Chapter

16

Practical Considerations

All of the digital filter designs presented up until now have been based on infinite-precision mathematics. That is, we have assumed that all of the signal samples, filter coefficients, and results of mathematical computations are represented exactly or with *infinite precision*. In most cases we have used the **double** data type in C to approximate such precision. In Think C for the Apple Macintosh, the **double** data type has a 64-bit mantissa that provides approximately 19 decimal digits of precision. In Turbo C for the PC, the **double** data type has a 52-bit mantissa that provides approximately 15 decimal digits of precision. For most practical situations, either 15 or 19 digits of precision is a reasonable approximation to infinite precision. Furthermore, the **double** type is a floating-point format and thus provides good dynamic range in addition to high precision.

Although floating-point formats are used in some digital filters, cost and speed considerations will often dictate the use of fixed-point formats having a relatively short word length. Such formats will force some precision to be lost in representations of the signal samples, filter coefficients, and computation results. A digital filter designed under the infinite-precision assumption will not perform up to design expectations if implemented with short-word-length, fixed-point arithmetic. In many cases, the degradations can be so severe as to make the filter unuseable. This chapter examines the various types of degradations caused by finite-precision implementations and explores what can be done to achieve acceptable filter performance in spite of the degradations.

16.1 Binary Representation of Numeric Values

Fixed-point formats

Binary fixed-point representation of numbers enjoys widespread use in digital signal processing applications where there is usually some control over the range of values that must be represented. Typically, all of the coefficients $h[n]$ for a digital filter will be scaled such that

$$|h[n]| \leq 1.0 \qquad \text{for } n = 1, 2, \ldots, N \tag{16.1}$$

Once scaled in this way, each coefficient can be expressed as

$$h = b_0 2^0 + b_1 2^{-1} + b_2 2^{-2} + \cdots \tag{16.2}$$

where each of the b_n is a single bit; that is, $b_n \in \{0, 1\}$. If we limit our representation to a length of $L + 1$ bits, the coefficients can be represented as a fixed-point binary number of the form shown in Fig. 16.1. As shown in the figure, a small triangle is often used to represent the binary point so that it cannot be easily confused with a decimal point. The expansion of Eq. (16.2) can then be written as

$$h = \sum_{k=0}^{L} b_k 2^{-k} \tag{16.3}$$

The bit shown to the left of the binary point in Fig. 16.1 is necessary to represent coefficients for which the equality in (16.1) holds, but its presence complicates the implementation of arithmetic operations. If we eliminate the need to exactly represent coefficients that equal unity, we can use the fixed-point fractional format shown in Fig. 16.2. Using this scheme, some values are easy to write:

$$\tfrac{1}{2} = {}_{\Delta}1000$$

$$\tfrac{3}{8} = {}_{\Delta}01100$$

$$\tfrac{5}{64} = {}_{\Delta}000101$$

Some other values are not so easy. Consider the case of $\tfrac{1}{10}$, which expands as

$$\begin{aligned}\tfrac{1}{10} &= 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \cdots \\ &= \sum_{k=1}^{\infty} (2^{-4k} + 2^{-4k-1})\end{aligned}$$

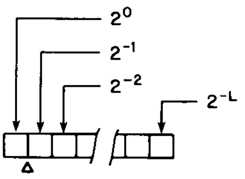

Figure 16.1 Fixed-point binary number format.

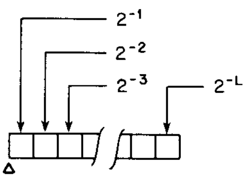

Figure 16.2 Alternative fixed-point binary number format.

The corresponding fixed-point binary representation is a repeating fraction given by

$$1/10 = {}_{\Delta}000\overline{1100}11 \cdots$$

If we are limited to a 16-bit fixed-point binary representation, we can truncate the fraction after 16 bits to obtain

$$1/10 \cong {}_{\Delta}0001100110011001$$

The actual value of this 16-bit representation is

$$2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} = \frac{6553}{65{,}536} \cong 0.099990845$$

Thus the value represented in 16 bits is too small by approximately 9.155×10^{-6}.

Instead of truncating, we could use a rounding approach. Rounding a binary value is easy—just add 1 to the first (leftmost) bit that is not being retained in the rounded format. In the current example we add 1 to bit 16. This generates a carry into b_{15} which propagates into b_{14} to yield

$${}_{\Delta}0001100110011010 = \frac{6554}{65{,}536} \cong 0.100006104$$

This value is too big by approximately 6.1×10^{-6}.

In many DSP applications where design simplicity, low cost, or high speed is important, the word length may be significantly shorter than 16 bits, and the error introduced by either truncating or rounding the coefficients can be quite severe, as we will see in Sec. 16.2.

Floating-point formats

A fixed-point fractional format has little use in a general-purpose computer where there is little or no a priori control over the range of values that may need to be represented. Clearly, any time a value equals or exceeds 1.0, it cannot be represented in the format of Fig. 16.2. Floating-point formats remove this limitation by effectively allowing the binary point to shift position as needed. For floating-point representations, a number is typically expanded in the form

$$h = 2^a \sum_{k=0}^{L} b_k 2^{-k}$$

In Think C for the Macintosh, a floating-point value has the form shown in Fig. 16.3. The fields denoted i and f contain a fixed-point value of the form shown in Fig. 16.1 where the binary point is assumed to lie between i and the most significant bit of f. This fixed-point value is referred to as the *mantissa.*

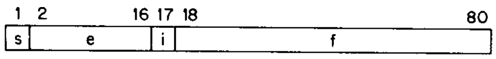

Figure 16.3 Floating-point binary number format used in Think C for the Macintosh.

If the bits in field f are designated from left to right as $f_1, f_2, \ldots, f_{63}$, the value of the mantissa is given by

$$m = \mathrm{i} + \sum_{k=1}^{63} f_k 2^{-k}$$

The field denoted as e is a 15-bit integer value used to indicate the power of 2 by which the numerator must be multiplied in order to obtain the value being represented. This can be a positive or negative power of 2, but rather than using a sign in conjunction with the exponent, most floating-point formats use an offset. A 15-bit binary field can have values ranging from 0 to 32,767. Values from 0 to 16,382 are interpreted as negative powers of 2, and values from 16,384 to 32,766 are interpreted as positive powers of 2. The value 16,383 is interpreted as $2^0 = 1$, and the value 32,767 is reserved for representing infinity and specialized values called **NaN** (*not-a-number*). The sign bit denoted by s is the sign of the overall number. Thus the value represented by a floating-point number in the format of Fig. 16.3 can be obtained as

$$v = (-1)^s \, 2^{(e - 16{,}383)} \left(i + \sum_{k=1}^{63} f_k 2^{-k} \right)$$

provided $e \neq 32{,}767$.

Suppose we wish to represent $\frac{1}{10}$ in the floating-point format of Fig. 16.3. One way to accomplish this is to set the mantissa equal to a 64-bit fixed-point representation of $\frac{1}{10}$ and set $e = 16{,}383$ to indicate a multiplier of unity. Using the hexadecimal notation discussed previously, we can write the results of such an approach as

$$s = 0$$

$$e = 0\mathrm{x}3fff$$

$$i = 0$$

$$f = 0\mathrm{x}0\mathrm{cccccccccccccccc}$$

With the various fields packed together, the resulting 80-bit floating-point representation of $\frac{1}{10}$ is $W = $ 0x3fff0cccccccccccccccc. Slightly more precision can be squeezed into the representation if we shift f 4 places to the left and modify e to indicate multiplication by 2^{-4}. Such an approach yields

$$W = 0\mathrm{x}3\mathrm{ffbcccccccccccccccccc}$$

Numbers greater than 1.0 present no problem for this format. The value 57 is represented as

$$s = 0$$

$$e = 0\text{x}4004 \qquad \text{(that is, } 2^5\text{)}$$

$$i = 1$$

$$f = 0\text{x}6400000000000000$$

$$W = 0\text{x}4004\text{e}400000000000000$$

In other words, this representation stores 57 by making use of the fact

$$57 = 2^5(2^0 + 2^{-1} + 2^{-2} + 2^{-5})$$

16.2 Quantized Coefficients

When the coefficients of a digital filter are quantized, the filter becomes a different filter. The resulting filter is still a discrete-time linear time-invariant system—it's just not the system we set out to design. Consider the 21-tap lowpass filter using a von Hann window that was designed in Example 11.6. The coefficients of this filter are reproduced in Table 16.1. The values given in the table, having 15 decimal digits in the fractional part, will be used as the baseline approximation to the coefficients' infinite-precision values. Let's force the coefficient values into a fixed-point fractional format having a 16-bit magnitude plus 1 sign bit. After truncating the bits in excess of 16, the coefficient values listed in Table 16.2 are obtained. The magnitude response of a filter using such coefficients is virtually identical to the response obtained using the floating-point coefficients of Table 16.1. If the coefficients are

TABLE 16.1 Coefficients for 21-tap Lowpass Filter Using a von Hann Window

n	$h[n]$
0, 20	0.000
1, 19	−0.000823149720361
2, 18	−0.002233281959082
3, 17	0.005508892585759
4, 16	0.017431813641454
5, 17	−0.000000000000050
6, 16	−0.049534952531101
7, 15	−0.049511869643024
8, 14	0.084615800641299
9, 13	0.295322344140975
10	0.40

TABLE 16.2 Truncated 16-bit Coefficients for 21-tap Lowpass Filter

n	Sign	Hex value	Decimal value
0, 20	+	0000	0.0
1, 19	−	0035	−0.000808715820312
2, 18	−	0092	−0.002227783203125
3, 17	+	0169	0.005508422851562
4, 16	+	0476	0.017425537109375
5, 15	+	0000	0.0
6, 14	−	0cae	−0.049530029296875
7, 13	−	0cac	−0.049499511718750
8, 12	+	15a9	0.084609985351562
9, 11	+	4b9a	0.295318603515625
10	+	6666	0.399993896484375

TABLE 16.3 Truncated 10-bit Coefficients for 21-tap Lowpass Filter

n	Sign	Hex value	Decimal value
0, 20	+	000	0.0
1, 19	−	000	0.0
2, 18	−	008	−0.001953125
3, 17	+	014	0.0048828125
4, 16	+	044	0.0166015625
5, 15	+	000	0.0
6, 14	−	0c8	−0.048828125
7, 13	−	0c8	−0.048828125
8, 12	+	158	0.083984375
9, 11	+	4b8	0.294921875
10	+	664	0.3994140625

further truncated to 14- or 12-bit magnitudes, slight degradations in stopband attenuation can be observed.

The degradations in filter response are really quite significant for the 10-bit coefficients listed in Table 16.3. As shown in Fig. 16.4, the fourth sidelobe is narrowed, and the fifth sidelobe peaks at −50.7 dB—a value significantly worse than the −68.2 dB of the baseline case. The filter response for 8- and 6-bit coefficients are shown in Figs. 16.5 and 16.6, respectively.

16.3 Quantization Noise

The finite digital word lengths used to represent numeric values within a digital filter limit the precision of other quantities besides the filter coefficients. Each sample of the input and output, as well as all intermediate results of mathematical operations, must be represented with finite precision. As we saw in the previous section, the effects of coefficient quantization are straightforward and easy to characterize. The effects of signal quantization are somewhat different.

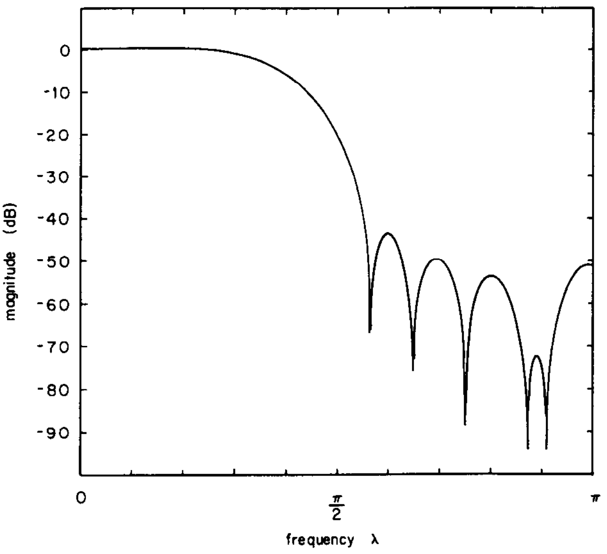

Figure 16.4 Magnitude response for a von Hann-windowed 21-tap lowpass filter with coefficients quantized to 10 bits plus sign.

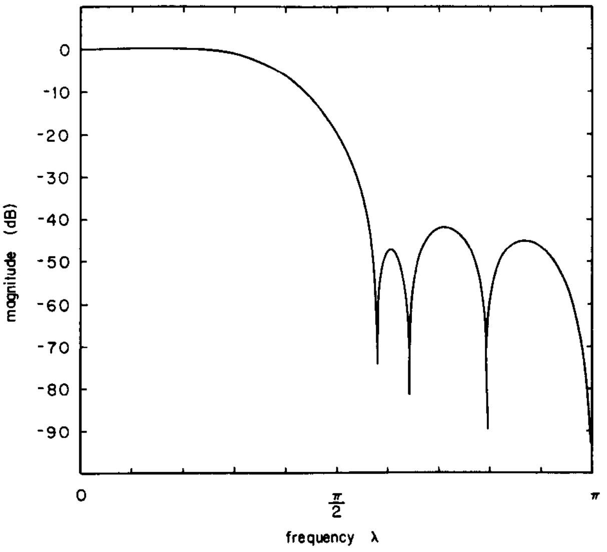

Figure 16.5 Magnitude response for a von Hann-windowed 21-tap lowpass filter with coefficients quantized to 8 bits plus sign.

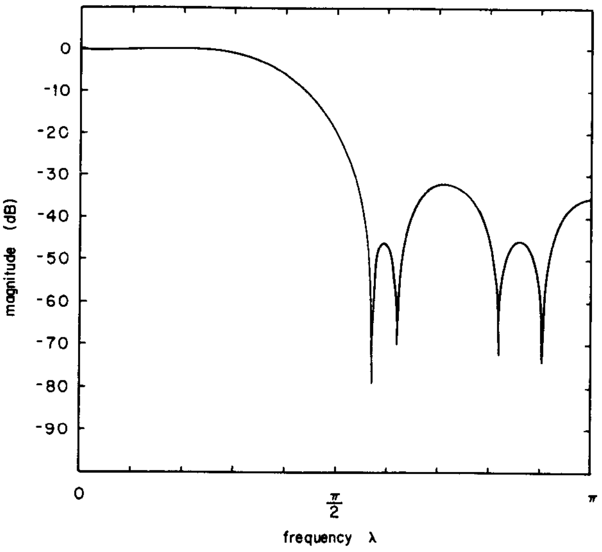

Figure 16.6 Magnitude response for a von Hann-windowed 21-tap lowpass filter with coefficients quantized to 6 bits plus sign.

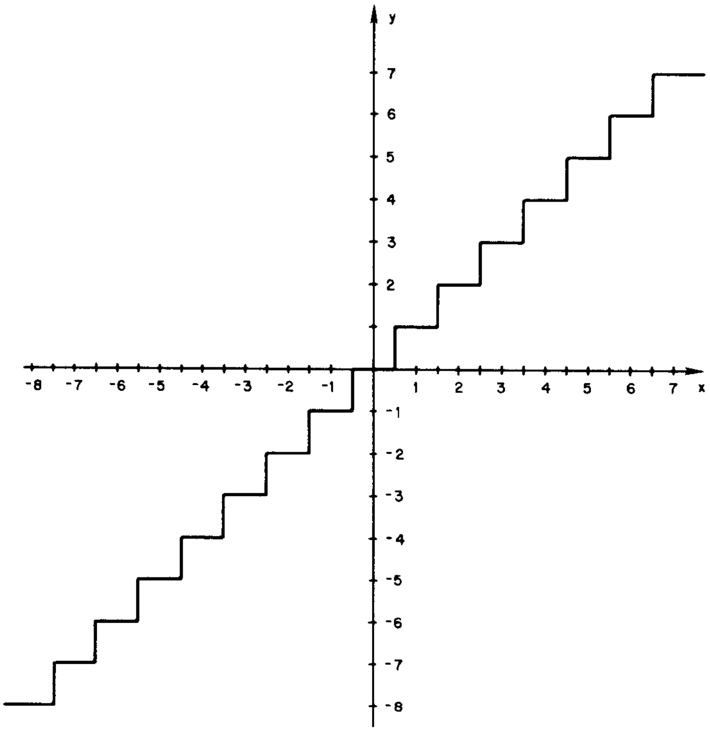

Figure 16.7 Typical transfer characteristic for a rounding quantizer.

Typically, an *analog-to-digital converter* (ADC) is used to sample and quantize an analog signal that can be thought of as a continuous amplitude function of continuous time. The ADC can be viewed as a sampler and quantizer in cascade. Sampling was discussed in Chap. 7, and in this section we examine the operation of quantization. The transfer characteristic of a typical quantizer is shown in Fig. 16.7. This particular quantizer *rounds* the analog value to the nearest "legal" quantized value. The resulting sequence of quantized signal values $y[n]$ can be viewed as the sampled continuous-time signal $x[n]$ plus an error sequence $e[n]$ whose values are equal to the errors introduced by the quantizer:

$$y[n] = x[n] + e[n]$$

A typical discrete-time signal along with the corresponding quantized sequence and error sequence are shown in Fig. 16.8. Because the quantizer rounds to the nearest quantizer level, the magnitude of the error will never exceed $Q/2$, where Q is the increment between two consecutive legal quantizer output levels, that is,

$$\frac{-Q}{2} \le e(t) \le \frac{Q}{2} \qquad \text{for all } t$$

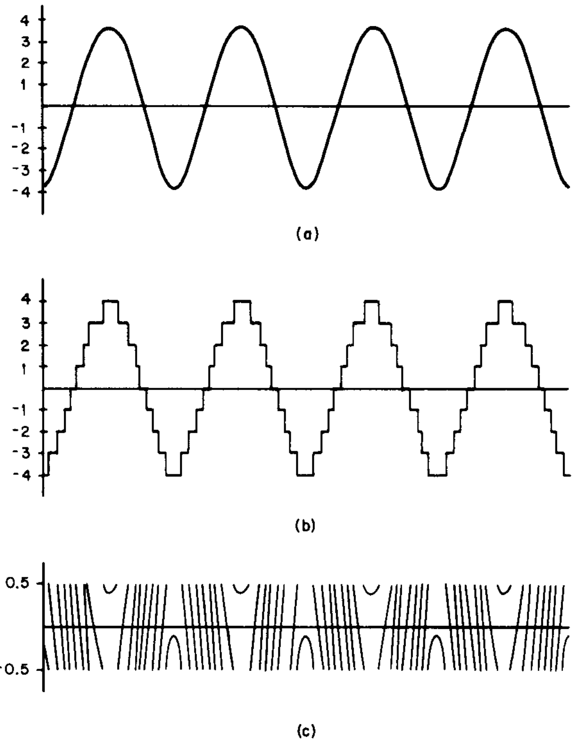

Figure 16.8 (*a*) Discrete-time continuous amplitude signal, (*b*) corresponding quantized signal, and (*c*) error sequence.

The error is usually assumed to be uniformly distributed between $-Q/2$ and $Q/2$ and consequently to have a mean and variance of 0 and $Q^2/12$, respectively. For most practical applications, this assumption is reasonable. The quantization interval Q can be related to the number of bits in the digital word. Assume a word length of $L+1$ bits with 1 bit used for the sign and L bits for the magnitude. For the fixed-point format of Fig. 16.2, the relationship between Q and L is then given by $Q = 2^{-L}$.

It is often useful to characterize the quantization noise by means of a *signal-to-noise ratio* (SNR). In order to accomplish this characterization, the following additional assumptions are usually made:

1. The error sequence is assumed to be a sample sequence of a stationary random proces; that is, the statistical properties of the error sequence do not change over time.
2. The error is a white-noise process; or equivalently, the error signal is uncorrelated.
3. The error sequence $e[n]$ is uncorrelated with the sequence of unquantized samples $x[n]$.

Based on these assumptions, the power of the quantization noise is equal to the error variance that was given previously as

$$\sigma_e^2 = \frac{Q^2}{12} = \frac{2^{-2L}}{12}$$

If we let σ_x^2 denote the signal power, then the SNR is given by

$$\frac{\sigma_x^2}{\sigma_e^2} = \frac{\sigma_x^2}{2^{-2L}/12} = (12 \cdot 2^{2L})\sigma_x^2$$

Expressed in decibels, this SNR is

$$\begin{aligned} 10 \log\left(\frac{\sigma_x^2}{\sigma_e^2}\right) &= 10 \log 12 + 20L \log 2 + 10 \log \sigma_x^2 \\ &= 10.792 + 6.021L + 10 \log \sigma_x^2 \qquad (16.4) \end{aligned}$$

The major insight to be gained from (16.4) is that the SNR improves by 6.02 dB for each bit added to the digital word format. We are not yet in a position to compute an SNR using Eq. (16.4), because the term σ_x^2 needs some further examination. How do we go about obtaining a value for σ_x^2? Whatever the value of σ_x^2 may be originally, we must realize that in practical systems, the input signal is subjected to some amplification prior to digitization. For a constant amplifier gain of A, the unquantized signal becomes $Ax[n]$, the signal power becomes $A^2\sigma_x^2$, and the corresponding SNR is given by

$$\text{SNR} = 10 \log\left(\frac{A^2\sigma_x^2}{\sigma_e^2}\right) = 10.792 + 6.021L + 10 \log(A^2\sigma_x^2) \qquad (16.5)$$

A general rule of thumb often used in practical DSP applications is to set A so that $A\sigma_x$ is equal to 25 percent of the ADC full-scale value. Since we have been treating full scale as being normalized to unity, this indicates a value of A such that

$$A\sigma_x = 0.25 \qquad \text{or} \qquad A = \frac{1}{4\sigma_x}$$

Substituting this value of A into (16.5) yields

$$\begin{aligned}\text{SNR} &= 10.79 + 6.02L + 10\log\left(\frac{1}{16}\right)\\ &= 6.02L - 1.249 \text{ dB}\end{aligned}$$

Using a value of $A = 1/(4\sigma_x)$ means that the ADC will introduce clipping any time the unquantized input signal exceeds $4\sigma_x$. Increasing A improves the SNR but decreases the *dynamic range*, that is, the range of signal values that can be accommodated without clipping. Thus, for a fixed word length, we can improve the SNR at the expense of degraded dynamic range. Conversely, by decreasing A, we could improve dynamic range at the expense of degraded SNR. The only way to simultaneously improve both dynamic range and quantization SNR is to increase the number of bits in the digital word length.