Chapter 14

Digital Signal Processing Applications*

Throughout this book, we have encountered applications of digital signal processing in examples and problems. This chapter is exclusively devoted to applications. We assume that you have mastered most of the book by now, and are ready to explore DSP as it is used in real life. It is impossible, in a single chapter, to do justice to all but a small sample of the DSP world. Our small sample comprises seven topics.

First, we present signal compression; we have already touched upon this topic in Example 12.8, in the context of filter banks and subband coding. Here we present a compression method considerably more important than subband coding: the discrete cosine transform. The DCT has been standardized in recent years for image and motion picture compression. Our treatment of DCT-based signal compression is limited to temporal signals, however, and we will not deal with image compression.

The second topic is speech modeling and compression by a technique called linear predictive coding (LPC). LPC is not a new technique: its principles have been known since the mid-1960s, and it has been applied to speech since the mid-1970s. However, recent years have seen substantial developments in this area. In particular, cellular telephone services now use LPC as a standard. Here we present, as an example, the speech compression and coding technique used in the Pan-European Digital Mobile Radio System, better known by the French acronym GSM (Groupe Special Mobile).

The third topic concerns modeling and processing of musical signals. Compared with other natural signals, musical signals are characterized by an orderly structure; this makes them convenient for modeling and interpretation. On the other hand, high fidelity is an extremely important factor in our enjoyment of music; this makes synthesis of musical signals difficult and challenging.

The fourth topic is probably the one getting the most attention in today's technical world: communication. Until recently, electronic circuitry in wireless digital communication systems has been chiefly analog. However, DSP is rapidly taking over, and future communication systems will undoubtedly be based on digital processing. As an example of this vast field, we present a digital receiver for frequency-shift keying (FSK) signals.

The fifth topic presents an example from the biomedical world: electrocardiogram (ECG) analysis. We have chosen ECG since, of all electrical signals measured from the human body, it is probably the easiest to analyze, at least at a basic level. Our example concerns the use of ECG for measurement of heart rate variations.

The last two topics are concerned with technology. In the first, we discuss microprocessors for DSP applications. We have chosen, as an example, a particular DSP chip of the mid-1990s vintage: the Motorola DSP56301. We use this example for illustrating common trends and techniques in DSP hardware today. Among all topics covered in this book, this will probably be the fastest to become obsolete, since digital technology progresses at an enormously fast rate. Finally, we present a modern A/D converter technology: sigma-delta A/D. Sigma-delta A/D converters provide a fine example of the advantages gained by combining very large-scale integration (VLSI) technology and digital signal processing principles.

14.1 Signal Compression Using the DCT

Any information stored digitally is inherently finite, say of N bits. *Compression* is the operation of representing the information by N_c bits, where $N_c < N$. Compression is useful for economical reasons: it saves storage space, transmission time, or transmission bandwidth. The ratio N/N_c is called the *compression ratio*. The greater the compression ratio, the better the compression.

There are two basic types of compression: lossless and lossy. Lossless compression is defined by the property that the original information can be retrieved *exactly* from the compressed information. Mathematically, lossless compression is an invertible operation. For example, compression of text must be lossless, since otherwise the text cannot be exactly retrieved. The highest possible lossless compression ratio of a given information is related to the *entropy* of the source of information, a term perhaps known to those who have studied information theory (but which we shall not attempt to define here). Typical ratios achievable with lossless compression are 2 to 3. The best-known compression methods are the *Huffman code* [Huffman, 1952] and the *Ziv-Lempel algorithms* [Ziv and Lempel, 1977, 1978]. We shall not discuss lossless compression further here; see Cover and Thomas [1991] for a detailed exposition of this subject.

When data are subjected to lossy compression, the original information cannot be retrieved exactly from the compressed information. Mathematically, lossy compression is a noninvertible operation. The advantage of lossy compression over lossless one is that much higher compression ratios can be achieved. However, lossy compression is limited to applications in which we can tolerate the loss. For example, speech signals can be compressed at high ratios (10 and above), and the quality of the reconstructed speech will be only slightly inferior to the original speech. Most people can tolerate some distortion of a speech signal without impairment of their ability to comprehend its contents. Therefore, compression is highly useful for transmitting speech over telephone lines or via wireless channels. Even higher compression ratios can be obtained for images. Compression is very useful for storing images, since image storage is highly space consuming. Still higher compression ratios can be achieved for motion pictures, a fact that is useful for storing motion pictures on digital media (such as video discs) and in video transmission (video conferencing, digital TV).

There are many methods for lossy signal compression. Here we describe the principle of operation of compression by orthonormal transforms. Consider a signal $x[n]$ of length N; let $\boldsymbol{O}_N$ be an $N \times$N orthonormal matrix, and $X^{\mathrm{o}}[k]$ the result of transforming $x[n]$ by $\boldsymbol{O}_N$, that is,

$$\boldsymbol{X}_N^{\mathrm{o}} = \boldsymbol{O}_N \boldsymbol{x}_N. \tag{14.1}$$

The transform is selected with an attempt to decorrelate the components of the transformed signal $\boldsymbol{X}_N^{\mathrm{o}}$. Successful decorrelation often results in nonuniform distribution of the magnitudes of the components of $\boldsymbol{X}_N^{\mathrm{o}}$. Typically, there are a few components whose magnitudes are large, and a large number of components whose magnitudes are small. We now explain how this nonuniform distribution is used for compression.

Assume that we represent each component of $\boldsymbol{x}_N$ by B bits, so the complete vector requires NB bits. In general, the vector $\boldsymbol{X}_N^{\mathrm{o}}$ is represented by NB bits or more. (Extra bits may be needed to guarantee that there will be no overflows in the additions.) The next step in the compression procedure takes advantage of the nonuniform distribution of the magnitudes of the $X_N^{\mathrm{o}}[k]$ through *quantization*: Each component $X_N^{\mathrm{o}}[k]$ is represented by a number of bits proportional to its magnitude. As a result, the total number of bits needed for representing the complete vector $\boldsymbol{X}_N^{\mathrm{o}}$ will often be smaller than NB. In an extreme example of quantization, every $X_N^{\mathrm{o}}[k]$ smaller in magnitude than a certain threshold is completely neglected, that is, replaced by zero. We denote the compressed version of $\boldsymbol{X}_N^{\mathrm{o}}$, resulting from the quantization procedure, by $\hat{\boldsymbol{X}}_N^{\mathrm{o}}$.

Because the transform is orthonormal, the signal can be recovered exactly from its (uncompressed) transform by

$$\boldsymbol{x}_N = \boldsymbol{O}_N' \boldsymbol{X}_N^{\mathrm{o}}. \tag{14.2}$$

Correspondingly, an approximate (lossy) reconstruction of the signal from its compressed version is obtained by the operation

$$\hat{\boldsymbol{x}}_N = \boldsymbol{O}_N' \hat{\boldsymbol{X}}_N^{\mathrm{o}}. \tag{14.3}$$

In summary, signal compression by an orthonormal transform is carried out by the following operations:

1. Transforming the signal using (14.1).
2. Modifying the transform values by quantization. This should be done according to a rule designed to guarantee that distortion due to quantization will not exceed a certain percent of the total signal energy.

The modified transformed values are used for storage or transmission of the information. Compression is efficient if the total number of bits in these values is much smaller than NB. Note that, because of orthonormality, the total energy of the quantization error in the transform domain is equal to that in the signal domain. Signal reconstruction is carried out by the operation (14.3). The result again consists of NB bits, but these are not identical to the bits of the original signal.

In choosing an orthonormal transform for signal compression, there are two major considerations:

1. The compression ratios that can be achieved for the class of signals in question, under the constraint that distortion due to compression (which affects the *quality* of the compressed signal) be within acceptable limits.
2. The amount of computation needed to carry out the compression and reconstruction operations.

For a general orthonormal transform, each of the operations (14.1) and (14.3) requires a number of operations proportional to N^2. This is prohibitively large in many applications. However, the transforms we have studied all have fast versions, requiring a number of operations on the order of $N \log_2 N$. The fast Fourier transform, the discrete cosine and sine transforms, and the Hartley transform are all candidates for signal compression.

Of the aforementioned transforms, the DCT has been shown to be particularly effective for first-order autoregressive signals. Recall from (13.19) that a first-order AR signal obeys the first-order difference equation

$$x[n] = a_1 x[n-1] + v[n], \tag{14.4}$$

where a_1 is a constant parameter, smaller than 1 in magnitude, and $v[n]$ is white noise. In particular, when a_1 is nearly 1, the signal varies slowly over time. Natural images often have characteristics similar to first-order autoregressive signals, so the DCT is an effective compression technique for them. Image processing is beyond the scope of this book, so we just mention two simple facts, which can be understood without delving deeply into this subject:

1. An image is a two-dimensional array of real numbers. The numbers represent the intensity levels of the points (*pixels*) in the image.
2. A two-dimensional transform such as the DFT or DCT can be performed by transforming each row of the image individually as a one-dimensional signal, then transforming each column of the result individually; similarly for an inverse transform.

We now illustrate signal compression by DCT. We choose DCT-II, which is the one in standard use for image compression applications; see Section 4.9.2 for the definition and properties of DCT-II. We use a simple quantization rule for compression, as follows. The components of the transform are sorted in order of decreasing magnitudes. Denote the sorted components by $\{X^{\text{c2}}_{\text{sort}}[k],\ 0 \le k \le N-1\}$. Since the transform is orthonormal, it satisfies Parseval's energy conservation property, that is,

$$E \triangleq \sum_{n=0}^{N-1} (x[n])^2 = \sum_{k=0}^{N-1} (X^{\text{c2}}[k])^2. \tag{14.5}$$

Let β be a number smaller than 1, representing the fraction of energy we wish to preserve in the compressed signal (e.g., $\beta = 0.99$). The first M components (after sorting) are retained to their full precision, where M is the minimum value for which

$$\sum_{k=0}^{M-1} (X^{\text{c2}}_{\text{sort}}[k])^2 \ge \beta E. \tag{14.6}$$

The values of $\{X^{\text{c2}}_{\text{sort}}[k],\ k \ge M\}$ are set to zero. The compressed information consists of $\{X^{\text{c2}}_{\text{sort}}[k],\ 0 \le k \le M-1\}$ and their indices in the original transform vector.

Example 14.1 The signal shown in Figure 14.1(a) is first-order AR, with $a_1 = 0.99$. Its DCT-II components are shown in Figure 14.1(b), in their natural order. As we see, the signal has random appearance. Its time variation is generally slow, but it also has high-frequency components. The DCT-II components have large magnitudes for small values of k (corresponding to low frequencies), and small magnitudes for large values of k. We compressed the signal using a threshold parameter $\beta = 0.99$. In this case, the number of coefficients after compression is $M = 50$. Figure 14.2(a) shows the result of reconstructing the compressed signal as a solid line. For reference, we show the original signal as a dotted line. As we see, the reconstructed signal follows the outline of the original signal very well, but not the high-frequency oscillations. Figure 14.2(b) emphasizes this effect by showing the error signal $\hat{x}[n] - x[n]$ scaled by 5. The error signal has most of its energy in the high frequencies, a phenomenon typical of this type of compression. We can regard the compression-reconstruction procedure as a special kind of low-pass filtering of the input signal.

For comparison, we compress the same signal using the same procedure, but with the DFT matrix. Since the DFT matrix is complex, only the first $N/2$ coefficients are used for compression. These and their complex conjugates are then used for reconstruction. For a threshold $\beta = 0.99$, 34 complex coefficients, equivalent to 68 real coefficients, are needed. The conclusion is that DCT-II has a higher compression ratio than DFT in this case. □

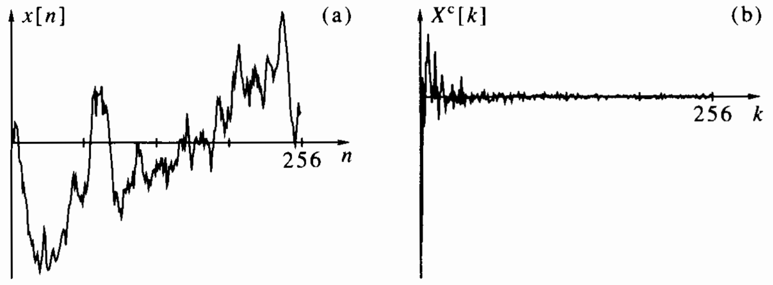

Figure 14.1 The signal (a) and its DCT (b) in Example 14.1.

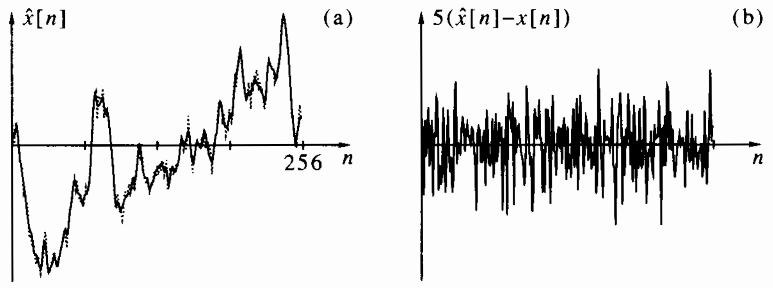

Figure 14.2 The reconstructed signal (a) and the error signal (b) in Example 14.1 (the error signal is scaled by 5).

In summary, the discrete cosine transform is a proven effective compression tool. It has become the method of choice in internationally accepted image compression standards, such as JPEG (from the *Joint Photographic Experts Group*) for still image compression, and MPEG (from the *Motion Picture Expert Group*) for motion pictures compression. See Pennebaker and Mitchell [1993] for further reading.

14.2 Speech Signal Processing

In Chapter 12 we described speech compression by subband coding; see Example 12.8, page 493. In this section we examine an application of digital signal processing to modeling and compression of speech signals. We describe a modeling method known as *linear predictive coding*, and its application to the speech compression standard GSM. At the time this book is written, GSM has become an almost worldwide standard for cellular telephone communications (although not in the United States).

14.2.1 Speech Modeling

Human voice is generated mainly by the vocal cords. The vibration of the vocal cords excites the *vocal tract*, which includes the larynx (the part of the respiratory tube containing the vocal cords), the pharynx (the part between the larynx and the mouth), the mouth cavity, and the nasal cavity. Sound waves are emitted from the lips and, to a lesser degree, from the nose.

The basic units of speech are called *phonemes*. The phonemes in the English language are listed in Rabiner and Juang [1993, page 25]. They are broadly classified into the following groups: (1) vowels, (2) diphthongs, (3) semivowels, (4) consonants. The consonants are further divided into nasals, voiced stops, unvoiced stops, voiced fricatives, unvoiced fricatives, whisper, and affricates. Most phonemes are *voiced*; that is, they are generated by the vocal cords. A few are *unvoiced*; that is, the vocal cords are not involved in their generation.

In this section we discuss modeling of voiced sound, which is largely the most important part of speech. The signal generated by the vocal cords has roughly the shape of a quasi-periodic impulse train. By *quasi-periodic* we mean that (1) the intervals between successive impulses are not exactly constant, but vary slightly and (2) the amplitudes of different impulses are not exactly constant. The mathematical description of a discrete-time, quasi-periodic impulse train is

$$u[n] = \sum_{m=-\infty}^{\infty} c_m \delta[n - n_m] = \begin{cases} c_m, & n = n_m, \\ 0, & \text{otherwise.} \end{cases} \tag{14.7}$$

The impulses occur at instants n_m and their amplitudes are c_m. The impulses are relatively sparse; that is, the differences $n_m - n_{m-1}$ are much larger than 1. These differences are nearly, but not exactly, constant. The reciprocal of the average interval between successive impulses is called the *pitch frequency*, or the pitch for short. The pitch is characteristic of the person. The pitch of adult males is typically 80–120 Hz, and that of adult females is about 150–300 Hz. Children have a higher pitch. When a person sings, the pitch varies according to the melody. Pitch varies to a certain extent even during normal speaking. For example, a question is articulated by raising the pitch toward the end.

The vocal tract is commonly modeled by a time-varying linear system. During a single phoneme, the parameters of the system are approximately constant; however, they vary widely from phoneme to phoneme, and this is why the phonemes sound different. The frequency response of the linear system representing a particular phoneme typically exhibits several peaks. The frequencies of these peaks are called *formants*. The assumption that the linear system is time invariant during the phoneme is rather crude: in reality, it varies during the phoneme. However, for relatively short time intervals (up to about 30 milliseconds) it is common to assume that the vocal tract system is LTI.

The common LTI model for the vocal tract is as a *rational, all-pole* transfer function, that is,

$$H^z(z) = \frac{1}{a(z)}, \quad \text{where} \quad a(z) = 1 + a_1 z^{-1} + \cdots + a_p z^{-p}. \tag{14.8}$$

Typical values of p are 8 to 12. In summary, a common short-time model for voiced speech is

$$y[n] = -a_1 y[n-1] - \cdots - a_p y[n-p] + u[n], \tag{14.9}$$

with $u[n]$ as in (14.7).

Example 14.2 The file `female-w.mat` contains a speech fragment 0.3 second long, of a female saying "why" (see page vi for information on how to download this file). The sampling frequency is 11,025 Hz. You can hear it by loading the file into the MATLAB environment and issuing the command `sound(s,11025)` (assuming that your computer is equipped with a sound system). Figure 14.3 shows the speech waveform. Part a shows the full 0.3 second and part b zooms in on 30 milliseconds in the middle. As we see, the signal indeed looks like a response to a sequence of nearly periodic impulses. □

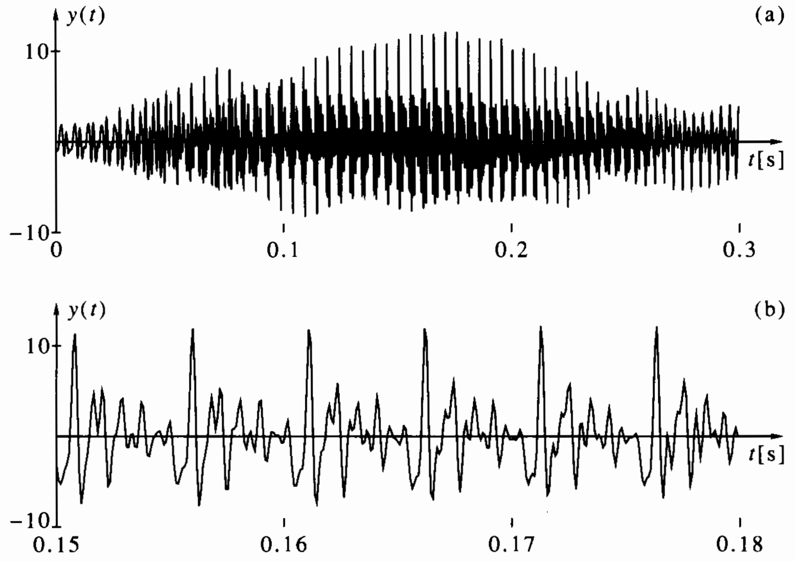

Figure 14.3 The speech signal in Example 14.2 ("why" pronounced by a female): (a) full waveform; (b) 30-millisecond segment.

The voiced speech model (14.7), (14.9) is known as *linear predictive coding* (LPC). Recalling what we have studied in Section 13.3, we see that (14.9) is essentially an autoregressive model for $y[n]$. However, in the present context, the excitation signal $u[n]$ is not white noise and $y[n]$ is not a random WSS signal. Still, $y[n]$ can be thought of comprising a predicted value—the linear combination of past values on the right side of (14.9)—and an error term $u[n]$. If $u[n] = 0$, the prediction is exact. A nonzero value of $u[n]$ can be interpreted as a correction term to the prediction, to make it exact. The term LPC has become standard in the speech literature, so we will prefer it to the term "AR model" in this application.

The LPC model is useful for applications such as speech generation by computer (synthetic speech) and speech compression. Speech compression by LPC modeling consists of the following conceptual stages:

1. Division of the sampled speech into short intervals, typically 10-30 milliseconds long. These intervals, called *frames*, can be overlapping or nonoverlapping.
2. Computation of a set of LPC parameters $\{a_k,\ 1 \leq k \leq p\}$ in the model (14.9), describing the vocal tract system for the phoneme being pronounced during the

present frame. This is done either by the Yule-Walker method, as described in Sections 13.4.1 and 13.4.6 (the autocorrelation method), or by least-squares modeling, as described in Section 13.4.7 (the covariance method). In the former case, the data $y[n]$ in the frame are sometimes windowed. Although data windowing can hardly be justified from a theoretical viewpoint,[1] experience shows that it tends to improve the quality of the model.

3. Computation of the excitation signal $u[n]$.
4. Modeling of the excitation signal, in particular its pitch and amplitude during the frame. Such modeling may lead to a secondary excitation signal, which generates $u[n]$ via another LTI system.
5. Coding and compression of the variables found by this procedure: the LPC parameters, the excitation signal parameters, and the secondary excitation signal (if present). The compressed speech consists of the coded variables in every frame; the rate of the compressed speech is the number of bits in the coded variables divided by the duration of the frame.

Reconstruction of a speech signal from its compressed representation is basically a reversal of the aforementioned process. The variables in each frame are decoded, the secondary excitation signal is regenerated, used for building the primary excitation signal $u[n]$, which is used in turn for reconstructing the speech signal $y[n]$.

Example 14.2 (continued) The signal contained in the file `female-w.mat` and plotted in Figure 14.3 is divided into frames of 160 samples each (corresponding to about 14.5 milliseconds). For each frame, the estimated covariance sequence $\hat{\kappa}_y[m]$ is computed as in (13.66), and the Yule-Walker equation (13.26a) is solved to give a corresponding vector of $\{a_k\}$. This way, 20 vectors are generated (since there are 20 frames). We use an LPC model of order $p = 10$. Figure 14.4 shows the magnitude response of the modeling filter $1/a(z)$ for two frames: the 11th and the 16th. The frequency range in the plot is up to 4 kHz, since the speech signal was band limited to 3.8 kHz prior to sampling. In the first graph we see two formants, at frequencies about 1 and 2.5 kHz. In the second we see three formants, at frequencies about 0.6, 1, and 2.8 kHz.

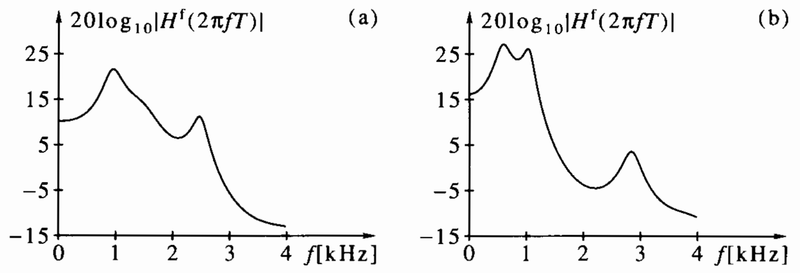

Figure 14.4 Magnitude responses of the modeling filter in Example 14.2: (a) the 11th frame; (b) the 16th frame.

The excitation signal $u[n]$ is generated by passing $y[n]$ through the FIR filter $a(z)$, as expressed in (14.9). At each frame, the coefficients of $a(z)$ are set to their respective values obtained from the solution of the Yule-Walker equation (13.26a). The signal thus obtained is shown in Figure 14.5. Part a shows the full time interval, and part b zooms in on 30 milliseconds in the middle. As we see, the signal looks approximately like a nearly periodic impulse train.

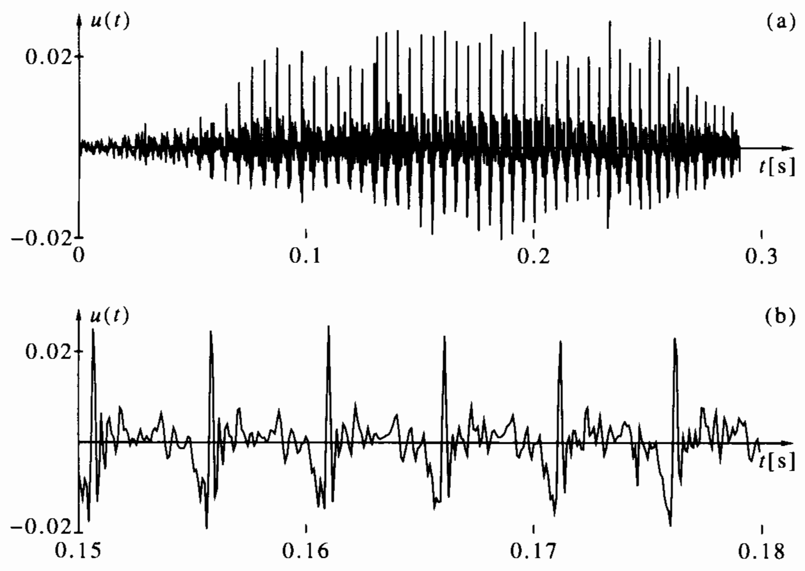

Figure 14.5 The excitation signal in Example 14.2: (a) full waveform; (b) 30-millisecond segment.

Examination of Figure 14.5 reveals a change in pitch during the syllable "why." The pitch frequency is the lowest in the beginning and increases gradually. We find, by detecting the epochs in the $u[n]$ waveform and computing the time intervals between them, that the pitch frequency is 170 Hz in the beginning and increases linearly to 260 Hz at the end. □

14.2.2 Modeling of the Excitation Signal

A simple way of modeling the excitation signal $u[n]$ is as a perfectly periodic impulse train during the frame. The pitch frequency and the amplitude of the impulses need to be estimated from $u[n]$. This modeling ignores pitch and amplitude variations, and assigns zero values to $u[n]$ between the impulses. It was the common way of speech compression and generation in older systems (mainly during the 1980s). Speech reconstructed from a perfectly periodic impulse train excitation sounds metallic. Therefore, LPC modeling was considered a low-quality compression technique during that time. In recent years, more sophisticated approaches have been developed for modeling of the excitation signal. These methods have considerably improved the quality of LPC modeling, and this technique is used today for many commercial applications. Here we describe a modeling method of the excitation signal based on linear prediction.

A perfectly periodic impulse train starting at $n = 0$, extending to infinity, and having unity amplitude and period M has the z-transform

$$U^z(z) = \sum_{n=0}^{\infty} z^{-Mn} = \frac{1}{1 - z^{-M}}. \tag{14.10}$$

Ideally, $u[n]$ can be generated as the response of the linear time-invariant system $G^z(z) = 1/(1 - z^{-M})$ to a single impulse at $n = 0$, that is,

$$u[n] = u[n - M] + \delta[n]. \tag{14.11}$$

The model (14.11) is not adequate, however, for representing the actual excitation signal. First, this model is not stable, since its poles are *on* the unit circle. Second, in reality $u[n]$ is not perfectly periodic, the amplitude of the impulses is not constant, and $u[n]$ assumes nonzero values between the impulses.

A model that is conceptually derived from (14.11) but better represents the true excitation signal is

$$u[n] = \eta u[n - M] + e[n], \tag{14.12}$$

where $0 < \eta < 1$. We can interpret (14.12) as a linear prediction of $u[n]$ from $u[n-M]$, where $e[n]$ serves as a correction term. We call $e[n]$ the *secondary excitation signal.* In contrast with the LPC model (14.9), which predicts $y[n]$ from its immediate past values, here we predict $u[n]$ from a value M points in the past. Therefore, it is common to refer to (14.12) as *long-term prediction.*

The long-term prediction model (14.12) is good if the energy of $e[n]$ is small. Therefore, we can compute η by requiring that $\sum_n (e[n])^2$ be minimum. We have

$$\sum_n (e[n])^2 = \sum_n (u[n] - \eta u[n - M])^2. \tag{14.13}$$

Differentiating with respect to η, equating the derivative to zero, and solving for η gives

$$\eta = \frac{\sum_n u[n]u[n - M]}{\sum_n (u[n - M])^2}. \tag{14.14}$$

In practice, we do not know the period M exactly. Furthermore, M is not even well defined, since the period is not exactly constant, even during short intervals. The solution is to compute η from (14.14) for all $M_{\min} \le M \le M_{\max}$, where the upper and lower limits on M are determined from the lower and upper limits on the pitch frequency, respectively. It can be shown that the value of η closest to 1 yields the minimum value of $\sum_n (e[n])^2$. All integer values of M in the selected range must be examined, since η is relatively sensitive to small deviations of M from its optimal value. The computation should be carried out for every frame, where the frame can be the same as the LPC frame or shorter. However, large values of M typically involve values of $u[n]$ across frame boundaries.

Example 14.2 (continued) We illustrate the result of modeling the excitation signal $u[n]$ obtained from the speech signal in the file `female-w.mat`. We use the same frame length as for the LPC modeling, that is, 160 samples. The values of η and M found in the 20 frames are

$$\begin{aligned} M &= 67, 65, 63, 63, 62, 62, 60, 59, 58, 57, 57, 56, 55, 53, 52, 51, 49, 47, 45, 43; \\ \eta &= 0.37, 0.68, 0.70, 0.72, 0.67, 0.73, 0.80, 0.94, 0.79, 0.75, 0.90, 0.92, 0.81, \\ &\quad\; 0.82, 0.94, 0.89, 0.56, 0.70, 0.83, 0.85. \end{aligned}$$

As we see, the optimal M decreases gradually, thereby tracking the increasing pitch frequency. The parameter η is close to 1 most of the time, but occasionally decreases.

Figure 14.6 shows the secondary excitation signal $e[n]$ obtained by the filtering operation

$$e[n] = u[n] - \eta u[n - M], \tag{14.15}$$

with η and M varying from frame to frame. As we see, the secondary excitation has average amplitude lower than that of the primary excitation, and it looks much more random. □

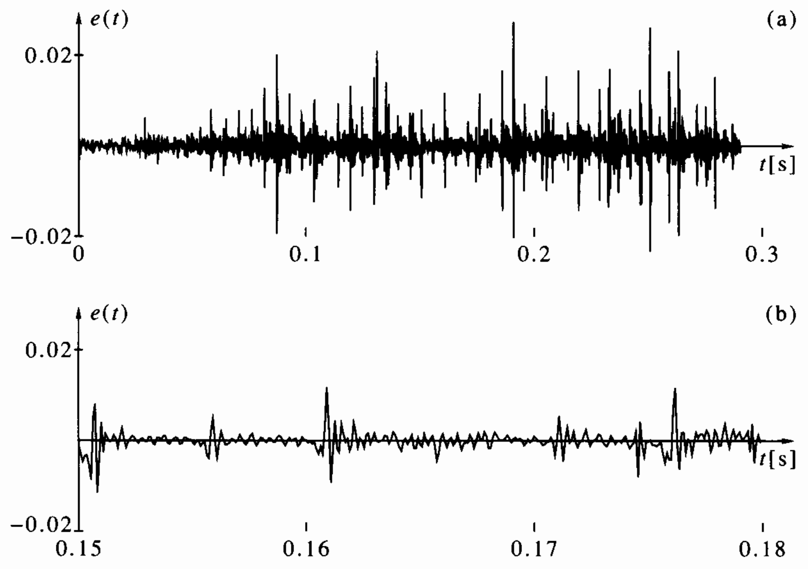

Figure 14.6 The secondary excitation signal in Example 14.2: (a) full waveform; (b) 30-millisecond segment.

14.2.3 Reconstruction of Modeled Speech

Reconstruction, or synthesis, of a speech signal is conceptually straightforward. Assuming that we are given the secondary waveform $e[n]$, the long-term prediction parameters η and M, and the LPC parameters $\{a_k\}$, the signal $y[n]$ can be reconstructed as shown in Figure 14.7. The filter $1/a(z)$ is guaranteed to be stable, because the roots of $a(z)$ are inside the unit circle. The filter $1/(1 - \eta z^{-M})$ is stable, because $0 < \eta < 1$.

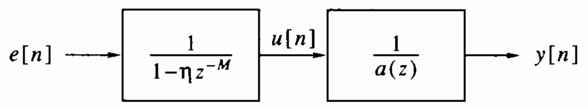

Figure 14.7 Speech reconstruction by LPC model and long-term prediction.

In implementing Figure 14.7, the following point should be kept in mind: Both reconstruction filters are IIR, and they are not time invariant, because their parameters are updated periodically (every frame). When the parameters are updated, it is necessary to preserve continuity of the state. It would be a gross mistake to initialize the state of the filters to zero at the beginning of every frame. Preserving the state of the LPC synthesis filter is straightforward. Preserving the state of the long-term synthesis filter is tricky, since the dimension of the state vector of this filter is M, and M varies from frame to frame. The solution is to keep a state vector whose dimension is the maximum value of M, say $M_{\max}$. The order of the corresponding IIR filter is then $M_{\max}$, but its coefficients are extremely sparse: Only one is nonzero at any given time. The location of the nonzero coefficient varies from frame to frame.

When a speech signal is reconstructed as in Figure 14.7, it will ideally reproduce the original signal. This happens when there is no compression. Compression distorts the parameters and the secondary excitation signal, hence the reconstructed speech; this is discussed next.

14.2.4 Coding and Compression

The term *coding* means a certain transformation of the model variables. The term *compression*, as we already know, means reduction of the bit rate. All the building blocks of the model are candidates for coding and compression. LPC modeling has been the basis for numerous compression schemes. The quality of speech reconstructed from its compressed model depends heavily on a successful choice of coding and compression schemes, and on the bit rate. Experience shows that the quality of LPC modeling is most sensitive to the way the excitation signal $u[n]$ is compressed and is less sensitive to compression of the LPC parameters. LPC parameters typically require 1.5 to 2.5 kilobits per second after compression and coding. Research on LPC in recent years has focused on the compression of the excitation signal.

Example 14.3 We give here a fairly detailed description of GSM, the compression standard used in the Pan-European Digital Mobile Radio System. It is based on LPC modeling followed by modeling of the excitation signal, as described previously. A major advantage of this standard is that the quality of the reconstructed speech depends little on the language or accent. This is necessary in Europe, where many languages and accents are used. By comparison, standards for North America are typically optimized to English, and the reconstructed speech suffers degradation when other languages or foreign accents are used.

The standard we describe is GSM 06.10, and is documented by the European body [GSM, 1991]. The input signal is sampled at 8 kHz, 13 bits, or 104 kilobits per second. The rate of the compressed speech is 13 kilobits per second, so the compression ratio is 8. The main features of the standard follow.

1. Offset compensation. The input signal is passed through the filter

$$G_1^z(z) = \frac{1 - z^{-1}}{1 - \alpha z^{-1}}, \quad \text{where} \quad \alpha = 32,735 \times 2^{-15} \approx 0.999. \tag{14.16}$$

   The purpose of this filter is to remove any DC component that may be present, without affecting the signal in any other way.

2. Pre-emphasis. The DC-free signal is passed through the filter

$$G_2^z(z) = 1 - \beta z^{-1}, \quad \text{where} \quad \beta = 28,180 \times 2^{-15} \approx 0.86. \tag{14.17}$$

   The purpose of this filter is to emphasize high frequencies, partly compensating for their low energy in the input signal. High-frequency pre-emphasis improves the quality of the LPC model. At the last stage of reconstruction de-emphasis of high frequencies is performed (the reverse of pre-emphasis).

3. The signal at the output of $G_2^z(z)$ is divided into frames of 160 samples each, or 20 milliseconds. The operations described next are performed for every frame.

4. The parameters $\{\kappa_y[k],\ 0 \le k \le 8\}$ are computed, as defined in (13.66). Thus, the order of the LPC model is $p = 8$.

5. Instead of computing the parameters a_k of the LPC model, the reflection coefficients $\{\rho_k,\ 1 \le k \le p\}$ are computed. This is performed by means of the Schur algorithm (13.64).

6. The parameters ρ_k are transformed to an equivalent set of parameters, called the *log-area ratio* (LAR), and defined as
$$\mathrm{LAR}[k] = \log_{10}\left(\frac{1+\rho_k}{1-\rho_k}\right). \tag{14.18}$$
This transformation helps to improve the quality of the parameters after quantization. In practice, an approximation to the log function is used.
7. The LAR parameters are quantized, using $7 - \lceil k/2 \rceil$ bits for $\mathrm{LAR}[k]$. Thus, the numbers of bits are 6, 6, 5, 5, 4, 4, 3, 3, or a total of 36 bits for the eight parameters. We skip the details of the quantization scheme.
8. The parameters ρ_k are reconstructed from the quantized LAR parameters, and used for filtering the speech signal and for computing the excitation signal $u[n]$. Filtering is performed using the IIR lattice discussed in Section 13.4.4 and shown in Figure 13.10.
9. The values of $u[n]$ in each frame are further divided into 4 subframes of length 40 each (or 5 milliseconds). All subsequent operations are performed on each subframe.
10. Long-term prediction is performed on $u[n]$ to determine the parameters M and η. In the GSM standard, the former is called *lag*, and the latter *gain*. The range of M is between 40 and 120, corresponding to pitch frequencies in the range 67 to 200 Hz. If the pitch is higher, a fraction of the pitch (or a multiple of the period) is detected. The algorithm is conceptually as described in Section 14.2.2, but differs in the details. The main difference is that instead of correlating $u[n]$ with its own past values, it is correlated with past values of the *reconstructed* signal. This leads to minimization of the energy of the error between the actual excitation and the predicted reconstructed excitation, which helps to improve the quality of the compressed signal.
11. The parameter M is coded in 7 bits and the parameter η is quantized to 2 bits. This requires 9 bits in each subframe.
12. The secondary excitation signal $e[n]$ is computed as in (14.15). In the GSM standard, this signal is called *regular pulse excitation* (RPE). Next it is necessary to compress this signal. Compression of $e[n]$ involves the following steps:
    (a) The signal is convolved with a tenth-order, low-pass FIR filter $G_3^z(z)$ whose coefficients, multiplied by 2^{13}, are
    $$g[0] = g[10] = -134,\ g[1] = g[9] = -374,\ g[2] = g[8] = 0,$$
    $$g[3] = g[7] = 2054,\ g[4] = g[6] = 5741,\ g[5] = 8192.$$
    Since the length of $e[n]$ is 40, the result of the convolution has length 50, but only the 40 middle terms are retained. We denote the 40 values thus obtained by $x[n]$. The purpose of the low-pass filter is to limit the bandwidth of the secondary excitation signal to about $\pi/3$.
    (b) The objective is now to decimate $x[n]$ by 3. This is done by considering the four decimated sequences
    $$x_l[i] = x[l+3i],\quad 0 \le l \le 3,\ 0 \le i \le 12.$$
    The value of l for which $\sum_{i=0}^{12}(x_l[i])^2$ is the largest is selected, and the corresponding subsequence $x_l[i]$ (of length 13) is retained. All other points of $x[n]$ are discarded.

(c) The parameter

$$x_{\max} = \max\{|x_l[i]|,\ 0 \le i \le 12\}$$

is computed.

(d) The integer l is coded in 2 bits, the parameter $x_{\max}$ is quantized to 6 bits, and each of $x_l[i]$ is quantized to 3 bits. Nonlinear quantization is used, the details of which we omit.

In summary, compression and coding of the secondary excitation signal results in 56 bits per subframe: 7 for M, 2 for η, 2 for l, 6 for $x_{\max}$, and 39 for the 13 values of $x_l[i]$. The total number of bits per frame is therefore 224. Adding to this the 36 bits used for the LPC modeling stage, we get 260 bits per frame, or 13 kilobits per second.

13. The encoder also contains most of what is needed in the decoder (at the receiving end). It reconstructs a distorted version of $e[n]$ by expanding $x_l[i]$, starting at the time point l and using simple zero filling for the missing points. It then reconstructs a distorted version of the primary excitation using the parameters M and η. This signal is used in the next subframe for the new values of M and η.
14. The decoder includes, in addition to the part that reconstructs the primary excitation, the LPC reconstruction filter and a de-emphasis filter whose transfer function is the inverse of $G_2^z(z)$.

In summary, the GSM standard is relatively simple, yet efficient and robust. It achieves mean opinion score (see Example 12.8) of about 3.5. Higher compression ratios can be achieved at this quality, but more complex algorithms are needed, hence more expensive hardware. □

14.3 Musical Signals

Audio signals generated by musical instruments typically have a harmonic structure. A simple mathematical model of a musical signal is

$$x(t) = a(t) \sum_{m=1}^{\infty} c_m \cos(2\pi m f_0 t + \phi_m), \tag{14.19}$$

where:

1. f_0 is the fundamental frequency, or the *pitch*.
2. c_m is the amplitude and ϕ_m is the phase of the mth harmonic.
3. $a(t)$ is the *envelope*. It is a low-frequency signal, which modulates the amplitude of the sound.

The frequency f_0 is determined by the note being played. In Western music, the frequencies of musical instruments obey a geometric series formula

$$f_i = f_{\text{ref}} q^i, \quad q = 2^{1/12}, \tag{14.20}$$

where f_{ref} is a reference frequency. The standard reference frequency is 440 Hz, and the corresponding note is called A (also known as *la*). The ratio $2^{1/12}$ is called a *semitone*. The notes B, C, D, E, F, G (also known as *ti, do, re, mi, fa, sol*, respectively) are 2, 3, 5, 7, 8, and 10 semitones above A. These are called the *natural notes*. Between A and B we have the note called A-sharp or B-flat, depending on the scale being used. Similarly, we have sharp and flat notes between C and D, D and E, F and G, and G and A. The note 12 semitones above A is again called A, and is said to be an *octave* higher. The ratio between two notes an octave apart is exactly 2. The seven natural notes A through G and their octaves are played by the white keys of the piano. The sharp and flat notes are played by the black keys. There are twelve semitones in an octave.

The *range* of a musical instrument is the set of notes it can play. This corresponds to the range of indices i in (14.20). For example, the range of a piano is $-48 \le i \le 39$. The corresponding frequencies are 27.5 Hz and 4186 Hz (the lowest note is A and the highest is C).

Western music is largely based on *diatonic scales.* A diatonic scale consists of 7 notes per octave. The diatonic C major scale consists of the natural notes C, D, E, F, G, A, B and their octaves. There are 12 major keys, each starting at a different semitone. The frequency ratios of the 7 notes of all major keys are identical. There are three kinds of minor key: natural, harmonic, and melodic. There are 12 keys of each of these kinds. The natural A minor scale consists of the natural notes A, B, C, D, E, F, G. In the harmonic A minor scale, the G is replaced by G-sharp. In the melodic A minor scale, the F and G is replaced by F-sharp and G-sharp, respectively, when the scale is ascending. These two notes revert to natural when the scale is descending.

The harmonic structure of a musical instrument is the series of amplitudes of the various harmonics relative to the fundamental frequency (measured in dB). The harmonic structure depends on the type of the instrument, the individual instrument, the note played, and the way it is played. Different instruments have their typical harmonic structures. It is the differences in harmonic structure, as well as the envelope, that make different instruments sound differently. The relative phases of the various harmonics are of little importance, since the human ear is almost insensitive to phase.

The envelope $a(t)$ is characteristic of the instrument, and also depends on the note and the way the note is played. For example, notes in bow instruments (such as violin or cello) and wind instruments (such as flute or horn) can be sustained for long times. Notes played on plucked string instruments (such as guitar) and keyboard instruments (such as piano) are limited in the time they can be sustained.

The model above does not represent all acoustic effects produced by musical instruments. For example, *vibrato* is a form of periodic frequency modulation around the nominal frequency of the note; *glissando* is a rapid succession of notes that sounds almost as a continuously varying pitch frequency.

Many instruments are capable of playing chords. A *chord* is a group of notes played either simultaneously or in a quick succession. For example, the C major chord consists of C, E, and G, and possibly a few of their octaves. The A minor chord consists of A, C, and E, and possibly a few of their octaves. When a chord is played, the signal can be described to a good approximation as superposition of the signals (14.19) of the individual notes.

Example 14.4 The files `cello.bin`, `guitar.bin`, `flute.bin`, and `frhorn.bin` contain about 1 second sounds of cello, classical guitar, flute, and French horn, respectively (see page vi for information on how to download these files). The note played is A in each case. The envelope waveforms of these four instruments are shown in Figure 14.8. As we see, the cello has a gradual rise of the amplitude, followed by a gradual decay. The guitar is characterized by a steep rise when the string is plucked, followed by a steep decay after it is released. The flute has a characteristic low-frequency amplitude modulation. The French horn has a steady amplitude during the time the note is played.

Figure 14.9 shows a 10-millisecond waveform segment of each instrument. As a comparison of the waveforms suggests, the pitch of the cello is 220 Hz, that of the guitar and the French horn is 440 Hz, and that of the flute is 880 Hz.

Figure 14.10 shows the magnitude DFT of a 186-millisecond segment (8192 samples) of each instrument. We plot only the frequency range 0 to 11 kHz, since the energy at higher frequencies is negligible. □

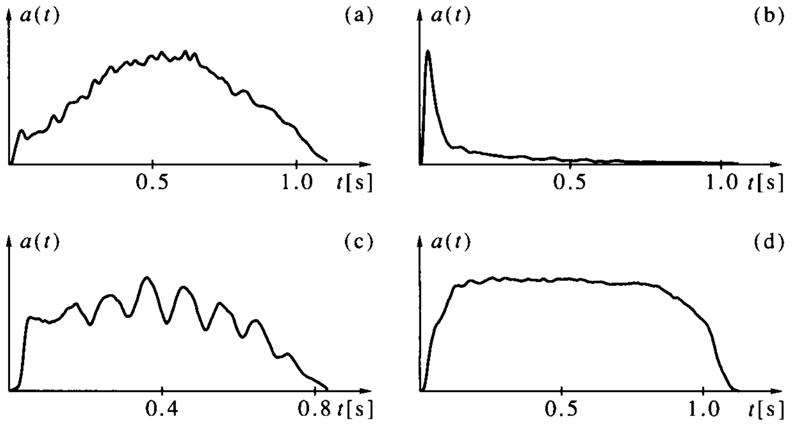

Figure 14.8 Envelope waveforms of musical instruments: (a) cello; (b) classical guitar; (c) flute; (d) French horn.

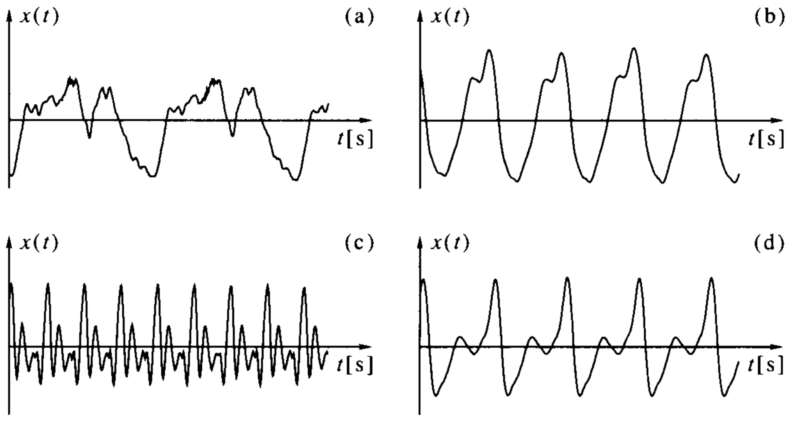

Figure 14.9 Waveforms of musical instruments, note played is A, 10-millisecond segments: (a) cello; (b) classical guitar; (c) flute; (d) French horn.

The model (14.19) can be applied for the synthesis of musical signals. To synthesize a note of a particular instrument, we need to know the characteristic envelope signal of the instrument and its characteristic harmonic structure. Chords are synthesized by superposition of individual notes. This method of synthesizing musical signals is called *additive synthesis*. It is relatively simple to implement, and you are encouraged to program it in MATLAB and test it, using the information in the waveforms of the four instruments. The musical quality of signals synthesized this way is not high, however, and present-day synthesis methods of higher quality have rendered the additive synthesis method all but obsolete.

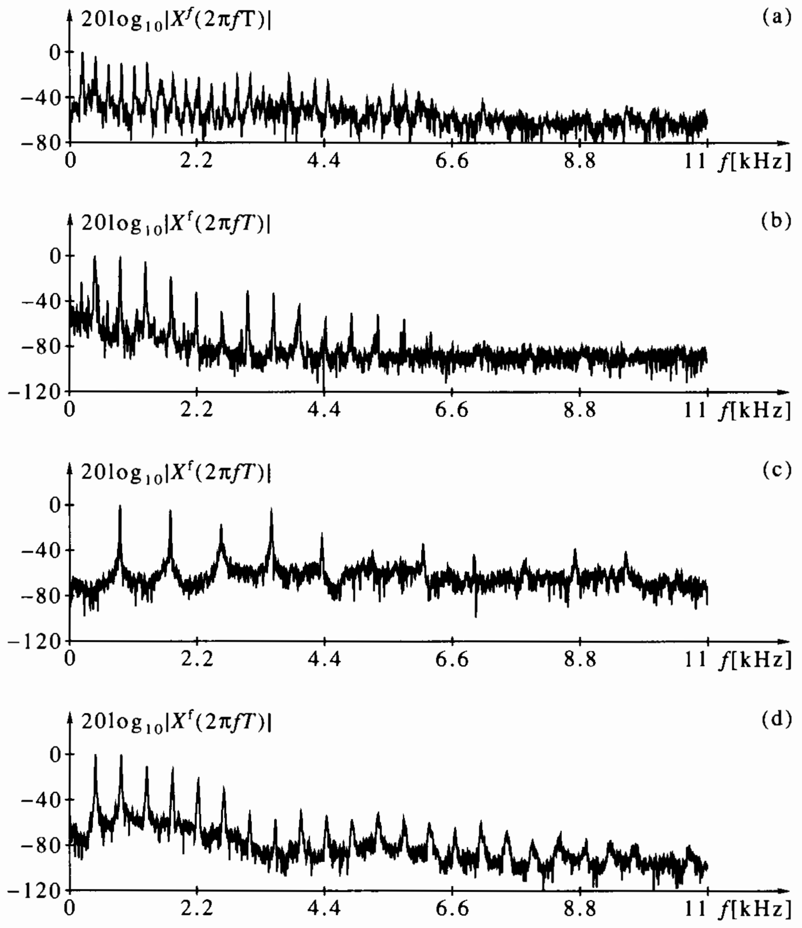

Figure 14.10 Spectra of musical instruments, note played is A, 186-millisecond segments: (a) cello; (b) classical guitar; (c) flute; (d) French horn.

14.4 An Application of DSP in Digital Communication

Digital communication has been in use for many years. However, until recently, the circuitry used for generation, transmission, and reception of digital communication signals was chiefly analog. Recent years have seen a growing trend of replacing analog functions needed in digital communications by digital algorithms, implemented on DSP microprocessors. Today this still applies mainly to base-band signals (either before modulation or after demodulation), because modulated signals are still too fast varying to be handled by digital means in most applications. In this book, we have already described several digital communication techniques in various examples and exercises. Here we describe, as an example of DSP application in communication systems, a reasonably complete system for receiving digital communication signals.

Since we have already met BPSK and OQPSK, we choose another common signaling method this time: frequency-shift keying (FSK). To make our example more interesting, we choose four-level FSK (rather than the simpler binary FSK). In four-level FSK we group the bits in pairs, and denote each pair by a *symbol*. For example, let us assign the symbols $-3, -1, 1, 3$ to the bit pairs $00, 01, 11, 10$, respectively. Then we allocate a frequency to each symbol, so we need four different frequencies. Suppose we wish to transmit at a rate f_0 symbols per second, which is the same as $2f_0$ bits per second. Then, the time interval allocated to each symbol is $T_0 = 1/f_0$. During each interval of length T_0, we transmit a signal at the frequency corresponding to the symbol in that interval. Thus, the frequency of the transmitted signal changes according to the transmitted symbols. The task of the receiver is to find, at each interval, which frequency was transmitted and to assign to it the corresponding symbol.

14.4.1 The Transmitted Signal

The *base-band signal* is piecewise constant, representing each symbol by a constant level. In the typical base-band signal shown in Figure 14.11, we have assigned levels $-3, -1, 1, 3$ to the different symbols. The mathematical description of the base-band signal is

$$s(t) = \sum_{m=-\infty}^{\infty} u[m]\text{rect}\left(\frac{t - mT_0}{T_0}\right), \tag{14.21}$$

where $u[m]$ is the sequence of symbols, and $\text{rect}(\cdot)$ is the rectangular function (2.32).

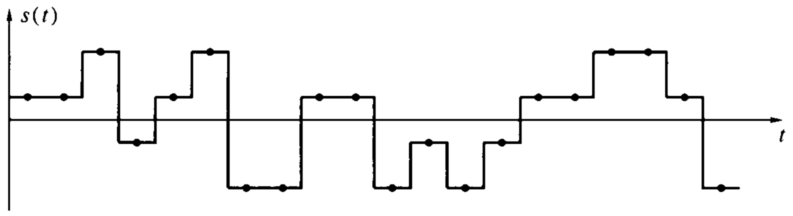

Figure 14.11 Modulating waveform of a four-level FSK signal.

Frequency modulation is defined by the mathematical operation

$$x(t) = \cos[2\pi f_c t + \phi(t)], \quad \text{where} \quad \phi(t) = 2\pi K_f \int_0^t s(\tau)d\tau, \tag{14.22}$$

f_c is the carrier frequency, and K_f is a constant. The *instantaneous frequency* of $x(t)$ is the time derivative of its instantaneous phase, divided by 2π, that is,

$$f(t) = f_c + K_f s(t). \tag{14.23}$$

The *peak frequency deviation* of an FSK signal is defined as the maximum value of $K_f|s(t)|$. In our case, since the maximum value of $|s(t)|$ is 3, the peak frequency deviation is $3K_f$. An important parameter of an FSK signal is the ratio between the peak frequency deviation and the symbol rate f_0. Denoting this ratio by β, we get that $K_f = f_0\beta/3$. Therefore,

$$f(t) = f_c + (\beta/3)f_0 s(t). \tag{14.24}$$

The parameter β controls the bandwidth of the modulated signal: the larger is β, the larger the bandwidth. The choice $\beta = 1.5$ has a special meaning. Recall that the separation between adjacent levels of $s(t)$ is 2. Therefore, the separation between the corresponding frequencies, with $\beta = 1.5$, is exactly f_0. Thus, during the symbol interval T_0, there is a difference of exactly one cycle between the numbers of cycles corresponding to adjacent symbols. We shall use this value of β in our system. Figure 14.12 shows the spectrum of the FSK signal $x(t)$ in this case. This figure was obtained by sampling the signal at a rate $16f_0$ and averaging 128 DFT magnitudes of 512 points each. We have plotted the spectrum around the carrier frequency f_c for convenience. Distinctive in this figure are the four spectral lines at frequencies $f_c \pm 0.5f_0$ and $f_c \pm 1.5f_0$. Otherwise, the spectrum is flat up to $\pm 1.5f_0$, with a sharp decay and relatively low side lobes at higher deviations from the carrier frequency. The bandwidth of $x(t)$ is about $\pm 2f_0$ around the carrier frequency.

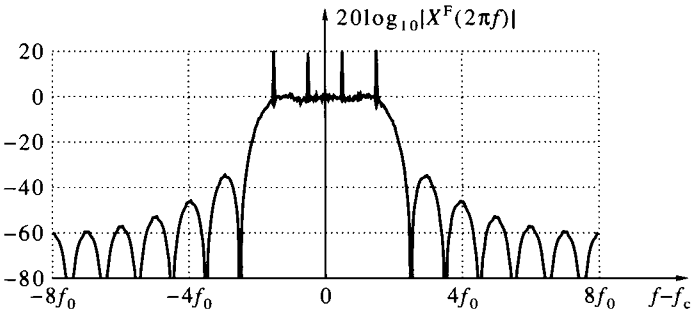

Figure 14.12 Spectrum of a four-level FSK signal.

14.4.2 The Received Signal

The received signal is delayed with respect to the transmitted signal, due to the finite propagation time, and is shifted by an unknown phase; it is given by

$$r(t) = \cos[2\pi f_c(t - t_0) + \phi(t - t_0) + \phi_0'] = \cos[2\pi f_c t + \phi(t - t_0) + \phi_0], \quad (14.25)$$

where t_0 is an unknown delay and $\phi_0 = \phi_0' - 2\pi f_c t_0$ is an unknown phase. The delay is constant if both transmitter and receiver are stationary, but it can change over time if there is relative motion between them. For example, if the receiver is located on a car moving at 30 m/s then, since the speed of light is 3×10^8 m/s, t_0 will change at a rate up to 0.1 microsecond per second. If the transmitter is located on a satellite moving at 7000 m/s, the rate of change of t_0 can be over 20 microseconds per second. The received signal also includes a noise component. However, for the time being we ignore the noise; we shall consider its effect when we examine the performance of the receiver.

If we were to build an analog receiver, we would construct a circuit for extracting the instantaneous frequency from $r(t)$. Such a circuit is called an *FM discriminator*. Since we aim at a digital receiver, we shall not use an analog FM discriminator. However, we still need to convert the signal to a sufficiently low frequency, to process it digitally. This is done by simple DSB demodulation, as explained in Problem 3.43. The result is a *low-IF* signal. We explained the concepts of IF and superheterodyne in Example 3.12. In the present case we call it low-IF, because it is close to the base band.

Assuming that we are going to use a sampling rate Lf_0 for the A/D converter (where L is an integer), it is convenient to choose the IF frequency as $0.25Lf_0$. The frequency of the local oscillator should then be $f_c - 0.25Lf_0$. However, physical oscillators are never completely accurate; an error of a few parts per million is typical. Since typical values of f_c are in the hundreds-of-megahertz range, a relative error of 10^{-6} in the local oscillator amounts to a few hundred hertz. This is not negligible with respect to f_0, which is typically in kilohertz. Therefore, we shall later have to face the problem of estimating the center frequency of the received signal relative to the receiver reference frequency. Let us therefore express the demodulated signal as

$$y(t) = \cos[2\pi(0.25Lf_0 + \Delta f)t + \phi(t - t_0) + \phi_0], \tag{14.26}$$

where Δf is the *carrier offset*, resulting from the frequency error. The demodulated signal is passed through an analog antialiasing filter, and from it to the A/D. The discrete-time low-IF signal is thus

$$\begin{aligned} y[n] = y(n/Lf_0) &= \cos[2\pi(0.25Lf_0 + \Delta f)(n/Lf_0) + \phi(n/Lf_0 - t_0) + \phi_0] \\ &= \cos[2\pi(0.25 + \Delta f/Lf_0)n + \phi(n/Lf_0 - t_0) + \phi_0]. \end{aligned} \tag{14.27}$$

14.4.3 Choosing the Sampling Rate

We now choose the sampling rate of the digital receiver. We have already constrained the rate to be an integer multiple of the frequency f_0, so it remains to determine the factor L. To do this, we must assume an upper limit on $|\Delta f|$, say $\Delta f_{\max}$.

Recall that the bandwidth of the FSK signal is $\pm 2f_0$ around its center frequency, so after sampling it becomes $\pm 2/L$ (in units of $\theta/2\pi$). Since the center frequency is $0.25 + \Delta f/Lf_0$, the signal occupies the band

$$\left[0.25 + \frac{\Delta f}{Lf_0} - \frac{2}{L}, 0.25 + \frac{\Delta f}{Lf_0} + \frac{2}{L}\right].$$

The following constraints must be imposed:

1. Since the signal $y[n]$ is real, its spectrum is two sided. Therefore, when $\Delta f = -\Delta f_{\max}$, the lower band-edge frequency must be greater than zero, for otherwise the positive and negative frequency bands will overlap and distort the spectral contents. This leads to the constraint

$$0.25 - \frac{\Delta f_{\max}}{Lf_0} - \frac{2}{L} > 0. \tag{14.28}$$

2. When $\Delta f = \Delta f_{\max}$, the higher band-edge frequency must be less than 0.5, for otherwise we will get aliasing. This leads to the constraint

$$0.25 + \frac{\Delta f_{\max}}{Lf_0} + \frac{2}{L} < 0.5. \tag{14.29}$$

When (14.28) and (14.29) are solved for L, they both yield

$$L > 4\left(2 + \frac{\Delta f_{\max}}{f_0}\right). \tag{14.30}$$

We already see one advantage of choosing a nominal IF frequency of $0.25Lf_0$: This way, both constraints lead to the same lower limit on L. Later we shall see another advantage of this choice.

To be specific, let us assume that $\Delta f_{\max} = f_0$. Then we get from (14.30) that $L > 12$. A convenient number is $L = 16$, and this will be our choice. With this value of L, the signal occupies the band $[1/16, 5/16]$ when $\Delta f = -\Delta f_{\max}$, or the band $[3/16, 7/16]$ when $\Delta f = \Delta f_{\max}$.

14.4.4 Quadrature Signal Generation

The information about the symbols is in the time derivative of the phase function of $y(t)$. Symbol detection is complicated by the presence of the three unknown parameters: the phase ϕ_0, the delay t_0, and the carrier offset Δf. A common way of making the detector insensitive to unknown phase is to generate the *quadrature signal*

$$y_q[n] = \sin[2\pi(0.25 + \Delta f/Lf_0)n + \phi(n/Lf_0 - t_0) + \phi_0]. \tag{14.31}$$

The signal $y[n]$ itself is called the *in-phase* signal, and the pair $\{y[n], y_q[n]\}$ is called the I-Q signals. We explain later how the availability of I-Q signals overcomes the problem of unknown phase.

The signal $y_q[n]$ can be generated by passing $y[n]$ through a Hilbert transformer. Therefore, the first component of our digital receiver will be a digital Hilbert transformer. To design a digital Hilbert transformer, we need to define the parity of the order (even or odd), the pass-band width, and the pass-band ripple. Note that a Hilbert transformer does not have stop bands, but it does have two transition bands, around $\theta = 0$ and $\theta = \pi$.

Since we are going to use $\{y[n], y_q[n]\}$ as a pair, and since $y_q[n]$ is going to be delayed by the group delay of the Hilbert transformer, we must delay the in-phase signal $y[n]$ by the same amount. Choosing an even order makes the group delay integer, so $y[n]$ must be delayed by an integer number of samples, and this is a simple operation. On the other hand, choosing an odd order makes the group delay fractional, and then we must delay $y[n]$ by an interpolated-delay filter. Therefore, we use a Hilbert transformer of an even order.

To determine the width of the pass band, recall the analysis in Section 14.4.3. There we concluded that the lowest frequency of $y[n]$ when $\Delta f = -\Delta f_{\max}$ is $1/16$ of the sampling rate, and the highest frequency when $\Delta f = \Delta f_{\max}$ is $7/16$ of the sampling rate. To accommodate any value of Δf within the specified range, we choose the pass band as $[\pi/8, 7\pi/8]$.

A typical pass-band ripple δ_p for our application is between 0.01 and 0.001. The former is marginal, whereas the latter is conservative. Designing a Hilbert transformer by the Parks-McClellan algorithm, we find that an order $N = 30$ is the minimum for obtaining $\delta_p = 0.01$, whereas $N = 38$ gives $\delta_p = 0.001$; here we choose $N = 38$.

14.4.5 Complex Demodulation

Having obtained the in-phase and quadrature components, we can describe them as a complex signal, as we did in Section 9.2.5:

$$y_a[n] = y[n] + jy_q[n] = \exp\{j[2\pi(0.25 + \Delta f/Lf_0)n + \phi(n/Lf_0 - t_0) + \phi_0]\}. \tag{14.32}$$

The signal $y_a[n]$ is the *analytic signal* of $y[n]$. We emphasize again the need to delay $y[n]$ by half the order of the Hilbert transformer (that is, by 19 in this case): Failing to do so will prevent the system from working properly.

We recall that the spectrum of the analytic signal occupies only positive frequencies. It is now convenient to shift the spectrum to the base band. Ideally we would like to shift by $2\pi(0.25 + \Delta f/Lf_0)$, and then the spectrum would be centered around zero. However, since Δf is unknown, we shift only by 0.5π. To perform this shift, we must multiply $y_a[n]$ by $e^{-j0.5\pi n}$, that is, to compute

$$z[n] = y_a[n]e^{-j0.5\pi n} = \exp\{j[2\pi(\Delta f/Lf_0)n + \phi(n/Lf_0 - t_0) + \phi_0]\}. \tag{14.33}$$

The operation (14.33) is called *complex demodulation.* We have $e^{-j0.5\pi n} = (-j)^n$, so

$$z[n] = z_r[n] + jz_i[n] = (y[n] + jy_q[n])(-j)^n.$$

This can be written as

$$z_r[n] = \begin{cases} y[n], & n \bmod 4 = 0, \\ y_q[n], & n \bmod 4 = 1, \\ -y[n], & n \bmod 4 = 2, \\ -y_q[n], & n \bmod 4 = 3, \end{cases} \quad z_i[n] = \begin{cases} y_q[n], & n \bmod 4 = 0, \\ -y[n], & n \bmod 4 = 1, \\ -y_q[n], & n \bmod 4 = 2, \\ y[n], & n \bmod 4 = 3. \end{cases} \tag{14.34}$$

As we see, complex demodulation requires no computations, only rearrangement of the data and sign reversals. This simplification occurs because we have chosen the low-IF frequency as 0.25 of the sampling rate.

14.4.6 Symbol Detection: Preliminary Discussion

The parts that we have discussed so far, the Hilbert transformer and the complex demodulator, are just the front end of the receiver. The main task, extracting the symbols from $z[n]$, is still ahead of us. This task is considerably complicated by the unknown time delay t_0 and the unknown carrier offset Δf. Before devising methods for tackling these difficulties, it is helpful to discuss how we would have performed symbol detection if these two parameters were zero. In such a case, the signal $z[n]$ during the mth symbol would be

$$z[n] = \exp\left\{j\left[2\pi\frac{0.5u[m](n-Lm)}{L} + \phi_0\right]\right\}, \quad Lm \le n \le Lm+L-1, \tag{14.35}$$

where $u[m]$ is the mth symbol. Since $u[m]$ can assume only four values, we can perform *matched filtering* to the four possible waveforms. By this we mean the following: We define four complex filters of length L each, and denote their impulse responses by $g_k[n]$, where $k = -3, -1, 1, 3$. The impulse responses are

$$g_k[n] = \exp\left[-j2\pi\frac{0.5k(L-1-n)}{L}\right], \quad 0 \le n \le L-1. \tag{14.36}$$

We convolve $z[n]$ with each of the four filters and sample the outputs at time $n = Lm + L - 1$ to get

$$\{z * g_k\}[Lm+L-1] = e^{j\phi_0}\sum_{l=0}^{L-1}\exp\left[j2\pi\frac{0.5(u[m]-k)(L-1-l)}{L}\right]. \tag{14.37}$$

Since $u[m] - k$ is an integer multiple of 2 for all possible values of k and $u[m]$, we get

$$\{z * g_k\}[Lm+L-1] = \begin{cases} Le^{j\phi_0}, & k = u[m], \\ 0, & \text{otherwise.} \end{cases} \tag{14.38}$$

Therefore, by observing the absolute values of $\{z * g_k\}[Lm+L-1]$ for the four filters, we can determine the symbol $u[m]$: It is the value of k for which the result is not zero. In practice, due to noise, none of the outputs will be zero, so we choose k for which the absolute value is the largest of the four. By taking the absolute value, we eliminate the unknown phase ϕ_0.

If you have solved Problem 8.21, you know that matched filtering is optimal, in the sense of maximizing the signal-to-noise ratio at the filter output when white noise is added to the input signal. Therefore, the matched filtering scheme is the best for symbol detection, provided we have zero (or perfectly known) carrier offset and delay.

If the unknown delay and carrier offset are nonzero, (14.38) will not hold any more. All four outputs will be nonzero in general, and we may well get that the largest absolute value is not at the right k. Reliable detection in the presence of carrier and timing offsets is a major problem in digital communication. We must devise a way of estimating the unknown frequency and delay parameters before we can use the matched filters.

14.4.7 FM to AM Conversion

One way of estimating the carrier and timing offsets is to extract the derivative of the phase function $\phi(n/Lf_0 - t_0)$ from $z[n]$ and use the derivative for estimating these two parameters. Extraction of the phase derivative is called *FM-to-AM conversion*, or *FM discrimination*. Carrier and timing estimation by FM discrimination is not necessarily the best approach (from accuracy point of view), but it is simple and convenient. Besides, there is a lot to learn from seeing how such a scheme works, and our aim here is to teach. Therefore, we now build a digital FM discriminator.

If a continuous-time complex signal $z(t)$ were available, we could have extracted the phase derivative as follows. Since

$$z(t) = \exp\{j[2\pi\Delta f t + \phi(t-t_0) + \phi_0]\}, \tag{14.39}$$

we have

$$\frac{dz(t)}{dt} = j\left[2\pi\Delta f + \frac{d\phi(t-t_0)}{dt}\right]z(t). \tag{14.40}$$

Therefore,

$$2\pi\Delta f + \frac{d\phi(t-t_0)}{dt} = -j\frac{dz(t)}{dt}\frac{\bar{z}(t)}{|z(t)|^2}. \tag{14.41}$$

Division by $|z(t)|^2$ is necessary here, since $z(t)$ has nonunity amplitude in general. Until now we ignored the signal amplitude, because it was immaterial; now we must account for it.

To approximate (14.41) using the discrete-time signal, we must approximate the differentiation. In Section 9.2.4 we learned how to build digital differentiators. Unfortunately, those differentiators are of little use here. The reason is that $d\phi(t-t_0)/dt$ is discontinuous at points where the symbol changes value; therefore, a differentiator with a long memory (i.e., a filter with a long impulse response) will lead to significant distortions. A simple alternative is suggested by noting that

$$\frac{z[n]\bar{z}[n-1]}{|z[n]\bar{z}[n-1]|} = \exp\left\{j\left[\frac{2\pi\Delta f}{Lf_0} + \phi\left(\frac{n}{Lf_0} - t_0\right) - \phi\left(\frac{n-1}{Lf_0} - t_0\right)\right]\right\}, \tag{14.42}$$

therefore,

$$\arcsin\left(\frac{\Im\{z[n]\bar{z}[n-1]\}}{|z[n]\bar{z}[n-1]|}\right) = \frac{2\pi\Delta f}{Lf_0} + \phi\left(\frac{n}{Lf_0} - t_0\right) - \phi\left(\frac{n-1}{Lf_0} - t_0\right). \tag{14.43}$$

We have from (14.22),

$$\phi\left(\frac{n}{Lf_0} - t_0\right) - \phi\left(\frac{n-1}{Lf_0} - t_0\right) = \pi f_0 \int_{(n-1)/Lf_0 - t_0}^{n/Lf_0 - t_0} s(\tau)d\tau = \frac{\pi u[m]}{L} \tag{14.44}$$

whenever the interval $[(n-1)/Lf_0 - t_0, n/Lf_0 - t_0]$ falls within the mth symbol. If there is a symbol change in this interval, there is a discontinuity and the left side of (14.44) assumes a different value. We finally get from (14.43) and (14.44),

$$u[m] + \frac{2\Delta f}{f_0} = \frac{L}{\pi}\arcsin\left(\frac{\Im\{z[n]\bar{z}[n-1]\}}{|z[n]\bar{z}[n-1]|}\right), \tag{14.45}$$

except at the discontinuity points.

We denote the discrete-time signal defined by the right side of (14.45) as $\hat{s}[n]$, because it represents an estimate of $s(t - t_0)$, except for the additive term $2\Delta f/f_0$. The signal $\hat{s}[n]$ defines our digital FM discriminator.

Figure 14.13 shows the waveform of $\hat{s}[n]$ at the output of the FM discriminator, corresponding to the waveform $s(t)$ shown in Figure 14.11. Here we compensated for the delay t_0 to enable comparison between the waveforms $\hat{s}[n]$ and $s(t)$. Part a shows the result when there is no noise. As we see, even in this case $\hat{s}[n]$ is not identical to $s(t)$, for two reasons: the nonideality of the Hilbert transformer and the transients caused by the approximate differentiator (14.45) at the discontinuity points. Part b shows the result when there is white noise at SNR of 12 dB at the output of the A/D. As we see, reliable detection of the transmitted symbols is not likely to be easy in this case.

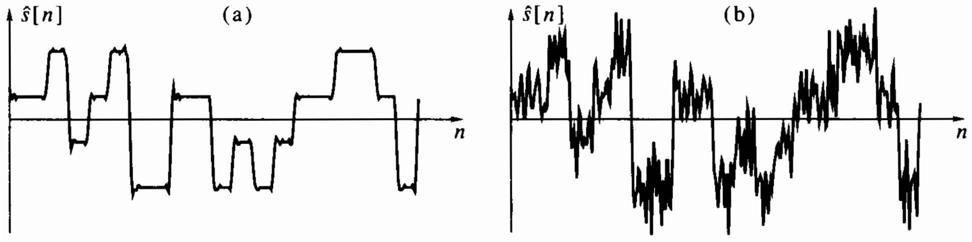

Figure 14.13 Output signal of the FM discriminator: (a) infinite SNR; (b) SNR = 12 dB.

14.4.8 Timing Recovery

Timing recovery means identifying the transition instants between symbols. Since the signal is sampled L times per symbol, our goal is to identify the transition instants up to the nearest sample. This will leave us with timing error that can be up to $\pm 0.5T_0/L$. It is possible to reduce the timing error below this value, using interpolation, but this complicates the system and we shall not attempt it here.

If the signal $\hat{s}[n]$ were clean, as in Figure 14.13(a), we could obtain the transition instants from the peaks of the magnitude-difference signal $|\hat{s}[n] - \hat{s}[n-1]|$ [which is approximately proportional to the magnitude of the derivative of $s(t)$]. However, when $\hat{s}[n]$ is noisy, as in Figure14.13(b), we must filter $\hat{s}[n]$ before we can use it for timing recovery.

A simple and effective timing recovery circuit is shown in the upper part of Figure 14.14. This circuit also contains a part related to symbol detection (lower part of the figure). We now explain the timing recovery part, then proceed to the symbol detection part.

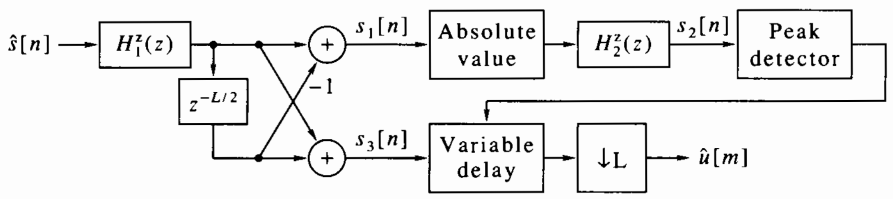

Figure 14.14 Timing recovery and matched filtering.

1. The transfer function $H_1^z(z)$ is
$$H_1^z(z) = 1 + z^{-1} + \cdots + z^{-(0.5L-1)}. \tag{14.46}$$
Let $s_1[n]$ denote the output of the subtractor in the upper line of Figure 14.14. Then, as we see from the figure, the transfer function from $\hat{s}[n]$ to $s_1[n]$ is
$$\begin{aligned} H^z(z) &= H_1^z(z)[1 - z^{-0.5L}] \\ &= 1 + z^{-1} + \cdots + z^{-(0.5L-1)} - z^{-0.5L} - z^{-(0.5L+1)} - \cdots - z^{-(L-1)}. \end{aligned} \tag{14.47}$$
Therefore, $s_1[n]$ is related to $\hat{s}[n]$ by
$$s_1[n] = \sum_{k=0}^{0.5L-1} \hat{s}[n-k] - \sum_{k=0.5L}^{L-1} \hat{s}[n-k]. \tag{14.48}$$
Suppose that a symbol change has occurred in the interval between n_0T_0 and $(n_0 + 1)T_0$, and the symbol values were $u[m-1]$ and $u[m]$ before and after the change, respectively. Then, assuming that $\hat{s}[n]$ is identical to $s[n]$ (except for the unknown delay), we get from (14.48) that
$$s_1[n_0] = 0,\ s_1[n_0 + 0.5L] = 0.5L(u[m] - u[m-1]),\ s_1[n_0 + L] = 0. \tag{14.49}$$
Also, $s_1[n]$ changes linearly between n_0 and $n_0 + 0.5L$, and between $n_0 + 0.5L + 1$ and $n_0 + L$. Figure 14.15 illustrates the operation of the filter $H^z(z)$ on the sequence $\hat{s}[n]$. We conclude that $s_1[n]$ exhibits a local extremum each time there is a change in the symbol value. The extremum occurs $0.5L$ time points after the change; it is positive if $u[m] > u[m-1]$ and negative otherwise.

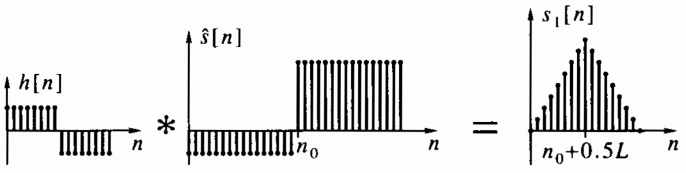

Figure 14.15 Operation of the early-late gate filter.

The filter $H^z(z)$ is a discrete-time version of an *early-late gate synchronizer.* This filter estimates the center of gravity of the transition area. In case of an ideal signal, the center of gravity coincides with the transition instant. In reality, $\hat{s}[n]$ is different from $s[n]$ due to noise and the nonideal operation of the FM discriminator. Then each of the two halves of the filter (the early and the late gates) attenuates the noise in the corresponding symbol by averaging over $0.5L$ samples. The filter $H_1^z(z)$ can be implemented efficiently by noting that
$$H_1^z(z) = \frac{1 - z^{-0.5L}}{1 - z^{-1}}. \tag{14.50}$$
The difference equation corresponding to (14.50) requires only two additions and no multiplications per time point; see Problem 11.4.

2. The transition instants can be found, in principle, by finding the local maxima of $|s_1[n]|$. Ideally, these maxima will be exactly L samples apart. However, when two adjacent symbols are equal, there is no local maximum. Also, noise may cause the points of local maximum to move randomly from their true locations, a phenomenon known as *timing jitter.* It is therefore desirable to filter the timing jitter by averaging the estimated transition instants over time. This operation is accomplished by the filter $H_2^z(z)$.

The desired impulse response of $H_2^z(z)$ is a train of impulses spaced L samples apart, that is,

$$h_2[n] = \sum_{k=0}^{\infty} \delta[n - kL]. \tag{14.51}$$

When such an impulse train is convolved with $|s_1[n]|$, it will exhibit peaks at the peaks of $|s_1[n]|$, but it will also perform averaging when $|s_1[n]|$ is subject to timing jitter. The problem with $h_2[n]$ is that its memory is too strong, since it averages an ever-increasing number of peaks. In practice, we want $h_2[n]$ to forget the past gradually, since the delay t_0 may vary slowly because of changes in the distance between the transmitter and the receiver (if either or both are in motion). The sequence

$$h_2[n] = \sum_{k=0}^{\infty} \alpha^k \delta[n - kL], \quad 0 < \alpha < 1 \tag{14.52}$$

is a decaying impulse train. When this sequence is convolved with $|s_1[n]|$, it attenuates past peaks exponentially. The closer α to 1, the longer is the memory of the filter. The transfer function corresponding to (14.52) is

$$H_2^z(z) = \frac{1}{1 - \alpha z^{-L}}. \tag{14.53}$$

This filter requires one multiplication and one addition per sample.

3. The output of $H_2^z(z)$, denoted by $s_2[n]$, has its local peaks spaced L samples apart most of the time. This signal is passed to a peak detector, which is responsible for finding these peaks. The peak detector operates as follows. Initially it finds the largest value among the last L consecutive points of $s_2[n]$ and marks the time of this peak. It then idles for $L - M - 1$ samples, where M is a small number, typically 1 or 2. Then it examines $2M + 1$ consecutive values of $s_2[n]$. In most cases, it finds the maximum at the $(M + 1)$st (i.e., the middle) point, which is L samples after the preceding peak. The same cycle then repeats itself continuously. Occasionally, the maximum will move a sample or two forward or backward. If this happens due to noise, it will usually correct itself later. If the true delay has changed by a physical motion, the peak detector will start yielding the new transition instants. The output of the peak detector—the sequence of estimated transition instants—is passed to the matched filter, to be discussed next.

Figure 14.16 shows typical waveforms of the signals in the timing recovery circuit. Part a shows the signal $s_1[n]$ when there is no noise, whereas part b shows this signal when there is noise at SNR of 6 dB. Part c shows the signal $s_2[n]$ when there is no noise, whereas part d shows this signal when there is noise at SNR of 6 dB. The signal $s_2[n]$ is shifted to the left by $0.5L$ samples, so its peaks indicate the true transition instants. We shall discuss parts e and f in Section 14.4.9.

14.4.9 Matched Filtering

The signal $s[n]$ has the property that its waveform is the same for all four symbol values; only its amplitude depends on the symbol. Therefore, unlike the four matched filters discussed in Section 14.4.6, we only need one matched filter for $\hat{s}[n]$. The symbol is then detected based on the amplitude of the matched filter output.

Since the level of $s[n]$ is constant during each symbol, the matched filter is a rectangular window of length L, that is,

$$G^z(z) = 1 + z^{-1} + \cdots + z^{-(L-1)}. \tag{14.54}$$

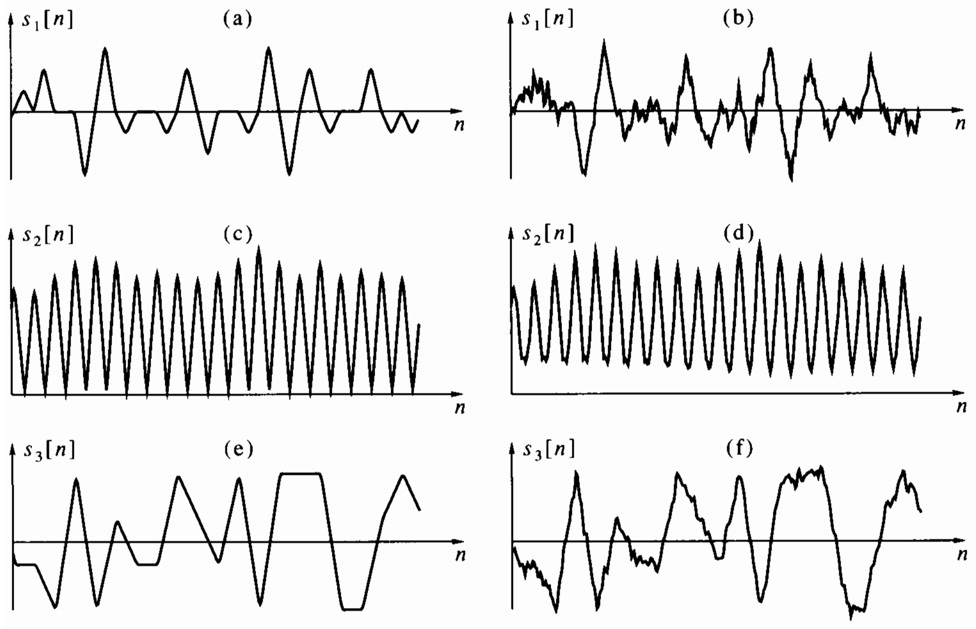

Figure 14.16 Waveforms in the timing recovery and matched filter circuits: (a) the signal $s_1[n]$ at infinite SNR; (b) the signal $s_1[n]$ at SNR = 6 dB; (c) the signal $s_2[n]$ at infinite SNR; (d) the signal $s_2[n]$ at SNR = 6 dB; (e) the signal $s_3[n]$ at infinite SNR; (f) the signal $s_3[n]$ at SNR = 6 dB.

Recalling the definition (14.46) of $H_{\mathrm{I}}^{z}(z)$, we see that

$$G^{z}(z) = H_{\mathrm{I}}^{z}(z)[1 + z^{-0.5L}]. \tag{14.55}$$

The response of $G^{z}(z)$ to $\hat{s}[n]$, to be denoted by $s_3[n]$, is generated at the output of the bottom adder shown in Figure 14.14.

The proper time to sample the output of the matched filter is at the end of every symbol. This happens $0.5L$ samples after every peak of $s_2[n]$. The purpose of the variable delay block is to perform this delay, relative to the current estimate of the transition instant. The delayed signal $s_3[n]$ is then sampled once every L time points, and the resulting signal $\hat{u}[m]$ is an estimate of the mth symbol (up to a constant scale factor L). Figure 14.16(e) shows the waveform of $s_3[n]$ when there is no noise, whereas Figure 14.16(f) shows this signal when there is noise at SNR of 6 dB.

14.4.10 Carrier Recovery and Symbol Detection

The estimated symbol $\hat{u}[m]$ is, except for noise, equal to $u[m] + 2\Delta f/f_0$, as seen from (14.45). Therefore, the frequency error Δf appears as a DC term. If we remove this DC term, we will be able to detect the symbol $u[m]$ as the number closest to $\hat{u}[m]$ in the set $\{-3, -1, 1, 3\}$. The carrier recovery and symbol detection algorithm we now describe has two modes of operation: an *initialization* mode and a *decision-directed* mode.

1. **Initialization mode:** We collect a set of M consecutive samples of $\hat{u}[m]$, where M is fairly large (say 100 or more). If the transmitted symbols are sufficiently random, the average of the true values of $u[m]$ must be nearly zero. We therefore

estimate Δf by

$$\widehat{\Delta f} = \frac{f_0}{2M} \sum_{m=1}^{M} \hat{u}[m]. \tag{14.56}$$

We now form the corrected sequence

$$u_1[m] = \hat{u}[m] - \frac{2\widehat{\Delta f}}{f_0} \tag{14.57}$$

and use it for detecting the symbols according to the quantization rule

$$u_2[m] = 2\,\text{round}\{0.5u_1[m] + 1.5\} - 3 \tag{14.58a}$$

$$u_3[m] = \max\{\min\{u_2[m], 3\}, -3\}. \tag{14.58b}$$

We see that $u_3[m]$ is the number closest to $u_1[m]$ in the set $\{-3, -1, 1, 3\}$, so it serves as the estimated value of $u[m]$. We now refine the estimated frequency error by replacing (14.56) with

$$\widehat{\Delta f} = \frac{f_0}{2M} \sum_{m=1}^{M} (\hat{u}[m] - u_3[m]). \tag{14.59}$$

2. **Decision-directed mode:** Each time a new value of $\hat{u}[m]$ is generated, we have new information about Δf. This information can be used for updating the estimated value of Δf. If we do this for every m, we get a sequence $\widehat{\Delta f}[m]$ of estimates. To understand the principle of operation of the decision-directed mode, we need to carry the following mathematical derivation. Let

$$\hat{u}[m] = u[m] + \frac{2\Delta f}{f_0} + v[m], \tag{14.60}$$

where $v[m]$ is the additive noise. If we generate $u_1[m]$ using (14.57b) with $\widehat{\Delta f}[m-1]$, the most recently available estimate of Δf, we get

$$u_1[m] = \hat{u}[m] - \frac{2\widehat{\Delta f}[m-1]}{f_0} = u[m] + \frac{2(\Delta f - \widehat{\Delta f}[m-1])}{f_0} + v[m]. \tag{14.61}$$

If we now generate $u_3[m]$ using (14.58), we can assume that $u_3[m] = u[m]$ most of the time, so we can subtract $u_3[m]$ from $u_1[m]$ to obtain

$$u_1[m] - u_3[m] = \frac{2(\Delta f - \widehat{\Delta f}[m-1])}{f_0} + v[m]. \tag{14.62}$$

Therefore, $u_1[m] - u_3[m]$ is proportional to the error in the frequency estimate, up to the additive noise. We can use this error signal for updating $\widehat{\Delta f}[m]$ as

$$\frac{2\widehat{\Delta f}[m]}{f_0} = \frac{2\widehat{\Delta f}[m-1]}{f_0} + \eta(u_1[m] - u_3[m]), \tag{14.63}$$

where η is a positive small number. Substitution of (14.62) in (14.63) gives

$$\frac{2\widehat{\Delta f}[m]}{f_0} = (1-\eta)\frac{2\widehat{\Delta f}[m-1]}{f_0} + \eta\frac{2\Delta f}{f_0} + \eta v[n]. \tag{14.64}$$

As we see, the transfer functions from $2\Delta f/f_0$ and $v[n]$ to $2\widehat{\Delta f}[m]/f_0$ are both $\eta/[1-(1-\eta)z^{-1}]$. The DC gain of this transfer function is 1, so $\widehat{\Delta f}[m]$ approaches Δf in steady state. The noise gain is (see Problem 7.37)

$$\text{NG} = \frac{\eta^2}{1-(1-\eta)^2} = \frac{\eta}{2-\eta} \approx 0.5\eta. \tag{14.65}$$

Therefore, the noise is considerably attenuated if η is small.

Figure 14.17 shows the decision-directed system for carrier recovery. The state variable $\widehat{\Delta f}[m]$ is initialized to the value obtained by (14.59) during the initialization mode. The quantizer is as given in (14.58). Note that the system operates at the symbol rate, which is L times lower than the sampling rate of the receiver.

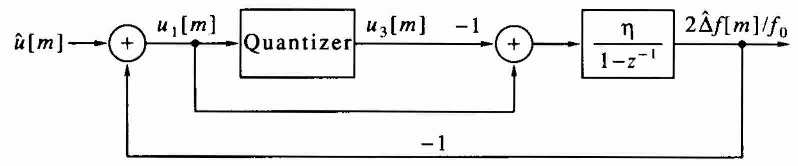

Figure 14.17 Decision-directed carrier recovery.

14.4.11 Improved Carrier Recovery and Symbol Detection

The decision-directed carrier recovery circuit shown in Figure 14.17 provides both the carrier offset and the symbols, so it appears as though the receiver is complete. However, the reliability of symbol detection of this circuit is dubious when the SNR is low. The main reason for this is the FM discriminator, which is, as we have seen, sensitive to both noise and discontinuities of the instantaneous frequency. If, however, the timing and carrier offset estimates provided by the circuits described previously are good enough, symbol detection can be greatly improved. Moreover, the carrier offset estimate can be improved as well. This is done by invoking the matched filters $g_k[n]$ introduced in Section 14.4.6. We make the following assumptions:

1. The timing recovery circuit described in Section 14.4.8 is accurate to within half the sampling interval.
2. The error $\delta f \triangleq \Delta f - \widehat{\Delta f}$, with $\widehat{\Delta f}$ obtained by the circuit described in Section 14.4.10, is a small fraction of the symbol rate f_0.

Let us demodulate the signal $z[n]$ given in (14.33) as follows:

$$z_1[n] \triangleq z[n]\exp\left\{-j2\pi\frac{\widehat{\Delta f}n}{Lf_0}\right\} = \exp\left\{j\left[2\pi\frac{\delta fn}{Lf_0} + \phi\left(\frac{n}{Lf_0}\right) + \phi_0\right]\right\}. \qquad (14.66)$$

Note that t_0 does not appear in (14.66), since we assume that timing offset has been compensated. During the mth symbol, $z_1[n]$ is given by [cf. (14.35)]

$$z_1[n] = \exp\left\{j\left[2\pi\frac{0.5u[m](n-Lm)}{L} + 2\pi\frac{\delta fn}{Lf_0} + \phi_0\right]\right\}, \quad Lm \le n \le Lm + L - 1. \qquad (14.67)$$

Let us convolve $z_1[n]$ with each of the four matched filters $g_k[n]$, defined in (14.36), and sample the outputs at time $n = Lm + L - 1$. We then get, similarly to (14.37),

$$\begin{aligned}
&\{z_1 * g_k\}[Lm + L - 1] \\
&= \sum_{l=0}^{L-1}\exp\left\{j2\pi\left[\frac{0.5(k-u[m])(L-1-l)}{L} + \frac{\delta f(Lm+L-1-l)}{Lf_0}\right] + j\phi_0\right\} \\
&= \exp\left\{j2\pi\left[\frac{\delta fm}{f_0} + \frac{0.25(k-u[m])(L-1)}{L} + \frac{0.5\delta f(L-1)}{Lf_0}\right] + j\phi_0\right\} \\
&\quad\cdot D\left(2\pi\left[\frac{0.5(k-u[m])}{L} + \frac{\delta f}{Lf_0}\right], L\right).
\end{aligned} \qquad (14.68)$$

For $k = u[m]$, the magnitude of the right side of (14.68) is $D(\delta f/Lf_0, L)$, which is nearly L if $|\delta f/f_0| \ll 1$. For $k \neq u[m]$, the argument of the Dirichlet kernel is in a side lobe, so the magnitude will be considerably less than L. Usually, therefore, unless the SNR is low, the magnitude of the matched filter corresponding to $k = u[m]$ will be the largest of the four. The conclusion is that for small timing and carrier errors, the matched filters $g_k[n]$ can be used for symbol detection, with only slight performance degradation with respect to the case of zero timing and carrier errors.

Considering (14.68) further, we see that the output of the matched filter of largest magnitude contains information about δf, which can be used for estimating this parameter. We have

$$\begin{aligned} z_2[m] &= \{z_1 * g_{u[m]}\}[Lm + L - 1] \\ &= D\left(\frac{2\pi\delta f}{Lf_0}, L\right) \exp\left\{j2\pi\left[\frac{\delta f m}{f_0} + \frac{0.5\delta f(L-1)}{Lf_0}\right] + j\phi_0\right\}. \end{aligned} \qquad (14.69)$$

The sequence $z_2[m]$ is obtained by taking, at each m, the complex output of the matched filter whose magnitude is the largest of the four. As we see, this sequence is a complex exponential in the discrete-time variable m, with frequency $\theta_0 = 2\pi\delta f/f_0$. Therefore, we can estimate δf from the DFT of N consecutive values of this sequence, as we learned in Section 6.5. The number N is not necessarily large; $N = 32$ or 64 is often sufficient. The estimate of δf is added to $\widehat{\Delta f}$ used for forming the signal $z_1[n]$. It is desirable, in most applications, to repeat the estimation of δf periodically, since the carrier offset may vary due to component aging and environmental conditions such as temperature and vibrations.

14.4.12 Summary

In this section we described a digital receiver for four-level FSK signals. The receiver consists of front end, FM discriminator, timing recovery circuit, carrier recovery circuit, and symbol detection circuit. The front end includes a Hilbert transformer and a complex demodulator. The FM discriminator extracts the real modulating signal from the complex frequency-modulated signal. The timing recovery circuit determines the symbol transition instants. The carrier recovery circuit estimates the carrier offset and compensates for it. Finally, the symbol detection circuit decides which symbol was transmitted at each interval of T_0 seconds. Symbol detection can be performed using the real signal, but it is better to perform matched filtering on the complex frequency-modulated signal for this purpose. The outputs of the matched filters can also be used for improving carrier offset estimation.

The main computational load of the system is in the Hilbert transformer. As we have seen, the Hilbert transformer requires about 38 real operations per sample (the order of the filter). The four matched filters together require 4 complex operations per sample, since the length of each filter is L and their outputs are decimated by L. The FM discriminator requires only few operations, one being an arcsine operation. The other parts require only few computations, thanks to the simplicity of the filters they use.

Similar techniques to those described here can be used for other types of digital communication signals; see Frerking [1994] for a detailed exposition of digital techniques in communication systems.

14.5 Electrocardiogram Analysis

A typical ECG signal, the electrical signal measured from the heart, is shown in Figure 14.18. An ECG signal looks roughly like an impulse train whose frequency is the heart rate.

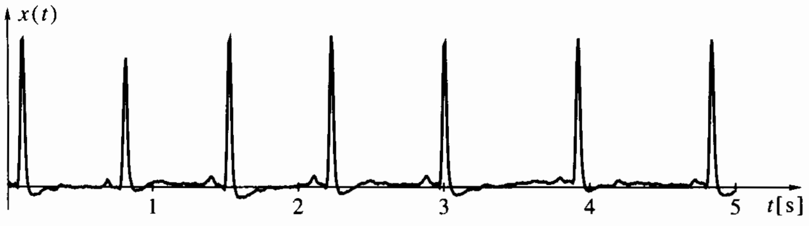

Figure 14.18 Typical electrocardiogram.

Close examination of a single ECG pulse reveals a characteristic pattern, as shown in Figure 14.19. The highest positive wave is called R. Shortly before and after the R wave there are negative waves, called Q and S, respectively. Before the Q wave there is a positive wave called the P, and after the S wave there is another positive wave called T. Both P and T are typically much lower than R. The Q, R, and S waves together are called the QRS complex.

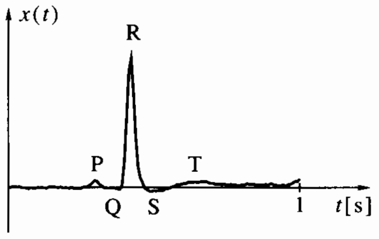

Figure 14.19 A single ECG wave.

The properties of the QRS complex—its rate of occurrence and the times, heights, and widths of its components—provide a wealth of information to the cardiologist on various pathological conditions of the heart. ECG instruments have been used by cardiologists for many years. In common instruments, the ECG signal is plotted on a chart recorder, and its evaluation is done manually. In modern instruments, processing of ECG signals is done digitally. Typical sampling frequencies of ECG signals are from 100 to 250 Hz.

In this section we discuss a particular application of ECG: measurement of the heart rate. The heart rate (the pulse) is not constant, even for a healthy person in a relaxed condition. One type of heart rate variation is related to control of the respiratory system and has a period of about 4-5 seconds, the normal breathing interval. Other variations are related to the control effects of the autonomic nervous system; these have periods of about 10-50 seconds. Various heart rate irregularities are developed by cardiac pathologies.

The ECG signal we use for our analysis is available in the file ecg.bin (see page vi for information on how to download this file). This signal has been sampled at a frequency of 128 Hz and contains 2 minutes of ECG of a healthy person in a relaxed condition.

Observation of the ECG signal reveals that it is slightly noisy, and the noise frequency is nearly half the sampling rate. This noise is most likely due to power mains interference, which is 50 Hz in this case. For detailed analysis of the QRS complex, this noise should be filtered. However, for heart rate determination, which relies on the R waves alone, the noise can be ignored to first approximation.

To determine the heart rate, we need to find the intervals between successive R points. We use the procedure `locmax` (Program 6.4) for this. To avoid finding the noise peaks and the P and T peaks, we limit the signal from below to a large fraction of its maximum value. We then sort the indices of the local peaks in increasing order and find their differences. The following MATLAB fragment implements this procedure:

```
[y,ind] = locmax(max(ecg,0.4*max(ecg)));
dt = (1/128)*diff(sort(ind));
```

The division by 128 converts the heart intervals to seconds.

Figure 14.20(a) shows the sequence of heart intervals $T[n]$. The mean interval is 0.887 second, corresponding to a mean rate of 1.13 beats per second. The instantaneous interval varies in the range 0.7-1 second, so the instantaneous rate varies in the range 1-1.4 beats per second.

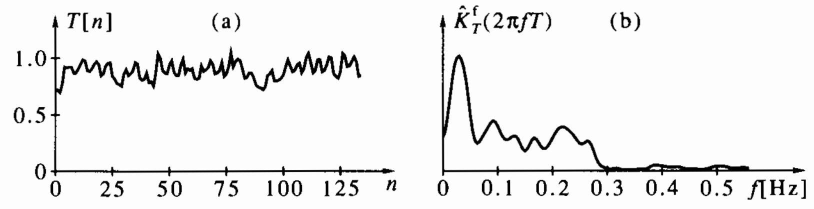

Figure 14.20 The sequence of heart intervals (a) and its estimated spectrum (b).

Next we wish to find periodicities in the heart rate. Since the sequence of heart intervals looks rather random, finding periodicities by observing the magnitude DFT does not appear promising. The Welch periodogram, presented in Section 13.1, is not adequate for this purpose either; the data length ($N = 135$) is too short for effective segmentation. We therefore use the smoothed periodogram, presented in Section 13.2, for analyzing the spectrum of the heart interval sequence $T[n]$. Before we show the results, we need to discuss the physical interpretation of the frequency variable in this application. The sequence $T[n]$, as shown in Figure 14.20(a), is in discrete time. However, we can interpret the sampling interval of this sequence as the average heart interval, which is 0.887 second in this case. Therefore, the physical frequency f is related to the variable θ of the periodogram by $f = \theta/(2\pi \times 0.887)$.

Figure 14.20(b) shows the smoothed periodogram of the sequence $T[n]$. This periodogram was computed with Hamming window of length 101. We can see three peaks, indicating the existence of three periodic components. The two lower ones have frequencies 0.03 and 0.093 Hz. These are associated with the control effects of the autonomic nervous system. The higher frequency is 0.22 Hz, and is associated with the control of the respiratory system.

14.6 Microprocessors for DSP Applications

Texas Instruments, in 1982, introduced the first microprocessor specifically designed for DSP applications—the TMS32010. Since then, several manufacturers have developed DSP microprocessors (also called DSP chips) of their own. The leading DSP chip manufacturers, at the time this book is written, are (in alphabetical order):

1. **Analog Devices:** the ADSP-21xx family of 16-bit, fixed-point chips and the ADSP-21xxx family of 32-bit, floating-point chips (each x refers to a decimal digit in the designation of a member of the family).
2. **AT&T:** the ADSP16xx family of 16-bit, fixed-point chips and the ADSP32xx family of 32-bit, floating-point chips.
3. **Motorola:** the DSP56xxx family of 24-bit, fixed-point chips and the DSP96xxx family of 32-bit, floating-point chips.
4. **NEC:** the μPD77xxx family of 16-bit and 24-bit, fixed-point chips.
5. **Texas Instruments:** the TMS320Cxx families of 16-bit, fixed-point and 32-bit, floating-point chips.

Besides those, there are numerous smaller manufacturers of both general-purpose and special-purpose DSP chips. We shall not attempt to do justice to all here.

In this section, we first discuss general concepts related to DSP chips. We then describe a single fixed-point chip of mid-1990s vintage: the Motorola DSP56301. We have chosen this particular chip arbitrarily. By the time you read this book, the chip will most probably be obsolete, thanks to the rapid progress of chip technology. However, we believe that the basic concepts and principles will last longer.

14.6.1 General Concepts

To appreciate the benefits offered by DSP microprocessors, we consider, as an example, the FIR filtering operation

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]. \tag{14.70}$$

Let us dissect this computation, as performed by a simple single-instruction, single-data (SISD) processor. Assume we have stored the $N+1$ coefficients in a storage vector $\boldsymbol{h}$, and the $N+1$ most recent values of the input signal in a storage vector $\boldsymbol{x}$. Just prior to the execution of the computations corresponding to time n, the vector $\boldsymbol{x}$ holds $\{x[n], x[n-1], \ldots, x[n-N]\}$. At time n, we need to perform the following operations:

1. Set a temporary variable y to 0. When the operation is complete, y will hold the computed value of $y[n]$.
2. Set a loop counter k to 0.
3. Repeat the following operations until $k = N$:
   (a) Load $h[k]$ from the kth location of $\boldsymbol{h}$ to the CPU.
   (b) Load $x[n-k]$ from the kth location of $\boldsymbol{x}$ to the CPU.
   (c) Multiply $h[k]$ by $x[n-k]$ and optionally round the result to single precision.
   (d) Load y, add to it the product, and store back in y.
   (e) Increase the loop counter k by 1, check if $k > N$ and exit if true.
4. Output y to its destination (e.g., to a D/A converter).
5. Shift the elements of $\boldsymbol{x}$ one place to the right, deleting $x[n-N]$ and making room for $x[n+1]$ in the first position.
6. Input $x[n+1]$ from its source (e.g., from an A/D converter) and store in the first position of $\boldsymbol{x}$.

In addition, each instruction needs to be read from the program memory into the control unit of the CPU.

As we see, a single FIR update operation can take many CPU cycles if implemented on a SISD computer. Let us now explore a few possibilities for expediting this procedure, at the expense of additional hardware.

1. Suppose we have two memory areas that can be accessed simultaneously. Then we can keep $\boldsymbol{h}$ in one area, $\boldsymbol{x}$ in the second, and load $h[k]$ and $x[n-k]$ simultaneously.
2. We can keep the temporary variable y in a CPU register, thus eliminating its loading from memory and storing back at each count of k. Such a register is called an *accumulator.* Furthermore, we can let the accumulator have a double length compared with that of $h[k]$ and $x[n-k]$. Then the product need not be rounded, but can be added directly to the current value of y. The combination of multiplier, double-length accumulator, and double-length adder is called multiplier-accumulator, or MAC for short.
3. We can use *hardware loop control,* which causes the sequence of operations in part 2 to repeat itself automatically $N+1$ times, without explicit program control.
4. We can use a *circular buffer* for the vector $\boldsymbol{x}$. The principle of operation of a circular buffer is illustrated in Figure 14.21. A pointer indicates the location of the most recent data point $x[n]$. Older data points are stored clockwise from the pointer. After $y[n]$ is computed, $x[n-N]$ is replaced with $x[n+1]$ and the pointer moves counterclockwise by one position to point at $x[n+1]$. As we see, all storage variables but one do not change their positions in the buffer, and only one variable is replaced.

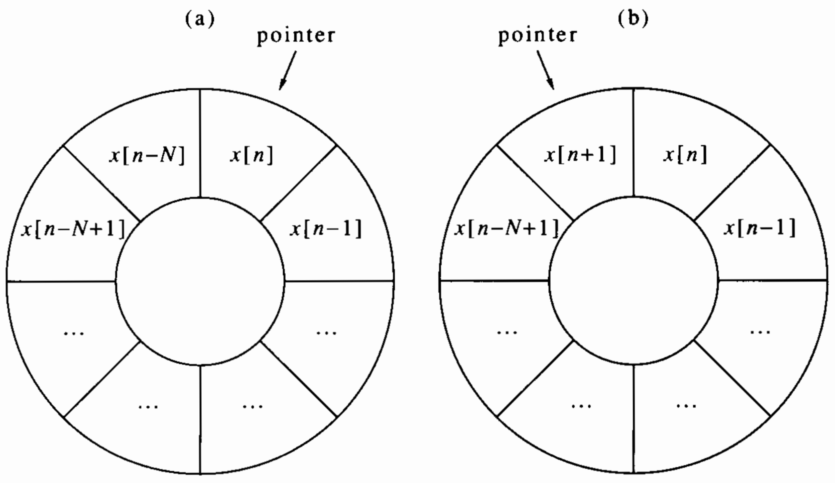

Figure 14.21 A circular buffer: (a) before the nth time point; (b) after the nth time point.

In practice, a circular buffer is implemented by using modular arithmetic for the memory address; that is, the pointer advances by 1 modulo $N+1$ each time k is increased. At the end of the loop, we decrease the pointer by 1 modulo $N+1$. It is convenient to store the filter coefficients $h[k]$ in a circular buffer as well. However, the pointer of this buffer is not decreased by 1 at the end of the loop.

A processor that takes advantage of all these possibilities can, in the limit, perform all operations needed at each count of k in a single machine cycle. To achieve this limit, parallel processing and pipelining are necessary. The former refers to performing independent operations simultaneously (such as loading the two multiplicands from memory). The latter refers to performing operations belonging to different cycles at the same time. For example, addition of a product $h[k]x[n-k]$ to the accumulator can be performed while the multiplier already computes $h[k+1]x[n-k-1]$.

14.6.2 The Motorola DSP56301

The Motorola DSP56301 is a member of the DSP56300 family of 24-bit microprocessors [Motorola, 1995]. Its CPU is built around a multiplier, capable of multiplying 24-bit numbers and producing a 48-bit result, and two 56-bit accumulators. The CPU gets its operands from two independent memory areas, denoted by X and Y. A 48-bit X register holds data transferred to and from the X memory, and a 48-bit Y register holds data transferred to and from the Y memory. The inputs to the multiplier can come only from the X and Y registers. The output of the multiplier can be added to (or subtracted from) either of the two accumulators, denoted by A and B. The outputs of A and B can be moved to either X or Y memory areas.

Figure 14.22 shows the structure of the DSP56301 registers. Each of X and Y consists of two parts: X1, X0, Y1, and Y0. These parts can be used as independent 24-bit registers, or combined to form 48-bit registers. The DSP56301 arithmetic is *fractional*: The numbers represented by a 24-bit word are in the range -1 to $1-2^{-23}$, and the numbers represented by a 48-bit word are in the range -1 to $1-2^{-47}$. Usually, X0 and X1 are used for 24-bit numbers each, as are Y0 and Y1. The main reason for combining them is evident when we wish to multiply two double-precision numbers, or a double-precision number by a single-precision number. Such multiplications cannot be performed by the multiplier in a single step, but in a sequence of steps.

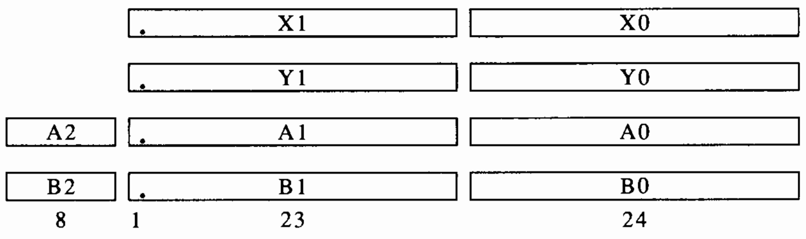

Figure 14.22 The registers of the Motorola DSP56301 microprocessor.

The A and B accumulators contain 56 bits each. The 8-bit parts A2 and B2 are called extension registers. They extend the range of the two accumulators to about ± 256. In other words, these two accumulators have both an integer and a fractional part, as shown in Figure 14.22. It follows that when the multiplicands are in single precision, no quantization noise is generated, since the product is accumulated in double precision. Also, overflow up to ± 256 does not lead to loss of information, since the overflow bits are properly stored in the extension register. This greatly alleviates the scaling problem in both FIR and IIR filters.

The MAC performs a multiply-add operation in two machine cycles. However, since it is pipelined, a new multiply-add can be initiated at every cycle, thus yielding an effective computation rate of one multiply-add per cycle.

The DSP56301 microprocessor contains several other features that enhance digital signal processing applications:

1. The accumulators can be switched to a saturation mode, as explained in Section 11.6.4. In saturation mode, the extension accumulators A2, B2 are not used, and A, B are limited to fractional values.
2. Regardless of whether the accumulators are in saturation mode, data are moved around in a *saturation transfer* mode. Thus, when the number in an accumulator is larger than 1 in magnitude, the number passed back to the X and Y registers or to memory is saturated with the proper sign.
3. The result of a MAC operation can optionally be multiplied by 2 or divided by 2. This is useful for implementing block floating-point FFT, as explained in Section 5.3.4. It is also useful for implementing second-order sections of IIR filters since, as we saw in Section 11.6, the denominator coefficients g_i of the sections are usually scaled by 0.5.
4. There is hardware loop control, enabling automatic repetition of either a single instruction or a block of instructions a desired number of times (with no latency).
5. The DSP56301 has two rounding modes: two's-complement rounding and convergent rounding. The two differ only in the way the number 0.5 is rounded.[2]
6. There are special instructions to facilitate double-precision multiplication, as well as division (which, however, require more than one machine cycle).
7. The availability of two accumulators is convenient for implementing complex arithmetic, for example, in FFT.
8. The CPU can be switched to a 16-bit mode, in which single-precision numbers have 16 bits and double-precision numbers have 32 bits. The extension accumulators A2, B2 continue to have 8 bits and fulfill the same tasks as in 24-bit mode.
9. There are eight 24-bit address registers, each having three fields. The field Rk (where $0 \le k \le 7$) holds the memory address, the field Nk holds the offset with respect to that address, and the field Mk contains information related to the mode of offset calculation. There are three modes of offset calculation: linear, modular, and reverse carry. The first simply adds the offset to the memory address; it is useful for accessing arrays of data. The second adds the offset to the memory address, modulo a given positive number; it is useful for implementing a circular buffer, as explained previously. The third implements bit-reversed offset; it is useful for loading or storing data in radix-2 FFT, as we recall from Section 5.3.
10. The DSP56301 has five on-chip memories: 2K X random-access memory (RAM), 2K Y RAM, 3K program RAM, 1K instruction cache, and 192 words bootstrap read-only memory (ROM) (K is 1024 words; in this case, each word is 24 bits long).
11. The DSP56301 has host interface to industry standard buses, enabling connections to other computers, as well as synchronous and serial interfaces to various peripherals.

We now illustrate how the FIR convolution operation (14.70) is implemented on the DSP56301. The following assembler code fragment performs this calculation for a single time point n.

```
movep   y:input,y:(r4)                                  ; line 1
clr     a                x:(r0)+,x0     y:(r4)+,y0      ; line 2
rep     #N                                              ; line 3
mac     x0,y0,a          x:(r0)+,x0     y:(r4)+,y0      ; line 4
macr    x0,y0,a                         (r4)-           ; line 5
movep   a,y:output                                      ; line 6
```

Here is an explanation of this code fragment:

1. The order of the filter N is given by the constant value #N.
2. The memory area X is assumed to hold the coefficients $h[k]$ in an increasing order, in a circular buffer of length $N + 1$. The address register R0 holds the address of $h[0]$. The modifier field M0 holds the number N. This causes R0 to be incremented modulo $N + 1$ when needed (the number in the modifier field is 1 less than the modulus).
3. The memory area Y is assumed to hold the signal samples $h[n-k]$ in a decreasing order, in a circular buffer of length $N + 1$. The address register R4 holds the address of the most recent data point. The modifier field M4 holds the number N. This causes R4 to be incremented modulo $N + 1$ when needed.
4. In line 1, the sample $x[n]$ is loaded from an input port mapped to the memory address y:input (e.g., from an A/D converter) and stored in the Y memory area, in the address specified by the contents of R4.
5. In line 2, the accumulator A is cleared. At the same time, the registers X0 and Y0 are loaded with $h[0]$ and $x[n]$, respectively. When loading is complete, R0 and R4 are incremented. Therefore, R0 now contains the address of $h[1]$ and R1 contains the address of $x[n - 1]$.
6. Line 3 instructs the CPU to perform the next instruction (in line 4) N times.
7. In line 4, the product $h[k]x[n - k]$ is calculated for all $0 \le k \le N - 1$ and added to the contents of A. Each time, the next coefficient and data sample are loaded to X0 and Y0, and the address registers R0, R4 are incremented.
8. In line 5, the product $h[N]x[n - N]$ is calculated, added to the contents of A, and the result is rounded. Now A1 contains the number $y[n]$. We note that both R0 and R4 have been incremented a total of $N + 1$ times. Therefore, they now point again at $h[0]$ and $x[n]$, respectively. By decrementing R4, we cause it to point at $x[n - N]$. This is the address to be overwritten by $x[n + 1]$ at the next time point.
9. In line 6, the accumulator contents, $y[n]$, is sent to an output port (e.g., a D/A converter) mapped to the memory address `y:output`.

14.7 Sigma-Delta A/D Converters

In Example 12.4 we demonstrated the possibility of trading speed and accuracy in A/D converters. However, the technique presented there can gain only half a bit accuracy for each doubling of the sampling rate. In this section we describe a state-of-the-art technique for A/D converter implementation that further exploits the speed-accuracy trade-off. This technique, called *sigma-delta A/D*, provides a fine example of the advantages gained by combining VLSI technology and digital signal processing principles. As we shall see, sigma-delta A/D converters require internal A/D and D/A converters of only few bits; all other bits are gained by increasing the sampling rate. In the extreme case, only 1-bit internal A/D and D/A are required. One-bit converters are extremely simple to implement: 1-bit A/D is just a sign detector, and 1-bit D/A is just a short circuit (or a constant gain).

Looking back at Example 12.4, we realize that only half a bit is gained for each doubling of the sampling rate because of the whiteness of the quantization noise. The key to improving the system is therefore to distribute the noise energy nonuniformly over the frequency band, to push most of the energy to higher frequencies. Noise at higher frequencies will be eliminated by the decimation filter, thus reducing the quantization noise level at the output of the filter. Figure 14.23(a) shows a block diagram of such a system. This is called a first-order sigma-delta A/D converter.

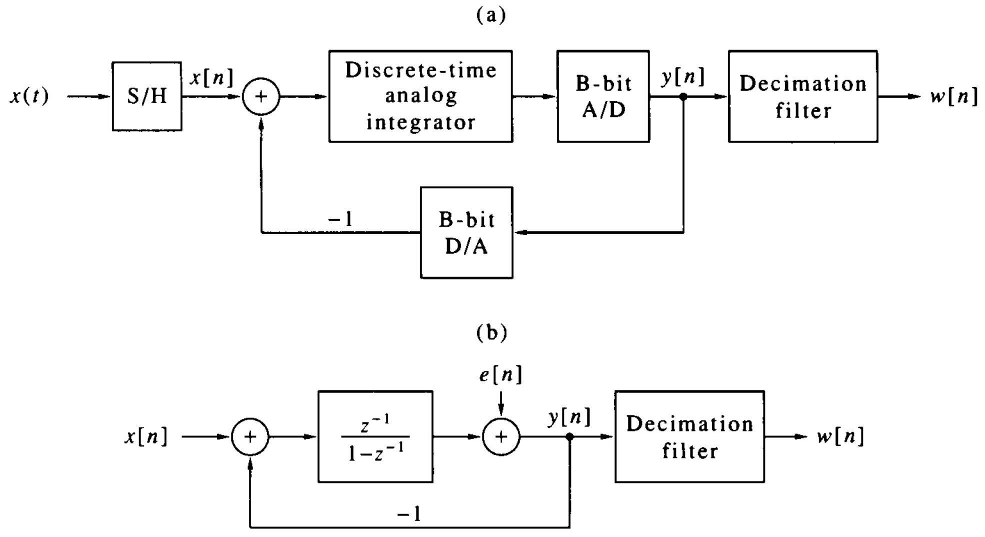

Figure 14.23 First-order sigma-delta A/D converter: (a) physical block diagram; (b) equivalent block diagram.

A first-order sigma-delta A/D converter operates as follows. The analog input signal is first sampled at a high rate f_{sam}. The output of the S/H is an analog (i.e., continuous-amplitude) discrete-time signal $x[n]$. An analog discrete-time error signal is generated by subtracting from $x[n]$ a reconstructed version of the output signal. The error signal is fed to a discrete-time analog integrator. Such an integrator is commonly implemented using switched-capacitor VLSI technology.[3] The actual A/D converter, assumed to have B bits, is placed at the output of the integrator. This is where the quantization noise $e[n]$ is generated. The A/D output $y[n]$ is a digital signal at a rate f_{sam}. This signal is forwarded to the decimation filter. It is also fed back to the input via a D/A converter that also has B bits. Since the number of bits is the same in the A/D and the D/A, together they form a unity system (neglecting secondary effects). Thus, the output of the integrator is fed back to its input at an inverted polarity. The name *sigma-delta* is because of the integration and subtraction performed by the circuit.

To analyze the operation of the first-order sigma-delta converter, we represent it by an equivalent block diagram, as shown in Figure 14.23(b). The transfer function of the integrator is $z^{-1}/(1 - z^{-1})$. We therefore get, by straightforward computation,

$$Y^z(z) = z^{-1}X^z(z) + (1 - z^{-1})E^z(z). \tag{14.71}$$

As we see, the transfer function from $x[n]$ to $y[n]$ is pure delay, so the input signal is not distorted. The noise $e[n]$, on the other hand, undergoes differencing, which is approximately differentiation. Therefore, the noise at $y[n]$ has little energy at low frequencies, as desired.

Let us assume that the signal bandwidth is f_m and that the decimation filter is an ideal low pass with cutoff frequency $\theta_0 = 2\pi f_m/f_{sam}$. Then the variance of the noise at the output of the decimation filter is given by

$$\begin{aligned}\gamma_w &= \frac{2^{-2(B-1)}}{12} \cdot \frac{1}{2\pi}\int_{-\theta_0}^{\theta_0} |1 - e^{-j\theta}|^2 d\theta = \frac{2^{-2(B-1)}}{12} \cdot \frac{1}{\pi}\int_{-\theta_0}^{\theta_0} (1 - \cos\theta) d\theta \\ &= \frac{2^{-2(B-1)}}{12} \cdot \frac{2}{\pi}(\theta_0 - \sin\theta_0). \end{aligned} \tag{14.72}$$

If $\theta_0 \ll \pi$ (as it will usually be in practical systems) we can use the approximation

$$\sin\theta_0 \approx \theta_0 - \frac{\theta_0^3}{6};$$

then we will get

$$\gamma_w \approx \frac{2^{-2(B-1)}}{12} \cdot \frac{2}{\pi} \cdot \frac{\theta_0^3}{6} = \frac{2^{-2(B-1)}}{12} \cdot \frac{\pi^2}{3} \cdot \left(\frac{2f_m}{f_{sam}}\right)^3. \tag{14.73}$$

Because the cubic dependence of γ_w on $2f_m/f_{sam}$, the noise variance decreases by 9 dB with each doubling of the sampling rate. This is equivalent to an additional 1.5 bit for each doubling of the sampling rate. However, the factor $\pi^2/3$ detracts about 1 bit (explain why!). Also, since the decimation filter is not ideal, the noise in the stop band is not completely eliminated, so in practice the equivalent number of bits is smaller.

As an example, consider the sigma-delta A/D converter for speech applications, described in Leung et al. [1988]. This A/D converter uses 1-bit A/D at the output of the integrator, and a sampling frequency of 4 MHz. The signal bandwidth is 4 kHz. Therefore $f_{sam}/2f_m = 500$, so the equivalent number of bits is

$$1 + 1.5\log_2 500 - 1 \approx 13.$$

Being able to produce 13 bits from a 1-bit A/D may sound incredible, but it is true.

The sigma-delta idea can be exploited yet further. The block diagram shown in Figure 14.24 represents a second-order sigma-delta A/D converter. Each block $z^{-1}/(1 - z^{-1})$ is a switched-capacitor integrator. The output $y[n]$ is related to the input $x[n]$ and the noise $e[n]$ through

$$Y^z(z) = z^{-1}X^z(z) + (1 - z^{-1})^2 E^z(z). \tag{14.74}$$

The variance of the noise at the output of the decimation filter is

$$\gamma_w \approx \frac{2^{-2(B-1)}}{12} \cdot \frac{\pi^4}{5} \cdot \left(\frac{2f_m}{f_{sam}}\right)^5. \tag{14.75}$$

Therefore, a second-order sigma-delta A/D converter gives 2.5 bits for each doubling of the sampling rate. From this we must subtract about 4.5 bits because of the factor $\pi^4/5$.

As an example, consider the sigma-delta A/D converter for high-fidelity audio applications, described in Sarhang-Nejad and Temes [1993]. This A/D converter uses 4-bit A/D and a sampling frequency of 5.25 MHz. The signal bandwidth is 20.5 kHz. Therefore $f_{sam}/2f_m = 128$, so the equivalent number of bits is

$$4 + 2.5\log_2 128 - 4.5 \approx 17.$$

In practice, this A/D converter gives only 16 bits.

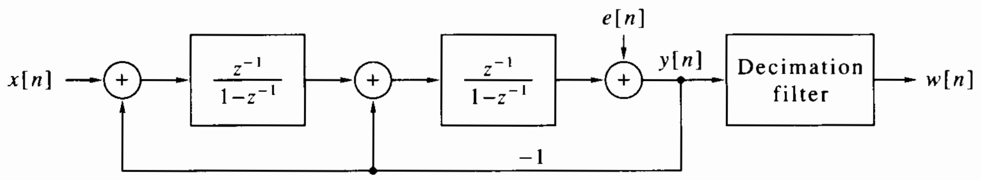

Figure 14.24 Second-order sigma-delta A/D converter.

14.8 Summary and Complements

14.8.1 Summary

We devoted this chapter to applications of digital signal processing and DSP technology. We presented applications from speech, music, communication, biomedicine, and signal compression. We then discussed features of current DSP microprocessors, and state-of-the-art A/D converter technology.

If you have mastered the contents of this book, you are ready for pursuing many advanced topics in digital signal processing, whether related to the aforementioned applications or not. Here is a selected list of such topics.

1. *Image processing* is a natural and highly important extension of signal processing. Images are two-dimensional signals: Instead of varying over time, they vary over the x and y coordinates of the image. Video (or motion picture) is a three-dimensional signal: It varies over the x and y coordinates of each frame, and the frames vary over time. Image processing has many aspects similar to conventional signal processing—sampling, frequency-domain analysis, z-domain analysis, filtering—and many unique aspects.
2. *Statistical signal processing* is concerned with the analysis and treatment of random signals: modeling, estimation, adaptive filtering, detection, pattern recognition.
3. *Speech processing* is concerned with speech signals and includes operations such as compression, enhancement, echo cancellation, speaker separation, recognition, speech-to-text and text-to-speech conversion.
4. *Biomedical signal processing* is concerned with signals generated by the human body, with the auditory and visual systems, with medical imaging, with artificial organs, and more.
5. *Array signal processing* is concerned with the utilization of sensor arrays for localization and reception of multiple signals. Array signal processing has long been used for military applications (for both electromagnetic and underwater acoustic signals), but has been extended to commercial applications in recent years.
6. *DSP technology* is concerned with general-purpose and application-specific architectures, parallel processing, VLSI implementations, converters, and more.

I shall let a pen worthier than mine write the final word:

'Tis pleasant, sure, to see one's name in print;
a book's a book, although there's nothing in't.
Lord Byron (1788-1824)

14.8.2 Complements

1. [p. 557] If $\{y[n],\ 0 \le n \le N-1\}$ is a WSS signal and $w[n]$ is a window, then $\{y[n]w[n],\ 0 \le n \le N-1\}$ cannot be WSS, because

$$E(y[n+m]w[n+m]y[n]w[n]) = w[n+m]w[n]\kappa_y[m],$$

   and this depends on n as well as on m.

2. [p. 585] Suppose we want to round a binary number from double precision to single precision. Let x_0 be the least significant part, x_1 the most significant part, and y the LSB of x_1. Then, if $x_0 < 0.5$, x_1 is rounded downward (i.e., it is not changed); if $x_0 > 0.5$, x_1 is rounded upward (i.e., increased by 1 LSB). The question is how to handle the case $x_0 = 0.5$. In two's-complement rounding, x_1 is always rounded upward. In convergent rounding, x_1 is rounded downward (not changed) if $y = 0$, or upward (increased by 1 LSB) if $y = 1$.

3. [p. 587] A switched-capacitor integrator is based on the property that the voltage across a capacitor is proportional to the integral of the current passing through it. The current is controlled by the input voltage such that each sampling interval, a charge proportional to the input voltage is added to the capacitor. This makes the switched capacitor a discrete-time, continuous-amplitude integrator.