This Third Edition of Real-Time Digital Signal Processing: Fundamentals, Implementations and Applications has been updated to include the latest DSP applications and introduces new development tools. The software design process has been adjusted to enable readers to concentrate on learning DSP fundamentals and innovative applications by relaxing the intensive programming efforts, namely, the traditional DSP assembly coding efforts. The low-cost, portable TMS320C5505 ezdsp USB stick device is now used in this latest edition as the required development tool for hands-on experiments.

New features of this edition:
- Places renewed emphasis on C code experiments and reduces the exercises using assembly coding, effective use of C programming, fixed-point C-code, and intrinsics will become the main focus of the new edition, illustrated by step-by-step hands-on experiments for the complete design cycle.
- Updates to application areas to reflect latest advances such as speech coding techniques used for next-generation networks, speech codecs (such as wideband ITU G.722.2), graphic and parametric audio equalizers, several audio effects, 2-D discrete wavelet transform examples used in JPEG2000, variety of 2-D filter kernels, and fingerprint for image processing.
- Contains new addition of several projects as exercise problems that can be used as semester projects, as well as many new real-time experiments using TI libraries - the experiments are prepared with a flexible interface and are modular for readers to adapt and modify to create other useful applications from the provided basic programs.
- Consists of more MATLAB® experiments, such as filter design, transforms, image color space formatting and conversion, algorithm evaluation, prototyping for C-code architecture, and simulations to aid readers to learn DSP fundamentals.
- Includes supplementary material of program and data files for examples, applications, and experiments hosted on a companion website.

www.wiley.com/go/kuo_dsp
REAL-TIME DIGITAL SIGNAL PROCESSING
REAL-TIME DIGITAL SIGNAL PROCESSING
FUNDAMENTALS, IMPLEMENTATIONS AND APPLICATIONS

Third Edition

Sen M. Kuo
Northern Illinois University, USA

Bob H. Lee
Ittiam Systems, Inc., USA

Wenshun Tian
Sonus Networks, Inc., USA

WILEY
Contents

Preface xv

Acknowledgments xix

1 Introduction to Real-Time Digital Signal Processing 1
  1.1 Basic Elements of Real-Time DSP Systems 2
  1.2 Analog Interface
    1.2.1 Sampling 3
    1.2.2 Quantization and Encoding 7
    1.2.3 Smoothing Filters 8
    1.2.4 Data Converters 9
  1.3 DSP Hardware 10
    1.3.1 DSP Hardware Options 11
    1.3.2 Digital Signal Processors 13
    1.3.3 Fixed- and Floating-Point Processors 14
    1.3.4 Real-Time Constraints 15
  1.4 DSP System Design 16
    1.4.1 Algorithm Development 16
    1.4.2 Selection of DSP Hardware 18
    1.4.3 Software Development 19
    1.4.4 Software Development Tools 20
  1.5 Experiments and Program Examples 21
    1.5.1 Get Started with CCS and eZdsp 22
    1.5.2 C File I/O Functions 26
    1.5.3 User Interface for eZdsp 30
    1.5.4 Audio Playback Using eZdsp 35
    1.5.5 Audio Loopback Using eZdsp 38

Exercises 42

References 43

2 DSP Fundamentals and Implementation Considerations 44
  2.1 Digital Signals and Systems 44
    2.1.1 Elementary Digital Signals 44
    2.1.2 Block Diagram Representation of Digital Systems 47
2.2 System Concepts
2.2.1 LTI Systems 48
2.2.2 The z-transform 52
2.2.3 Transfer Functions 54
2.2.4 Poles and Zeros 58
2.2.5 Frequency Responses 61
2.2.6 Discrete Fourier Transform 65

2.3 Introduction to Random Variables 66
2.3.1 Review of Random Variables 67
2.3.2 Operations of Random Variables 68

2.4 Fixed-Point Representations and Quantization Effects 72
2.4.1 Fixed-Point Formats 72
2.4.2 Quantization Errors 75
2.4.3 Signal Quantization 75
2.4.4 Coefficient Quantization 78
2.4.5 Roundoff Noise 78
2.4.6 Fixed-Point Toolbox 79

2.5 Overflow and Solutions 81
2.5.1 Saturation Arithmetic 81
2.5.2 Overflow Handling 82
2.5.3 Scaling of Signals 82
2.5.4 Guard Bits 83

2.6 Experiments and Program Examples 83
2.6.1 Overflow and Saturation Arithmetic 83
2.6.2 Function Approximations 86
2.6.3 Real-Time Signal Generation Using eZdsp 94

Exercises 99
References 101

3 Design and Implementation of FIR Filters 102
3.1 Introduction to FIR Filters 102
3.1.1 Filter Characteristics 102
3.1.2 Filter Types 104
3.1.3 Filter Specifications 106
3.1.4 Linear Phase FIR Filters 108
3.1.5 Realization of FIR Filters 110

3.2 Design of FIR Filters 114
3.2.1 Fourier Series Method 114
3.2.2 Gibbs Phenomenon 116
3.2.3 Window Functions 118
3.2.4 Design of FIR Filters Using MATLAB® 120
3.2.5 Design of FIR Filters Using the FDATool 122

3.3 Implementation Considerations 125
3.3.1 Quantization Effects in FIR Filters 125
3.3.2 MATLAB® Implementations 127
3.3.3 Floating-Point C Implementations 128
3.3.4 Fixed-Point C Implementations 129
### 3.4 Applications: Interpolation and Decimation Filters

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>Interpolation</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Decimation</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Sampling Rate Conversion</td>
</tr>
<tr>
<td>3.4.4</td>
<td>MATLAB® Implementations</td>
</tr>
</tbody>
</table>

### 3.5 Experiments and Program Examples

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.1</td>
<td>FIR Filtering Using Fixed-Point C</td>
</tr>
<tr>
<td>3.5.2</td>
<td>FIR Filtering Using C55xx Assembly Program</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Symmetric FIR Filtering Using C55xx Assembly Program</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Optimization Using Dual-MAC Architecture</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Real-Time FIR Filtering</td>
</tr>
<tr>
<td>3.5.6</td>
<td>Decimation Using C and Assembly Programs</td>
</tr>
<tr>
<td>3.5.7</td>
<td>Interpolation Using Fixed-Point C</td>
</tr>
<tr>
<td>3.5.8</td>
<td>Sampling Rate Conversion</td>
</tr>
<tr>
<td>3.5.9</td>
<td>Real-Time Sampling Rate Conversion</td>
</tr>
</tbody>
</table>

### Exercises
- 144

### References
- 147

### 4 Design and Implementation of IIR Filters

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Analog Systems</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Mapping Properties</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Characteristics of Analog Filters</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Frequency Transforms</td>
</tr>
<tr>
<td>4.2</td>
<td>Design of IIR Filters</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Bilinear Transform</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Filter Design Using the Bilinear Transform</td>
</tr>
<tr>
<td>4.3</td>
<td>Realization of IIR Filters</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Direct Forms</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Cascade Realizations</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Parallel Realizations</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Realization of IIR Filters Using MATLAB®</td>
</tr>
<tr>
<td>4.4</td>
<td>Design of IIR Filters Using MATLAB®</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Filter Design Using MATLAB®</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Frequency Transforms Using MATLAB®</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Filter Design and Realization Using the FDATool</td>
</tr>
<tr>
<td>4.5</td>
<td>Implementation Considerations</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Stability</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Finite-Precision Effects and Solutions</td>
</tr>
<tr>
<td>4.5.3</td>
<td>MATLAB® Implementations of IIR Filters</td>
</tr>
<tr>
<td>4.6</td>
<td>Practical Applications</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Recursive Resonators</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Recursive Quadrature Oscillators</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Parametric Equalizers</td>
</tr>
<tr>
<td>4.7</td>
<td>Experiments and Program Examples</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Direct-Form I IIR Filter Using Floating-Point C</td>
</tr>
</tbody>
</table>
5 Frequency Analysis and the Discrete Fourier Transform 195

5.1 Fourier Series and Fourier Transform 195
   5.1.1 Fourier Series 195
   5.1.2 Fourier Transform 197

5.2 Discrete Fourier Transform 198
   5.2.1 Discrete-Time Fourier Transform 198
   5.2.2 Discrete Fourier Transform 200
   5.2.3 Important Properties 202

5.3 Fast Fourier Transforms 205
   5.3.1 Decimation-in-Time 206
   5.3.2 Decimation-in-Frequency 208
   5.3.3 Inverse Fast Fourier Transform 209

5.4 Implementation Considerations 210
   5.4.1 Computational Issues 210
   5.4.2 Finite-Precision Effects 210
   5.4.3 MATLAB® Implementations 211
   5.4.4 Fixed-Point Implementation Using MATLAB® 212

5.5 Practical Applications 214
   5.5.1 Spectral Analysis 214
   5.5.2 Spectral Leakage and Resolution 215
   5.5.3 Power Spectral Density 219
   5.5.4 Convolution 222

5.6 Experiments and Program Examples 224
   5.6.1 DFT Using Floating-Point C 224
   5.6.2 DFT Using the C55xx Assembly Program 226
   5.6.3 FFT Using Floating-Point C 227
   5.6.4 FFT Using Fixed-Point C with Intrinsics 227
   5.6.5 Experiment with the FFT and IFFT 231
   5.6.6 FFT Using the C55xx Hardware Accelerator 231
   5.6.7 Real-Time FFT Using the C55xx Hardware Accelerator 233
   5.6.8 Fast Convolution Using the Overlap–Add Technique 234
   5.6.9 Real-Time Fast Convolution 235

Exercises 236
References 238
6 Adaptive Filtering 239
6.1 Introduction to Random Processes 239
6.2 Adaptive Filters 243
   6.2.1 Introduction to Adaptive Filtering 243
   6.2.2 Performance Function 244
   6.2.3 Method of Steepest Descent 248
   6.2.4 LMS Algorithm 249
   6.2.5 Modified LMS Algorithms 251
6.3 Performance Analysis 252
   6.3.1 Stability Constraint 252
   6.3.2 Convergence Speed 253
   6.3.3 Excess Mean-Square Error 254
   6.3.4 Normalized LMS Algorithm 254
6.4 Implementation Considerations 255
   6.4.1 Computational Issues 255
   6.4.2 Finite-Precision Effects 256
   6.4.3 MATLAB® Implementations 257
6.5 Practical Applications 259
   6.5.1 Adaptive System Identification 259
   6.5.2 Adaptive Prediction 262
   6.5.3 Adaptive Noise Cancellation 264
   6.5.4 Adaptive Inverse Modeling 267
6.6 Experiments and Program Examples 268
   6.6.1 LMS Algorithm Using Floating-Point C 268
   6.6.2 Leaky LMS Algorithm Using Fixed-Point C 270
   6.6.3 Normalized LMS Algorithm Using Fixed-Point C and Intrinsics 270
   6.6.4 Delayed LMS Algorithm Using Assembly Program 274
   6.6.5 Experiment of Adaptive System Identification 275
   6.6.6 Experiment of Adaptive Predictor 276
   6.6.7 Experiment of Adaptive Channel Equalizer 277
   6.6.8 Real-Time Adaptive Prediction Using eZdsp 279
Exercises 280
References 282

7 Digital Signal Generation and Detection 283
7.1 Sine Wave Generators 283
   7.1.1 Lookup Table Method 283
   7.1.2 Linear Chirp Signal 286
7.2 Noise Generators 288
   7.2.1 Linear Congruential Sequence Generator 288
   7.2.2 Pseudo-random Binary Sequence Generator 289
   7.2.3 White, Color, and Gaussian Noise 290
7.3 DTMF Generation and Detection 291
   7.3.1 DTMF Generator 291
   7.3.2 DTMF Detection 292

Exercises 280
References 282
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4 Experiments and Program Examples</td>
<td>298</td>
</tr>
<tr>
<td>7.4.1 Sine Wave Generator Using Table Lookup</td>
<td>298</td>
</tr>
<tr>
<td>7.4.2 Siren Generator Using Table Lookup</td>
<td>299</td>
</tr>
<tr>
<td>7.4.3 DTMF Generator</td>
<td>299</td>
</tr>
<tr>
<td>7.4.4 DTMF Detection Using Fixed-Point C</td>
<td>300</td>
</tr>
<tr>
<td>7.4.5 DTMF Detection Using Assembly Program</td>
<td>301</td>
</tr>
<tr>
<td>Exercises</td>
<td>302</td>
</tr>
<tr>
<td>References</td>
<td>302</td>
</tr>
<tr>
<td>8 Adaptive Echo Cancellation</td>
<td>304</td>
</tr>
<tr>
<td>8.1 Introduction to Line Echoes</td>
<td>304</td>
</tr>
<tr>
<td>8.2 Adaptive Line Echo Canceler</td>
<td>306</td>
</tr>
<tr>
<td>8.2.1 Principles of Adaptive Echo Cancellation</td>
<td>306</td>
</tr>
<tr>
<td>8.2.2 Performance Evaluation</td>
<td>308</td>
</tr>
<tr>
<td>8.3 Practical Considerations</td>
<td>309</td>
</tr>
<tr>
<td>8.3.1 Pre-whitening of Signals</td>
<td>309</td>
</tr>
<tr>
<td>8.3.2 Delay Estimation</td>
<td>309</td>
</tr>
<tr>
<td>8.4 Double-Talk Effects and Solutions</td>
<td>312</td>
</tr>
<tr>
<td>8.5 Nonlinear Processor</td>
<td>314</td>
</tr>
<tr>
<td>8.5.1 Center Clipper</td>
<td>314</td>
</tr>
<tr>
<td>8.5.2 Comfort Noise</td>
<td>315</td>
</tr>
<tr>
<td>8.6 Adaptive Acoustic Echo Cancellation</td>
<td>315</td>
</tr>
<tr>
<td>8.6.1 Acoustic Echoes</td>
<td>316</td>
</tr>
<tr>
<td>8.6.2 Acoustic Echo Canceler</td>
<td>317</td>
</tr>
<tr>
<td>8.6.3 Subband Implementations</td>
<td>318</td>
</tr>
<tr>
<td>8.6.4 Delay-Free Structures</td>
<td>321</td>
</tr>
<tr>
<td>8.6.5 Integration of Acoustic Echo Cancellation with Noise Reduction</td>
<td>321</td>
</tr>
<tr>
<td>8.6.6 Implementation Considerations</td>
<td>322</td>
</tr>
<tr>
<td>8.7 Experiments and Program Examples</td>
<td>323</td>
</tr>
<tr>
<td>8.7.1 Acoustic Echo Canceler Using Floating-Point C</td>
<td>323</td>
</tr>
<tr>
<td>8.7.2 Acoustic Echo Canceler Using Fixed-Point C with Intrinsics</td>
<td>325</td>
</tr>
<tr>
<td>8.7.3 Integration of AEC and Noise Reduction</td>
<td>326</td>
</tr>
<tr>
<td>Exercises</td>
<td>328</td>
</tr>
<tr>
<td>References</td>
<td>329</td>
</tr>
<tr>
<td>9 Speech Signal Processing</td>
<td>330</td>
</tr>
<tr>
<td>9.1 Speech Coding Techniques</td>
<td>330</td>
</tr>
<tr>
<td>9.1.1 Speech Production Model Using LPC</td>
<td>331</td>
</tr>
<tr>
<td>9.1.2 CELP Coding</td>
<td>332</td>
</tr>
<tr>
<td>9.1.3 Synthesis Filter</td>
<td>334</td>
</tr>
<tr>
<td>9.1.4 Excitation Signals</td>
<td>337</td>
</tr>
<tr>
<td>9.1.5 Perceptual Based Minimization Procedure</td>
<td>340</td>
</tr>
<tr>
<td>9.1.6 Voice Activity Detection</td>
<td>341</td>
</tr>
<tr>
<td>9.1.7 ACELP Codecs</td>
<td>343</td>
</tr>
<tr>
<td>9.2 Speech Enhancement</td>
<td>350</td>
</tr>
<tr>
<td>9.2.1 Noise Reduction Techniques</td>
<td>350</td>
</tr>
</tbody>
</table>
9.2.2 Short-Time Spectrum Estimation 351
9.2.3 Magnitude Spectrum Subtraction 353
9.3 VoIP Applications 355
  9.3.1 Overview of VoIP 355
  9.3.2 Discontinuous Transmission 357
  9.3.3 Packet Loss Concealment 358
  9.3.4 Quality Factors of Media Stream 359
9.4 Experiments and Program Examples 360
  9.4.1 LPC Filter Using Fixed-Point C with Intrinsics 360
  9.4.2 Perceptual Weighting Filter Using Fixed-Point C with Intrinsics 364
  9.4.3 VAD Using Floating-Point C 365
  9.4.4 VAD Using Fixed-Point C 367
  9.4.5 Speech Encoder with Discontinuous Transmission 368
  9.4.6 Speech Decoder with CNG 369
  9.4.7 Spectral Subtraction Using Floating-Point C 370
  9.4.8 G.722.2 Using Fixed-Point C 372
  9.4.9 G.711 Companding Using Fixed-Point C 373
  9.4.10 Real-Time G.711 Audio Loopback 373
Exercises 374
References 375

10 Audio Signal Processing 377
10.1 Introduction 377
10.2 Audio Coding 378
  10.2.1 Basic Principles 378
  10.2.2 Frequency-Domain Coding 383
  10.2.3 Lossless Audio Coding 386
  10.2.4 Overview of MP3 387
10.3 Audio Equalizers 389
  10.3.1 Graphic Equalizers 389
  10.3.2 Parametric Equalizers 391
10.4 Audio Effects 397
  10.4.1 Sound Reverberation 398
  10.4.2 Time Stretch and Pitch Shift 399
  10.4.3 Modulated and Mixed Sounds 401
  10.4.4 Spatial Sounds 409
10.5 Experiments and Program Examples 411
  10.5.1 MDCT Using Floating-Point C 411
  10.5.2 MDCT Using Fixed-Point C and Intrinsics 415
  10.5.3 Pre-echo Effects 416
  10.5.4 MP3 Decoding Using Floating-Point C 419
  10.5.5 Real-Time Parametric Equalizer Using eZdsp 421
  10.5.6 Flanger Effects 422
  10.5.7 Real-Time Flanger Effects Using eZdsp 423
  10.5.8 Tremolo Effects 424
  10.5.9 Real-Time Tremolo Effects Using eZdsp 425
10.5.10 Spatial Sound Effects
10.5.11 Real-Time Spatial Effects Using eZdsp
Exercises
References

11 Introduction to Digital Image Processing

11.1 Digital Images and Systems
11.1.1 Digital Images
11.1.2 Digital Image Systems
11.2 Color Spaces
11.3 YC_bC_r Sub-sampled Color Space
11.4 Color Balance and Correction
11.4.1 Color Balance
11.4.2 Color Correction
11.4.3 Gamma Correction
11.5 Histogram Equalization
11.6 Image Filtering
11.7 Fast Convolution
11.8 Practical Applications
11.8.1 DCT and JPEG
11.8.2 Two-Dimensional DCT
11.8.3 Fingerprint
11.8.4 Discrete Wavelet Transform
11.9 Experiments and Program Examples
11.9.1 YC_bC_r to RGB Conversion
11.9.2 White Balance
11.9.3 Gamma Correction and Contrast Adjustment
11.9.4 Image Filtering
11.9.5 DCT and IDCT
11.9.6 Image Processing for Fingerprints
11.9.7 The 2-D Wavelet Transform
Exercises
References

Appendix A Some Useful Formulas and Definitions
A.1 Trigonometric Identities
A.2 Geometric Series
A.3 Complex Variables
A.4 Units of Power
References

Appendix B Software Organization and List of Experiments

Appendix C Introduction to the TMS320C55xx Digital Signal Processor
C.1 Introduction
C.2 TMS320C55xx Architecture
C.2.1 Architecture Overview 490
C.2.2 On-Chip Memories 494
C.2.3 Memory-Mapped Registers 495
C.2.4 Interrupts and Interrupt Vector 498

C.3 TMS320C55xx Addressing Modes 498
C.3.1 Direct Addressing Modes 501
C.3.2 Indirect Addressing Modes 502
C.3.3 Absolute Addressing Modes 505
C.3.4 MMR Addressing Mode 505
C.3.5 Register Bits Addressing Mode 506
C.3.6 Circular Addressing Mode 507

C.4 TMS320C55xx Assembly Language Programming 508
C.4.1 Arithmetic Instructions 508
C.4.2 Logic and Bit Manipulation Instructions 509
C.4.3 Move Instruction 509
C.4.4 Program Flow Control Instructions 510
C.4.5 Parallel Execution 514
C.4.6 Assembly Directives 516
C.4.7 Assembly Statement Syntax 518

C.5 C Programming for TMS320C55xx 520
C.5.1 Data Types 520
C.5.2 Assembly Code Generation by C Compiler 520
C.5.3 Compiler Keywords andPragma Directives 522

C.6 Mixed C and Assembly Programming 525

C.7 Experiments and Program Examples 529
C.7.1 Examples 529
C.7.2 Assembly Program 530
C.7.3 Multiplication 530
C.7.4 Loops 531
C.7.5 Modulo Operator 532
C.7.6 Use Mixed C and Assembly Programs 533
C.7.7 Working with AIC3204 533
C.7.8 Analog Input and Output 534

References 535

Index 537
Preface

In recent years, real-time digital signal processing (DSP) using general-purpose DSP processors has provided an effective way to design and implement DSP systems for practical applications. Many companies are actively engaged in real-time DSP research for developing new applications. The study of real-time DSP applications has been and will continue to be a challenging field for students, engineers, and researchers. It is important to master not only the theory, but also the skill of system design and implementation techniques.

Since the publication of the first edition of the book entitled Real-Time Digital Signal Processing in 2001 and the second edition in 2006, the use of digital signal processors has penetrated into a much wider range of applications. This has led to curriculum changes in many universities to offer new real-time DSP courses that focus on implementations and applications, as well as enhancing the traditional theoretical lectures with hands-on real-time experiments. In the meantime, advances in new processors and development tools constantly demand up-to-date books in order to keep up with the rapid evolution of DSP developments, applications, and software updates. We intend with the third edition of this book to integrate the theory, design, applications, and implementations using hands-on experiments for the effective learning of real-time DSP technologies.

This book presents fundamental DSP principles along with many MATLAB examples and emphasizes real-time applications using hands-on experiments. The book is designed to be used as a textbook for senior undergraduate/graduate students. The prerequisites for this book are concepts of signals and systems, basic microprocessor architecture, and MATLAB and C programming. These topics are usually covered at the sophomore and junior levels in electrical and computer engineering, computer science, and other related science and engineering fields. This book can also serve as a reference for engineers, algorithm developers, and embedded system designers and programmers to learn DSP principles and implementation techniques for developing practical applications. We use a hands-on approach by conducting experiments and evaluating the results in order to help readers to understand the principles behind complicated theories. A list of textbooks and technical papers with mathematical proofs are provided as references at the end of each chapter for those who are interested in going beyond the coverage of this book.

The major aims and changes for this third edition are summarized as follows:

1. Focus on practical applications and provide step-by-step hands-on experiments for the complete design cycle starting from the evaluation of algorithms using MATLAB to the implementation using floating-point C programming, and updated to fixed-point C
programming, and software optimization using mixed C and assembly programming with C intrinsics and assembly routines for fixed-point digital signal processors. This methodology enables readers to concentrate on learning DSP fundamentals and innovative applications by relaxing the intensive programming efforts, especially the time-consuming assembly programming.

2. Enhance many examples and hands-on experiments to make the DSP principles more interesting and interactive with real-world applications. All the C and assembly programs are carefully updated using the most recent versions of development tools, the Code Composer Studio and the low-cost TMS320C5505 (a member of C55xx family) eZdsp USB stick, for real-time experiments. Due to its affordable cost and portability, the eZdsp enables students, engineers, professionals, and hobbyists to conduct DSP experiments and projects at more convenient locations instead of in traditional laboratories. This new hardware tool is widely used by universities and industrial organizations to replace the older and more expensive development tools.

3. Add attractive and challenging DSP applications such as speech coding techniques for next generation networks and cell (mobile) phones; audio coding for portable players; several audio effects including spatial sounds, graphic and parametric audio equalizers for music, and audio post-recording effects; two-dimensional discrete wavelet transform for JPEG2000; image filtering for special effects; and fingerprint image processing. Also develop real-time experiments with modular designs and flexible interfaces such that the software may serve as prototyping programs to create other related applications.

4. Organize chapters in a more flexible and logical manner. Some related applications are grouped together. We also removed some topics, such as channel coding techniques, that may not be suitable for a semester-long course. The hardware-dependent topics in the second edition have been greatly simplified and presented here as an appendix for readers who are required or interested to learn about TMS320C55xx architecture and assembly programming. All of these changes are made intentionally for the purpose of focusing on the fundamental DSP principles with enhanced hands-on experiments for practical applications.

Many DSP algorithms and applications are available in MATLAB\textsuperscript{®} and floating-point C programs. This book provides a systematic software development process for converting these programs to fixed-point C and optimizing them for implementation on fixed-point processors. To effectively illustrate DSP principles and applications, MATLAB\textsuperscript{®} is used for the demonstration, design, and analysis of algorithms. This development stage is followed by floating-point and fixed-point C programming for implementing DSP algorithms. Finally, we integrate the CCS with the C5505 eZdsp for hands-on experiments. To utilize the advanced architecture and instruction set for efficient software development and maintenance, we emphasize using mixed C and assembly programs for real-time applications.

This book is organized into two parts: principles and applications. The first part (Chapters 1–6) introduces DSP principles, algorithms, analysis methods, and implementation considerations. Chapter 1 reviews the fundamentals of real-time DSP functional blocks, DSP hardware options, fixed- and floating-point DSP devices, real-time constraints, and algorithm and software development processes. Chapter 2 presents fundamental DSP concepts and practical considerations for the implementation of DSP algorithms. The theory, design, analysis, implementation, and application of finite impulse response and infinite impulse response filters are presented in Chapters 3 and 4, respectively.
The concepts and considerations of using the discrete Fourier transform for frequency analysis, and the implementation and application of the fast Fourier transform, are introduced in Chapter 5. Basic principles of adaptive signal processing with many practical considerations for applications are presented in Chapter 6.

The second part (Chapters 7–11) introduces several important DSP applications that have played important roles in the realization of modern real-world systems and devices. These selected applications include digital signal generation and dual-tone multi-frequency (DTMF) detection in Chapter 7; adaptive echo cancellation especially for VoIP and hands-free phone applications in Chapter 8; speech processing algorithms including speech enhancement and coding techniques for mobile phones in Chapter 9; audio signal processing including audio effects, equalizers, and coding methods for portable players in Chapter 10; and image processing fundamentals for applications including JPEG2000 and fingerprints in Chapter 11. Finally, Appendix A summarizes some useful formulas used for the derivation of equations and solving exercise problems in the book, and Appendix C introduces the architecture and assembly programming of the TMS320C55xx for readers who are interested in these topics.

As with any book attempting to capture the state of the art at a given time, there will certainly be updates that are necessitated by the rapidly evolving developments in this dynamic field. We hope that this book can serve as a guide for what has already come and as an inspiration for what will follow.

**Software Availability**

This book utilizes various MATLAB®, floating-point and fixed-point C, and TMS320C55xx assembly programs in examples, experiments, and applications. These programs along with many data files are available in the companion software package from the Wiley website (http://www.wiley.com/go/kuo_dsp). The directory and the subdirectory structure and names of these software programs and data files are explained and listed in Appendix B. The software is required for conducting the experiments presented in the last section of each chapter and Appendix C, and can enhance the understanding of DSP principles. The software can also be modified to serve as prototypes for speeding up other practical uses.
Acknowledgments

We are grateful to Cathy Wicks and Gene Frantz of Texas Instruments, and to Naomi Fernandes and Courtney Esposito of MathWorks, for providing us with the support needed to write this book. We would like to thank several individuals at John Wiley & Sons, Ltd for their support of this project: Alexandra King, Commissioning Editor; Liz Wingett, Project Editor; and Richard Davies, Senior Project Editor. We would also like to thank the staff at John Wiley & Sons, Ltd for the final preparation of this book. Our thanks also go to Hui Tian for creating the special audio clips used in the examples and experiments. Finally, we thank our families for their endless love, encouragement, patience, and understanding they have shown since we began our work on the first edition in the late 1990s.

Sen M. Kuo, Bob H. Lee, and Wenshun Tian
Introduction to Real-Time Digital Signal Processing

Signals can be classified into three categories: continuous-time (analog) signals, discrete-time signals, and digital signals. The signals that we encounter daily are mostly analog signals. These signals are defined continuously in time, have infinite resolution of amplitude values, and can be processed using analog electronics containing both active and passive circuit elements. Discrete-time signals are defined only at a particular set of time instances, thus they can be represented as a sequence of numbers that have a continuous range of values. Digital signals have discrete values in both time and amplitude, thus they can be stored and processed by computers or digital hardware. In this book, we focus on the design, implementation, and applications of digital systems for processing digital signals [1–6]. However, the theoretical analysis usually uses discrete-time signals and systems for mathematical convenience. Therefore, we use the terms “discrete-time” and “digital” interchangeably.

Digital signal processing (DSP) is concerned with the digital representation of signals and the use of digital systems to analyze, modify, store, transmit, or extract information from these signals. In recent years, the rapid advancement in digital technologies has enabled the implementation of sophisticated DSP algorithms for real-time applications. DSP is now used not only in areas where analog methods were used previously, but also in areas where analog techniques are very difficult or impossible to apply.

There are many advantages in using digital techniques for signal processing rather than analog devices such as amplifiers, modulators, and filters. Some of the advantages of DSP systems over analog circuitry are summarized as follows:

1. **Flexibility.** Functions of a DSP system can be easily modified and upgraded with software that implements the specific operations. One can design a DSP system to perform a wide variety of tasks by executing different software modules. A digital device can be easily upgraded in the field through the on-board memory (e.g., flash memory) to meet new requirements, add new features, or enhance its performance.
2. **Reproducibility.** The functions of a DSP system can be repeated precisely from one unit to another. In addition, by using DSP techniques, digital signals can be stored, transferred, or reproduced many times without degrading the quality. By contrast, analog circuits will not have the same characteristics even if they are built following identical specifications, due to analog component tolerances.

3. **Reliability.** The memory and logic of DSP hardware do not deteriorate with age. Therefore, the performance of DSP systems will not drift with changing environmental conditions or aged electronic components as their analog counterparts do.

4. **Complexity.** DSP allows sophisticated applications such as speech recognition to be implemented using low-power and lightweight portable devices. Furthermore, there are some important signal processing algorithms such as image compression and recognition, data transmission and storage, and audio compression, which can only be performed using DSP systems.

With the rapid evolution in semiconductor technologies, DSP systems have lower overall cost compared to analog systems for most applications. DSP algorithms can be developed, analyzed, and simulated using high-level language software such as C and MATLAB. The performance of the algorithms can be verified using low-cost, general-purpose computers. Therefore, DSP systems are relatively easy to design, develop, analyze, simulate, test, and maintain.

There are some limitations associated with DSP. For example, the bandwidth of a DSP system is limited by the sampling rate. Also, most of the DSP algorithms are implemented using a fixed number of bits with limited precision and dynamic range, resulting in undesired quantization and arithmetic errors.

### 1.1 Basic Elements of Real-Time DSP Systems

There are two types of DSP applications: non-real-time and real-time. Non-real-time signal processing involves manipulating signals that have already been stored in digital form. This may or may not represent a current action, and the processing result is not a function of real time. Real-time signal processing places stringent demands on DSP hardware and software design to complete predefined tasks within a given timeframe. This section reviews the fundamental functional blocks of real-time DSP systems.

The basic functional blocks of DSP systems are illustrated in Figure 1.1, where a real-world analog signal is converted to a digital signal, processed by DSP hardware, and converted back

![Figure 1.1](image_url)
to an analog signal. For some applications, the input signal may be already in digital form and/or the output data may not need to be converted to an analog signal, for example, the processed digital information may be stored in memory for later use. In other applications, DSP systems may be required to generate signals digitally, such as speech synthesis and signal generators.

1.2 Analog Interface

In this book, a time-domain signal is denoted with a lowercase letter. For example, $x(t)$ in Figure 1.1 is used to name an analog signal of $x$ which is a function of time $t$. The time variable $t$ and the amplitude of $x(t)$ take on a continuum of values between $-\infty$ and $\infty$. For this reason we say $x(t)$ and $y(t)$ are continuous-time (or analog) signals. The signals $x(n)$ and $y(n)$ in Figure 1.1 depict digital signals which have values only at time instant (or index) $n$. In this section, we first discuss how to convert analog signals into digital signals. The process of converting an analog signal to a digital signal is called the analog-to-digital (A/D) conversion, usually performed by an A/D converter (ADC).

The purpose of A/D conversion is to convert the analog signal to digital form for processing by digital hardware. As shown in Figure 1.1, the analog signal $x(t)$ is picked up by an appropriate electronic sensor that converts pressure, temperature, or sound into electrical signals. For example, a microphone can be used to pick up speech signals. The sensor output signal $x_0(t)$ is amplified by an amplifier with a gain of value $g$ to produce the amplified signal

$$x(t) = gx'(t). \quad (1.1)$$

The gain value $g$ is determined such that $x(t)$ has a dynamic range that matches the ADC used by the system. If the peak-to-peak voltage range of the ADC is $\pm 2$ volts (V), then $g$ may be set so that the amplitude of signal $x(t)$ to the ADC is within $\pm 2$ V. In practice, it is very difficult to set an appropriate fixed gain because the level of $x'(t)$ may be unknown and changing with time, especially for signals with larger dynamic ranges such as human speech. Therefore, many practical systems use digital automatic gain control algorithms to determine and update the gain value $g$ based on the statistics of the input signal $x'(t)$.

Once the digital signal has been processed by the DSP hardware, the result $y(n)$ is still in digital form. In many DSP applications, we have to convert the digital signal $y(n)$ back to the analog signal $y(t)$ before it can be applied to appropriate analog devices. This process is called the digital-to-analog (D/A) conversion, typically performed by a D/A converter (DAC). One example is a digital audio player, in which the audio music signals are stored in digital format. An audio player reads the encoded digital audio data from the memory and reconstructs the corresponding analog waveform for playback.

The system shown in Figure 1.1 is a real-time system if the signal to the ADC is continuously sampled and processed by the DSP hardware at the same rate. In order to maintain real-time processing, the DSP hardware must perform all required operations within the fixed time, and present the output sample to the DAC before the arrival of the next sample from the ADC.

1.2.1 Sampling

As shown in Figure 1.1, the ADC converts the analog signal $x(t)$ into the digital signal $x(n)$. The A/D conversion, commonly referred to as digitization, consists of the sampling (digitization in time) and quantization (digitization in amplitude) processes as illustrated in Figure 1.2. The basic sampling function can be carried out with an ideal “sample-and-hold”
A/D converter

Ideal sampler  Quantizer

$x(t)$  $x(nT)$  $x(n)$

**Figure 1.2** Block diagram of an ADC

circuit, which maintains the sampled signal level until the next sample is taken. The quantization process approximates the waveform by assigning a number to represent its value for each sample. Therefore, the A/D conversion performs the following steps:

1. The signal $x(t)$ is sampled at uniformly spaced time instants $nT$, where $n$ is a positive integer and $T$ is the sampling period in seconds. This sampling process converts an analog signal into a discrete-time signal $x(nT)$ with continuous amplitude value.
2. The amplitude of each discrete-time sample $x(nT)$ is quantized into one of $2^B$ levels, where $B$ is the number of bits used to represent each sample. The discrete amplitude levels are represented (or encoded) into binary words $x(n)$ with the fixed wordlength $B$.

The reason for making this distinction is that these two processes introduce different distortions. The sampling process causes aliasing or folding distortion, while the encoding process results in quantization noise. As shown in Figure 1.2, the sampler and quantizer are integrated on the same chip. However, a high-speed ADC typically requires an external sample-and-hold device.

An ideal sampler can be considered as a switch that periodically opens and closes every $T$ seconds. The sampling period is defined as

$$T = \frac{1}{f_s}, \quad (1.2)$$

where $f_s$ is the sampling frequency in hertz (Hz) or sampling rate in samples per second. The intermediate signal $x(nT)$ is a discrete-time signal with continuous value (a number with infinite precision) at discrete time $nT$, $n = 0, 1, \ldots, \infty$, as illustrated in Figure 1.3. The

**Figure 1.3** Sampling of analog signal $x(t)$ and the corresponding discrete-time signal $x(nT)$
analog signal $x(t)$ is continuous in both time and amplitude. The sampled discrete-time signal $x(nT)$ is continuous in amplitude, but defined only at discrete sampling instants $t = nT$.

In order to represent the analog signal $x(t)$ accurately by the discrete-time signal $x(nT)$, the sampling frequency $f_s$ must be at least twice the maximum frequency component $f_M$ in the analog signal $x(t)$. That is,

$$f_s \geq 2f_M,$$

where $f_M$ is also called the bandwidth of the bandlimited signal $x(t)$. This is Shannon’s sampling theorem, which states that when the sampling frequency is greater than or equal to twice the highest frequency component contained in the analog signal, the original analog signal $x(t)$ can be perfectly reconstructed from the uniformly sampled discrete-time signal $x(nT)$.

The minimum sampling rate, $f_s = 2f_M$, is called the Nyquist rate. The frequency, $f_N = f_s/2$, is called the Nyquist frequency or folding frequency. The frequency interval, $[-f_s/2, f_s/2]$, is called the Nyquist interval. When the analog signal is sampled at $f_s$, frequency components higher than $f_s/2$ will fold back into the frequency range $[0, f_s/2]$. The folded back frequency components overlap with the original frequency components in the same frequency range, resulting in the corrupted signal. Therefore, the original analog signal cannot be recovered from the folded digital samples. This undesired effect is known as aliasing.

**Example 1.1**

Consider two sine waves of frequencies $f_1 = 1$ Hz and $f_2 = 5$ Hz that are sampled at $f_s = 4$ Hz, rather than at least 10 Hz according to the sampling theorem. The analog waveforms and the digital samples are illustrated in Figure 1.4(a), while their digital samples and reconstructed waveforms are illustrated in Figure 1.4(b). As shown in the figures, we can reconstruct the original waveform from the digital samples for the sine wave of frequency $f_1 = 1$ Hz. However, for the original sine wave of frequency $f_2 = 5$ Hz, the resulting digital samples are the same as $f_1 = 1$ Hz, thus the reconstructed signal is identical to the sine wave of frequency 1 Hz. Therefore, $f_1$ and $f_2$ are said to be aliased to one another, that is, they cannot be distinguished by their discrete-time samples.

Note that the sampling theorem assumes the signal is bandlimited by $f_M$. For many practical applications, the analog signal $x(t)$ may have significant frequency components outside the highest frequency of interest, or may contain noise with a wider bandwidth. In some applications, the sampling rate is predetermined by given specifications. For example, most voice communication systems define the sampling rate of 8 kHz (kilohertz). Unfortunately, the frequency components in typical speech can be much higher than 4 kHz. To guarantee that the sampling theorem is satisfied, we must eliminate the frequency components above the Nyquist frequency. This can be done by using an antialiasing filter which is an analog lowpass filter with the cutoff frequency bounded by

$$f_c \leq \frac{f_s}{2}.$$  

Ideally, an antialiasing filter should remove all frequency components above the Nyquist frequency. In many practical systems, a bandpass filter is preferred to remove frequency
components above the Nyquist frequency, as well as to eliminate undesired DC offset, 60 Hz hum, or other low-frequency noises. For example, a bandpass filter with a passband from 300 to 3400 Hz is widely used in telecommunication systems to attenuate the signals whose frequencies lie outside this passband.

\[ x(t), f_1 = 1 \text{ Hz} \]

\[ x(nT), f_1 = 1 \text{ Hz} \]

\[ x(t), f_2 = 5 \text{ Hz} \]

\[ x(nT), f_2 = 5 \text{ Hz} \]

Figure 1.4 Example of the aliasing phenomenon

Example 1.2

The frequency range of signals is large, from approximately gigahertz (GHz) in radar down to fractions of hertz in instrumentation. For a specific application with given sampling rate, the sampling period can be determined by (1.2). For example, some real-world applications use the following sampling frequencies and periods:

1. In International Telecommunication Union (ITU) speech coding/decoding standards ITU-T G.729 [7] and G.723.1 [8], the sampling rate is \( f_s = 8 \text{ kHz} \), thus the sampling period \( T = 1/8000 \text{ seconds} = 125 \text{ µs} \) (microseconds). Note that \( 1 \text{ µs} = 10^{-6} \text{ seconds}. \)
2. Wideband telecommunication speech coding standards, such as ITU-T G.722 [9] and G.722.2 [10], use the sampling rate of \( f_s = 16 \text{ kHz} \), thus \( T = 1/16000 \text{ seconds} = 62.5 \text{ µs} \).
3. High-fidelity audio compression standards, such as MPEG-2 (Moving Picture Experts Group) [11], AAC (Advanced Audio Coding), MP3 (MPEG-1 layer 3) [12] audio, and...
Dolby AC-3, support the sampling rate of $f_s = 48$ kHz, thus $T = 1/48000$ seconds = 20.833 µs. The sampling rate for MPEG-2 AAC can be as high as 96 kHz.

The speech coding algorithms will be discussed in Chapter 9 and the audio coding techniques will be introduced in Chapter 10.

### 1.2.2 Quantization and Encoding

In previous sections, we assumed that the sample values $x(nT)$ are represented exactly using an infinite number of bits (i.e., $B \to \infty$). We now discuss the quantization and encoding processes for representing the sampled discrete-time signal $x(nT)$ by a binary number with a finite number of bits. If the wordlength of an ADC is $B$ bits, there are $2^B$ different values (levels) that can be used to represent a digital sample $x(n)$. If $x(nT)$ lies in between two quantization levels, it will either be rounded or truncated to produce $x(n)$. Rounding assigns to $x(nT)$ the value of the nearest quantization level while truncation replaces $x(nT)$ by the value of the level below it. Since rounding produces a less biased representation of the true value, it is widely used by ADCs. Therefore, quantization is a process that represents a continuous-valued sample $x(nT)$ with its nearest level that corresponds to the digital signal $x(n)$.

For example, 2 bits define four equally spaced levels (00, 01, 10, and 11), which can be used to classify the signal into the four subranges illustrated in Figure 1.5. In this figure, the open circles represent the discrete-time signal $x(nT)$, and the solid circles the digital signal $x(n)$. The spacing between two consecutive quantization levels is called the quantization width, step, or resolution. A uniform quantizer has the same spacing between these levels. For uniform quantization, the resolution is determined by dividing the full-scale range by the total number of quantization levels, $2^B$.

In Figure 1.5, the difference between the quantized number and the original value is defined as the quantization error, which appears as noise in the output of the converter. Thus, the quantization error is also called the quantization noise, which is assumed to be a random noise. If a $B$-bit quantizer is used, the signal-to-quantization-noise ratio (SQNR) is approximated by the following equation (to be derived in Chapter 2):

$$\text{SQNR} \approx 6B \text{ dB}. \quad (1.5)$$

![Figure 1.5 Digital samples using 2-bit quantizer](image)
In practice, the achievable SQNR will be less than this theoretical value due to imperfections in the fabrication of converters. Nevertheless, Equation (1.5) provides a simple guideline to determine the required bits for a given application. For each additional bit, a digital signal will have about 6 dB gain in SQNR. The problems of quantization noise and their solutions will be further discussed in Chapter 2.

**Example 1.3**

If the analog signal varies between 0 and 5 V, we have the resolutions and SQNRs for the following commonly used ADCs:

1. An 8-bit ADC with 256 \(2^8\) levels can only provide 19.5 mV resolution and 48 dB SQNR.
2. A 12-bit ADC has 4096 \(2^{12}\) levels of 1.22 mV resolution, and provides 72 dB SQNR.
3. A 16-bit ADC has 65536 \(2^{16}\) levels, and thus provides 76.294 \(\mu\)V resolution with 96 dB SQNR.

Obviously, using more bits results in more quantization levels (or finer resolution) and higher SQNR.

The dynamic range of speech signals is usually very large. If the uniform quantization scheme is adjusted for loud sounds, most of the softer sounds may be pressed into the same small values. This means that soft sounds may not be distinguishable. To solve this problem, we can use a quantizer with quantization level varying according to the signal amplitude. For example, if the signal has been compressed by a logarithm function, we can use a uniform level quantizer to perform non-uniform quantization by quantizing the logarithm-scaled signal. The compressed signal can be reconstructed by expanding it. The process of compression and expansion is called companding (compressing and expanding). The ITU-T G.711 \(\mu\)-law (used in North America and parts of Northeast Asia) and A-law (used in Europe and most of the rest of the world) schemes [13] are examples of using companding technology, which will be further discussed in Chapter 9.

As shown in Figure 1.1, the input signal to DSP hardware may be digital signals from other digital systems that use different sampling rates. The signal processing techniques called interpolation or decimation can be used to increase or decrease the sampling rates of the existing digital signals. Sampling rate changes may be required in many multi-rate DSP systems, for example, between the narrowband voice sampled at 8 kHz and wideband voice sampled at 16 kHz. The interpolation and decimation processes will be introduced in Chapter 3.

**1.2.3 Smoothing Filters**

Most commercial DACs are zero-order-hold devices, meaning they convert the input binary number to the corresponding voltage level and then hold that level for \(T\) seconds. Therefore, the DAC produces the staircase-shaped analog waveform \(y'(t)\) as shown by the solid line in Figure 1.6, which is a rectangular waveform with amplitude corresponding to the signal value with a duration of \(T\) seconds. Obviously, this staircase output signal contains many high-frequency components due to an abrupt change in signal levels. The reconstruction or