BRITISH LIBRARY
DOCUMENT SUPPLY CENTRE
26 OCT 1998
98/.28268

The Scientist and Engineer's Guide to

Digital Signal Processing

by
Steven W. Smith

California Technical Publishing
San Diego, California

The Scientist and Engineer's Guide to
Digital Signal Processing

by
Steven W. Smith

copyright © 1997 by California Technical Publishing

All rights reserved. No portion of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, without written permission of the publisher.

ISBN 0-9660176-3-3
LCCN 97-80293

California Technical Publishing
P.O. Box 502407
San Diego, CA 92150-2407

To contact the author or publisher through the internet:
website: **DSPguide.com**
e-mail: **Smith@DSPguide.com**

Printed in the United States of America
First Edition
First printing 1997

Important Legal Information: Warning and Disclaimer

This book presents the fundamentals of Digital Signal Processing using examples from common science and engineering problems. While the author believes that the concepts and data contained in this book are accurate and correct, they should not be used in any application without proper verification by the person making the application. Extensive and detailed testing is essential where incorrect functioning could result in personal injury or damage to property. The material in this book is intended solely as a teaching aid, and is not represented to be an appropriate or safe solution to any particular problem. For this reason, the author and publisher make no warranties, express or implied, that the concepts, examples, data, algorithms, techniques, or programs contained in this book are free from error, conform to any industry standard, or are suitable for any application. The author and publisher disclaim all liability and responsibility to any person or entity with respect to any loss or damage caused, or alleged to be caused, directly or indirectly, by the information contained in this book. If you do not wish to be bound by the above, you may return this book to the publisher for a full refund.

Contents at a Glance

FOUNDATIONS

Chapter 1. The Breadth and Depth of DSP 1
Chapter 2. Statistics, Probability and Noise 11
Chapter 3. ADC and DAC 35
Chapter 4. DSP Software 67

FUNDAMENTALS

Chapter 5. Linear Systems 87
Chapter 6. Convolution 107
Chapter 7. Properties of Convolution 123
Chapter 8. The Discrete Fourier Transform 141
Chapter 9. Applications of the DFT 169
Chapter 10. Fourier Transform Properties 185
Chapter 11. Fourier Transform Pairs 209
Chapter 12. The Fast Fourier Transform 225
Chapter 13. Continuous Signal Processing 243

DIGITAL FILTERS

Chapter 14. Introduction to Digital Filters 261
Chapter 15. Moving Average Filters 277
Chapter 16. Windowed-Sinc Filters 285
Chapter 17. Custom Filters 297
Chapter 18. FFT Convolution 311
Chapter 19. Recursive Filters 319
Chapter 20. Chebyshev Filters 333
Chapter 21. Filter Comparison 343

APPLICATIONS

Chapter 22. Audio Processing 351
Chapter 23. Image Formation and Display 373
Chapter 24. Linear Image Processing 397
Chapter 25. Special Imaging Techniques 423
Chapter 26. Neural Networks (and more!) 451
Chapter 27. Data Compression 481

COMPLEX TECHNIQUES

Chapter 28. Complex Numbers 503
Chapter 29. The Complex Fourier Transform 519
Chapter 30. The Laplace Transform 533
Chapter 31. The z-Transform 557

Study Guide 583
Glossary 607
Index 619

Nintendo Exhibit 1021
Page 003

Table of Contents

FOUNDATIONS

Chapter 1. The Breadth and Depth of DSP 1

The Roots of DSP 1
Telecommunications 4
Audio Processing 5
Echo Location 7
Imaging Processing 9

Chapter 2. Statistics, Probability and Noise 11

Signal and Graph Terminology 11
Mean and Standard Deviation 13
Signal vs. Underlying Process 17
The Histogram, Pmf and Pdf 19
The Normal Distribution 26
Digital Noise Generation 29
Precision and Accuracy 32

Chapter 3. ADC and DAC 35

Quantization 35
The Sampling Theorem 39
Digital-to-Analog Conversion 44
Analog Filters for Data Conversion 48
Selecting the Antialias Filter 55
Multirate Data Conversion 58
Single Bit Data Conversion 60

Chapter 4. DSP Software 67

Computer Numbers 67
Fixed Point (Integers) 68
Floating Point (Real Numbers) 70
Number Precision 72
Execution Speed: Program Language 76
Execution Speed: Hardware 80
Execution Speed: Programming Tips 84

Nintendo Exhibit 1021
Page 004

FUNDAMENTALS

Chapter 5. Linear Systems 87
Signals and Systems 87
Requirements for Linearity 89
Static Linearity and Sinusoidal Fidelity 92
Examples of Linear and Nonlinear Systems 94
Special Properties of Linearity 96
Superposition: the Foundation of DSP 98
Common Decompositions 100
Alternatives to Linearity 104

Chapter 6. Convolution 107
The Delta Function and Impulse Response 107
Convolution 108
The Input Side Algorithm 112
The Output Side Algorithm 116
The Sum of Weighted Inputs 122

Chapter 7. Properties of Convolution 123
Common Impulse Responses 123
Mathematical Properties 132
Correlation 136
Speed 140

Chapter 8. The Discrete Fourier Transform 141
The Family of Fourier Transforms 141
Notation and Format of the real DFT 146
The Frequency Domain's Independent Variable 148
DFT Basis Functions 150
Synthesis, Calculating the Inverse DFT 152
Analysis, Calculating the DFT 156
Duality 160
Polar Notation 160
Polar Nuisances 163

Chapter 9. Applications of the DFT 169
Spectral Analysis of Signals 169
Frequency Response of Systems 176
Convolution via the Frequency Domain 179

Chapter 10. Fourier Transform Properties 185
Linearity of the Fourier Transform 185
Characteristics of the Phase 188

Nintendo Exhibit 1021
Page 005

Periodic Nature of the DFT 194
Compression and Expansion, Multirate methods 200
Multiplying Signals (Amplitude Modulation) 204
The Discrete Time Fourier Transform 206
Parseval's Relation 208

Chapter 11. Fourier Transform Pairs 209
Delta Function Pairs 209
The Sinc Function 212
Other Transform Pairs 215
Gibbs Effect 218
Harmonics 220
Chirp Signals 222

Chapter 12. The Fast Fourier Transform 225
Real DFT Using the Complex DFT 225
How the FFT Works 228
FFT Programs 233
Speed and Precision Comparisons 237
Further Speed Increases 238

Chapter 13. Continuous Signal Processing 243
The Delta Function 243
Convolution 246
The Fourier Transform 252
The Fourier Series 255

DIGITAL FILTERS

Chapter 14. Introduction to Digital Filters 261
Filter Basics 261
How Information is Represented in Signals 265
Time Domain Parameters 266
Frequency Domain Parameters 268
High-Pass, Band-Pass and Band-Reject Filters 271
Filter Classification 274

Chapter 15. Moving Average Filters 277
Implementation by Convolution 277
Noise Reduction vs. Step Response 278
Frequency Response 280
Relatives of the Moving Average Filter 280
Recursive Implementation 282

Nintendo Exhibit 1021
Page 006

Chapter 16. Windowed-Sinc Filters 285
Strategy of the Windowed-Sinc 285
Designing the Filter 288
Examples of Windowed-Sinc Filters 292
Pushing it to the Limit 293

Chapter 17. Custom Filters 297
Arbitrary Frequency Response 297
Deconvolution 300
Optimal Filters 307

Chapter 18. FFT Convolution 311
The Overlap-Add Method 311
FFT Convolution 312
Speed Improvements 316

Chapter 19. Recursive Filters 319
The Recursive Method 319
Single Pole Recursive Filters 322
Narrow-band Filters 326
Phase Response 328
Using Integers 332

Chapter 20. Chebyshev Filters 333
The Chebyshev and Butterworth Responses 333
Designing the Filter 334
Step Response Overshoot 338
Stability 339

Chapter 21. Filter Comparison 343
Match #1: Analog vs. Digital Filters 343
Match #2: Windowed-Sinc vs. Chebyshev 346
Match #3: Moving Average vs. Single Pole 348

APPLICATIONS

Chapter 22. Audio Processing 351
Human Hearing 351
Timbre 355
Sound Quality vs. Data Rate 358
High Fidelity Audio 359
Companding 362

Nintendo Exhibit 1021
Page 007

Speech Synthesis and Recognition 364
Nonlinear Audio Processing 368

Chapter 23. Image Formation and Display 373
Digital Image Structure 373
Cameras and Eyes 376
Television Video Signals 384
Other Image Acquisition and Display 386
Brightness and Contrast Adjustments 387
Grayscale Transforms 390
Warping 394

Chapter 24. Linear Image Processing 397
Convolution 397
3×3 Edge Modification 402
Convolution by Separability 404
Example of a Large PSF: Illumination Flattening 407
Fourier Image Analysis 410
FFT Convolution 416
A Closer Look at Image Convolution 418

Chapter 25. Special Imaging Techniques 423
Spatial Resolution 423
Sample Spacing and Sampling Aperture 430
Signal-to-Noise Ratio 432
Morphological Image Processing 436
Computed Tomography 442

Chapter 26. Neural Networks (and more!) 451
Target Detection 451
Neural Network Architecture 458
Why Does it Work? 463
Training the Neural Network 465
Evaluating the Results 473
Recursive Filter Design 476

Chapter 27. Data Compression 481
Data Compression Strategies 481
Run-Length Encoding 483
Huffman Encoding 484
Delta Encoding 486
LZW Compression 488
JPEG (Transform Compression) 494
MPEG 501

Nintendo Exhibit 1021
Page 008

COMPLEX TECHNIQUES

Chapter 28. Complex Numbers 503
The Complex Number System 503
Polar Notation 507
Using Complex Numbers by Substitution 509
Complex Representation of Sinusoids 511
Complex Representation of Systems 513
Electrical Circuit Analysis 515

Chapter 29. The Complex Fourier Transform 519
The Real DFT 517
Mathematical Equivalence 521
The Complex DFT 523
The Family of Fourier Transforms 528
Why the Complex Fourier Transform is Used 532

Chapter 30. The Laplace Transform 533
The Nature of the s-Domain 533
Strategy of the Laplace Transform 540
Analysis of Electric Circuits 544
The Importance of Poles and Zeros 549
Filter Design in the s-Domain 552

Chapter 31. The z-Transform 557
The Nature of the z-Domain 557
Analysis of Recursive Systems 562
Cascade and Parallel Stages 568
Spectral Inversion 571
Gain Changes 573
Chebyshev-Butterworth Filter Design 575
The Best and Worst of DSP 582

Study Guide 583

Glossary 607

Index 619

Visit our website at **www.DSPguide.com** for free software, homework problems, suggested reading, biography of the author, and other up-to-date DSP information.

Nintendo Exhibit 1021
Page 009

CHAPTER
14

Introduction to Digital Filters

Digital filters are used for two general purposes: (1) separation of signals that have been combined, and (2) restoration of signals that have been distorted in some way. Analog (electronic) filters can be used for these same tasks; however, digital filters can achieve far superior results. The most popular digital filters are described and compared in the next seven chapters. This introductory chapter describes the parameters you want to look for when learning about each of these filters.

Filter Basics

Digital filters are a very important part of DSP. In fact, their extraordinary performance is one of the key reasons that DSP has become so popular. As mentioned in the introduction, filters have two uses: signal *separation* and signal *restoration*. Signal separation is needed when a signal has been contaminated with interference, noise, or other signals. For example, imagine a device for measuring the electrical activity of a baby's heart (EKG) while still in the womb. The raw signal will likely be corrupted by the breathing and heartbeat of the mother. A filter might be used to separate these signals so that they can be individually analyzed.

Signal restoration is used when a signal has been distorted in some way. For example, an audio recording made with poor equipment may be filtered to better represent the sound as it actually occurred. Another example is the deblurring of an image acquired with an improperly focused lens, or a shaky camera.

These problems can be attacked with either analog or digital filters. Which is better? Analog filters are cheap, fast, and have a large dynamic range in both amplitude and frequency. Digital filters, in comparison, are vastly superior in the level of performance that can be achieved. For example, a low-pass digital filter presented in Chapter 16 has a gain of 1 +/- 0.0002 from DC to 1000 hertz, and a gain of less than 0.0002 for frequencies above

Nintendo Exhibit 1021
Page 010

1001 hertz. The entire transition occurs within only 1 hertz. Don't expect this from an op amp circuit! Digital filters can achieve *thousands* of times better performance than analog filters. This makes a dramatic difference in how filtering problems are approached. With analog filters, the emphasis is on handling limitations of the electronics, such as the accuracy and stability of the resistors and capacitors. In comparison, digital filters are so good that the performance of the filter is frequently ignored. The emphasis shifts to the limitations of the *signals*, and the *theoretical* issues regarding their processing.

It is common in DSP to say that a filter's input and output signals are in the *time domain*. This is because signals are usually created by sampling at regular intervals of *time*. But this is not the only way sampling can take place. The second most common way of sampling is at equal intervals in *space*. For example, imagine taking simultaneous readings from an array of strain sensors mounted at one centimeter increments along the length of an aircraft wing. Many other domains are possible; however, time and space are by far the most common. When you see the term *time domain* in DSP, remember that it may actually refer to samples taken over time, or it may be a general reference to any domain that the samples are taken in.

As shown in Fig. 14-1, every linear filter has an **impulse response**, a **step response** and a **frequency response.** Each of these responses contains complete information about the filter, but in a different form. If one of the three is specified, the other two are fixed and can be directly calculated. All three of these representations are important, because they describe how the filter will react under different circumstances.

The most straightforward way to implement a digital filter is by *convolving* the input signal with the digital filter's *impulse response*. All possible linear filters can be made in this manner. (This should be obvious. If it isn't, you probably don't have the background to understand this section on filter design. Try reviewing the previous section on DSP fundamentals). When the *impulse response* is used in this way, filter designers give it a special name: the **filter kernel**.

There is also another way to make digital filters, called **recursion**. When a filter is implemented by convolution, each sample in the output is calculated by *weighting* the samples in the input, and adding them together. Recursive filters are an extension of this, using previously calculated values from the *output*, besides points from the *input*. Instead of using a filter kernel, recursive filters are defined by a set of **recursion coefficients**. This method will be discussed in detail in Chapter 19. For now, the important point is that all linear filters have an impulse response, even if you don't use it to implement the filter. To find the impulse response of a recursive filter, simply feed in an impulse, and see what comes out. The impulse responses of recursive filters are composed of sinusoids that exponentially decay in amplitude. In principle, this makes their impulse responses *infinitely long*. However, the amplitude eventually drops below the round-off noise of the system, and the remaining samples can be ignored. Because

Nintendo Exhibit 1021
Page 011

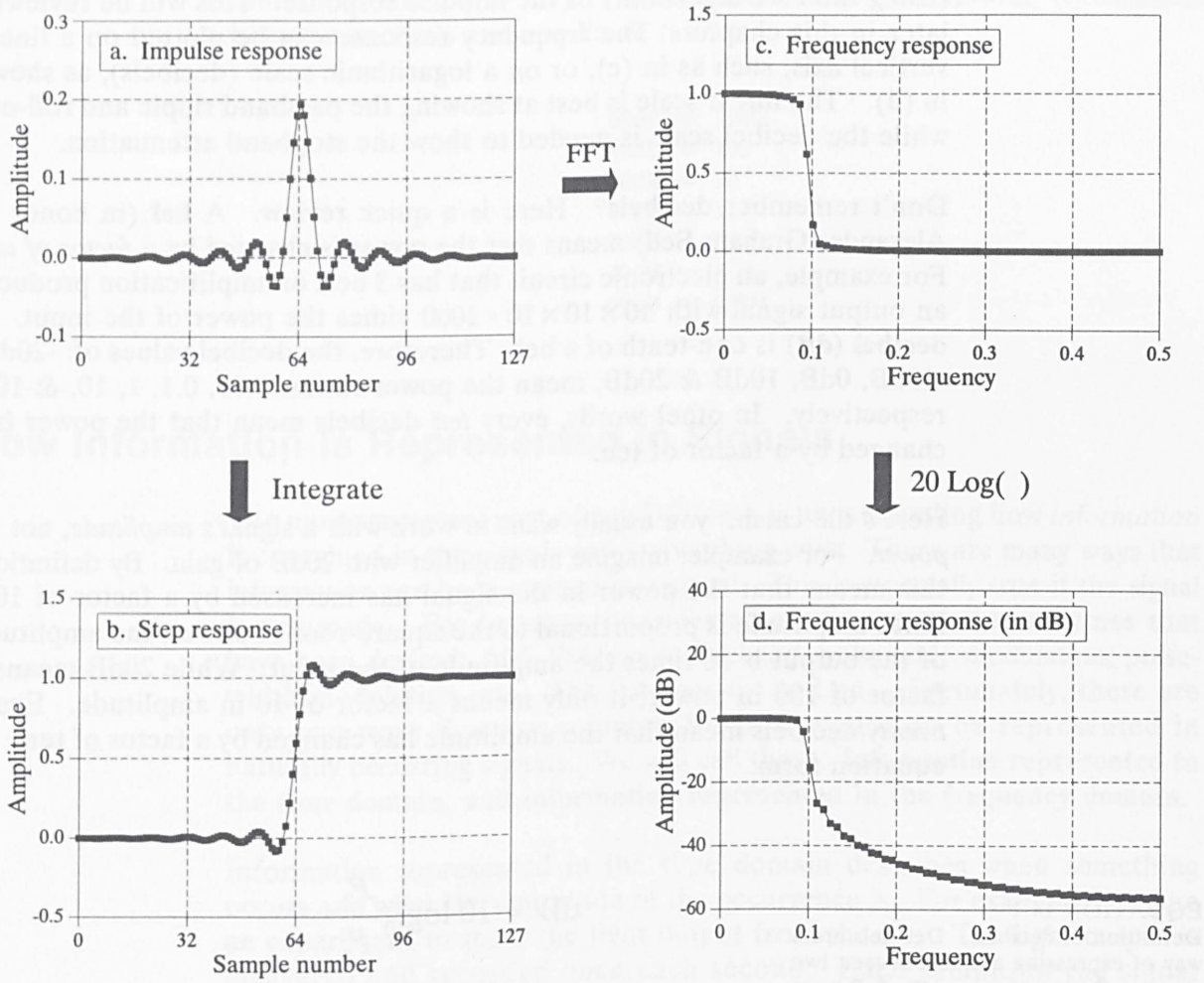


FIGURE 14-1
Filter parameters. Every linear filter has an impulse response, a step response, and a frequency response. The step response, (b), can be found by discrete integration of the impulse response, (a). The frequency response can be found from the impulse response by using the Fast Fourier Transform (FFT), and can be displayed either on a linear scale, (c), or in decibels, (d).

of this characteristic, recursive filters are also called **Infinite Impulse Response or IIR** filters. In comparison, filters carried out by convolution are called **Finite Impulse Response or FIR** filters.

As you know, the *impulse response* is the output of a system when the input is an *impulse*. In this same manner, the *step response* is the output when the input is a *step* (also called an *edge*, and an *edge response*). Since the step is the integral of the impulse, the step response is the integral of the impulse response. This provides two ways to find the step response: (1) feed a step waveform into the filter and see what comes out, or (2) integrate the impulse response. (To be mathematically correct: *integration* is used with continuous signals, while *discrete integration*, i.e., a running sum, is used with discrete signals). The frequency response can be found by taking the DFT

(using the FFT algorithm) of the impulse response. This will be reviewed later in this chapter. The frequency response can be plotted on a linear vertical axis, such as in (c), or on a logarithmic scale (decibels), as shown in (d). The linear scale is best at showing the passband ripple and roll-off, while the decibel scale is needed to show the stopband attenuation.

Don't remember decibels? Here is a quick review. A **bel** (in honor of Alexander Graham Bell) means that the power is changed by a *factor of ten*. For example, an electronic circuit that has 3 bels of amplification produces an output signal with $10 \times 10 \times 10 = 1000$ times the power of the input. A **decibel (dB)** is one-tenth of a bel. Therefore, the decibel values of: -20dB, -10dB, 0dB, 10dB & 20dB, mean the power ratios: 0.01, 0.1, 1, 10, & 100, respectively. In other words, every *ten* decibels mean that the power has changed by a factor of ten.

Here's the catch: you usually want to work with a signal's *amplitude*, not its *power*. For example, imagine an amplifier with 20dB of gain. By definition, this means that the power in the signal has increased by a factor of 100. Since amplitude is proportional to the square-root of power, the amplitude of the output is 10 times the amplitude of the input. While 20dB means a factor of 100 in power, it only means a factor of 10 in amplitude. Every *twenty* decibels mean that the amplitude has changed by a factor of ten. In equation form:

EQUATION 14-1
Definition of decibels. Decibels are a way of expressing a *ratio* between two signals. Ratios of power (P_1 & P_2) use a different equation from ratios of amplitude (A_1 & A_2).

$$\text{dB} = 10 \log_{10} \frac{P_2}{P_1}$$

$$\text{dB} = 20 \log_{10} \frac{A_2}{A_1}$$

The above equations use the base 10 logarithm; however, many computer languages only provide a function for the base *e* logarithm (the natural log, written $\log_e x$ or $\ln x$). The natural log can be use by modifying the above equations: $\text{dB} = 4.342945 \log_e (P_2/P_1)$ and $\text{dB} = 8.685890 \log_e (A_2/A_1)$.

Since decibels are a way of expressing the ratio between two signals, they are ideal for describing the gain of a system, i.e., the ratio between the output and the input signal. However, engineers also use decibels to specify the amplitude (or power) of a *single* signal, by referencing it to some standard. For example, the term: **dBV** means that the signal is being referenced to a 1 volt rms signal. Likewise, **dBm** indicates a reference signal producing 1 mW into a 600 ohms load (about 0.78 volts rms).

If you understand nothing else about decibels, remember two things: First, -3dB means that the amplitude is reduced to 0.707 (and the power is

therefore reduced to 0.5). Second, memorize the following conversions between decibels and *amplitude* ratios:

60dB = 1000
40dB = 100
20dB = 10
0dB = 1
-20dB = 0.1
-40dB = 0.01
-60dB = 0.001

How Information is Represented in Signals

The most important part of any DSP task is understanding how *information* is contained in the signals you are working with. There are many ways that information can be contained in a signal. This is especially true if the signal is manmade. For instance, consider all of the modulation schemes that have been devised: AM, FM, single-sideband, pulse-code modulation, pulse-width modulation, etc. The list goes on and on. Fortunately, there are only two ways that are common for information to be represented in naturally occurring signals. We will call these: **information represented in the time domain**, and **information represented in the frequency domain**.

Information represented in the time domain describes when something occurs and what the amplitude of the occurrence is. For example, imagine an experiment to study the light output from the sun. The light output is measured and recorded once each second. Each sample in the signal indicates what is happening at that instant, and the level of the event. If a solar flare occurs, the signal directly provides information on the time it occurred, the duration, the development over time, etc. Each sample contains information that is interpretable without reference to any other sample. Even if you have only one sample from this signal, you still know something about what you are measuring. This is the simplest way for information to be contained in a signal.

In contrast, information represented in the frequency domain is more indirect. Many things in our universe show periodic motion. For example, a wine glass struck with a fingernail will vibrate, producing a ringing sound; the pendulum of a grandfather clock swings back and forth; stars and planets rotate on their axis and revolve around each other, and so forth. By measuring the frequency, phase, and amplitude of this periodic motion, information can often be obtained about the system producing the motion. Suppose we sample the sound produced by the ringing wine glass. The fundamental frequency and harmonics of the periodic vibration relate to the mass and elasticity of the material. A single sample, in itself, contains no information about the periodic motion, and therefore no information about the wine glass. The information is contained in the *relationship* between many points in the signal.

Nintendo Exhibit 1021
Page 014

This brings us to the importance of the step and frequency responses. The *step response* describes how information represented in the *time domain* is being modified by the system. In contrast, the *frequency response* shows how information represented in the *frequency domain* is being changed. This distinction is absolutely critical in filter design because it is not possible to optimize a filter for both applications. Good performance in the time domain results in poor performance in the frequency domain, and vice versa. If you are designing a filter to remove noise from an EKG signal (information represented in the time domain), the step response is the important parameter, and the frequency response is of little concern. If your task is to design a digital filter for a hearing aid (with the information in the frequency domain), the frequency response is all important, while the step response doesn't matter. Now let's look at what makes a filter optimal for time domain or frequency domain applications.

Time Domain Parameters

It may not be obvious why the step response is of such concern in time domain filters. You may be wondering why the impulse response isn't the important parameter. The answer lies in the way that the human mind understands and processes information. Remember that the step, impulse and frequency responses all contain identical information, just in different arrangements. The step response is useful in time domain analysis because it matches the way humans view the information contained in the signals.

For example, suppose you are given a signal of some unknown origin and asked to analyze it. The first thing you will do is divide the signal into regions of similar characteristics. You can't stop from doing this; your mind will do it automatically. Some of the regions may be smooth; others may have large amplitude peaks; others may be noisy. This segmentation is accomplished by identifying the points that separate the regions. This is where the step function comes in. The step function is the purest way of representing a division between two dissimilar regions. It can mark when an event starts, or when an event ends. It tells you that whatever is on the *left* is somehow different from whatever is on the *right*. This is how the human mind views time domain information: a group of step functions dividing the information into regions of similar characteristics. The step response, in turn, is important because it describes how the dividing lines are being modified by the filter.

The step response parameters that are important in filter design are shown in Fig. 14-2. To distinguish events in a signal, the duration of the step response must be shorter than the spacing of the events. This dictates that the step response should be as *fast* (the DSP jargon) as possible. This is shown in Figs. (a) & (b). The most common way to specify the **risetime** (more jargon) is to quote the number of samples between the 10% and 90% amplitude levels. Why isn't a very fast risetime always possible? There are many reasons, noise reduction, inherent limitations of the data acquisition system, avoiding aliasing, etc.

Nintendo Exhibit 1021
Page 015

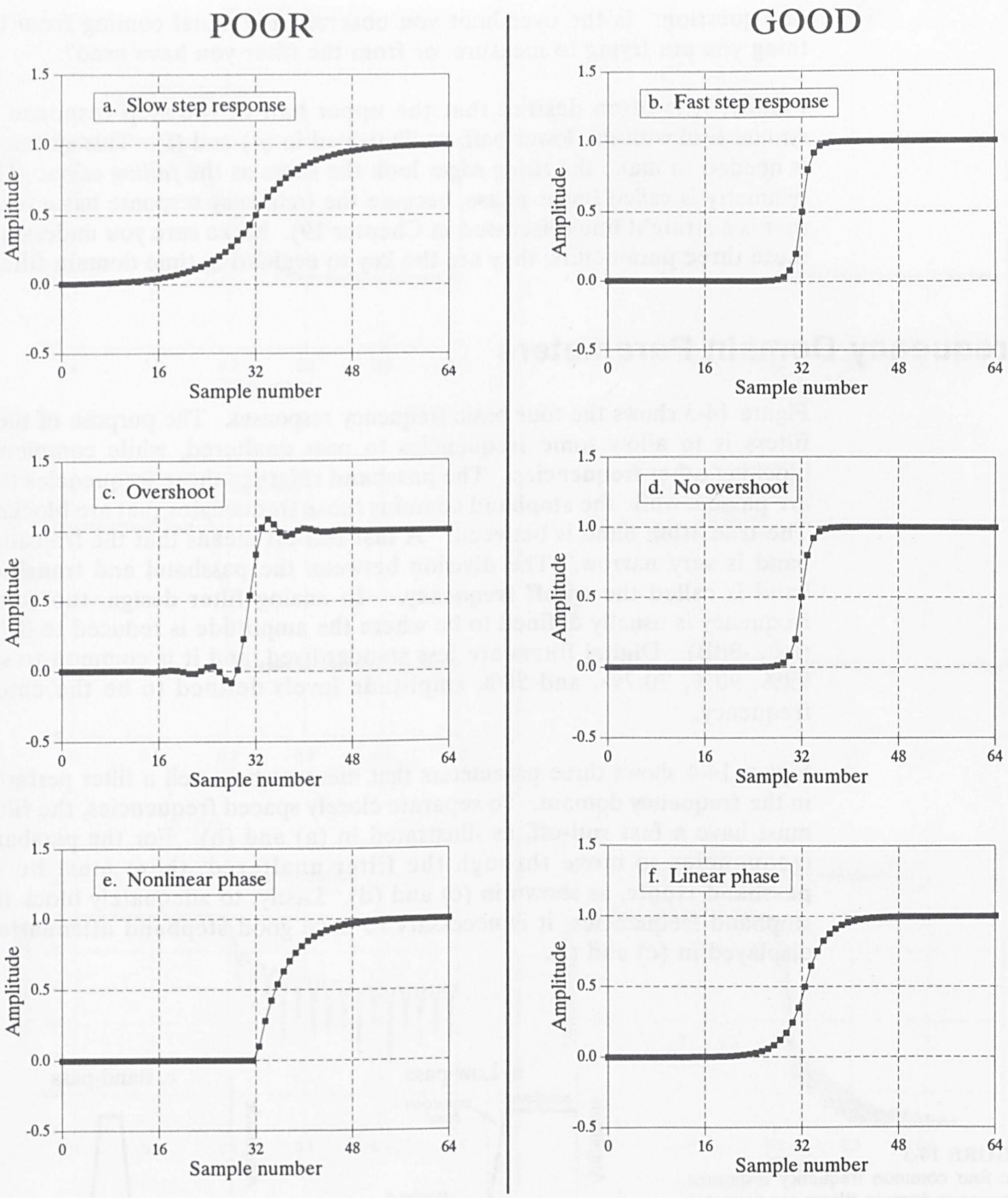


FIGURE 14-2
Parameters for evaluating *time domain* performance. The step response is used to measure how well a filter performs in the time domain. Three parameters are important: (1) transition speed (risetime), shown in (a) and (b), (2) overshoot, shown in (c) and (d), and (3) phase linearity (symmetry between the top and bottom halves of the step), shown in (e) and (f).

Figures (c) and (d) shows the next parameter that is important: **overshoot** in the step response. Overshoot must generally be eliminated because it changes the amplitude of samples in the signal; this is a basic distortion of the information contained in the time domain. This can be summed up in

one question: Is the overshoot you observe in a signal coming from the thing you are trying to measure, or from the filter you have used?

Finally, it is often desired that the upper half of the step response be symmetrical with the lower half, as illustrated in (e) and (f). This symmetry is needed to make the *rising edges* look the same as the *falling edges*. This symmetry is called **linear phase**, because the frequency response has a phase that is a straight line (discussed in Chapter 19). Make sure you understand these three parameters; they are the key to evaluating time domain filters.

Frequency Domain Parameters

Figure 14-3 shows the four basic frequency responses. The purpose of these filters is to allow some frequencies to pass unaltered, while completely blocking other frequencies. The **passband** refers to those frequencies that are passed, while the **stopband** contains those frequencies that are blocked. The **transition band** is between. A **fast roll-off** means that the transition band is very narrow. The division between the passband and transition band is called the **cutoff frequency.** In analog filter design, the cutoff frequency is usually defined to be where the amplitude is reduced to 0.707 (i.e., -3dB). Digital filters are less standardized, and it is common to see 99%, 90%, 70.7%, and 50% amplitude levels defined to be the cutoff frequency.

Figure 14-4 shows three parameters that measure how well a filter performs in the frequency domain. To separate closely spaced frequencies, the filter must have a **fast roll-off**, as illustrated in (a) and (b). For the passband frequencies to move through the filter unaltered, there must be no **passband ripple**, as shown in (c) and (d). Lastly, to adequately block the stopband frequencies, it is necessary to have good **stopband attenuation**, displayed in (e) and (f).

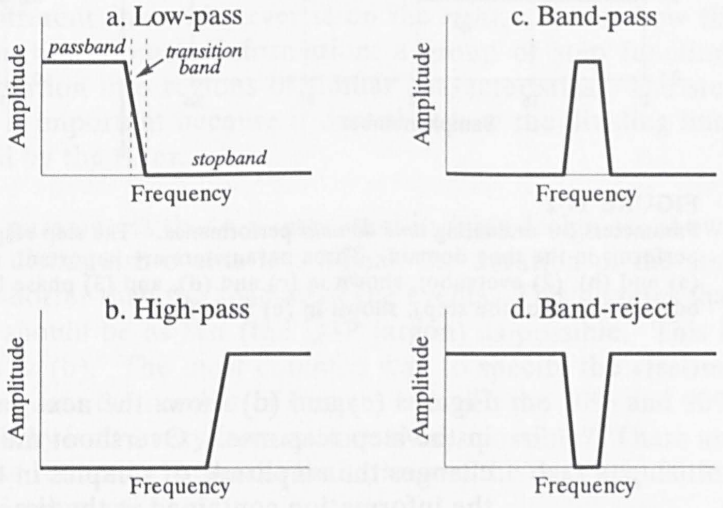


FIGURE 14-3
The four common frequency responses. Frequency domain filters are generally used to pass certain frequencies (the *passband*), while blocking others (the *stopband*). Four responses are the most common: low-pass, high-pass, band-pass, and band-reject.

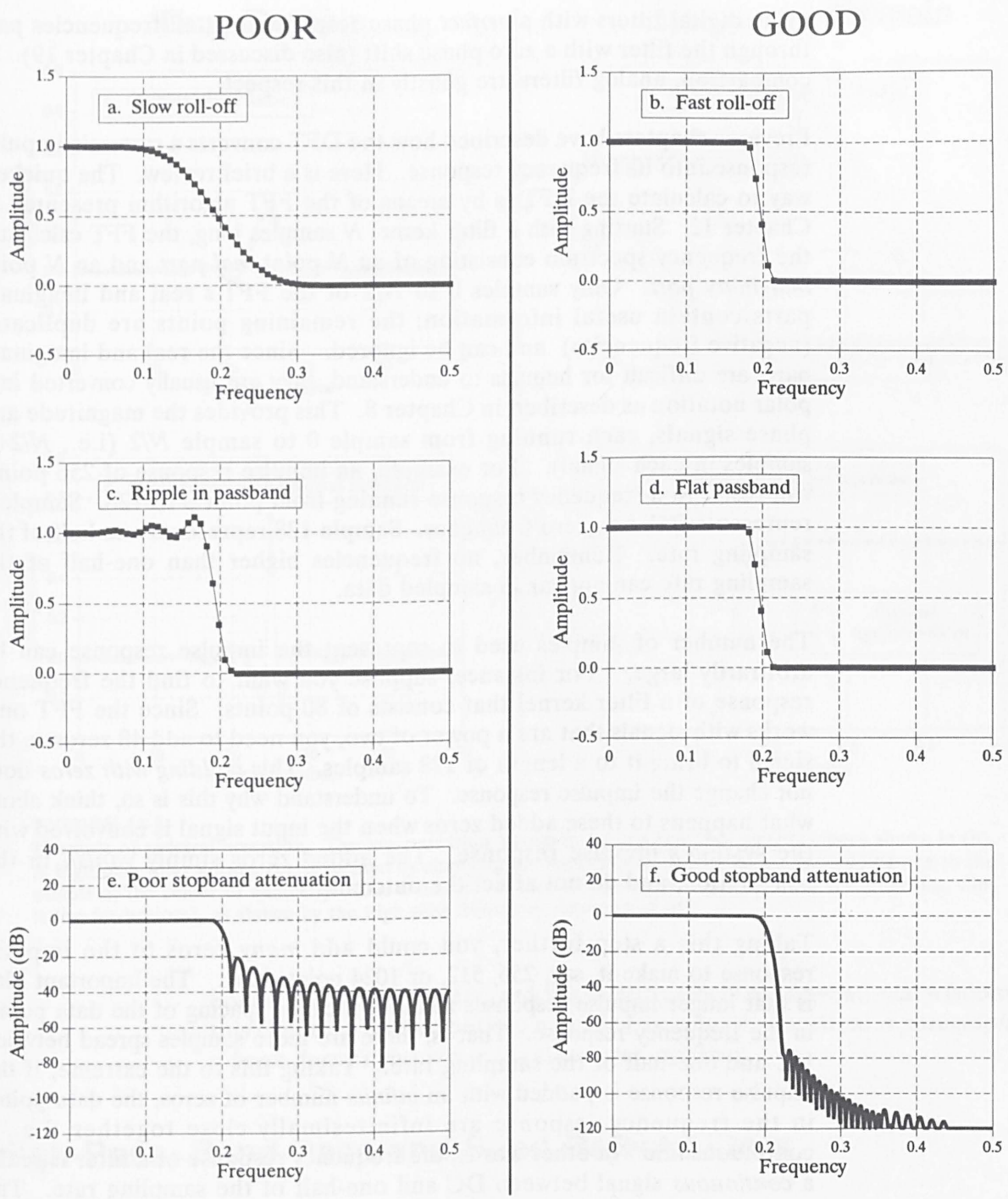


FIGURE 14-4
Parameters for evaluating *frequency domain* performance. The frequency responses shown are for low-pass filters. Three parameters are important: (1) roll-off sharpness, shown in (a) and (b), (2) passband ripple, shown in (c) and (d), and (3) stopband attenuation, shown in (e) and (f).

Why is there nothing about the *phase* in these parameters? First, the phase isn't important in most frequency domain applications. For example, the phase of an audio signal is almost completely random, and contains little useful information. Second, if the phase is important, it is very easy to

make digital filters with a *perfect* phase response, i.e., all frequencies pass through the filter with a zero phase shift (also discussed in Chapter 19). In comparison, analog filters are ghastly in this respect.

Previous chapters have described how the DFT converts a system's impulse response into its frequency response. Here is a brief review. The quickest way to calculate the DFT is by means of the FFT algorithm presented in Chapter 12. Starting with a filter kernel N samples long, the FFT calculates the frequency spectrum consisting of an N point *real part* and an N point *imaginary part*. Only samples 0 to $N/2$ of the FFT's real and imaginary parts contain useful information; the remaining points are duplicates (negative frequencies) and can be ignored. Since the real and imaginary parts are difficult for humans to understand, they are usually converted into polar notation as described in Chapter 8. This provides the magnitude and phase signals, each running from sample 0 to sample $N/2$ (i.e., $N/2+1$ samples in each signal). For example, an impulse response of 256 points will result in a frequency response running from point 0 to 128. Sample 0 represents DC, i.e., zero frequency. Sample 128 represents one-half of the sampling rate. Remember, no frequencies higher than one-half of the sampling rate can appear in sampled data.

The number of samples used to represent the impulse response can be arbitrarily large. For instance, suppose you want to find the frequency response of a filter kernel that consists of 80 points. Since the FFT only works with signals that are a power of two, you need to add 48 zeros to the signal to bring it to a length of 128 samples. This *padding with zeros* does not change the impulse response. To understand why this is so, think about what happens to these added zeros when the input signal is convolved with the system's impulse response. The added zeros simply *vanish* in the convolution, and do not affect the outcome.

Taking this a step further, you could add *many* zeros to the impulse response to make it, say, 256, 512, or 1024 points long. The important idea is that longer impulse responses result in a closer spacing of the data points in the frequency response. That is, there are more samples spread between DC and one-half of the sampling rate. Taking this to the extreme, if the impulse response is padded with an *infinite* number of zeros, the data points in the frequency response are infinitesimally close together, i.e., a continuous line. In other words, the frequency response of a filter is really a *continuous* signal between DC and one-half of the sampling rate. The output of the DFT is a *sampling* of this continuous line. What length of impulse response should you use when calculating a filter's frequency response? As a first thought, try $N=1024$, but don't be afraid to change it if needed (such as insufficient resolution or excessive computation time).

Keep in mind that the "good" and "bad" parameters discussed in this chapter are only generalizations. Many signals don't fall neatly into categories. For example, consider an EKG signal contaminated with 60 hertz interference. The information is encoded in the *time domain*, but the interference is best dealt with in the *frequency domain*. The best design for this application is

Nintendo Exhibit 1021
Page 019

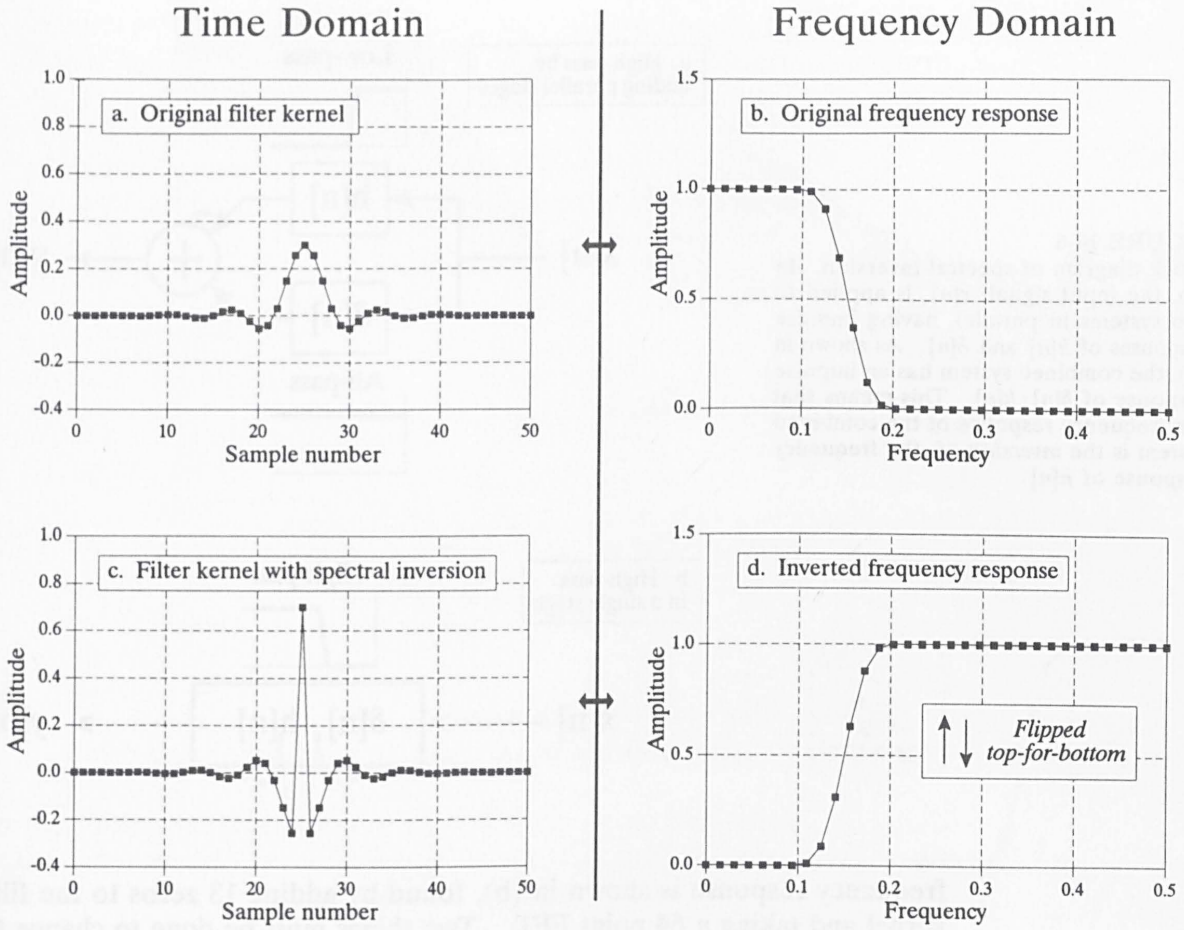


FIGURE 14-5
Example of spectral inversion. The low-pass filter kernel in (a) has the frequency response shown in (b). A high-pass filter kernel, (c), is formed by changing the sign of each sample in (a), and adding one to the sample at the center of symmetry. This action in the time domain *inverts* the frequency spectrum (i.e., flips it top-for-bottom), as shown by the high-pass frequency response in (d).

bound to have trade-offs, and might go against the conventional wisdom of this chapter. Remember the number one rule of education: *A paragraph in a book doesn't give you a license to stop thinking.*

High-Pass, Band-Pass and Band-Reject Filters

High-pass, band-pass and band-reject filters are designed by starting with a low-pass filter, and then converting it into the desired response. For this reason, most discussions on filter design only give examples of low-pass filters. There are two methods for the low-pass to high-pass conversion: **spectral inversion** and **spectral reversal**. Both are equally useful.

An example of *spectral inversion* is shown in 14-5. Figure (a) shows a low-pass filter kernel called a windowed-sinc (the topic of Chapter 16). This filter kernel is 51 points in length, although many of samples have a value so small that they appear to be zero in this graph. The corresponding

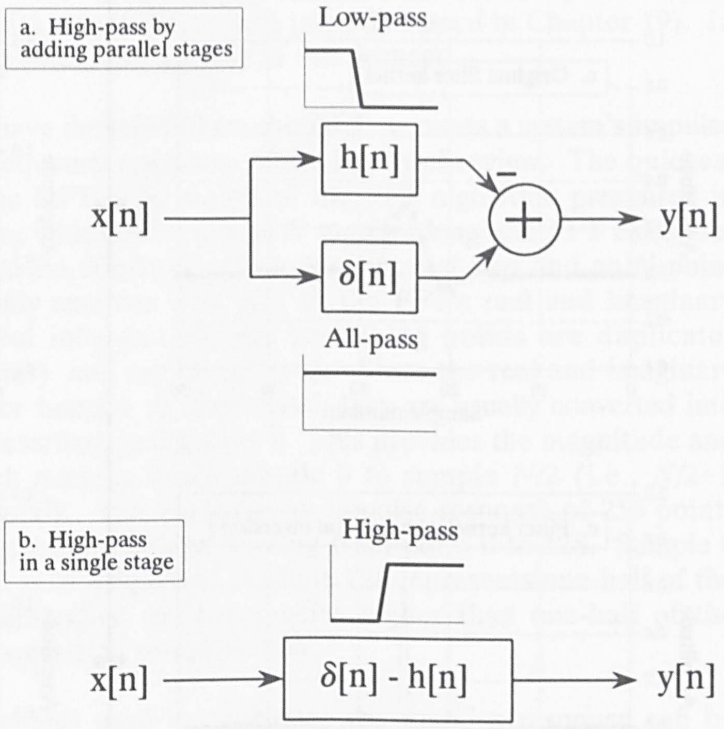


FIGURE 14-6
Block diagram of spectral inversion. In (a), the input signal, $x[n]$, is applied to two systems in parallel, having impulse responses of $h[n]$ and $\delta[n]$. As shown in (b), the combined system has an impulse response of $\delta[n]-h[n]$. This means that the frequency response of the combined system is the *inversion* of the frequency response of $h[n]$.

frequency response is shown in (b), found by adding 13 zeros to the filter kernel and taking a 64 point FFT. Two things must be done to change the low-pass filter kernel into a high-pass filter kernel. First, change the sign of each sample in the filter kernel. Second, add *one* to the sample at the center of symmetry. This results in the high-pass filter kernel shown in (c), with the frequency response shown in (d). Spectral inversion *flips* the frequency response *top-for-bottom*, changing the passbands into stopbands, and the stopbands into passbands. In other words, it changes a filter from low-pass to high-pass, high-pass to low-pass, band-pass to band-reject, or band-reject to band-pass.

Figure 14-6 shows why this two step modification to the time domain results in an inverted frequency spectrum. In (a), the input signal, $x[n]$, is applied to two systems in parallel. One of these systems is a low-pass filter, with an impulse response given by $h[n]$. The other system does *nothing* to the signal, and therefore has an impulse response that is a delta function, $\delta[n]$. The overall output, $y[n]$, is equal to the output of the all-pass system *minus* the output of the low-pass system. Since the low frequency components are subtracted from the original signal, only the high frequency components appear in the output. Thus, a high-pass filter is formed.

This could be performed as a two step operation in a computer program: run the signal through a low-pass filter, and then subtract the filtered signal from the original. However, the entire operation can be performed in a signal stage by combining the two filter kernels. As described in Chapter

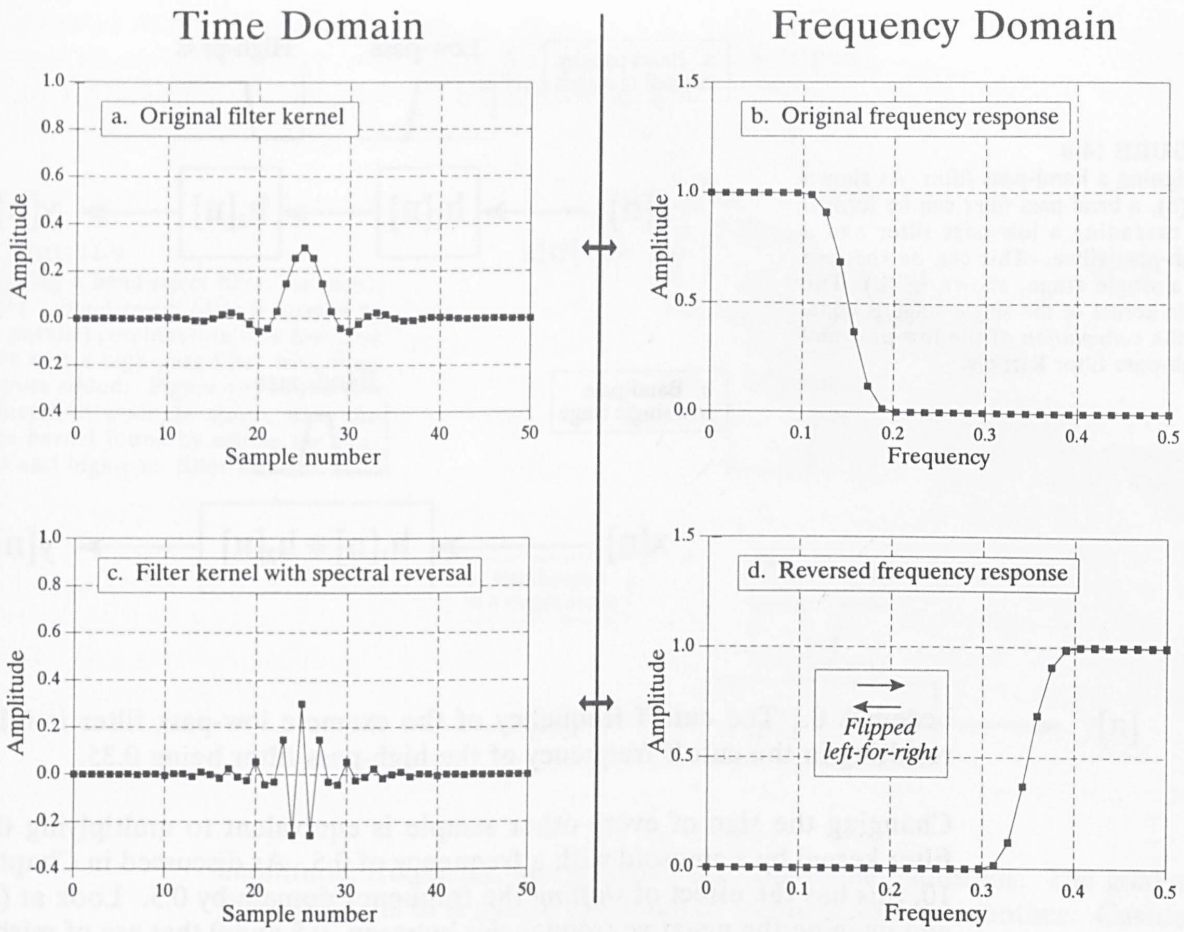


FIGURE 14-7
Example of spectral reversal. The low-pass filter kernel in (a) has the frequency response shown in (b). A high-pass filter kernel, (c), is formed by changing the sign of every other sample in (a). This action in the time domain results in the frequency domain being flipped *left-for-right*, resulting in the high-pass frequency response shown in (d).

7, parallel systems with added outputs can be combined into a single stage by adding their impulse responses. As shown in (b), the filter kernel for the high-pass filter is given by: $\delta[n] - h[n]$. That is, change the sign of all the samples, and then add one to the sample at the center of symmetry.

For this technique to work, the low-frequency components exiting the low-pass filter must have the same phase as the low-frequency components exiting the all-pass system. Otherwise a complete subtraction cannot take place. This places two restrictions on the method: (1) the original filter kernel must have left-right symmetry (i.e., a zero or linear phase), and (2) the impulse must be added at the center of symmetry.

The second method for low-pass to high-pass conversion, *spectral reversal*, is illustrated in Fig. 14-7. Just as before, the low-pass filter kernel in (a) corresponds to the frequency response in (b). The high-pass filter kernel, (c), is formed by *changing the sign of every other sample* in (a). As shown in (d), this flips the frequency domain *left-for-right*: 0 becomes 0.5 and 0.5

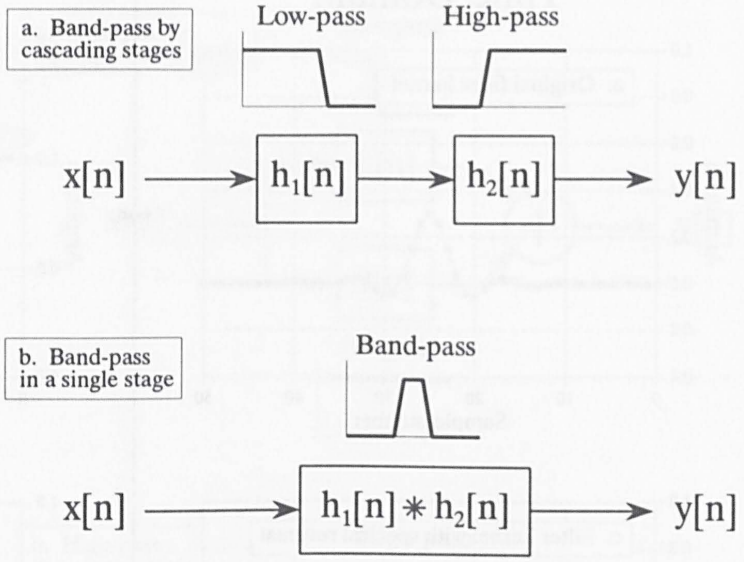


FIGURE 14-8
Deigning a band-pass filter. As shown in (a), a band-pass filter can be formed by cascading a low-pass filter and a high-pass filter. This can be reduced to a single stage, shown in (b). The filter kernel of the single stage is equal to the *convolution* of the low-pass and high-pass filter kernels.

becomes 0. The cutoff frequency of the example low-pass filter is 0.15, resulting in the cutoff frequency of the high-pass filter being 0.35.

Changing the sign of every other sample is equivalent to multiplying the filter kernel by a sinusoid with a frequency of 0.5. As discussed in Chapter 10, this has the effect of *shifting* the frequency domain by 0.5. Look at (b) and imagine the negative frequencies between -0.5 and 0 that are of mirror image of the frequencies between 0 and 0.5. The frequencies that appear in (d) are the negative frequencies from (b) shifted by 0.5.

Lastly, Figs. 14-8 and 14-9 show how low-pass and high-pass filter kernels can be combined to form band-pass and band-reject filters. In short, *adding* the filter kernels produces a *band-reject* filter, while *convolving* the filter kernels produces a *band-pass* filter. These are based on the way cascaded and parallel systems are be combined, as discussed in Chapter 7. Multiple combination of these techniques can also be used. For instance, a band-pass filter can be designed by adding the two filter kernels to form a band-pass filter, and then use *spectral inversion* or *spectral reversal* as previously described. All these techniques work very well with few surprises.

Filter Classification

Table 14-1 summarizes how digital filters are classified by their *use* and by their *implementation*. The use of a digital filter can be broken into three categories: *time domain*, *frequency domain* and *custom*. As previously described, time domain filters are used when the information is encoded in the shape of the signal's waveform. Time domain filtering is used for such actions as: smoothing, DC removal, waveform shaping, etc. In contrast, frequency domain filters are used when the information is contained in the

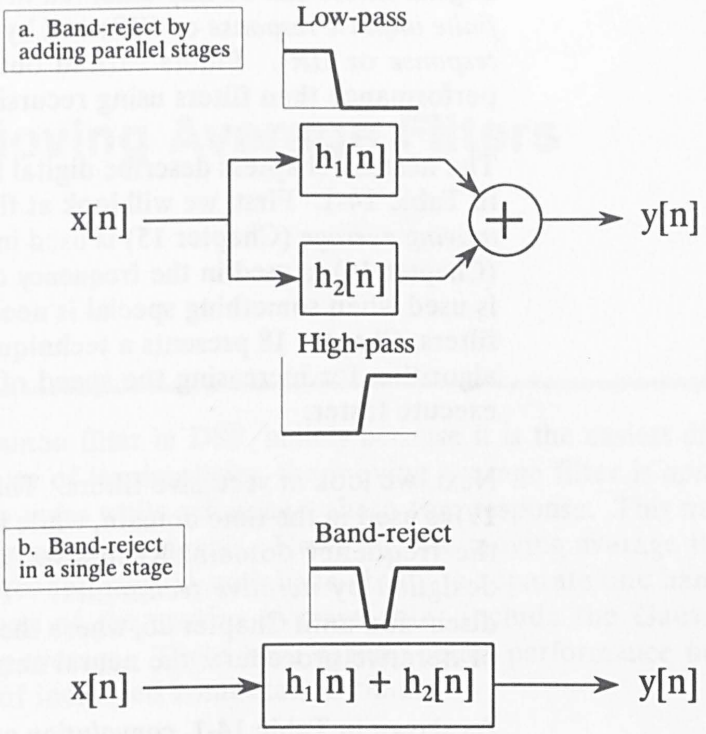


FIGURE 14-9
Deigning a band-reject filter. As shown in (a), a band-reject filter is formed by the parallel combination of a low-pass filter and a high-pass filter with their outputs added. Figure (b) shows this reduced to a single stage, with the filter kernel found by *adding* the low-pass and high-pass filter kernels.

amplitude, frequency, and phase of the component sinusoids. The goal of these filters is to separate one band of frequencies from another. Custom filters are used when a special action is required by the filter, something more elaborate than the four basic responses (high-pass, low-pass, band-pass and band-reject). For instance, Chapter 17 describes how custom filters can be used for *deconvolution*, a way of counteracting an unwanted convolution.

FILTER USED FOR:	FILTER IMPLEMENTED BY: Convolution *Finite Impulse Response (FIR)*	Recursion *Infinite Impulse Response (IIR)*
Time Domain *(smoothing, DC removal)*	Moving average (Ch. 15)	Single pole (Ch. 19)
Frequency Domain *(separating frequencies)*	Windowed-sinc (Ch. 16)	Chebyshev (Ch. 20)
Custom *(Deconvolution)*	FIR custom (Ch. 17)	Iterative design (Ch. 26)

Table 14-1
Filter classification. Filters can be divided by their *use*, and how they are *implemented*.

Digital filters can be implemented in two ways, by *convolution* (also called *finite impulse response* or *FIR*) and by *recursion* (also called *infinite impulse response* or *IIR*). Filters carried out by convolution can have far better performance than filters using recursion, but execute much more slowly.

The next six chapters describe digital filters according to the classifications in Table 14-1. First, we will look at filters carried out by convolution. The *moving average* (Chapter 15) is used in the time domain, the *windowed-sinc* (Chapter 16) is used in the frequency domain, and *FIR custom* (Chapter 17) is used when something special is needed. To finish the discussion of FIR filters, Chapter 18 presents a technique called FFT convolution. This is an algorithm for increasing the speed of convolution, allowing FIR filters to execute faster.

Next, we look at recursive filters. The *single pole* recursive filter (Chapter 19) is used in the time domain, while the *Chebyshev* (Chapter 20) is used in the frequency domain. Recursive filters having a custom response are designed by *iterative techniques*. For this reason, we will delay their discussion until Chapter 26, where they will be presented with another type of iterative procedure: the neural network.

As shown in Table 14-1, *convolution* and *recursion* are rival techniques; you must use one or the other for a particular application. How do you choose? Chapter 21 presents a head-to-head comparison of the two, in both the time and frequency domains.

Nintendo Exhibit 1021
Page 025