CHAPTER

Waveform Quantization and Compression

11

CHAPTER OUTLINE

OBJECTIVES:

This chapter studies speech quantization and compression techniques such as signal companding, differential pulse code modulation, and adaptive differential pulse code modulation. The chapter continues to explore the discrete-cosine transform (DCT) and the modified DCT and shows how to apply the developed concepts to understand the MP3 audio format. The chapter also introduces industry standards that are widely used in the digital signal processing field.

11.1 LINEAR MIDTREAD QUANTIZATION

As we discussed in Chapter 2, in the digital signal processing (DSP) system, the first step is to sample and quantize the continuous signal. Quantization is the process of rounding off the sampled signal voltage to the predetermined levels that will be encoded by analog-to-digital conversion (ADC). We described the quantization process in Chapter 2, in which we studied unipolar and bipolar linear quantizers in detail. In

Digital Signal Processing. http://dx.doi.org/10.1016/B978-0-12-415893-1.00011-1

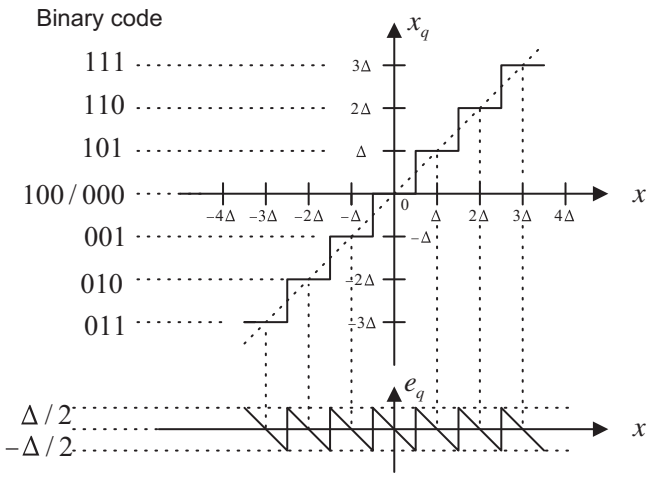

FIGURE 11.1

Characteristics of a 3-bit midtread quantizer.

this section, we focus on a linear midtread quantizer, which is used in digital communications (Roddy and Coolen, 1997; Tomasi, 2004), and its use to quantize speech waveforms. The linear midtread quantizer is similar to the bipolar linear quantizer discussed in Chapter 2 except that the midtread quantizer offers the same decoded magnitude range for both positive and negative voltages.

Let us look at a midtread quantizer. The characteristics and binary codes for a 3-bit midtread quantizer are depicted in Figure 11.1, where the code is in a sign magnitude format. Positive voltage is coded using a sign bit of logic 1, while negative voltage is coded by a sign bit of logic 0; the next two bits are the magnitude bits. The key feature of the linear midtread quantizer is noted as follows: when $0 \leq x < \Delta/2$, the binary code of 100 is produced; when $-\Delta/2 \leq x < 0$, the binary code of 000 is generated, where Δ is the quantization step size. However, the quantized values for both codes 100 and 000 are the same and equal to $x_q = 0$. We can also see details in Table 11.1. For the 3-bit midtread

Table 11.1 Quantization Table for the 3-bit Midtread Quantizer

Binary Code	Quantization Level x_q (V)	Input Signal Subrange (V)
0 1 1	-3Δ	$-3.5\Delta \leq x < -2.5\Delta$
0 1 0	-2Δ	$-2.5\Delta \leq x < -1.5\Delta$
0 0 1	$-\Delta$	$-1.5\Delta \leq x < -0.5\Delta$
0 0 0	0	$-0.5\Delta \leq x < 0$
1 0 0	0	$0 \leq x < 0.5\Delta$
1 0 1	Δ	$0.5\Delta \leq x < 1.5\Delta$
1 1 0	2Δ	$1.5\Delta \leq x < 2.5\Delta$
1 1 1	3Δ	$2.5\Delta \leq x < 3.5\Delta$

Note: Step size $= \Delta = (x_{max} - x_{min})/(2^3 - 1)$; x_{max} *= maximum voltage; and* $x_{min} = -x_{max}$. *Coding format: a. sign bit: 1= plus, 0 = minus; b. 2 magnitude bits.*

qunatizer, we expect seven quantized values instead of eight; that is; there are $2^n - 1$ quantization levels for the n-bit midtread quantizer. Notice that quantization signal range is $(2^n - 1)\Delta$ and the magnitudes of the quantized values are symmetric, as shown in Table 11.1. We apply the midtread quantizer particularly for speech waveform coding.

The following example serves to illustrate the coding principles of the 3-bit midtread quantizer.

EXAMPLE 11.1

For the 3-bit midtread quantizer described in Figure 11.1 and the analog signal with a range from -5 volts to 5 volts,

a. determine the quantization step size;
b. determine the binary codes, recovered voltages, and quantization errors when the input is -3.6 volts and 0.5 volt, respectively.

Solution:

a. The quantization step size is calculated as

$$\Delta = \frac{5 - (-5)}{2^3 - 1} = 1.43 \text{ volts}$$

b. For $x = -3.6$ volts, we have $x = \dfrac{-3.6}{1.43} = -2.52\Delta$. From quantization characteristics, it follows that the binary code = 011 and the recovered voltage is $x_q = -3\Delta = -4.29$ volts. Thus the quantization error is computed as

$$e_q = x_q - x = -4.29 - (-3.6) = -0.69 \text{ volts}$$

For $x = 0.5 = \dfrac{0.5}{1.43}\Delta = 0.35\Delta$, we get binary code = 100. Based on Figure 11.1, the recovered voltage and quantization error are found to be

$$x_q = 0 \quad \text{and} \quad e_q = 0 - 0.5 = -0.5 \text{ volts}$$

As discussed in Chapter 2, the linear midtread quantizer introduces quantization noise, as shown in Figure 11.1; the signal-to-noise power ratio (SNR) is given by

$$SNR = 10.79 + 20 \cdot \log_{10}\left(\frac{x_{rms}}{\Delta}\right) \text{ dB} \tag{11.1}$$

where x_{rms} designates the root mean squared value of the speech data to be quantized. The practical equation for estimating the SNR for the speech data sequence $x(n)$ of N data points is written as

$$SNR = \left(\frac{\sum_{n=0}^{N-1} x^2(n)}{\sum_{n=0}^{N-1} \left(x_q(n) - x(n)\right)^2}\right) \tag{11.2}$$

$$SNR\,dB = 10 \cdot \log_{10}(SNR) \text{ dB} \tag{11.3}$$

Notice that $x(n)$ and $x_q(n)$ are the speech data to be quantized and the quantized speech data, respectively. Equation (11.2) gives the absolute SNR, and Equation (11.3) produces the SNR in terms of decibels (dB). Quantization error is the difference between the quantized speech data (or quantized voltage level) and speech data (or analog voltage), that is, $x_q(n) - x(n)$. Also note that from Equation (11.1), adding 1 bit to the linear quantizer would improve SNR by approximately 6 dB. Let us examine performance of the 5-bit linear midtread quantizer.

In the following simulation, we use a 5-bit midtread quantizer to quantize the speech data. After quantization, the original speech, quantized speech, and quantized error after quantization are plotted in Figure 11.2. Since the program calculates $x_{rms}/x_{\max} = 0.203$, we yield x_{rms} as $x_{rms} = 0.203 \times x_{\max} = 0.0203 \times 5 = 1.015$ and $\Delta = 10/(2^5 - 1) = 0.3226$. Applying Equation (11.1) gives $SNR = 21.02$ dB. The SNR using Equations (11.2) and (11.3) is approximately 21.6 dB.

The first plot in Figure 11.2 is the original speech, and the second plot shows the quantized speech. Quantization error is displayed in the third plot, where the error amplitude interval is uniformly distributed between –0.1613 and 0.1613, indicating the bounds of the quantized error ($\Delta/2$). The details of the MATLAB implementation are given in Program 11.1 in Section 11.7.

To improve the SNR, the number of bits must be increased. However, increasing the number of encoding bits will cost an expansive ADC device, larger storage media for storing the speech data, and more bandwidth for transmitting the digital data. To gain a more efficient quantization approach, we will study μ-law companding in the next section.

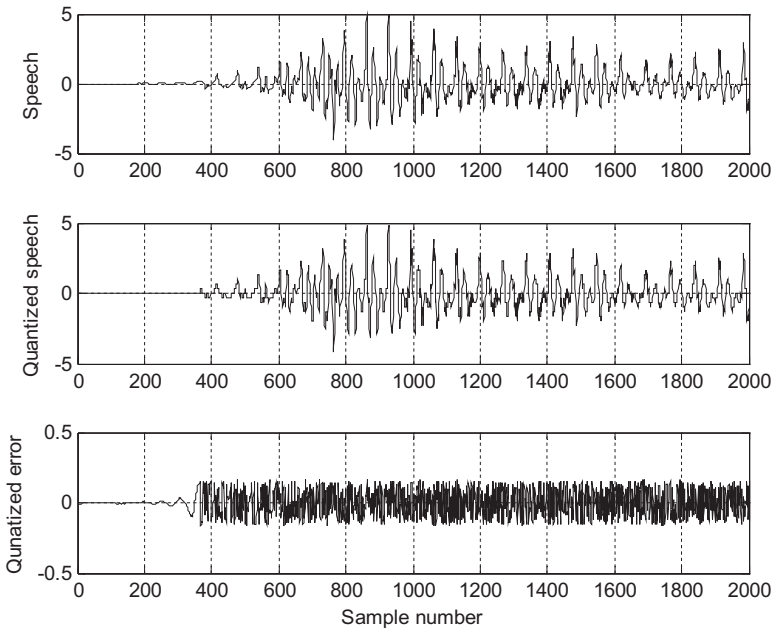

FIGURE 11.2

Plots of original speech, quantized speech, and quantization error.

11.2 μ-LAW COMPANDING

In this section, we will study analog μ-law companding, which takes an analog input signal; and digital μ-law companding, which deals with linear pulse code modulation (PCM) codes.

11.2.1 Analog μ-Law Companding

To reduce the number of bits required to encode each speech datum, μ-law companding, called log-PCM coding, is applied. μ-law companding (Roddy and Coolen, 1997; Tomasi, 2004) was first used in the United States and Japan in the telephone industry (G.711 standard). μ-law companding is a compression process. It explores the principle that the higher amplitudes of analog signals are compressed before ADC and expanded after digital-to-analog conversion (DAC). As studied in the linear quantizer, the quantization error is uniformly distributed. This means that the maximum quantization error stays the same no matter how big or small the speech samples are. μ-law companding can be employed to make the quantization error smaller when the sample amplitude is smaller and to make the quantization error bigger when the sample amplitude is bigger, using the same number of bits per sample. It is described in Figure 11.3.

As shown in Figure 11.3, x is the original speech sample, which is the input to the compressor, while y is the output from the μ-law compressor; then the output y is uniformly quantized. Assuming that the quantized sample y_q is encoded and sent to the μ-law expander, the expander will perform the reverse process to obtain the quantized speech sample x_q. The compression and decompression processes cause the maximum quantization error $|x_q - x|_{\max}$ to be small for the smaller sample amplitudes and large for the larger sample amplitudes.

The equation for the μ-law compressor is given by

$$y = sign(x)\frac{\ln\left(1+\mu\dfrac{|x|}{|x|_{\max}}\right)}{\ln(1+\mu)} \tag{11.4}$$

where $|x|_{\max}$ is the maximum amplitude of the inputs, while μ is a positive parameter to control the degree of the compression. $\mu = 0$ corresponds to no compression, while $\mu = 255$ is adopted in the industry. The compression curve with $\mu = 255$ is plotted in Figure 11.4. Note that the sign function $sign(x)$ shown in Equation (11.4) is defined as

$$sign(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{11.5}$$

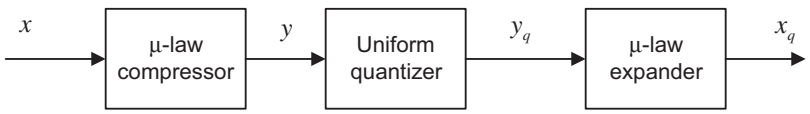

FIGURE 11.3

Block diagram for μ-law compressor and μ-law expander.

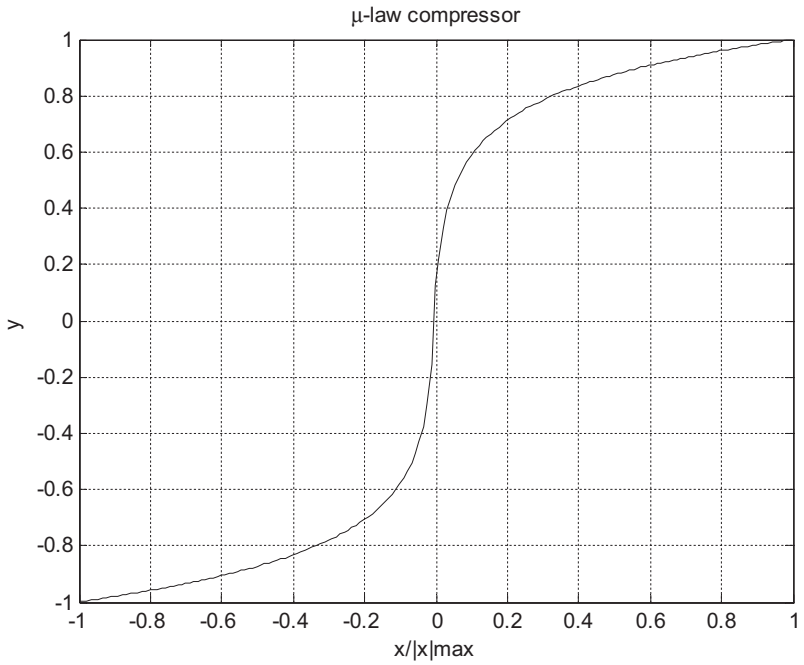

FIGURE 11.4

Characteristics for the μ-law compander.

Solving Equation (11.4) by substituting the quantized value, that is, $y = y_q$ we achieve the expander equation as

$$x_q = |x|_{\max} sign(y_q) \frac{(1+\mu)^{|y_q|} - 1}{\mu} \tag{11.6}$$

For the case $\mu = 255$, the expander curve is plotted in Figure 11.5.

Let's look at Example 11.2 on μ-law compression.

EXAMPLE 11.2

For the μ-law compression and expansion process shown in Figure 11.3, with $\mu = 255$, the 3-bit midtread quantizer described in Figure 11.1, and an analog signal ranging from -5 to 5 volts, determine the binary codes, recovered voltages, and quantization errors when the input is

a. -3.6 volts
b. 0.5 volts

Solution:

a. For μ-law compression and $x = -3.6$ volts, we can determine the quantization input as

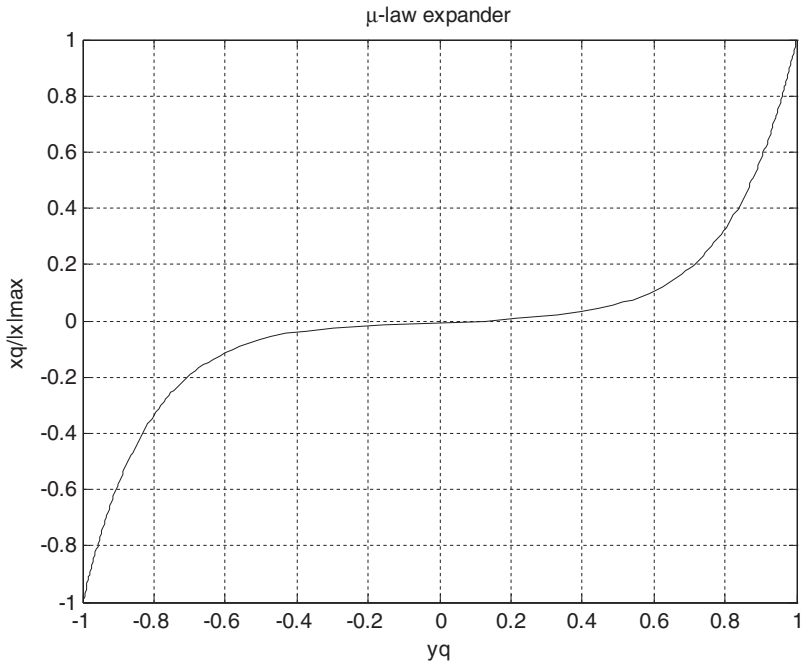

FIGURE 11.5

Characteristics for the μ-law expander.

$$y = sign(-3.6)\frac{\ln\left(1+255\frac{|-3.6|}{|5|_{\max}}\right)}{\ln(1+255)} = -0.94$$

As shown in Figure 11.4, the range of y is 2, thus the quantization step size is calculated as

$$\Delta = \frac{2}{2^3-1} = 0.286 \quad \text{and} \quad y = \frac{-0.94}{0.286} = -3.28\Delta$$

From quantization characteristics, it follows that the binary code is 011 and the recovered signal is $y_q = -3\Delta = -0.858$.

Applying the μ-law expander leads to

$$x_q = |5|_{\max} sign(-0.858)\frac{(1+255)^{|-0.858|}-1}{255} = -2.264$$

Thus the quantization error is computed as

$$e_q = x_q - x = -2.264 - (-3.6) = 1.336 \text{ volts}$$

b. Similarly, for $x = 0.5$, we get

$$y = sign(0.5)\frac{\ln\left(1+255\frac{|0.5|}{|5|_{\max}}\right)}{\ln(1+255)} = 0.591$$

In terms of the quantization step, we get

$$y = \frac{0.519}{0.286}\Delta = 2.1\Delta \quad \text{and binary code} = 110$$

Based on Figure 11.1, the recovered signal is

$$y_q = 2\Delta = 0.572$$

and the expander gives

$$x_q = |5|_{\max} sign(0.572)\frac{(1+255)^{|0.572|}-1}{255} = 0.448 \text{ volts}$$

Finally, the quantization error is given by

$$e_q = 0.448 - 0.5 = -0.052 \text{ volts}$$

As we can see, with 3 bits per sample, the stronger signal is encoded with more quantization error, while the weak signal is encoded with less quantization error.

In the following simulation, we apply a 5-bit μ-law compander with $\mu = 255$ in order to quantize and encode the speech data used in the last section. Figure 11.6 is a block diagram of compression and decompression.

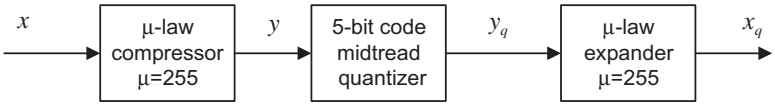

FIGURE 11.6

The 5-bit midtread uniform quantizer with $\mu = 255$ used for simulation.

Figure 11.7 shows the original speech data, the quantized speech data using μ-law compression, and the quantization error for comparisons. The quantized speech wave is very close to the original speech wave. From the plots in Figure 11.7, we can observe that the amplitude of the quantization error changes according to the amplitude of the speech being quantized. More quantization error is introduced when the amplitude of speech data is larger; on the other hand, a smaller quantization error is produced when the amplitude of speech data is smaller.

Compared with the quantized speech using the linear quantizer shown in Figure 11.2, the decompressed signal using the μ-law compander looks and sounds much better, since the quantized signal can better track the original large amplitude signal and original small amplitude signal as well. The MATLAB implementation is shown in Program 11.2 in Section 11.7.

11.2.2 Digital μ-Law Companding

In many multimedia applications, the analog signal is first sampled and then it is digitized into a linear PCM code with a larger number of bits per sample. Digital μ-law companding further compresses the linear PCM code using the compressed PCM code with a smaller number of bits per sample without losing sound quality. The block diagram of a digital μ-law compressor and expander is shown in Figure 11.8.

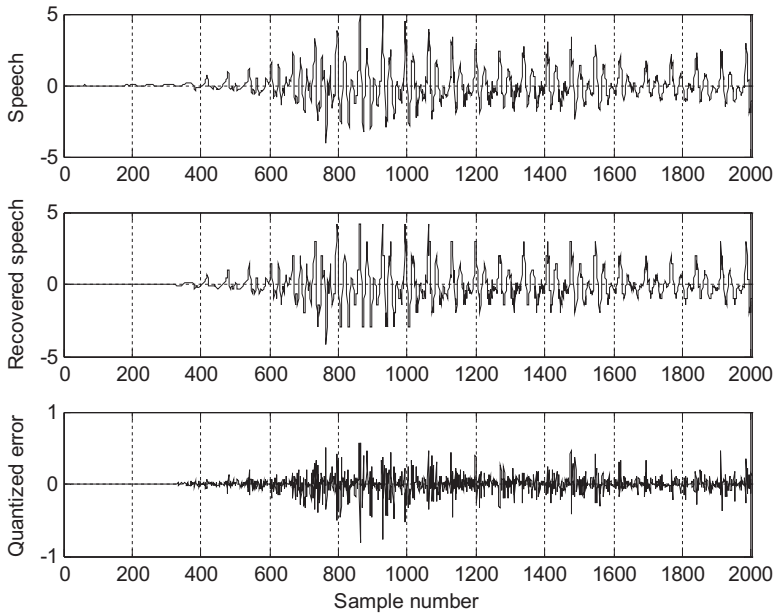

FIGURE 11.7

Plots of the original speech, quantized speech, and quantization error with the μ-law compressor and expander.

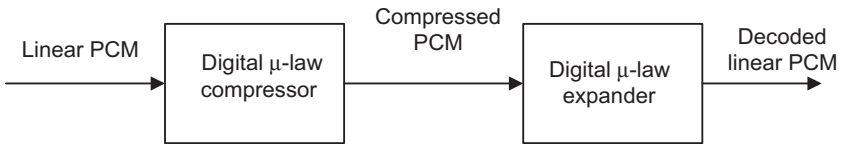

FIGURE 11.8

The block diagram for a μ-law compressor and expander.

The typical digital μ-law companding system compresses a 12-bit linear PCM code to an 8-bit compressed code. This companding characteristic is depicted in Figure 11.9, where it closely resembles an analog compression curve with $\mu = 255$ by approximating the curve using a set of eight straight-line segments. The slope of each successive segment is exactly one-half that of the previous segment. Figure 11.9 shows the 12-bit to 8-bit digital companding curve for the positive portion only. There are 16 segments, accounting for both positive and negative portions.

However, like the midtread quantizer discussed in the first section of this chapter, only 13 segments are used, since segments $+0$, -0, $+1$, and -1 form a straight line with a constant slope and are considered one segment. As shown in Figure 11.9, when the relative input is very small, such as in segment 0 or segment 1, there is no compression, while when the relative input is larger such as in segment 3 or segment 4, the compression occurs with compression ratios of 2:1 and 4:1, respectively.

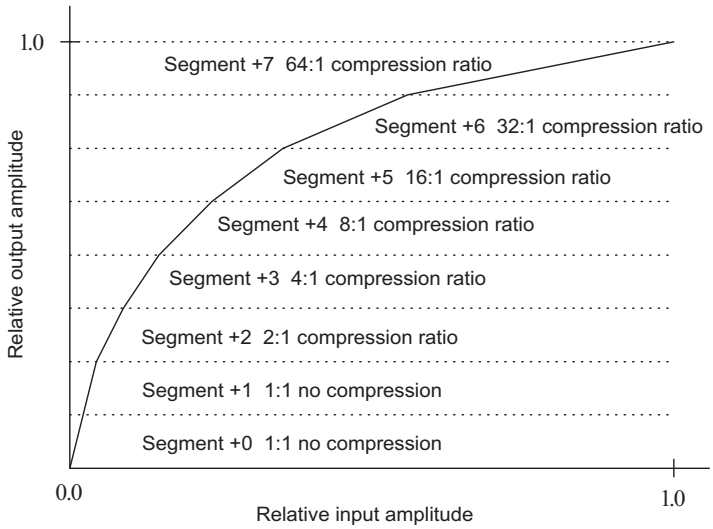

FIGURE 11.9

$\mu - 255$ compression characteristics (for positive portion only).

The format of the 12-bit linear PCM code is in the sign-magnitude form with the most significant bit (MSB) as the sign bit (1 = positive value and 0 = negative value) plus 11 magnitude bits. The compressed 8-bit code has the format shown in Table 11.2, where it consists of a sign bit, a 3-bit segment identifier, and a 4-bit quantization interval within the specified segment. Encoding and decoding procedures are very simple, as illustrated in Tables 11.3 and 11.4, respectively.

As shown in those two tables, the prefix "S" is used to indicate the sign bit, which could be either 1 or 0; A, B, C, and D, are transmitted bits; and the bit position with an "X" is the truncated bit during the compression and hence would be lost during decompression. For the 8-bit compressed PCM code in Table 11.3, the 3 bits between "S" and "ABCD" indicate the segment number that is obtained by subtracting the number of consecutive zeros (less than or equal to 7) after the "S" bit in the original 12-bit PCM code from 7. Similarly, to recover the 12-bit linear code in Table 11.4, the number of consecutive zeros after the "S" bit can be determined by subtracting the segment number in the 8-bit compressed code from 7. We will illustrate the encoding and decoding processes in Examples 11.3 and 11.4.

Table 11.2 The Format of 8-Bit Compressed PCM Code

Sign bit: 1 = + 0 = −	3-bit segment identifier: 000 to 111	4-bit quantization interval: A B C D 0000 to 1111

Table 11.3 $\mu - 255$ Encoding Table

Segment	12-Bit Linear Code	12-Bit Amplitude Range in Decimal	8-Bit Compressed Code
0	S0000000ABCD	0 to 15	S000ABCD
1	S0000001ABCD	16 to 31	S001ABCD
2	S000001ABCDX	32 to 63	S010ABCD
3	S00001ABCDXX	64 to 127	S011ABCD
4	S0001ABCDXXX	128 to 255	S100ABCD
5	S001ABCDXXXX	256 to 511	S101ABCD
6	S01ABCDXXXXX	512 to 1023	S110ABCD
7	S1ABCDXXXXXX	1023 to 2047	S111ABCD

Table 11.4 $\mu - 255$ Decoding Table

8-Bit Compressed Code	8-Bit Amplitude Range in Decimal	Segment	12-Bit Linear Code
S000ABCD	0 to 15	0	S0000000ABCD
S001ABCD	16 to 31	1	S0000001ABCD
S010ABCD	32 to 47	2	S000001ABCD1
S011ABCD	48 to 63	3	S00001ABCD10
S100ABCD	64 to 79	4	S0001ABCD100
S101ABCD	80 to 95	5	S001ABCD1000
S110ABCD	96 to 111	6	S01ABCD10000
S111ABCD	112 to 127	7	S1ABCD100000

EXAMPLE 11.3

In a digital companding system, encode each of the following 12-bit linear PCM codes into an 8-bit compressed PCM code.

a. 1 0 0 0 0 0 0 0 0 1 0 1
b. 0 0 0 0 1 1 1 0 1 0 1 0

Solution:

a. Based on Table 11.3, we identify the 12-bit PCM code as S = 1, A = 0, B = 1, C = 0, and D = 1, which is in segment 0. From the fourth column in Table 11.3, we get the 8-bit compressed code as

1 0 0 0 0 1 0 1

b. For the second 12 bit PCM code, we note that S = 0, A = 1, B = 1, C = 0, D = 1, and XXX = 010, and the code belongs to segment 4. Thus, from the fourth column in Table 11.3, we have

0 1 0 0 1 1 0 1

EXAMPLE 11.4

In a digital companding system, decode each of the following 8-bit compressed PCM codes into a 12-bit linear PCM code.

a. 1 0 0 0 0 1 0 1
b. 0 1 0 0 1 1 0 1

Solution:

a. Using Table 11.4, we notice that S = 1, A = 0, B = 1, C = 0, and D = 1, and the code is in segment 0. Decoding leads to 1 0 0 0 0 0 0 0 1 0 1, which is identical to the 12-bit PCM code in (a) in Example 11.3. We expect this result, since there is no compression for segment 0 and segment 1.
b. Applying Table 11.4, it follows that S = 0, A = 1, B = 1, C = 0, and D = 1, and the code resides in segment 4. Decoding achieves 0 0 0 0 1 1 1 0 1 1 0 0. As expected, this code is the approximation of the code in (b) in Example 11.3. Since segment 4 has compression, the last 3 bits in the original 12-bit linear code, that is, XXX = 010 = 2 in decimal, are discarded during transmission or storage. When we recover these three bits, the best guess should be the middle value: XXX = 100 = 4 in decimal for the 3-bit coding range from 0 to 7.

Now we apply the $\mu - 255$ compander to compress 12-bit speech data as shown in Figure 11.10(a). The 8-bit compressed code is plotted in Figure 11.10(b). Plots (c) and (d) in the figure show the 12-bit

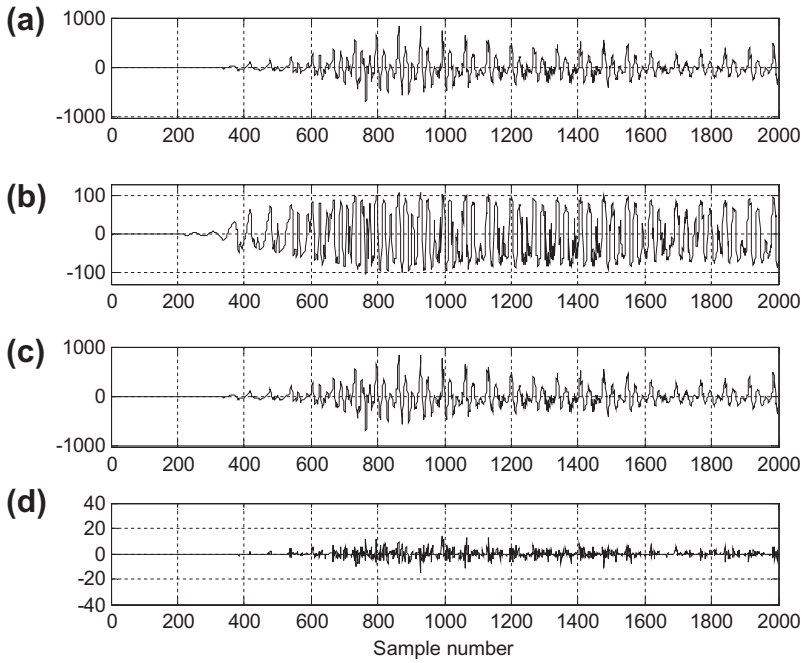

FIGURE 11.10

The $\mu - 255$ compressor and expander: (a) 12-bit speech data; (b) 8-bit compressed data; (c) 12-bit decoded speech; (d) quantization error.

speech after decoding and quantization error, respectively. We can see that the quantization error follows the amplitude of speech data relatively. The decoded speech sounds no different when compared with the original speech. Programs 11.8 to 11.10 in Section 11.7 show the details of the MATLAB implementation.

11.3 EXAMPLES OF DIFFERENTIAL PULSE CODE MODULATION (DPCM), DELTA MODULATION, AND ADAPTIVE DPCM G.721

Data compression can be further achieved using *differential pulse code modulation* (DPCM). The general idea is to use past recovered values as the basis to predict the current input data and then encode the difference between the current input and the predicted input. Since the difference has a significantly reduced signal dynamic range, it can be encoded with fewer bits per sample. Therefore, we obtain data compression. First, we study the principles of the DPCM concept that will help us understand adaptive DPCM in the next subsection.

11.3.1 Examples of Differential Pulse Code Modulation and Delta Modulation

Figure 11.11 shows a schematic diagram for the DPCM encoder and decoder. We denote the original signal $x(n)$; the predicted signal $\tilde{x}(n)$; the quantized or recovered signal $\hat{x}(n)$; the difference signal to be quantized $d(n)$; and the quantized difference signal $d_q(n)$. The quantizer can be chosen as a uniform quantizer, a midtread quantizer (e.g., see Table 11.5), or any others available. The encoding block produces a binary bit stream in the DPCM encoding. The predictor uses the past predicted signal and quantized difference signal to predict the current input value $x(n)$ as close as possible. The digital filter or adaptive filter can serve as the predictor. On the other hand, the decoder recovers the quantized difference signal, which can be added to the predictor output signal to produce the quantized and recovered signal, as shown in Figure 11.11(b).

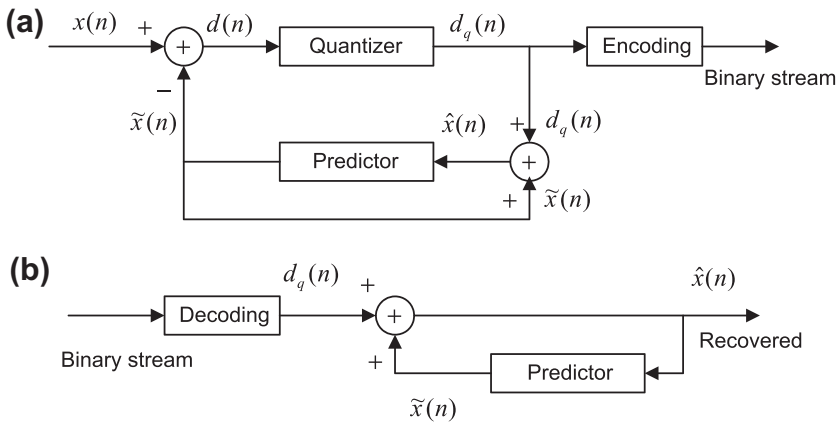

FIGURE 11.11

DPCM block diagram: (a) encoder; (b) decoder.

Table 11.5 Quantization Table for the 3-bit Quantizer in Example 11.5

Binary Code	Quantization Value $d_q(n)$	Subrange in $d(n)$
0 1 1	−11	$-15 \le d(n) < -7$
0 1 0	−5	$-7 \le d(n) < -3$
0 0 1	−2	$-3 \le d(n) < -1$
0 0 0	0	$-1 \le d(n) < 0$
1 0 0	0	$0 \le d(n) \le 1$
1 0 1	2	$1 < d(n) \le 3$
1 1 0	5	$3 < d(n) \le 7$
1 1 1	11	$7 < d(n) \le 15$

In Example 11.5, we examine a simple DPCM coding system via the process of encoding and decoding numerical actual data.

EXAMPLE 11.5

A DPCM system has the following specifications:

$$\text{Encoder scheme: } \tilde{x}(n) = \hat{x}(n-1); \text{ predictor}$$
$$d(n) = x(n) - \tilde{x}(n)$$
$$d_q(n) = Q[d(n)] = \text{quantizer in Table 11.5}$$
$$\hat{x}(n) = \tilde{x}(n) + d_q(n)$$

$$\text{Decoding scheme: } \tilde{x}(n) = \hat{x}(n-1); \text{ predictor}$$
$$d_q(n) = \text{quantizer in Table 11.5}$$
$$\hat{x}(n) = \tilde{x}(n) + d_q(n)$$

5-bit input data: $x(0) = 6$, $x(1) = 8$, $x(2) = 13$.

a. Perform DPCM encoding to produce the binary code for each input datum.
b. Perform DCPM decoding to recover the data using the binary code in (a).

Solution:

a. Let us perform encoding according to the encoding scheme.
For $n = 0$, we have
$\tilde{x}(0) = \hat{x}(-1) = 0$
$d(0) = x(0) - \tilde{x}(0) = 6 - 0 = 6$
$d_q(0) = Q[d(0)] = 5$
$\hat{x}(0) = \tilde{x}(0) + d_q(0) = 0 + 5 = 5$
Binary code =110

For $n = 1$, it follows that
$\tilde{x}(1) = \hat{x}(0) = 5$
$d(1) = x(1) - \tilde{x}(1) = 8 - 5 = 3$
$d_q(1) = Q[d(1)] = 2$
$\hat{x}(1) = \tilde{x}(1) + d_q(1) = 5 + 2 = 7$
Binary code = 101

For $n = 2$, the results are
$\tilde{x}(2) = \hat{x}(1) = 7$
$d(2) = x(2) - \tilde{x}(2) = 13 - 7 = 6$

$d_q(2) = Q[d(2)] = 5$
$\hat{x}(2) = \tilde{x}(2) + d_q(2) = 7 + 5 = 12$
Binary code = 110

b. We conduct the decoding scheme as follows.
For $n = 0$, we get
Binary code = 110
$d_q(0) = 5$; from Table 11.5
$\tilde{x}(0) = \hat{x}(-1) = 0$
$\hat{x}(0) = \tilde{x}(0) + d_q(0) = 0 + 5 = 5$ (recovered)

For $n = 1$, decoding shows:
Binary code = 101
$d_q(1) = 2$; from Table 11.5
$\tilde{x}(1) = \hat{x}(0) = 5$
$\hat{x}(1) = \tilde{x}(1) + d_q(1) = 5 + 2 = 7$ (recovered)

For $n = 2$, we have:
Binary code = 110
$d_q(2) = 5$; from Table 11.5
$\tilde{x}(2) = \hat{x}(1) = 7$
$\hat{x}(2) = \tilde{x}(2) + d_q(2) = 7 + 5 = 12$ (recovered)

From this example, we can verify that the 5-bit code is compressed to the 3-bit code. However, we can see that each piece of recovered data has a quantization error. Hence, DPCM is a lossy data compression scheme.

DPCM for which a single bit is used in the quantization table becomes *delta modulation* (DM). The quantization table contains two quantized values, A and $-A$, where A is the quantization step size. Delta modulation quantizes the difference of the current input sample and the previous input sample using a 1-bit code word. To conclude the idea, we list the equations for encoding and decoding as follows:

$$\text{Encoder scheme: } \tilde{x}(n) = \hat{x}(n-1); \text{ predictor}$$
$$d(n) = x(n) - \tilde{x}(n)$$
$$d_q(n) = \begin{cases} +A & d_q(n) \geq 0, \text{ output bit: } 1 \\ -A & d_q(n) < 0, \text{ output bit: } 0 \end{cases}$$
$$\hat{x}(n) = \tilde{x}(n) + d_q(n)$$

$$\text{Decoding scheme : } \tilde{x}(n) = \hat{x}(n-1); \text{ predictor}$$
$$d_q(n) = \begin{cases} +A & \text{input bit: } 1 \\ -A & \text{input bit: } 0 \end{cases}$$
$$\hat{x}(n) = \tilde{x}(n) + d_q(n)$$

Note that the predictor has a sample delay.

EXAMPLE 11.6

Consider a DM system with 5-bit input data

$$x(0) = 6,\ x(1) = 8,\ x(2) = 13$$

and a quantized constant of $A = 7$.

a. Perform DM encoding to produce the binary code for each input datum.
b. Perform DM decoding to recover the data using the binary code in (a).

Solution:

a. Applying encoding accordingly, we have
For $n = 0$,
$\tilde{x}(0) = \hat{x}(-1) = 0$, $d(0) = x(0) - \tilde{x}(0) = 6 - 0 = 6$
$d_q(0) = 7$, $\hat{x}(0) = \tilde{x}(0) + d_q(0) = 0 + 7 = 7$
Binary code = 1

For $n = 1$,
$\tilde{x}(1) = \hat{x}(0) = 7$, $d(1) = x(1) - \tilde{x}(1) = 8 - 7 = 1$
$d_q(1) = 7$, $\hat{x}(1) = \tilde{x}(1) + d_q(1) = 7 + 7 = 14$
Binary code = 1.

For $n = 2$,
$\tilde{x}(2) = \hat{x}(1) = 14$, $d(2) = x(2) - \tilde{x}(2) = 13 - 14 = -1$
$d_q(2) = -7$, $\hat{x}(2) = \tilde{x}(2) + d_q(2) = 14 - 7 = 7$
Binary code = 0

b. Applying the decoding scheme leads to
For $n = 0$,
Binary code = 1
$d_q(0) = 7$, $\tilde{x}(0) = \hat{x}(-1) = 0$
$\hat{x}(0) = \tilde{x}(0) + d_q(0) = 0 + 7 = 7$ (recovered)

For $n = 1$,
Binary code 1
$d_q(1) = 7$, $\tilde{x}(1) = \hat{x}(0) = 7$
$\hat{x}(1) = \tilde{x}(1) + d_q(1) = 7 + 7 = 14$ (recovered)

For $n = 2$,
Binary code 0
$d_q(2) = -7$, $\tilde{x}(2) = \hat{x}(1) = 14$
$\hat{x}(2) = \tilde{x}(2) + d_q(2) = 14 - 7 = 7$ (recovered)

We can see that coding causes a larger quantization error for each recovered sample. In practice, this can be solved by using a very high sampling rate (much larger than the Nyquist rate), and by making the quantization step size A adaptive. The quantization step size increases by a factor when the slope magnitude of the input sample curve becomes bigger, that is, the condition in which the encoder produces continuous logic 1s or continuous logic 0s in the coded bit stream. Similarly, the quantization step decreases by a factor when the encoder generates logic 1 and logic 0 alternatively. Hence, the resultant DM is called *adaptive* DM. In practice, the DM chip replaces the predictor, feedback path, and summer (see Figure 11.11) with an integrator for both the encoder and the decoder. Detailed information can be found in Li and Drew (2004), Roddy and Coolen (1997), and Tomasi (2004).

11.3.2 Adaptive Differential Pulse Code Modulation G.721

In this subsection, an efficient compression technique for speech waveform is described, that is, *adaptive DPCM* (ADPCM), per recommendation G.721 of the CCITT (the Comité Consultatif

International Téléphonique et Télégraphique). General discussion can be found in Li and Drew (2004), Roddy and Coolen (1997), and Tomasi (2004). The simplified block diagrams of the ADPCM encoder and decoder are shown in Figures 11.12A and 11.12B.

As shown in Figure 11.12A for the ADPCM encoder, first a difference signal $d(n)$ is obtained, by subtracting an estimate of the input signal $\tilde{x}(n)$ from the input signal $x(n)$. An adaptive 16-level quantizer is used to assign four binary digits $I(n)$ to the value of the difference signal for transmission to the decoder. At the same time, the inverse quantizer produces a quantized difference signal $d_q(n)$ from the same four binary digits $I(n)$. The adaptive quantizer and inverse quantizer operate based on the quantization table and the scale factor obtained from the quantizer adaptation to keep tracking the energy change of the difference signal to be quantized. The input signal estimate from the

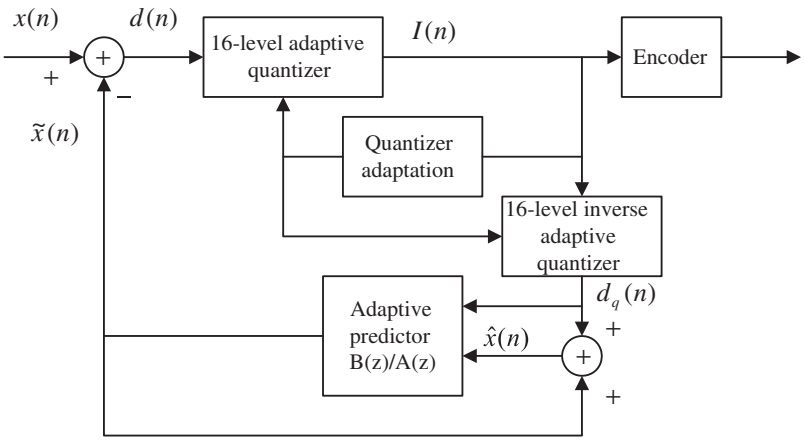

FIGURE 11.12A

ADPCM encoder.

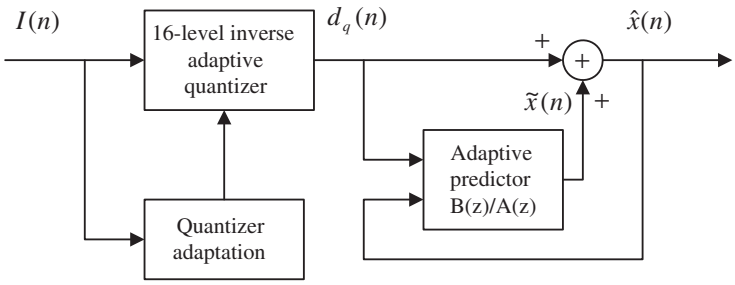

FIGURE 11.12B

ADPCM decoder.

adaptive predictor is then added to this quantized difference signal to produce the reconstructed version of the input $\hat{x}(n)$. Both the reconstructed signal and the quantized difference signal are operated on by an adaptive predictor, which generates the estimate of the input signal, thereby completing the feedback loop.

The decoder shown in Figure 11.12B includes a structure identical to the feedback part of the encoder as depicted in Figure 11.12A. It first converts the received 4-bit data $I(n)$ to the quantized difference signal $d_q(n)$ using the adaptive quantizer. Then, at the second stage, the adaptive predictor uses the recovered quantized difference signal $d_q(n)$ and recovered current output $\tilde{x}(n)$ to generate the next output. Notice that the adaptive predictors of both the encoder and the decoder change correspondingly based on the signal to be quantized. The details of the adaptive predictor will be discussed.

Now, let us examine the ADPCM encoder principles. As shown in Figure 11.12A, the difference signal is computed as

$$d(n) = x(n) - \tilde{x}(n) \tag{11.7}$$

A 16-level nonuniform adaptive quantizer is used to quantize the difference signal $d(n)$. Before quantization, $d(n)$ is converted to a base-2 logarithmic representation and scaled by $y(n)$, which is computed by the scale factor algorithm. Four binary codes $I(n)$ are used to specify the quantized signal level representing $d_q(n)$, and the quantized difference $d_q(n)$ is also fed to the inverse adaptive quantizer. Table 11.6 shows the quantizer normalized input and output characteristics.

The scaling factor for the quantizer and the inverse quantizer $y(n)$ is computed according to the 4-bit quantizer output $I(n)$ and the adaptation speed control parameter $a_l(n)$, the fast (unlocked) scale factor $y_u(n)$, the slow (locked) scale factor $y_l(n)$, and the discrete function $W(I)$, defined in Table 11.7:

$$y_u(n) = (1 - 2^{-5})y(n) + 2^{-5}W(I(n)) \tag{11.8}$$

where $1.06 \leq y_u(n) \leq 10.00$.

The slow scale factor $y_l(n)$ is derived from the fast scale factor $y_u(n)$ using a lowpass filter as follows:

$$y_l(n) = (1 - 2^{-6})y_l(n-1) + 2^{-6}y_u(n) \tag{11.9}$$

Table 11.6 Quantizer Normalized Input and Output Characteristics

Normalized Quantizer Input Range: $\log_2\|d(n)\| - y(n)$	**Magnitude:** $\|I(n)\|$	**Normalized Quantizer Output:** $\log_2\|d_q(n)\| - y(n)$
[3.12, $+\infty$)	7	3.32
[2.72, 3.12)	6	2.91
[2.34, 2.72)	5	2.52
[1.91, 2.34)	4	2.13
[1.38, 1.91)	3	1.66
[0.62, 1.38)	2	1.05
[−0.98, 0.62)	1	0.031
($-\infty$, −0.98)	0	$-\infty$

TABLE 11.7 Discrete Function $W(I)$

$\lvert I(n)\rvert$	7	6	5	4	3	2	1	0
$W(I)$	70.13	22.19	12.38	7.00	4.0	2.56	1.13	−0.75

The fast and slow scale factors are then combined to compute the scale factor:

$$y(n) = a_l(n)y_u(n-1) + (1 - a_l(n))y_l(n-1) \tag{11.10}$$

Next the controlling parameter $0 \leq a_l(n) \leq 1$ tends toward unity for speech signals and toward zero for voice band data signals and tones. It is updated based on the following parameters: $d_{ms}(n)$, which is the relatively short-term average of $F(I(n))$; $d_{ml}(n)$, which is the relatively long-term average of $F(I(n))$; and the variable $a_p(n)$, where $F(I(n))$ is defined as in Table 11.8.

TABLE 11.8 Discrete Function $F(I(n))$

$\lvert I(n)\rvert$	7	6	5	4	3	2	1	0
$F(I(n))$	7	3	1	1	1	0	0	0

Hence, we have

$$d_{ms}(n) = (1 - 2^{-5})d_{ms}(n-1) + 2^{-5}F(I(n)) \tag{11.11}$$

and

$$d_{ml}(n) = (1 - 2^{-7})d_{ml}(n-1) + 2^{-7}F(I(n)) \tag{11.12}$$

while the variable $a_p(n)$ is given by

$$a_p(n) = \begin{cases} (1 - 2^{-4})a_p(n-1) + 2^{-3} & \text{if} \quad |d_{ms}(n) - d_{ml}(n)| \geq 2^{-3}d_{ml}(n) \\ (1 - 2^{-4})a_p(n-1) + 2^{-3} & \text{if} \quad y(n)\langle 3 \\ (1 - 2^{-4})a_p(n) + 2^{-3} & \text{if} \quad t_d(n) = 1 \\ 1 & \text{if} \quad t_r(n) = 1 \\ (1 - 2^{-4})a_p(n) & \text{otherwise} \end{cases} \tag{11.13}$$

$a_p(n)$ approaches 2 when the difference between $d_{ms}(n)$ and $d_{ml}(n)$ is large and approaches 0 when the difference is small. Also $a_p(n)$ approaches 2 for an idle channel (indicated by $y(n) <3$) or partial band signals (indicated by $t_d(n) = 1$). Finally, $a_p(n)$ is set to 1 when a partial band signal transition is detected ($t_r(n) = 1$).

$a_l(n)$, which is used in Equation (11.10), is defined as

$$a_l(n) = \begin{cases} 1 & a_p(n-1) \rangle 1 \\ a_p(n-1) & a_p(n-1) \leq 1 \end{cases} \tag{11.14}$$

The partial band signal $t_d(n)$ and the partial band signal transition $t_r(n)$ that appear in Equation (11.13) will be discussed later.

The predictor computes the signal estimate $\tilde{x}(n)$ from the quantized difference signal $d_q(n)$. The predictor z-transfer function, which is suitable for a variety of input signals, is given by

$$\frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 - a_1 z^{-1} - a_2 z^{-2}} \tag{11.15}$$

It consists of a fifth-order portion that models the zeros and a second-order portion that models the poles of the input signals. The input signal estimate is expressed in terms of the processed signal $\hat{x}(n)$ and the signal $x_z(n)$ processed by the finite impulse response (FIR) filter as follows:

$$\tilde{x}(n) = a_1(n)\hat{x}(n-1) + a_2(n)\hat{x}(n-2) + x_z(n) \tag{11.16}$$

where

$$\hat{x}(n-i) = \tilde{x}(n-i) + d_q(n-i) \tag{11.17}$$

$$x_z(n) = \sum_{i=0}^{5} b_i(n) d_q(n-i) \tag{11.18}$$

Both sets of predictor coefficients are updated using a simplified gradient algorithm:

$$a_1(n) = (1 - 2^{-8}) a_1(n-1) + 3 \cdot 2^{-8} signn(p(n)) sign(p(n-1)) \tag{11.19}$$

$$\begin{aligned} a_2(n) = (1 - 2^{-7}) a_2(n-1) + 2^{-7} \{ & signn(p(n)) sign(p(n-2)) \\ & - f(a_1(n-1)) signn(p(n)) sign(p(n-1)) \} \end{aligned} \tag{11.20}$$

where $p(n) = d_q(n) + x_z(n)$ and

$$f(a_1(n)) = \begin{cases} 4a_1(n) & |a_1(n)| \leq 2^{-1} \\ 2\, sign(a_1(n)) & |a_1(n)| > 2^{-1} \end{cases} \tag{11.21}$$

Note that the function *sign(x)* is defined in Equation (11.5), while the function *signn(x)* equals 1 when $x > 0$, equals 0 when $x = 0$; and equals $= -1$ when $x < 0$ with stability constrains

$$|a_2(n)| \leq 0.75 \text{ and } |a_1(n)| \leq 1 - 2^{-4} - a_2(n) \tag{11.22}$$

$$a_1(n) = a_2(n) = 0 \quad \text{if} \quad t_r(n) = 1 \tag{11.23}$$

Also, the equations for updating the coefficients for the zero-order portion are given by

$$b_i(n) = (1 - 2^{-8}) b_i(n-1) + 2^{-7} signn(d_q(n)) sign(d_q(n-i)) \text{ for } i = 0, 1, 2, \cdots, 5 \tag{11.24}$$

with the following constrains:

$$b_0(n) = b_1(n) = b_2(n) = b_3(n) = b_4(n) = b_5(n) = 0 \quad \text{if} \quad t_r(n) = 1 \tag{11.25}$$

$$t_d(n) = \begin{cases} 1 & a_2(n) < -0.71875 \\ 0 & otherwise \end{cases} \tag{11.26}$$

$$t_r(n) = \begin{cases} 1 & a_2(n) < -0.71875 \quad and \quad |d_q(n)| > 24 \cdot 2^{y_l} \\ 0 & otherwise \end{cases} \tag{11.27}$$

$t_d(n)$ is the indicator that detects a partial band signal (tone). If a tone is detected ($t_d(n) = 1$), Equation (11.13) is invoked to drive the quantizer into the fast mode of adaptation. $t_r(n)$ is the indicator for a transition from a partial band signal. If it is detected ($t_r(n) = 1$), setting the predictor coefficients to be zero as shown in Equations (11.23) and (11.25) will force the quantizer into the fast mode of adaptation.

Simulation Example

To illustrate performance, we apply the ADPCM encoder to the speech data used in Section 11.1 and then operate the ADPCM decoder to recover the speech signal. As described, the ADPCM uses 4 bits to encode each speech sample. The MATLAB implementations for the encoder and decoder are listed in Programs 11.11 to 11.13 in Section 11.7. Figure 11.13 plots the original speech samples, decoded

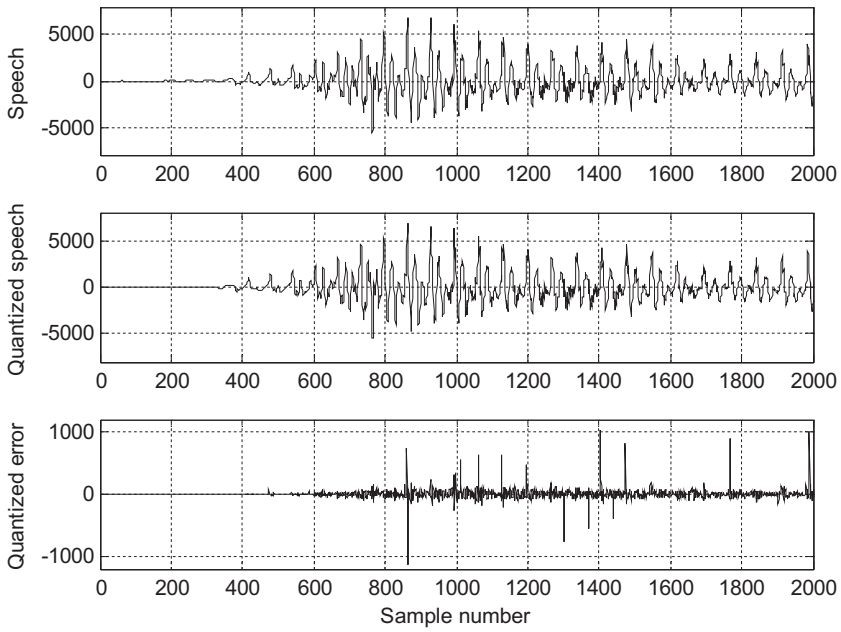

FIGURE 11.13

Original speech, quantized speech, and quantization error using ADPCM.

speech samples, and the quantization errors. From the figure, we see that the decoded speech data are very close to the original speech data; the quantization error is very small as compared with the speech sample, and its amplitude follows the change in amplitude of the speech data. In practice, we cannot tell the difference between the original speech and the decoded speech by listening to them. However, ADPCM encodes each speech sample using 4 bits per sample, while the original data are presented using 16 bits; thus the compression ratio (CR) of 4:1 is achieved.

In practical applications, data compression can reduce the storage media and bit rate for efficient digital transmission. To measure performance of data compression, we use

- the data CR, which is the ratio of original data file size to the compressed data file size, or ratio of the original code size in bits to the compressed code size in bits for fixed-length coding, and
- the bit rate, which is in terms of bits per second (bps) and can be calculated by

$$\text{bit rate} = m \times f_s \text{ (bps)} \tag{11.28}$$

where $m =$ number of bits per sample (bits) and $f_s =$ sampling rate (samples per second).

Now, let us look at an application example.

EXAMPLE 11.7

Speech is sampled at 8 kHz and each sample is encoded at 12 bits per sample. Using (1) no compression, (2) standard μ-law compression, and (3) standard ADPCM encoding,

a. determine the CR and the bit rate for each of the encoders and decoders;
b. determine the number of channels that the phone company can carry if a telephone system can transport the digital voice channel over a digital link with a capacity of 1.536 Mbps.

Solution:

a(1). For no compression:

$$\text{CR} = 1:1$$

$$\text{Bit rate} = 12\frac{bits}{sample} \times 8{,}000\frac{sample}{second} = 96 \text{ (kbps)}$$

b(1).

$$\text{Number of channels} = \frac{1.536}{96}\frac{MBPS}{KBPS} = 16$$

a(2). For standard μ-law compression, each sample is encoded using 8 bits per sample. Hence, we have

$$\text{CR} = \frac{12\ bits/sample}{8\ bits/sample} = 1.5:1$$

$$\text{Bit rate} = 8\frac{bits}{sample} \times 8{,}000\frac{sample}{second} = 64 \text{ (kbps)}$$

b(2).

$$\text{Number of channels} = \frac{1.536}{64}\frac{MBPS}{KBPS} = 24$$

a(3). For standard ADPCM with 4 bits per sample, it follows that

$$\text{CR} = \frac{12}{4}\frac{bits/sample}{bits/sample} = 3:1$$

$$\text{Bit rate} = 4\frac{bits}{sample} \times 8{,}000\frac{sample}{second} = 32 \text{ (kbps)}$$

b(3).

$$\text{Number of channels} = \frac{1.536}{32}\frac{MBPS}{KBPS} = 48.$$

11.4 DISCRETE COSINE TRANSFORM, MODIFIED DISCRETE COSINE TRANSFORM, AND TRANSFORM CODING IN MPEG AUDIO

This section introduces *discrete cosine transform* (DCT) and explains how to apply it in transform coding. This section will also show how to remove the block effects in transform coding using a modified DCT (MDCT). Finally, we will examine how MDCT coding is used in the MPEG (Motion Picture Experts Group) audio format, which is used as a part of MPEG audio, such as MP3 (MPEG-1 layer 3).

11.4.1 Discrete Cosine Transform

Given N data samples, we define the one-dimensional (1D) DCT pair as follows:

Forward transform:

$$X_{DCT}(k) = \sqrt{\frac{2}{N}}C(k)\sum_{n=0}^{N-1}x(n)\cos\left[\frac{(2n+1)k\pi}{2N}\right],\ k = 0, 1, \cdots, N-1 \tag{11.29}$$

Inverse transform:

$$x(n) = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1}C(k)X_{DCT}(k)\cos\left[\frac{(2n+1)k\pi}{2N}\right],\ n = 0, 1, \cdots, N-1 \tag{11.30}$$

$$C(k) = \begin{cases} \dfrac{\sqrt{2}}{2} & k = 0 \\ 1 & otherwise \end{cases} \tag{11.31}$$

where $x(n)$ is the input data sample and $X_{DCT}(k)$ is the DCT coefficient. The DCT transforms the time domain signal to frequency domain coefficients. However, unlike the discrete Fourier transform (DFT), there are no complex number operations for both the forward and inverse transforms. Both the forward and inverse transforms use the same scale factor:

$$\sqrt{\frac{2}{N}}C(k)$$

In terms of transform coding, the DCT decomposes a block of data into a direct-current (DC) coefficient that corresponds to the average of the data samples and alternating-current (AC) coefficients that correspond to the frequency component (fluctuation). The terms "DC" and "AC" come from basic

electrical engineering. In transform coding, we can quantize the DCT coefficients and encode them into binary information. The inverse DCT can transform the DCT coefficients back to the input data. Let us proceed to Examples 11.8 and 11.9.

EXAMPLE 11.8

Assume that the following input data can each be encoded by 5 bits, including a sign bit:

$$x(0) = 10,\ x(1) = 8,\ x(2) = 10 \text{ and } x(3) = 12$$

a. Determine the DCT coefficients.
b. Use the MATLAB function **dct()** to verify all the DCT coefficients.

Solution:

a. Using Equation (11.29) leads to

$$X_{DCT}(k) = \sqrt{\frac{1}{2}}C(k)\left[x(0)\cos\left(\frac{\pi k}{8}\right) + x(1)\cos\left(\frac{3\pi k}{8}\right) + x(2)\cos\left(\frac{5\pi k}{8}\right) + x(3)\cos\left(\frac{7\pi k}{8}\right)\right]$$

When $k = 0$, we see that the DC component is calculated as

$$X_{DCT}(0) = \sqrt{\frac{1}{2}}C(0)\left[x(0)\cos\left(\frac{\pi \times 0}{8}\right) + x(1)\cos\left(\frac{3\pi \times 0}{8}\right) + x(2)\cos\left(\frac{5\pi \times 0}{8}\right) + x(3)\cos\left(\frac{7\pi \times 0}{8}\right)\right]$$

$$= \sqrt{\frac{1}{2}} \times \frac{\sqrt{2}}{2}[x(0) + x(1) + x(2) + x(3)] = \frac{1}{2}(10 + 8 + 10 + 12) = 20$$

We clearly see that the first DCT coefficient is a scaled average value.

For $k = 1$,

$$X_{DCT}(1) = \sqrt{\frac{1}{2}}C(1)\left[x(0)\cos\left(\frac{\pi \times 1}{8}\right) + x(1)\cos\left(\frac{3\pi \times 1}{8}\right) + x(2)\cos\left(\frac{5\pi \times 1}{8}\right) + x(3)\cos\left(\frac{7\pi \times 1}{8}\right)\right]$$

$$= \sqrt{\frac{1}{2}} \times 1\left[10 \times \cos\left(\frac{\pi}{8}\right) + 8 \times \cos\left(\frac{3\pi}{8}\right) + 10 \times \cos\left(\frac{5\pi}{8}\right) + 12 \times \cos\left(\frac{7\pi}{8}\right)\right] = -1.8478$$

Similarly, we have

$$X_{DCT}(2) = 2 \text{ and } X_{DCT}(3) = 0.7654$$

b. Using the MATLAB 1D-DCT function **dct(),** we can verify the DCT coeffcients:

```
>> dct([10 8 10 12])
ans = 20.0000 −1.8478 2.0000 0.7654
```

EXAMPLE 11.9

Assume the following DCT coefficients:

$$X_{DCT}(0) = 20,\ X_{DCT}(1) = -1.8478,\ X_{DCT}(0) = 2, \quad \text{and} \quad X_{DCT}(0) = 0.7654$$

a. Determine $x(0)$.
b. Use the MATLAB function **idct()** to verify all the recovered data samples.

Solution:

a. Applying Equations (11.30) and (11.31), we have

$$
\begin{aligned}
x(0) &= \sqrt{\frac{1}{2}}\left[C(0)X_{DCT}(0)\cos\left(\frac{\pi}{8}\right) + C(1)X_{DCT}(1)\cos\left(\frac{3\pi}{8}\right)\right. \\
&\quad \left. + C(2)X_{DCT}(2)\cos\left(\frac{5\pi}{8}\right) + C(3)X_{DCT}(3)\cos\left(\frac{7\pi}{8}\right)\right] \\
&= \sqrt{\frac{1}{2}}\left[\frac{\sqrt{2}}{2}\times 20\times\cos\left(\frac{\pi}{8}\right) + 1\times(-1.8478)\times\cos\left(\frac{3\pi}{8}\right)\right. \\
&\quad \left. + 1\times 2\times\cos\left(\frac{5\pi}{8}\right) + 1\times 0.7654\times\cos\left(\frac{7\pi}{8}\right)\right] = 10
\end{aligned}
$$

b. With the MATLAB 1D inverse DCT function **idct(),** we obtain
>> idct([20 −1.8478 2 0.7654])
ans = 10.0000 8.0000 10.0000 12.0000

We verify that the input data samples are the same as those in Example 11.8.

In Example 11.9, we obtained an exact recovery of the input data from the DCT coefficients, since infinite precision of each DCT coefficient is preserved. However, in transform coding, each DCT coefficient is quantized using the number of bits per sample assigned by a bit allocation scheme. Usually the DC coefficient requires a larger number of bits to encode, since it carries more energy of the signal, while each AC coefficient requires a smaller number of bits to encode. Hence, the quantized DCT coefficients approximate the DCT coefficients in infinite precision, and the recovered input data with the quantized DCT coefficients will certainly have quantization errors.

EXAMPLE 11.10

Assuming the DCT coefficients

$$X_{DCT}(0) = 20,\ X_{DCT}(1) = -1.8478,\ X_{DCT}(2) = 2, \quad \text{and} \quad X_{DCT}(3) = 0.7654$$

in infinite precision, we recovered the exact data 10, 8, 10, and 12; this was verified in Example 11.9. If a bit allocation scheme quantizes the DCT coefficients using a scale factor of 4 in the form

$$X_{DCT}(0) = 4\times 5 = 20,\ X_{DCT}(1) = 4\times(-0) = 0,\ X_{DCT}(2) = 4\times 1 = 4, \quad \text{and} \quad X_{DCT}(3) = 4\times 0 = 0$$

we can code the scale factor of 4 with 3 bits (magnitude bits only), the scaled DC coefficient of 5 with 4 bits (including a sign bit), and the scaled AC coefficients of 0, 1, and 0 using 2 bits each. 13 bits in total are required.

Use the MATLAB function **idct()** to recover the input data samples.

Solution:
Using the MATLAB function **idct()** and the quantized DCT coefficients, we obtain

>> idct([20 0 4 0])
ans = 12 8 8 12

As we see, the original sample requires 5 bits (4 magnitude bits and 1 sign bit) to encode each of 10, 8, 10, and 12 for a total of 20 bits. Hence, 7 bits are saved for coding this data block using the DCT. We expect many more bits to be saved in practice, in which a longer frame of the correlated data samples is used. However, quantization errors are introduced.

For comprehensive coverage of the topics on DCT, see Li and Drew (2004), Nelson (1992), Saywood (2000), and Stearns (2003).

11.4.2 Modified Discrete Cosine Transform

In the previous section, we observed how a 1D-DCT is adopted for coding a block of data. When we apply the 1D-DCT to audio coding, we first divide the audio samples into blocks and then transform each block of data with DCT. The DCT coefficients for each block are quantized according to the bit allocation scheme. However, when we decode DCT blocks, we encounter edge artifacts at boundaries of the recovered DCT blocks, since DCT coding is block based. This effect of edge artifacts produces periodic noise and is annoying in the decoded audio. To solve for such a problem, the windowed MDCT has been developed (described in Pan, 1995; Princen and Bradley, 1986). The principles are illustrated in Figure 11.14. As we shall see, the windowed MDCT is used in MPEG-1 MP3 audio coding.

We describe and discuss only the main steps for coding data blocks using the windowed MDCT (W-MDCT) based on Figure 11.14.

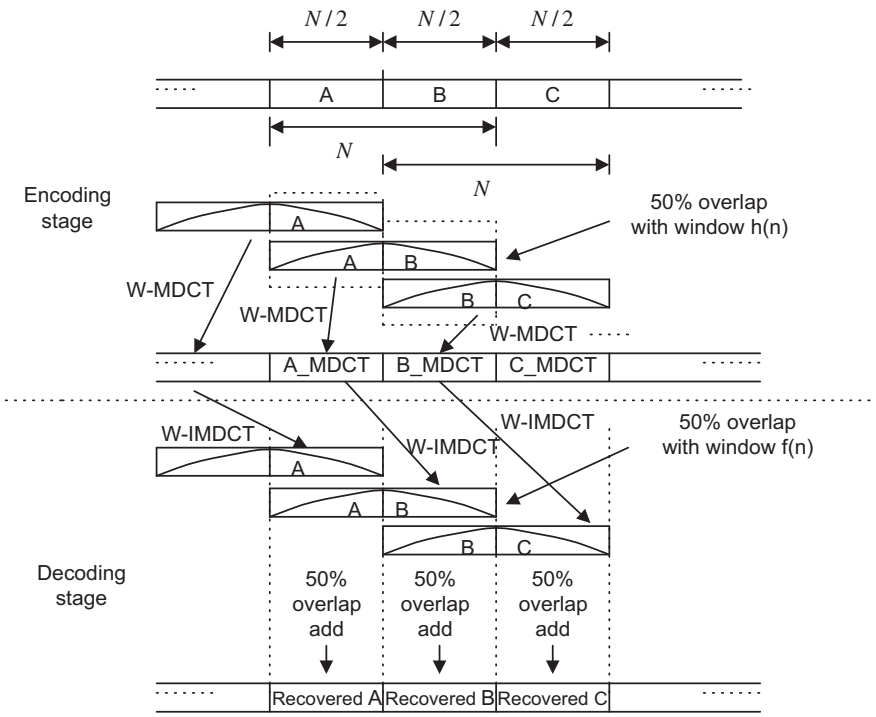

FIGURE 11.14

Modified discrete cosine transform (MDCT).

Encoding stage:

1. Divide the data samples into blocks that each have N (must be an even number) samples, and further divide each block into two subblocks, each with $N/2$ samples for data overlap purposes.
2. Apply the window function for the overlapped blocks. As shown in Figure 11.14, if one block contains the subblocks A and B, the next one would consist of subblocks B and C. The subblock B is the overlapped block. This procedure continues. A window function $h(n)$ is applied to each block with N samples to reduce possible edge effects. Next, the W-MDCT is applied. The W-MDCT is given by

$$X_{MDCT}(k) = 2\sum_{n=0}^{N-1} x(n)h(n)\cos\left[\frac{2\pi}{N}(n+0.5+N/4)(k+0.5)\right] \quad \text{for} \quad k = 0,1,\cdots,N/2-1 \tag{11.32}$$

   Note that we need to compute and encode only half of the MDCT coefficients (since the other half can be reconstructed based on the first half of the MDCT coefficients).
3. Quantize and encode the MDCT coefficients.

Decoding stage:

1. Receive the $N/2$ MDCT coefficients, and use Equation (11.33) to recover the second half of the coefficients:

$$X_{MDCT}(k) = (-1)^{\frac{N}{2}+1} X_{MDCT}(N-1-k), \quad \text{for} \quad k = N/2, N/2+1,\cdots,N-1 \tag{11.33}$$

2. Apply the windowed inverse MDCT (W-IMDCT) to each N MDCT coefficient block using Equation (11.34) and then apply a decoding window function $f(n)$ to reduce the artifacts at the block edges:

$$x(n) = \frac{1}{N} f(n)\sum_{k=0}^{N-1} X_{MDCT}(k)\cos\left[\frac{2\pi}{N}(n+0.5+N/4)(k+0.5)\right] \quad \text{for} \quad n = 0,1,\cdots,N-1 \tag{11.34}$$

   Note that the recovered sequence contains the overlap portion. As shown in Figure 11.14, if a decoded block has the decoded subblocks A and B, the next one would have subblocks B and C, where the subblock B is an overlapped block. The procedure continues.
3. Reconstruct the subblock B using the overlap and add operation, as shown in Figure 11.14, where two subblocks labeled B are overlapped and added to generate the recovered subblock B. Note that the first subblock B comes from the recovered block with N samples containing A and B, while the second subblock B belongs to the next recovered block with N samples consisting of B and C.

In order to obtain the perfect reconstruction, that is, the full cancellation of all aliasing introduced by the MDCT, the following two conditions must be met for selecting the window functions, in which one is used for encoding while the other is used for decoding (Princen and Bradley, 1986):

$$f\left(n+\frac{N}{2}\right)h\left(n+\frac{N}{2}\right)+f(n)h(n) = 1 \tag{11.35}$$

$$f\left(n+\frac{N}{2}\right)h(N-n-1)-f(n)h\left(\frac{N}{2}-n-1\right)\ =\ 0 \tag{11.36}$$

Here, we choose the following simple function for the W-MDCT:

$$f(n)\ =\ h(n)\ =\ \sin\left(\frac{\pi}{N}(n+0.5)\right) \tag{11.37}$$

Equation (11.37) must satisfy the conditions described in Equations (11.35) and (11.36). This will be left for an exercise in the Problems section at the end of this chapter. The MATLAB functions **wmdcth()** and **wimdctf()** relate to this topic and are listed in Programs 11.14-11.5 in section (11.7). Now, let us examine the W-MDCT in Example 11.11.

EXAMPLE 11.11

Given the data 1, 2, −3, 4, 5, −6, 4, 5 ...,

a. determine the W-MDCT coefficients for the first three blocks using a block size of 4;
b. determine the first two overlapped subblocks, and compare the results with the original data sequence using the W-MDCT coefficients in (a).

Solution:

a. We divided the first two data blocks using the overlapping of 2 samples:
First data block: 1 2 −3 4
Second data block −3 4 5 −6
Third data block: 5 −6 4 5
We apply the W-MDCT to get
>> wmdct([1 2 −3 4])
ans = 1.1716 3.6569
>> wmdct([−3 4 5 −6])
ans = −8.0000 7.1716
>> wmdct([5 −6 4 5])
ans =−4.6569 −18.0711
b. The results from W-IWDCT are as follows:
>> x1=wimdct([1.1716 3.6569])
x1 =−0.5607 1.3536 −1.1465 −0.4749
>> x2=wimdct([-8.0000 7.1716])
x2 = −1.8536 4.4749 2.1464 0.8891
>> x3=wimdct([-4.6569 -18.0711])
x3 =2.8536 −6.8891 5.1820 2.1465
Applying the overlap and add, we have
>> [x1 0 0 0 0]+ [0 0 x2 0 0]+ [0 0 0 0 x3]
ans = -0.5607 1.3536 -3.0000 4.0000 5.0000 −6.0000 5.1820 2.1465

The first two recovered subblocks contain the values −3, 4, 5 −6, which are consistent with the input data.

Figure 11.15 shows coding of speech data *we.dat* using the DCT transform and W-MDCT transform. To be able to see the block edge artifacts, the following parameters are used for both DCT and W-MDCT transform coding:

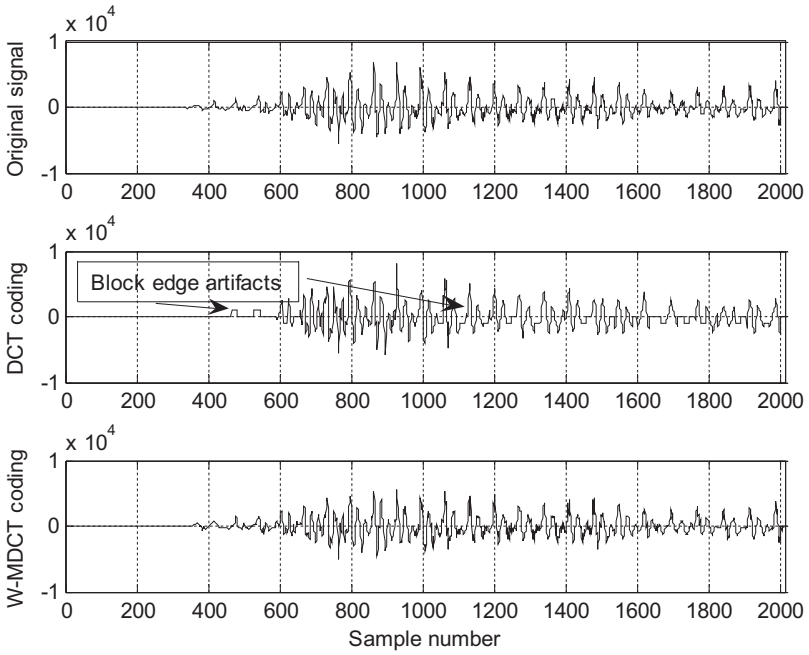

FIGURE 11.15

Waveform coding using DCT and W-MDCT.

Speech data: 16-bits per sample, 8,000 samples per second
Block size: 16 samples
Scale factor: 2-bit nonlinear quantizer
Coefficients: 3-bit linear quantizer

Note that we assume a lossless scheme will further compress the quantized scale factors and coefficients. This stage does not affect the simulation results.

We use a 2-bit nonlinear quantizer with four levels to select the scale factor so that the block artifacts can be clearly displayed in Figure 11.15. We also apply a 3-bit linear quantizer to the scaled coefficients for both DCT and W-MDCT coding. As shown in Figure 11.15, the W-MDCT demonstrates significant improvement in smoothing out the block edge artifacts. The MATLAB simulation code is given in Programs 11.14 to 11.16 in Section 11.7, where Program 11.6 is the main program.

11.4.3 Transform Coding in MPEG Audio

With the DCT and MDCT concepts developed, we now explore the MPEG audio data format, where the DCT plays a key role. MPEG was established in 1988 to develop a standard for delivery of digital video and audio. Since MPEG audio compression contains so many topics, we focus here on examining its data format briefly, using the basic concepts developed in this book. Readers can further explore this subject by reading Pan's (1995) tutorial on MPEG audio compression, as well as Li and Drew (2004).

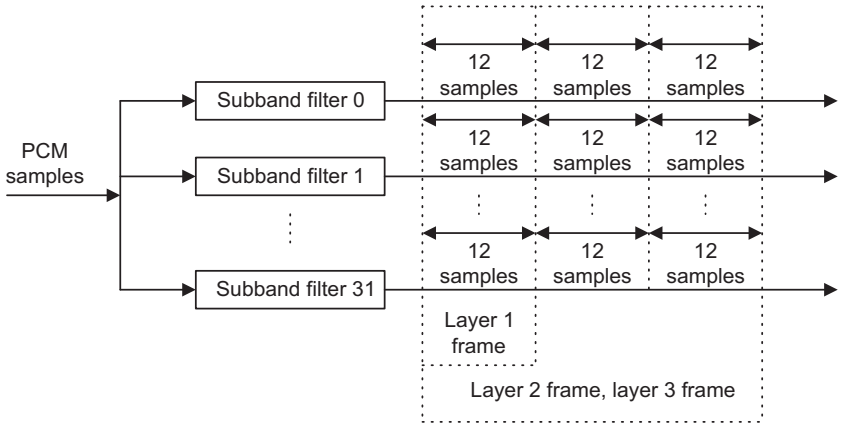

FIGURE 11.16

MPEG audio frame.

Figure 11.16 shows the MPEG audio frame. First, the input PCM samples—with possible sampling rates of 32 kHz, 44.1 kHz, and 48 kHz—are divided into 32 frequency subbands. All the subbands have equal bandwidths. The sum of their bandwidths covers up to the folding frequency, that is, $f_s/2$, which is the Nyquist limit in the DSP system. The subband filters are designed to minimize aliasing in the frequency domain. Each subband filter outputs one sample for every 32 input PCM samples continuously, and forms a data segment for every 12 output samples. The purpose of the filter banks is to separate the data into different frequency bands so that the psycho-acoustic model of the human auditory system (Yost, 1994) can be applied to activate the bit allocation scheme for a particular subband. The data frames are formed before quantization.

There are three types of data frames, as shown in Figure 11.16. Layer 1 contains 32 data segments, each coming from one subband with 12 samples, so the total frame has 384 data samples. As we see, layer 2 and layer 3 have the same size data frame, consisting of 96 data segments, where each filter outputs 3 data segments of 12 samples. Hence, layer 2 and layer 3 each have 1,152 data samples.

Next, let us examine briefly the content of each data frame, as show in Figure 11.17. Layer 1 contains 384 audio samples from 32 subbands, each having 12 samples. It begins with a header followed by a cyclic redundancy check (CRC) code. The numbers within parentheses indicate the possible number of bits to encode each field. The bit allocation informs the decoder of the number of bits used for each encoded sample in the specific band. Bit allocation can also be set to zero bits for a particular subband if analysis of the psycho-acoustic model finds that the data in the band can be discarded without affecting the audio quality. In this way, the encoder can achieve more data compression. Each scale factor is encoded with 6 bits. The decoder will multiply the scale factor by the decoded quantizer output to get the quantized subband value. Use of the scale factor allows for utilization of the full range of the quantizer. The field "ancillary data" is reserved for "extra" information.

The layer 2 encoder takes 1,152 samples per frame, with each subband channel having 3 data segments of 12 samples. These 3 data segments may have a bit allocation and up to to 3 scale factors.

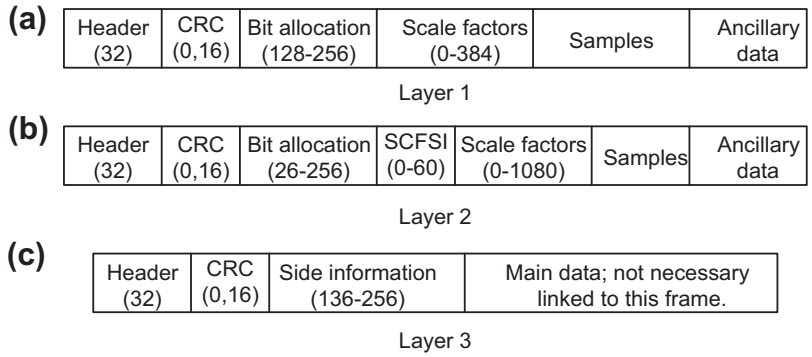

FIGURE 11.17

MPEG audio frame formats.

Using one scale factor for 3 data segments would be called for when values of the scale factors per subband are sufficiently close and the encoder applies temporal noise masking (a type of noise masking by the human auditory system) to hide any distortion. In Figure 11.17, the field "SCFSI" (scale-factor selection information) contains the information to inform the decoder. A different scale factor is used for each subband channel when avoidance of audible distortion is required. The bit allocation can also provide a possible single compact code word to represent three consecutive quantized values.

The layer 3 frame contains side information and main data that come from Huffman encoding (lossless coding with an exact recovery) of the W-MDCT coefficients to gain improvement over layer 1 and layer 2.

Figure 11.18 shows the MPEG-1 layer 1 and 2 encoder, and the layer 3 encoder. For MPEG-1 layer 1 and layer 2, the encoder examines the audio input samples using a 1,024-point fast Fourier transform (FFT). The psycho-acoustic model is analyzed based on the FFT coefficients. This includes possible frequency masking (hiding noise in frequency domain) and noise temporal masking (hiding noise in time domain). The result of the analysis of the psycho-acoustic model instructs the bit allocation scheme.

The major difference in layer 3, called MP3 (the most popular format in the multimedia industry), is that it adopts the MDCT. First, the encoder can gain further data compression by transforming the data segments from each subband channel using DCT and then quantizing the DCT coefficients, which, again, are losslessly compressed using Huffman encoding. As shown in Examples 11.8 to 11.11, since the DCT uses block-based processing, it produces block edge effects, where the beginning samples and ending samples show discontinuity and cause audible periodic noise. This periodic edge noise can be alleviated, as discussed in the previous section, by using the W-MDCT, in which there is 50% overlap between successive transform windows.

There are two sizes of windows. One has 36 samples and other 12 samples used in MPEG-1 layer 3 (MP3) audio. The larger block length offers better frequency resolution for low-frequency tonelike signals, hence it is used for the lowest two subbands. For the rest of the subbands, the shorter block is used, since it allows better time resolution for noiselike transient signals. Other improvements of MP3

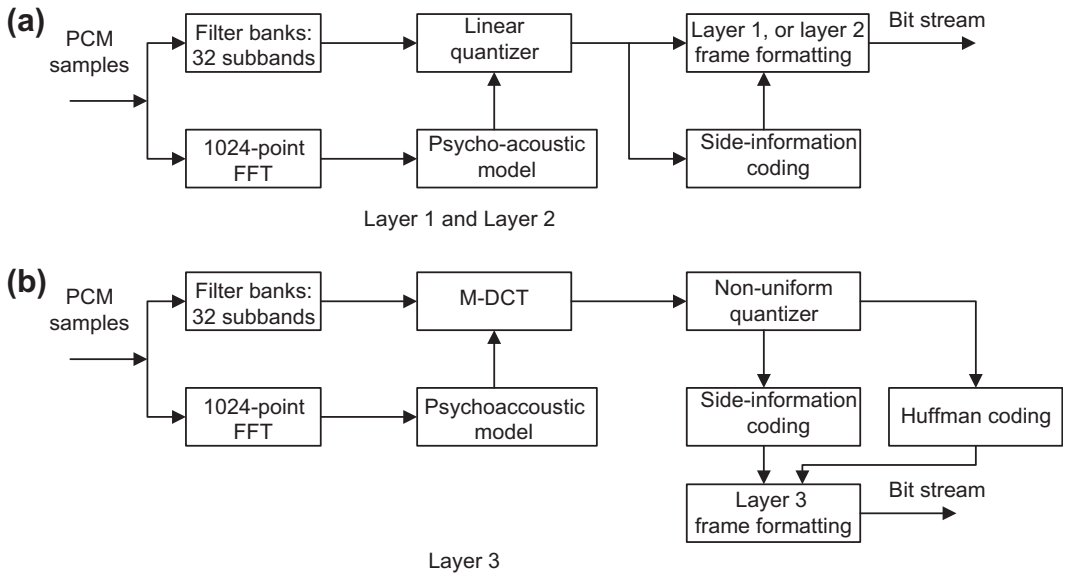

FIGURE 11.18

Encoder block diagrams for layers 1 and 2 and for layer 3.

over layers 1 and 2 include use of the scale-factor band, where the W-MDCT coefficients are regrouped from the original 32 uniformly divided subbands into 25 actual critical bands based on the human auditory system. Then the corresponding scale factors are assigned, and a nonlinear quantizer is used.

Finally, Huffman coding is applied to the quantizer outputs to obtain more compression. Particularly in CD-quality audio, MP3 (MPEG-1 layer 3) can achieve CRs varying from 12:1 to 8:1, corresponding to bit rates from 128 kbps to 192 kbps. Besides the use of DCT in MP3, MPEG-2 audio coding methods such as AC-2, AC-3, ATRAC, and PAC/MPAC also use W-MDCT coding. Readers can further explore these subjects in Brandenburg (1997) and Li and Drew (2004).

11.5 LABORATORY EXAMPLES OF SIGNAL QUANTIZATION USING THE TMS320C6713 DSK

Linear quantization can be implemented as shown in C Program 11.1. The program only demonstrates left channel coding; right channel coding can be easily extended from the program. The program consists of both an encoder and decoder. First, it converts the 16-bit 2's complement data to the sign-magnitude format with truncated magnitude bits as required. Then the decoder converts the compressed PCM code back to the 16-bit data. The encoding and decoding are explained in Example 11.12.

EXAMPLE 11.12

Given a 16-bit datum lc = −2050 (decimal), convert it to a 5-bit linear PCM code using the information in Program 11.1.

Solution:

a. Encoding:
   tmp = 2050 (decimal) = 0x0802 (hex) = 0000 1000 0000 0010 (binary)
   After shifting 11 bits, tmp = 0x0001 (hex)
   PCMcode = tmp&mask[5-1]=0x0001&0x000f=0x0001 (hex) // Get magnitude bits
   if lc >0 {
   PCMcode = PCMcode | sign[5-1] // Add positive sign bit
   } // This line is not executed since lc < 0
   PCMcode = 00001 (binary)

b. Decoding:
   dec = PCMcode & mask[4] = 0x0001&0x000f=0x0001 (hex)
   tmp = PCMcode&sign[4] = 0x0001&0x0010=0x0000 (hex) (the number is negative)
   dec = dec<<11=0x0800
   dec =dec | rec[4]=0x0800| 0x0400=0x0C00 (hex)=3072 (decimal)
   lc = −3072 (recovered decimal)

As expected, the recovered PCM code is different from the original code, since a PCM is the lossy coding scheme. The same procedure can be followed for coding a positive decimal number.

C Program 11.1. Encoding and decoding using the TMS320C6713 DSK.

```
int PCMcode;
/* Sign-magnitude format: s magnitude bits, see Section 11.1, in Chapter 11 */
/* convert a 16-bit PCM code a 5-bit PCM code, and recover the 5-bit PCM code to the 16-bit PCM code */
// See the following example:
/* Convert a 16-bit sign-magnitude code 5-bit PCM code 16-bit recovered sign-magnitude code */
/* sADCDEFGHIJKLMNO     sABCDE     sABCD10000000000 */
/* Note that audio input/output data are in 2's complement form */
/* For encoder, convert the 2's complement input to the sign-magnitude form before
quantization         */
/* For decoder, convert the sign-magnitude form to the 2's complement output before DAC  */
int nofbits=5; // specify the number of quantization bits
int sign[16]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
      0x100,0x200,0x400,0x800,0x1000,0x2000,0x4000,0x8000}; // add sign bit (MSB)
int mask[16]={0x0,0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,
      0xff,0x1ff,0x3ff,0x7ff,0xfff,0x1fff,0x3fff,0x7fff}; // mask for obtaining
magnitude bits
int rec[16]={0x4000,0x2000,0x1000,0x0800,
      0x0400,0x0200,0x0100,0x0080,
      0x0040,0x0020,0x0010,0x0008,
      0x0004,0x0002,0x0001,0x0000};// the midpoint of the quantization interval
interrupt void c_int11()
{
  int lc; /*left channel input */
  int rc; /*right channel input */
  int lcnew; /*left channel output */
```

```
  int rcnew; /*right channel output */
  int temp, dec;
//Left channel and right channel inputs
AIC23_data.combo=input_sample();
  lc=(int) (AIC23_data.channel[LEFT]);
  rc= (int) (AIC23_data.channel[RIGHT]);
// Insert DSP algorithm below
 /* Encoder :*/
 tmp =lc; // for the Left Line In channel
  if (tmp <0 )
  {
   tmp=-tmp; // Get magnitude bits to work with
  }
  tmp =(tmp>>(16-nofbits));
  PCMcode=tmp&mask[nofbits-1]; // Get magnitude bits
 if (lc>=0)
 {
   PCMcode= PCMcode | sign[nofbits-1]; // Add sign bit
 }
 /* PCM code (stored in the lower portion) */
 /* Decoder: */
  dec = PCMcode&mask[nofbits-1]; // Obtain magnitude bits
  tmp= PCMcode&sign[nofbits-1]; // Obtain the sign bit
  dec = (dec<<(16-nofbits)); // Scale to 15-bit magnitude
  dec = dec | rec[nofbits-1]; // Recover the midpoint of the quantization interval
 lc =dec;
 if (tmp == 0x00)
 {
  lc =-dec; // Back to 2's complement form (change the sign)
 }
// End of the DSP algorithm
  lcnew=lc; /* Send to DAC */
  rcnew=lc;
  AIC23_data.channel[LEFT]=(short) lcnew;
  AIC23_data.channel[RIGHT]=(short) rcnew;
  output_sample(AIC23_data.combo);
}
```

C Program 11.2 demonstrates digital μ-law encoding and decoding. It converts a 12-bit linear PCM code to an 8-bit compressed PCM code using the principles discussed in Section 11.2. Note that the program only performs left-channel coding.

C Program 11.2. Digital μ-law encoding and decoding.

```
int ulawcode;
/* Digital mu-law definition*/
// Sign-magnitude format: s segment quantization
// s = 1 for the positive value, s = 0 for the negative value
// Segment defines compression
// quantization with 16 levels
```

```
// See Section 11.2, Chapter 11
/* Segment  12-bit PCM  4-bit quantization interval */
/* 0    s0000000ABCD    s000ABCD      */
/* 1    s0000001ABCD    s001ABCD      */
/* 2    s000001ABCDX   s010ABCD       */
/* 3    s00001ABCDXX   s011ABCD       */
/* 4    s0001ABCDXXX   s100ABCD       */
/* 5    s001ABCDXXXX  s101ABCD        */
/* 6    s01ABCDXXXXX  s110ABCD        */
/* 7    s1ABCDXXXXXX  s111ABCD        */
//
/* Segment Recovered 12-bit PCM       */
/* 0    s0000000ABCD                  */
/* 1    s0000001ABCD                  */
/* 2    s000001ABCD1                  */
/* 3    s00001ABCD10                  */
/* 4    s0001ABCD100                  */
/* 5    s001ABCD1000                  */
/* 6    s01ABCDq0000                  */
/* 7    s1ABCD100000                  */
/* Note that audio input/output data are in 2's complement form */
/* For encoder, convert the 2's complement form to the sign-magnitude form before
quantization         */
/* For decoder, convert the sign-magnitude form to the 2's complement output before DAC  */
interrupt void c_int11()
{
  int lc; /*left channel input */
  int rc; /*right channel input */
  int lcnew; /*left channel output */
  int rcnew; /*right channel output */
  int tmp,ulawcode, dec;
//Left channel and right channel inputs
  AIC23_data.combo=input_sample();
  lc=(int) (AIC23_data.channel[LEFT]);
  rc= (int) (AIC23_data.channel[RIGHT]);
// Insert DSP algorithm below
 /* Encoder :*/
 tmp =lc;
 if (tmp <0 )
 { tmp=-tmp; // Get magnitude bits to work with
 }
 tmp =(tmp>>4); // Linear scale down to 12 bits to use the u-255 law table
 if( (tmp&0x07f0)==0x0) // Segment 0
 { ulawcode= (tmp&0x000f); }
 if( (tmp&0x07f0)==0x0010) // Segment 1
 { ulawcode= (tmp&0x00f);
   ulawcode= ulawcode | 0x10; }
 if( (tmp&0x07E0)==0x0020) // Segment 2
```

```
{ ulawcode= (tmp&0x001f)>>1;
  ulawcode = ulawcode |0x20; }
if( (tmp&0x07c0)==0x0040) // Segment 3
{ ulawcode= (tmp&0x003f)>>2;
  ulawcode= ulawcode | 0x30; }
if( (tmp&0x0780)==0x0080) // Segment 4
{ ulawcode= (tmp&0x007f)>>3;
  ulawcode =ulawcode | 0x40;}
if( (tmp&0x0700)==0x0100) // Segment 5
{ ulawcode= (tmp&0x00ff)>>4;
  ulawcode = ulawcode | 0x50;}
if( (tmp&0x0600)==0x0200) // Segment 6
{ ulawcode= (tmp&0x01ff)>>5;
  ulawcode = ulawcode | 0x60; }
if( (tmp&0x0400)==0x0400) // Segment 7
{ ulawcode= (tmp&0x03ff)>>6;
  ulawcode=ulawcode | 0x70; }
if (lc>=0)
{
  ulawcode= ulawcode|0x80;
}
/* u-law code (8 bit compressed PCM code) for transmission and storage */
/* Decoder: */
  tmp = ulawcode&0x7f;
  tmp = (tmp>>4);
if ( tmp == 0x0) // Segment 0
{ dec = ulawcode&0xf; }
if ( tmp == 0x1) // Segment 1
{ dec = ulawcode&0xf | 0x10; }
if ( tmp == 0x2) // Segment 2
{ dec = ((ulawcode&0xf)<<1) | 0x20;
  dec= dec |0x01; }
if ( tmp == 0x3) // Segment 3
{ dec = ((ulawcode&0xf)<<2) | 0x40;
  dec = dec|0x02; }
if ( tmp == 0x4) // Segment 4
{ dec = ((ulawcode&0xf)<<3) | 0x80;
  dec = dec|0x04; }
if ( tmp == 0x5) // Segment 5
{ dec = ((ulawcode&0xf)<<4) | 0x0100;
  dec = dec|0x08; }
if ( tmp == 0x6) // Segment 6
{ dec = ((ulawcode&0xf)<<5) | 0x0200;
  dec =dec|0x10; }
if ( tmp == 0x7) // Segment 7
{ dec = ((ulawcode&0xf)<<6) | 0x0400;
  dec = dec |0x20; }
 tmp =ulawcode & 0x80;
```

```
 lc =dec;
 if (tmp == 0x00)
 { lc=-dec; // Back to 2's complement form
 }
 lc= (lc<<4); // Linear scale up to 16 bits
// End of the DSP algorithm
  lcnew=lc; /* Send to DAC */
  rcnew=lc;
  AIC23_data.channel[LEFT]=(short) lcnew;
  AIC23_data.channel[RIGHT]=(short) rcnew;
  output_sample(AIC23_data.combo);
}
```

11.6 SUMMARY

1. The linear midtread quantizer used in PCM coding has an odd number of quantization levels, that is, $2^n - 1$. It accommodates the same decoded magnitude range for quantizing the positive and negative voltages.
2. Analog and digital μ-law compression improve coding efficiency. 8-bit μ-law compression of speech is equivalent to 12-bit linear PCM coding, with no difference in sound quality. These methods are widely used in the telecommunications industry and in multimedia system applications.
3. DPCM encodes the difference between the input sample and predicted sample using a predictor to achieve coding efficiency.
4. DM coding is essentially a 1-bit DPCM.
5. ADPCM is similar to DPCM except that the predictor transfer function has six zeros and two poles and is an adaptive filter. ADPCM is superior to 8-bit μ-law compression, since it provides the same sound quality with only 4 bits per code.
6. Data compression performance is measured in terms of the data compression ratio and the bit rate.
7. The DCT decomposes a block of data to the DC coefficient (average) and AC coefficients (fluctuation) so that different numbers of bits are assigned to encode DC coefficients and AC coefficients to achieve data compression.
8. W-MDCT alleviates the block effects introduced by the DCT.
9. MPEG-1 audio formats such as MP3 (MPEG-1, layer 3) include W-MDCT, filter banks, a psychoacoustic model, bit allocation, a nonlinear quantizer, and Huffman lossless coding.

11.7 MATLAB PROGRAMS

Program 11.1 MATLAB program for the linear midtread quantizer.

```
clear all;close all
disp('load speech: We');
load we.dat;              % Provided by your instructor
sig = we;
```

```
lg=length(sig);             % Length of the speech data
t=[0:1:lg-1];               % Time index
sig=5*sig/max(abs(sig));    % Normalize signal to the range between -5 to 5
Emax = max(abs(sig));
Erms = sqrt( sum(sig .* sig) / length(sig))
k=Erms/Emax
disp('20*log10(k)=>');
k = 20*log10(k)
bits = input('input number of bits =>');
lg = length(sig);
% Encoding
for x=1:lg
 [indx(x) qy] = mtrdenc(bits, 5, sig(x));
end
disp('Finished and transmitted');
% Decoding
for x=1:lg
 qsig(x) = mtrddec(bits, 5, indx(x));
end
disp('decoding finished');
 qerr = sig-qsig;     % Calculate quantization errors
subplot(3,1,1);plot(t, sig);grid
ylabel('Speech');axis([0 length(we) -5 5]);
subplot(3,1,2);plot(t, qsig);grid
ylabel('Quantized speech');axis([0 length(we) -5 5]);
subplot(3,1,3);plot(qerr);grid
axis([0 length(we) -0.5 0.5]);
ylabel('Qunatized error');xlabel('Sample number');
disp('signal to noise ratio due to quantization noise')
snr(sig,qsig);      % Calculate signal to noise ratio due to quantization
```

Program 11.2. MATLAB program for μ-law encoding and decoding.

```
close all; clear all
disp('load speech file');
load we.dat;                % Provided by your instructor
lg=length(we);              % Length of the speech data
we=5*we/max(abs(we));       % Normalize the speech data
we_nor=we/max(abs(we));     % Normalization
t=[0:1:lg-1];               % Time index
disp('mulaw companding')
mu=input('input mu =>');
for x=1:lg
 ymu(x) =mulaw(we_nor(x),1,mu);
end
disp('finished mu-law companding');
disp('start to quantization')
bits = input('input bits=>');
% Midtread quantization and encoding
```

```
for x=1:lg
 [indx(x) qy] = mtrdenc(bits, 1, ymu(x));
end
disp('finished and transmitted');
%
% Midtread decoding
for x=1:lg
 qymu(x) = mtrddec(bits, 1, indx(x));
end
disp('expander');
for x=1:lg
 dymu(x)= muexpand(qymu(x),1,mu)*5;
end
disp('finished')
qerr = dymu-we; % Quantization error
subplot(3,1,1);plot(we);grid
ylabel('Speech');axis([0 length(we) -5 5]);
subplot(3,1,2);plot(dymu);grid
ylabel('recovered speech');axis([0 length(we) -5 5]);
subplot(3,1,3);plot(qerr);grid
ylabel('Quantized error');xlabel('Sample number');
axis([0 length(we) -1 1]);
snr(we,dymu);      % Calculate signal to noise ratio due to quantization
```

Program 11.3. MATLAB function for μ-law companding.

```
Function qvalue = mulaw(vin, vmax, mu)
% This function performs mu-law companding
% Usage:
% function qvalue = mulaw(vin, vmax, mu)
% vin = input value
% vmax = input value
% mu = parameter for controlling the degree of compression which must be the same
% qvalue = output value from the mu-law compander
% as the mu-law expander
%
 vin = vin/vmax;    % Normalization
% mu-law companding formula
 qvalue = vmax*sign(vin)*log(1+mu*abs(vin))/log(1+mu);
```

Program 11.4. MATLAB program for μ-law expanding.

```
function rvalue = muexpand(y,vmax, mu)
% This function performs mu-law exoanding
% Usage:
% function rvalue = muexpand(y,vmax, mu)
% y = input signal
% vmax = maximum amplitude
% mu = parameter for controlling the degree of compression, which must be the same
% as the mu-law compander
```

```
% rvalue = output value from the mu-law expander
%
 y=y/vmax;     % Normalization
% mu-law expanding
 rvalue=sign(y)*(vmax/mu)*((1+mu)^abs(y) -1);
```

Program 11.5. MATLAB function for midread quantizer encoding.

```
function [indx, pq ] = mtrdenc(NoBits,Xmax,value)
% function pq = mtrdenc(NoBits, Xmax, value)
% This routine is created for simulation of midtread uniform quantizer.
%
% NoBits: number of bits used in quantization.
% Xmax: overload value.
% value: input to be quantized.
% pq: output of quantized value
% indx: integer index
%
% Note: the midtread method is used in this quantizer.
%
 if NoBits == 0
  pq = 0;
  indx=0;
 else
  delta = 2*abs(Xmax)/(2^NoBits-1);
  Xrmax=delta*(2^NoBits/2-1);
  if abs(value) >= Xrmax
   tmp = Xrmax;
 else
   tmp = abs(value);
 end
 indx=round(tmp/delta);
 pq =indx*delta;
 if value < 0
  pq = -pq;
  indx=-indx;
 end
end
```

Program 11.6. MATLAB function for midtread quantizer decoding.

```
function pq = mtrddec(NoBits,Xmax,indx)
% function pq = mtrddec(NoBits, Xmax, value)
% This routine is the dequantizer.
%
% NoBits: number of bits used in quantization.
% Xmax: overload value.
% pq: output of quantized value
% indx: integer index
%
```

```
% Note: the midtread method is used in this quantizer.
%
 delta = 2*abs(Xmax)/(2^NoBits-1);
 pq =indx*delta;
```

Program 11.7. MATLAB function for calculation of signal to quantization noise ratio (SNR).

```
function snr = calcsnr(speech, qspeech)
% function snr = calcsnr(speech, qspeech)
% This routine was created for calculation of SNR.
%
% speech: original speech waveform.
% qspeech: quantized speech.
% snr: output SNR in dB.
%
% Note: the midrise method is used in this quantizer.
%
 qerr = speech-qspeech;
 snr = 10*log10(sum(speech.*speech)/sum(qerr.*qerr))
```

Program 11.8. Main program for digital μ-law encoding and decoding.

```
load we12b.dat
for i=1:1:length(we12b)
  code8b(i)=dmuenc(12, we12b(i));  % Encoding
  qwe12b(i)=dmudec(code8b(i));   % Decoding
end
subplot(4,1,1),plot(we12b);grid
ylabel('a');axis([0 length(we12b) -1024 1024]);
subplot(4,1,2),plot(code8b);grid
ylabel('b');axis([0 length(we12b) -128 128]);
subplot(4,1,3),plot(qwe12b);grid
ylabel('c');axis([0 length(we12b) -1024 1024]);
subplot(4,1,4),plot(qwe12b-we12b);grid
ylabel('d');xlabel('Sample number');axis([0 length(we12b) -40 40]);
```

Program 11.9. The digital μ-law compressor.

```
function [cmp_code ] = dmuenc(NoBits, value)
% This routine is created for simulation of 12-bit mu law compression.
% function [cmp_code ] = dmuenc(NoBits, value)
% NoBits = number of bits for the data
% value = input value
% cmp_code = output code
%
 scale = NoBits-12;
 value=value*2^(-scale); % Scale to 12 bit
 if (abs(value) >=0) & (abs(value)<16)
   cmp_code=value;
 end
 if (abs(value) >=16) & (abs(value)<32)
```

```
 cmp_code=sgn(value)*(16+fix(abs(value)-16));
end
if (abs(value) >=32) & (abs(value)<64)
 cmp_code=sgn(value)*(32+fix((abs(value) -32)/2));
end
if (abs(value) >=64) & (abs(value)<128)
  cmp_code=sgn(value)*(48+fix((abs(value) -64)/4));
end
if (abs(value) >=128) & (abs(value)<256)
 cmp_code=sgn(value)*(64+fix((abs(value) -128)/8));
end
if (abs(value) >=256) & (abs(value)<512)
 cmp_code=sgn(value)*(80+fix((abs(value) -256)/16));
end
if (abs(value) >=512) & (abs(value)<1024)
 cmp_code=sgn(value)*(96+fix((abs(value) -512)/32));
end
if (abs(value) >=1024) & (abs(value)<2048)
  cmp_code=sgn(value)*(112+fix((abs(value) -1024)/64));
end
```

Program 11.10. The digital μ-law expander.

```
function [value ] = dmudec(cmp_code)
% This routine is created for simulation of 12-bit mu law decoding.
% Usage:
% unction [value ] = dmudec(cmp_code)
% cmp_code = input mu-law encoded code
% value = recovered output value
%
 if (abs(cmp_code) >=0) & (abs(cmp_code)<16)
   value=cmp_code;
 end
 if (abs(cmp_code) >=16) & (abs(cmp_code)<32)
  value=sgn(cmp_code)*(16+(abs(cmp_code)-16));
 end
 if (abs(cmp_code) >=32) & (abs(cmp_code)<48)
  value=sgn(cmp_code)*(32+(abs(cmp_code)-32)*2+1);
 end
 if (abs(cmp_code) >=48) & (abs(cmp_code)<64)
   value=sgn(cmp_code)*(64+(abs(cmp_code)-48)*4+2);
 end
 if (abs(cmp_code) >=64) & (abs(cmp_code)<80)
  value=sgn(cmp_code)*(128+(abs(cmp_code)-64)*8+4);
 end
 if (abs(cmp_code) >=80) & (abs(cmp_code)<96)
  value=sgn(cmp_code)*(256+(abs(cmp_code)-80)*16+8);
 end
 if (abs(cmp_code) >=96) & (abs(cmp_code)<112)
```

```
 value=sgn(cmp_code)*(512+(abs(cmp_code)-96)*32+16);
end
if (abs(cmp_code) >=112) & (abs(cmp_code)<128)
  value=sgn(cmp_code)*(1024+(abs(cmp_code)-112)*64+32);
end
```

Program 11.11. Main program for ADPCM coding.

```
% This program is written for offline simulation.
% file: adpcm.m
clear all; close all
load we.dat                    % Provided by the instructor
speech=we;
desig= speech;
lg=length(desig);              % Length of speech data
enc = adpcmenc(desig);         % ADPCM encoding
%ADPCM finished
dec = adpcmdec(enc);           % ADPCM decoding
snrvalue = snr(desig,dec)      % Calculate signal to noise ratio due to quantization
subplot(3,1,1);plot(desig);grid;
ylabel('Speech');axis([0 length(we) -8000 8000]);
subplot(3,1,2);plot(dec);grid;
ylabel('Quantized speech');axis([0 length(we) -8000 8000]);
subplot(3,1,3);plot(desig-dec);grid
ylabel('Quantized error');xlabel('Sample number');
axis([0 length(we) -1200 1200]);
```

Program 11.12. MATLAB function for ADPCM encoding.

```
function iiout = adpcmenc(input)
% This function performs ADPCM encoding.
% function iiout = adpcmenc(input)
% Usage:
% input = input value
% iiout = output index
%
% Quantization tables
fitable = [0 0 0 1 1 1 1 3 7];
witable = [-0.75 1.13 2.56 4.00 7.00 12.38 22.19 70.13 ];
qtable = [ -0.98 0.62 1.38 1.91 2.34 2.72 3.12 ];
invqtable = [0.031 1.05 1.66 2.13 2.52 2.91 3.32 ];
lgth = length(input);
sr = zeros(1,2); pk = zeros(1,2);
a = zeros(1,2); b = zeros(1,6);
dq = zeros(1,6); ii= zeros(1,lgth);
y=0; ap = 0; al = 0; yu=0; yl = 0; dms = 0; dml = 0; tr = 0; td = 0;
for k = 1:lgth
 sl = input(k);
%
% predict zeros
```

```
%
 sez = b(1)*dq(1);
 for i=2:6
 sez = sez + b(i)*dq(i);
end
se = a(1)*sr(1)+a(2)*sr(2)+ sez;
 d = sl - se;
%
% Perform quantization
%
 dqq = log10(abs(d))/log10(2.0)-y;
  ik= 0;
  for i=1:7
   if dqq > qtable(i)
    ik = i;
   end
  end
 if d < 0
  ik = -ik;
 end
 ii(k) = ik;
 yu = (31.0/32.0)*y + witable(abs(ik)+1)/32.0;
 if yu > 10.0
  yu = 10.0;
 end
 if yu < 1.06
  yu = 1.06;
 end
 yl = (63.0/64.0)*yl+yu/64.0;
%
%Inverse quantization
%
 if ik == 0
 dqq = 2^(-y);
 else
 dqq = 2^(invqtable(abs(ik))+y);
 end
 if ik < 0
 dqq = -dqq;
 end
 srr = se + dqq;
 dqsez = srr+sez-se;
%
% Update state
%
pk1 = dqsez;
%
% Obtain adaptive predictor coefficients
```

```
%
  if tr == 1
    a = zeros(1,2); b = zeros(1,6);
    tr = 0;
    td = 0; % Set for the time being
  else
% Update predictor poles
% Update a2 first
  a2p = (127.0/128.0)*a(2);
  if abs(a(1)) <= 0.5
   fa1 = 4.0*a(1);
  else
   fa1 = 2.0*sgn(a(1));
  end
  a2p=a2p+(sign(pk1)*sgn(pk(1))-fa1*sign(pk1)*sgn(pk(2)))/128.0;
  if abs(a2p) > 0.75
   a2p = 0.75*sgn(a2p);
  end
   a(2) = a2p;
%
% Update a1
  a1p = (255.0/256.0)*a(1);
     a1p = a1p + 3.0*sign(pk1)*sgn(pk(2))/256.0;
   if abs(a1p) > 15.0/16.0-a2p
     a1p = 15.0/16.0 -a2p;
   end
  a(1) = a1p;
%
% Update b coefficients
%
 for i= 1:6
 b(i) = (255.0/256.0)*b(i)+sign(dqq)*sgn(dq(i))/128.0; % see Program 11.17 for sgn().
 end
 if a2p < -0.7185
 td = 1;
 else
 td = 0;
 end
 if a2p < -0.7185 & abs(dq(6)) > 24.0*2^(yl)
 tr = 1;
 else
 tr = 0;
 end
 for i=6:-1:2
 dq(i) = dq(i-1);
 end
 dq(1) = dqq; pk(2) = pk(1); pk(1) = pk1; sr(2) = sr(1); sr(1) = srr;
%
```

```
% Adaptive speed control
%
 dms = (31.0/32.0)*dms; dms = dms + fitable(abs(ik)+1)/32.0;
 dml = (127.0/128.0)*dml; dml = dml + fitable(abs(ik)+1)/128.0;
 if ap > 1.0
 al = 1.0;
 else
 al = ap;
 end
 ap = (15.0/16.0)*ap;
 if abs(dms-dml) >= dml/8.0
  ap = ap + 1/8.0;
 end
 if y < 3
  ap = ap +1/8.0;
 end
 if td == 1
  ap = ap + 1/8.0;
 end
 if tr == 1
  ap = 1.0;
 end
 y = al*yu + (1.0-al)*yl;
  end
end
iiout = ii;v
```

Program 11.13. MATLAB function for ADPCM decoding.

```
function iiout = adpcmdec(ii)
% This function performs ADPCM decoding.
% function iiout = adpcmdec(ii)
% Usage:
% ii = input ADPCM index
% iiout = decoded output value
%
% Quantization tables:
fitable = [0 0 0 1 1 1 1 3 7];
witable = [-0.75 1.13 2.56 4.00 7.00 12.38 22.19 70.13 ];
qtable = [ -0.98 0.62 1.38 1.91 2.34 2.72 3.12 ];
invqtable = [0.031 1.05 1.66 2.13 2.52 2.91 3.32 ];

lgth = length(ii);
sr = zeros(1,2); pk = zeros(1,2);
a = zeros(1,2); b = zeros(1,6);
dq = zeros(1,6); out= zeros(1,lgth);
y=0; ap = 0; al = 0; yu=0; yl = 0; dms = 0; dml = 0; tr = 0; td = 0;
for k = 1:lgth
%
 sez = b(1)*dq(1);
```

```
for i=2:6
 sez = sez + b(i)*dq(i);
end
se = a(1)*sr(1)+a(2)*sr(2)+ sez;
%
% Inverse quantization
%
 ik = ii(k);
 yu = (31.0/32.0)*y + witable(abs(ik)+1)/32.0;
 if yu > 10.0
 yu = 10.0;
 end
 if yu < 1.06
 yu = 1.06;
 end
 yl = (63.0/64.0)*yl+yu/64.0;
 if ik == 0
  dqq = 2^(-y);
 else
  dqq = 2^(invqtable(abs(ik))+y);
 end
 if ik < 0
  dqq = -dqq;
 end
 srr = se + dqq;
 dqsez = srr+sez-se;
 out(k) =srr;
%
% Update state
%
pk1 = dqsez;
%
% Obtain adaptive predictor coefficients
%
  if tr == 1
    a = zeros(1,2);
    b = zeros(1,6);
    tr = 0;
    td = 0; % Set for the time being
  else
% Update predictor poles
% Update a2 first
  a2p = (127.0/128.0)*a(2);
  if abs(a(1)) <= 0.5
    fa1 = 4.0*a(1);
  else
    fa1 = 2.0*sgn(a(1));
  end
```

```
   a2p=a2p+(sign(pk1)*sgn(pk(1))-fa1*sign(pk1)*sgn(pk(2)))/128.0;
  if abs(a2p) > 0.75
   a2p = 0.75*sgn(a2p);
  end
   a(2) = a2p;
%
% Update a1
   a1p = (255.0/256.0)*a(1);
     a1p = a1p + 3.0*sign(pk1)*sgn(pk(2))/256.0;
   if abs(a1p) > 15.0/16.0-a2p
     a1p = 15.0/16.0-a2p;
   end
  a(1) = a1p;
%
% Update b coefficients
%
 for i= 1: 6
  b(i) = (255.0/256.0)*b(i)+sign(dqq)*sgn(dq(i))/128.0;
 end
 if a2p < -0.7185
  td = 1;
 else
  td = 0;
 end
 if a2p < -0.7185 & abs(dq(6)) > 24.0*2^(yl)
  tr = 1;
 else
  tr = 0;
 end
 for i=6:-1:2
  dq(i) = dq(i-1);
 end
 dq(1) = dqq; pk(2) = pk(1); pk(1) = pk1; sr(2) = sr(1); sr(1) = srr;
%
% Adaptive speed control
%
 dms = (31.0/32.0)*dms;
 dms = dms + fitable(abs(ik)+1)/32.0;
 dml = (127.0/128.0)*dml;
 dml = dml + fitable(abs(ik)+1)/128.0;
 if ap > 1.0
 al = 1.0;
 else
  al = ap;
 end
 ap = (15.0/16.0)*ap;
 if abs(dms-dml) >= dml/8.0
  ap = ap + 1/8.0;
```

```
 end
 if y < 3
  ap = ap +1/8.0;
 end
 if td == 1
  ap = ap + 1/8.0;
 end
 if tr == 1
  ap = 1.0;
 end
 y = al*yu + (1.0-al)*yl;
 end
end
iiout = out;
```

Program 11.14. W-MDCT function.

```
function [ tdac_coef ] = wmdct(ipsig)
 %
 % This function transforms the signal vector using the W-MDCT.
 % Usage:
 % ipsig: input signal block of N samples (N=even number)
 % tdac_coe: W-MDCT coefficents (N/2 coefficients)
 %
 N = length(ipsig);
 NN =N;
 for i=1:NN
  h(i) = sin((pi/NN)*(i-1+0.5));
 end
 for k=1:N/2
  tdac_coef(k) = 0.0;
  for n=1:N
  tdac_coef(k) = tdac_coef(k) + ...
  h(n)*ipsig(n)*cos((2*pi/N)*(k-1+0.5)*(n-1+0.5+N/4));
  end
 end
 tdac_coef=2*tdac_coef;
```

Program 11.15. Inverse W-IMDCT function.

```
function [ opsig ] = wimdct(tdac_coef)
 %
 % This function transforms the W-MDCT coefficients back to the signal.
 % Usage:
 % tdac_coeff: N/2 W-MDCT coeffcients
 % opsig: output signal black with N samples
 %
 N = length(tdac_coef);
 tmp_coef = ((-1)^(N+1))*tdac_coef(N:-1:1);
 tdac_coef = [ tdac_coef tmp_coef];
```

```
N = length(tdac_coef);
NN =N;
for i=1:NN
f(i) = sin((pi/NN)*(i-1+0.5));
end
for n=1:N
 opsig(n) = 0.0;
 for k=1:N
  opsig(n) = opsig(n) + ...
  tdac_coef(k)*cos((2*pi/N)*(k-1+0.5)*(n-1+0.5+N/4));
  end
 opsig(n) = opsig(n)*f(n)/N;
end
```

Program 11.16. Waveform coding using DCT and W-MDCT.

```
% Waveform coding using DCT and MDCT for a block size of 16 samples.
% Main program
close all; clear all
load we.dat        % Provided by the instructor
% Create a simple 3-bit scale fcator
scalef4bits=[1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768];
scalef3bits=[256 512 1024 2048 4096 8192 16384 32768];
scalef2bits=[4096 8192 16384 32768];
scalef1bit=[16384 32768];
scalef=scalef1bit;
nbits =3;
% Ensure the block size to be 16 samples.
x=[we zeros(1,16-mod(length(we),16))];
Nblock=length(x)/16;
DCT_code=[]; scale_code=[];
% DCT transform coding
% Encoder
 for i=1:Nblock
 xblock_DCT=dct(x((i-1)*16+1:i*16));
 diff=abs(scalef-(max(abs(xblock_DCT))));
 iscale(i)=min(find(diff<=min(diff))); % find a scale factor
 xblock_DCT=xblock_DCT/scalef(iscale(i)); % scale the input vector
  for j=1:16
  [DCT_coeff(j) pp]=biquant(nbits,-1,1,xblock_DCT(j));
  end
  DCT_code=[DCT_code DCT_coeff ];
end
% Decoder
Nblock=length(DCT_code)/16;
xx=[];
for i=1:Nblock
  DCT_coefR=DCT_code((i-1)*16+1:i*16);
 for j=1:16
```

```
   xrblock_DCT(j)=biqtdec(nbits,-1,1,DCT_coefR(j));
 end
   xrblock=idct(xrblock_DCT.*scalef(iscale(i)));
 xx=[xx xrblock];
end
% Transform coding using MDCT
xm=[zeros(1,8) we zeros(1,8-mod(length(we),8)), zeros(1,8)];
Nsubblock=length(x)/8;
MDCT_code=[];
% Encoder
 for i=1:Nsubblock
 xsubblock_DCT=wmdct(xm((i-1)*8+1:(i+1)*8));
 diff=abs(scalef-max(abs(xsubblock_DCT)));
 iscale(i)=min(find(diff<=min(diff))); % find a scale factor
 xsubblock_DCT=xsubblock_DCT/scalef(iscale(i)); % scale the input vector
 for j=1:8
  [MDCT_coeff(j) pp]=biquant(nbits,-1,1,xsubblock_DCT(j));
  end
  MDCT_code=[MDCT_code MDCT_coeff];
end
% Decoder
% Recover the first subblock
Nsubblock=length(MDCT_code)/8;
xxm=[];
MDCT_coeffR=MDCT_code(1:8);
for j=1:8
   xmrblock_DCT(j)=biqtdec(nbits,-1,1,MDCT_coeffR(j));
end
xmrblock=wimdct(xmrblock_DCT*scalef(iscale(1)));
xxr_pre=xmrblock(9:16) % recovered first block for overlap and add
for i=2:Nsubblock
 MDCT_coeffR=MDCT_code((i-1)*8+1:i*8);
 for j=1:8
   xmrblock_DCT(j)=biqtdec(nbits,-1,1,MDCT_coeffR(j));
 end
   xmrblock=wimdct(xmrblock_DCT*scalef(iscale(i)));
 xxr_cur=xxr_pre+xmrblock(1:8); % overlap and add
 xxm=[xxm xxr_cur];
 xxr_pre=xmrblock(9:16);     % set for the next overlap
end

subplot(3,1,1);plot(x,'k');grid; axis([0 length(x) -10000 10000])
ylabel('Original signal');
subplot(3,1,2);plot(xx,'k');grid;axis([0 length(xx) -10000 10000]);
ylabel('DCT coding')
subplot(3,1,3);plot(xxm,'k');grid;axis([0 length(xxm) -10000 10000]);
ylabel('W-MDCT coding');
xlabel('Sample number');
```

Program 11.17. Sign function.

```
function sgn = sgn(sgninp)
%
% Sign function
% if signp >=0 then sign=1
% else sign =-1
%
 if sgninp >= 0
  opt = 1;
 else
  opt = -1;
 end
 sgn = opt;
```

11.8 PROBLEMS

11.1. For the 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from −2.5 to 2.5 volts, determine

a. the quantization step size;

b. the binary codes, recovered voltages, and quantization errors when each input is 1.6 volts and −0.2 volt.

11.2. For the 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from −4 to 4 volts, determine

a. the quantization step size;

b. the binary codes, recovered voltages, and quantization errors when each input is −2.6 volts and 0.1 volt.

11.3. For the 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from −5 to 5 volts, determine

a. the quantization step size;

b. the binary codes, recovered voltages, and quantization errors when each input is −2.6 volts and 3.5 volts.

11.4. For the 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from −10 to 10 volts, determine

a. the quantization step size;

b. the binary codes, recovered voltages, and quantization errors when each input is −5 volts, 0 volts, and 7.2 volts.

11.5. For the μ-law compression and expanding process shown in Figure 11.3 with $\mu = 255$, a 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from −2.5 to 2.5 volts, determine the binary codes, recovered voltages, and quantization errors when each input is 1.6 volts and −0.2 volt.

11.6. For the μ-law compression and expanding process shown in Figure 11.3 with $\mu = 255$, a 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from -4 to 4 volts, determine the binary codes, recovered voltages, and quantization errors when each input is -2.6 volts and 0.1 volt.

11.7. For the μ-law compression and expanding process shown in Figure 11.3 with $\mu = 255$, a 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from -5 to 5 volts, determine the binary codes, recovered voltages, and quantization errors when each input is -2.6 volts and 3.5 volts.

11.8. For the μ-law compression and expanding process shown in Figure 11.3 with $\mu = 255$, a 3-bit midtread quantizer described in Figure 11.1, and an analog signal range from -10 to 10 volts, determine the binary codes, recovered voltages, and quantization errors when each input is -5, 0, and 7.2 volts.

11.9. In a digital companding system, encode each of the following 12-bit linear PCM codes into 8-bit compressed PCM code:

a. 0 0 0 0 0 0 0 1 0 1 0 1

b. 1 0 1 0 1 1 1 0 1 0 1 0

11.10. In a digital companding system**,** decode each of the following 8-bit compressed PCM codes into 12-bit linear PCM code:

a. 0 0 0 0 0 1 1 1

b. 1 1 1 0 1 0 0 1

11.11. In a digital companding system, encode each of the following 12-linear PCM codes into the 8-bit compressed PCM code:

a. 0 0 1 0 1 0 1 0 1 0 1 0

b. 1 0 0 0 0 0 0 0 1 1 0 1

11.12. In a digital companding system**,** decode each of the following 8-bit compressed PCM codes into the 12-bit linear PCM code:

a. 0 0 1 0 1 1 0 1

b. 1 0 0 0 0 1 0 1

11.13. Consider a 3-bit DPCM encoding system with the following specifications (Figure 11.19):

$$\begin{aligned}\text{Encoder scheme}: \ \tilde{x}(n) &= \hat{x}(n-1) \ \text{(predictor)}\\ d(n) &= x(n) - \tilde{x}(n)\\ d_q(n) &= Q[d(n)] = \text{quantizer in Table 11.9}\\ \hat{x}(n) &= \tilde{x}(n) + d_q(n)\end{aligned}$$

5-bit input data: $x(0) = -6$, $x(1) = -8$, and $x(2) = -13$

Perform DPCM encoding to produce the binary code for each input data.

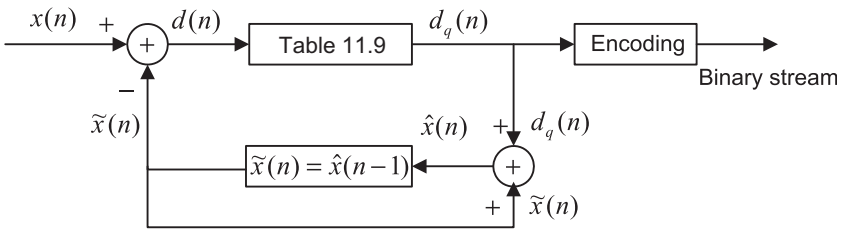

FIGURE 11.19

DPCM encoding in Problem 11.13.

Table 11.9 Quantization Table for the 3-bit Quantizer in Problem 11.13

Binary Code	Quantization Value $d_q(n)$	Subrange in $d(n)$
0 1 1	−11	$-15 \le d(n) < -7$
0 1 0	−5	$-7 \le d(n) < -3$
0 0 1	−2	$-3 \le d(n) < -1$
0 0 0	0	$-1 \le d(n) < 0$
1 0 0	0	$0 \le d(n) \le 1$
1 0 1	2	$1 < d(n) \le 3$
1 1 0	5	$3 < d(n) \le 7$
1 1 1	11	$7 < d(n) \le 15$

11.14. Consider a 3-bit DPCM decoding system with the following specifications (Figure 11.20):

$$\text{Decoding scheme}: \ \tilde{x}(n) = \hat{x}(n-1) \ \text{(predictor)}$$
$$d_q(n) = \text{quantizer in Table 11.9}$$
$$\hat{x}(n) = \tilde{x}(n) + d_q(n)$$

Three received binary codes: 110, 100, 101

Perform DCPM decoding to recover each digital value using its binary code.

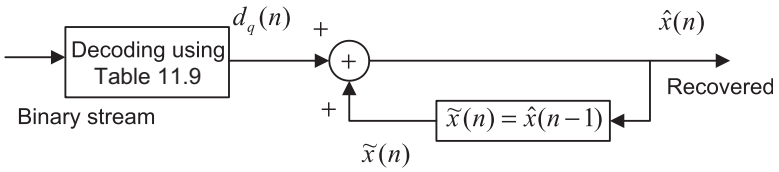

FIGURE 11.20

DPCM decoding in Problem 11.14.

11.15. For a 3-bit DPCM encoding system shown in Problem 11.13 and the given 5-bit input data $x(0) = 6, x(1) = 8$, and $x(2) = 13$, perform DPCM encoding to produce the binary code for each of input data.

11.16. For the 3-bit DPCM decoding system shown in Problem 11.14 and the received data 010, 000, 001, perform DCPM decoding to recover each digital value using its binary code.

11.17. Assuming that a speech waveform is sampled at 8 kHz and each sample is encoded by 16 bits, determine the compression ratio for each of the following encoding methods:

a. no compression

b. standard μ-law compression (8 bits per sample)

c. standard ADPCM encoding (4 bits per sample)

11.18. Assuming that a speech waveform is sampled at 8 kHz and each sample is encoded by 16 bits, determine the bit rate for each of the following encoding methods:

a. no compression

b. standard μ-law companding (8 bits per sample)

c. standard ADPCM encoding (4 bits per sample)

11.19. Assuming that an audio waveform is sampled at 44.1 kHz and each sample is encoded by 16 bits, determine the compression ratio for each of the following encoding methods:

a. no compression

b. standard μ-law compression (8 bits per sample)

c. standard ADPCM encoding (4 bits per sample)

11.20. Assuming that an audio waveform is sampled at 44.1 kHz and each sample is encoded by 12 bits, determine the bit rate for each of the encoding methods.

a. no compression

b. standard μ-law companding (8 bits per sample)

c. standard ADPCM encoding (4 bits per sample)

11.21. Speech is sampled at 8 kHz and each sample is encoded by 16 bits. A telephone system can transport the digital voice channel over a digital link with a capacity of 1.536 MBPS. Determine the number of channels that the phone company can carry for each of the following encoding methods:

a. no compression

b. standard 8-bit μ-law companding (8 bits per sample)

c. standard ADPCM encoding (4 bits per sample)

11.22. Given the input data

$$x(0) = 25,\ x(1) = 30,\ x(2) = 28, \quad \text{and} \quad x(3) = 25$$

determine the DCT coefficients.

11.23. Given the input data

$$x(0) = 25 \quad \text{and} \quad x(1) = 30$$

determine the DCT coefficients.

11.24. Given the input data

$$x(0) = 25,\ x(1) = 30,\ x(2) = 28,\ x(3) = 25,$$
$$x(4) = 10,\ x(5) = 0,\ x(6) = 0,\ \text{and}\ x(7) = 0$$

determine the DCT coefficients $X_{DCT}(0)$, $X_{DCT}(2)$, $X_{DCT}(4)$, and $X_{DCT}(6)$.

11.25. Given the input data

$$x(0) = 25,\ x(1) = 30,\ x(2) = 28,\ x(3) = 25,$$
$$x(4) = 10,\ x(5) = 0,\ x(6) = 0,\ \text{and}\ x(7) = 0$$

determine the DCT coefficients $X_{DCT}(1)$, $X_{DCT}(3)$, $X_{DCT}(5)$, and $X_{DCT}(7)$.

11.26. Assume the following DCT coefficients with infinite precision:

$$X_{DCT}(0) = 14,\ X_{DCT}(1) = 6,\ X_{DCT}(2) = -6,\ \text{and}\ X_{DCT}(3) = 8$$

a. Determine the input data using the MATLAB function **idct().**

b. Recover the input data samples using the MATLAB function **idct()** if a bit allocation scheme quantizes the DCT coefficients as follows: 2 magnitude bits plus 1 sign bit (3 bits) for the DC coefficient, 1 magnitude bit plus 1 sign bit (2 bits) for each AC coefficient and a scale factor of 8, that is,

$$X_{DCT}(0) = 8 \times 2 = 16,\ X_{DCT}(1) = 8 \times 1 = 8,\ X_{DCT}(2) = 8 \times (-1)$$
$$= -8,\ \text{and}\ X_{DCT}(3) = 8 \times 1 = 8$$

c. Compute the quantized error in part (b) of this problem.

11.27. Assume the following DCT coefficients with infinite precision:

$$X_{DCT}(0) = 11,\ X_{DCT}(1) = 5,\ X_{DCT}(2) = 7,\ \text{and}\ X_{DCT}(3) = -3$$

a. Determine the input data using the MATLAB function **idct().**

b. Recover the input data samples using the MATLAB function **idct()** if a bit allocation scheme quantizes the DCT coefficients as follows: 2 magnitude bits plus 1 sign bit (3 bits) for the DC coefficient, 1 magnitude bit plus 1 sign bit (2 bits) for each AC coefficient and a scale factor of 8, that is,

$$X_{DCT}(0) = 8 \times 1 = 8,\ X_{DCT}(1) = 8 \times 1 = 8,\ X_{DCT}(2) = 8 \times 1 = -8, \quad \text{and}$$
$$X_{DCT}(3) = 8 \times 0 = 0$$

c. Compute the quantized error in part (b) of this problem.

11.28. **a.** Verify that the window function

$$f(n) = h(n) = \sin\left(\frac{\pi}{N}(n+0.5)\right)$$

used in MDCT is satisfied with Equations (11.35) and (11.36).

b. Verify the relation for W-MDCT coefficients

$$X_{MDCT}(k) = (-1)^{\frac{N}{2}+1} X_{MDCT}(N-1-k) \quad \text{for} \quad k = N/2, N/2+1, \cdots, N-1$$

11.29. Given data 1, 2, 3, 4, 5, 4, 3, 2, …,

a. determine the W-MDCT coefficients for the first three blocks using a block size of 4;

b. determine the first two overlapped subblocks and compare the results with the original data sequence using the W-MDCT coefficients in part (a).

11.30. Given data 1, 2, 3, 4, 5, 4, 3, 2, 1, 2, 3, 4, 5,…,

a. determine the W-MDCT coefficients for the first three blocks using a block size of 6;

b. determine the first two overlapped subblocks and compare the results with the original data sequence using the W-MDCT coefficients in part (a).

11.8.1 Computer Problems with MATLAB

Use the MATLAB programs in Section 11.7 for Problems 11.31 to 11.33.

11.31. Consider the data file "speech.dat" with 16 bits per sample and a sampling rate of 8 kHz.

a. Use PCM coding (midtread quantizer) to perform compression and decompression and apply the MATLAB function **sound()** to evaluate the sound quality in terms of "excellent", "good", "intelligent", and "unacceptable" for the following bit rates:

1. 4 bits/sample (32 kbits per second)
2. 6 bits/sample (48 kbits per second)
3. 8 bits/sample (64 kbits per second)

b. Use μ-law PCM coding to perform compression and decompression and apply the MATLAB function **sound()** to evaluate the sound quality for the following bit rates:

1. 4 bits/sample (32 kbits per second)
2. 6 bits/sample (48 kbits per second)
3. 8 bits/sample (64 kbits per second)

11.32. Given the data file "speech.dat" with 16 bits per sample, a sampling rate of 8 kHz, and ADCPM coding, perform compression and decompression and apply the MATLAB function **sound()** to evaluate the sound quality.

11.33. Given the data file "speech.dat" with 16 bits per sample, a sampling rate of 8 kHz, and DCT and M-DCT coding as described in the programs in Section 11.7, perform compression and

decompression using the following specified parameters in Program 11.16 to compare the sound quality:

a. nbits=3, scalef=scalef2bits

b. nbits=3, scalef=scalef3bits

c. nbits=4, scalef=scalef2bits

d. nbits=4, scalef=scalef3bits