mobile communications series

SOFTWARE-DEFINED RADIO for ENGINEERS

TRAVIS F. COLLINS
ROBIN GETZ
DI PU
ALEXANDER M. WYGLINSKI

Software-Defined Radio for Engineers

For a listing of recent titles in the *Artech House Mobile Communications*, turn to the back of this book.

Software-Defined Radio for Engineers

Travis F. Collins
Robin Getz
Di Pu
Alexander M. Wyglinski

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalog record for this book is available from the British Library.

ISBN-13: 978-1-63081-457-1

Cover design by John Gomes

10 9 8 7 6 5 4 3 2 1

Dedication

To my wife Lauren
—Travis Collins

To my wonderful children, Matthew, Lauren, and Isaac, and my patient wife, Michelle—sorry I have been hiding in the basement working on this book. To all my fantastic colleagues at Analog Devices: Dave, Michael, Lars-Peter, Andrei, Mihai, Travis, Wyatt and many more, without whom Pluto SDR and IIO would not exist.
—Robin Getz

To my lovely son Aidi, my husband Di, and my parents Lingzhen and Xuexun
—Di Pu

To my wife Jen
—Alexander Wyglinski

Contents

Preface xiii

CHAPTER 1
Introduction to Software-Defined Radio 1

1.1 Brief History 1
1.2 What is a Software-Defined Radio? 1
1.3 Networking and SDR 7
1.4 RF architectures for SDR 10
1.5 Processing architectures for SDR 13
1.6 Software Environments for SDR 15
1.7 Additional readings 17
References 18

CHAPTER 2
Signals and Systems 19

2.1 Time and Frequency Domains 19
2.1.1 Fourier Transform 20
2.1.2 Periodic Nature of the DFT 21
2.1.3 Fast Fourier Transform 22
2.2 Sampling Theory 23
2.2.1 Uniform Sampling 23
2.2.2 Frequency Domain Representation of Uniform Sampling 25
2.2.3 Nyquist Sampling Theorem 26
2.2.4 Nyquist Zones 29
2.2.5 Sample Rate Conversion 29
2.3 Signal Representation 37
2.3.1 Frequency Conversion 38
2.3.2 Imaginary Signals 40
2.4 Signal Metrics and Visualization 41
2.4.1 SINAD, ENOB, SNR, THD, THD + N, and SFDR 42
2.4.2 Eye Diagram 44
2.5 Receive Techniques for SDR 45
2.5.1 Nyquist Zones 47
2.5.2 Fixed Point Quantization 49

2.5.3 Design Trade-offs for Number of Bits, Cost, Power, and So Forth 55
2.5.4 Sigma-Delta Analog-Digital Converters 58
2.6 Digital Signal Processing Techniques for SDR 61
2.6.1 Discrete Convolution 61
2.6.2 Correlation 65
2.6.3 Z-Transform 66
2.6.4 Digital Filtering 69
2.7 Transmit Techniques for SDR 73
2.7.1 Analog Reconstruction Filters 75
2.7.2 DACs 76
2.7.3 Digital Pulse-Shaping Filters 78
2.7.4 Nyquist Pulse-Shaping Theory 79
2.7.5 Two Nyquist Pulses 81
2.8 Chapter Summary 85
References 85

CHAPTER 3
Probability in Communications 87
3.1 Modeling Discrete Random Events in Communication Systems 87
3.1.1 Expectation 89
3.2 Binary Communication Channels and Conditional Probability 92
3.3 Modeling Continuous Random Events in Communication Systems 95
3.3.1 Cumulative Distribution Functions 99
3.4 Time-Varying Randomness in Communication Systems 101
3.4.1 Stationarity 104
3.5 Gaussian Noise Channels 106
3.5.1 Gaussian Processes 108
3.6 Power Spectral Densities and LTI Systems 109
3.7 Narrowband Noise 110
3.8 Application of Random Variables: Indoor Channel Model 113
3.9 Chapter Summary 114
3.10 Additional Readings 114
References 115

CHAPTER 4
Digital Communications Fundamentals 117
4.1 What Is Digital Transmission? 117
4.1.1 Source Encoding 120
4.1.2 Channel Encoding 122
4.2 Digital Modulation 127
4.2.1 Power Efficiency 128
4.2.2 Pulse Amplitude Modulation 129

4.2.3 Quadrature Amplitude Modulation 131
4.2.4 Phase Shift Keying 133
4.2.5 Power Efficiency Summary 139
4.3 Probability of Bit Error 141
4.3.1 Error Bounding 145
4.4 Signal Space Concept 148
4.5 Gram-Schmidt Orthogonalization 150
4.6 Optimal Detection 154
4.6.1 Signal Vector Framework 155
4.6.2 Decision Rules 158
4.6.3 Maximum Likelihood Detection in an AWGN Channel 159
4.7 Basic Receiver Realizations 160
4.7.1 Matched Filter Realization 161
4.7.2 Correlator Realization 164
4.8 Chapter Summary 166
4.9 Additional Readings 168
References 169

CHAPTER 5
Understanding SDR Hardware 171
5.1 Components of a Communication System 171
5.1.1 Components of an SDR 172
5.1.2 AD9363 Details 173
5.1.3 Zynq Details 176
5.1.4 Linux Industrial Input/Output Details 177
5.1.5 MATLAB as an IIO client 178
5.1.6 Not Just for Learning 180
5.2 Strategies For Development in MATLAB 181
5.2.1 Radio I/O Basics 181
5.2.2 Continuous Transmit 183
5.2.3 Latency and Data Delays 184
5.2.4 Receive Spectrum 185
5.2.5 Automatic Gain Control 186
5.2.6 Common Issues 187
5.3 Example: Loopback with Real Data 187
5.4 Noise Figure 189
References 190

CHAPTER 6
Timing Synchronization 191
6.1 Matched Filtering 191
6.2 Timing Error 195
6.3 Symbol Timing Compensation 198

6.3.1 Phase-Locked Loops 200
6.3.2 Feedback Timing Correction 201
6.4 Alternative Error Detectors and System Requirements 208
6.4.1 Gardner 208
6.4.2 Müller and Mueller 208
6.5 Putting the Pieces Together 209
6.6 Chapter Summary 212
References 212

CHAPTER 7
Carrier Synchronization 213
7.1 Carrier Offsets 213
7.2 Frequency Offset Compensation 216
7.2.1 Coarse Frequency Correction 217
7.2.2 Fine Frequency Correction 219
7.2.3 Performance Analysis 224
7.2.4 Error Vector Magnitude Measurements 226
7.3 Phase Ambiguity 228
7.3.1 Code Words 228
7.3.2 Differential Encoding 229
7.3.3 Equalizers 229
7.4 Chapter Summary 229
References 230

CHAPTER 8
Frame Synchronization and Channel Coding 231
8.1 O Frame, Where Art Thou? 231
8.2 Frame Synchronization 232
8.2.1 Signal Detection 235
8.2.2 Alternative Sequences 239
8.3 Putting the Pieces Together 241
8.3.1 Full Recovery with Pluto SDR 242
8.4 Channel Coding 244
8.4.1 Repetition Coding 244
8.4.2 Interleaving 245
8.4.3 Encoding 246
8.4.4 BER Calculator 251
8.5 Chapter Summary 251
References 251

CHAPTER 9
Channel Estimation and Equalization 253
9.1 You Shall Not Multipath! 253

9.2 Channel Estimation 254
9.3 Equalizers 258
9.3.1 Nonlinear Equalizers 261
9.4 Receiver Realization 263
9.5 Chapter Summary 265
References 266

CHAPTER 10
Orthogonal Frequency Division Multiplexing 267
10.1 Rationale for MCM: Dispersive Channel Environments 267
10.2 General OFDM Model 269
10.2.1 Cyclic Extensions 269
10.3 Common OFDM Waveform Structure 271
10.4 Packet Detection 273
10.5 CFO Estimation 275
10.6 Symbol Timing Estimation 279
10.7 Equalization 280
10.8 Bit and Power Allocation 284
10.9 Putting It All Together 285
10.10 Chapter Summary 286
References 286

CHAPTER 11
Applications for Software-Defined Radio 289
11.1 Cognitive Radio 289
11.1.1 Bumblebee Behavioral Model 292
11.1.2 Reinforcement Learning 294
11.2 Vehicular Networking 295
11.3 Chapter Summary 299
References 299

APPENDIX A
A Longer History of Communications 303
A.1 History Overview 303
A.2 1750–1850: Industrial Revolution 304
A.3 1850–1945: Technological Revolution 305
A.4 1946–1960: Jet Age and Space Age 309
A.5 1970–1979: Information Age 312
A.6 1980–1989: Digital Revolution 313
A.7 1990–1999: Age of the Public Internet (Web 1.0) 316
A.8 Post-2000: Everything comes together 319
References 319

APPENDIX B
Getting Started with MATLAB and Simulink 327
B.1 MATLAB Introduction 327
B.2 Useful MATLAB Tools 327
B.2.1 Code Analysis and M-Lint Messages 328
B.2.2 Debugger 329
B.2.3 Profiler 329
B.3 System Objects 330
References 332

APPENDIX C
Equalizer Derivations 333
C.1 Linear Equalizers 333
C.2 Zero-Forcing Equalizers 335
C.3 Decision Feedback Equalizers 336

APPENDIX D
Trigonometric Identities 337

About the Authors 339

Index 341

CHAPTER 2

Signals and Systems

SDR is an application-specific area of signal processing, and everyone involved in SDR development needs to not only have a solid background in signals and systems, but also RF and analog baseband processing. This chapter covers the many aspects of linear time-invariant (LTI) signals and systems, analog processing, and RF that forms the fundamental basis for the latter chapters. It is not expected that the reader will go through each topic in detail, but if any of the short topics are new, the reader should refer to the more comprehensive books on these subjects.

2.1 Time and Frequency Domains

The time and frequency domains are alternative ways of representing the same signals. The Fourier transform, named after its initial instigator, French mathematician and physicist Jean-Baptiste Joseph Fourier, is the mathematical relationship between these two representations. If a signal is modified in one domain, it will also be changed in the other domain, although usually not in the same way. For example, convolution in the time domain is equivalent to multiplication in the frequency domain. Other mathematical operations, such as addition, scaling, and shifting, also have a matching operation in the opposite domain. These relationships are called properties of the Fourier transform, which describe how a mathematical change in one domain results in a mathematical change in the other domain.

The Fourier transform is just a different way to describe a signal. For example, investigating the Gibbs phenomenon, which states if you add sine waves at specific frequency/phase/amplitude combinations you can approximate a square wave, can be expressed mathematically as (2.1), and shown in Figure 2.1.

$$x(t) = \sin(t) + \frac{\sin(3t)}{3} + \frac{\sin(5t)}{5} + ... = \sum_{n=1}^{n=\infty} \frac{\sin(n \times t)}{n}; \quad n = odd \tag{2.1}$$

When we look at the signal across the time axis that is perpendicular to the frequency axis, we observe the time domain. We cannot see the frequency of the sine waves easily since we are perpendicular to the frequency axis. When we transform domains and observe phenomenon across the frequency axis, which is perpendicular to the time axis, we observe the frequency or Fourier domain. We can easily make out the signal magnitude and frequency, but have lost that time aspect. Both views represents the same signal such that, they are just being observed things from different domains via transforms.

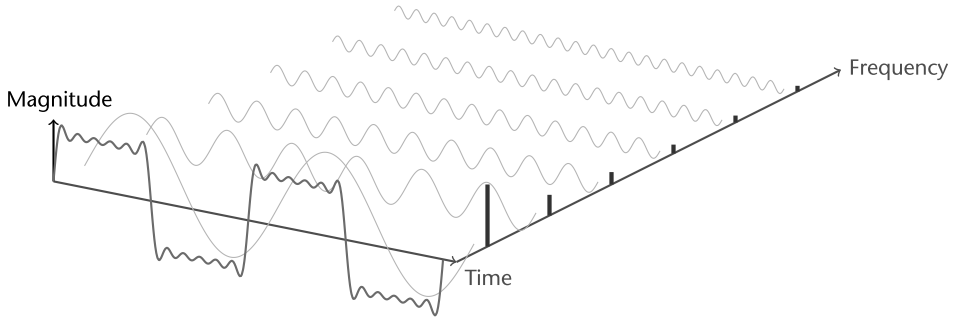


Figure 2.1 Gibbs phenomenon, looking left, the time domain, looking right, the Fourier domain.

Investigate the Gibbs phenomenon in MATLAB by using Code 2.1 to better understand how adding sine waves with variations in frequency/phase/amplitude affect the time domain, and frequency domains

Code 2.1 Gibbs phenomenon: **`gibbs.m`**

```
 2 max = 9;
 3 fs = 1000;
11 for i = 1:2:max
12     % dsp.SineWave(amp,freq,phase,Name,Value);
13     wave = dsp.SineWave(1/i, i*2*pi, 0, ...
14     'SamplesPerFrame', 5000, 'SampleRate', fs);
15     y = wave();
16     if i == 1
17         wavesum = y;
18     else
19         wavesum = wavesum + y;
20     end
28     scope(wavesum());
29     pause(.5);
30     % waitforbuttonpress;
31 end
```

2.1.1 Fourier Transform

The Fourier transform includes four members in its family: the Fourier transform, Fourier series, discrete Fourier transform (DFT), and discrete-time Fourier transform (DTFT). The commonly referred to FFT (fast Fourier transform) and its inverse, the inverse FFT (IFFT), is a specific implementation of the DFT.

The Fourier transform of $x(t)$ is defined as [1]:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt, \tag{2.2}$$

where t is the time variable in seconds across the time domain, and ω is the frequency variable in radian per seconds across frequency domain.

Applying the similar transform to $X(\omega)$ yields the inverse Fourier transform [1]:

$$x(t) = \int_{-\infty}^{\infty} X(\omega) e^{j2\pi\omega t} d\omega, \tag{2.3}$$

where we write $x(t)$ as a weighted sum of complex exponentials.

The Fourier transform pair above can be denoted as [2]:

$$x(t) \overset{\mathcal{F}}{\leftrightarrow} X(\omega), \tag{2.4}$$

where the left-hand side of the symbol $\overset{\mathcal{F}}{\leftrightarrow}$ is before Fourier transform, while the right-hand side of the symbol $\overset{\mathcal{F}}{\leftrightarrow}$ is after Fourier transform. There are several commonly used properties of Fourier transform that are useful when studying SDR Fourier domain, which have been summarized in Table 2.1 for your reference.

2.1.2 Periodic Nature of the DFT

Unlike the other three Fourier transforms, the DFT views both the time domain and the frequency domain signals as periodic (they repeat forever). This can be confusing and inconvenient since most of the signals used in many signal processing applications are not periodic. Nevertheless, if you want to use the DFT (and its fast implementation, the FFT), you must conform with the DFT's view of the world.

Figure 2.2 shows two different interpretations of the time domain signal. First, observing the upper signal, the time domain viewed as N points. This represents how signals are typically acquired by SDRs, by a buffer of N points. For instance, these 128 samples might have been acquired by sampling some analog signal at regular intervals of time. Sample 0 is distinct and separate from sample 127 because they were acquired at different times. From the way this signal was formed, there is no reason to think that the samples on the left of the signal are even related to the samples on the right.

Unfortunately, the DFT does not see things this way. As shown in the lower part of Figure 2.2, the DFT views these 128 points to be a single period of an infinitely long periodic signal. This means that the left side of the acquired signal is connected to the right side of a duplicate signal. Likewise, the right side of the acquired signal

Table 2.1 Fourier Transform Properties*

Property	*Time Signal*	*Fourier Transform Signal*
Definition	$x(t)$	$\int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$
Inversion formula	$\int_{-\infty}^{\infty} X(\omega)e^{j2\pi\omega t}d\omega$	$X(\omega)$
Linearity	$\sum_{n=1}^{N} a_n x_n(t)$	$\sum_{n=1}^{N} a_n X_n(\omega)$
Symmetry	$x(-t)$	$X(-\omega)$
Time shift	$x(t-t_0)$	$X(\omega)e^{-j\omega t_0}$
Frequency shift	$x(t)e^{j\omega_0 t}$	$X(\omega-\omega_0)$
Scaling	$x(\alpha t)$	$\frac{1}{\lvert\alpha\rvert}X(\frac{\omega}{\alpha})$
Derivative	$\frac{d^n}{dt^n}x(t)$	$(j\omega)^n X(\omega)$
Integration	$\int_{-\infty}^{\infty} x(\tau)d\tau$	$\frac{X(\omega)}{j\omega} + \pi X(0)\delta(\omega)$
Time convolution	$x(t) * h(t)$	$X(\omega)H(\omega)$
Frequency convolution	$x(t)h(t)$	$\frac{1}{2\pi}X(\omega) * H(\omega)$

* based on [2]. Suppose the time signal is $x(t)$, and its Fourier transform signal is $X(\omega)$

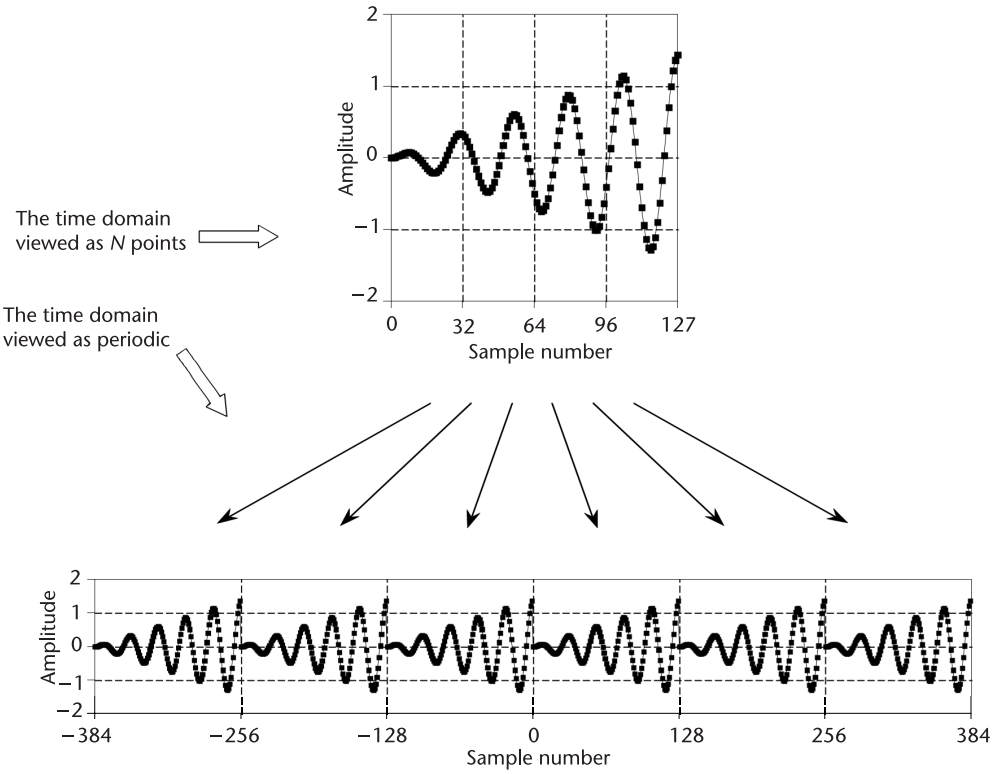


Figure 2.2 Periodicity of the DFT's time domain signal. The time domain can be viewed as N samples in length, shown in the upper figure, or as an infinitely long periodic signal, shown in the lower figures [4].

is connected to the left side of an identical period. This can also be thought of as the right side of the acquired signal wrapping around and connecting to its left side. In this view, sample 127 occurs next to sample 0, just as sample 43 occurs next to sample 44. This is referred to as being circular, and is identical to viewing the signal as being periodic. This is the reason that window [3] functions need to be preapplied to signal captures before applying an FFT function, which is multiplied by the signal and removes the discontinuities by forcing them to zero [4].

2.1.3 Fast Fourier Transform

There are several ways to calculate the DFT, such as solving simultaneous linear equations or correlation method. The FFT is another method for calculating the DFT. While it produces the same result as the other approaches, it is incredibly more efficient, often reducing the computation time by multiple orders of magnitude. While the FFT only requires a few dozen lines of code, it is one of the more complicated algorithms in signal processing, and its internal workings details are left to those that specialize in such things. You can easily use existing and proven FFT routines [4, 5] without fully understanding the internal workings as long as you understand how it is operating.

An FFT analysis using a generalized test setup shown in Figure 2.3. The spectral output of the FFT is a series of $\frac{M}{2}$ points in the frequency domain (M is the size of the FFT, the number of samples stored in the buffer memory). The spacing between

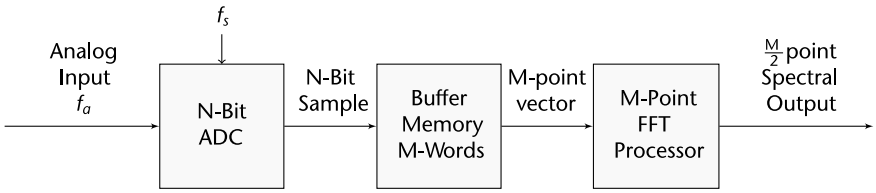


Figure 2.3 Generalized test set up for FFT analysis of ADC output [6].

the points is $\frac{f_s}{M}$, and the total frequency range covered is DC to $\frac{f_s}{2}$, where f_s is the sampling rate. The width of each frequency bin (sometimes called the resolution of the FFT) is $\frac{f_s}{M}$.

Figure 2.4 shows an FFT output for an ideal 12-bit ADC using the Analog Devices' ADIsimADC®program. Note that the theoretical noise floor of the FFT is equal to the theoretical signal-to-noise ratio (SNR) plus the FFT process gain of $10log_{10}(\frac{M}{2})$. It is important to remember the value for noise used in the SNR calculation is the noise that extends over the entire Nyquist bandwidth (DC to $\frac{f_s}{2}$), but the FFT acts as a narrowband spectrum analyzer with a bandwidth of $\frac{f_s}{M}$ that sweeps over the spectrum. This has the effect of pushing the noise down by an amount equal to the process gain—the same effect as narrowing the bandwidth of an analog spectrum analyzer.

The FFT output can be used like an analog spectrum analyzer to measure the amplitude of the various harmonics and noise components of a digitized signal. The harmonics of the input signal can be distinguished from other distortion products by their location in the frequency spectrum.

2.2 Sampling Theory

A continuous-time analog signal can be converted to a discrete-time digital signal using sampling and quantization, as shown in Figure 2.5, where a continuous analog input signal $x_a(t)$ is converted to a discrete digital output signal $x[n]$. *Sampling* is the conversion of a continuous-time signal into a discrete-time signal obtained by taking the samples of the continuous-time signal at discrete-time instants [1]. The quantization process converts the sample amplitude into a digital format. Section 2.2.1 will introduce a frequently used sampling method; namely, *uniform sampling*.

Similarly, a discrete-time signal can also be converted to a continuous-time signal using reconstruction. However, reconstruction is not always successful. Sometimes, the reconstructed signal is not the same as the original signal. Since for a given sampled signal, it can represent an infinite number of different continuous-time signals that can fit into the same quantized sample points. However, if the sampling satisfies certain criterion, the signal can be reconstructed without losing information. This criterion, called Nyquist sampling theorem, will be introduced in Sections 2.2.3 and 2.5.1.

2.2.1 Uniform Sampling

There are many ways to perform sampling of an analog signal into a digital representation. However, if we specify the sampling interval as a constant number

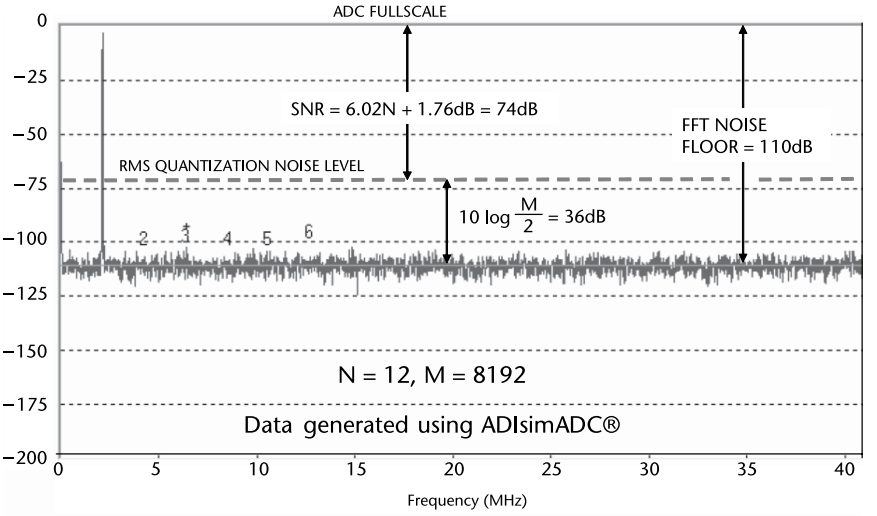


Figure 2.4 FFT output for an ideal 12-bit ADC, $f_a = 2.111$ MHz, $f_s = 82$ MSPS, average of 5 FFTs, $M = 8192$. Data generated from ADIsimADC® [6].

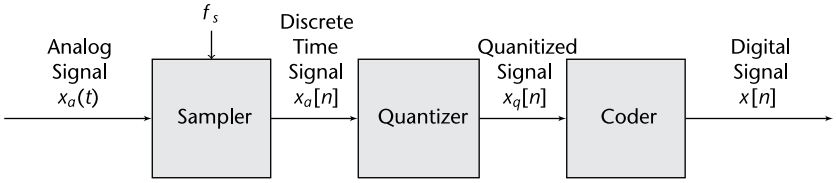


Figure 2.5 Basic parts of an analog-to-digital converter (ADC) [1]. Sampling takes place in the sampler block. $x_a(t)$ = analog continuous time signal; f_s is the digital sample rate; $x_a[n]$ is the discrete time continuous analog signal; $x_q[n]$ is the discrete time, discrete digital signal, which may come out as grey code; and $x[n]$ is the output of the coder in 2s complement form [7].

T_s, we get the most widely used sampling, called uniform sampling or periodic sampling. Using this method, we are taking samples of the continuous-time signal every T_s seconds, which can be defined as

$$x[n] = x(nT_s), \quad -\infty < n < \infty, \tag{2.5}$$

where $x(t)$ is the input continuous-time signal, $x[n]$ is the output discrete-time signal, T_s is the sampling period, and $f_s = 1/T_s$ is the sampling frequency.

An equivalent model for the uniform sampling operation is shown in Figure 2.6(a), where the continuous-time signal $x(t)$ is multiplied by an impulse train $p(t)$ to form the sampled signal $x_s(t)$, which can be defined as

$$x_s(t) = x(t)p(t), \tag{2.6}$$

where the signal $p(t)$ is referred to as the sampling function.

The sampling function is assumed to be a series of narrow pulses, which is either zero or one. Thus, $x_s(t) = x(t)$ when $p(t) = 1$, and $x_s(t) = 0$ when $p(t) = 0$. Since $p(t) = 1$ only at time instants T_s, the resulting $x_s(t) = x(nT_s) = x[n]$, which proves that this is indeed an equivalent model for the uniform sampling operation. This model will help us to obtain the frequency domain representation of uniform sampling in Section 2.2.2.

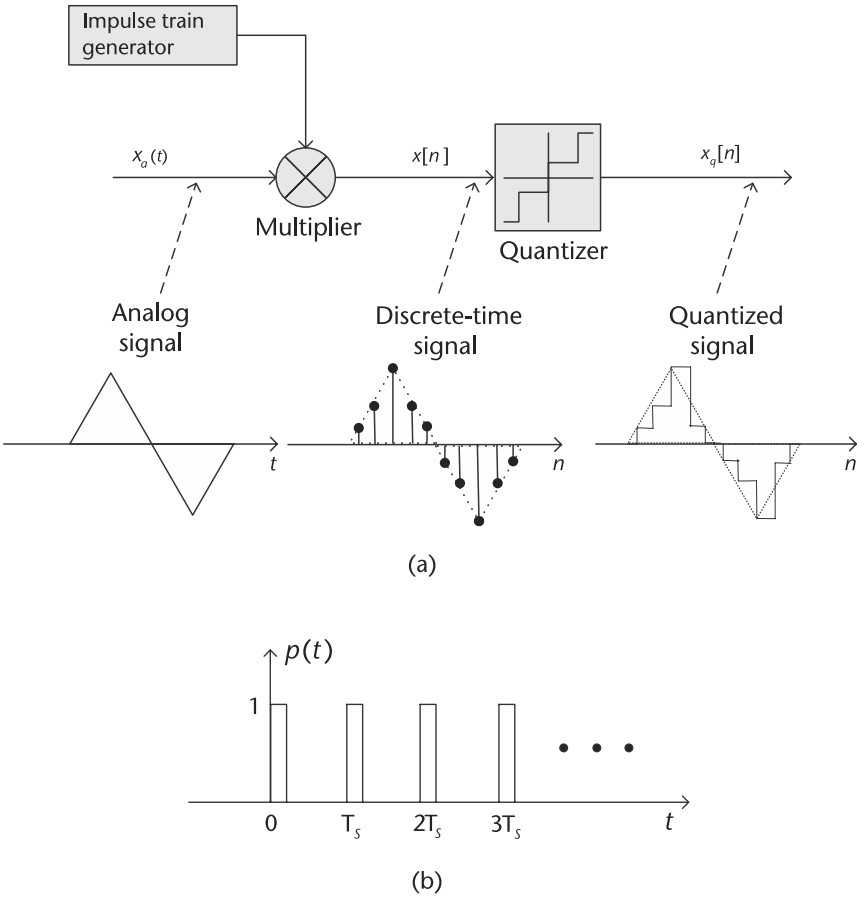


Figure 2.6 An equivalent model for the uniform sampling operation. (a) A continuous-time signal $x(t)$ is multiplied by a periodic pulse $p(t)$ to form the sampled signal $x_s(t)$, and (b) a periodic pulse $p(t)$.

2.2.2 Frequency Domain Representation of Uniform Sampling

Since it is easier to derive the Nyquist sampling theorem in frequency domain, in this section we will try to represent the uniform sampling process in frequency domain.

According to Figure 2.6(b), we can define the sampling function $p(t)$ as

$$p(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s), \quad k = 0, 1, 2, ..., \tag{2.7}$$

where at time instants kT_s, we have $p(t) = 1$. According to [8], $p(t)$ is a Dirac comb constructed from Dirac delta functions.

Substitution of (2.7) into (2.6) gives

$$x_s(t) = x(t)p(t) = x(t) \sum_{k=-\infty}^{\infty} \delta(t - kT_s). \tag{2.8}$$

In order to understand the sampling process in frequency domain, we need to take the Fourier transform of $x_s(t)$. According to frequency-domain convolution

property in Table 2.1, multiplication in time domain will lead to convolution in frequency domain. Therefore, multiplication of $x(t)$ and $p(t)$ will yield the convolution of $X(\omega)$ and $P(\omega)$:

$$X_s(\omega) = \frac{1}{2\pi} X(\omega) * P(\omega), \tag{2.9}$$

where $X(\omega)$ is the Fourier transform of $x(t)$, and $P(\omega)$ is the Fourier transform of $p(t)$.

The Fourier transform of a Dirac comb is also a Dirac comb [8], namely

$$P(\omega) = \frac{\sqrt{2\pi}}{T_s} \sum_{k=-\infty}^{\infty} \delta(\omega - k\frac{2\pi}{T_s}) = \frac{\sqrt{2\pi}}{T_s} \sum_{k=-\infty}^{\infty} \delta(\omega - k\omega_s), \tag{2.10}$$

where $\omega_s = 2\pi f_s$ is the sampling frequency.

Performing convolution with a collection of delta function pulses at the pulse location, we get

$$X_s(\omega) = \frac{1}{\sqrt{2\pi} T_s} \sum_{k=-\infty}^{\infty} X(\omega - k\omega_s). \tag{2.11}$$

Equation (2.11) tells us that the uniform sampling creates images of the Fourier transform of the input signal, and images are periodic with sampling frequency f_s.

2.2.3 Nyquist Sampling Theorem

Based on (2.11), we draw the spectrum of original signal $x(t)$ and the sampled signal $x_s(t)$ on frequency domain, as shown in Figure 2.7. We assume the bandwidth of the original signal is $[-f_h, f_h]$, as shown in Figure 2.7(a). For now, we do not pay attention to the signal amplitude, so we use A and A_s to represent them. Assuming the sampling frequency is f_s, then the sampled signal will have replicas at location kf_s. In order to reconstruct the original signal from the sampled signal, we will apply a lowpass filter on the sampled signal, trying to extract the $n = 0$ term from $X_s(f)$, as shown in Figure 2.7(b). Therefore, accomplishing reconstruction without error requires that the portion of the spectrum of $X_s(f)$ at $f = \pm f_s$ does not overlap with the portion of the spectrum at $f = 0$. In other words, this requires that $f_s - f_h > f_h$ or $f_s > 2f_h$, which leads to the Nyquist sampling theorem.

Nyquist sampling theorem applies for the *bandlimited signal*, which is a signal $x(t)$ that has no spectral components beyond a frequency B Hz [9]. That is,

$$X(\omega) = 0, \quad |\omega| > 2\pi B. \tag{2.12}$$

The Nyquist sampling theorem states that a real signal, $x(t)$, which is bandlimited to B Hz can be reconstructed without error from samples taken uniformly at a rate $R > 2B$ samples per second. This minimum sampling frequency, $F_s = 2B$ Hz, is called the Nyquist rate or the Nyquist frequency. The corresponding sampling interval, $T = \frac{1}{2B}$, is called the Nyquist interval [1]. A signal bandlimited to B Hz, which is sampled at less than the Nyquist frequency of $2B$ (i.e., which was sampled at an interval $T > \frac{1}{2B}$), is said to be undersampled.

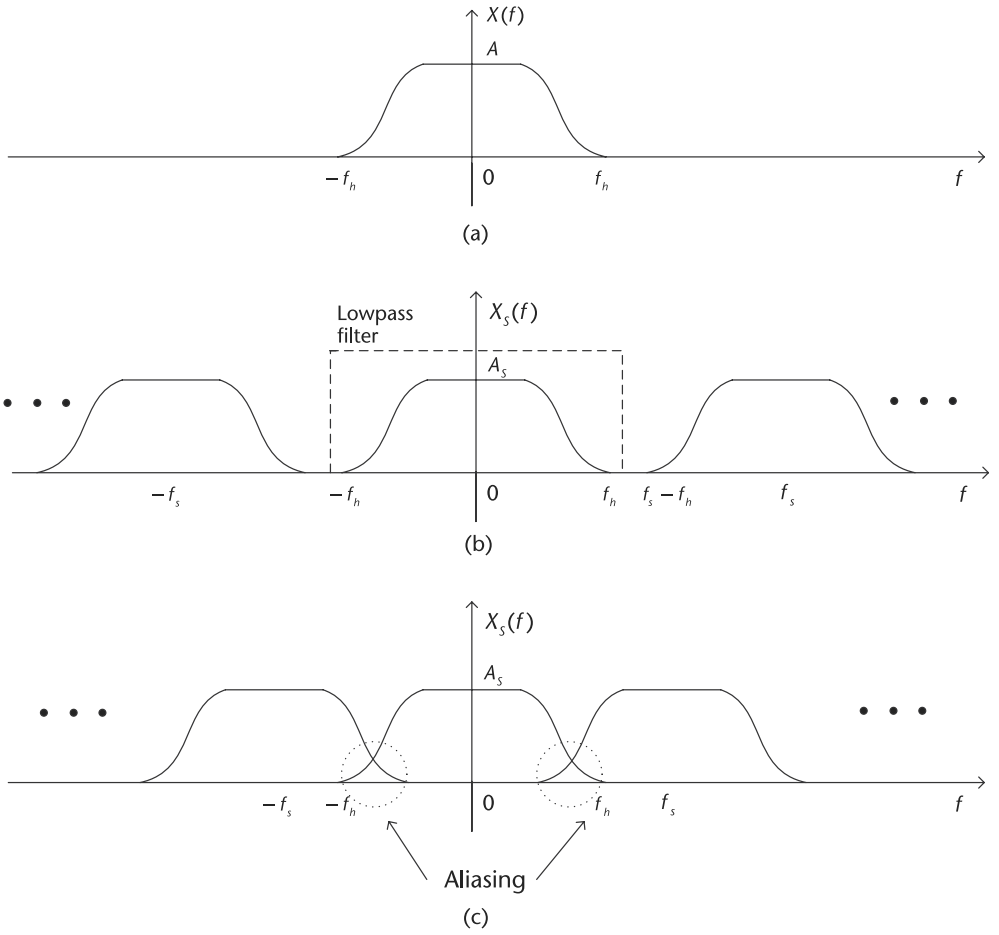


Figure 2.7 The spectrum of original signal $x(t)$ and the sampled signal $x_s(t)$ in the frequency domain. (a) The spectrum of original continuous-time signal $x(t)$, with bandwidth $-f_h$ to f_h, and amplitude A. (b) The spectrum of the digitally sampled signal $x_s(t)$, $f_s > f_h$ which satisfies Nyquist sampling theorem. (c) The spectrum of the digitally sampled signal $x_s(t)$, $f_s < f_h$ which does not satisfies Nyquist sampling theorem and has aliasing.

When a signal is undersampled, its spectrum has overlapping spectral tails, or images, where $X_s(f)$ no longer has complete information about the spectrum and it is no longer possible to recover $x(t)$ from the sampled signal. In this case, the tailing spectrum does not go to zero, but is folded back onto the apparent spectrum. This inversion of the tail is called spectral folding or aliasing, as shown in Figure 2.7(c) [10].

Hands-On MATLAB Example: Let us now explain via computer simulation how the Nyquist criteria requires that the sampling frequency be at least twice the highest frequency contained in the signal or information about the signal will be lost. Furthermore, in Section 2.5.1, the phenomena known as *aliasing* will occur and the frequency will be folded back into the first Nyquist band. In order to describe the implications of aliasing, we can investigate things in the time domain.

Consider the case of a time domain representation of a single tone sinewave sampled as shown in Figure 2.8(a). In this example, the sampling frequency (f_s) is

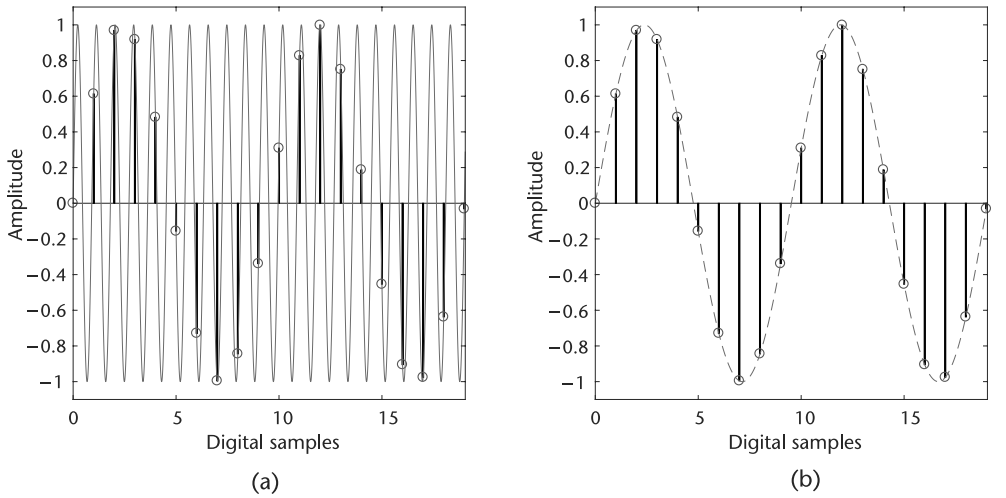


Figure 2.8 Aliasing in the time domain. Digital samples are the same in both figures. (a) Analog input (F_A) solid line; digital sample data (circles) at (f_s), and (b) digital reconstruction dashed line; digital sample data (circles) at (f_s).

not at least two times the analog input frequeny (F_A), but actually slightly more than f_s. Therefore, the Nyquist criteria is violated by definition. Notice that the pattern of the actual samples produces an aliased sinewave at a lower frequency, as shown in Figure 2.8(b).

Code 2.2 can create similar figures as shown in Figure 2.8(a) and Figure 2.8(b), and may be helpful to better understand how aliasing works by manipulating `Fs` and `Fa`. Figures 2.7 and 2.25 demonstrate the same effect in the frequency domain.

Code 2.2 Time domain aliasing: **`nyquist.m`**

```
Fs = 1000;      % Sample rate (Hz)
Fa = 1105;      % Input Frequency (Hz)
% Determine Nyquist zones
zone = 1 + floor(Fa / (Fs/2));
alias = mod(Fa, Fs);
if ~mod(zone,2) % 2nd, 4th, 6th, ... Nyquist Zone
    % Its not really a negative amplitude, but it is 180 degrees out
    % of phase, which makes it harder to see on the time domain side,
    % so we cheat to make the graphs look better.
    alias = -(Fs - alias)/Fs;
else            % 3rd, 5th, 7th, ... Nyquist Zone
    alias = (alias)/Fs;
end

% Create the analog/time domain and digital sampling vectors
N = 2*1/abs(alias) + 1;        % Number of Digital samples
points = 256;                  % Analog points between digital samples
analogIndexes = 0:1/points:N-1;
samplingIndexes = 1:points:length(analogIndexes);
wave = sin(2*pi*Fa/Fs*analogIndexes);
```

2.2.4 Nyquist Zones

The Nyquist bandwidth itself is defined to be the frequency spectrum from DC to $\frac{f_s}{2}$. However, the frequency spectrum is divided into an infinite number of Nyquist zones, each having a width equal to 0.5 f_s as shown in Figure 2.9. The frequency spectrum does not just end because you are not interested in those frequencies.

This implies that some filtering ahead of the sampler (or ADC) is required to remove frequency components that are outside the Nyquist bandwidth, but whose aliased components fall inside it. The filter performance will depend on how close the out-of-band signal is to $\frac{f_s}{2}$ and the amount of attenuation required. It is important to note that with no input filtering at the input of the ideal sampler (or ADC), any frequency component (either signal or noise) that falls outside the Nyquist bandwidth in any Nyquist zone will be aliased back into the first Nyquist zone. For this reason, an analog antialiasing filter is used in almost all sampling ADC applications to remove these unwanted signals.

How do you think the relationship changes between the measured frequency and the absolute frequency, as it goes up into the third or fourth or higher Nyquist zones as shown in Figure 2.9? See if you can confirm your hypothesis by modifying Code 2.2 to plot absolute frequency on the *x*-axis, and measured frequency on the *y*-axis.

2.2.5 Sample Rate Conversion

In real-world applications, we often would like to lower the sampling rate because it reduces storage and computation requirements. In many cases we prefer a higher sampling rate because it preserves fidelity. Sampling rate conversion is a general term for the process of changing the time interval between the adjacent elements in a sequence consisting of samples of a continuous-time function [10].

Decimation: The process of lowering the sampling rate is called *decimation*, which is achieved by ignoring all but every Dth sample. In time domain, it can be defined as

$$y[n] = x[nD], \quad D = 1, 2, 3, ..., \tag{2.13}$$

where $x[n]$ is the original signal, $y[n]$ is the decimated signal, and D is the decimation rate. According to (2.13), the sampling rates of the original signal and the decimated

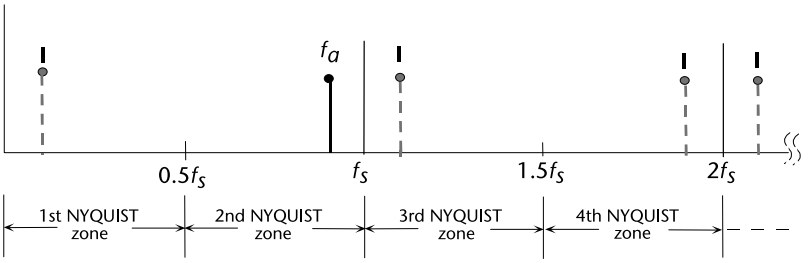


Figure 2.9 Analog signal f_a sampled at f_s has images (aliases) at $\pm kF_s \pm F_a, k = 1, 2, 3, ...$

signal can be expressed as

$$F_y = \frac{F_x}{D}, \tag{2.14}$$

where F_x is the sampling rates of the original signal, and F_y is the sampling rates of the decimated signal.

Since the frequency variables in radians, ω_x and ω_y, can be related to sampling rate, F_x and F_y, by

$$\omega_x = 2\pi FT_x = \frac{2\pi F}{F_x}, \tag{2.15}$$

and

$$\omega_y = 2\pi FT_y = \frac{2\pi F}{F_y}, \tag{2.16}$$

it follows from the distributive property that ω_x and ω_y are related by

$$\omega_y = D\omega_x, \tag{2.17}$$

which means that the frequency range of ω_x is stretched into the corresponding frequency range of ω_y by a factor of D.

In order to avoid aliasing of the decimated sequence $y[n]$, it is required that $0 \leq |\omega_y| \leq \pi$. Based on (2.17), it implies that the spectrum of the original sequence should satisfy $0 \leq |\omega_x| \leq \frac{\pi}{D}$. Therefore, in reality, decimation is usually a two-step process, consisting of a lowpass antialiasing filter and a downsampler, as shown in Figure 2.10. The lowpass antialiasing filter is used to constrain the bandwidth of the input signal to the downsampler $x[n]$ to be $0 \leq |\omega_x| \leq \frac{\pi}{D}$.

In frequency domain, the spectrum of the decimated signal, $y[n]$, can be expressed as [1]

$$Y(\omega_y) = \frac{1}{D}\sum_{k=0}^{D-1} H_D\left(\frac{\omega_y - 2\pi k}{D}\right) S\left(\frac{\omega_y - 2\pi k}{D}\right), \tag{2.18}$$

where $S(\omega)$ is the spectrum of the input signal $s[n]$, and $H_D(\omega)$ is the frequency response of the lowpass filter $h_D[n]$. With a properly designed filter $H_D(\omega)$, the aliasing is eliminated, and consequently, all but the first $k = 0$ term in (2.18) vanish [1]. Hence, (2.18) becomes

$$Y(\omega_y) = \frac{1}{D}H_D\left(\frac{\omega_y}{D}\right) S\left(\frac{\omega_y}{D}\right) = \frac{1}{D}S\left(\frac{\omega_y}{D}\right), \tag{2.19}$$

for $0 \leq |\omega_y| \leq \pi$. The spectra for the sequence $x[n]$ and $y[n]$ are illustrated in Figure 2.11, where the frequency range of the intermediate signal is $0 \leq |\omega_x| \leq \frac{\pi}{D}$, and the frequency range of the decimated signal is $0 \leq |\omega_y| \leq \pi$.

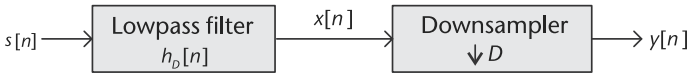


Figure 2.10 The structure of decimation, consisting of a lowpass antialiasing filter and a downsampler.

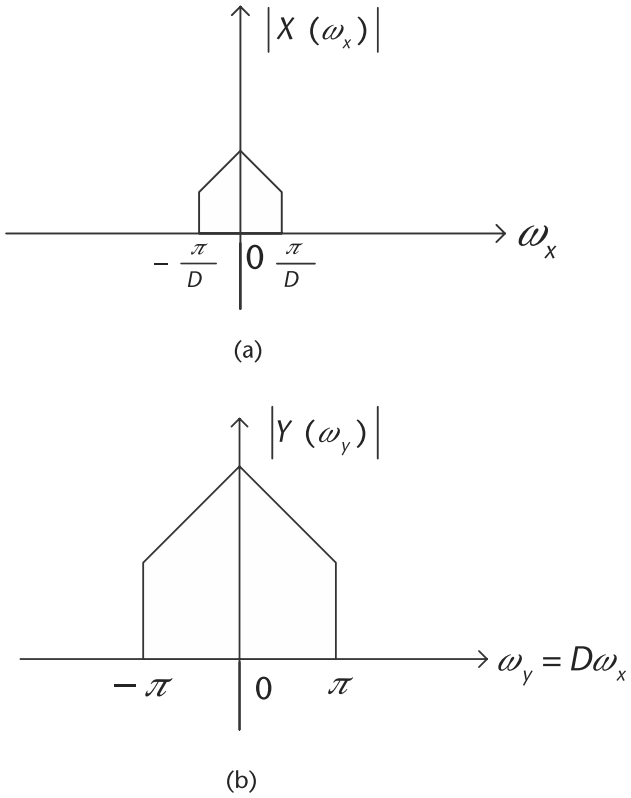


Figure 2.11 The spectra for the sequence $x[n]$ and $y[n]$, where the frequency range of ω_x is stretched into the corresponding frequency range of ω_y by a factor of D. (a) Spectrum of the intermediate sequence, and (b) spectrum of the decimated sequence.

Interpolation: The process of increasing the sampling rate is called *interpolation*, which can be accomplished by interpolating (stuffing zeros) $I - 1$ new samples between successive values of signal. In time domain, it can be defined as

$$y[n] = \begin{cases} x[n/I] & n = 0, \pm I, \pm 2I, \ldots \\ 0 & \text{otherwise} \end{cases}, \quad I = 1, 2, 3, \ldots, \tag{2.20}$$

where $x[n]$ is the original signal, $y[n]$ is the interpolated signal, and I is the interpolation rate.

According to (2.20), the sampling rates of the original signal and the interpolated signal can be expressed as

$$F_y = IF_x, \tag{2.21}$$

where F_x is the sampling rates of the original signal, and F_y is the sampling rates of the interpolated signal. Since (2.15) and (2.16) also hold here, it follows that ω_x and ω_y are related by

$$\omega_y = \frac{\omega_x}{I}, \tag{2.22}$$

which means that the frequency range of ω_x is compressed into the corresponding frequency range of ω_y by a factor of I. Therefore, after the interpolation, there will be I replicas of the spectrum of $x[n]$, where each replica occupies a bandwidth of $\frac{\pi}{I}$.

Since only the frequency components of $y[n]$ in the range $0 \leq |\omega_y| \leq \frac{\pi}{I}$ are unique (i.e., all the other replicas are the same as this one), the images of $Y(\omega)$ above $\omega_y = \frac{\pi}{I}$ should be rejected by passing it through a lowpass filter with the following frequency response:

$$H_I(\omega_y) = \begin{cases} C & 0 \leq |\omega_y| \leq \frac{\pi}{I} \\ 0 & \text{otherwise} \end{cases}, \tag{2.23}$$

where C is a scale factor.

Therefore, in reality, interpolation is also a two-step process, consisting of an upsampler and a lowpass filter, as shown in Figure 2.12. The spectrum of the output signal $z[n]$ is

$$Z(\omega_z) = \begin{cases} CX(\omega_z I) & 0 \leq |\omega_z| \leq \frac{\pi}{I} \\ 0 & \text{otherwise} \end{cases}, \tag{2.24}$$

where $X(\omega)$ is the spectrum of the output signal $x[n]$.

The spectra for the sequence $x[n]$, $y[n]$ and $z[n]$ are illustrated in Figure 2.13, where the frequency range of the original signal is $0 \leq |\omega_x| \leq \pi$, and the frequency range of the decimated signal is $0 \leq |\omega_z| \leq \frac{\pi}{I}$.

Hands-On MATLAB Example: Now let us experiment with decimation and interpolation using MATLAB. The following example will manipulate two types of signals: a continuous wave (CW) signal (a sine wave) and a random signal. Furthermore, we will visualize the effects of upsampling and downsampling in the the time and frequency domains. We begin by generating these signals using MATLAB Code 2.3, and then pass the data through a lowpass filter in Code 2.4 to band-limit them. Using these band-limited versions we will observe the effects of correct and incorrect up and downsampling in the time and frequency domains.

When left unfiltered in Figure 2.14(a) and Figure 2.14(e), the sine wave signal mirrors a discrete version of a sine wave function. On the other hand in Figure 2.14(b) and Figure 2.14(f), the random binary signal consist of a string of random ones and negative ones.

Using the least-squares linear-phase FIR filter design or MATLAB's `firls` function, we are able to quickly generate FIR filter coefficients where the cut-off frequency is approximately at 0.21π.

After passing the data through Code 2.4, the resulting filtered discrete time signals (sine and random binary) represented in both the time and frequency domains are presented in Figure 2.14(c) and Figure 2.14(g). The differences between the original sine wave shown in Figure 2.14(e) and the band-limited sine wave in Figure 2.14(g) are negligible since the frequency of the sine wave is less than the low pass/band-limiting filter in Code 2.4. The differences of the random data shown in shown in Figure 2.14(b) and the band-limited in Figure 2.14(d) can been seen easily, and the reduction of bandwidth in the frequency domain show in Figure 2.14(h) is very noticeable compared to the orginal in Figure 2.14(f). In the time domain, we

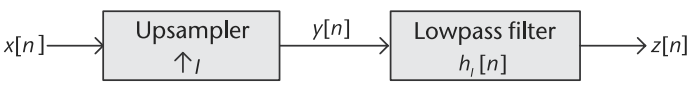


Figure 2.12 The structure of interpolation, consisting of an upsampler and a lowpass filter.

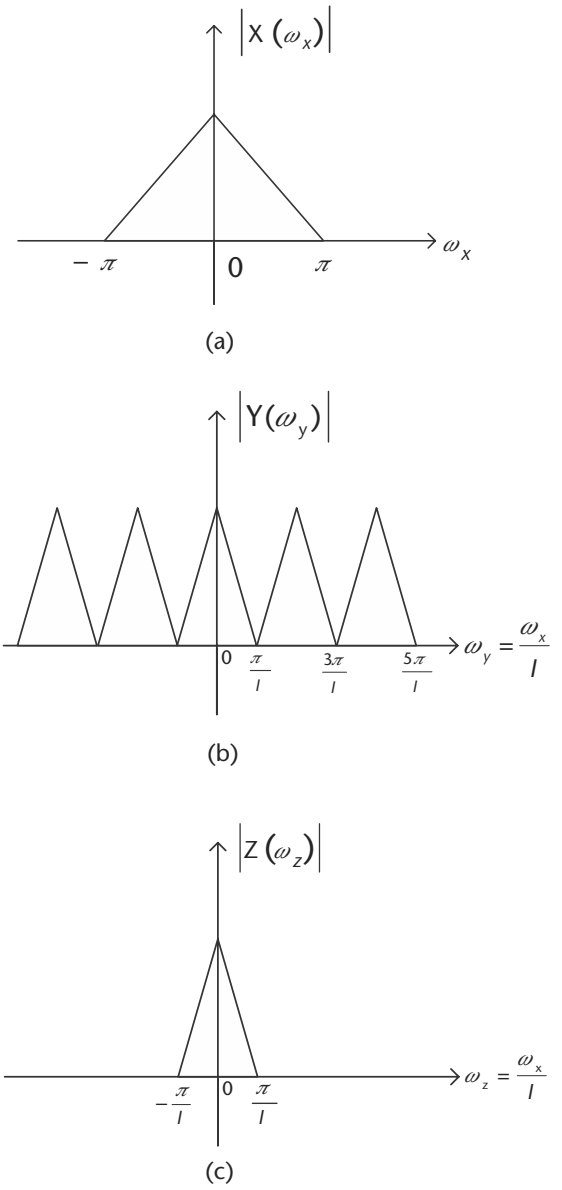


Figure 2.13 The spectra for the sequence $x[n]$, $y[n]$ and $z[n]$, where the frequency range of ω_x is compressed into the corresponding frequency range of ω_y by a factor of I. (a) Spectrum of the original sequence, (b) spectrum of the intermediate sequence, and (c) spectrum of the interpolated sequence.

Code 2.3 Create data sets: **up-down-sample.m**

```
19 % Create deterministic and stochastic digital data streams
20 n = 0:1/Fs1:100-(1/Fs1);                % Time index vector
21 sin_wave = sin(5*n*2*pi);               % Generation of sinusoidal
                                           % signal
22 random = 2*round(rand(1,length(n)))-1;  % Random string of +1 and
                                           % -1 values
```

Code 2.4 Create and apply lowpass filter to band-limit signal: **up-down-sample.m**

```
44 % Create lowpass filter and apply it to both data streams
45 % b = firls(n,f,a),
46 %    n is the FIR filter order
47 %    f is a vector of pairs of frequency points,
48 %    a is a vector containing the desired amplitude at the points in f
49 coeffs1 = firls(taps,[0 0.2 0.22 1],[1 1 0 0]);   % FIR filter
                                                     % coefficients
50 sin_bwlimited = filter(coeffs1,1,sin_wave);
51 random_bwlimited = filter(coeffs1,1,random);
```

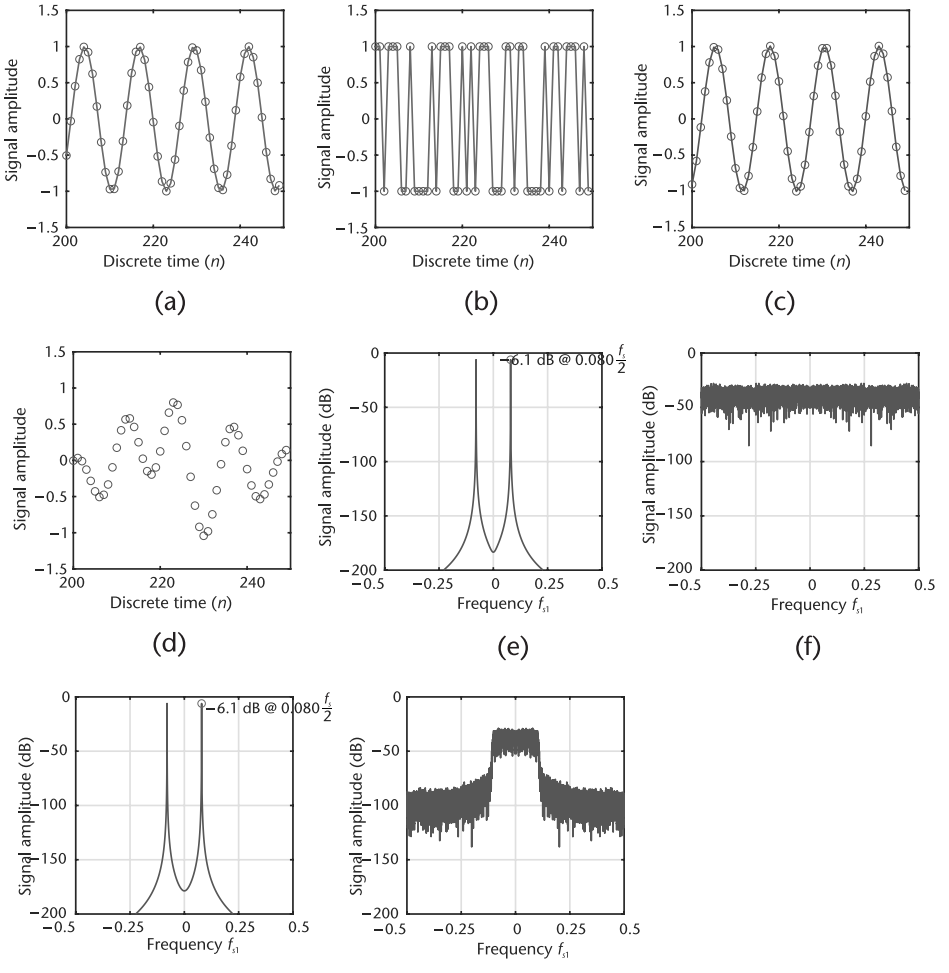


Figure 2.14 Sine and random data, created by Code 2.3, then bandlimited by Code 2.4. (a) Original sine wave: time domain, (b) original random data: time domain, (c) band-limited sine wave: time domain, (d) band-limited random data: time domain, (e) original sine wave: Fourier domain, (f) original random data: Fourier domain, (g) band-limited sine wave: Fourier domain, and (h) band-limited random data: Fourier domain.

examine the signal starting at a sample large enough to ignore the initial artifacts of the lowpass filter, which is why we do not look at things when $t = 0$. In the frequency domain, the lowpass filtering effectively limits the bandwidth of these signals, which will make them more clearly visible in the subsequent exercises in this section.

With the filtered sine wave and filtered random binary signals, we now want to explore the impact of upsampling these signals by observing the resulting outcomes in the frequency domain. Using Code 2.5, we can use the function `upsample` to take the filtered signals and upsample them by a factor of N (in this case, `N=5`). It should be noted that all `upsample` simply inserts `N-1` zeros between each original input sample. This function is quite different than the `interp` interpolation function, which combines upsampling and filtering operations.

The impact of upsampling can be clearly observed from the before-and-after frequency responses of the signals. Specifically, we expect that the frequency responses should be compressed by the upsampling factor of N, and also contain N periodic replicas across the original frequency band. Referring to Figures 2.15(a) and 2.15(b), we can observe this phenomena after upsampling our filtered sine wave and random binary signals, which are upsampled by a factor of `N=5` from Code 2.5.

Notice the difference between Figure 2.14(g) and Figure 2.15(e) for the filtered sine wave signal, or between Figure 2.14(h) and Figure 2.15(f) for the filtered random data signal. In both cases, we can readily see that the spectra of these signals have been compressed by the upsampling factor, and that we now have periodic replicas across frequency. It is also obvious that the amplitude as changed, as the average signal amplitude has been effected by this insertion of zeros. Although the literature may discuss this effect as *compression*, it is interesting to note that the actual signal has not changed in frequency. However, it appears in a different location with respect to the $-\frac{f_s}{2}$ to $\frac{f_s}{2}$ scale. Therefore, it is important to remember that f_s in both figures are not the same (differs by a factor of 5), which is a slight abuse of notation.

Now that we see how upsampling can compress the frequency responses of signals and make periodic replicas across the spectrum, let us now explore how downsampling these signals can either result in frequency expansion without any aliasing or with substantial amounts of aliasing. Recall that the downsampling process involves the periodic removal of $M - 1$ samples, which results in a frequency response that expands by a factor of M. This frequency expansion can be problematic when the frequency spectra begins to overlap with its periodic replicas

Code 2.5 Upsample the signals: **`up-down-sample.m`**

```
73 % y = upsample(x,n)
74 %     increases the sampling rate of x by inserting (n  1) zeros
75 %     between samples.
76 N = 5;
77 sin_up = upsample(sin_bwlimited,N);
78 random_up = upsample(random_bwlimited,N);
```

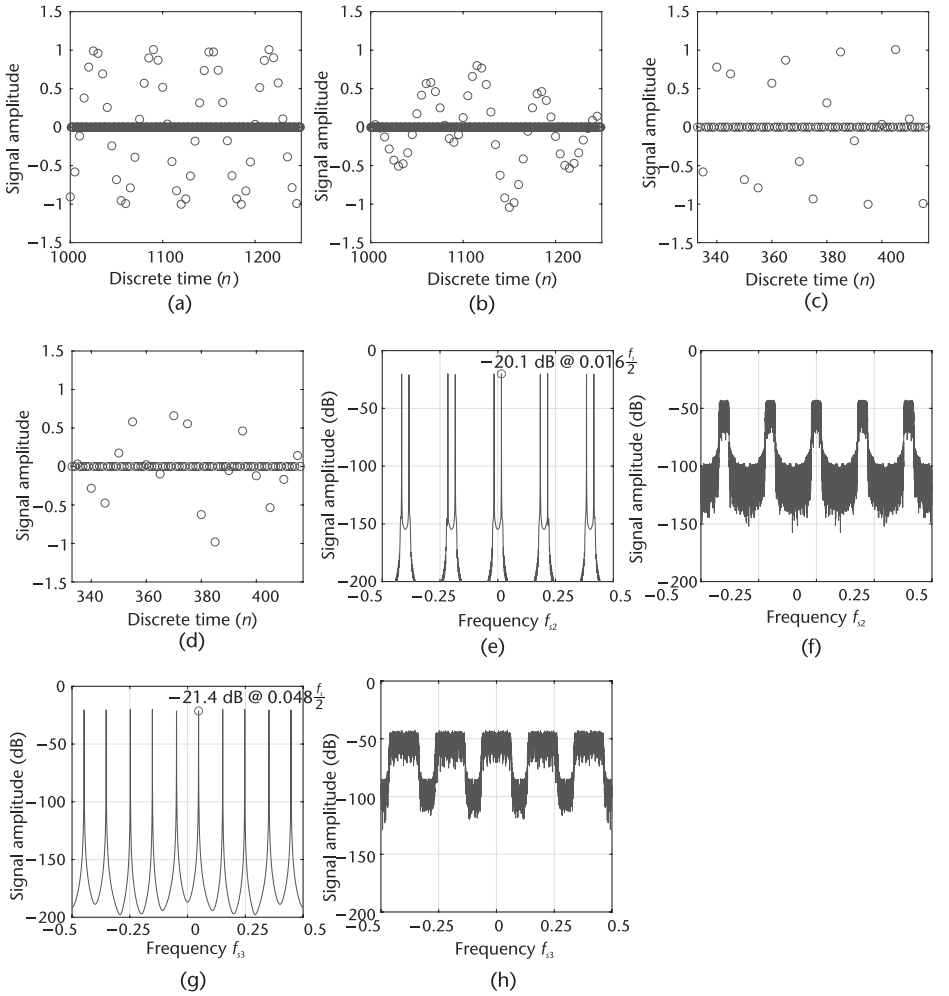


Figure 2.15 Upsampled data: Code 2.5, then processed by a downsampler: Code 2.6. (a) Upsampled, band-limited, sine wave: time domain, (b) band-limited random data: Fourier domain, (c) incorrect, upsampled, then downsampled, band-limited, sine wave: Fourier domain, (d) incorrect, upsampled, then downsampled, band-limited, random data: Fourier domain, (e) upsampled, band-limited, sine wave: Fourier domain, (f) band-limited random data: Fourier domain, (g) incorrect, upsampled, then downsampled, band-limited, sine wave: Fourier domain, and (h) incorrect, upsampled, and then downsampled, band-limited random data: Fourier domain.

centered at every multiple of 2π (sampled signals have spectra that repeat every 2π). This overlapping results in the aliasing of the signal, thus distorting it and making it very difficult to recover at the receiver. Code 2.6 will downsample the band-limited signals and Code 2.5 will upsample the created signals by a factor of `M=3`. Note that the MATLAB function `downsample` is different than the MATLAB function `decimate`, where the latter combines the downsampling and filtering operations to perform signal decimation.

In Code 2.6, which produces Figure 2.15(h) and Figure 2.15(g), we observed the incorrect case in which downsampling without filtering out one of the periodic replicas caused aliasing. Next, in Code 2.7 we will perform the necessary filtering to remove the periodic replicas before downsampling. In this code we apply an amplitude correct of the upsampling rate to compensate for these operations.

Code 2.6 Downsample the signals: **up-down-sample.m**

```
% Attempt to downsampling by M without filtering
% This is incorrect, but is instructive to show what artifacts occur
M = 3;
sin_up_down = downsample(sin_up,M);
random_up_down = downsample(random_up,M);
```

Code 2.7 Lowpass filter, then downsample the data: **up-down-sample.m**

```
% Lowpass filtering of baseband periodic replica followed by
    % downsampling
% (correct approach)
coeffs2 = firls(taps,[0 0.15 0.25 1],[N N 0 0]); % FIR filter
                                                  % coefficients
sin_up_filtered = filter(coeffs2,1,sin_up);
sin_up_filtered_down = downsample(sin_up_filtered,M);
random_up_filtered = filter(coeffs2,1,random_up);
random_up_filtered_down = downsample(random_up_filtered,M);
```

When the signal is upsampled, the periodic replicas generated by this process span across the entire $-\pi$ to π radians (or $-\frac{f_s}{2}$ to $\frac{f_s}{2}$) of spectra. Without adequate filtering of these replicas before downsampling, these periodic replicas will begin to expand into other periodic replicas and result in aliasing. This phenomena is illustrated in Figure 2.15(d) and Figure 2.15(h), where we downsample the upsampled filtered sine wave and random binary signals previously described. For the downsampling of the upsampled filtered sine wave signal, we observe aliasing in Figure 2.15(g) with the spectra from other integers of Nyquist bands. When performed properly, we should only have one replica that is expanded by the downsampling factor, as observed in Figure 2.16(g) and Figure 2.17(b)

Since it is difficult to observe the spectra expansion of the sine wave signal because it is narrow, let us also observe the frequency responses of the random binary signal shown in Figures 2.15(h) and 2.17(b). In these figures, it is clearly evident that aliasing is occurring since the replicas from the upsampling process were not filtered out. On the other hand, when the upsampling replicas are filtered, we observe a clean, unaliased, frequency response of the downsampled signal shown in Figure 2.17(b).

We can do a final comparison of the signals in the time domain and see that the shape of the time domain signal is nearly exactly the same in amplitude and absolute time (seconds). It just has been sample rate converted from f_s to $\frac{5}{3}f_s$ adding more samples to the same signal.

2.3 Signal Representation

Understanding how a signal is represented can greatly enhance one's ability to analyze and design digital communication systems. We need multiple convenient numeric mathematical frameworks to represent actual RF, baseband, and noise

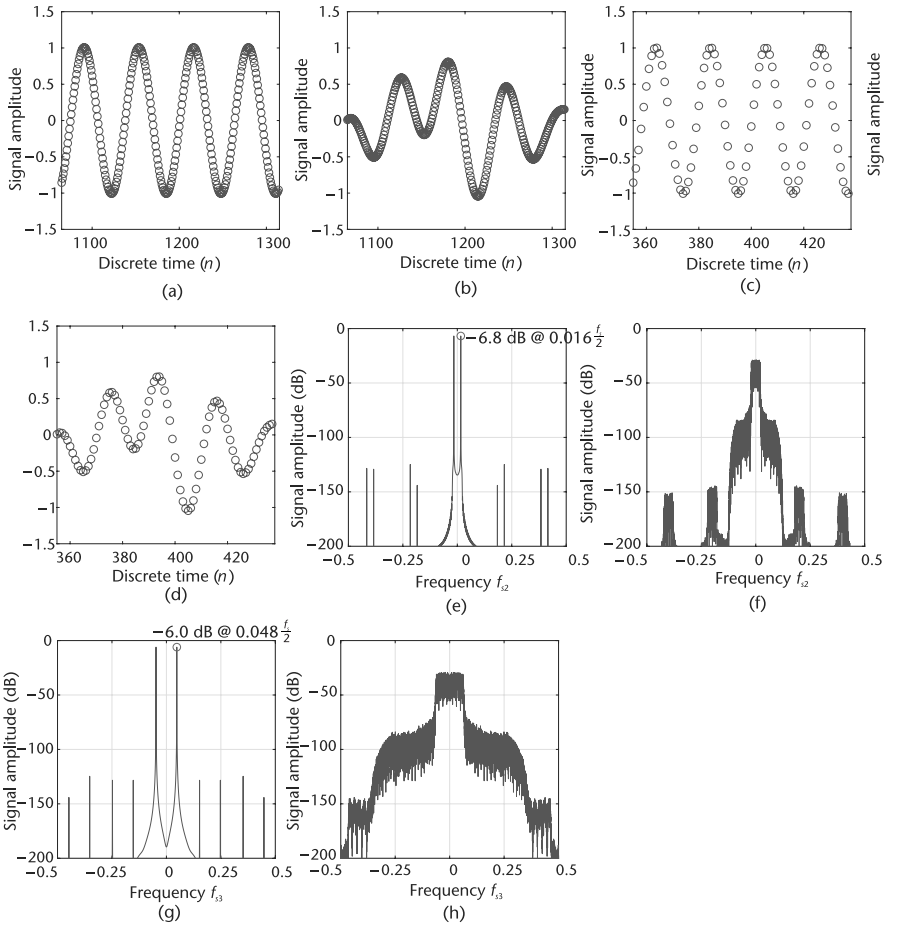


Figure 2.16 Upsampled by $N = 5$, then filtered band-limited waveforms produced by Code 2.7. You can still see the replicas of the signals, but the filter has suppressed it to very low levels. The filtering has corrected the amplitude loss from the upsampling. (a) Upsampled, then filtered band-limited, sine wave: Fourier domain, (b) upsampled, filtered, then downsampled, band-limited sine wave: Fourier domain, (c) upsampled, then filtered band-limited, sine wave: Fourier domain, (d) upsampled, filtered, then downsampled, band-limited sine wave: Fourier domain, (e) upsampled, then filtered band-limited, random data: Fourier domain, (f) upsampled, then filtered band-limited, random data: Fourier domain, (g) upsampled, then filtered band-limited, random Data: Fourier domain, and (h) upsampled, filtered, then band-limited random data: Fourier domain.

signals. We usually have two: envelope/phase and in-phase/quadrature, and both can be expressed in the time and Fourier domains.

2.3.1 Frequency Conversion

To understand how we can move signals from baseband to RF and from RF to baseband, let us look more closely at modulators and demodulators. For example, in Figure 2.18 we see a very classical quadrature modulator. The ADL5375 accepts two differential baseband inputs and a single-ended LO, which generates a single-ended output. The LO interface generates two internal LO signals in quadrature (90° out of phase) and these signals are used to drive the mixers, which simply multiply the LO signals with the input.

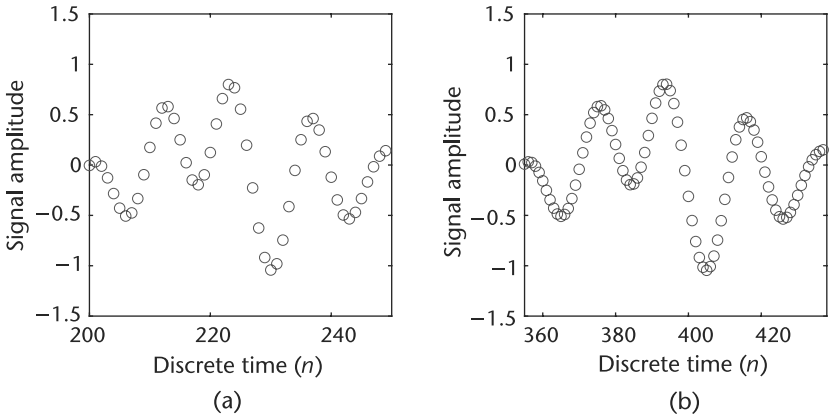


Figure 2.17 Upsampled by $N = 5$, then filtered, and then downsampled by $M = 3$, band-limited random data. A phase shift (time shift) occurs from the filtering. (a) Original band-limited random data time domain at f_s. (b) Upsampled, filtered, and the band-limited random data: time domain at $\frac{5}{3}f_s$.

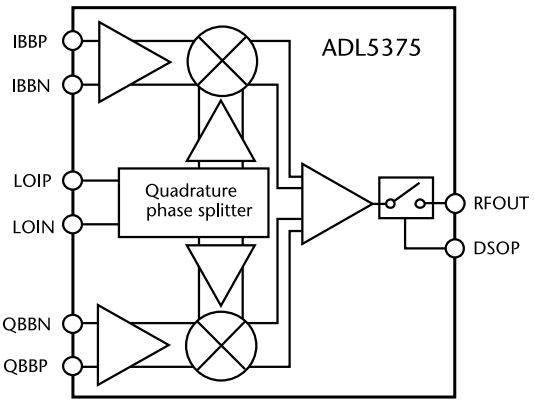


Figure 2.18 ADL5375 broadband quadrature modular with range from 400 MHz to 6 GHz.

Mathematically, this mixing process will take the two inputs *IBB* and *QBB*, which we will denote by $I(t)$ and $Q(t)$, multiply them by our LO at frequency ω_c, and add the resulting signal to form our transmitted signal $r(t)$. The LO used to multiply $Q(t)$ is phase shifted by 90° degree to make it orthogonal with the multiplication of the $I(t)$ signal. Consequently, this yields in the following equation:

$$r(t) = I(t)cos(\omega_c t) - Q(t)sin(\omega_c t). \tag{2.25}$$

The LO frequency is denote as ω_c since it will be typically called the carrier frequency, which exploits the phase relationship between the in-phase ($I(t)cos(\omega_c t)$) and quadrature ($Q(t)sin(\omega_c t)$) components. Therefore, the transmitted signal will contain both components but will appear as a single sinusoid.

At the receiver, we will translate or down mix $r(t)$ back into our in-phase and quadrature baseband signals through a similar process but in reverse. By applying the same LO with a second phase-shifted component to $r(t)$ with a lowpass filter,

we arrive at

$$I_r(t) = LPF\{r(t)cos(\omega_c t)\} = LPF\{(I(t)cos(\omega_c t) - Q(t)sin(\omega_c t))cos(\omega_c t)\} = \frac{I(t)}{2}, \tag{2.26}$$

$$Q_r(t) = LPF\{r(t)sin(\omega_c t)\} = LPF\{(-I(t)cos(\omega_c t) + Q(t)sin(\omega_c t))sin(\omega_c t)\} = \frac{Q(t)}{2}. \tag{2.27}$$

In practice, there will be some phase difference between the transmitter LO and receiver LO, which can cause rotation to $r(t)$. However, the phase relation between $I(t)$ and $Q(t)$ will alway be maintained.

2.3.2 Imaginary Signals

Discussing signals as *quadrature* or *complex* signal is taking advantage of the mathematical constructs that Euler and others have created to make analysis easier. We try to refer to signals as in-phase (I) and quadrature (Q) since that is actually pedantically correct. As described in Section 2.3.1, the in-phase (I) refers to the signal that is in the same phase as the local oscillator, and the quadrature (Q) refers to the part of the signal that is in phase with the LO shifted by 90°.

It is convenient to describe this as I being *real* and Q being *imaginary* since it enables many mathematical techniques but at the end of the day is just a construct. A prime example of this convience is frequency translation, which is performed by the mixer. We start from the Euler relation of

$$e^{jx} = \cos(x) + j\sin(x), \tag{2.28}$$

where we can define x as target frequency plus time. Taking the conventions from Section 2.3.1 but redefining $I(t)$ and $Q(t)$ as real and imaginary, we arrive at

$$y(t) = I(t) + jQ(t). \tag{2.29}$$

Now, if we assume $y(t)$ is a CW tone at frequency ω_a for illustration purposes, $y(t)$ becomes

$$y(t) = cos(\omega_a t) + jsin(\omega_a t). \tag{2.30}$$

Now applying (2.28) we can frequency shift $y(t)$ by the desired frequency ω_c:

$$\begin{aligned} y(t)e^{j\omega_c t} &= \big(I(t)cos(\omega_c t) - Q(t)sin(\omega_c t)\big) + j\big(Q(t)cos(\omega_c t) + I(t)sin(\omega_c t)\big) \\ &= cos((\omega_c + \omega_a)t) + jsin((\omega_c + \omega_a)t). \end{aligned} \tag{2.31}$$

Now our resulting signal will exist at frequency $\omega_a + \omega_c$ through a simple application of Euler's identity.

Let us motivate the usefulness of a complex number representation further from the perspective of hardware. If we consider the mixer itself as in Figure 2.18, the IQ mixer will transmit signals with arbitrary phase and amplitude (within power constraints). This is an example of how we encode information into the data we transmit through differences in phase and amplitude. This is actually accomplished through the relation of our in-phase and quadrature signal, which can be used to create a single sinusoid with arbitrary phase and amplitude. Mathematically, we can

produce a sinusoid with a specific envelope and phase (**A**,ϕ) with two orthogonal components sine and cosine. This relationship is written as

$$\mathbf{A}\sin(\omega t + \phi) = (\mathbf{A}\cos\phi)\sin(\omega t) + (\mathbf{A}\sin\phi)\cos(\omega t). \tag{2.32}$$

Therefore, by just modifying the amplitude of our sine and cosine components over time, we can create the desired waveform from a fixed frequency and phase LO. Alternatively, we can consider others coordinate systems to visualize complex values. Expanding these complex numbers (rectangular coordinates) can be translated in order to be represent a magnitude and angle (polar) or even plotted as a vector.

The in-phase and quadrature sine waves plotted in Figure 2.19 show how things look via a phasor plot as time increases, with the vector indicating magnitude rotating around the axis. One can clearly see the phase shift between I and Q. In the time domain, you can also see the differences between in-phase and magnitude, although phase differences can be a little more subtle to notice. In a Cartesian plane, the signal appears as a rotating circle over time. The phasor plot will always rotate counterclockwise with time, and the Cartesian plot can rotate in either direction depending on if the phase difference between I and Q is positive or negative. While the time domain plot shows things moving as time changes, the phasor plot and Cartesian plane are snapshots in time (at $t = 0$ on the time domain plot). Finally, the frequency domain plot provides the spectrum of the phasor but only communicates magnitude and loses phase information. This happens because we are only plotting the real component of the spectrum. However, comparing the two waveforms in Figures 2.19 and 2.20, changes in the I and Q components (amplitude and phase relationship), will effect the other domains as well.

2.4 Signal Metrics and Visualization

Before an engineer can decide whether a project is done, he or she needs to conduct some form of verification. However, communications systems are complex to evaluate given their integrated nature and depth of transmit and receive chains. As referenced in Section 1.4, communication systems may have a variety of metrics beyond size, weight, power, and cost (SWaP-C). Performance metrics such as bit error rate (BER), data throughput, and distance are also only top-level system

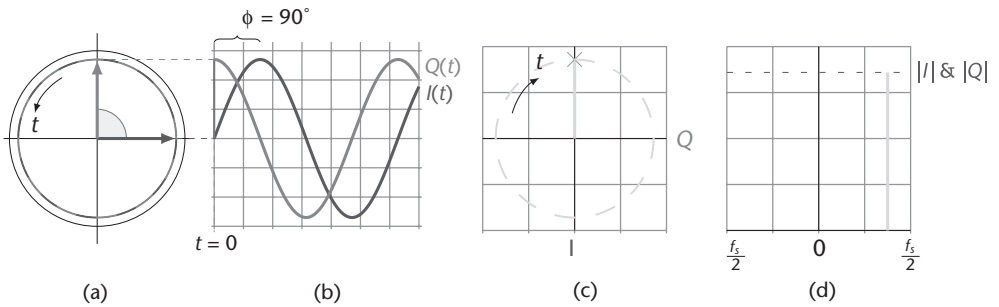


Figure 2.19 Same continuous wave signal, plotted in multiple domains. (a) Phasor *rad*(t), (b) time $x(t) \rightarrow$, (c) Cartesian $(I, Q)(t)$, and (d) frequency $X(\omega)$.

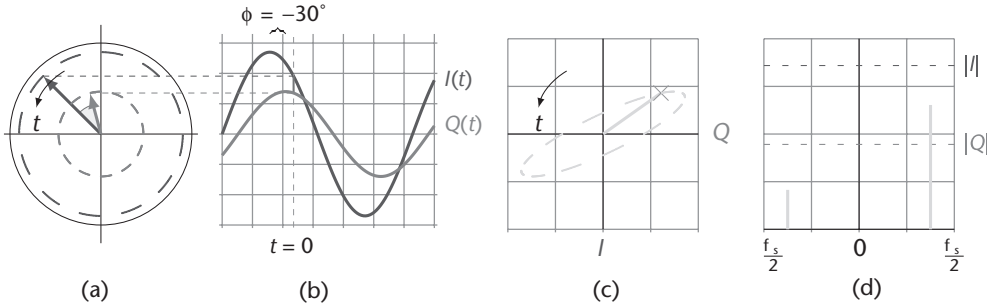


Figure 2.20 Continuous wave signal differences in magnitude and phase cause shifts in various domains. (a) Phasor $rad(t)$, (b) time $x(t) \rightarrow$, (c) Cartesian $(I, Q)(t)$, and (d) frequency $X(\omega)$.

specifications. Just as we have system-level specifications for the entire system, we have specifications and measurement techniques for other subsystems and SDR building blocks. In this way, we know we are not over- or underdesigning specific components. However, trade-offs should always be considered at the system level since upstream modifications can effect downstreaming components or implementations.

System-level specifications are met by ensuring each block in the system will allow those specifications to be met. Making a world-class communications system requires world-class hardware and world-class algorithmic design and implementation. That is the issue with many aspects of engineering—quantitatively determining when something is complete or functional and that it can be connected to the rest of the system. It is never more true than in communications systems that a system is only as good as the weakest link. If you have bolted everything together, and a system-level specification like bit error rate is not meeting your top-level specifications, unless you understand how to measure each part of the communications system, from the RF to the SDR to the algorithmic design, you will be lost and unable to determine what to do next.

Depending on what you are looking at, there are numerous techniques and tools to measure almost everything. Studying communications is not for those who do not want to be rigorous.

2.4.1 SINAD, ENOB, SNR, THD, THD + N, and SFDR

Six popular specifications for quantifying analog dynamic performance are found in Table 2.2 [6]; namely, list out by using and understanding these measurements will help you analyze your designs and make sure you are designing something to be the most robust. Although most device and system manufacturers have adopted the same definitions for these specifications, some exceptions still exist. Due to their importance in comparing devices and systems, it is important not only to understand exactly what is being specified, but the relationships between the specifications.

- *Spurious free dynamic range* (SFDR) is the ratio of the root mean squared (RMS) value of the signal to the rms value of the worst spurious signal regardless of where it falls in the frequency spectrum. The worst spur may or may not be a harmonic of the original signal. This is normally measured over the bandwidth of interest, which is assumed to be the Nyquist bandwidth

Table 2.2 Six Popular Specifications

Property	*Definition*	*MATLAB Function*
SFDR	Spurious free dynamic range	`sfdr`
SINAD	Signal-to-noise-and-distortion ratio	`sinad`
ENOB	Effective number of bits	
SNR	Signal-to-noise ratio	`snr`
THD	Total harmonic distortion	`thd`
THD + N	Total harmonic distortion plus noise	

unless otherwise stated; DC to $f_s/2$ (for baseband), and $-f_s/2$ to $f_s/2$ for complex (RF) converters, which are found on devices like the Pluto SDR. SFDR is an important specification in communications systems because it represents the smallest value of signal that can be distinguished from a large interfering signal (blocker). SFDR is generally plotted as a function of signal amplitude and may be expressed relative to the signal amplitude (dBc) or the ADC full-scale (dBFS) as shown in Figure 2.21. MATLAB's `sfdr` function provides results in terms of dBc. For a signal near full-scale, the peak spectral spur is generally determined by one of the first few harmonics of the fundamental. However, as the signal falls several dB below full-scale, other spurs generally occur that are not direct harmonics of the input signal. This is due to the differential nonlinearity of the systems transfer functions normally dominates at smaller signals. Therefore, SFDR considers all sources of distortion regardless of their origin, and is a useful tool in evaluating various communication systems.

- *Total harmonic distortion* (THD) is the ratio of the rms value of the fundamental signal to the mean value of the root-sum-square of its harmonics (generally, only the first five harmonics are significant). THD of an ADC is also generally specified with the input signal close to full-scale, although it can be specified at any level.
- *Total harmonic distortion plus noise* (THD + N) is the ratio of the rms value of the fundamental signal to the mean value of the root-sum-square of its harmonics plus all noise components (excluding DC). The bandwidth over which the noise is measured must be specified. In the case of an FFT, the bandwidth is DC to $\frac{f_s}{2}$. (If the bandwidth of the measurement is DC to $\frac{f_s}{2}$ (the Nyquist bandwidth), THD + N is equal to SINAD).
- *Signal-to-noise-and-distortion* (SINAD, or S/(N + D) is the ratio of the rms signal amplitude to the mean value of the root-sum-square (rss) of all other spectral components, including harmonics, but excluding DC. SINAD is a good indication of the overall dynamic performance of an analog system because it includes all components that make up noise and distortion. SINAD is often characterized for various input amplitudes and frequencies. For a given input frequency and amplitude, SINAD is equal to THD + N, provided the bandwidth for the noise measurement is the same for both (the Nyquist bandwidth)
- *Signal-to-noise ratio* (SNR, or sometimes called SNR-without-harmonics) is calculated from the FFT data the same as SINAD, except that the signal harmonics are excluded from the calculation, leaving only the noise terms. In practice, it is only necessary to exclude the first five harmonics, since they

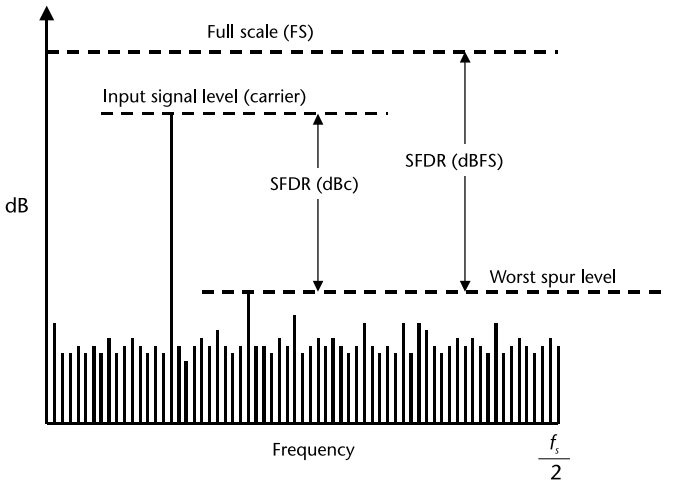


Figure 2.21 Spurious free dynamic range (SFDR) for BW DC to $f_s/2$.

dominate. The SNR plot will degrade at high input frequencies, but generally not as rapidly as SINAD because of the exclusion of the harmonic terms.

2.4.2 Eye Diagram

Although its obvious to state, time domain plots are used to observe changes of an electrical signal over time. Any number of phenomena such as amplitude, frequency, rise time, time interval, distortion, noise floor, and others can be empirically determined, and how these characteristics change over time. In telecommunication, an *eye diagram*, also known as an *eye pattern*, is an time domain display in which a digital data signal from a receiver is repetitively sampled and applied to the vertical input, while the data rate is used to trigger the horizontal sweep [9]. It is called an eye diagram because the pattern looks like a series of eyes between a pair of rails.

Several system performance measures can be derived by analyzing the display, especially the extent of the intersymbol-interference (ISI). As the eye closes, the ISI increases; as the eye opens, the ISI decreases. Furthermore, if the signals are too long, too short, poorly synchronized with the system clock, too high, too low, too noisy, or too slow to change, or have too much undershoot or overshoot, this can be observed from the eye diagram. For example, Figure 2.22 shows a typical eye pattern for the noisy quadrature phase-shift keying (QPSK) signal.

Since the eye diagram conveys and measures many different types of critical data, this can help quantify how well an algorithm or system is working. The two key measurements are the *vertical opening*, which is the distance between BER threshold points, and the *eye height*, which is the minimum distance between eye levels. Larger vertical and horizontal openings in the eye are always better.

Hands-On MATLAB Example: To provide some hands-on experience with eye diagrams, let us use the MATLAB function `eyediagram`, which is a very handy way of visually analyzing a transmission regarding the amount of noise and intersymbol interference present in the signal. Using the pulse shaped signals, we should be able to observe any distortion present within the transmission.

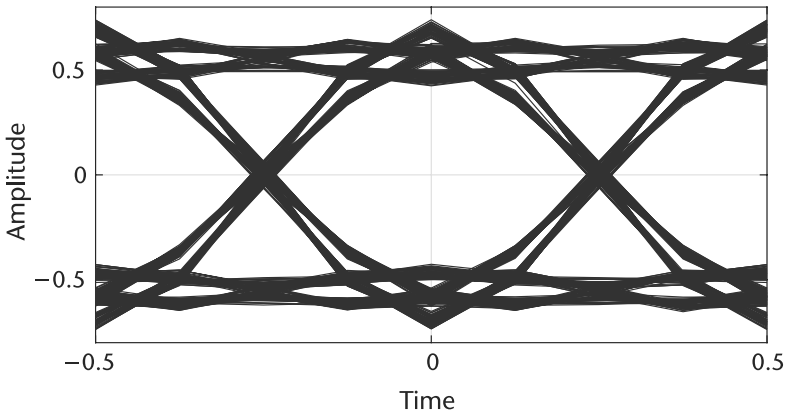


Figure 2.22 A typical eye pattern for the BPSK signal. The width of the opening indicates the time over which sampling for detection might be performed. The optimum sampling time corresponds to the maxmum eye opening, yielding the greatest protection against noise.

From Figure 2.23, we can see that the pulse shaped transmissions do not have any distortion present (we know this in advance since we have intentionally omitted any sort of noise and distortion from the transmission in the first place). When we have eye diagrams such as those shown in Figures 2.23(a) and 2.23(b), we refer to these situations as the eye being open. The prime indicator of having some sort of distortion present within the transmission is when the aperture at time instant 0 is not at its maximum.

Let us explore the scenarios when distortion is present within the transmission and how this translates into an eye diagram. Suppose with take the `y_impulse1` and `y_impulse2` output signals from Code 2.8 and introduce some noise to it. In Code 2.9, we introduced some Gaussian noise using the function `randn`.

We can clearly see in Figure 2.24 the impact of the additional noise on the transmitted signals via the eye diagram. In both Figures 2.24(a) and 2.24(b), it is observed that the eye walls of the diagram are no longer smooth when compared with Figured 2.23(a) and 2.23(b). Although the eye is still open in both cases, we only introduced as small amount of noise into the transmission; the impact of a large amount of noise introduced into the transmission could potential close the eye, meaning the desired sampling instant at time 0 could potentially be corrupted and translate into bit errors. Later on in this book, we will explore how other forms of distortion will affect the aperture of the eye diagram.

2.5 Receive Techniques for SDR

The study of modern communications maintains a great duality when considering both the analog and digital domains. Both domains are manipulated efficiently and with great speed. However, analog signals maintain a perspective of infinite precision but will always contain some degree of randomness due to their very nature. Digital signals, on the other hand, are exact and precisely defined, but are limited by the boundaries of computational complexity and their fundamental foundations. Digital communications must effectively manage both of these world

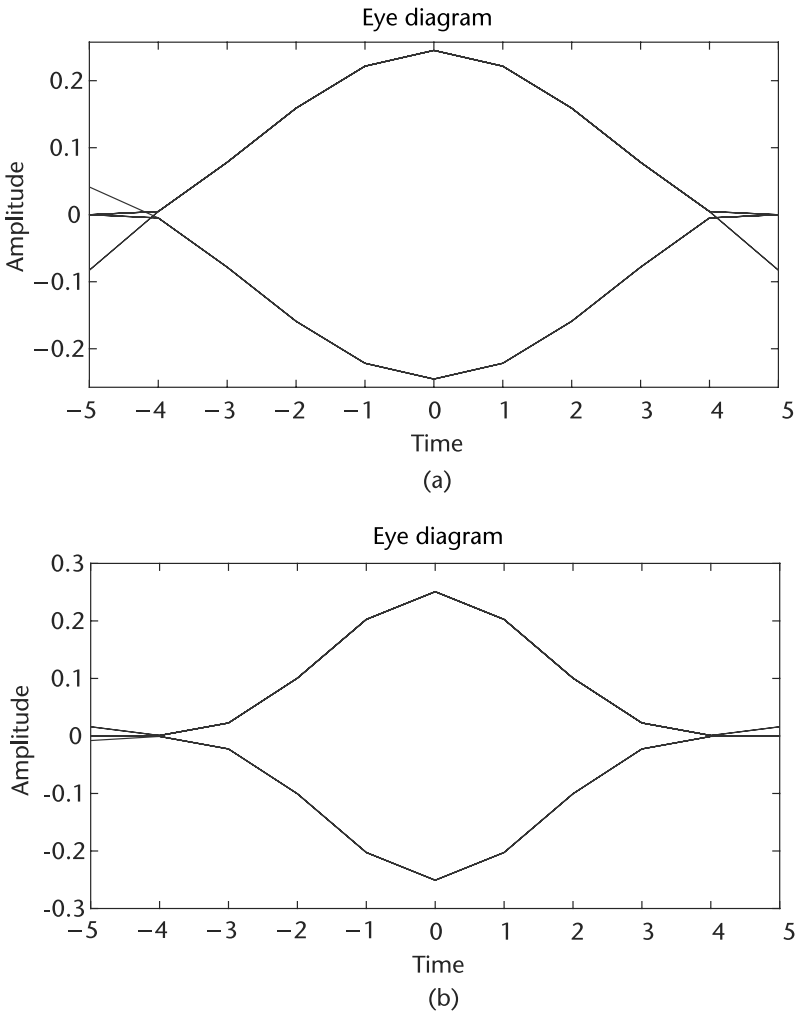


Figure 2.23 Eye diagrams of signals filtered by a system possessing a rectangular frequency response and a triangular frequency response. (a) Rectangular frequency response pulse filtering, and (b) triangular frequency response pulse filtering.

to design robust links between points. Therefore, both domains are highly dependent on one another.

Even in today's world of abundant and cost-effective digital signal processing (DSP) devices, an analog signal is processed, amplified, filtered, and only then converted into binary form by an ADC. The output of the ADC is just a binary representation of the analog signal and is processed on a number of computational units from FPGAs to general purpose CPUs. After processing, the information obtained from the digitized signal, it may be converted back into analog form using a digital-to-analog converter (DAC). Signals physically recovered by a SDR start out as a time-varying electric field, which induces a current in the receiving antenna and resulting in a detectable voltage at the receiver. Transmission by the SDR, on the other hand, are time-varying voltages being applied to an antenna, which causes movement of electrons as an outwardly radiating electric field.

Code 2.8 Eye diagram example: **eye_example.m**

```
% Create impulse train of period L and length len with random +/- one
% values
L = 10;
impulse_num = 100; % Total number of impulses in impulse train
len = L*impulse_num;
temp1 = [2*round(rand(impulse_num,1))-1 zeros(impulse_num,L-1)];
x_impulse = reshape(temp1.',[1,L*impulse_num]);
% Create two transmit filter pulse shapes of order L
% Approximate rectangular frequency response --> approximate
% sinc(x) impulse response
txfilt1 = firls(L,[0 0.24 0.25 1],[4 4 0 0]);
% Approximate triangular frequency response --> approximate
% sinc(x)^2 impulse response
txfilt2 = firls(L,[0 0.5 0.52 1],[4 0 0 0]);
% Pulse shape impulse train
y_impulse1 = filter(txfilt1,1,x_impulse);
y_impulse2 = filter(txfilt2,1,x_impulse);
% eyediagram(x,n,period,offset)
% creates an eye diagram for the signal x, plotting n samples in each
% trace horizontal axis range between -period/2 and period/2.
eyediagram(y_impulse1,L,L,floor(L/2));
eyediagram(y_impulse2,L,L,floor(L/2));
```

Code 2.9 Eye diagram example: **eye_example.m**

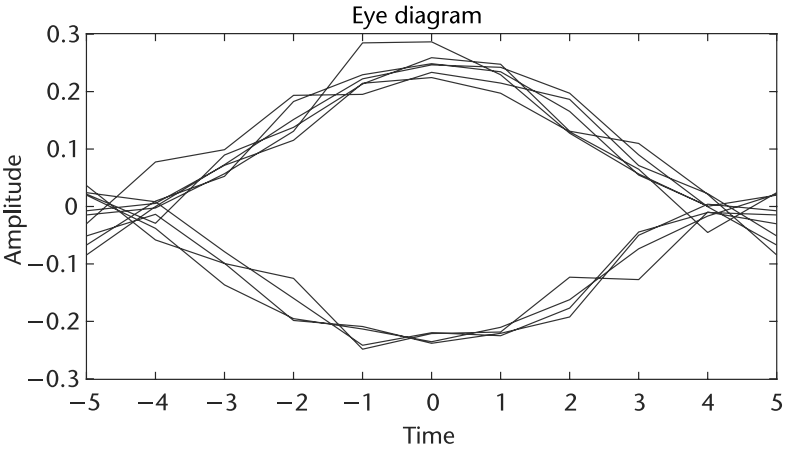
```
eyediagram((y_impulse1+0.1*randn(1,length(y_impulse1))),L,L,floor(L/2));
eyediagram((y_impulse2+0.1*randn(1,length(y_impulse2))),L,L,floor(L/2));
```

2.5.1 Nyquist Zones

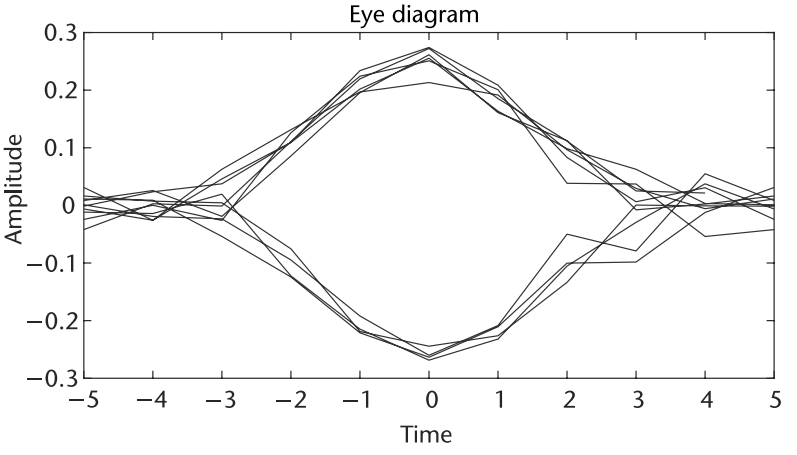
In Section 2.2.3, we considered the case of baseband sampling (i.e., all the signals of interest lie within the first Nyquist zone). Figure 2.25 shows such a case, where the band of sampled signals is limited to the first Nyquist zone and images of the original band of frequencies appear in each of the other Nyquist zones. Consider the case shown in Figure 2.25 B, where the sampled signal band lies entirely within the second Nyquist zone. The process of sampling a signal outside the first Nyquist zone is often referred to as undersampling, or harmonic sampling. Note that the image, which falls in the first Nyquist zone, contains all the information in the original signal with the exception of its original location.

Figure 2.25 shows the sampled signal restricted to the third Nyquist zone. Note that the image that falls into the first Nyquist zone has no frequency reversal. In fact, the sampled signal frequencies may lie in any unique Nyquist zone, and the image falling into the first Nyquist zone is still an accurate representation. At this point we can clearly restate the Nyquist criteria:

A signal must be sampled at a rate equal to or greater than twice its bandwidth in order to preserve all the signal information.



(a)



(b)

Figure 2.24 Eye diagrams of signals filtered by a system possessing a rectangular frequency response and a triangular frequency response with additive white Gaussian noise present. (a) Rectangular frequency response pulse filtering, and (b) triangular frequency response pulse filtering.

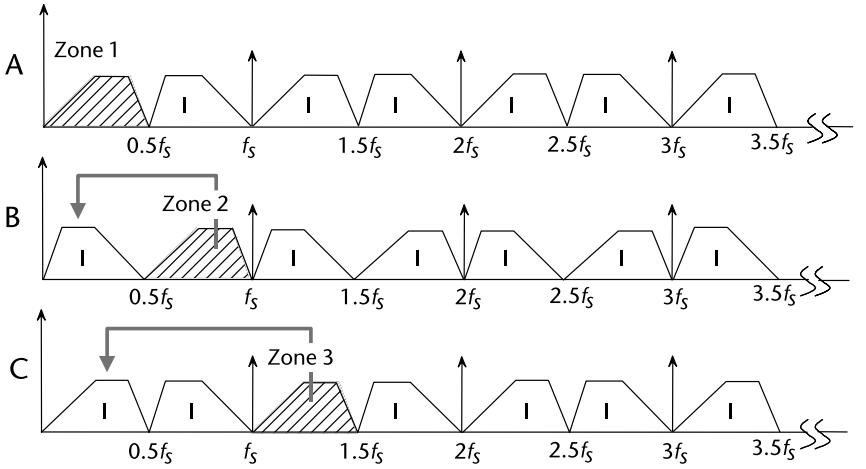


Figure 2.25 Nyquist region folding.

Notice that there is no mention of the absolute location of the band of sampled signals within the frequency spectrum relative to the sampling frequency. The only constraint is that the band of sampled signals be restricted to a single Nyquist zone (i.e., the signals must not overlap any multiple of $\frac{f_s}{2}$). In fact, this is the primary function of the antialiasing filter. Sampling signals above the first Nyquist zone has become popular in communications because the process is equivalent to analog demodulation. It is becoming common practice to sample IF signals directly and then use digital techniques to process the signal, thereby eliminating the need for an IF demodulator and filters. However, as the IF frequencies become higher, the dynamic performance requirements (bandwidth, linearity, distortion, etc.) on the ADC become more critical as performance must be adequate at the second or third Nyquist zone, rather than only baseband. This presents a problem for many ADCs designed to process signals in the first Nyquist zone. Therefore, an ADC suitable for undersampling applications must maintain dynamic performance into the higher-order Nyquist zones. This is specifically important in devices like the Pluto SDR, which includes *DC correction* to remove local oscillator leakage. This DC correction can be through of a highpass filter, which is set close to DC (25 kHz). For those modulation schemes, which do not strictly transmit information at DC, such as QPSK and quadrature amplitude modulation (QAM), this does not matter. For those modulation schemes that pass information at DC, this can be very difficult to work around without using an undersampling technique as described above. Capturing the data above DC, and then digitally moving it down can improve performance.

2.5.2 Fixed Point Quantization

The only errors (DC or AC) associated with an ideal *N*-bit data converter are those related to the sampling and quantization processes. The maximum error an ideal converter makes when digitizing a signal is $\pm\frac{1}{2}$ LSB, directly in between two digital values. This is obvious from the transfer function of an ideal *N*-bit ADC, which is

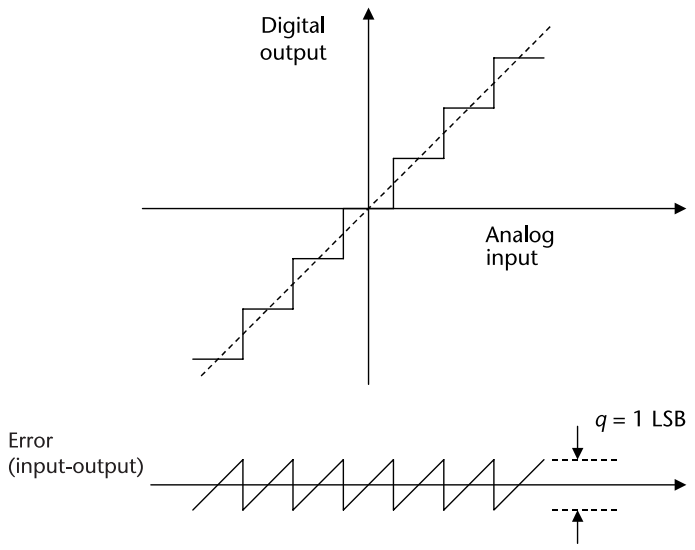


Figure 2.26 Ideal N-bit ADC quantization noise.

shown in Figure 2.26. The quantization error for any AC signal, which spans more than a few LSBs, can be approximated by an uncorrelated sawtooth waveform having a peak-to-peak amplitude of q, the weight of an LSB. Although this analysis is not precise it is accurate enough for most applications.

The quantization error as a function of time is shown in Figure 2.27. Again, a simple sawtooth waveform provides a sufficiently accurate model for analysis. The equation of the sawtooth error is given by

$$e(t) = st, \quad \frac{-q}{2s} < t < \frac{q}{2s}, \tag{2.33}$$

where s is the slope of the quantized noise. The mean-square value of $e(t)$ can be written:

$$\overline{e}^2(t) = \frac{q}{s}\int_{\frac{q}{2s}}^{\frac{-q}{2s}} (st)^2 dt = \frac{q^2}{12}. \tag{2.34}$$

The square root of (2.34), the root mean squared (RMS) noise quantization error, is approximately Gaussian and spread more or less uniformly over the Nyquist bandwidth of DC to $\frac{f_s}{2}$.

The theoretical SNR can now be calculated assuming a full-scale input sine wave $v(t)$

$$v(t) = \frac{q2^N}{2} sin(\omega t), \tag{2.35}$$

by first calculating the RMS value of the input signal defined as

$$\sqrt{\overline{v}(t)^2} = \frac{q2^N}{2\sqrt{2}}. \tag{2.36}$$

Therefore, the RMS signal-to-noise ratio for an ideal N-bit converter is

$$SNR = 20\,log_{10}\left(\frac{RMS\ of\ full\ scale\ input}{RMS\ of\ quatization\ noise}\right) = 20\,log_{10}\left(\frac{\frac{q2^N}{2\sqrt{2}}}{\frac{q}{\sqrt{12}}}\right). \tag{2.37}$$

After some simplification of (2.37) we arrive at our SNR in dB:

$$SNR = 20\,log_{10}\left(\sqrt{\frac{3}{2}}2^N\right) = 6.02N + 1.76. \tag{2.38}$$

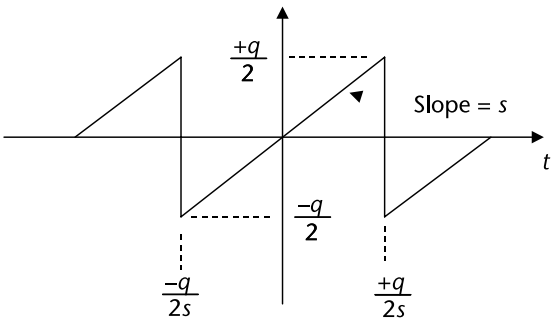


Figure 2.27 Ideal N-bit ADC quantization noise as a function of time.

Again, this is for over DC to $\frac{f_s}{2}$ bandwidth. In many applications the actual signal of interest occupies a smaller bandwidth (BW). For example, if digital filtering is used to filter out noise components outside BW, then a correction factor (called *process gain*) must be included in the equation to account for the resulting increase in SNR. The process of sampling a signal at a rate, which is greater than twice its bandwidth, is often referred to as oversampling. In fact oversampling in conjunction with quantization noise shaping and digital filtering is a key concept in sigma-delta converters, which will be discussed in Section 2.5.4.

Hands-On MATLAB Example: In Section 2.5.2 we made the following assumptions [11]:

- The sequence of error samples $e(t)$ is a sample sequence of a stationary random process;
- The error sequence is uncorrelated with the sequence of exact samples, $v(t)$;
- The random variables of the error process are uncorrelated; that is, the error is a white-noise process;
- The probability of the errpr process is uniform over the range of quantization error.

The underlying assumption here is that the quantization noise is uncorrelated to the input signal. We will see in certain common trivial examples that is not true. Under certain conditions where the sampling clock and the signal are harmonically related, the quantization noise becomes correlated and the energy is concentrated at the harmonics of the signal. In a practical ADC application, the quantization error generally appears as random noise because of the random nature of the wideband input signal and the additional fact that there is a usually a small amount of system noise that acts as a dither signal to further randomize the quantization error spectrum. It is important to understand the above point because single-tone sinewave FFT testing of ADCs is one of the universally accepted methods of performance evaluation. In order to accurately measure the harmonic distortion of an ADC, steps must be taken to ensure that the test setup truly measures the ADC distortion, not the artifacts due to quantization noise correlation.

We will use variations on the MATLAB code from Code 2.10 to explore the SFDR. We will expand on this concept with regards to the precision allowed for the computations. We begin with a double-precision floating point number

Code 2.10 SFDR test: **`sfdr_test.m`**

```
10 deltat = 1e-8;
11 fs = 1/deltat;
12 t = 0:deltat:1e-5-deltat;
13 fundamental = 3959297;
14 x = 10e-3*sin(2*pi*fundamental*t);
15 r = sfdr(x,fs)
17 sfdr(x,fs);
```

representation (MATLAB default). To keep the signal uncorrelated, we choose a tone at 3,959,297 Hz (a prime number).

This provides the results in Figure 3.28, where a SFDR measurement of 288.96 dBc is obtained, which is a very impressive measurement. However, a real radio such as Pluto SDR can not accept double-precision floating-point numbers, and thus fixed-point (12-bit) representation must be used.

The fixed-point format Pluto SDR and many other devices use is a signed format. The MSB bit is for sign and 11 remaining bits are for the magnitude. In our MATLAB script, we use 2^{11} as the magnitude to maximize the dynamic range of our signal before transmission. Therefore, we multiply to our integer value, round, and then scale down to ± 1 to normalize the amplitude.

This provides an underwhemling 45.97 dBc for the SFDR, as shown in Figure 2.29, which is not even 8-bits of performance. This is because in the example we scaled our signal to 10^{-3}, and the dynamic range of a fixed-point number, does not scale equally with a double-precision floating-point number.

To resolve the dynamic range issue, we will remove the 10^{-3} scaling and use 12-bit full scale. This results in Figure 2.30(a) with a respectable 86.82 dBc.

To improve this result even more, we can take advantage of a concept known as the FFT processing gain. We simply increase the number of samples to $10,000$ by using the following code in code 2.15, which simply changes the length of `t`. This provides the results in Figure 2.30(b), with an SFDR of 93.98 dBc. This is accomplished by simply increasing the number of samples, which increases the number of FFT bins, which in turn decreases the energy accumulated in each bin.

The example code in Code 2.10, Code 2.11, Code 2.12 and Code 2.15 utilized an uncorrelated F_A of $3,959,297$ Hz. What happens when it is correlated? If we simply round F_A to 4 MHz in the example (see Codes 2.13 and 2.14), we can see the results in Figure 2.31(a), which yields an SFDR of 82.58 dBc, a loss of 11.4 dB from our previous result in Figure 2.30(b).

To regain this loss in Figure 2.31(b), we can use a technique known as dithering. This moves the energy accumulated in the harmonics and pushes it out into the rest of the noise floor. The noise floor is higher but the worse case spur is lower, which is the figure of merit when calculating SFDR. This results in an SFDR of 91.68 dBc, only a 2.3-dB difference from our uncorrelated results in Figure 2.30(b).

Code 2.11 SFDR test: **`sfdr_test.m`**

```
41 bits=2^11;
42 x = round(10e-3*bits*sin(2*pi*fundamental*t))/bits;
45 sfdr(x,fs)
```

Code 2.12 SFDR test: **`sfdr_test.m`**

```
bits=2^11;
x = round(bits*sin(2*pi*fundamental*t))/bits;
sfdr(x,fs)
```

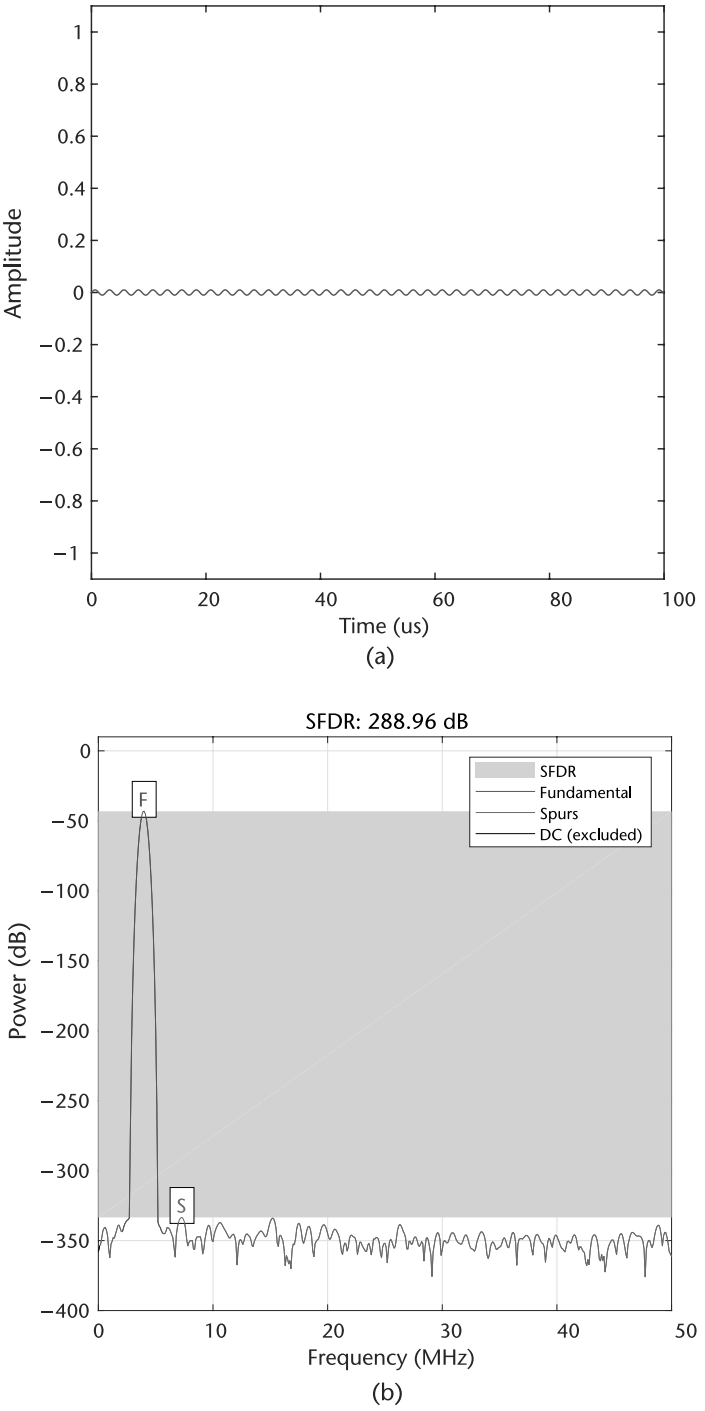


Figure 2.28 SFDR for a double-precision floating-point format. (a) Time domain floating-point representation, and (b) SFDR of double-precision floating-point, 1k points, F_A of 3,959,297 Hz.

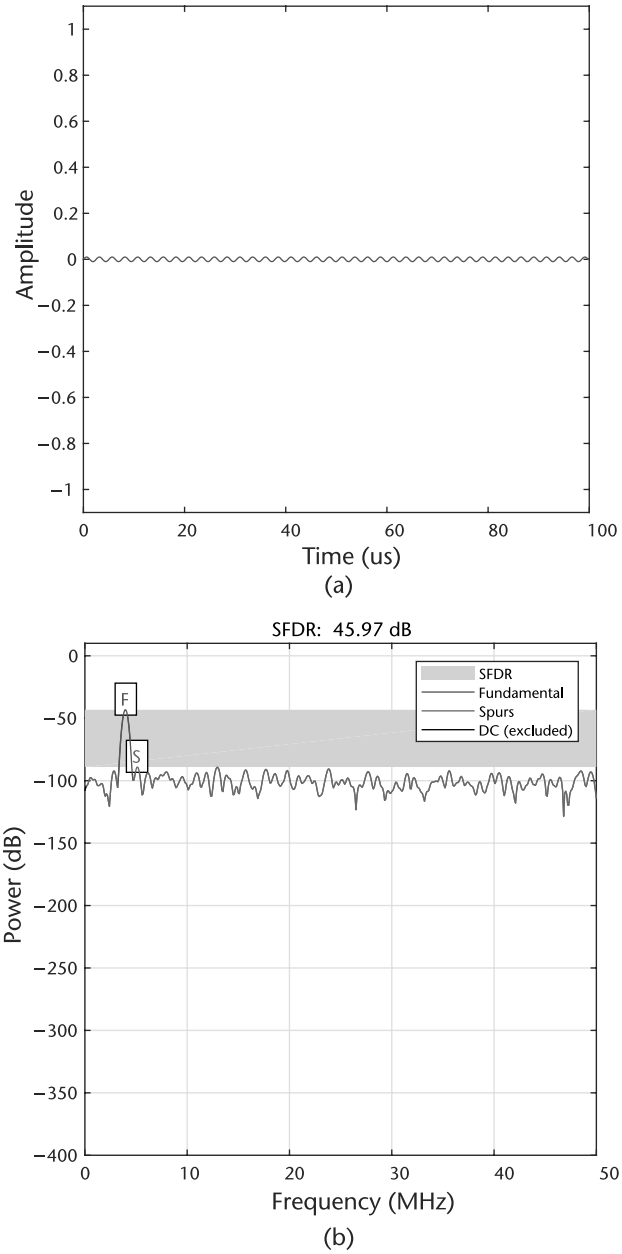


Figure 2.29 SFDR for 12-bit, scaled number. (a) Time domain floating-point representation, and (b) SFDR of scaled 12-bit fixed-point, 1k points, F_A of 3,959,297 Hz.

Rather than rounding up or down in a repeating pattern, we randomly round up or down by applying a ± 0.5 offset to the vector before we round. This is a very simple dither algorithm, but more complex implementations will exist in hardware.

The effects of finite bit length and data correlation should be understood before sending data to the hardware. The hardware will only make effects worse, not better.

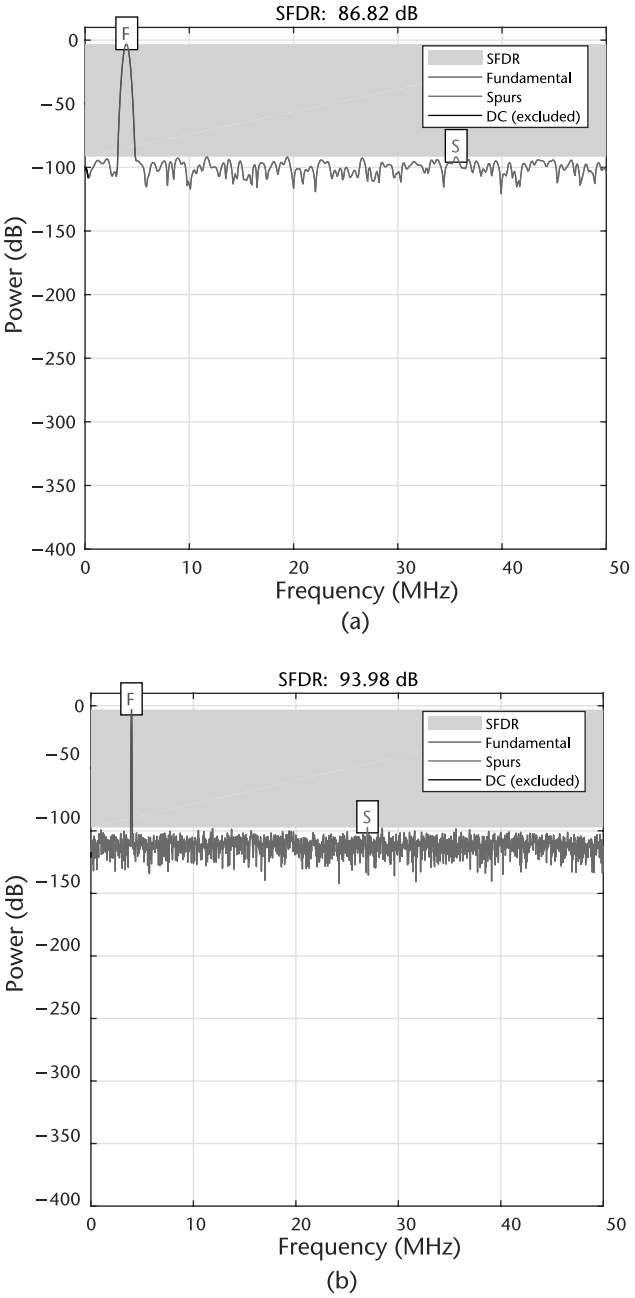


Figure 2.30 SFDR for 12-bit, scaled number. (a) SFDR of fullscale 12-bit fixed point, 1k points, F_A of 3,959,297 Hz, and (b) SFDR of full scale 12-bit fixed point, 10k points, F_A of 3,959,297 Hz.

2.5.3 Design Trade-offs for Number of Bits, Cost, Power, and So Forth

The most important aspect to remember about both receive chains (I/Q) is the effect of quantization from the ADC itself. That is, an N-bit word represents one of 2^N possible states, and therefore an N-bit ADC (with a fixed reference) an have only 2^N possible digital outputs. The resolution of data converters may be expressed in several different ways: the weight of the least significant bit (LSB), parts per million of full-scale (ppm FS), and millivolts (mV). Different devices, even from the same

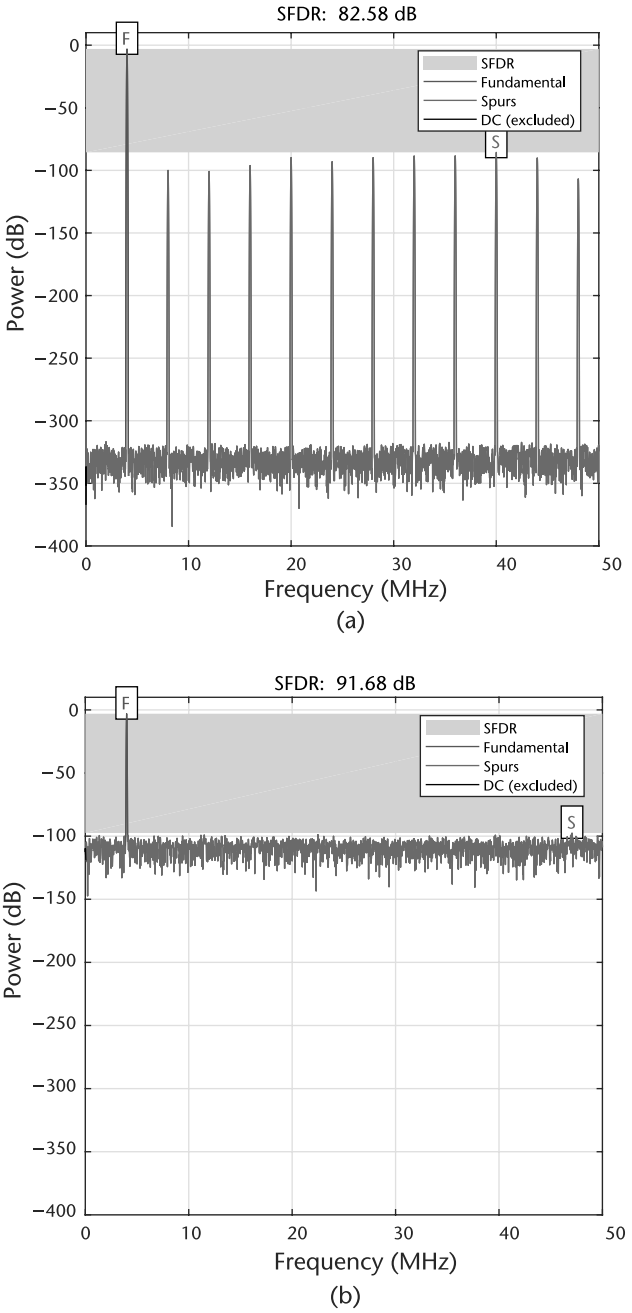


Figure 2.31 SFDR for 12-bit, scaled number. (a) SFDR of full scale 12-bit fixed point, 10k points, without dithering F_A of 4,000,000 Hz, and (b) SFDR of full scale 12-bit fixed point, 10k points, with dithering F_A of 4,000,000 Hz.

manufacturer, will be specified differently. Therefore, converter users must learn to translate between the different types of specifications if they are to compare devices successfully.

The size of the least significant bit for various resolutions for a 10 watt (20 V peak-to-peak) input is shown in Table 2.3.

Code 2.13 SFDR test: **sfdr_test.m**

```
 98 t = 0:deltat:1e-4-deltat;
 99 x = round(bits*sin(2*pi*fundamental*t))/bits;
100 r = sfdr(x,fs)
102 sfdr(x,fs);
```

Code 2.14 SFDR test: **sfdr_test.m**

```
126 fundamental=4000000;
127 x = round(bits*sin(2*pi*fundamental*t))/bits;
128 r = sfdr(x,fs)
130 sfdr(x,fs);
```

Code 2.15 SFDR test: **sfdr_test.m**

```
154 ran = rand(1,length(t)) - 0.5;
155 x = round(bits*sin(2*pi*fundamental*t) + ran)/bits;
158 sfdr(x,fs);
```

Table 2.3 Quantization: The Size of a Least Significant Bit

Resolution (N)	2^N	*Voltage (20 Vpp)*[1]	*PPM FS*	*%FS*	*dBFS*
2-bit	4	5.00 V	250,000	25	−12
4-bit	16	1.25 V	62,500	6.25	−24
6-bit	64	313 mV	15,625	1.56	−36
8-bit	256	78.1 mV	3,906	.391	−48
10-bit	1,024	19.5 mV	977	.097	−60
12-bit	4,096	4.88 mV	244	.024	−72
14-bit	16,384	1.22 mV	61.0	.0061	−84
16-bit	65,536	305 μV	15.2	.0015	−96
18-bit	262,144	76.2 μV	3.81	.00038	−108
20-bit	1,048,576	19.0 μV	.953	.000095	−120
22-bit	4,194,304	4.77 μV	.238	.000024	−132
24-bit	16,777,216	1.19 μV	.0596	.0000060	−144
26-bit	67,108,864	298 nV [1]	.0149	.0000015	−156

[1] 600 nV is the Johnson (thermal) noise in a 10-kHz BW of a 2.2 $k\Omega$ resistor at 25°C.

While a 24-bit converter with −144 dB of performance may sound like a good idea, it is not practical from a power or speed perspective. Althrough many 24-bit ADCs exist, they are not wideband such as the AD7177, which is a state-of-the-art 32-bit converter that is limited to 5 SPS to 10 kSPS output data rate and is not suitable for SDR but is a great solution for things like temperature and pressure measurement, chromatography, or weigh scales [12]. On the other hand, higher-speed ADCs do exist, such as the AD9208, which is a dual-channel 14-Bit 3GSPS ADC, providing an SFDR of 70 dBFS with 9-GHz analog input full-power bandwidth (the sixth Nyquist band). However, the AD9208 power draw is over 3W, which exceeds the entire power consumption of the Pluto SDR while streaming data over USB [13]. Nonetheless, a system based on the AD9208 could provide up to 12 Gbytes/second, which is more data than could be processed by software and would require custom signal processing hardware inside a FPGA. Devices like the AD9208

are used in 60-GHz bands RF bands, where single channels have bandwidths of over 2 GHz. These high-speed and wideband links use the same concepts and techniques described in the remaining text, but possess higher data rates, are more power-hungry, and are much more expensive. Nevertheless, learning the basic techniques of digital communications can be performed in with cost-effective devices such as the Pluto SDR in 20 MHz, or in many cases with much less of bandwidth. With regard to the ADC, in order to enable 12-bit devices to be used for a radio applications such as Pluto SDR, the signal chain for the AD9361 includes programmable analog gain as shown in Figure 2.32. This allows the input to the ADC to be driven to full scale as much as possible, which as we learned in Section 2.4.1 provides the best possible performance.

In a brief recap from operational amplifier theory, two types of gain are associated with amplifiers: signal gain and noise gain. We want to increase the signal but at the same time keep the noise as as low as possible. This is accomplished by increasing the signal in the analog domain before digitizing it with the ADC, as shown in Figure 2.32.

2.5.4 Sigma-Delta Analog-Digital Converters

Sigma-delta (Σ-Δ) analog-digital converters (ADCs) have been known for over 50 years, but only recently has the technology (high-density digital VLSI) existed to manufacture them as inexpensive monolithic integrated circuits. They are now used in many applications where a low-cost, medium-bandwidth, low-power, high-resolution ADC is required. There have been innumerable descriptions of the architecture and theory of Σ-Δ ADCs, but most commence with a deep description of the math, starting at the integrals and go on from there. Since this is not an ADC textbook, we will try to refrain from the mathematical development and explore things based on the previous topics covered in this chapter.

There is nothing particularly difficult to understand about Σ-Δ ADCs. The Σ-Δ ADC contains very simple analog electronics (a comparator, voltage reference, a switch, and one or more integrators and analog summing circuits), and digital computational circuitry. This circuitry consists of a filter, which is generally, but not invariably, a lowpass filter. It is not necessary to know precisely how the filter works to appreciate what it does. To understand how a Σ-Δ ADC works, familiarity with the concepts of oversampling, quantization noise shaping, digital filtering, and decimation is required, all topics covered earlier in this chapter.

Let us consider the technique of oversampling with an analysis in the frequency domain. Where a DC conversion has a quantization error of up to $\frac{1}{2}$ LSB, a sampled data system has quantization noise. A perfect classical N-bit sampling ADC has an RMS quantization noise of $\frac{q}{\sqrt{12}}$ uniformly distributed within the Nyquist band of DC to $\frac{f_s}{2}$, where q is the value of an LSB, as shown in Figure 2.33(a). Therefore, its SNR with a full-scale sine wave input will be $(6.02N + 1.76)$ dB. If the ADC is less than perfect and its noise is greater than its theoretical minimum quantization noise, then its effective resolution will be less than N-bits. Its actual resolution, often known as its effective number of bits (ENOB), will be defined by

$$ENOB = \frac{SNR - 1.76dB}{6.02dB} \tag{2.39}$$

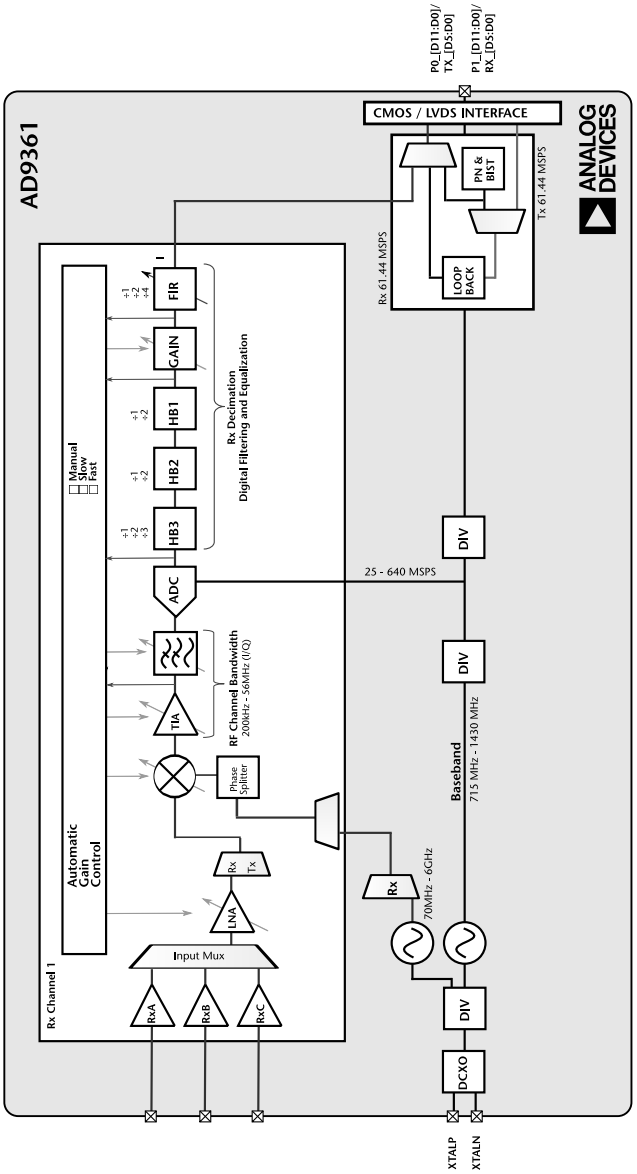


Figure 2.32 Simplified AD9361 receive block diagram.

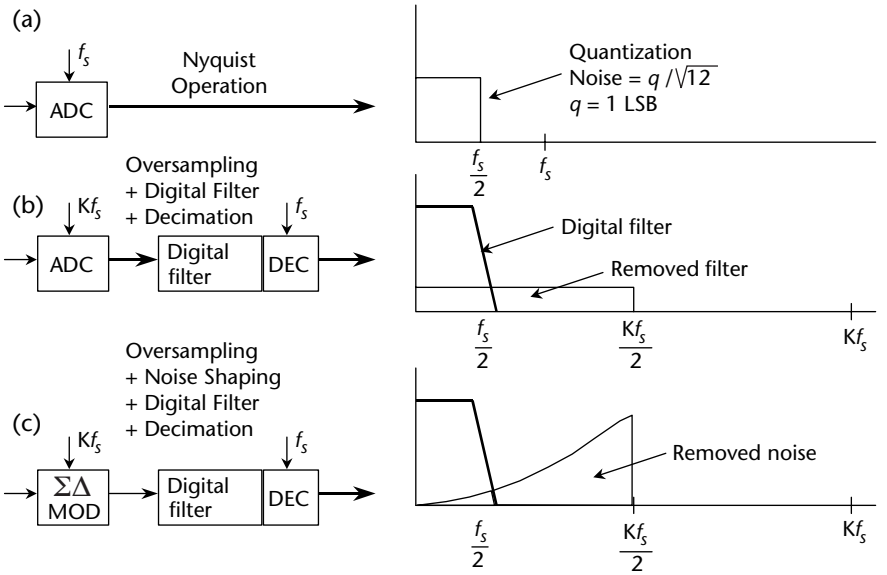


Figure 2.33 Oversampling, digital filtering, noise shaping, and decimation in a Σ-ΔADC.

Practically, ENOB is calculated from measuring signal-to-noise-and-distortion (SINAD, or S/(N + D)), which is the ratio of the RMS signal amplitude to the mean value of the root-sum-square (RSS) of all other spectral components, including harmonics but excluding DC, and correcting for a nonfull-scale input signal [6]. We can modify (2.39) to take into account the full-scale amplitude A_{FS} and the true input amplitude A_{IN} as

$$ENOB = \frac{SINAD - 1.76dB + 20log_{10}\frac{A_{FS}}{A_{IN}}}{6.02dB}. \tag{2.40}$$

If we choose a much higher sampling rate, Kf_s (see Figure 2.33[b]), the RMS quantization noise remains $\frac{q}{\sqrt{12}}$ but the noise is now distributed over a wider bandwidth DC to $\frac{Kf_s}{2}$. If we then apply a digital lowpass filter (LPF) to the output, we can remove much of the quantization noise but do not affect the wanted signal, resulting in an improved ENOB. Therefore, we can accomplished a high-resolution A/D conversion with a low-resolution ADC. The factor K is generally referred to as the oversampling ratio. It should be noted at this point that oversampling has an added benefit in that it relaxes the requirements on the analog antialiasing filter.

Since the bandwidth is reduced by the digital output filter, the output data rate may be lower than the original sampling rate (Kf_s) and still satisfy the Nyquist criterion. This may be achieved by passing every M^{th} result to the output and discarding the remainder. The process is known as decimation by a factor of M. Decimation does not cause any loss of information (see Figure 2.33[b]) as long as the decimation does not violate the Nyquist criterion. For a given input frequency, higher-order analog filters offer more attenuation. The same is true of Σ-Δ modulators, provided certain precautions are taken. By using more than one integration and summing stage in the Σ-Δ modulator, we can achieve higher orders

of quantization noise shaping and even better ENOB for a given oversampling ratio as is shown in Figure 2.34.

The actual Σ-Δ ADC found in the AD9363 used in the Pluto SDR is a fourth order, as shown in Figure 2.35 and described in the *Analog Devices Transceiver Support* Simulink model. As can be seen, reality is always a little more complicated than theory or first-order approximations.

2.6 Digital Signal Processing Techniques for SDR

DSP is a field always on the edge of mathematical complexity, computational performance, and growing mobility, influencing communications, medical imagining, radar, entertainment, and even scientific exploration. However, all these fields rely on the concept of translating analog information into digital representations and by some mechanisms processing that data. To do so, engineers and scientists rely on common tools and languages including C and Verilog, all of which enable manipulation of digital information in an efficient and procedural way. Regardless of the language, many important DSP software issues are specific to hardware, such as truncation error, bit patterns, and computational speed and efficiency of processors [14]. For now, we will mostly ignore those issues and focus on the algorithmic issues of signal processing and discuss the commonly used algorithms.

2.6.1 Discrete Convolution

Convolution is a mathematical tool of combining two signals to form a third signal, and forms the foundation for all DSP. Using the strategy of impulse decomposition, systems are described by a signal called the *impulse response*. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response.

Figure 2.36 presents the notation of convolution as applied to linear systems. A discrete sampled input signal, $x[n]$, enters a linear system with an impulse

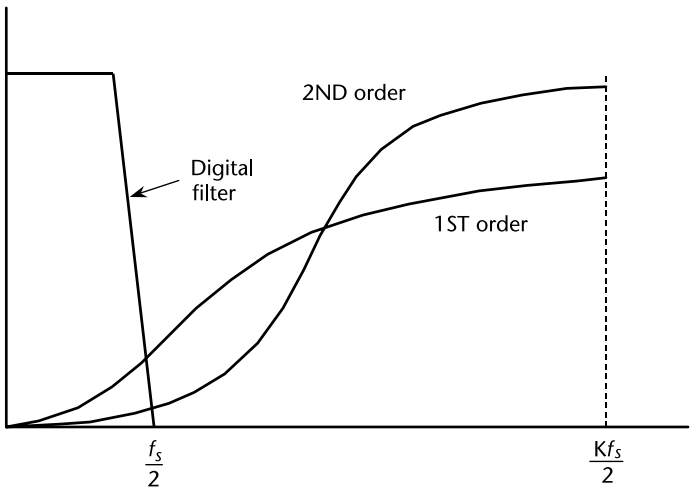


Figure 2.34 Σ-Δ modulators shape quantization noise.

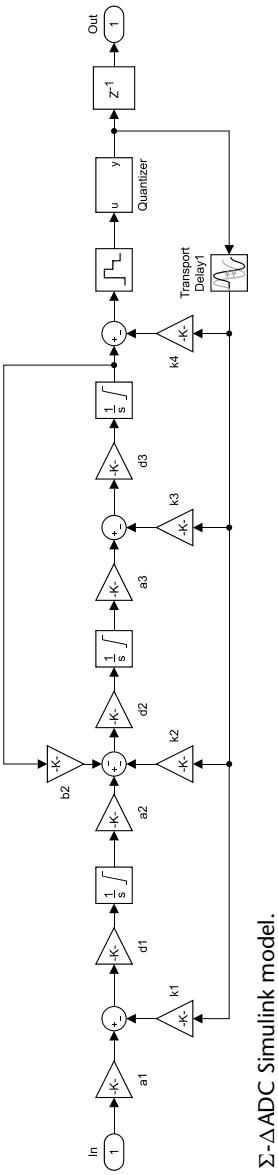


Figure 2.35 AD9361 Σ-ΔADC Simulink model.

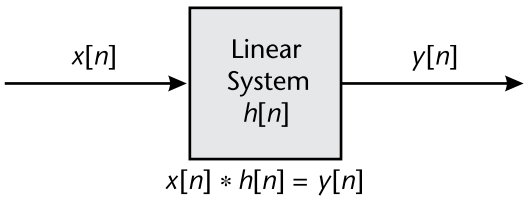


Figure 2.36 How convolution is used in DSP. The output signal from a linear system is equal to the input signal convolved with the system's impulse response.

response, $h[n]$ resulting in an output signal, $y[n]$. Expressed in words, the input signal convolved with the impulse response is equal to the output signal. As denoted in (2.41), convolution is represented by the $*$ operator. It is unfortunate that most programming languages, such as MATLAB, use the star to indicate multiplication and use special functions like MATLAB's `conv` function to indicate convolution. A star in a computer program means multiplication, while a star here notes convolution.

Fundamentally, the mathematics of convolution consists of several multiplications and additions. If $x[n]$ is an N point signal running from data sample 0 to $N-1$, and $h[n]$ is an M point signal running from 0 to $M-1$, the convolution of the two $y[n] = x[n] * h[n]$, is an $N + M - 1$ point signal running from 0 to $N + M - 2$, given by

$$y[i] = \sum_{j=0}^{M-1} h[j] \times x[i-j] = h[i] * x[i], \tag{2.41}$$

This equation is called the *convolution sum*. It allows each point in the output signal to be calculated independently of all other points in the output signal. The index, i, determines which sample in the output signal is being calculated. The use should not be confused by the n in $y[n] = x[n] * h[n]$, which is merely a placeholder to indicate that some variable is the index into the array. An implementation of the convolution sum of two vectors in MATLAB is shown in Code 2.16.

As used in signal processing, convolution can be understood in two separate ways. The first looks at convolution from the viewpoint of the input signal. This involves analyzing how each sample in the input signal contributes to many points in the output signal. The second way looks at convolution from the viewpoint of the output signal. This examines how each sample in the output signal has received information from many points in the input signal. Keep in mind that these two perspectives are different ways of thinking about the same mathematical operation. The first viewpoint is important because it provides a conceptual understanding of how convolution pertains to signal processing. The second viewpoint describes the mathematics of convolution. This typifies one of the most difficult tasks you will encounter in the signal processing field, making your conceptual understanding fit with the jumble of mathematics used to communicate the ideas.

Figure 2.37 shows convolution being used for lowpass and highpass filtering, which we will cover in more detail in Section 2.6.4. The example input signal is the sum of two components: three cycles of a sine wave (representing a high frequency),

Code 2.16 Convolution: **my_convolution.m**

```
% Receive two vectors and return a vector resultant of
% convolution operation
function conv = simple_conv(f, g)
    % Transform the vectors f and g in new vectors with the same length
    F = [f,zeros(1,length(g))];
    G = [g,zeros(1,length(f))];

    % FOR Loop to put the result of convolution between F and G vectors
    % in a new vector C. According to the convolution operation
    % characteristics, the length of a resultant vector of convolution
    % operation between two vector is the sum of vectors length minus 1
    for i=1:length(g)+length(f)-1
        % Create a new vector C
        C(i) = 0;
        % FOR Loop to walk through the vector F ang G
        for j=1:length(f)
           if(i-j+1>0)
               C(i) = C(i) + F(j) * G(i-j+1);
           end
        end
    end
    out = C;
end
```

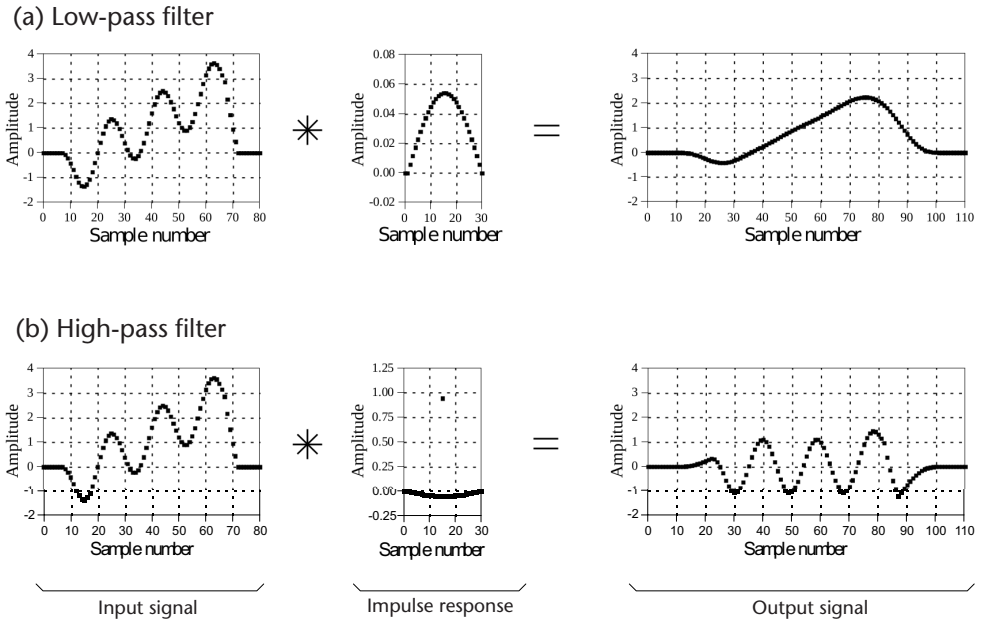


Figure 2.37 Examples of (a) lowpass and (b) highpass filtering using convolution. In this example, the input signal is a few cycles of a sine wave plus a slowly rising ramp. These two components are separated by using properly selected impulse responses.

plus a slowly rising ramp (composed of low frequencies). In (a), the impulse response for the lowpass filter is a smooth arch, resulting in only the slowly changing ramp waveform being passed to the output. Similarly, the highpass filter, (b), allows only the more rapidly changing sinusoid to pass.

2.6.2 Correlation

Cross-correlation and *autocorrelation* are two important concepts in SDR. Cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other.

The concept of correlation can best be presented with an example. Figure 2.38 shows the key elements of a radar system. A specially designed antenna transmits a short burst of radio wave energy in a selected direction. If the propagating wave strikes an object, such as the helicopter in this illustration, a small fraction of the energy is reflected back toward a radio receiver located near the transmitter. The transmitted pulse is a specific shape that we have selected, such as the triangle shown in this example. The received signal will consist of two parts: (1) a shifted and scaled version of the transmitted pulse, and (2) random noise, resulting from interfering radio waves, thermal noise in the electronics, and so forth. Since radio signals travel at a known rate, the speed of light, the shift between the transmitted and received

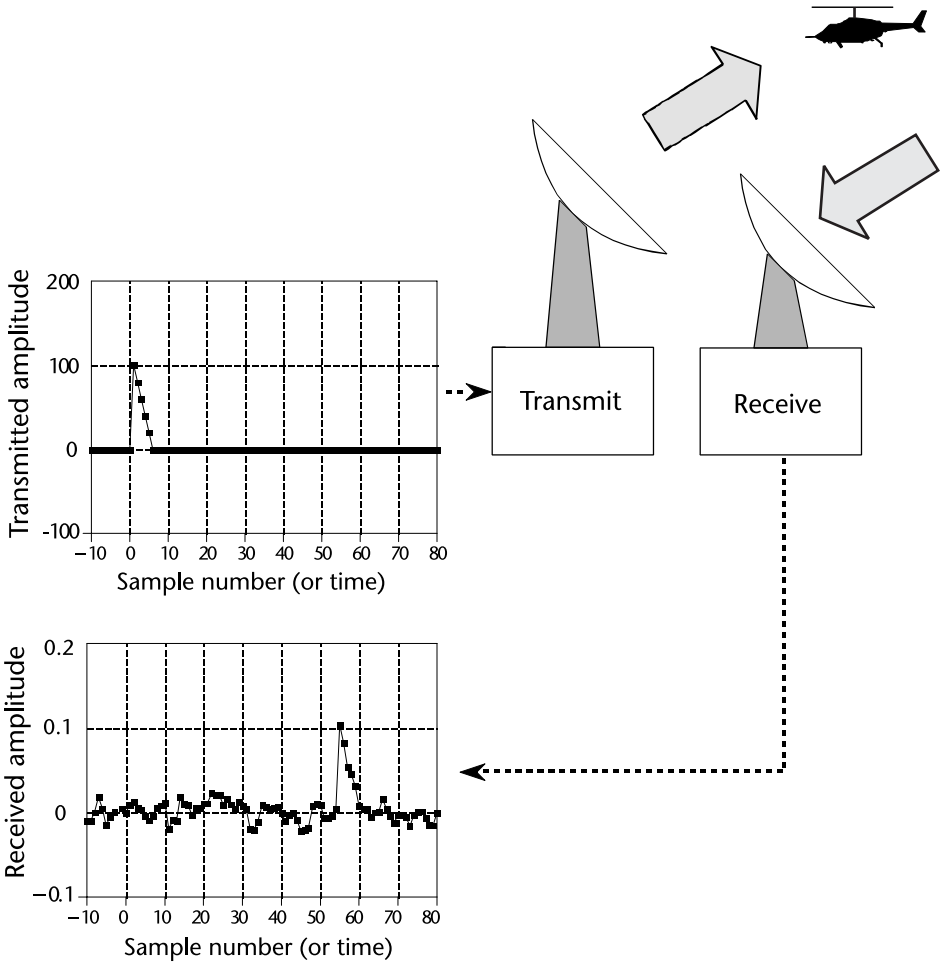


Figure 2.38 Key elements of a radar system. Like other echo location systems, radar transmits a short pulse of energy that is reflected by objects being examined. This makes the received waveform a shifted version of the transmitted waveform, plus random noise. Detection of a known waveform in a noisy signal is the fundamental problem in echo location. The answer to this problem is *correlation*.

pulse is a direct measure of the distance to the object being detected given a signal of some known shape. The challenge is determine the best way which signal occurs in another signal. Correlation is the solution to this problem.

Correlation is a mathematical operation that is very similar to convolution. Just as with convolution, correlation uses two signals to produce a third signal. This third signal is called the cross-correlation of the two input signals. If a signal is correlated with itself, the resulting signal is instead called the autocorrelation. The amplitude of each sample in the cross-correlation signal is a measure of how much the received signal resembles the target signal at that location. This means that a peak will occur in the cross-correlation signal for every target signal that is present in the received signal. In other words, the value of the cross-correlation is maximized when the target signal is aligned with the same features in the received signal.

One of the optimal techniques for detecting a known waveform in random noise is correlation. That is, the peak is higher above the noise using correlation than can be produced by any other linear system. (To be perfectly correct, it is only optimal for random white noise.) Using correlation to detect a known waveform is frequently called *matched filtering*. More on this in Section 4.7.1.

For discrete functions f and g, the cross-correlation is defined as

$$(f \star g)[n] = \sum_{j=-\infty}^{\infty} f^*[m] \times g[m+n], \tag{2.42}$$

where h^* denotes the complex conjugate of h. The cross-correlation of functions f(t) and g(t) is equivalent to the convolution of f*(â^'t) and g(t). That is

$$(f \star g)[n] = f^*(-t) * g(t) \tag{2.43}$$

Do not let the mathematical similarity between convolution and correlation fool you; they represent very different signal processing concepts. Convolution is the relationship between a system's input signal, output signal, and impulse response. Correlation is a way to detect a known waveform in a noisy background. The similar mathematics is a convenient coincidence that allows for algorithmic optimizations.

2.6.3 Z-Transform

In Section 2.1.1, we have introduced the Fourier transform, which deals with continuous-time signals on frequency domain. Since we are focusing on digital filter design in this section, where discrete-time signals are involved, we need to introduce a new type of transform; namely, the z-transform.

The z-transform of a discrete-time signal $x[n]$ is defined as the power series:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}, \tag{2.44}$$

where z is a complex variable [1].

The z-transform is used to analyze discrete-time systems. Its continuous-time counterpart is the Laplace transform, defined as following:

$$X(s) = \int_{-\infty}^{\infty} x(t)e^{-st}dt, \tag{2.45}$$

where t is the time variable in seconds across the time domain and $s = \sigma + j\omega$ is a complex variable. When evaluated along the $j\omega$ axis (i.e., $\sigma = 0$), the Laplace transform reduces to the Fourier transform defined in (2.2). Thus, the Laplace transform generalizes the Fourier transform from the real line (the frequency axis $j\omega$) to the entire complex plane.

According to Section 2.2.2, we know that if a continuous-time signal $x(t)$ is uniformly sampled, its sampling signal $x_s(t)$ can be expressed as

$$x_s(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT), \tag{2.46}$$

where T is the sampling interval. If we take the Laplace transform of both sides, we will get

$$X_s(s) = \int_{-\infty}^{\infty} x_s(t)e^{-st}dt = \int_{-\infty}^{\infty} \left[\sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT)\right] e^{-st}dt. \tag{2.47}$$

Since integration and summation are both linear operators, we can exchange their order. Then, based on the sampling property of the delta function, we can further get

$$X_s(s) = \sum_{n=-\infty}^{\infty} x(nT)\left[\int_{-\infty}^{\infty} \delta(t - nT)e^{-st}dt\right] = \sum_{n=-\infty}^{\infty} x(nT)e^{-snT}. \tag{2.48}$$

Let $z = e^{sT}$, or $s = \frac{1}{T}\ln z$, then (2.48) becomes

$$X(z) = \sum_{n=-\infty}^{\infty} x(nT)z^{-n}. \tag{2.49}$$

Since T is the sampling interval, $x(nT) = x[n]$. The equation above can be further written as

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}, \tag{2.50}$$

which is exactly the definition of z-transform in (2.44). Therefore, the z-transform and the Laplace transform can be connected by

$$z = e^{sT}, \tag{2.51}$$

or

$$s = \frac{1}{T}\ln(z). \tag{2.52}$$

According to (2.44), we know that z-transform is the series of z^{-1}. Actually, the z-transform has no meaning unless the series converge. Given a limitary-amplitude sequence $x[n]$, the set of all the z values that makes its z-transform converge is

called region of convergence (ROC). Based on the theory of series, these z-values must satisfy

$$\sum_{n=-\infty}^{\infty} \left| x[n]z^{-n} \right| < \infty. \tag{2.53}$$

The frequently used z-transform pairs and their region of convergence are listed in Table 2.4.

When discussing a linear time-invariant system, the z-transform of its system impulse response can be expressed as the ratio of two polynomials:

$$H(z) = \frac{b_m z^m + b_{m-1}z^{m-1} + \cdots + b_1 z + b_0}{a_n z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0} = \frac{B(z)}{A(z)}, \tag{2.54}$$

where the roots of $A(z) = 0$ are called the *poles* of the system and the roots of $B(z) = 0$ are called the *zeros* of the system. It is possible that the system can have multiple poles and zeros.

If we factorize the numerator $B(z)$ and denominator $A(z)$, (2.54) can be written as:

$$H(z) = C\frac{(z - z_1)(z - z_2)\cdots(z - z_m)}{(z - p_1)(z - p_2)\cdots(z - p_n)}, \tag{2.55}$$

where C is a constant, $\{p_k\}$ are all the poles, and $\{z_k\}$ are all the zeros. It will help us draw the pole-zero plot of $H(z)$.

Suppose we have a linear time-invariant system whose system impulse response is defined as

$$h[n] = n^2 a^n u[n]. \tag{2.56}$$

According to Table 2.4, its z-transform is as follows:

$$H(z) = a\frac{z(z + a)}{(z - a)^3}. \tag{2.57}$$

Comparing (2.57) with (2.55), we can easily get that this system has two zeros, $z_1 = 0$ and $z_2 = -a$, and three poles, $p_1 = p_2 = p_3 = a$. Therefore, its pole-zero plot is shown in Figure 2.39.

Table 2.4 Z-Transform Table: Selected Pairs[1]

$x[n]$	$X(z)$	*Region of Convergence*
$\delta[n]$	1	all z
$a^n u[n]$	$\frac{z}{z-a}$	$\lvert z\rvert > \lvert a\rvert$
$na^n u[n]$	$\frac{az}{(z-a)^2}$	$\lvert z\rvert > \lvert a\rvert > 0$
$n^2 a^n u[n]$	$\frac{az(z+a)}{(z-a)^3}$	$\lvert z\rvert > a > 0$
$\left(\frac{1}{a^n} + \frac{1}{b^n}\right) u[n]$	$\frac{az}{az-1} + \frac{bz}{bz-1}$	$\lvert z\rvert > \max(\frac{1}{\lvert a\rvert}, \frac{1}{\lvert b\rvert})$
$a^n u[n] \sin(\omega_0 n)$	$\frac{az \sin\omega_0}{z^2 - 2az\cos\omega_0 + a^2}$	$\lvert z\rvert > a > 0$
$a^n u[n] \cos(\omega_0 n)$	$\frac{z(z - a\cos\omega_0)}{z^2 - 2az\cos\omega_0 + a^2}$	$\lvert z\rvert > a > 0$
$e^{an} u[n]$	$\frac{z}{z-e^a}$	$\lvert z\rvert > e^{-a}$
$e^{-an} u[n] \sin(\omega_0 n)$	$\frac{ze^a \sin\omega_0}{z^2 e^{2a} - 2ze^a\cos\omega_0 + 1}$	$\lvert z\rvert > e^{-a}$
$e^{-an} u[n] \cos(\omega_0 n)$	$\frac{ze^a(ze^a - \cos\omega_0)}{z^2 e^{2a} - 2ze^a\cos\omega_0 + 1}$	$\lvert z\rvert > e^{-a}$

[1] From [15]

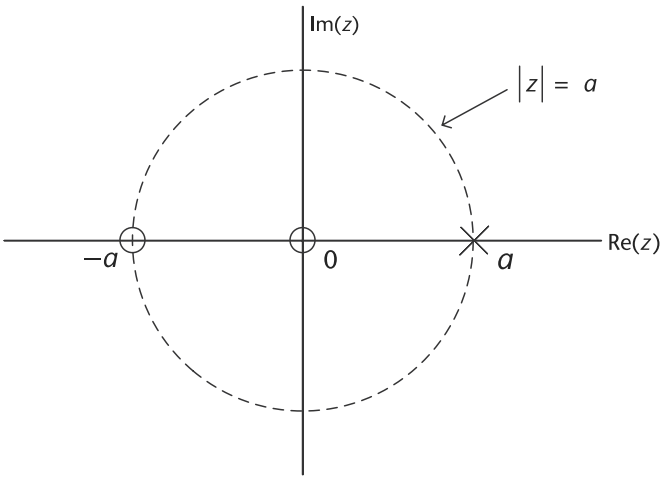


Figure 2.39 Pole-zero plot of the system defined in (2.57). Poles are denoted using crossings, and zeros are denoted using circles. The region of convergence of this system is the region outside the circle $z = |a|$.

There are several properties of the z-transform that are useful when studying signals and systems in the z-transform domain. Since these properties are very similar to those of the Fourier transform introduced in Section 2.1.1, we list them in Table 2.5 without further discussion.

2.6.4 Digital Filtering

When doing signal processing, we usually need to get rid of the noise and extract the useful signal. This process is called filtering, and the device employed is called filter, which discriminates, according to some attribute of the objects applied at its input, what passes through. A typical filter is a frequency-selective circuit. If noise and useful signal possess different frequency distributions and are present together at input of the filter, then by applying this circuit, the noise will be attenuated or even eliminated while useful signal is retained.

Filters can be classified from different aspects. For example, according to its frequency response, filter can be classified as lowpass, highpass, bandpass and bandstop. According to the signal it deals with, a filter can be classified as a analog filter or a digital filter [1]. Specifically, an analog filter deals with continuous-time signals, while a digital filter deals with discrete-time signals. This section will focus on digital filters. The ideal magnitude response characteristics of these types of filters are shown in Figure 2.40. According to Figure 2.40, the magnitude response characteristics of an ideal filter can be generalized as follows: In pass-band, the magnitude is a constant, while in stop-band, the magnitude falls to zero. However, in reality, this type of ideal filter cannot be achieved, so a practical filter is actually the optimum approximation of an ideal filter.

In order to perform digital filtering, input and output of a digital system must both be discrete-time series. If the input series is $x[n]$, the impulse response of the filter is $h[n]$, then the output series $y[n]$ will be

$$y[n] = x[n] * h[n]. \tag{2.58}$$

Table 2.5 Z-Transform Properties[1]

Property	*Time Signal*	*Z-Transform Signal*
Linearity	$\sum_{m=1}^{N} a_m x_m(t)$	$\sum_{m=1}^{N} a_m X_m(z)$
Symmetry	$x[-n]$	$X(z^{-1})$
Shifting	$x[n-m]$	$z^{-m}X(z)$
Scaling	$a^n x[n]$	$X\left(\frac{z}{a}\right)$
Derivative	$nx[n]$	$-z\frac{dX(z)}{dz}$
Integration	$\sum_{m=-\infty}^{n} x[m]$	$\frac{z}{z-1}X(z)$
Time convolution	$x[n] * h[n]$	$X(z)H(z)$
Frequency convolution	$x[n]h[n]$	$\frac{1}{2\pi j}\int X(v)H\left(\frac{z}{v}\right)\frac{dv}{v}$

[1] Based on [15]. Suppose the time signal is $x[n]$, and its z-transform signal is $X(z)$.

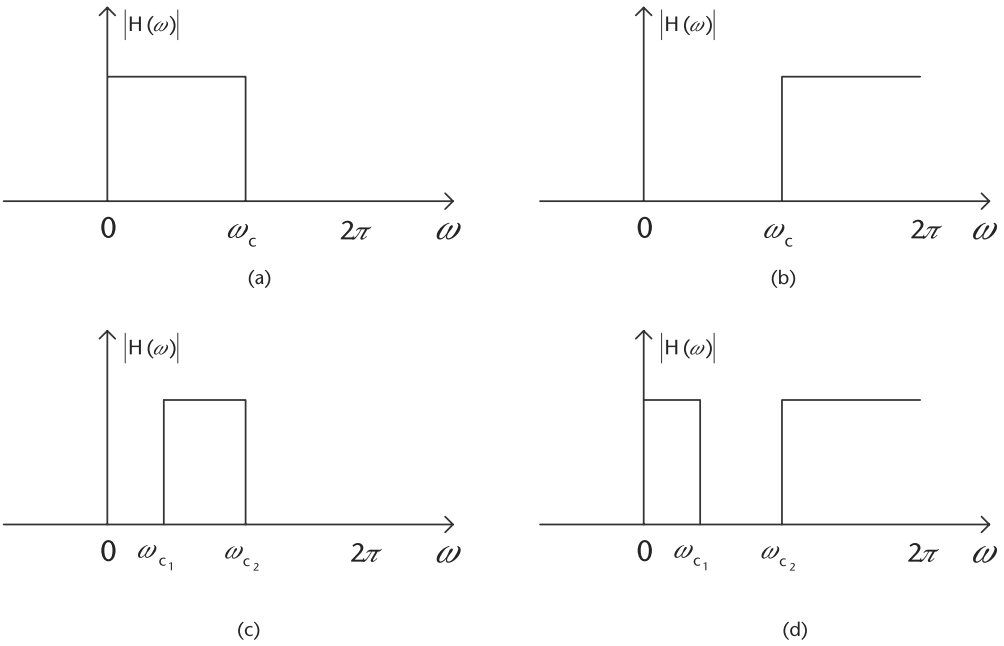


Figure 2.40 Ideal magnitude response characteristics of four types of filters on the frequency range $[0, 2\pi]$. (a) Lowpass filter, (b) highpass filter, (c) bandpass filter, where $(\omega_{c_1}, \omega_{c_2})$ is passband, and (d) bandstop filter, where $(\omega_{c_1}, \omega_{c_2})$ is stopband.

According to the time convolution property in Table 2.5, on the frequency domain, (2.58) is equivalent to

$$Y(z) = X(z)H(z), \tag{2.59}$$

where $X(z)$ and $Y(z)$ are the z-transforms of the input and output series, $x[n]$ and $y[n]$, and $H(z)$ is the z-transform of $h[n]$.

Since ideal brick wall filters are not achievable in practice, we limit our attention to the class of linear time-invariant systems specified by the difference equation [1]:

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k], \tag{2.60}$$

where $y[n]$ is the current filter output, the $y[n-k]$ are previous filter outputs, and the $x[n-k]$ are current or previous filter inputs. This system has the following frequency response:

$$H(z) = \frac{\sum_{k=0}^{M} b_k e^{-z}}{1 + \sum_{k=1}^{N} a_k e^{-z}}, \tag{2.61}$$

where the $\{a_k\}$ are the filter's feedback coefficients corresponding to the poles of the filter, and the $\{b_k\}$ are the filter's feed-forward coefficients corresponding to the zeros of the filter, and N is the filter's order.

The basic digital filter design problem is to approximate any of the ideal frequency response characteristics with a system that has the frequency response (2.61), by properly selecting the coefficients $\{a_k\}$ and $\{b_k\}$ [1].

There are two basic types of digital filters, finite impulse response (FIR) and infinite impulse response (IIR) filters. When excited by an unit sample $\delta[n]$, the impulse response $h[n]$ of a system may last a finite duration, as shown in Figure 2.41(a), or forever even before the input is applied, as shown in Figure 2.41(b). In the former case, the system is finite impulse response, and in the latter case, the system is infinite impulse response.

An FIR filter of length M with input $x[n]$ and output $y[n]$ can be described by the difference equation [1]:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_{M-1} x[n-M+1] = \sum_{k=0}^{M-1} b_k x[n-k], \tag{2.62}$$

where the filter has no feedback coefficients $\{a_k\}$, so $H(z)$ has only zeros.

IIR filter has been defined in (2.60), which has one or more nonzero feedback coefficients $\{a_k\}$. Therefore, once the filter has been excited with an impulse, there is always an output.

2.6.4.1 Case Study: Cascaded Integrator-Comb Filters

Cascaded integrator-comb filters (CIC filters) play an important role in the SDR hardware. They were invented by Eugene B. Hogenauer and are a class of FIR filters used in multirate signal processing. The CIC filter finds applications in interpolation and decimation. However, unlike most FIR filters, it has a decimator or interpolator built into the architecture [16].

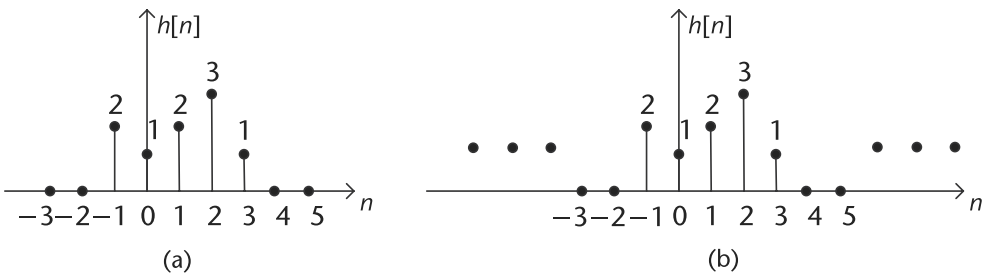


Figure 2.41 The impulse responses of an FIR filter and an IIR filter. (a) The impulse response of an FIR filter, and (b) the impulse response of an IIR filter.

A CIC filter consists of one or more integrator and comb filter pairs. In the case of a decimating CIC, the input signal is fed through one or more cascaded integrators, and then a downsampler, followed by one or more comb sections, whose number equals to the number of integrators. An interpolating CIC is simply the reverse of this architecture, with the downsampler replaced with a upsampler, as shown in Figure 2.42.

To illustrate a simple form of a comb filter, consider a moving average FIR filter described by the difference equation:

$$y[n] = \frac{1}{M+1}\sum_{k=0}^{M} x[n-k]. \tag{2.63}$$

The system function of this FIR filter is

$$H(z) = \frac{1}{M+1}\sum_{k=0}^{M} z^{-k} = \frac{1}{M+1}\frac{[1-z^{-(M+1)}]}{(1-z^{-1})}. \tag{2.64}$$

Suppose we replace z by z^L, where L is a positive integer; then the resulting comb filter has the system function:

$$H_L(z) = \frac{1}{M+1}\frac{[1-z^{-L(M+1)}]}{(1-z^{-L})}, \tag{2.65}$$

where L is decimation or interpolation ratio, M is number of samples per stage, usually 1 or 2.

This filter has zeros on the unit circle at

$$z_k = e^{j2\pi k/L(M+1)}, \tag{2.66}$$

for all integer value of k except $k = 0, L, 2L, ..., ML$, as shown in Figure 2.43.

The common form of the CIC filter usually consists of several stages, then the system function for the composite CIC filter is

$$H(z) = H_L(z)^N = \left(\frac{1}{M+1}\frac{1-z^{-L(M+1)}}{1-z^{-L}}\right)^N, \tag{2.67}$$

where N is number of stages in the composite filter.

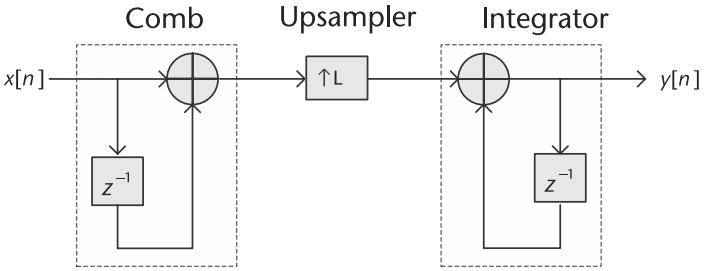


Figure 2.42 The structure of an interpolating cascaded integrator-comb filter [17], with input signal $x[n]$ and output signal $y[n]$. This filter consists of a comb and integrator filter pair, and an upsampler with interpolation ratio L.

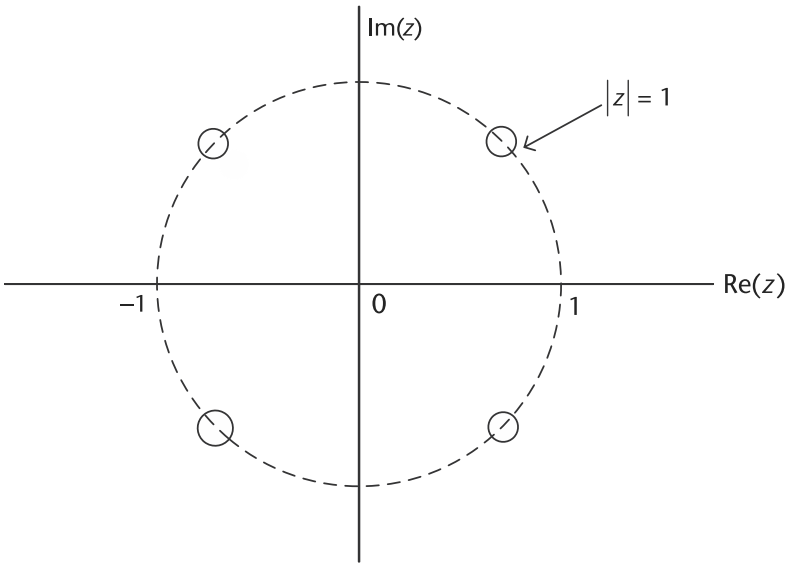


Figure 2.43 The zeros of a CIC filter defined in (2.65), where all the zeros are on the unit circle.

Characteristics of CIC filters include linear phase response and utilizing only delay and addition and subtraction. In other words, it requires no multiplication operations, thus making it suitable for hardware implementation.

2.6.4.2 Case Study: FIR Halfband Filter

FIR *halfband filters* are also widely used in multirate signal processing applications when interpolating/decimating. Halfband filters are implemented efficiently in polyphase form because approximately half of its coefficients are equal to zero. Halfband filters have two important characteristics, the passband and stopband ripples must be the same, and the passband-edge and stopband-edge frequencies are equidistant from the halfband frequency $\frac{f_s}{4}$.

For example the Pluto SDR has multiple programmable halfband filters in the receive and transmit chains. The RX HB3/DEC3 provides the choice between two different fixed-coefficient decimating filters, decimating by a factor of 2, 3, or 1 (bypassed). The input to the filter (the output of the ADC) is 2^4, or 16 values.

When the RX HB3 filter is used, the decimation factor is set to 2, and the following coefficients are used : [1, 4, 6, 4, 1]. Note that the full- scale range for the RX HB3 filter is 16 (2^4). When the RX DEC3 filter is used, the decimation factor is set to 3. and the following coefficients: [55, 83, 0, -393, -580, 0, 1914, 4041, 5120, 4041, 1914, 0, -580, -393, 0, 83, 55]. The full-scale range for the RX DEC3 filter is 16384 (2^{14}).

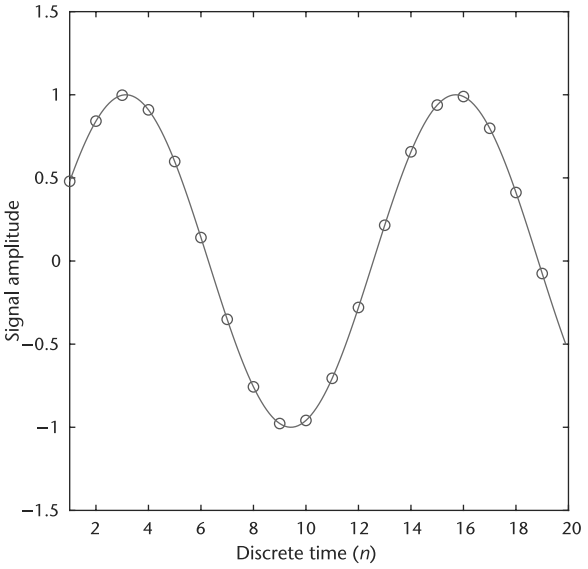
2.7 Transmit Techniques for SDR

In Section 2.2, it was described how an analog signal is converted to a digital signal using an ADC, as illustrated in Figure 2.5. Although these digital signals, which consist of 0 and 1, can be processed with various digital signal processing

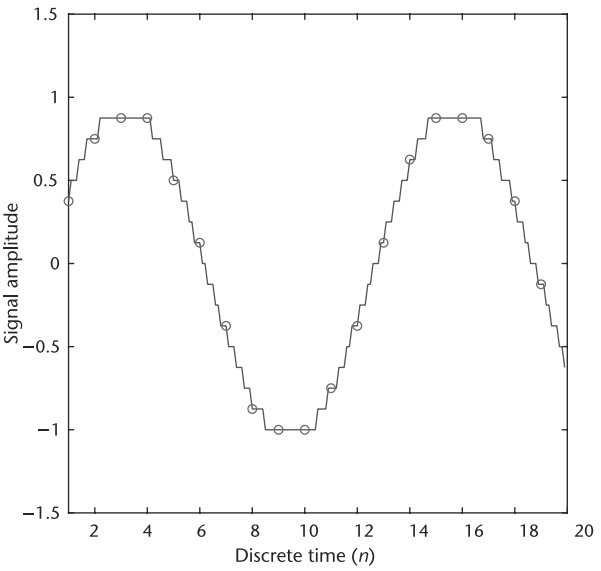
techniques, these digital signals cannot be directly used for transmission. These signals must be first conditioned then converted back into an analog signal DAC.

It can be seen from this that the extra decimation will allow bit growth and extra fidelity to gain in the system.

In Figure 2.44(a) a continuous analog sine wave is shown, which has been sampled at sample rate f_s. These samples goes through a quantizer, described in Section 2.2, which provides the output as shown in Figure 2.44(b). In this example,



(a)



(b)

Figure 2.44 Time domain. (a) Continuous analog sine wave: time domain, and (b) quanitized analog sine wave: time domain.

a 4-bit converter is used, which only provides 16 states (to make things easier to visualize). It is natural that many people want to connect the samples with smooth lines. However, the true waveform does have multiple discrete steps between the samples, as shown in Figure 2.44(b). Those values between samples are ignored and are not passed out of the ADC.

What comes out the ADC and is processed by the digital signal processing algorithms and eventually the digital to analog converter have no need for waveforms to be smooth. Therefore, they experience what is shown in Figure 2.45(a). Signals are not continuous and are not smooth. This is adequate as long as the topics described in Section 2.2.3 are understood and do not violate the Nyquist sampling theorem. The only time this is a problem is when we want to actually convert the sampled data back into the continuous analog domain. The bandwidth of the continuous analog domain is infinite and does not stop at $\frac{f_s}{2}$, as shown in Figure 2.45(b). When the time domain data is consider beyond the digital limits, aliases of signals are still visible.

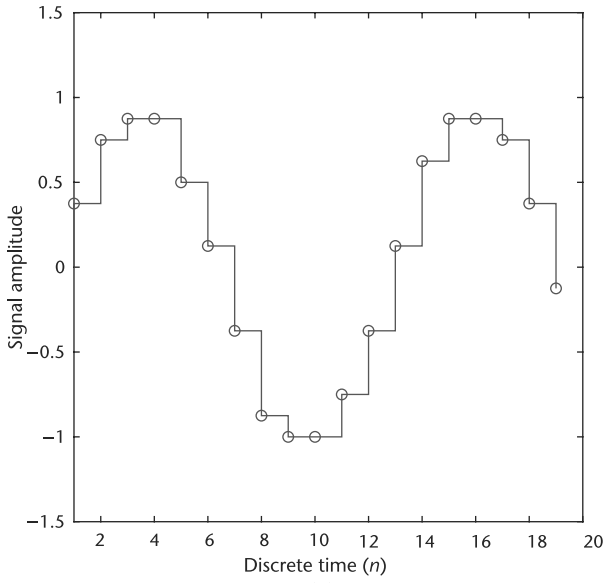
This is why we traditionally have two filters in the transmit portion of a SDR, a digital transmit or *pulse-shaping* filter, which changes each symbol in the digital message into a digital pulse stream for the DAC. This is followed by an *analog reconstruction filter*, which removes aliasing caused by the DAC [9]. This process is shown in Figure 2.46.

2.7.1 Analog Reconstruction Filters

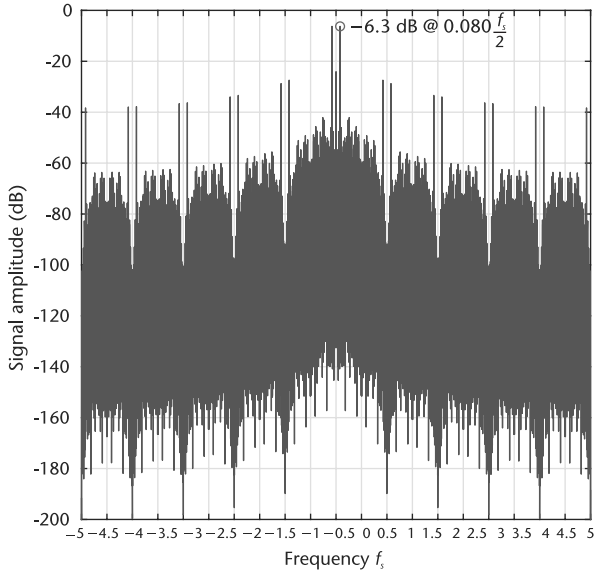
In Section 2.5.4, it was discussed how oversampling can ease the requirements on the antialiasing filter and how a sigma-delta ADC has this inherent advantage. In a DAC-based system, the concept of interpolation can be used in a similar manner with the analog reconstruction filter. This is common in digital audio CD players, where the basic update rate of the data from the CD is about 44.1 kSPS. As described in Section 2.2.5, zeros are inserted into the data, which is passed through a digital filter thereby increasing the effective sample update rate to four times, eight times, or sixteen times the original rate. The high oversampling rate moves the image frequencies higher, allowing a less complex filter with a wider transition band.

The same concept can be applied to a high-speed DAC. Suppose that a traditional DAC is driven at an input word rate of 30 MSPS, as shown in Figure 2.47 A; the DAC output frequency f_s is 10 MHz. The image frequency component at 20 MHz must be attenuated by the analog reconstruction filter, and the transition band of the filter is 10 to 20 MHz. The image frequency must be attenuated by 60 dB. Therefore, the filter must cover a passband of 10 MHz with 60-dB stopband attenuation over the transition band lying between 10 and 20 MHz (one octave). An analog Butterworth filter design gives 6 dB attenuation per octave for each pole. Therefore, a minimum of 10 poles are required to provide the desired attenuation. The necessary filter becomes even more complex as the transition band becomes narrower.

Next, let us assume that we increase the DAC update rate to 60 MSPS and interpolate the original data samples by 2, resulting in a data stream at 60 MSPS. The response of the DAC relative to the two-times oversampling frequency is shown in Figure 2.47 B. The analog reconstruction filter transition zone is now 10 to



(a)



(b)

Figure 2.45 (a) Upsampled, band-limited, sine wave: time domain, and (b) band-limited random data: fourier domain.

50 MHz (the first image occurs at $2f_c - f_o = 60 - 10 = 50$ MHz). This transition zone is now larger than two octaves, implying that a five- or six-pole Butterworth filter is sufficient, which is much easier to design and build.

2.7.2 DACs

In theory, the simplest method for digital-to-analog conversion is to pull the samples from memory and convert them into an impulse train. Similar to the sampling

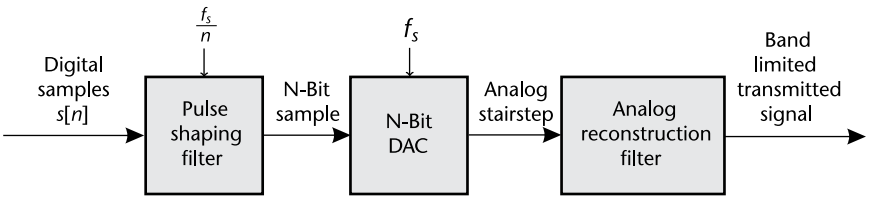


Figure 2.46 On the transmitter side, the DAC converts the digital symbols into an analog signal for transmission.

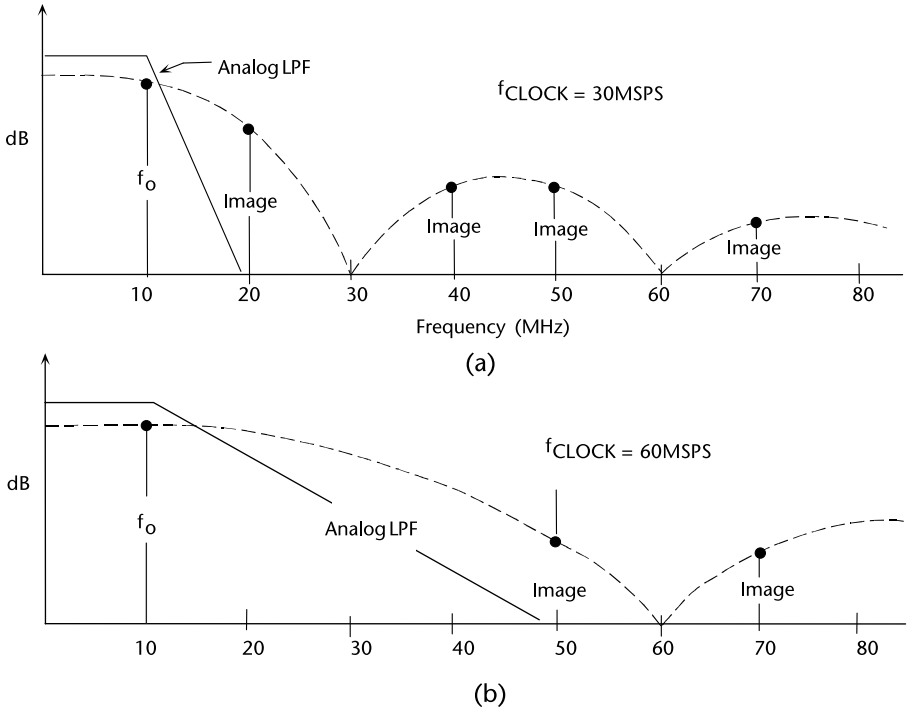


Figure 2.47 Analog reconstruction filter requirements for $f_o = 10$ MHz, with $f_s = 30$ MSPS, and $f_s = 60$ MSPS [18].

function in Section 2.2.2, the impulse modulator can be defined as

$$p(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT), \tag{2.68}$$

where $p(t) = 1$ for $t = kT$, and $p(t) = 0$ for all the other time instants. Therefore, the analog pulse train after the impulse modulator is

$$s_a(t) = s[n]p(t) = \sum_{k=-\infty}^{\infty} s(kT)\delta(t - kT) = \begin{cases} s(kT) & t = kT \\ 0 & t \neq kT \end{cases}, \tag{2.69}$$

where each digital symbol $s[n]$ initiates an analog pulse that is scaled by the value of the symbol. The original analog signal can be reconstructed by passing this impulse train through a lowpass filter, with the cutoff frequency equal to one-half of the sampling rate.

Although this method is mathematically pure, it is difficult to generate the required infinitively narrow impulse pulses even in modern in electronics. To get around this, nearly all DACs operate by holding the last value until another sample is received. This is called a *zeroth-order hold*, which is the DAC equivalent of the sample-and-hold used during ADC. The zeroth-order hold produces the staircase appearance in the time domain, as shown in Figure 2.44(b).

In the frequency domain, the zeroth-order hold results in the spectrum of the impulse train being multiplied by sinc function, given by the equation

$$H(f) = \left| \frac{\sin(\pi f / f_s)}{\pi f / f_s} \right|. \tag{2.70}$$

If you already have a background in this material, the zeroth-order hold can be understood as the convolution of the impulse train with a rectangular pulse, having a width equal to the sampling period. This results in the frequency domain being multiplied by the Fourier transform of the rectangular pulse, that is, the sinc function. The dashed line in Figure 2.47 shows the sinc function of the 30 MHz and 60 MHz DACs. It can be seen from Figure 2.47 that the sinc function attenuates signals in the passband. However, something in the system needs to compensate for this effect by the reciprocal of the zeroth-order hold's effect, $\frac{1}{sinc(x)}$, or simply accept this nonideality. Many ignore this problem, but it is very trivial to do with the Pluto SDR. Allowing a user to easily transmit with a very flat passband.

2.7.3 Digital Pulse-Shaping Filters

To understand why we need to include digital pulse-shaping filters in all radio designs, a short example will be shown. Phase-shift keying (PSK) is a simple but common digital modulation scheme that transmits data by changing the phase of a reference signal. This is shown in Figures 2.48(a) and 2.48(d), where the time and frequency domain for the carrier is shown. The frequency is very narrow and should be easy to transmit.

The alternating bitstream of ones and zeros shown in Figures 2.48(b) and 2.48(e) causes the problem. Examining this, we observe that the square wave has infinite frequency information, which is something very difficult to transmit in a practical consideration.

When we multiply the carrier in Figure 2.48(a) with the bitstream in Figure 2.48(b) to attempt to transmit this data out the antenna, it results in Figures 2.48(c) and 2.48(f), which is a signal with infinite bandwidth in the continuous time analog domain. Not only will we have difficulty transmitting things, our nearest neighbors in adjacent RF channels will not like either.

Going back to our mathematical model of impulses, (2.70) indicates that without a digital pulse-shaping filter, these pulses $s_a(t)$ will be transmitted through the channel right away. In theory, given infinite bandwidth on the receiver side, we should be able to get the same pulse train although with some delay. However, in reality we actually cannot recover such a signal due to finite bandwidth and interference between adjacent symbols.

There are two situations when adjacent symbols may interfere with each other: when the pulse shape is wider than a single symbol interval T, and when there is a

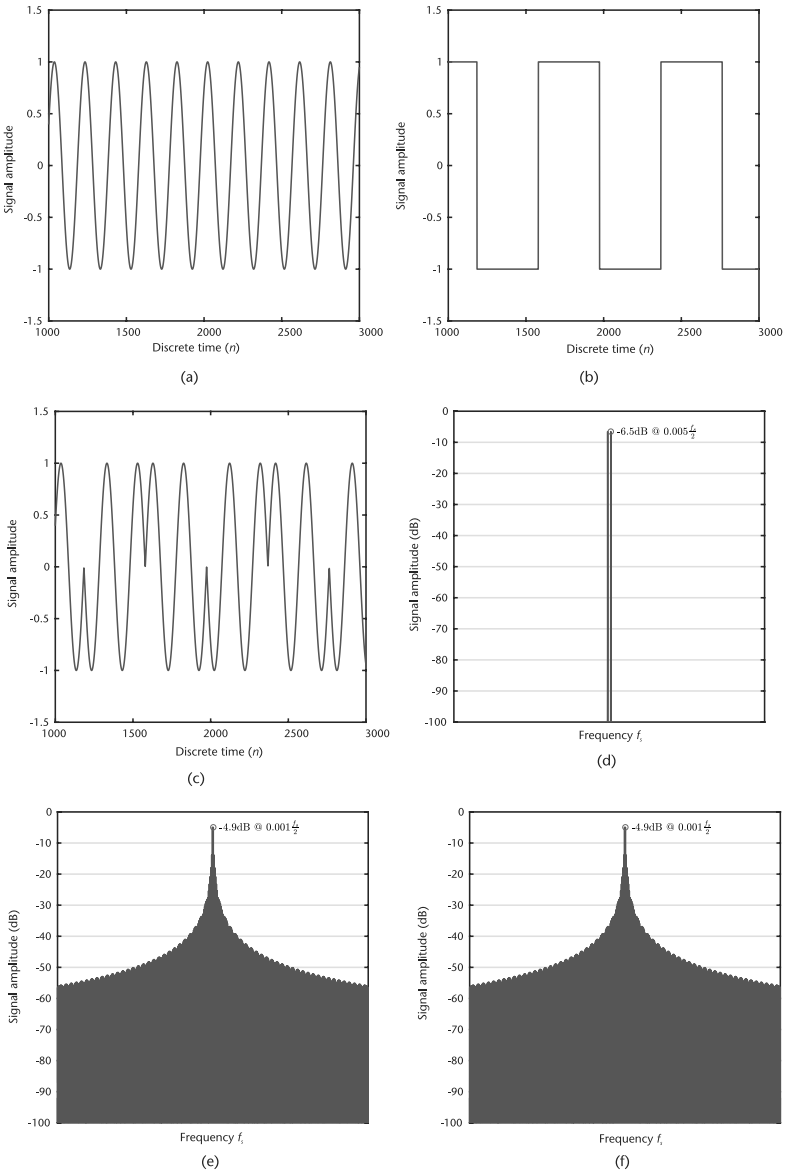


Figure 2.48 Phase-shift keyed modulated signal, carrier, data, and resulting modulated waveform, time and fourier domain. (a) Carrier: time domain, (b) PSK data: time domain, (c) PSK modulation: time domain, (d) carrier: Fourier domain, (e) PSK data: Fourier domain, and (f) PSK modulation: Fourier domain.

nonunity channel that smears nearby pulses, causing them to overlap. Both of these situations are called *intersymbol interference* (ISI) [9].

In order to solve these problems, pulse-shaping filters are introduced to bandlimit the transmit waveform.

2.7.4 Nyquist Pulse-Shaping Theory

In a communication system, there are normally two pulse-shaping filters, one on the transmitter side, and the other on the receiver side, as shown in Figure 2.49, where

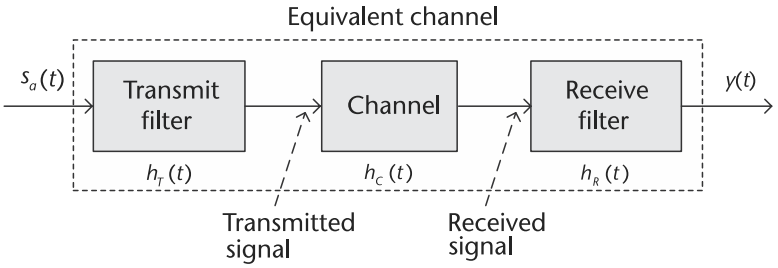


Figure 2.49 The equivalent channel of a communication system, which consists of the transmit filter, the channel, and the receive filter.

we use $h_T(t)$ and $h_R(t)$ to denote the impulse response of the transmit filter and receive filter. For simplicity, when considering the Nyquist pulse-shaping theory, we usually use the concept of equivalent channel, which not only includes the channel itself, but also the two pulse-shaping filters. Therefore, the impulse response of the equivalent channel is

$$h(t) = h_T(t) * h_C(t) * h_R(t). \tag{2.71}$$

Now, let us consider under which condition we can assure that there is no intersymbol interference between symbols. The input to the equivalent channel, $s_a(t)$, has been defined in (2.70). As mentioned before, each analog pulse is scaled by the value of the symbol, so we can express $s_a(t)$ in another way:

$$s_a(t) = \sum a_k \delta(t - kT), \tag{2.72}$$

where a_k is the value of the kth symbol. It yields the output to the equivalent channel, $y(t)$, which is

$$y(t) = s_a(t) * h(t) = \sum a_k[\delta(t - kT) * h(t)] = \sum a_k h(t - kT). \tag{2.73}$$

Therefore, given a specific time instant, for example, $t = mT$, where m is a constant, the input $s_a(t)$ is

$$s_a(mT) = \sum a_k \delta(mT - kT) = a_m. \tag{2.74}$$

Consequently, the output $y(t)$ becomes

$$y(mT) = \sum a_k h(mT - kT) = a_0 h(mT) + a_1 h(mT - T) + ... + a_m h(mT - mT). \tag{2.75}$$

Since we do not want the interference from the other symbols, we would like the output to contain only the a_m term, which is

$$y(mT) = a_m h(mT - mT). \tag{2.76}$$

Moreover, it means at a time instant $t = mT$, we need to have

$$h(mt - kT) = \begin{cases} C & k = m \\ 0 & k \neq m \end{cases}, \tag{2.77}$$

where C is some nonzero constant.

If we generalize (2.77) to any time instant t, we can get the Nyquist pulse-shaping theory as below. The condition that one pulse does not interfere with other

pulses at subsequent T-spaced sample instants is formalized by saying that $h(t)$ is a *Nyquist pulse* if it satisfies

$$h(t) = h(kT) = \begin{cases} C & k = 0 \\ 0 & k \neq 0 \end{cases}, \tag{2.78}$$

for all integers k.

2.7.5 Two Nyquist Pulses

In this section, we will introduce two important Nyquist pulses; namely, *sinc pulse* and *raised cosine pulse*. When considering (2.78), the most straightforward pulse we can think of is a rectangular pulse with time width less than T, or any pulse shape that is less than T wide. However, the bandwidth of the rectangular pulse (and other narrow pulse shapes) may be too wide. Narrow pulse shapes do not utilize the spectrum efficiently, but wide pulse shapes may cause ISI, so what is needed is a signal that is wide in time (and narrow in frequency) that also fulfills the Nyquist condition in (2.78) [9].

In mathematics, the sinc function is defined as

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}, \tag{2.79}$$

and is shown in Figure 2.50, when variable x takes an integer value k, the value of the sinc function will be

$$\text{sinc}(k) = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}. \tag{2.80}$$

In other words, zero crossings of the normalized sinc function occur at nonzero integer values.

Another important property of sinc function is that sinc pulse is the inverse Fourier transform of a rectangular signal, as shown in Figure 2.51(a). Suppose the

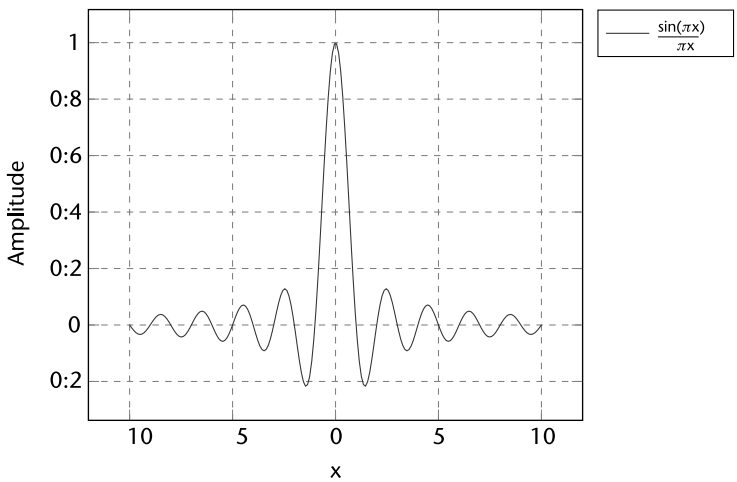
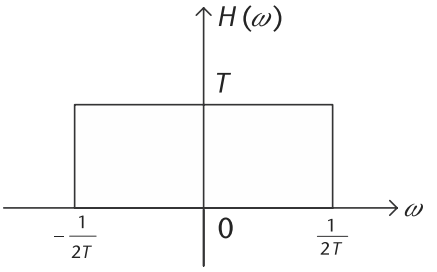
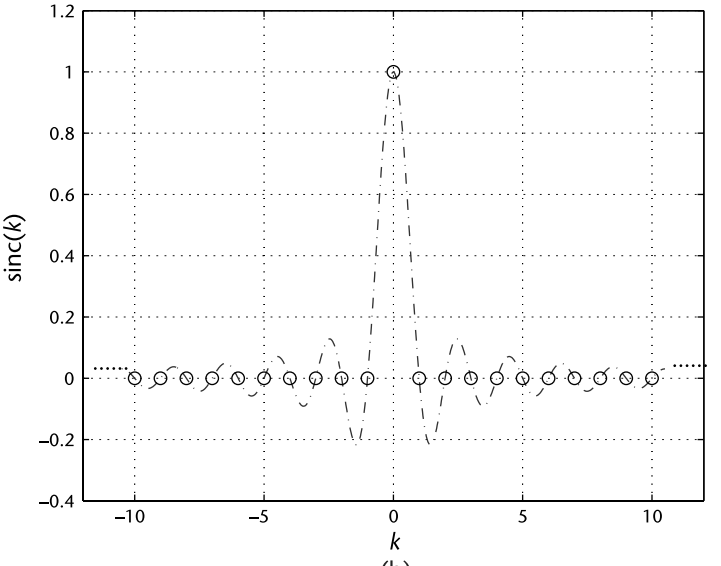


Figure 2.50 The plot of sinc function as defined in (2.79). The x-axis is x, and the y-axis is sinc(x).



(a)



(b)

Figure 2.51 The sinc pulse on time domain and its Fourier transform, rectangular pulse, on frequency domain. (a) The rectangular pulse on frequency domain, defined in (2.81), and (b) the sinc pulse defined in (2.84). The x-axis is k, where $k = \frac{t}{T}$, and the y-axis is sinc(k).

rectangular signal is defined as [19]:

$$H(\omega) = \begin{cases} T & |\omega| < \frac{1}{2T} \\ 0 & \text{otherwise} \end{cases} . \tag{2.81}$$

Taking the inverse Fourier transform of the rectangular signal will yield the sinc signal as

$$h(t) = \text{sinc}\left(\frac{t}{T}\right). \tag{2.82}$$

Change the variable $t = kT$ in (2.82) yields

$$h(t) = h(kT) = \text{sinc}\left(\frac{kT}{T}\right) = \text{sinc}(k). \tag{2.83}$$

Since k is an integer here, according to (2.80), we can continue writing (2.83) as

$$h(t) = h(kT) = \text{sinc}\left(\frac{kT}{T}\right) = \text{sinc}(k) = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}. \tag{2.84}$$

Comparing (2.84) with (2.78), we can easily find that if we make $t = kT$, the sinc pulse exactly satisfies Nyquist pulse-shaping theory in Section 2.7.4. In other words, by choosing sampling time at kT (sampling frequency equals $\frac{1}{T}$), our sampling instants are located at the equally spaced zero crossings, as shown in Figure 2.51(b), so there will be no intersymbol interference.

Recall the Nyquist sampling theorem in Section 2.2.3 states that a real signal, $x(t)$, which is bandlimited to B Hz can be reconstructed without error using a minimum sampling frequency of $F_s = 2B$ Hz. In this case, the minimum sampling frequency is $F_s = \frac{1}{T}$ Hz. Therefore, the corresponding minimum bandwidth is

$$B = \frac{F_s}{2} = \frac{1}{2T}, \tag{2.85}$$

which is exactly the bandwidth of the rectangular signal defined in (2.81). Based on the discussion above, this choice of sinc pulse $h(t)$ yields the minimum bandwidth $B = B_{\text{min}} = \frac{1}{2T}$, so it is called the Nyquist-I Pulse [20].

Sinc pulses are a very attractive option because they are wide in time and narrow in frequency, which means they have the advantages of both spectrum efficiency and no ISI. However, sinc pulses are not practical since they have ISI sensitivity due to timing errors. For instance, for large t, the sinc pulse defined in (2.82) has the following approximation:

$$h(t) \sim \frac{1}{t}, \tag{2.86}$$

so it is obvious that timing error can cause large ISI. We must also note that sinc pulse are infinite in time, making them unrealizable.

Consequentially, we need to introduce Nyquist-II pulses, which have a larger bandwidth $B > B_{\text{min}}$, but with less ISI sensitivity. Since this type of pulse is more practical, it is much more widely used in practice.

The raised cosine pulse is one of the most important type of Nyquist-II pulses, which has the frequency transfer function defined as

$$H_{RC}(f) = \begin{cases} T & 0 \leq |f| \leq \frac{1-\beta}{2T} \\ \frac{T}{2}\left(1 + \cos\left(\frac{\pi T}{\beta}(|f| - \frac{1-\beta}{2T})\right)\right) & \frac{1-\beta}{2T} \leq |f| \leq \frac{1+\beta}{2T} \\ 0 & |f| \geq \frac{1+\beta}{2T} \end{cases}, \tag{2.87}$$

where β is the rolloff factor, which takes value from 0 to 1, and $\frac{\beta}{2T}$ is the excess bandwidth.

The spectrum of raised cosine pulse is shown in Figure 2.52. In general, it has the bandwidth $B \geq 1/(2T)$. When $\beta = 0$, the bandwidth $B = 1/(2T)$, and there is no excess bandwidth, which is actually a rectangular pulse. On the other end, when $\beta = 1$, it reaches the maximum bandwidth $B = 1/T$.

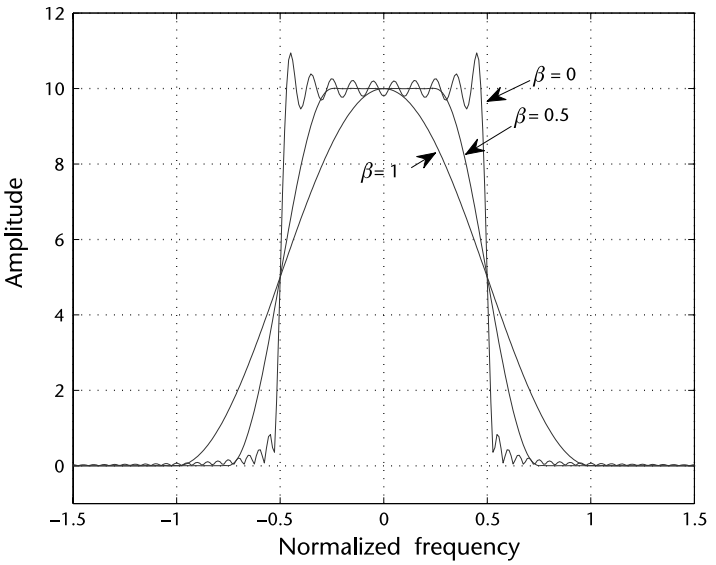


Figure 2.52 Spectrum of a raised cosine pulse defined in (2.87), which varies by the rolloff factor β. The x-axis is the normalized frequency f_0. The actual frequency can be obtained by f_0/T.

By taking the inverse Fourier transform of $H_{RC}(f)$, we can obtain the impulse response of raised cosine pulse, defined as

$$h_{RC}(t) = \frac{\cos\left(\pi \frac{\beta t}{T}\right)}{1 - \left(2\frac{\beta t}{T}\right)^2} \text{sinc}\left(\frac{\pi t}{T}\right). \tag{2.88}$$

Nyquist-II pulses do not have an ISI sensitivity because its peak distortion, the tail of $h_{RC}(t)$, converges quickly, which can be expressed as

$$D_p = \sum_{n=-\infty}^{\infty} |h_{RC}(\epsilon' + (n-k))| \sim \frac{1}{n^3}. \tag{2.89}$$

Therefore, when timing error occurs, the distortion will not accumulate to infinity in a similar fashion related to Nyquist-I pulses [20].

Actually, in many practical communications systems, *root raised cosine filters* are usually used [21]. If we consider the communication channel as an ideal channel and we assume that the transmit filter and the receive filter are identical, we can use root raised cosine filters for both of them, and their net response must equal to $H_{RC}(f)$ defined in (2.87). Since the impulse response of the equivalent channel can be expressed as

$$h(t) = h_T(t) * h_C(t) * h_R(t), \tag{2.90}$$

where $h_C(t)$ is the impulse response of the communication channel, and $h_T(t)$ and $h_R(t)$ are the impulse responses of the transmit filter and the receive filter, it means on frequency domain, we have

$$H_{RC}(f) = H_T(f)H_R(f). \tag{2.91}$$

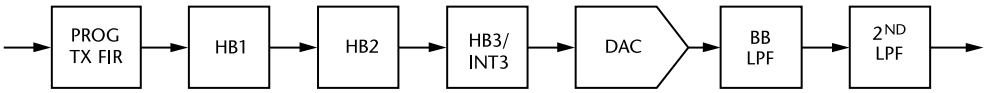


Figure 2.53 Internal AD9361 Tx signal path.

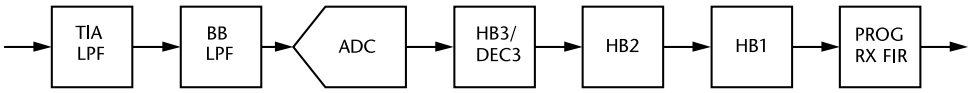


Figure 2.54 Internal AD9361 Rx signal path.

Therefore, the frequency response of root raised cosine filter must satisfy

$$|H_T(f)| = |H_R(f)| = \sqrt{|H_{RC}(f)|}. \tag{2.92}$$

2.8 Chapter Summary

SDR experimentation requires both strong computer skills and extensive knowledge of digital signal processing. This chapter lists some useful topics including sampling, pulse shaping, and filtering. The purpose of this chapter is to help the readers to get prepared for the experimentation chapters, especially for the Pluto SDR hardware. For example, on the Pluto SDR transmit path (see Figure 2.53), there is a programmable 128-tap FIR filter, interpolating halfband filters, a DAC, followed by two lowpass analog reconstruction filters (LPF). In order to understand how these work together and properly configure things, you need to understand sampling theory.

On the Pluto SDR receive path (see Figure 2.54), the signal flows through an antialiasing filter, the ADC, through digital decimating half band filters, and eventually a 128-tap programmable FIR, where the filtering knowledge is useful. This 128-tap programmable FIR can be used to compensate for the loss in the passband because of the antialiasing filter.

In addition, when you build a real communication system, you will need additional transmit and receive filters, which requires expertise in pulse shaping.

References

[1] Proakis, J. G., and Dimitris G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Third Edition, Upper Saddle River, NJ: Prentice Hall, 1996.

[2] Elali, T. S., and M. A. Karim, *Continuous Signals and Systems with MATLAB*, Boca Raton, FL: CRC Press, 2001.

[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, January 1978.

[4] Smith, S. W., *The Scientist and Engineers Guide to Digital Signal Processing*, Second Edition, 1999, http://www.analog.com/en/education/education-library/scientist_engineers_guide.html.

[5] Frigo, M., "A Fast Fourier Transform Compiler," Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99), Atlanta, GA, May 1999, http://www.fftw.org/.

[6] Kester, W., Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost inthe Noise Floor, Analog Devices, 2008, http://www.analog.com/MT-003.

[7] Kester, W., Mixed Signal and DSP Design Techniques, Analog Devices, 2002, http://www.analog.com/en/education/education-library/mixed_signal_dsp_design_book.html.

[8] James, J. F., *A Student's Guide to Fourier Transforms: With Applications in Physics and Engineering*, Third Edition, Cambridge, UK: Cambridge University Press, 2011.

[9] Johnson, C. R., Jr., and W. A. Sethares, *Telecommunications Breakdown: Concepts of Communication Transmitted via Software-Defined Radio*. Prentice Hall, 2004.

[10] Cavicchi, T. J., Digital Signal Processing. John Wiley and Sons, 2000.

[11] Oppenheim, A. V., and R. W., Schafer, *Digital Signal Processing*, Prentice-Hall, 1975.

[12] Analog Devices AD7177 Product Page, 2016 http://www.analog.com/AD7177-2.

[13] Analog Devices AD9208 Product Page, 2017 http://www.analog.com/AD9208.

[14] Smith, S., *The Scientist and Engineer's Guide to Digital Signal Processing*, San Diego: California TechnicalPublishing, 1997.

[15] Jeffrey, A., and D. Zwillinger, *Table of Integrals, Series, and Products*, Seventh edition, San Diego: Academic Press, 2007.

[16] Hogenauer, E. B., "An Economical Class of Digital Filters for Decimation and Interpolation," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 29, No. 2, 1981, pp. 155–162, 1981.

[17] Lyons, R. G., Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook, Piscataway, NJ: Wiley-IEEE Press, 2007.

[18] Kester, W., *The Data Conversion Handbook*, 2005, http://www.analog.com/en/education/education-library/data-conversion-handbook.html.

[19] Johnson, C. R., W. A. Sethares, and A. G. Klein, *Software Receiver Design: Build Your Own Digital Communications System in Five Easy Steps*, Cambridge, UK: Cambridge University Press, 2011.

[20] Wyglinski, A. M., M. Nekovee, and Y. T. Hou, *Cognitive Radio Communications and Networks: Principles and Practice*, Burlington, MA: Academic Press, 2009.

[21] Barry, J., E. A. Lee, and D. G. Messerschmitt, Digital Communication, Third Edition, Kluwer AcademicPress, 2004.