

Quantization

2.1 Quantization Process

Sampling and quantization are the necessary prerequisites for any digital signal processing operation on analog signals. A sampler and quantizer are shown in Fig. 2.1.1 [40–45]. The hold capacitor in the sampler holds each measured sample $x(nT)$ for at most T seconds during which time the A/D converter must convert it to a quantized sample, $x_Q(nT)$, which is representable by a finite number of bits, say B bits. The B -bit word is then shipped over to the digital signal processor.

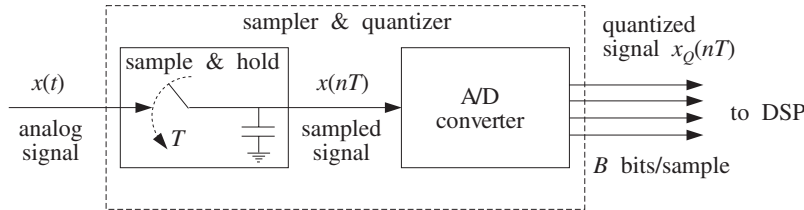


Fig. 2.1.1 Analog to digital conversion.

After digital processing, the resulting B -bit word is applied to a D/A converter which converts it back to analog format generating a staircase output. In practice, the sample/hold and ADC may be separate modules or may reside on board the same chip.

The quantized sample $x_Q(nT)$, being represented by B bits, can take only one of 2^B possible values. An A/D converter is characterized by a *full-scale range* R , which is divided equally (for a uniform quantizer) into 2^B *quantization levels*, as shown in Fig. 2.1.2. The spacing between levels, called the *quantization width* or the quantizer resolution, is given by:

$$Q = \frac{R}{2^B} \quad (2.1.1)$$

This equation can also be written in the form:

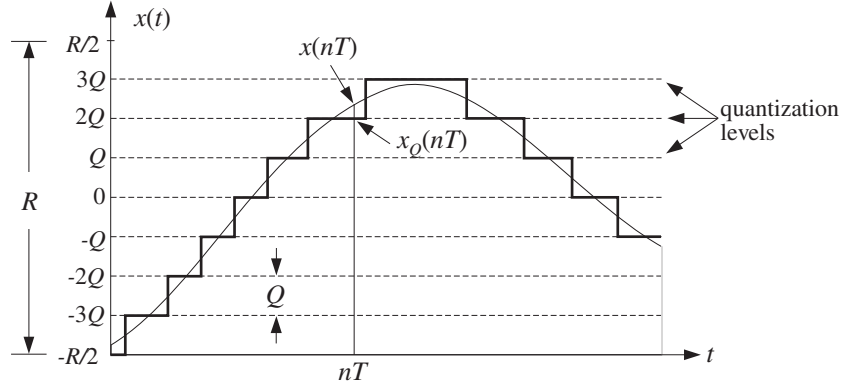


Fig. 2.1.2 Signal quantization.

$$\boxed{\frac{R}{Q} = 2^B} \quad (2.1.2)$$

which gives the number of quantization levels. Typical values of R in practice are between 1-10 volts. Figure 2.1.2 shows the case of $B = 3$ or $2^B = 8$ levels, and assumes a *bipolar* ADC for which the possible quantized values lie within the symmetric range:

$$-\frac{R}{2} \leq x_Q(nT) < \frac{R}{2}$$

For a *unipolar* ADC, we would have instead $0 \leq x_Q(nT) < R$. In practice, the input signal $x(t)$ must be preconditioned by analog means to lie within the full-scale range of the quantizer, that is, $-R/2 \leq x(t) < R/2$, before it is sent to the sampler and quantizer. The upper end, $R/2$, of the full-scale range is not realized as one of the levels; rather, the maximum level is $R/2 - Q$.

In Fig. 2.1.2, quantization of $x(t)$ was done by *rounding*, that is, replacing each value $x(t)$ by the value of the *nearest* quantization level. Quantization can also be done by *truncation* whereby each value is replaced by the value of the level below it. Rounding is preferred in practice because it produces a less biased quantized representation of the analog signal.

The *quantization error* is the error that results from using the quantized signal $x_Q(nT)$ instead of the true signal $x(nT)$, that is,[†]

$$e(nT) = x_Q(nT) - x(nT) \quad (2.1.3)$$

In general, the error in quantizing a number x that lies in $[-R/2, R/2)$ is:

$$e = x_Q - x$$

[†]A more natural definition would have been $e(nT) = x(nT) - x_Q(nT)$. The choice (2.1.3) is more convenient for making quantizer models.

where x_Q is the quantized value. If x lies between two levels, it will be rounded up or down depending on which is the closest level. If x lies in the upper (lower) half between the two levels, it will be rounded up (down). Thus, the error e can only take the values[†]

$$-\frac{Q}{2} \leq e \leq \frac{Q}{2} \quad (2.1.4)$$

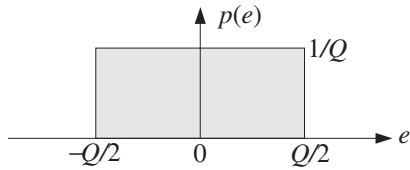
Therefore, the maximum error is $e_{\max} = Q/2$ in magnitude. This is an overestimate for the typical error that occurs. To obtain a more representative value for the average error, we consider the *mean* and *mean-square* values of e defined by:

$$\bar{e} = \frac{1}{Q} \int_{-Q/2}^{Q/2} e \, de = 0, \quad \text{and} \quad \overline{e^2} = \frac{1}{Q} \int_{-Q/2}^{Q/2} e^2 \, de = \frac{Q^2}{12} \quad (2.1.5)$$

The result $\bar{e} = 0$ states that on the average half of the values are rounded up and half down. Thus, \bar{e} cannot be used as a representative error. A more typical value is the *root-mean-square* (rms) error defined by:

$$e_{\text{rms}} = \sqrt{\overline{e^2}} = \frac{Q}{\sqrt{12}} \quad (2.1.6)$$

Equations (2.1.5) can be given a probabilistic interpretation by assuming that the quantization error e is a *random variable* which is distributed *uniformly* over the range (2.1.4), that is, having probability density:

$$p(e) = \begin{cases} \frac{1}{Q} & \text{if } -\frac{Q}{2} \leq e \leq \frac{Q}{2} \\ 0 & \text{otherwise} \end{cases}$$


The normalization $1/Q$ is needed to guarantee:

$$\int_{-Q/2}^{Q/2} p(e) \, de = 1$$

It follows that Eqs. (2.1.5) represent the *statistical expectations*:

$$E[e] = \int_{-Q/2}^{Q/2} e p(e) \, de \quad \text{and} \quad E[e^2] = \int_{-Q/2}^{Q/2} e^2 p(e) \, de$$

Thinking of R and Q as the ranges of the signal and quantization noise, the ratio in Eq. (2.1.2) is a *signal-to-noise ratio* (SNR). It can be expressed in dB:

$$20 \log_{10} \left(\frac{R}{Q} \right) = 20 \log_{10} (2^B) = B \cdot 20 \log_{10} 2, \quad \text{or,}$$

$$\boxed{SNR = 20 \log_{10} \left(\frac{R}{Q} \right) = 6B \text{ dB}} \quad (2.1.7)$$

[†]If the midpoint between levels is always rounded up, then we should have more strictly $-Q/2 < e \leq Q/2$.

which is referred to as the *6 dB per bit* rule. Eq. (2.1.7) is called the *dynamic range* of the quantizer. Equations (2.1.1) and (2.1.6) can be used to determine the wordlength B if the full-scale range and desired rms error are given.

Example 2.1.1: In a digital audio application, the signal is sampled at a rate of 44 kHz and each sample quantized using an A/D converter having a full-scale range of 10 volts. Determine the number of bits B if the rms quantization error must be kept below 50 microvolts. Then, determine the actual rms error and the bit rate in bits per second.

Solution: Write Eq. (2.1.6) in terms of B , $e_{\text{rms}} = Q/\sqrt{12} = R2^{-B}/\sqrt{12}$ and solve for B :

$$B = \log_2 \left[\frac{R}{e_{\text{rms}} \sqrt{12}} \right] = \log_2 \left[\frac{10}{50 \cdot 10^{-6} \sqrt{12}} \right] = 15.82$$

which is rounded to $B = 16$ bits, corresponding to $2^B = 65536$ quantization levels. With this value of B , we find $e_{\text{rms}} = R2^{-B}/\sqrt{12} = 44$ microvolts. The bit rate will be $Bf_s = 16 \cdot 44 = 704$ kbits/sec. This is a typical bit rate for CD players.

The dynamic range of the quantizer is $6B = 6 \cdot 16 = 96$ dB. Note that the dynamic range of the human ear is about 100 dB. Therefore, the quantization noise from 16-bit quantizers is about at the threshold of hearing. This is the reason why “CD quality” digital audio requires at least 16-bit quantization. \square

Example 2.1.2: By comparison, in digital speech processing the typical sampling rate is 8 kHz and the quantizer's wordlength 8 bits, corresponding to 256 levels. An 8-bit ADC with full-scale range of 10 volts, would generate an rms quantization noise $e_{\text{rms}} = R2^{-B}/\sqrt{12} = 11$ millivolts. The bit rate in this case is $Bf_s = 8 \cdot 8 = 64$ kbits/sec. \square

The probabilistic interpretation of the quantization noise is very useful for determining the effects of quantization as they propagate through the rest of the digital processing system. Writing Eq. (2.1.3) in the form[†]

$$x_Q(n) = x(n) + e(n) \quad (2.1.8)$$

we may think of the quantized signal $x_Q(n)$ as a noisy version of the original unquantized signal $x(n)$ to which a noise component $e(n)$ has been added. Such an additive noise model of a quantizer is shown in Fig. 2.1.3.

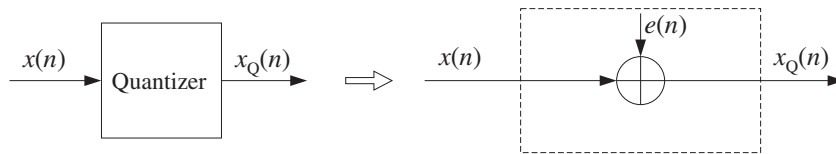


Fig. 2.1.3 Additive noise model of a quantizer.

In general, the statistical properties of the noise sequence $e(n)$ are very complicated [46–51,54]. However, for so-called *wide-amplitude wide-band* signals, that is, signals that vary through the entire full-scale range R crossing often all the quantization levels, the

[†]For simplicity, we denoted $x(nT)$ by $x(n)$, etc.

sequence $e(n)$ may be assumed to be a *stationary* zero-mean *white noise* sequence with *uniform* probability density over the range $[-Q/2, Q/2]$. Moreover, $e(n)$ is assumed to be *uncorrelated* with the signal $x(n)$. The average *power* or variance of $e(n)$ has already been computed above:

$$\sigma_e^2 = E[e^2(n)] = \frac{Q^2}{12} \quad (2.1.9)$$

The assumption that $e(n)$ is white noise means that it has a delta-function autocorrelation (see Appendix A.1):

$$R_{ee}(k) = E[e(n+k)e(n)] = \sigma_e^2 \delta(k) \quad (2.1.10)$$

for all lags k . Similarly, that it is uncorrelated with $x(n)$ means that it has zero cross-correlation:

$$R_{ex}(k) = E[e(n+k)x(n)] = 0 \quad (2.1.11)$$

for all k . Later on we will illustrate this statistical model for $e(n)$ with a simulation example and verify equations (2.1.9)–(2.1.11), as well as the uniform distribution for the density $p(e)$.

The model is not accurate for low-amplitude slowly varying signals. For example, a sinusoid that happens to lie exactly in the middle between two levels and has amplitude less than $Q/2$ will be quantized to be a square wave, with all the upper humps of the sinusoid being rounded up and all the lower ones rounded down. The resulting error $e(n)$ will be highly periodic, that is, correlated from sample to sample, and not resembling random white noise. It will also be highly correlated with input sinusoid $x(n)$.

In digital audio, quantization distortions arising from low-level signals are referred to as *granulation* noise and correspond to unpleasant sounds. They can be virtually eliminated by the use of *dither*, which is low-level noise added to the signal before quantization.

The beneficial effect of dithering is to make the overall quantization error behave as a white noise signal, which is *perceptually* much more preferable and acceptable than the gross granulation distortions of the undithered signal [52–69]. On the negative side, dithering reduces the signal-to-noise ratio somewhat—between 3 to 6 dB depending on the type of dither used. Dither will be discussed in Section 2.5.

2.2 Oversampling and Noise Shaping*

In the frequency domain, the assumption that $e(n)$ is a white noise sequence means that it has a flat spectrum. More precisely, the total average power σ_e^2 of $e(n)$ is distributed *equally* over the Nyquist interval $[-f_s/2, f_s/2]$, as shown in Fig. 2.2.1.

Thus, the power per unit frequency interval or *power spectral density* of $e(n)$ will be[†]

[†]In units of digital frequency $\omega = 2\pi f/f_s$, it is $S_{ee}(\omega) = \sigma_e^2/2\pi$.

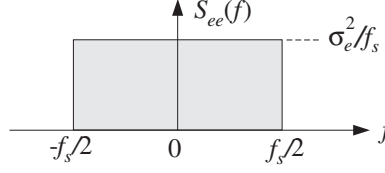


Fig. 2.2.1 Power spectrum of white quantization noise.

$$S_{ee}(f) = \frac{\sigma_e^2}{f_s}, \quad \text{for } -\frac{f_s}{2} \leq f \leq \frac{f_s}{2} \quad (2.2.1)$$

and it is periodic outside the interval with period f_s . The noise power within any Nyquist subinterval $[f_a, f_b]$ of width $\Delta f = f_b - f_a$ is given by

$$S_{ee}(f) \Delta f = \sigma_e^2 \frac{\Delta f}{f_s} = \sigma_e^2 \frac{f_b - f_a}{f_s}$$

As expected, the total power over the entire interval $\Delta f = f_s$ will be

$$\frac{\sigma_e^2}{f_s} f_s = \sigma_e^2$$

Noise shaping quantizers reshape the spectrum of the quantization noise into a more convenient shape. This is accomplished by filtering the white noise sequence $e(n)$ by a noise shaping filter $H_{NS}(f)$. The *equivalent noise model* for the quantization process is shown in Fig. 2.2.2. The corresponding quantization equation, replacing Eq. (2.1.8), becomes:

$$x_Q(n) = x(n) + \varepsilon(n) \quad (2.2.2)$$

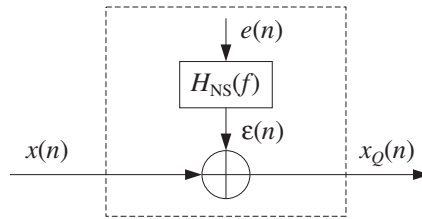


Fig. 2.2.2 Model of noise shaping quantizer.

where $\varepsilon(n)$ denotes the filtered noise. The sequence $\varepsilon(n)$ is no longer white. Its power spectral density is not flat, but acquires the shape of the filter $H_{NS}(f)$:

$$S_{\varepsilon\varepsilon}(f) = |H_{NS}(f)|^2 S_{ee}(f) = \frac{\sigma_e^2}{f_s} |H_{NS}(f)|^2 \quad (2.2.3)$$

The noise power within a given subinterval $[f_a, f_b]$ is obtained by integrating $S_{\varepsilon\varepsilon}(f)$ over that subinterval:

$$\text{Power in } [f_a, f_b] = \int_{f_a}^{f_b} S_{\varepsilon\varepsilon}(f) df = \frac{\sigma_e^2}{f_s} \int_{f_a}^{f_b} |H_{\text{NS}}(f)|^2 df \quad (2.2.4)$$

Noise shaping quantizers are implemented by the so-called *delta-sigma* A/D converters [276], which are increasingly being used in commercial products such as digital audio sampling systems for hard disk or digital tape recording. We will discuss implementation details in Chapter 12. Here, we give a broad overview of the advantages of such quantizers.

The concepts of sampling and quantization are independent of each other. The first corresponds to the quantization of the time axis and the second to the quantization of the amplitude axis. Nevertheless, it is possible to trade off one for the other. Oversampling was mentioned earlier as a technique to alleviate the need for high quality prefilters and postfilters. It can also be used to trade off bits for samples. In other words, if we sample at a higher rate, we can use a coarser quantizer. Each sample will be less accurate, but there will be many more of them and their effect will average out to recover the lost accuracy.

The idea is similar to performing multiple measurements of a quantity, say x . Let σ_x^2 be the mean-square error in a single measurement. If L independent measurements of x are made, it follows from the law of large numbers that the measurement error will be reduced to σ_x^2/L , improving the accuracy of measurement. Similarly, if σ_x^2 is increased, making each individual measurement worse, one can maintain the *same* level of quality as long as the number of measurements L is also increased commensurately to keep the ratio σ_x^2/L constant.

Consider two cases, one with sampling rate f_s and B bits per sample, and the other with higher sampling rate f'_s and B' bits per sample. The quantity:

$$L = \frac{f'_s}{f_s}$$

is called the *oversampling ratio* and is usually an integer. We would like to show that B' can be less than B and still maintain the same level of quality. Assuming the same full-scale range R for the two quantizers, we have the following quantization widths:

$$Q = R2^{-B}, \quad Q' = R2^{-B'}$$

and quantization noise powers:

$$\sigma_e^2 = \frac{Q^2}{12}, \quad \sigma_e'^2 = \frac{Q'^2}{12}$$

To maintain the same quality in the two cases, we require that the power spectral densities remain the same, that is, using Eq. (2.2.1):

$$\frac{\sigma_e^2}{f_s} = \frac{\sigma_e'^2}{f'_s}$$

which can be rewritten as

$$\sigma_e^2 = f_s \frac{\sigma_e'^2}{f_s'} = \frac{\sigma_e'^2}{L} \quad (2.2.5)$$

Thus, the total quantization power σ_e^2 is less than $\sigma_e'^2$ by a factor of L , making B greater than B' . The meaning of this result is shown pictorially in Fig. 2.2.3. If sampling is done at the higher rate f_s' , then the total power $\sigma_e'^2$ of the quantization noise is spread evenly over the f_s' Nyquist interval.

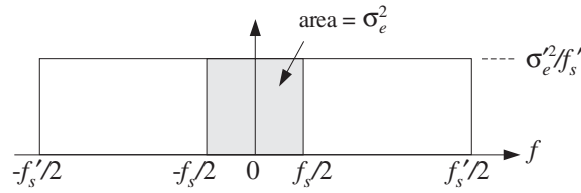


Fig. 2.2.3 Oversampled quantization noise power, without noise shaping.

The shaded area in Fig. 2.2.3 gives the *proportion* of the $\sigma_e'^2$ power that lies within the smaller f_s interval. Solving Eq. (2.2.5) for L and expressing it in terms of the difference $\Delta B = B - B'$, we find:

$$L = \frac{\sigma_e'^2}{\sigma_e^2} = 2^{2(B-B')} = 2^{2\Delta B}$$

or, equivalently

$$\Delta B = 0.5 \log_2 L \quad (2.2.6)$$

that is, a saving of half a bit per doubling of L . This is too small to be useful. For example, in order to reduce a 16-bit quantizer for digital audio to a 1-bit quantizer, that is, $\Delta B = 15$, one would need the unreasonable oversampling ratio of $L = 2^{30}$.

A *noise shaping* quantizer operating at the higher rate f_s' can reshape the flat noise spectrum so that most of the power is squeezed out of the f_s Nyquist interval and moved into the outside of that interval. Fig. 2.2.4 shows the power spectrum of such a quantizer.

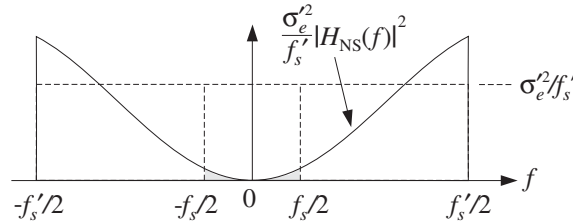


Fig. 2.2.4 Spectrum of oversampling noise shaping quantizer.

The total quantization noise power that resides within the original f_s Nyquist interval is the shaded area in this figure. It can be calculated by integrating Eq. (2.2.4) over $[-f_s/2, f_s/2]$:

$$\sigma_e^2 = \frac{\sigma_e'^2}{f_s'} \int_{-f_s'/2}^{f_s'/2} |H_{\text{NS}}(f)|^2 df \quad (2.2.7)$$

Note that it reduces to Eq. (2.2.5) if there is no noise shaping, that is, $H_{\text{NS}}(f) = 1$. We will see in Section 12.7 that a typical p th order noise shaping filter operating at the high rate f_s' has magnitude response:

$$|H_{\text{NS}}(f)|^2 = \left| 2 \sin \left(\frac{\pi f}{f_s'} \right) \right|^{2p}, \quad \text{for } -\frac{f_s'}{2} \leq f \leq \frac{f_s'}{2} \quad (2.2.8)$$

For small frequencies f , we may use the approximation, $\sin x \simeq x$, to obtain:

$$|H_{\text{NS}}(f)|^2 = \left(\frac{2\pi f}{f_s'} \right)^{2p}, \quad \text{for } |f| \ll f_s'/2 \quad (2.2.9)$$

Assuming a large oversampling ratio L , we will have $f_s \ll f_s'$, and therefore, we can use the approximation (2.2.9) in the integrand of Eq. (2.2.7). This gives:

$$\begin{aligned} \sigma_e^2 &= \frac{\sigma_e'^2}{f_s'} \int_{-f_s'/2}^{f_s'/2} \left(\frac{2\pi f}{f_s'} \right)^{2p} df = \sigma_e'^2 \frac{\pi^{2p}}{2p+1} \left(\frac{f_s'}{f_s} \right)^{2p+1} \\ &= \sigma_e'^2 \frac{\pi^{2p}}{2p+1} \frac{1}{L^{2p+1}} \end{aligned}$$

Using $\sigma_e^2 / \sigma_e'^2 = 2^{-2(B-B')} = 2^{-2\Delta B}$, we obtain:

$$2^{-2\Delta B} = \frac{\pi^{2p}}{2p+1} \frac{1}{L^{2p+1}}$$

Solving for ΔB , we find the gain in bits:

$$\Delta B = (p + 0.5) \log_2 L - 0.5 \log_2 \left(\frac{\pi^{2p}}{2p+1} \right) \quad (2.2.10)$$

Now, the savings are $(p + 0.5)$ bits per doubling of L . Note that Eq. (2.2.10) reduces to Eq. (2.2.6) if there is no noise shaping, that is, $p = 0$. Practical values for the order p are at present $p = 1, 2, 3$, with $p = 4, 5$ becoming available. Table 2.2.1 compares the gain in bits ΔB versus oversampling ratio L for various quantizer orders.

The first CD player built by Philips used a first-order noise shaper with 4-times oversampling, that is, $p = 1$, $L = 4$, which according to the table, achieves a savings of $\Delta B = 2.1$ bits. Because of that, the Philips CD player used a 14-bit, instead of a 16-bit, D/A converter at the analog reconstructing stage [279].

We also see from the table that to achieve 16-bit CD-quality resolution using 1-bit quantizers, that is, $\Delta B = 15$, we may use a second-order 128-times oversampling quantizer. For digital audio rates $f_s = 44.1$ kHz, this would imply oversampling at $f_s' = Lf_s = 5.6$ MHz, which is feasible with the present state of the art. Alternatively, we may use third-order noise shaping with 64-times oversampling.

An overall DSP system that uses oversampling quantizers with noise shaping is shown in Fig. 2.2.5. Sampling and reconstruction are done at the fast rate f_s' and at

| p | L | 4 | 8 | 16 | 32 | 64 | 128 |
|-----|----------------------------------|-----|------|------|------|------|------|
| 0 | $\Delta B = 0.5 \log_2 L$ | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | 3.5 |
| 1 | $\Delta B = 1.5 \log_2 L - 0.86$ | 2.1 | 3.6 | 5.1 | 6.6 | 8.1 | 9.6 |
| 2 | $\Delta B = 2.5 \log_2 L - 2.14$ | 2.9 | 5.4 | 7.9 | 10.4 | 12.9 | 15.4 |
| 3 | $\Delta B = 3.5 \log_2 L - 3.55$ | 3.5 | 7.0 | 10.5 | 14.0 | 17.5 | 21.0 |
| 4 | $\Delta B = 4.5 \log_2 L - 5.02$ | 4.0 | 8.5 | 13.0 | 17.5 | 22.0 | 26.5 |
| 5 | $\Delta B = 5.5 \log_2 L - 6.53$ | 4.5 | 10.0 | 15.5 | 21.0 | 26.5 | 32.0 |

Table 2.2.1 Performance of oversampling noise shaping quantizers.

the reduced resolution of B' bits. Intermediate processing by the DSP is done at the low rate f_s and increased resolution of B bits. The overall quality remains the same through all the processing stages. Such a system replaces the traditional DSP system, shown in Fig. 1.7.1.

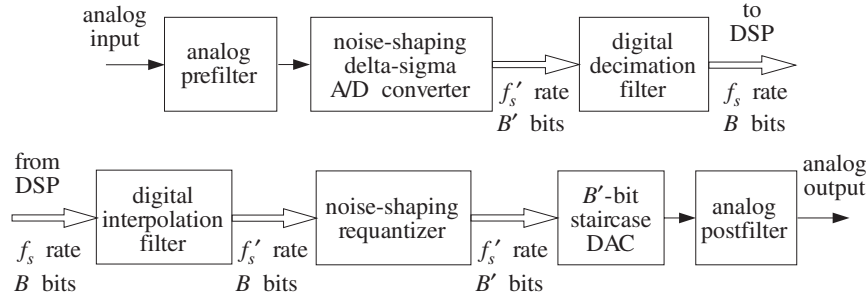


Fig. 2.2.5 Oversampling DSP system.

The faster sampling rate f'_s also allows the use of a less expensive, lower quality, antialiasing prefilter. The digital decimation filter converts the fast rate f'_s back to the desired low rate f_s at the higher resolution of B bits and removes the out-of-band quantization noise that was introduced by the noise shaping quantizer into the outside of the f_s Nyquist interval.

After digital processing by the DSP, the interpolation filter increases the sampling rate digitally back up to the fast rate f'_s . The noise shaping requantizer rounds the B -bit samples to B' bits, without reducing quality. Finally, an ordinary B' -bit staircase D/A converter reconstructs the samples to analog format and the postfilter smooths out the final output. Again, the fast rate f'_s allows the use of a low-quality postfilter.

Oversampling DSP systems are used in a variety of applications, such as digital transmission and coding of speech, the playback systems of CD players, and the sampling/playback systems of DATs. We will discuss the design of oversampling digital interpolation and decimation filters and the structure of noise shaping quantizers and $\Delta\Sigma$ converters in Chapter 12.

2.3 D/A Converters

Next, we discuss some coding details for standard A/D and D/A converters, such as binary representations of quantized samples and the successive approximation method of A/D conversion. We begin with D/A converters, because they are used as the building blocks of successive approximation ADCs. We take a functional view of such converters without getting into the electrical details of their construction.

Consider a B -bit DAC with full-scale range R , as shown in Fig. 2.3.1. Given B input bits of zeros and ones, $\mathbf{b} = [b_1, b_2, \dots, b_B]$, the converter outputs an analog value x_Q , that lies on one of the 2^B quantization levels within the range R . If the converter is unipolar, the output x_Q falls in the range $[0, R)$. If it is bipolar, it falls in $[-R/2, R/2)$.

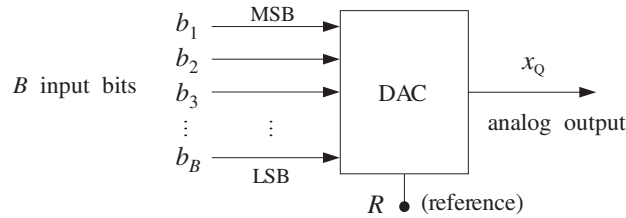


Fig. 2.3.1 B -bit D/A converter.

The manner in which the B bits $[b_1, b_2, \dots, b_B]$ are associated with the analog value x_Q depends on the type of converter and the coding convention used. We will discuss the three widely used types: (a) unipolar natural binary, (b) bipolar offset binary, and (c) bipolar two's complement converters.

The *unipolar natural binary* converter is the simplest. Its output x_Q is computed in terms of the B bits by:

$$x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \dots + b_B 2^{-B}) \quad (2.3.1)$$

The minimum level is $x_Q = 0$ and is reached when all the bits are zero, $\mathbf{b} = [0, 0, \dots, 0]$. The smallest nonzero level is $x_Q = Q = R 2^{-B}$ and corresponds to the least significant bit (LSB) pattern $\mathbf{b} = [0, 0, \dots, 0, 1]$. The most significant bit (MSB) pattern is $\mathbf{b} = [1, 0, 0, \dots, 0]$ and corresponds to the output value $x_Q = R/2$. The maximum level is reached when all bits are one, that is, $\mathbf{b} = [1, 1, \dots, 1]$ and corresponds to the analog output:

$$x_Q = R(2^{-1} + 2^{-2} + \dots + 2^{-B}) = R(1 - 2^{-B}) = R - Q$$

where we used the geometric series

$$\begin{aligned} 2^{-1} + 2^{-2} + \dots + 2^{-B} &= 2^{-1} (1 + 2^{-1} + 2^{-2} + \dots + 2^{-(B-1)}) \\ &= 2^{-1} \left(\frac{1 - 2^{-B}}{1 - 2^{-1}} \right) = 1 - 2^{-B} \end{aligned}$$

Eq. (2.3.1) can be written also in terms of the quantization width Q , as follows:

$$x_Q = R2^{-B}(b_12^{B-1} + b_22^{B-2} + \dots + b_{B-1}2^1 + b_B) \quad \text{or,}$$

$$\boxed{x_Q = Qm} \quad (2.3.2)$$

where m is the integer whose binary representation is $(b_1b_2 \dots b_B)$, that is,

$$m = b_12^{B-1} + b_22^{B-2} + \dots + b_{B-1}2^1 + b_B$$

As the integer m takes on the 2^B consecutive values $m = 0, 1, 2, \dots, 2^B - 1$, the analog output x_Q runs through the quantizer's consecutive levels. The *bipolar offset binary* converter is obtained by shifting Eq. (2.3.1) down by half-scale, $R/2$, giving the rule:

$$\boxed{x_Q = R(b_12^{-1} + b_22^{-2} + \dots + b_B2^{-B} - 0.5)} \quad (2.3.3)$$

The minimum and maximum attainable levels are obtained by shifting the corresponding natural binary values by $R/2$:

$$x_Q = 0 - \frac{R}{2} = -\frac{R}{2} \quad \text{and} \quad x_Q = (R - Q) - \frac{R}{2} = \frac{R}{2} - Q$$

The analog value x_Q can also be expressed in terms of Q , as in Eq. (2.3.2). In this case we have:

$$\boxed{x_Q = Qm'} \quad (2.3.4)$$

where m' is the integer m shifted by half the maximum scale, that is,

$$m' = m - \frac{1}{2}2^B = m - 2^{B-1}$$

It takes on the sequence of 2^B values

$$m' = -2^{B-1}, \dots, -2, -1, 0, 1, 2, \dots, 2^{B-1} - 1$$

One unnatural property of the offset binary code is that the level $x_Q = 0$ is represented by the nonzero bit pattern $\mathbf{b} = [1, 0, \dots, 0]$. This is remedied by the *two's complement* code, which is the most commonly used code. It is obtained from the offset binary code by *complementing* the most significant bit, that is, replacing b_1 by $\bar{b}_1 = 1 - b_1$, so that

$$\boxed{x_Q = R(\bar{b}_12^{-1} + b_22^{-2} + \dots + b_B2^{-B} - 0.5)} \quad (2.3.5)$$

Table 2.3.1 summarizes the three converter types and their input/output coding conventions and Table 2.3.2 compares the three coding conventions for the case $B = 4$ and $R = 10$ volts. The level spacing is $Q = R/2^B = 10/2^4 = 0.625$ volts. The codes $[b_1, b_2, b_3, b_4]$ in the first column, apply to both the natural and offset binary cases, but the quantized analog values that they represent are different.

| Converter type | I/O relationship |
|------------------|---|
| natural binary | $x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \dots + b_B 2^{-B})$ |
| offset binary | $x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \dots + b_B 2^{-B} - 0.5)$ |
| two's complement | $x_Q = R(\bar{b}_1 2^{-1} + b_2 2^{-2} + \dots + b_B 2^{-B} - 0.5)$ |

Table 2.3.1 Converter types.

| $b_1 b_2 b_3 b_4$ | natural binary | | offset binary | | 2's C |
|-------------------|----------------|------------|---------------|-------------|-------------------|
| | m | $x_Q = Qm$ | m' | $x_Q = Qm'$ | $b_1 b_2 b_3 b_4$ |
| — | 16 | 10.000 | 8 | 5.000 | — |
| 1 1 1 1 | 15 | 9.375 | 7 | 4.375 | 0 1 1 1 |
| 1 1 1 0 | 14 | 8.750 | 6 | 3.750 | 0 1 1 0 |
| 1 1 0 1 | 13 | 8.125 | 5 | 3.125 | 0 1 0 1 |
| 1 1 0 0 | 12 | 7.500 | 4 | 2.500 | 0 1 0 0 |
| 1 0 1 1 | 11 | 6.875 | 3 | 1.875 | 0 0 1 1 |
| 1 0 1 0 | 10 | 6.250 | 2 | 1.250 | 0 0 1 0 |
| 1 0 0 1 | 9 | 5.625 | 1 | 0.625 | 0 0 0 1 |
| 1 0 0 0 | 8 | 5.000 | 0 | 0.000 | 0 0 0 0 |
| 0 1 1 1 | 7 | 4.375 | -1 | -0.625 | 1 1 1 1 |
| 0 1 1 0 | 6 | 3.750 | -2 | -1.250 | 1 1 1 0 |
| 0 1 0 1 | 5 | 3.125 | -3 | -1.875 | 1 1 0 1 |
| 0 1 0 0 | 4 | 2.500 | -4 | -2.500 | 1 1 0 0 |
| 0 0 1 1 | 3 | 1.875 | -5 | -3.125 | 1 0 1 1 |
| 0 0 1 0 | 2 | 1.250 | -6 | -3.750 | 1 0 1 0 |
| 0 0 0 1 | 1 | 0.625 | -7 | -4.375 | 1 0 0 1 |
| 0 0 0 0 | 0 | 0.000 | -8 | -5.000 | 1 0 0 0 |

Table 2.3.2 Converter codes for $B = 4$ bits, $R = 10$ volts.

For the natural binary case, the values x_Q are positive, spanning the range $[0, 10)$ volts, with the maximum value being $R - Q = 10 - 0.625 = 9.375$. For offset binary, the level values are offset by half scale, $R/2 = 5$ volts, and span the range $[-5, 5)$ volts, with the maximum being $R/2 - Q = 5 - 0.625 = 4.375$ volts. Note that the upper ends of the full-scale range, $R = 10$ and $R/2 = 5$ volts, are shown in the table for reference and do not represent a level.

The last column shows the two's complement codes. They are obtained from the first column by complementing the MSB, b_1 . The quantized values x_Q represented by these codes are the *same* as in the offset binary case, that is, given in the fifth column of the table.

The two's complement code can be understood by wrapping the linear natural binary code around in a circle, as shown in Fig. 2.3.2. This figure shows the natural binary

integers m and their negative values in the lower half of the circle. The negative of any positive m in the upper semicircle can be obtained by the usual rule of complementing all its bits and adding one, that is, $m_{2c} = \overline{m} + 1$.

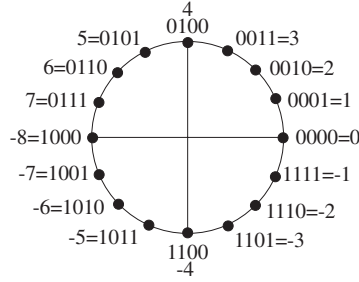


Fig. 2.3.2 Two's complement code.

Example 2.3.1: In Table 2.3.2 or Fig. 2.3.2, the level $m = 3$ corresponds to the natural binary quantized value of $x_Q = 1.875$ volts. The two's complement of m is obtained by the rule

$$m_{2c} = \overline{m} + 1 = \overline{(0011)} + (0001) = (1100) + (0001) = (1101) = -3$$

which, according to the fifth column of the table, corresponds to the two's complement quantized value $x_Q = -1.875$ volts. \square

The following C routine `dac.c` simulates the operation of the *bipolar two's complement* converter. Its inputs are the B bits $[b_1, b_2, \dots, b_B]$, and the full-scale range R , and its output is the analog value x_Q computed by Eq. (2.3.5).

```
/* dac.c - bipolar two's complement D/A converter */

double dac(b, B, R)
int *b, B;           bits are dimensioned as b[0], b[1], ..., b[B-1]
double R;
{
    int i;
    double dac = 0;

    b[0] = 1 - b[0];           complement MSB

    for (i = B-1; i >= 0; i--)   Hörner's rule
        dac = 0.5 * (dac + b[i]);

    dac = R * (dac - 0.5);       shift and scale

    b[0] = 1 - b[0];           restore MSB

    return dac;
}
```

Its usage is:

```
xQ = dac(b, B, R);
```

Because of the default indexing of arrays in C, the B -dimensional bit vector $b[i]$ is indexed for $i = 0, 1, \dots, B-1$. The declaration and dimensioning of $b[i]$ should be done in the main program. For example, if $B = 4$, the main program must include a line:

```
int b[4];
```

The array $b[i]$ can also be allocated dynamically for any desired value of B using `calloc`. The main program must include the lines:

```
int *b;           b is a pointer to int
B = 4;           B can also be read from stdin
b = (int *) calloc(B, sizeof(int));  allocates B int slots
```

The internal for-loop in `dac.c` implements a variant of Hörner's rule for evaluating a polynomial. The result is the binary sum $\bar{b}_1 2^{-1} + b_2 2^{-2} + \dots + b_B 2^{-B}$ which is then shifted by 0.5 and scaled by R . We leave the details of Hörner's algorithm for Problems 2.10–2.13. This algorithm will be used again later for the evaluation of z -transforms and DTFTs. (See the MATLAB function `dtft.m` in Appendix D.)

The routine `dac` may be modified easily to implement the natural binary and offset binary converter types, given by Eqs. (2.3.1) and (2.3.3).

2.4 A/D Converters

A/D converters quantize an analog value x so that it is represented by B bits $[b_1, b_2, \dots, b_B]$, as shown in Fig. 2.4.1. ADCs come in many varieties, depending on how the conversion process is implemented. One of the most popular ones is the *successive approximation* A/D converter whose main building block is a D/A converter in a feedback loop. It is shown in Fig. 2.4.2.

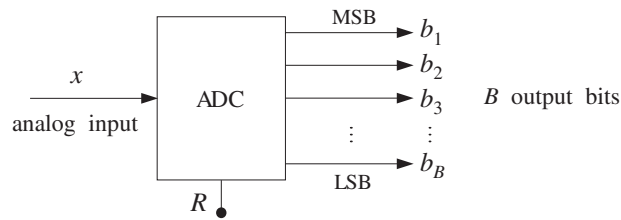


Fig. 2.4.1 B -bit A/D converter.

The conversion algorithm is as follows. Initially all B bits are cleared to zero, $\mathbf{b} = [0, 0, \dots, 0]$, in the successive approximation register (SAR). Then, starting with the MSB b_1 , each bit is turned *on* in sequence and a test is performed to determine whether that bit should be left *on* or turned *off*.

The control logic puts the correct value of that bit in the right slot in the SAR register. Then, leaving all the tested bits set at their correct values, the next bit is turned *on* in

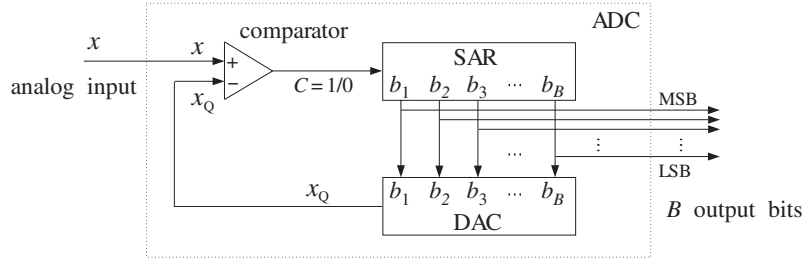


Fig. 2.4.2 Successive approximation A/D converter.

the SAR and the process repeated. After B tests, the SAR will hold the correct bit vector $\mathbf{b} = [b_1, b_2, \dots, b_B]$, which can be sent to the output.

At each test, the SAR bit vector \mathbf{b} is applied to the DAC which produces the analog quantized value x_Q . When a given bit is turned *on*, the output x_Q of the DAC is compared with the analog input x to the ADC. If $x \geq x_Q$, that bit is kept *on*; else, it is turned *off*. The output C of the comparator is the correct value of the bit being tested. The algorithm is summarized below:

```

for each  $x$  to be converted, do:
  initialize  $\mathbf{b} = [0, 0, \dots, 0]$ 
  for  $i = 1, 2, \dots, B$  do:
     $b_i = 1$ 
     $x_Q = \text{dac}(\mathbf{b}, B, R)$ 
    if  $(x \geq x_Q)$ 
       $C = 1$ 
    else
       $C = 0$ 
     $b_i = C$ 

```

Therefore, C becomes a *serial representation* of the bit vector \mathbf{b} . The algorithm imitates the operations shown in Fig. 2.4.2. It can be written more compactly as follows:

```

for each  $x$  to be converted, do:
  initialize  $\mathbf{b} = [0, 0, \dots, 0]$ 
  for  $i = 1, 2, \dots, B$  do:
     $b_i = 1$ 
     $x_Q = \text{dac}(\mathbf{b}, B, R)$ 
     $b_i = u(x - x_Q)$ 

```

where $u(x)$ is the unit-step function, defined by:

$$u(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

As stated above, the algorithm applies to the natural and offset binary cases (with corresponding versions of dac). It implements *truncation* of x to the quantization level just below, instead of rounding to the nearest level.

The algorithm converges to the right quantization level by performing a *binary search* through the quantization levels. This can be seen with the help of the first column of Table 2.3.2. The test $b_1 = 1$ or 0 determines whether x lies in the upper or lower half of the levels. Then, the test $b_2 = 1/0$ determines whether x lies in the upper/lower half of the first half, and so on. Some examples will make this clear.

Example 2.4.1: Convert the analog values $x = 3.5$ and $x = -1.5$ volts to their offset binary representation, assuming $B = 4$ bits and $R = 10$ volts, as in Table 2.3.2.

Solution: The following table shows the successive tests of the bits, the corresponding DAC output x_Q at each test, and the comparator output $C = u(x - x_Q)$.

| test | $b_1 b_2 b_3 b_4$ | x_Q | $C = u(x - x_Q)$ |
|-------|-------------------|-------|------------------|
| b_1 | 1 0 0 0 | 0.000 | 1 |
| b_2 | 1 1 0 0 | 2.500 | 1 |
| b_3 | 1 1 1 0 | 3.750 | 0 |
| b_4 | 1 1 0 1 | 3.125 | 1 |
| | 1 1 0 1 | 3.125 | |

For each bit pattern, the DAC inputs/outputs were looked up in the first/fifth columns of Table 2.3.2, instead of computing them via Eq. (2.3.3). When b_1 is tested, the DAC output is $x_Q = 0$ which is less than x ; therefore, b_1 passes the test. Similarly, b_2 passes the test and stays on. On the other hand bit b_3 fails the test because $x < x_Q = 3.75$; thus, b_3 is turned off. Finally, b_4 passes.

The last row gives the final content of the SAR register and the corresponding quantized value $x_Q = 3.125$. Even though $x = 3.5$ lies in the upper half between the two levels $x_Q = 3.75$ and $x_Q = 3.125$, it gets truncated down to the lower level. The C column is a serial representation of the final answer.

Note also the binary searching taking place: $b_1 = 1$ selects the upper half of the levels, $b_2 = 1$ selects the upper half of the upper half, and of these, $b_3 = 0$ selects the lower half, and of those, $b_4 = 1$ selects the upper half. For the case $x = -1.5$ we have the testing table

| test | $b_1 b_2 b_3 b_4$ | x_Q | $C = u(x - x_Q)$ |
|-------|-------------------|--------|------------------|
| b_1 | 1 0 0 0 | 0.000 | 0 |
| b_2 | 0 1 0 0 | -2.500 | 1 |
| b_3 | 0 1 1 0 | -1.250 | 0 |
| b_4 | 0 1 0 1 | -1.875 | 1 |
| | 0 1 0 1 | -1.875 | |

Bit b_1 fails the test because $x < x_Q = 0$, and therefore, $b_1 = 0$, and so on. Again, the final quantized value $x_Q = -1.875$ is that obtained by truncating $x = -1.5$ to the level below it, even though x lies nearer the level above it. \square

In order to quantize by *rounding* to the nearest level, we must shift x by half the spacing between levels, that is, use:

$$y = x + \frac{1}{2}Q$$

in place of x and perform *truncation* on y . If x is already in the upper half between two levels, then y will be brought above the upper level and will be truncated down to that level. The conversion algorithm for rounding is:

```

for each  $x$  to be converted, do:
   $y = x + Q/2$ 
  initialize  $\mathbf{b} = [0, 0, \dots, 0]$ 
  for  $i = 1, 2, \dots, B$  do:
     $b_i = 1$ 
     $x_Q = \text{dac}(\mathbf{b}, B, R)$ 
     $b_i = u(y - x_Q)$ 

```

Example 2.4.2: To quantize the value $x = 3.5$ by rounding, we shift it to $y = x + Q/2 = 3.5 + 0.625/2 = 3.8125$. The corresponding test table will be

| test | $b_1 b_2 b_3 b_4$ | x_Q | $C = u(y - x_Q)$ |
|-------|-------------------|-------|------------------|
| b_1 | 1 0 0 0 | 0.000 | 1 |
| b_2 | 1 1 0 0 | 2.500 | 1 |
| b_3 | 1 1 1 0 | 3.750 | 1 |
| b_4 | 1 1 1 1 | 4.375 | 0 |
| | 1 1 1 0 | 3.750 | |

Only b_4 fails the test because with it on, the DAC output $x_Q = 4.375$ exceeds y . The final value $x_Q = 3.750$ is the rounded up version of $x = 3.5$. For the case $x = -1.5$, we have $y = -1.5 + 0.625/2 = -1.1875$. The corresponding test table is

| test | $b_1 b_2 b_3 b_4$ | x_Q | $C = u(y - x_Q)$ |
|-------|-------------------|--------|------------------|
| b_1 | 1 0 0 0 | 0.000 | 0 |
| b_2 | 0 1 0 0 | -2.500 | 1 |
| b_3 | 0 1 1 0 | -1.250 | 1 |
| b_4 | 0 1 1 1 | -0.625 | 0 |
| | 0 1 1 0 | -1.250 | |

The value $x_Q = -1.250$ is the rounded up version of $x = -1.5$. □

The successive approximation algorithm for the *two's complement* case is slightly different. Because the MSB is complemented, it must be treated separately from the other bits. As seen in the last column of Table 2.3.2, the bit b_1 determines whether the number x is positive or negative. If $x \geq 0$ then, we must have $b_1 = 0$; else $b_1 = 1$. We can express this result by $b_1 = 1 - u(x)$, or, $b_1 = 1 - u(y)$ if we are quantizing by rounding. The remaining bits, $\{b_2, b_3, \dots, b_B\}$, are tested in the usual manner. This leads to the following two's complement conversion algorithm with rounding:

```

for each  $x$  to be converted, do:
   $y = x + Q/2$ 
  initialize  $\mathbf{b} = [0, 0, \dots, 0]$ 
   $b_1 = 1 - u(y)$ 
  for  $i = 2, 3, \dots, B$  do:
     $b_i = 1$ 
     $x_Q = \text{dac}(\mathbf{b}, B, R)$ 
     $b_i = u(y - x_Q)$ 

```

Example 2.4.3: The two's complement rounded 4-bit representations of $x = 3.5$ and $x = -1.5$ are:

$$\begin{aligned}
 x = 3.5 &\Rightarrow x_Q = 3.750 &\Rightarrow \mathbf{b} = [0, 1, 1, 0] \\
 x = -1.5 &\Rightarrow x_Q = -1.250 &\Rightarrow \mathbf{b} = [1, 1, 1, 0]
 \end{aligned}$$

They are obtained from the offset binary by complementing the MSB. The quantized values x_Q are the same as in the offset binary case — only the binary codes change. \square

Example 2.4.4: Consider the sampled sinusoid $x(n) = A \cos(2\pi f n)$, where $A = 3$ volts and $f = 0.04$ cycles/sample. The sinusoid is evaluated at the ten sampling times $n = 0, 1, \dots, 9$ and $x(n)$ is quantized using a 4-bit successive approximation ADC with full-scale range $R = 10$ volts. The following table shows the *sampled and quantized* values $x_Q(n)$ and the offset binary and two's complement binary codes representing them.

| n | $x(n)$ | $x_Q(n)$ | 2's C | offset |
|-----|--------|----------|-------|--------|
| 0 | 3.000 | 3.125 | 0101 | 1101 |
| 1 | 2.906 | 3.125 | 0101 | 1101 |
| 2 | 2.629 | 2.500 | 0100 | 1100 |
| 3 | 2.187 | 1.875 | 0011 | 1011 |
| 4 | 1.607 | 1.875 | 0011 | 1011 |
| 5 | 0.927 | 0.625 | 0001 | 1001 |
| 6 | 0.188 | 0.000 | 0000 | 1000 |
| 7 | -0.562 | -0.625 | 1111 | 0111 |
| 8 | -1.277 | -1.250 | 1110 | 0110 |
| 9 | -1.912 | -1.875 | 1101 | 0101 |

The 2's complement and offset binary codes differ only in their MSB. The quantized values they represent are the same. \square

The following routine `adc.c` simulates the operation of a *bipolar two's complement* successive approximation ADC. It makes successive calls to `dac.c` to determine each bit.

```

/* adc.c - successive approximation A/D converter */

#include <math.h>

double dac();
int u();

```

```

void adc(x, b, B, R)
double x, R;
int *b, B;
{
    int i;
    double y, xQ, Q;

    Q = R / pow(2, B);           quantization width  $Q = R/2^B$ 
    y = x + Q/2;                 rounding

    for (i = 0; i < B; i++)      initialize bit vector
        b[i] = 0;

    b[0] = 1 - u(y);            determine MSB

    for (i = 1; i < B; i++) {    loop starts with  $i = 1$ 
        b[i] = 1;                turn  $i$ th bit ON
        xQ = dac(b, B, R);       compute DAC output
        b[i] = u(y-xQ);          test and correct bit
    }
}

```

The inputs to the routine are the analog value x to be converted and the full-scale range R . The outputs are the B bits $\mathbf{b} = [b_1, b_2, \dots, b_B]$ representing x in the two's complement representation. The unit-step function $u(x)$ is implemented by the routine:

```

/* u.c - unit step function */

int u(x)
double x;
{
    if (x >= 0)
        return 1;
    else
        return 0;
}

```

Example 2.4.5: This example illustrates the usage of the routines `adc` and `dac`. Consider $L = 50$ samples of a sinusoid $x(n) = A \cos(2\pi f n)$ of digital frequency $f = 0.02$ cycles/sample and amplitude $A = 4$. The signal $x(n)$ is quantized using a successive approximation two's complement converter with rounding, as implemented by the routine `adc`. The following for-loop was used in the main program for calculating $x_Q(n)$:

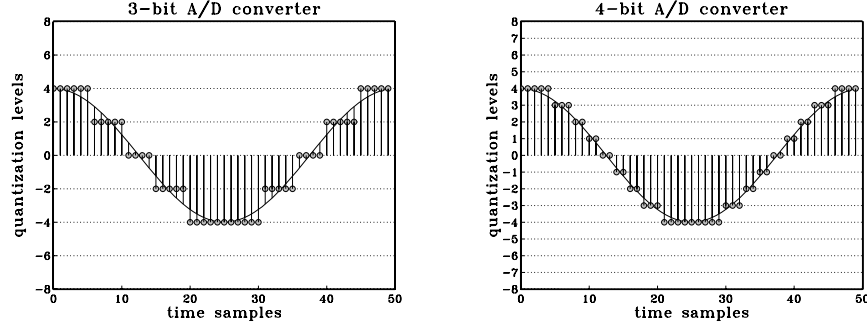
```

for (n=0; n<L; n++) {
    x[n] = A * cos(2 * pi * f * n);
    adc(x[n], b, B, R);
    xQ[n] = dac(b, B, R);
}

```

where each call to `adc` determines the bit vector \mathbf{b} , which is then passed to `dac` to calculate the quantized value.

The following figure shows the sampled and quantized signal $x_Q(n)$ plotted together with the exact values $x(n)$ for the two cases of a 3-bit and a 4-bit converter. The full-scale range was $R = 16$.



Example 2.4.6: This example illustrates the statistical properties of the quantization error. Consider L samples of the noisy sinusoidal signal:

$$x(n) = A \cos(2\pi f_0 n + \phi) + v(n), \quad n = 0, 1, \dots, L-1 \quad (2.4.1)$$

where ϕ is a random phase distributed uniformly in the interval $[0, 2\pi]$ and $v(n)$ is white noise of variance σ_v^2 . Using a B -bit two's complement converter with full-scale range R , these samples are quantized to give $x_Q(n)$ and the quantization error is computed:

$$e(n) = x_Q(n) - x(n), \quad n = 0, 1, \dots, L-1$$

According to the standard statistical model discussed in Section 2.1, the quantization noise samples $e(n)$ should be distributed uniformly over the interval $-Q/2 \leq e \leq Q/2$. This can be tested by computing the histogram of the L values of $e(n)$.

The theoretical statistical quantities given in Eqs. (2.1.9-2.1.11) can be calculated experimentally by the *time-average* approximations:

$$\sigma_e^2 = \frac{1}{L} \sum_{n=0}^{L-1} e^2(n) \quad (2.4.2)$$

$$R_{ee}(k) = \frac{1}{L} \sum_{n=0}^{L-1-k} e(n+k)e(n) \quad (2.4.3)$$

$$R_{ex}(k) = \frac{1}{L} \sum_{n=0}^{L-1-k} e(n+k)x(n) \quad (2.4.4)$$

We can also compute the autocorrelation of $x(n)$ itself:

$$R_{xx}(k) = \frac{1}{L} \sum_{n=0}^{L-1-k} x(n+k)x(n) \quad (2.4.5)$$

where in the last three equations, k ranges over a few lags $0 \leq k \leq M$, with M typically being much less than $L-1$. Note also that $\sigma_e^2 = R_{ee}(0)$. All four of the above expressions are special cases of the cross correlation, Eq. (2.4.4), which is implemented by the correlation routine `corr.c`, given in Appendix A.1.

For this experiment, we chose the following numerical values:

$B = 10$ bits
 $R = 1024$ volts, so that $Q = 1$ volt
 $L = 1000$ samples
 $M = 50$ lags
 $f_0 = 1/\sqrt{131} \simeq 0.08737$ cycles/sample
 $A = R/4 = 256$ volts and $\phi = 0$

The white noise $v(n)$ was distributed uniformly over the interval $[-R/4, R/4]$. Such numbers can be generated by:

$$v = 0.5R(u - 0.5) \quad (2.4.6)$$

where u is a *uniform* random number in the standardized interval $[0, 1]$. The quantity $(u - 0.5)$ is uniform over $[-0.5, 0.5]$, making v uniform over $[-R/4, R/4]$. We used the routine `ran` of Appendix B.1 to generate the u 's, but any other uniform random number generator could have been used. The samples $v(n)$ were generated by a for-loop of the form:

```

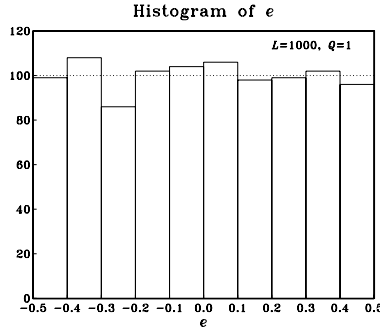
for (n=0; n<L; n++)
    v[n] = 0.5 * R * (ran(&i seed) - 0.5);

```

where the initial seed[†] was picked arbitrarily. With these choices, the sinusoidal and noise terms in $x(n)$ vary over half of the full-scale range, so that their sum varies over the full range $[-R/2, R/2]$, as is necessary for the model.

With $Q = 1$, the theoretical value of the noise variance is $\sigma_e = Q/\sqrt{12} = 1/\sqrt{12} = 0.289$. The experimental value computed using Eq. (2.4.2) was $\sigma_e = 0.287$.

The histogram of the computed $e(n)$ values was computed by dividing the interval $[-Q/2, Q/2] = [-0.5, 0.5]$ into 10 bins. It is shown below. Theoretically, for a uniform distribution, 1000 samples would distribute themselves evenly over the 10 bins giving $1000/10 = 100$ samples per bin.

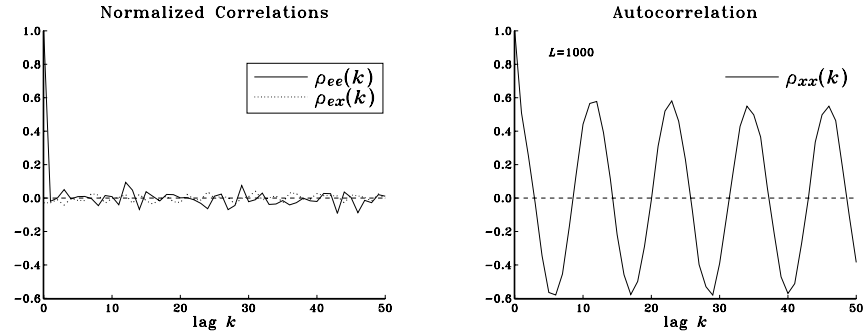


The next two figures show the standard *normalized* correlation functions:

$$\rho_{ee}(k) = \frac{R_{ee}(k)}{R_{ee}(0)}, \quad \rho_{ex}(k) = \frac{R_{ex}(k)}{\sqrt{R_{ee}(0)R_{xx}(0)}}, \quad \rho_{xx}(k) = \frac{R_{xx}(k)}{R_{xx}(0)}$$

[†]Note that `i seed` is passed by address in `ran(&i seed)`.

computed for lags $k = 0, 1, \dots, M$ using Eqs. (2.4.3–2.4.5).



Theoretically, $\rho_{ee}(k)$ should be $\delta(k)$ and $\rho_{ex}(k)$ should be zero. Using the results of Problem 2.20, the theoretical expression for $\rho_{xx}(k)$ will be, for the particular numerical values of the parameters:

$$\rho_{xx}(k) = 0.6 \cos(2\pi f_0 k) + 0.4 \delta(k)$$

Thus, although $x(n)$ itself is highly self-correlated, the quantization noise $e(n)$ is not. The above figures show the closeness of the experimental quantities to the theoretical ones, confirming the reasonableness of the standard statistical model. \square

Successive approximation A/D converters are used routinely in applications with sampling rates of 1 MHz or less. Slower converters also exist, the so-called counter or integrating type. They convert by searching through the quantization levels in a *linear* fashion, comparing each level with the value x to be converted. Therefore, they may require up to 2^B tests to perform a conversion. This is to be compared with the B binary searching steps of the successive approximation type.

For higher rates, *parallel* or flash A/D converters must be used. They determine all the bits simultaneously, in parallel, but they are electrically complex requiring $2^B - 1$ comparators internally. For this reason they are limited at present to $B \leq 12$ bits, achieving conversion rates of 500 MHz with 8 bits, or 50 MHz with 10 bits [41]. As discussed in Problem 2.21, two or more flash A/D converters can be cascaded together, in a so-called subranging configuration, to increase the effective quantization resolution.

2.5 Analog and Digital Dither*

Dither is a low-level white noise signal added to the input *before* quantization for the purpose of eliminating granulation or quantization distortions and making the total quantization error behave like white noise [52–69].

Analog dither can be added to the analog input signal before the A/D converter, but perhaps after the sample/hold operation. It is depicted in Fig. 2.5.1. In many applications, such as digital audio recordings, the inherent analog system noise of the

microphones or mixers may already provide some degree of dithering and therefore artificial dithering may not be necessary.

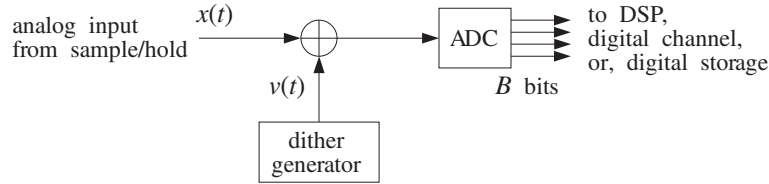


Fig. 2.5.1 Analog dither.

Digital dither can be added to a digital signal prior to a *requantization* operation that *reduces* the number of bits representing the signal.

This circumstance arises, for example, when an audio signal has been sampled and quantized with 20 bits for the purpose of high-quality digital mixing and processing, which then must be reduced to 16 bits for storing it on a CD. Another example is the noise shaping requantization required in oversampling D/A converters used in the playback systems of CD players and DAT decks.

Figure 2.5.2 shows a general model of the analog or digital dither process followed by the quantization operation. It represents a type of dither known as *nonsubtractive*.

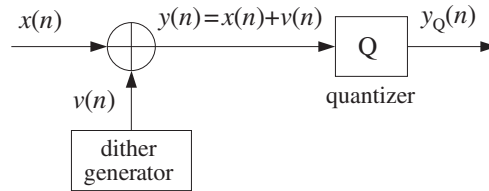


Fig. 2.5.2 Nonsubtractive dither process and quantization.

The input to the quantizer is the sum of the input signal $x(n)$ to be quantized or requantized and the dither noise $v(n)$, that is,

$$y(n) = x(n) + v(n)$$

The output of the quantizer is $y_Q(n)$, the quantized version of $y(n)$. The quantization error is

$$e(n) = y_Q(n) - y(n)$$

Thus, the *total* error resulting from dithering *and* quantization will be:

$$\epsilon(n) = y_Q(n) - x(n) \tag{2.5.1}$$

which can be written as

$$\epsilon(n) = (y(n) + e(n)) - x(n) = x(n) + v(n) + e(n) - x(n)$$

or,

$$\epsilon(n) = y_Q(n) - x(n) = e(n) + v(n) \quad (2.5.2)$$

that is, the sum of the dither noise plus the quantization error. Proper choice of the dither process $v(n)$ can guarantee that $e(n)$ and $v(n)$ will be uncorrelated from each other, and therefore the total error noise power will be

$$\sigma_\epsilon^2 = \sigma_e^2 + \sigma_v^2 = \frac{1}{12}Q^2 + \sigma_v^2 \quad (2.5.3)$$

The statistical properties of the dither signal $v(n)$, such as its probability density function (pdf), can affect drastically the nature of the total error (2.5.2). In practice, there are three commonly used types of dithers, namely, those with Gaussian, rectangular, or triangular pdf's. The pdf's of the rectangular and triangular cases are shown in Fig. 2.5.3. Note that the areas under the curves are equal to unity. In the Gaussian case, the zero-mean pdf is:

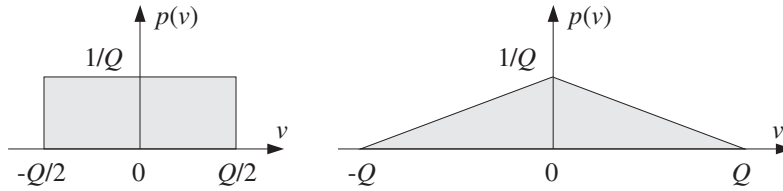


Fig. 2.5.3 Rectangular and triangular dither probability densities.

$$p(v) = \frac{1}{\sqrt{2\pi}\sigma_v} e^{-v^2/2\sigma_v^2} \quad (2.5.4)$$

with the recommended value for the variance:

$$\sigma_v^2 = \frac{1}{4}Q^2 \quad (2.5.5)$$

which corresponds to the rms value $v_{\text{rms}} = Q/2$, or half-LSB. It follows from Eq. (2.5.3) that the total error variance will be:

$$\sigma_\epsilon^2 = \frac{1}{12}Q^2 + \frac{1}{4}Q^2 = 4 \cdot \frac{1}{12}Q^2 = \frac{1}{3}Q^2$$

In the rectangular case, the pdf is taken to have width Q , that is, 1-LSB. Therefore, the dither signal can only take values in the interval:

$$-\frac{1}{2}Q \leq v \leq \frac{1}{2}Q$$

The corresponding pdf and variance are in this case:

$$p(v) = \begin{cases} \frac{1}{Q}, & \text{if } -\frac{1}{2}Q \leq v \leq \frac{1}{2}Q \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \sigma_v^2 = \frac{1}{12}Q^2 \quad (2.5.6)$$

Therefore, the total error variance will be:

$$\sigma_{\epsilon}^2 = \frac{1}{12}Q^2 + \frac{1}{12}Q^2 = 2 \cdot \frac{1}{12}Q^2 = \frac{1}{6}Q^2$$

Similarly, the width of the triangular dither pdf is taken to be $2Q$, that is, 2-LSB, and therefore, the corresponding pdf and variance are:

$$p(v) = \begin{cases} \frac{Q - |v|}{Q^2}, & \text{if } -Q \leq v \leq Q \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \sigma_v^2 = \frac{1}{6}Q^2 \quad (2.5.7)$$

and, the total error variance will be:

$$\sigma_{\epsilon}^2 = \frac{1}{12}Q^2 + \frac{1}{6}Q^2 = 3 \cdot \frac{1}{12}Q^2 = \frac{1}{4}Q^2$$

In summary, the total error variance in the three cases and the undithered case ($v = 0$) will be:

$$\sigma_{\epsilon}^2 = \begin{cases} Q^2/12, & \text{undithered} \\ 2Q^2/12, & \text{rectangular dither} \\ 3Q^2/12, & \text{triangular dither} \\ 4Q^2/12, & \text{Gaussian dither} \end{cases} \quad (2.5.8)$$

Thus, the noise penalty in using dither is to double, triple, or quadruple the noise of the undithered case. This corresponds to a decrease of the SNR by:

$$10 \log_{10} 2 = 3 \text{ dB}$$

$$10 \log_{10} 3 = 4.8 \text{ dB}$$

$$10 \log_{10} 4 = 6 \text{ dB}$$

which is quite acceptable in digital audio systems that have total SNRs of the order of 95 dB.

It has been shown that the *triangular* dither is the best (of the nonsubtractive types) in the sense that it accomplishes the main objective of the dithering process, namely, to eliminate the quantization distortions of the undithered case and to render the total error (2.5.2) equivalent to white noise [56].

Rectangular, uniformly distributed dither can be generated very simply by using a uniform random number generator such as `ran`. For example,

$$v = Q(u - 0.5)$$

where u is the random number returned by `ran`, that is, uniformly distributed over the interval $[0, 1)$. The shifting and scaling of u imply that v will be uniformly distributed within $-Q/2 \leq v < Q/2$.

Triangular dither can be generated just as simply by noting that the triangular pdf is the convolution of two rectangular ones and therefore v can be obtained as the *sum* of two independent rectangular random numbers, that is,

$$v = v_1 + v_2 \quad (2.5.9)$$

where v_1, v_2 are generated from two independent uniform u_1, u_2 , by

$$v_1 = Q(u_1 - 0.5) \quad \text{and} \quad v_2 = Q(u_2 - 0.5)$$

Example 2.5.1: This simulation example illustrates the impact of dither on the quantization of a low-amplitude sinusoid and the removal of quantization distortions. Consider the following dithered sinusoid:

$$y(n) = x(n) + v(n) = A \cos(2\pi f_0 n) + v(n)$$

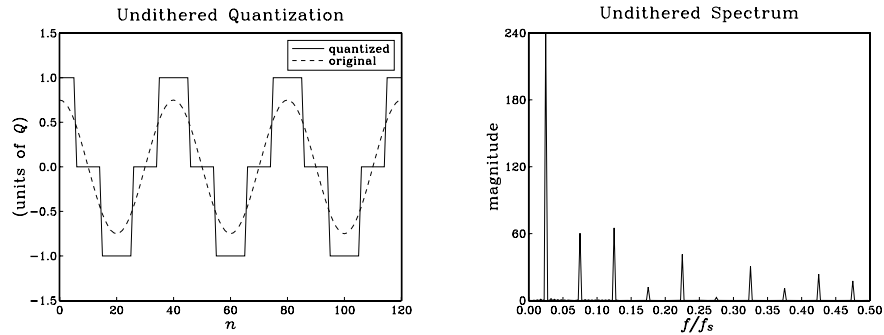
where A is taken to be below 1-LSB; for example, $A = 0.75Q$. The frequency f_0 is taken to be $f_0 = 0.025$ cycles/sample, which corresponds to $1/f_0 = 40$ samples per cycle. At an audio rate of $f_s = 40$ kHz, this frequency would correspond to a 1 kHz sinusoid.

The signal $y(n)$ is quantized using a 3-bit A/D converter, ($B = 3$), with full-scale range of $R = 8$ volts. Therefore, $Q = R/2^B = 8/2^3 = 1$, and $A = 0.75Q = 0.75$. Triangular dither was used, generated by Eq. (2.5.9). The dithered signal $y(n)$ and its quantized version $y_Q(n)$ were generated by the following loop:

```
for (n=0; n<Ntot; n++) {
    v1 = Q * (ran(&iseed) - 0.5);
    v2 = Q * (ran(&iseed) - 0.5);
    v = v1 + v2;
    y[n] = A * cos(2 * pi * f0 * n) + v;
    adc(y[n], b, B, R);
    yQ[n] = dac(b, B, R);
}
```

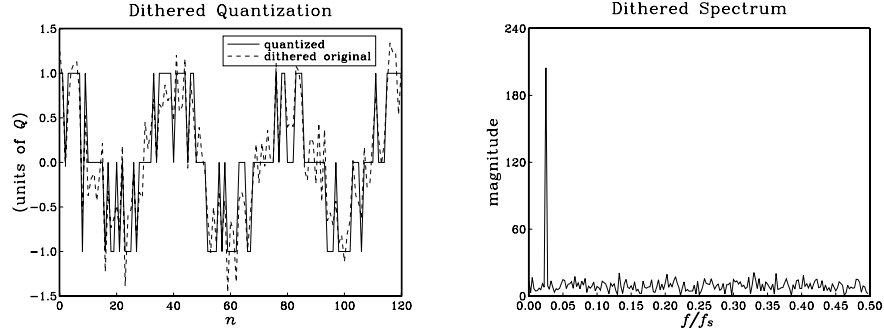
Note that $v1$ and $v2$ are independent of each other because each call to `ran` updates the seed to a new value.

The following graphs show the undithered sinusoid $x(n)$ and its quantized version $x_Q(n)$, together with its Fourier spectrum $|X_Q(f)|$ plotted over the right half of the Nyquist interval, that is, $0 \leq f \leq 0.5$, in units of cycles/sample.



The spectrum of $x_Q(n)$ has peaks at f_0 and the odd harmonics $3f_0, 5f_0$, and so forth. These harmonics were not present in $x(n)$. They are the artifacts of the quantization process which replaced the sinusoid by a square-wave-like signal.

The next two graphs show the dithered signal $y(n)$ and its quantized version $y_Q(n)$, together with its spectrum $|Y_Q(f)|$.



The main peak at f_0 is still there, but the odd harmonics have been eliminated by the dithering process and replaced by a typical featureless background noise spectrum. For digital audio, this noise is perceptually far more acceptable than the artificial harmonics introduced by the quantizer.

The above spectra were computed in the following way: The sequences $x_Q(n)$ and $y_Q(n)$ were generated for $0 \leq n \leq N_{\text{tot}} - 1$, with $N_{\text{tot}} = 1000$. Then, they were windowed using a length- N_{tot} Hamming window, that is,

$$y'_Q(n) = w(n)y_Q(n), \quad n = 0, 1, \dots, N_{\text{tot}} - 1$$

where,

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N_{\text{tot}} - 1}\right), \quad n = 0, 1, \dots, N_{\text{tot}} - 1$$

And, their DTFT

$$Y_Q(f) = \sum_{n=0}^{N_{\text{tot}}-1} y'_Q(n) e^{-2\pi jfn}$$

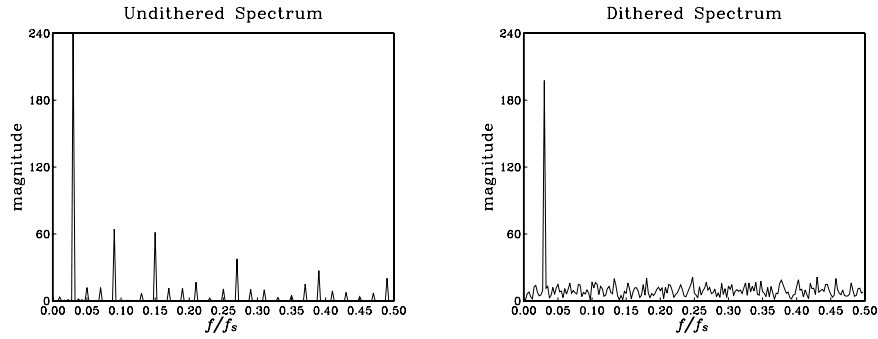
was evaluated at 200 equally spaced frequencies f over the interval $0 \leq f \leq 0.5$ [cycles/sample], that is, at $f_i = 0.5i/200$, $i = 0, 1, \dots, 199$.

This example is somewhat special in that the undithered spectrum $X_Q(f)$ contained only odd harmonics of the fundamental frequency f_0 . This is what one would expect if the quantized square-wave-like signal $x_Q(n)$ were an unsampled, analog, signal.

In general, the sampling process will cause all the odd harmonics that lie outside the Nyquist interval to be aliased back into the interval, onto frequencies that may or may not be odd harmonics. In the above example, because the sampling rate is an even multiple

of f_0 , that is, $f_s = 40f_0$, one can show that any odd harmonic of f_0 that lies outside the Nyquist interval will be wrapped onto one of the odd harmonics inside the interval.

But, for other values of f_0 , the out-of-band odd harmonics may be aliased onto in-band non-harmonic frequencies. For example, the following graphs show the undithered and dithered spectra in the case of $f_0 = 0.03$ cycles/sample.



In addition to the odd harmonics at $3f_0 = 0.09$, $5f_0 = 0.15$, and so forth, one sees non-harmonic peaks at:

$$f = 0.01, 0.05, 0.07, 0.13, 0.17, 0.19, 0.23, 0.25, \dots$$

which are the aliased versions of the following out-of-band odd harmonics:

$$33f_0, 35f_0, 31f_0, 29f_0, 39f_0, 27f_0, 41f_0, 25f_0, \dots$$

The beneficial effect of dithering works, of course, for any value of f_0 . □

An alternative dithering strategy is to use the so-called *subtractive* dither, as shown in Fig. 2.5.4. Here, the dither noise $v(n)$ that was added during the recording or transmission phase prior to quantization is subtracted at the playback or receiving end.

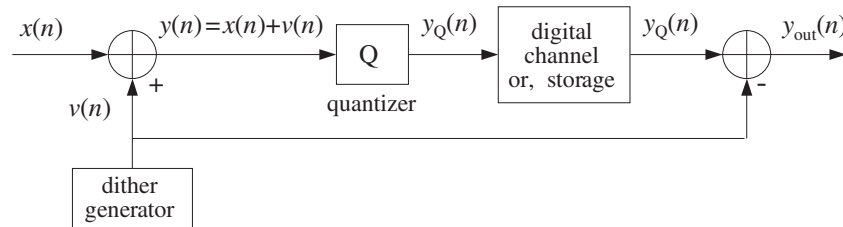


Fig. 2.5.4 Subtractive dither.

The total error in this case can be determined as follows:

$$\epsilon(n) = y_{\text{out}}(n) - x(n) = (y_Q(n) - v(n)) - x(n) = y_Q(n) - (x(n) + v(n))$$

or,

$$\epsilon(n) = y_Q(n) - y(n) = e(n)$$

that is, only the quantizer error. Therefore, its variance remains the same as the undithered case, $\sigma_e^2 = Q^2/12$, and the SNR remains the same.

It can be shown [56] that the *best* type of dither is subtractive rectangularly distributed dither with 1-LSB width, in the sense that it not only removes the quantization distortions but also renders the total error completely *independent* of the input signal. However, its practical implementation in digital audio and other applications is difficult because it requires a copy of the dither signal at the playback or receiving end.

By contrast, the triangular nonsubtractive dither that we considered earlier does not make the total error independent of the input—it only makes the power spectrum of the error independent of the input. In digital audio, this whitening of the total error appears to be enough perceptually. Therefore, triangular nonsubtractive dither is the best choice for practical use [56].

In summary, triangular nonsubtractive dither improves the quality of a digital processing system by removing the artifacts of the quantization process with only a modest decrease in the signal-to-noise ratio. It may be applied at any intermediate processing stage that involves reduction in the number of bits and, therefore, potential quantization distortions.

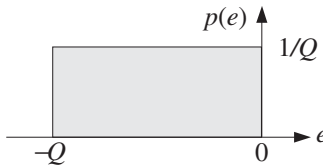
2.6 Problems

- 2.1 Consider a 3-bit successive approximation two's complement bipolar A/D converter with full scale range of $R = 16$ volts. Using the successive approximation algorithm, determine the quantized value as well as the corresponding 3-bit representation of the following analog input values: $x = 2.9, 3.1, 3.7, 4, -2.9, -3.1, -3.7, -4$.
Repeat using an offset binary converter.
- 2.2 Consider the signal $x(n) = 5 \sin(2\pi f n)$, where $f = 0.04$ [cycles/sample]. This signal is to be quantized using a 4-bit successive approximation bipolar ADC whose full-scale range is $R = 16$ volts. For $n = 0, 1, \dots, 19$, compute the numerical value of $x(n)$ and its quantized version $x_Q(n)$ as well as the corresponding bit representation at the output of the converter. Do this both for an *offset binary* converter and a *two's complement* converter.
- 2.3 It is desired to pick an A/D converter for a DSP application that meets the following specifications: The full-scale range of the converter should be 10 volts and the rms quantization error should be kept below 1 millivolt. How many bits should the converter have? What is the actual rms error of the converter? What is the dynamic range in dB of the converter?
- 2.4 Hard disk recording systems for digital audio are becoming widely available. It is often quoted that to record 1 minute of "CD quality" digital audio in *stereo*, one needs about 10 Megabytes of hard disk space. Please, derive this result, explaining your reasoning.
- 2.5 A digital audio mixing system uses 16 separate recording channels, each sampling at a 48 kHz rate and quantizing each sample with 20 bits. The digitized samples are saved on a hard disk for further processing.
 - a. How many megabytes of hard disk space are required to record a 3-minute song for a 16-channel recording?

- b. Each channel requires about 35 multiplier/accumulation (MAC) instructions to perform the processing of each input sample. (This corresponds to about 7 second-order parametric EQ filters covering the audio band.)

In how many nanoseconds should each MAC instruction be executed for: (i) each channel? (ii) all 16 channels, assuming they are handled by a single processor? Is this within the capability of present day DSP chips?

- 2.6 If the quantized value x_Q is obtained by *truncation* of x instead of rounding, show that the truncation error $e = x_Q - x$ will be in the interval $-Q < e \leq 0$. Assume a uniform probability density $p(e)$ over this interval, that is,

$$p(e) = \begin{cases} \frac{1}{Q} & \text{if } -Q < e \leq 0 \\ 0 & \text{otherwise} \end{cases}$$


Determine the mean $m_e = E[e]$ and variance $\sigma_e^2 = E[(e - m_e)^2]$ in terms of Q .

- 2.7 Using Eq. (2.2.10), determine the value of the oversampling ratio L to achieve 16-bit resolution using 1-bit quantizers for the cases of first-, second-, and third-order noise shaping quantizers. What would be the corresponding oversampled rate Lf_s for digital audio?
- 2.8 In a speech codec, it is desired to maintain quality of 8-bit resolution at 8 kHz sampling rates using a 1-bit oversampled noise shaping quantizer. For quantizer orders $p = 1, 2, 3$, determine the corresponding oversampling ratio L and oversampling rate Lf_s in Hz.
- 2.9 Show that the two's complement expression defined in Eq. (2.3.5) can be written in the alternative form:

$$x_Q = R \left(-b_1 2^{-1} + b_2 2^{-2} + \dots + b_B 2^{-B} \right)$$

- 2.10 Hörner's rule is an efficient algorithm for polynomial evaluation. Consider a polynomial of degree M

$$B(z) = b_0 + b_1 z + b_2 z^2 + \dots + b_M z^M$$

Hörner's algorithm for evaluating $B(z)$ at some value of z , say $z = a$, can be stated as follows:

```
initialize  $p = 0$ 
for  $i = M, M-1, \dots, 0$  do:
     $p = ap + b_i$ 
```

Verify that upon exit, p will be the value of the polynomial at $z = a$, that is, $p = B(a)$.

- 2.11 *Computer Experiment: Hörner's Rule.* Write a polynomial evaluation C routine `pol.c` that implements the algorithm of Problem 2.10. The routine should return the value $B(a)$ of the polynomial and should be dimensioned as follows:

```
double pol(M, b, a)
int M;                order of polynomial
double *b, a;         b is (M+1)-dimensional
```

- 2.12 Consider the following variation of Hörner's algorithm:

```

initialize  $q_{M-1} = b_M$ 
for  $i = M-1, M-2, \dots, 1$  do:
     $q_{i-1} = aq_i + b_i$ 
 $p = aq_0 + b_0$ 

```

where the final computation yields $p = B(a)$. Show that it is equivalent to the algorithm of Problem 2.10. This version is equivalent to “synthetic division” in the following sense.

The computed coefficients $\{q_0, q_1, \dots, q_{M-1}\}$ define a polynomial of degree $(M-1)$, namely, $Q(z) = q_0 + q_1z + \dots + q_{M-1}z^{M-1}$.

Show that $Q(z)$ is the *quotient* polynomial of the division of $B(z)$ by the monomial $z - a$. Moreover, the *last* computed $p = B(a)$ is the *remainder* of that division. That is, show as an identity in z

$$B(z) = (z - a)Q(z) + p$$

2.13 In the dac routines, the polynomial to be evaluated is of the form

$$B(z) = b_1z + b_2z^2 + \dots + b_Mz^M$$

The dac routines evaluate it at the specific value $z = 2^{-1}$. Show that the polynomial $B(z)$ can be evaluated at $z = a$ by the following modified version of Hörner’s rule:

```

initialize  $p = 0$ 
for  $i = M, M-1, \dots, 1$  do:
     $p = a(p + b_i)$ 

```

2.14 Consider a 4-bit successive approximation A/D converter with full-scale range of 8 volts. Using the successive approximation algorithm (with rounding), determine the 4-bit codes of the voltage values $x = 1.2, 5.2, -3.2$ volts, for the following types of converters:

- a. Natural binary.
- a. Bipolar two’s complement.

In each case, show all the steps of the successive approximation algorithm. Explain what happens if the analog voltage to be converted lies *outside* the full-scale range of the converter. This happens for $x = 5.2$ in two’s complement, and $x = -3.2$ in natural binary representations.

2.15 Carry out, by hand, the successive approximation conversion of all the signal values shown in the table of Example 2.4.4, for both the offset binary and two’s complement cases.

2.16 *Computer Experiment: DAC and ADC Routines.* Write C versions of the routines dac and adc for the natural binary, offset binary, and two’s complement cases that implement *truncation*. For the natural and offset binary cases, write another set of such routines that implement *rounding*.

2.17 *Computer Experiment: Simulating DAC and ADC Operations.* Generate $L = 50$ samples of a sinusoidal signal $x(n) = A \cos(2\pi fn)$, $n = 0, 1, \dots, L-1$ of frequency $f = 0.02$ [cycles/sample] and amplitude $A = 8$.

- a. Using a 3-bit ($B = 3$) bipolar two’s complement successive approximation A/D converter, as implemented by the routine adc, with full-scale range $R = 32$, quantize $x(n)$ and denote the quantized signal by $x_Q(n)$.

For $n = 0, 1, \dots, L - 1$, print in three parallel columns the true analog value $x(n)$, the quantized value $x_Q(n)$, and the corresponding two's complement bit vector \mathbf{b} .

On the same graph, plot the two signals $x(n)$ and $x_Q(n)$ versus n . Scale the vertical scales from $[-16, 16]$ and use 8 y-grid lines to indicate the 8 quantization levels.

- Repeat part (a) using a $B = 4$ bit A/D converter. In plotting $x(n)$ and $x_Q(n)$, use the *same* vertical scales as above, namely, from $[-16, 16]$, but use 16 y-grid lines to show the 16 quantization levels.
- What happens if the analog signal to be quantized has amplitude that *exceeds* the full-scale range of the quantizer? Most D/A converters will *saturate* to their largest (positive or negative) levels. To see this, repeat part (a) by taking the amplitude of the sinusoid to be $A = 20$.
- What happens if we use truncation instead of rounding? Repeat part (a) using the two's complement truncation routines `adc` and `dac` that you developed in the previous problem.

2.18 *Computer Experiment: Quantization Noise Model.* Reproduce the results of Example 2.4.6.

2.19 Show that the mean and variance of the random variable v defined by Eq. (2.4.6) of Example 2.4.6 are $m_v = 0$ and $\sigma_v^2 = R^2/48$.

2.20 Show that the normalized autocorrelation function $\rho_{xx}(k)$ of the signal $x(n)$ given by Eq. (2.4.1) in Example 2.4.6, is given by

$$\rho_{xx}(k) = \frac{R_{xx}(k)}{R_{xx}(0)} = a \cos(2\pi f_0 k) + (1 - a) \delta(k), \quad \text{where } a = \frac{SNR}{SNR + 1}$$

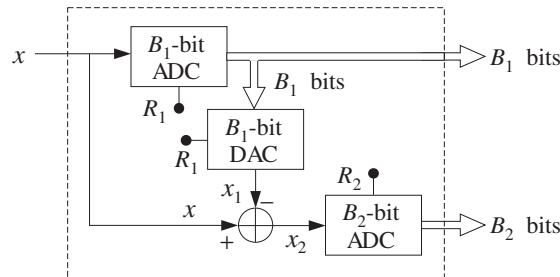
where $R_{xx}(k)$ defined as the statistical expectation value

$$R_{xx}(k) = E[x(n+k)x(n)]$$

Assume that phase of the sinusoid ϕ is not correlated with $v(n)$. The quantity SNR is the signal-to-noise ratio $SNR = A^2/(2\sigma_v^2)$. For the numerical values of Example 2.4.6, show $a = 0.6$.

2.21 *Computer Experiment: Subranging Converters.* It was mentioned that parallel A/D converters are at present limited in their bits. However, it is possible to use two of them in cascade. For example, using two identical 6-bit flash ADCs, one can achieve effective resolution of 12 bits at conversion rates of 10 MHz.

Consider a B -bit ADC and write B as the sum of two integers $B = B_1 + B_2$. The conversion of an analog value x to its B -bit representation can be done in two stages: First, convert x into its B_1 -bit representation. This is equivalent to keeping the first B_1 most significant bits of its B -bit representation. Let x_1 be the quantized B_1 -bit value. Then, form the difference $x_2 = x - x_1$ and quantize it to B_2 bits. These operations are shown in the following figure:



The B_1 -bit word from the first ADC is sent to the output and also to a B_1 -bit DAC whose output is x_1 . The analog subtracter forms x_2 , which is sent to the B_2 -bit ADC producing the remaining B_2 bits.

- a. What should be the full-scale ranges R_1 and R_2 of the two ADCs in order for this arrangement to be *equivalent* to a single $(B_1 + B_2)$ -bit ADC with full-scale range R ? What is the relationship of R_1 and R_2 in terms of R ?
- b. Using the routines `adc` and `dac` as building blocks, write a routine that implements this block diagram. Test your routine on the signal:

$$x(n) = A \cos(2\pi f_0 n), \quad n = 0, 1, \dots, L - 1$$

where $A = 4$, $f_0 = 0.02$, and $L = 50$. Take $B_1 = 5$, $B_2 = 5$, $B = B_1 + B_2 = 10$, and $R = 16$. Compare the results of your routine with the results of an equivalent single B -bit ADC with full-scale range R .

- c. How does the block diagram generalize in the case of cascading three such converters, such that $B = B_1 + B_2 + B_3$?

2.22 *Computer Experiment: Triangular Dither.* Reproduce the results and graphs of Example 2.5.1.