique is to automatically scale the outputs of each stage by a factor of 0.5, which would not allow the outputs to grow. Unfortunately, this would also limit any processing gain that the FFT might offer.

A third method, called *block floating point,* checks the magnitudes of all the outputs after each stage is computed and provides a single exponent for all output values. If any of the outputs have overflowed or come near to overflowing, then all of the outputs are scaled by a factor of 0.5, and the common exponent is incremented by 1. Enough bits have to be provided in the final mantissa to accommodate the dynamic range growth. This technique is popular because it only scales the output values when absolutely necessary.

25.5 DESIGN CONSIDERATIONS

This section addresses topics that need to be considered in the design of radar DSP systems as well as implementation alternatives.

Timing Dependencies. In coherent radar systems, all local oscillators (LOs) and clocks that generate system timing are derived from a single reference oscillator. However, this fact alone does not ensure that the transmitted waveform starts at the same RF phase on every pulse, which is a requirement for coherent systems.

Consider a system with a 5-MHz reference oscillator, from which is derived a 75 MHz IF center frequency (on transmit and receive) and a complex sample rate of 30 MHz. A rule of thumb is that the clock used to produce the pulse repetition interval (PRI) needs to be a common denominator of the IF center frequencies on transmit and receive and the complex sample frequency in order to assure pulse-to-pulse phase coherency. For this example, with an IF center frequency of 75 MHz and a complex sample rate of 30 MHz, allowable PRI clock frequencies would include 15 MHz and 5 MHz.

Hardware Implementation Technology. In the past, implementing a real-time radar digital signal processor typically required the design of a custom computing machine, using thousands of high performance integrated circuits (ICs). These machines were very difficult to design, develop, and modify. Digital technology has advanced to the point where several implementation alternatives exist that make the processor more programmable and, hence, easier to design and change.

Parallel General-purpose Computers. This architecture employs multiple general-purpose processors that are connected via high-speed communication networks. Included in this class are high-end servers and embedded processor architectures. Servers are typically homogeneous processors, where all of the processing nodes are identical and are connected by a very high-performance data bus architecture. Embedded processor architectures are typically composed of single-board computers (blades) that contain multiple general-purpose processors and plug into a standard backplane architecture, such as VME. This configuration offers the flexibility of supporting a heterogeneous architecture, where a variety of different processing blades or interface boards can be plugged into the standard backplane to configure a total system. At this writing, backplanes are migrating from parallel architectures, where data is typically passed as 32- or 64-bit words, to serial data links, which pass single bits at very high clock rates (currently in excess of 3 gigabits per second (Gbps)). These serial data links are typically point-to-point connections. In order to communicate with multiple boards, the serial links from each board go to a high-speed switch board that connects the appropriate source and destination serial links together to form a *serial fabric.* Examples of popular serial fabric backplanes at this writing include VXS, VPX, and ATCA. It is apparent that high-speed serial links will be the primary communication mechanism for multiprocessor machines into the future, with ever-increasing data bandwidths.

These parallel processor architectures offer the benefit of being programmable using high-level languages, such as C and C++. A related advantage is that programmers can design the system without knowing the intimate details of the hardware. Also, the software developed to implement the system can typically be moved relatively easily to a new hardware architecture as part of a technology refresh cycle.

On the negative side, these systems can be difficult to program to support real-time signal processing. The required operations need to be split up appropriately among the available processors, and the results need to be properly merged to form the final result. A major challenge in these applications is to support the processing *latency* requirements of the system, which defines the maximum length of time allowed to produce a result. The latency of a processor is defined as the amount of time required to observe the effect of a change at a processor's input on its output. Achieving latency goals often requires assigning smaller pieces of the workload to individual processors, leading to more processors and a more expensive system. Another challenge facing

these systems in a radar application is reset time. In a military application, when a system needs to be reset in order to fix a problem, the system needs to come back to full operation in a very short period of time. These multiprocessor systems typically take a long time to reboot from a central program store and, hence, have difficulty meeting reset requirements. Developing techniques to address these deficiencies is an active area of research. Finally, these processors are generally used for non-real-time or near-real-time data processing, as in target tracking and display processing. Since the 1990s, they have started to be applied to real-time signal processing applications. Although they might be cost-effective for relatively narrowband systems, their use in wideband DSP systems in the early 21st century is typically prohibitively expensive due to the large number of processors required. This situation should improve over time as faster and faster processors become available.

Custom-designed Hardware. Through the 1990s, real-time radar DSP systems were built using discrete logic. These systems were very difficult to develop and modify, but in order to achieve the required system performance, it was the only option available. Many systems were built using Application-Specific Integrated Circuits (ASICs), which are custom devices designed to perform a particular function. The use of ASICs allowed DSP systems to become very small with high performance. However, they were (and still are) difficult and expensive to develop, often requiring several design iterations before the device was fully operational. If an ASIC-based system needs to be modified, the ASICs need to be redesigned, incurring significant expense. Typically, the use of ASICs makes sense if tens or hundreds of thousands of units are to be sold, so that the development costs can be amortized over the life of the unit. This is rarely the case for radar systems. However, many ASICs have been developed to support the communication industry, such as digital up- and downconverters, which can be utilized in radar systems.

The introduction of the Field Programmable Gate Array (FPGA) in the 1980s heralded a revolution in the way real-time DSP systems were designed. FPGAs are integrated circuits that consist of a large array of configurable logic elements that are connected by a programmable interconnect structure. At the time of this writing, FPGAs can also incorporate hundreds of multipliers that can be clocked at rates up to a half billion operations per second, and memory blocks, microprocessors, and serial communication links that can support multigigabit-per-second data transfers. Circuits are typically designed using a hardware description language (HDL), such as VHDL (VHSIC Hardware Description Language) or Verilog. Software tools convert this high-level description of the processor to a file that is sent to the device to tell it how to configure itself. High-performance FPGAs store their configuration in volatile memory, which loses its contents when powered down, making the devices infinitely reprogrammable.

FPGAs allow the designer to fabricate complex signal processing architectures very efficiently. In typical large applications, FPGA-based processors can be a factor of ten (or more) smaller and less costly than systems based on general-purpose processors. This is due to the fact that most microprocessors only have one or very few processing elements, whereas FPGAs have an enormous number of programmable logic elements and multipliers. For example, to implement a 16-tap FIR filter in a microprocessor with a single multiplier and accumulator, it would take 16 clock cycles to perform the multiplications. In an FPGA, we could assign 16 multipliers and 16 accumulators to the task, and the filter could be performed in one clock cycle.

In order to use an FPGA most efficiently, we have to take advantage of all of the resources it offers. These include not only the large numbers of logic elements, multipliers, and memory blocks, but also the rate at which the components can be clocked. In the previous example, assume that the data sample rate is 1 MHz and also assume that the multipliers and logic can be clocked at 500 MHz. If we simply assign one multiplier to each coefficient, we would use 16 multipliers clocking at 500 MHz. Since the data rate is only 1 MHz, each multiplier would only perform one significant multiplication every microsecond and then be idle for the other 499 clocks in the microsecond, which is very inefficient. It would be much more efficient, in this case, to use one multiplier to perform as many products as possible. This technique, called *time-domain multiplexing,* requires additional logic to control the system and provide the correct operands to the multiplier at the right time. Since an FPGA can incorporate hundreds of multipliers, one can appreciate the power of this technique.

On the negative side, utilizing an FPGA to its best advantage typically requires the designer to have a thorough understanding of the resources available in the device. This typically makes efficient FPGA-based systems harder to design than systems based on general-purpose processors, where a detailed understanding of the processor architecture is not necessarily required. Also, FPGA designs tend to be aimed at a particular family of devices and take full advantage of the resources provided by that family. Hardware vendors are constantly introducing new products, invariably incorporating new and improved capabilities. Over time, the older devices become obsolete and need to be replaced during a *technology refresh* cycle. When a technology refresh occurs several years down the road, typically the available resources in the latest FPGAs have changed or a totally different device family is used, which probably requires a redesign. On the other hand, software developed for general-purpose processors may only need to be recompiled in order to move it to a new processor. Tools currently exist that synthesize C or Matlab code into an FPGA design, but these tools are typically not very efficient. The evolution of design tools for FPGAs to address these problems is an area of much research and development.

Hybrid Processors. Although it would be very desirable to simply write C code to implement a complex radar signal processor, the reality in the early 21st century is that, for many systems, implementing such a system would be prohibitively expensive or inflict major performance degradation. Although the steady increase in processor throughput may someday come to the rescue, the reality at this writing is that high-performance radar signal processors are usually a hybrid of application-specific and programmable processors. Dedicated processors, such as FPGAs or ASICs, are typically used in the high-speed front end of radar signal processors, performing demanding functions such as digital downconversion and pulse compression, followed by programmable processors in the rear, performing the lower-speed tasks such as detection processing. The location of the line that separates the two domains is application-dependent, but over time, it is constantly moving toward the front end of the system.

25.6 SUMMARY

The purpose of this chapter was to provide an overview of how digital signal processing has transformed radar system design and to give some insight into the techniques and tradeoffs that a designer has to consider. With manufacturers continually producing

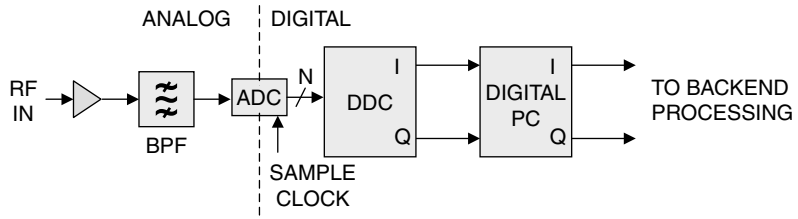

FIGURE 25.44 Direct-sampling radar digital receiver

faster and more powerful ADCs, DSP devices, and general-purpose processors, more and more of the radar system front end will move from analog to digital designs. For example, Figure 25.2 shows a typical digital receiver for a radar front end, which requires two stages of analog downconversion to bring the RF signal down to an IF that can be sampled by an ADC. This is required because of the characteristics of the ADC, which typically has poorer signal-to-noise ratio (SNR) and spur-free dynamic range (SFDR) when the input analog signal is too high, as would be the case if it were presented with the RF or high-IF signal directly. However, when faster ADCs become available, which can accommodate higher analog input frequencies while providing adequate SNR and SFDR, systems will be designed that sample the RF directly, as shown in Figure 25.44. At this writing, ADC technology allows direct sampling systems with respectable performance to be designed for radars in the HF and VHF bands. Doubtless, future components will extend this performance to higher RF frequencies.

ACKNOWLEDGMENTS

The authors would like to acknowledge the efforts of and extend their sincere gratitude to several individuals who helped them immensely in the preparation of this chapter. First, to Mr. Gregory Tavik of NRL for his thorough review of this chapter and the many excellent comments he made. Next, to Dr. Fred Harris of San Diego State University and Mr. Richard Lyons, who graciously reviewed sections of the chapter and offered several suggestions, all of which were incorporated.

REFERENCES

1. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, 2nd Ed., Englewood Cliffs, NJ: Prentice-Hall, 1989.
2. R. G. Lyons, *Understanding Digital Signal Processing*, 2nd Ed., Upper Saddle River, NJ: Prentice Hall, 2004.
3. J. O. Coleman, "Multi-rate DSP before discrete-time signals and systems," presented at First IEEE Workshop on Signal Processing Education (SPE 2000), Hunt, TX, October 2000.
4. W. M. Waters and B. R. Jarrett, "Bandpass signal sampling and coherent detection," *IEEE Trans. On Aerospace Electronic Systems,* vol. AES-18, no. 4, pp. 731–736, November 1982.
5. D. P. Scholnik and J. O. Coleman, "Integrated I-Q demodulation, matched filtering, and symbol-rate sampling using minimum-rate IF sampling," in *Proc. of the 1997 Symposium on Wireless Personal Communication*, Blacksburg, VA, June 1997.

6. B. Brannon and A. Barlow, "Aperture uncertainty and ADC system performance," Analog Devices Application Note AN-501, Rev. A, March 2006.
7. J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. on Electronic Computers*, vol. EC-8, pp. 330–334, 1959.
8. R. Andraka, "A survey of CORDIC algorithms for FPGA-based computers," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 1998, pp. 191–200.
9. E. B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP-29(2), pp. 155–162, April 1981.
10. F. J. Harris, *Multirate Signal Processing for Communication Systems,* Upper Saddle River, NJ: Prentice Hall, 2004.
11. F. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proc. IEEE*, vol. 66, no. 1, January 1978, pp. 51–83.
12. J. Cooley and J. Tukey, "An Algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, April 1965.