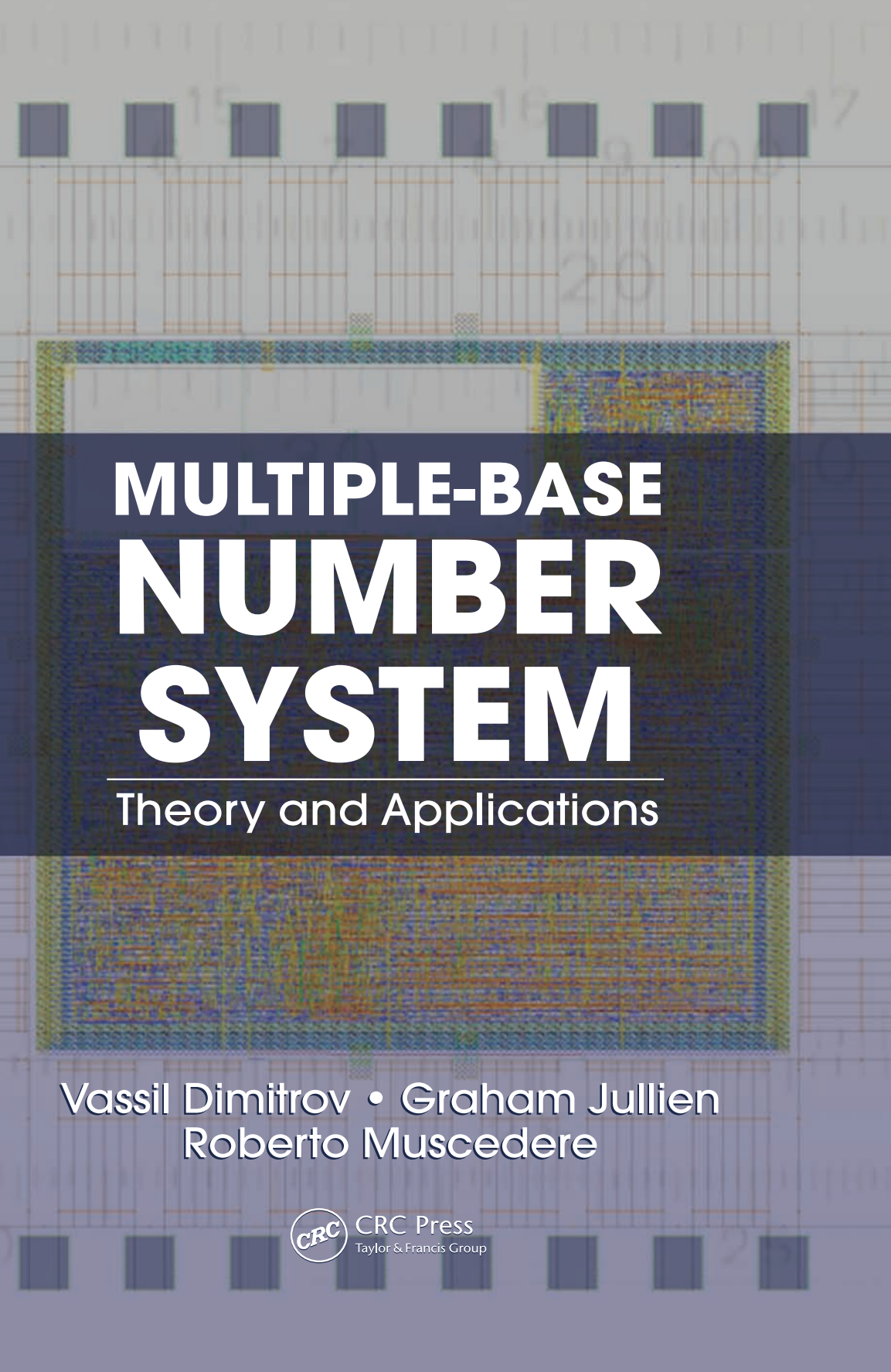
MULTIPLE-BASE
NUMBER
SYSTEM
Theory and Applications
Vassil Dimitrov • Graham Jullien
Roberto Muscedere
CRC Press
Taylor & Francis Group

MULTIPLE-BASE NUMBER SYSTEM

Theory and Applications

Circuits and Electrical Engineering Series

Series Editor
Wai-Kai Chen

MicroCMOS Design
Bang-Sup Song

Multiple-Base Number System: Theory and Applications
Vassil Dimitrov, Graham Jullien, and Roberto Muscedere

MULTIPLE-BASE NUMBER SYSTEM

Theory and Applications

Vassil Dimitrov • Graham Jullien
Roberto Muscedere

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20120104

International Standard Book Number-13: 978-1-4398-3047-5 (eBook - PDF)

Contents

Preface xiii
About the Authors xv

1. Technology, Applications, and Computation 1
1.1 Introduction 1
1.2 Ancient Roots 1
1.2.1 An Ancient Binary A/D Converter 1
1.2.2 Ancient Computational Aids 2
1.3 Analog or Digital? 3
1.3.1 An Analog Computer Fourier Analysis Tide Predictor 4
1.3.2 Babbage's Difference Engine 5
1.3.3 Pros and Cons 5
1.3.4 The Slide Rule: An Analog Logarithmic Processor 6
1.4 Where Are We Now? 8
1.4.1 Moore's Law 9
1.4.2 The Microprocessor and Microcontroller 9
1.4.3 Advances in Integrated Circuit Technology 10
1.5 Arithmetic and DSP 11
1.5.1 Data Stream Processing 11
1.5.2 Sampling and Conversion 12
1.5.3 Digital Filtering 14
1.6 Discrete Fourier Transform (DFT) 15
1.6.1 Fast Fourier Transform (FFT) 17
1.7 Arithmetic Considerations 19
1.7.1 Arithmetic Errors in DSP 19
1.8 Convolution Filtering with Exact Arithmetic 23
1.8.1 Number Theoretic Transforms (NTTs) 23
1.8.2 An NTT Example 24
1.8.3 Connections to Binary Arithmetic 25
1.9 Summary 27
References 28

2. The Double-Base Number System (DBNS) 31
2.1 Introduction 31
2.2 Motivation 32
2.3 The Double-Base Number System 32
2.3.1 The Original Unsigned Digit DBNS 32
2.3.2 Unsigned Digit DBNS Tabular Form 33
2.3.3 The General Signed Digit DBNS 34
2.3.4 Some Numerical Facts 34

2.4 The Greedy Algorithm 35
2.4.1 Details of the Greedy Algorithm 36
2.4.2 Features of the Greedy Algorithm 38
2.5 Reduction Rules in the DBNS 39
2.5.1 Basic Reduction Rules 39
2.5.2 Applying the Basic Reduction Rules 40
2.5.3 Generalized Reduction 42
2.6 A Two-Dimensional Index Calculus 44
2.6.1 Logarithmic Number Systems 44
2.6.2 A Filter Design Study 45
2.6.3 A Double-Base Index Calculus 47
2.7 Summary 49
References 50

3. Implementing DBNS Arithmetic 53
3.1 Introduction 53
3.1.1 A Low-Noise Motivation 54
3.2 Arithmetic Operations in the DBNS 55
3.2.1 DBNR Addition 55
3.2.2 DBNS Multiplication 58
3.2.3 Constant Multipliers 60
3.3 Conversion between Binary and DBNS Using Symbolic Substitution 60
3.4 Analog Implementation Using Cellular Neural Networks 62
3.4.1 Why CNNs? 62
3.4.2 The Systolic Array Approach 64
3.4.3 System Level Issues 64
3.4.4 CNN Basics 65
3.4.5 CNN Operation 66
3.4.6 Block and Circuit Level Descriptions of the CNN Cell 67
3.4.7 DBNS Addition CNN Templates Using Overlay and Row Reduction Rules 69
3.4.8 Designing the Templates 70
3.4.9 Handling Interference between Cells 72
3.4.10 CMOS Implementation of DBNS CNN Reduction Rules 74
3.4.11 A Complete CNN Adder Cell 77
3.4.12 Avoiding Feedback Races 79
3.5 Summary 81
References 82

4. Multiplier Design Based on DBNS 85
4.1 Introduction 85
4.1.1 A Brief Background 85
4.1.2 Extremely Large Numbers 86

4.2 Multiplication by a Constant Multiplier 86
4.3 Using the DBNS 87
4.4 DBNS Multiplication with Subquadratic Complexity 90
4.5 General Multiplier Structure 92
4.6 Results and Comparisons 100
4.7 Some Multiplier Designs 101
4.7.1 180 nm CMOS Technology 101
4.7.2 FPGA Implementation 103
4.8 Example Applications 104
4.9 Summary 105
References 105

5. The Multidimensional Logarithmic Number System (MDLNS) 109
5.1 Introduction 109
5.2 The Multidimensional Logarithmic Number System (MDLNS) 109
5.3 Arithmetic Implementation in the MDLNS 110
5.3.1 Multiplication and Division 111
5.3.2 Addition and Subtraction 111
5.3.4 Approximations to Unity 112
5.4 Multiple-Digit MDLNS 116
5.4.1 Error-Free Integer Representations 116
5.4.2 Non-Error-Free Integer Representations 120
5.5 Half-Domain MDLNS Filter 121
5.5.1 Inner Product Step Processor 121
5.5.2 Single-Digit 2DLNS Computational Unit 123
5.5.3 Extending to Multiple Bases 128
5.5.4 Extending to Multiple Digits 129
5.5.5 General Binary-to-MDLNS Conversion 130
5.6 Summary 131
References 132

6. Binary-to-Multidigit Multidimensional Logarithmic Number System Conversion 135
6.1 Introduction 135
6.2 Single-Digit 2DLNS Conversion 136
6.2.1 Single-Digit 2DLNS-to-Binary Conversion 136
6.2.2 Binary-to-Single-Digit 2DLNS Conversion 138
6.3 Range-Addressable Lookup Table (RALUT) 141
6.3.1 RALUT Architecture 141
6.3.1.1 Binary-to-Single-Digit 2DLNS Structure 144
6.4 Two-Digit 2DLNS-to-Binary Conversion 145
6.4.1 Two-Digit 2DLNS-to-Binary Conversion Architecture 145

6.5 Binary-to-Two-Digit 2DLNS Conversion 145
6.5.1 Binary-to-Two-Digit 2DLNS Conversion (Quick Method) 147
6.5.1.1 Quick Binary-to-Two-Digit 2DLNS Conversion Architecture 149
6.5.2 Binary-to-Two-Digit 2DLNS Conversion (High/Low Method) 149
6.5.2.1 Modifying the RALUT for the High/Low Approximation 151
6.5.2.2 High/Low Binary-to-Two-Digit 2DLNS Architecture 152
6.5.3 Binary-to-Two-Digit 2DLNS Conversion (Brute-Force Method) 152
6.5.3.1 Brute-Force Conversion Architecture 155
6.5.4 Binary-to-Two-Digit 2DLNS Conversion (Extended-Brute-Force Method) 156
6.5.5 Comparison of Binary-to-Two-Digit 2DLNS Conversion Methods 156
6.6 Multidigit 2DLNS Representation (n > 2) 157
6.6.1 Multidigit 2DLNS-to-Binary Conversion 157
6.6.2 Binary-to-Multidigit 2DLNS Conversion (Quick Method) 157
6.6.3 Binary-to-Multidigit 2DLNS Conversion (High/Low Method) 157
6.6.4 Binary-to-Multidigit 2DLNS Conversion (Brute-Force Method) 158
6.7 Extending to More Bases 158
6.8 Physical Implementation 159
6.9 Very Large-Bit Word Binary-to-DBNS Converter 160
6.9.1 Conversion Methods for Large Binary Numbers 161
6.9.2 Reducing the Address Decode Complexity 163
6.9.3 Results and Discussion 163
6.10 Summary 166
References 166

7. Multidimensional Logarithmic Number System: Addition and Subtraction 169
7.1 Introduction 169
7.2 MDLNS Representation 170
7.2.1 Simplified MDLNS Representation 170
7.3 Simple Single-Digit MDLNS Addition and Subtraction 171
7.4 Classical Method 172
7.4.1 LNS Implementation 173
7.4.2 MDLNS Implementation 173

7.5 Single-Base Domain 175
7.5.1 Single-Digit MDLNS to Single-Base Domain 176
7.5.2 Single-Base Domain to Single-Digit MDLNS 178
7.5.3 MDLNS Magnitude Comparison 180
7.6 Addition in the Single-Base Domain 180
7.6.1 Computing the Addition Table 180
7.6.1.1 Minimizing the Search Space 181
7.6.1.2 Table Redundancy 183
7.6.1.3 RALUT Implementation 183
7.6.1.4 Table Merging 183
7.6.1.5 Alternatives for m 186
7.6.2 The Complete Structure 187
7.6.3 Results 187
7.7 Subtraction in the Single-Base Domain 188
7.7.1 Computing the Subtraction Table 188
7.7.1.1 Minimizing the Search Space 189
7.7.1.2 RALUT Implementation and Table Reduction 190
7.7.2 The Complete Structure 192
7.7.3 Results 192
7.8 Single-Digit MDLNS Addition/Subtraction 193
7.8.1 The Complete Structure 193
7.8.2 Results 193
7.9 Two-Digit MDLNS Addition/Subtraction 194
7.10 MDLNS Addition/Subtraction with Quantization Error Recovery 195
7.10.1 Feedback Accumulation in SBD 196
7.10.2 Feedback Accumulation in SBD (with Full SBD Range) 197
7.10.3 Increasing the SBD Accuracy Internally 197
7.11 Comparison to an LNS Case 198
7.11.1 Addition 199
7.11.2 Subtraction 199
7.11.3 Comparisons 200
7.12 Summary 201
References 201

8. Optimizing MDLNS Implementations 203
8.1 Introduction 203
8.2 Background 203
8.2.1 Single-Digit 2DLNS Representation 203
8.2.2 Single-Digit 2DLNS Inner Product Computational Unit 204

8.3 Selecting an Optimal Base 205
8.3.1 The Impact of the Second Base on Hardware Cost 205
8.3.2 Defining a Finite Limit for the Second Base 206
8.3.3 Finding the Optimal Second Base 207
8.3.3.1 Algorithmic Search 207
8.3.3.2 Range Search 208
8.3.3.3 Fine Adjustment 208
8.4 One-Bit Sign Architecture 208
8.4.1 Representation Efficiency 209
8.4.2 Effects on Determining the Optimal Base 209
8.4.3 Effects on Hardware Architecture 209
8.5 Example Finite Impulse Response Filter 211
8.5.1 Optimizing the Base through Analysis of the Coefficients 214
8.5.1.1 Single-Digit Coefficients 214
8.5.1.2 Two-Digit Coefficients 215
8.5.1.3 Comparison of Single- and Two-Digit Coefficients 216
8.5.1.4 Effects on the Two-Digit Data 216
8.5.2 Optimizing the Base through Analysis of the Data 217
8.5.3 Optimizing the Base through Analysis of Both the Coefficients and Data 218
8.5.3.1 Single-Digit Coefficients and Two-Digit Data 219
8.5.3.2 Comparison to the Individual Optimal Base 219
8.5.3.3 Two-Digit Coefficients and Two-Digit Data 220
8.5.3.4 Comparison to the Individual Optimal Base 220
8.5.4 Comparison of Base 3 to the Optimal Bases 220
8.5.5 Comparison to General Number Systems 221
8.6 Extending the Optimal Base to Three Bases 223
8.7 Summary 224
References 225

9. Integrated Circuit Implementations and RALUT Circuit Optimizations 227
9.1 Introduction 227
9.2 A 15th-Order Single-Digit Hybrid DBNS Finite Impulse Response (FIR) Filter 227
9.2.1 Architecture 228
9.2.2 Specifications 228
9.2.3 Results 229
9.3 A 53rd-Order Two-Digit DBNS FIR Filter 229
9.3.1 Specifications for the Two-Digit 2DLNS Filter 230
9.3.2 Results 231
9.4 A 73rd-Order Low-Power Two-Digit MDLNS Eight-Channel Filterbank 231

9.4.1 Specifications 232
9.4.2 Results 232
9.4.3 Improvements 232
9.4.4 Results 233
9.5 Optimized 75th-Order Low-Power Two-Digit MDLNS Eight-Channel Filterbank 233
9.5.1 Results 234
9.6 A RISC-Based CPU with 2DLNS Signal Processing Extensions 235
9.6.1 Architecture and Specifications 235
9.6.2 Results 236
9.7 A Dynamic Address Decode Circuit for Implementing Range Addressable Look-Up Tables 237
9.7.1 Efficient RALUT Implementation 237
9.7.2 Dynamic Address Decoders (LUT) 238
9.7.3 Dynamic Range Address Decoders (RALUT) 240
9.7.4 Full Custom Implementation 242
9.7.5 Comparison with Past Designs 243
9.7.6 Additional Optimizations 243
9.8 Summary 244
References 244

10. Exponentiation Using Binary-Fermat Number Representations 247
10.1 Introduction 247
10.1.1 Some Examples of Exponentiation Techniques 247
10.1.2 About This Chapter 248
10.2 Theoretical Background 249
10.3 Finding Suitable Exponents 252
10.3.1 Bases 2 and 3 253
10.4 Algorithm for Exponentiation with a Low Number of Regular Multiplications 254
10.5 Complexity Analysis Using Exponential Diophantine Equations 257
10.6 Experiments with Random Numbers 260
10.6.1 Other Bases 261
10.7 A Comparison Analysis 262
10.8 Final Comments 262
10.9 Summary 263
References 263

Index 267

This page intentionally left blank

Preface

This is a book about a new number representation that has interesting properties for special applications. It is appropriately catalogued in the area of computer arithmetic, which, as the name suggests, is about arithmetic that is appropriate for implementing on calculating machines. These machines have changed over the millennia that humans have been building aids to performing arithmetic calculations. At the present time, arithmetic processors are buried in the architectural structures of computer processors, built mostly out of silicon, with a minimum lateral component spacing of the order of a few tens of nanometers, and vertical spacing down to just a few atoms.

Arithmetic is one of the fields that even young children learn about. Counting with the natural numbers (1, 2, 3 …) leads to learning to add and multiply. Negative numbers and the concept of zero lead to expanding the natural numbers to the integers (…, –3, –2, –1, 0, 1, 2, 3 …), and learning about division leads to fractions and the rational numbers. When we perform arithmetic "longhand" we use a positional number representation with a radix of 10—undoubtedly developed from the fact that humans have a total of 10 digits on their two hands. Early mechanical as well as some electronic digital computers maintained the radix of 10, but the two-state nature of digital logic gates and storage technology leads to a radix of 2 as being more natural for electronic machines. Binary number representations, which use a fixed radix of 2, are ubiquitous in the field of computer arithmetic, and there are many valuable textbooks that cover the special arithmetic hardware circuits and processing blocks that make use of binary representations.

The subject of computer arithmetic is fascinating since it deals with the basic computational underpinnings of processor functions in all digital electronic systems with applications that are ubiquitous in our modern technological world. In fact, computer arithmetic is so fundamental to digital design that there is a danger of it being dismissed by many designers, who see it as well-worn knowledge that is already encased in hardware (actual or in a design library) or that can be readily called up from a variety of software packages. What is often missing, however, is that computer arithmetic can be optimized based on targeted applications and technologies, and easily available standard arithmetic hardware may not be the most efficient implementation strategy. There is also a strong curiosity element associated with unusual approaches to implementing fundamental computer operations, and in this book we try to stir in a mixture of curiosity with our pragmatic approaches to implementing basic arithmetic for two application areas: digital signal processing and cryptography. As such, this book is somewhat different from most of the textbooks on computer arithmetic, in that it deals almost entirely with a rather new multiple-base number system

that is rapidly gaining in popularity. In its original form, the double-base number system (DBNS) can be viewed as an extension to the binary number system. By using a second base, in addition to the base 2 used by the binary number system, we uncover some rather remarkable properties of both the DBNS and its logarithmic extension, the multidimensional logarithmic number system (MDLNS).

We trust that the reader will find something fascinating about multiple bases and, if we have done our job well, will also be convinced that there are applications that can benefit from, at least, a serious consideration of this number representation and the techniques we have identified to efficiently implement calculations using multiple bases. The authors of the book have a collective interest in pursuing alternative representations that have the potential to improve aspects of the implementation of high-performance calculations in hardware. Two of us are engineers (Jullien and Muscedere), and the principal author, Dimitrov, has both a mathematics and an engineering background. All three of us are dedicated to implementing our discoveries in advanced integrated circuit technologies, and some of these designs are used to illustrate the theory and architectural techniques that are disclosed in this book.

In summary, the purpose of the book is to showcase the usefulness of the multiple-base number representation in various applications, and our main goal is to disseminate the results of our research work to as wide as possible an audience of engineers, computer scientists, and mathematicians. We hope this book at least partially achieves this goal.

Vassil S. Dimitrov
Graham Jullien
Roberto Muscedere

About the Authors

Vassil S. Dimitrov earned a PhD degree in applied mathematics from the Bulgarian Academy of Sciences in 1995. Since 1995, he has held postdoctoral positions at the University of Windsor, Ontario (1996–1998) and Helsinki University of Technology (1999–2000). From 1998 to 1999, he worked as a research scientist for Cigital, Dulles, Virginia (formerly known as Reliable Software Technology), where he conducted research on different cryptanalysis problems. Since 2001, he has been an associate professor in the Department of Electrical and Computer Engineering, Schulich School of Engineering, University of Calgary, Alberta. His main research areas include implementation of cryptographic protocols, number theoretic algorithms, computational complexity, image processing and compression, and related topics.

Graham Jullien recently retired as the iCORE chair in Advanced Technology Information Processing Systems, and the director of the ATIPS Laboratories, in the Department of Electrical and Computer Engineering, Schulich School of Engineering, at the University of Calgary. His long-term research interests are in the areas of integrated circuits (including SoC), VLSI signal processing, computer arithmetic, high-performance parallel architectures, and number theoretic techniques. Since taking up his chair position at Calgary, he expanded his research interests to include security systems, nanoelectronic technologies, and biomedical systems. He was also instrumental, along with his colleagues, in developing an integration laboratory cluster to explore next-generation integrated microsystems. Dr. Jullien is a fellow of the Royal Society of Canada, a life fellow of the IEEE, a fellow of the Engineering Institute of Canada, and until recently, was a member of the boards of directors of DALSA Corp., CMC Microsystems, and Micronet R&D. He has published more than 400 papers in refereed technical journals and conference proceedings, and has served on the organizing and program committees of many international conferences and workshops over the past 35 years. Most recently he was the general chair for the IEEE International Symposium on Computer Arithmetic in Montpellier in 2007, and was guest coeditor of the IEEE Proceedings special issue *System-on-Chip: Integration and Packaging*, June 2006.

Roberto Muscedere received his BASc degree in 1996, MASc degree in 1999, and PhD in 2003, all from the University of Windsor in electrical engineering. During this time he also managed the microelectronics computing

environment at the Research Centre for Integrated Microsystems (formally the VLSI Research Group) at the University of Windsor. He is currently an associate professor in the Electrical and Computer Engineering Department at the University of Windsor. His research areas include the implementation of high-performance and low-power VLSI circuits, full and semicustom VLSI design, computer arithmetic, HDL synthesis, digital signal processing, and embedded systems. Dr. Muscedere has been a member of the IEEE since 1994.

1

Technology, Applications, and Computation

1.1 Introduction

The field of computer arithmetic is a fascinating subject, and not at all the drudgery that most of us experienced with our first exposure to the third *r* of the three *r*'s at school. The first two *r*'s are related as requirements of the processes required to become literate. The third *r* relates to numeracy, the understanding and implementation of computation which is a basic requirement for "technical literacy," as important a skill in today's world as was traditional literacy in the past. In this introductory chapter, we provide a brief history and introduction to number systems and machine calculation; we emphasize special applications that drive the search for efficient machine arithmetic given the requirements of the applications and the available technology.

1.2 Ancient Roots

Ancient numbering systems abound, but the most striking is the system developed by the Babylonians starting around 5,000 years ago. It is striking in that it is a form of weighted positional system, which we use today. However, the Babylonians did not have a symbol for zero (instead they used a space) and the weighting was based on the number 60 rather than the number 10, which we use today for representing decimal numbers. (Vestiges of the weight 60 can still be found in the way we measure time and angles.) It also appears that the binary number system, which we naturally think of as being introduced with the advent of electronic computers built with logic gates, was used at least 4,000 years ago for determining weights using a simple balance and carefully weighed stones [1].

1.2.1 An Ancient Binary A/D Converter

We can imagine a trader from 4,000 years ago by the side of a river, setting up a balance and then searching for a handful of river-washed pebbles that

had specific weight characteristics. These characteristics were determined from the simple operation of producing a balance between sets of stones. The technology used here was based only on the force of gravity using a balance bar and fulcrum. The operation is demonstrated in Figure 1.1 for an equivalent 4-bit representation (1–15 times the smallest stone weight). The accuracy of the number system is determined by the accuracy with which the stones were selected and correctly balanced (including positioning the stones so that their accumulated center of gravity was always in the same position on the opposite sides of the balance). The relative weights of the stones in the full measurement set are shown in Figure 1.1 as {1, 1, 2, 4, 8}. Designers of binary-weighted digital/analog converters (D/As) know this sequence well! Such a D/A converter can be used to implement a successive approximations A/D converter. The two 1's are redundant in terms of a 4-bit measurement system, only being required to generate the full measurement set. In a sense, the traders of 4,000 years ago had also built an A/D converter in which an analog commodity weight was converted into a subset of stones from the full measurement set.

1.2.2 Ancient Computational Aids

Computational aids and calculators also have ancient roots. Counting boards (a precursor of the "more modern" abacus using beads) have been used for several thousand years [2], with some evidence that they were

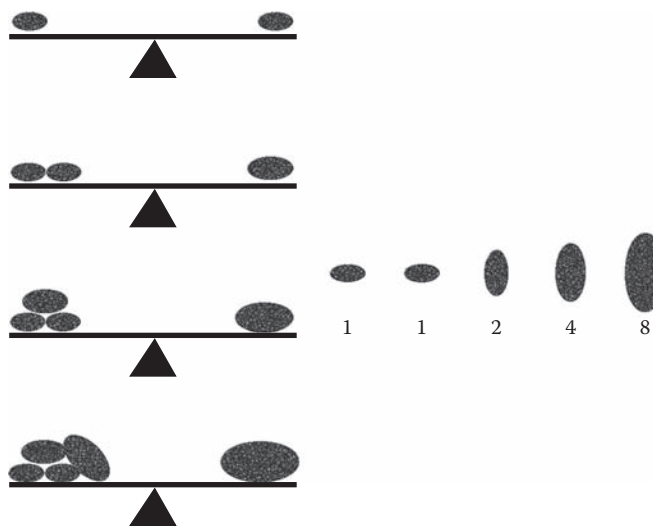

FIGURE 1.1 (See color insert)
Four-thousand-year-old binary number system.

initially developed by the Romans. Of some surprise, an astronomical prediction machine, circa 200 BC, was recovered from an ancient shipwreck off the Greek island of Antikythera in 1901. This mechanism was truly advanced based on the fact that mechanical calculators of similar complexity did not appear again (or, at least, have not been found) for at least another 1,500 years. Based on an analysis of CT scans of the components [3], the calculator used a variety of sophisticated precision gears, each with up to several hundred teeth, along with epicyclic gearing. The mechanism was able to compute addition, subtraction, and multiplication with precise fractions and, for example, could predict the position of the moon based on models available at the time.

1.3 Analog or Digital?

Over the past five centuries the interest in computational aids and calculating machines has steadily increased, with an explosive growth over the past 100 years, driven by mechanical devices, electromechanical devices, and electronics.

Mechanical and electromechanical devices were based on decimal arithmetic because of the need to interact with human operators. Some of the early electronic computers also used the decimal-based number systems for the same reason, even though the two-state property (0, 1) of signal propagation into and out of computer logic circuits provides a perfect match with pure binary number representations. Until the latter part of the 20th century, analog devices were also heavily used for advanced mathematical computations, such as finding solutions of nonlinear differential equations, where a physical quantity (e.g., voltage in electronic circuits or rotation in mechanical systems) is observed as the solution to a problem. In this case there is no implied number system, only that of the operator in interpreting the analog results. Analog systems have a relatively large computational error (several percent) compared to the much higher precision capable of digital machines, but there were application areas found for both analog and digital machines. We will discuss an analog method of computing with a double-base system in Chapter 3.

Examples of applications that took advantage of the disparity in complexity and error between analog and digital mechanisms are ideally portrayed in comparing Lord Kelvin's analog tide prediction machine [4] to Babbage's digital difference engine [5]; they were proposed within a few decades of each other in the 19th century.

1.3.1 An Analog Computer Fourier Analysis Tide Predictor

The tide predictor computed the height of the tide from a datum point as a function of time using the form of Equation (1.1):

$$h(t) = H_0 + \sum_{i=1}^{10} \{A_i \cos[g_i t + f_i]\} \tag{1.1}$$

The predictor combines tidal harmonic constituents, based on harmonic components of the changes of the position of the earth, sun, and moon, to perform the prediction. The technique is still used today to make tide predictions, though the 10 components in Lord Kelvin's first machine are augmented by a factor of 4 or more, based on location [4]. This analog computer used a system of pulleys, wires, and dials (see Figure 1.2), and the generation of sinusoid motion used a rotating wheel with an

FIGURE 1.2 (See color insert)
Lord Kelvin's tide predicting machine (1876), Science Museum, London. (Copyright 2005 William M. Connolley under the GNU Free Documentation License.)

off-center peg to drive a vertically moving arm (the inverse of a piston driving a crankshaft), and similar mechanisms can also be seen in the Antikythera machine of 2,000 years earlier! Lord Kelvin's predictor could print out calculations of harbor tide patterns for up to a year in about 4 hours by turning a crank. Clearly errors of a few percent were acceptable for this application, particularly considering the errors in the astronomical models used.

1.3.2 Babbage's Difference Engine

Charles Babbage's difference engine [5] was designed to tabulate the values of polynomials of nth degree using the method of finite differences. The engine design used mechanical wheels and gears to compute using the decimal number system; the main purpose of the engine was to produce accurate mathematical tables with no human error (including direct printing from the engine). His difference engine no. 2 was able to hold (store) eight numbers of 31 decimal digits each, allowing the machine to tabulate seventh degree polynomials to that precision. By using Newton's method of divided differences, the engine can compute a sequence of values of a polynomial with only the need to add and store, using a *ten's complement* form for negative numbers. There were some intriguing aspects to the design of the engine, including the addition and carry propagation, using carry save and carry skip techniques, techniques that are still used in modern microcomputers. Babbage never completed the building of the difference engine; we had to wait until the 1990s to be able to see replicas at the Science Museum in London and the Computer History Museum in Mountain View, California (see Figure 1.3).

1.3.3 Pros and Cons

Comparing the tide predictor to the difference engine, we note that the difference engine would not be able to automatically compute the complete tide prediction model of Equation (1.1); however, the harmonic components could be approximated by finite order polynomials so parts of the equation could be individually evaluated, with the final summation to be carried out at the end. Perhaps the analytical engine [5,6] that Babbage proposed after abandoning the construction of the difference engine would have been more suitable based on its programmability with punched cards. However, the complete analytical engine has never been built, and even if it were it would be so cumbersome as to require a real engine to turn the crank! Analog computing machines were clearly much easier to build than digital machines using the technology of the 19th century, and this was still the case up to the end of the vacuum tube era (middle of the 20th century).

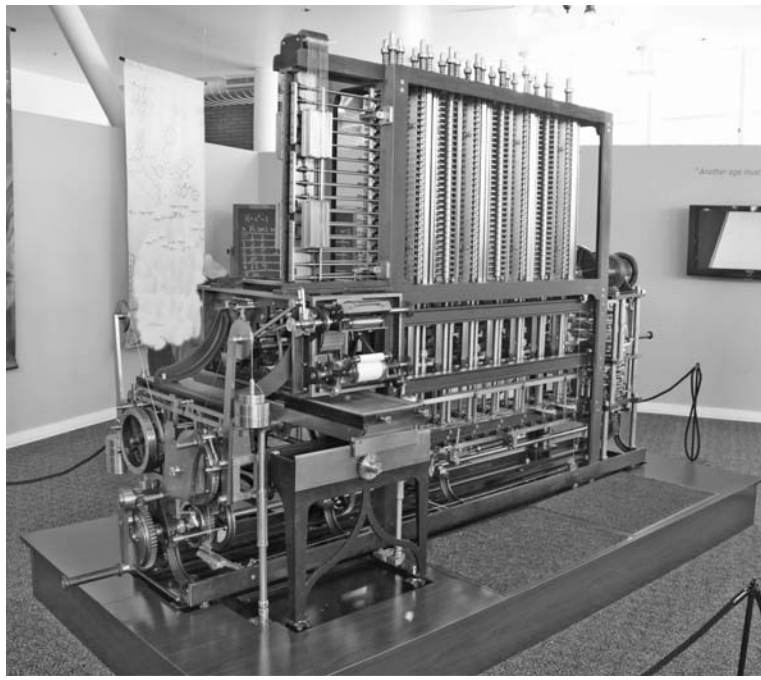

FIGURE 1.3 (See color insert)
Babbage's difference engine, Computer History Museum, Mountain View, California. (Copyright 2009 Allan J. Cronin under the GNU Free Documentation License.)

1.3.4 The Slide Rule: An Analog Logarithmic Processor

Analog and digital computational aids were available to students and engineers long before the pocket calculator arrived. One of the authors went through his entire college and university education using tables of logarithms and trigonometric functions (digital aids) and a slide rule (analog aid). The $6.50 purchase of the 1,046-page standards book edited by Abramowitz and Stegun [7] was a particularly good deal for the professional engineer in 1969! The tables of logarithms (and their inverses) were mainly used for accurate computations of multiplications (including reciprocals and powers), with the slide rule being used where lower accuracy could be tolerated—this meant most of the time! The slide rule uses logarithmic scales on both the static body and the slide, so clearly, in both the digital and analog approaches, we are using the mapping property of multiplication to addition in going from a linear scale to a log scale. For the tables we looked up the logarithms of the multiplier and multiplicand, added the logarithms, and looked up the inverse logarithm in the tables. It was assumed, of course, that we were skilled at adding numbers. For the slide rule calculations the logarithmic map was automatically performed with the logarithmic scale. The addition was performed by concatenating the slide and body markings and looking

up the sum, as shown in Figure 1.4 (using the author's slide rule—warts and all). The figure shows the cursor "looking up" the concatenation of the body scale at 1.5 with the slide scale at 2.5, giving a result of between 3.74 and 3.76 (the accurate answer is, of course, 3.75).

The master of the slide rule knew the trick to compute $\sqrt{a^2+b^2}$ in three moves without having to do a difficult addition. The trick was to rewrite the expression as shown in Equation (1.2):

$$\sqrt{a^2+b^2} = a \cdot \sqrt{1+\left(\frac{b}{a}\right)^2} \tag{1.2}$$

Computing the right-hand side of Equation (1.2) involves a division, a squaring, adding 1, a square root, and a multiplication. Assuming the slide rule has a scale of squares (all good slide rules did), then the only tricky move was the addition of 1, which is trivial.

Figure 1.5 shows the three moves used to compute $\sqrt{3^2+4^2}=5$. We elect to choose $a > b$ so that $b^2/a^2 < 1$ and the addition will yield a number with 1 to the left of the decimal point. From Figure 1.5(a) we see the first move, where we divide 3 by 4 (0.75) by subtracting the scales and then looking up the square (~0.562) on the right-hand side of the scale (at 100 on the x^2 scale). We now move the slide to 1.562 (or as close as possible) on the left-hand side of the slide and move the cursor to 4 on the x scale of the slide. Looking at the x scale on the body of the slide rule, we get the answer, 5. The trick here was to rewrite the expression so that the addition is trivial. This same idea is used in logarithmic arithmetic to turn addition with two addends into a unary lookup table. In our group we affectionately refer to this as the *slide rule trick*, and will so refer to it in subsequent chapters dealing with a multiple-base logarithmic arithmetic. As an aside, we note that we have to keep track of the magnitude of the numbers in the calculation. Clearly the 56.2 value in Figure 1.5(a) has to be interpreted as 0.562 so that the slide is located at 1.562 in Figure 1.5(b). Keeping track of magnitudes of calculations

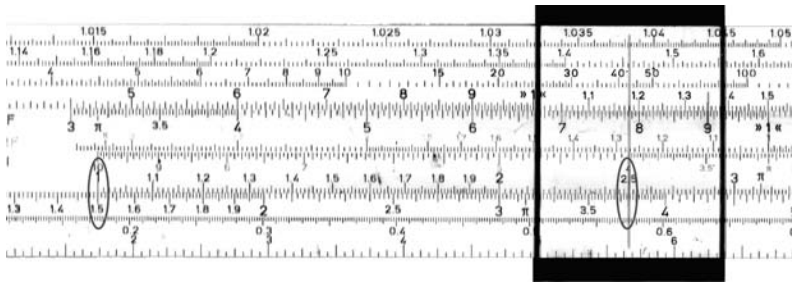

FIGURE 1.4 (See color insert)
Multiplication of 1.5 by 2.5 on a slide rule.

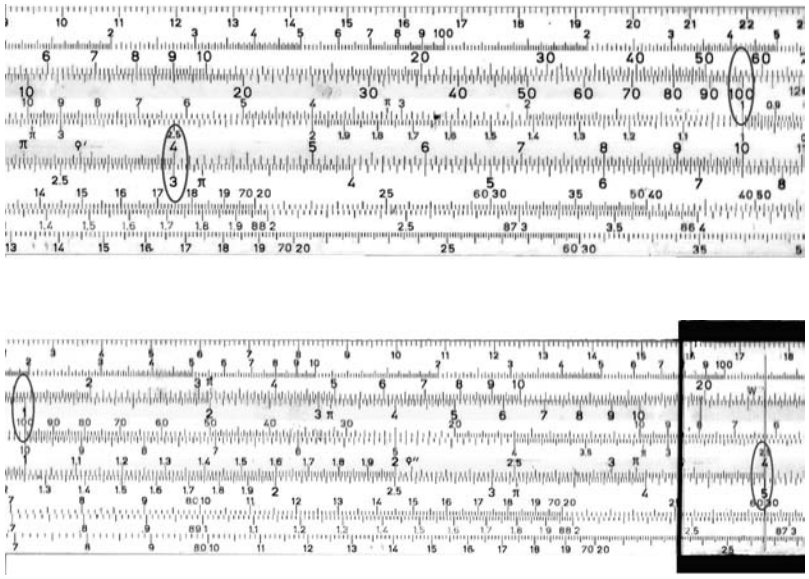

FIGURE 1.5 (See color insert)
Three moves to compute the slide rule trick.

is a skill that has seen some demise since the availability of the ubiquitous pocket calculator!

It is interesting to note that logarithmic arithmetic was suggested as a computational implementation tool for digital filters as far back as 1971 by Kingsbury and Rayner [8], with considerable interest since then in applying logarithmic arithmetic to this application area [9–11].

1.4 Where Are We Now?

Analog computers, used to solve mathematical equations, essentially disappeared during the early 1970s, although efforts were made to marry them to digital machines—a so-called hybrid computer approach. However, a different type of computation with analog devices started in the 1940s with the investigation of nervous (or neural) activity by McCulloch and Pitts [12], and further developed in the 1960s with the modeling of organic neural clusters (brains!) using nonlinear analog circuits. Thus it might be helpful to separate out analog computation into linear analog computers, which were used to evaluate differential equations with as much accuracy as the linearity of the amplifiers with passive component feedback allowed, and nonlinear analog processors, which include neural networks, cellular nonlinear networks, and their analog circuit implementations.

1.4.1 Moore's Law

The 1970s also saw the birth of the microprocessor and the start of the current revolution in high-performance computational systems. The advent of integrated transistor circuits (ICs) has allowed for an exponential increase in the complexity of single-chip computational circuits and the attendant advances in information processing capability. Moore's law [13] has stood the test of time (4½ decades and counting) in spite of the need for regularly finding solutions to major roadblocks in lithography and other aspects of IC fabrication technology. The predictions of an imminent departure from Moore's law abound throughout the decades of advances in IC technology. Though as we approach atomic dimensions in the active devices built on silicon, Moore's law certainly seems threatened, unless we can find new nanotechnologies to provide a continuation of the law. Advances in IC fabrication density and device speed have had a profound effect on the basic computational units that are present in almost every IC that is fabricated today. In fact, there has been a revolution in the number and type of new applications that have appeared simply because of the availability of billions of active devices on a single sliver of monolithic crystalline silicon.

The traditional applications for computing, such as the physical sciences, finance, and military applications, drove the first forays into electronic computers and have benefited enormously from the improvements, through vacuum tubes, transistors, and integrated circuits. For these applications, the requirements for computational accuracy have driven the design of the arithmetic units, in particular the use of floating point arithmetic as a way of representing numbers with precision and dynamic range, in spite of a relatively large overhead in the hardware required to perform the arithmetic. The demand for faster and more accurate processing power also led to the birth of supercomputers in the 1960s. These are computers that are able to process orders of magnitude more information per second than more accessible machines. A definition of number-crunching processing power, *floating point operations per second* (FLOPS), was also introduced, and we have seen this measure increase from mega-FLOPS to peta-FLOPS over just a few decades. Interestingly, even cheap consumer computers exceed the definition of a supercomputer of just a few years earlier, often faster than the U.S. military changes the definition! The ubiquitous FLOP has also been used as a more general measure of computer processing power, even if the main use is not continuous computation with floating point arithmetic. In fact, all of a computer's systems, including its architecture, in addition to arithmetic units, need to have performance improvements in order to provide sustained data to the arithmetic unit(s).

1.4.2 The Microprocessor and Microcontroller

The first microprocessors appeared in 1970–1971: the most well known is the 4004 from Intel Corp. [14]. The CPU chip was actually part of a four-chip set

used to build a complete computer system. By the mid-1970s, complete computer systems on a single chip were available in which all of the components required to build a complete computer system (albeit very limited) were contained on the chip. One of the first chips was the Intel 8748, which contained an erasable programmable read-only memory (EPROM) for programming, and most of the pins were available as input/output (I/O) lines. Along with programmable I/O pins, the 8748 contained a basic timer and interrupt structure that together would normally be considered nonessential for a standard computer system, let alone for a resource-limited single chip of that time. In fact, this and subsequent complete single-chip computers were targeted to a new market where the computer was embedded into a product that was not, itself, a computer. In reality, the chip was a very flexible microcontroller. As an example, one of the major applications for the 8748 was as a keyboard controller for IBM computers. The terms *microcontroller* and *embedded systems* are very familiar to us now, but their birth was with the development of devices such as the 8748. In terms of being a computational unit, they were very limited by today's standards, as were all of the microprocessor products of that time, but that did not stop users of these devices from exploring ways that fast computations could be carried out using alternative techniques [15].

The first desktop computers, based on microprocessors, started to appear in the late 1970s. They coexisted with mini-computers for several years until they became sufficiently powerful so that they could take on the same complexity of computational tasks that mini-computers had handled just a few years previously. In fact, whatever had been on a 19-inch rack-mount-printed circuit board in the mini-computer era was now available on one or two very large-scale integration (VSLI) chips. These included coprocessors that were expressly designed to offload floating point and integer arithmetic calculations from the main processor.

1.4.3 Advances in Integrated Circuit Technology

Advances in the implementation of computation have been driven by advances in integrated circuit technology and the evolution of the approaches to building computational circuits. The technology advances were predicted by Gordon Moore [13] with remarkable accuracy, though the fact that the industry uses Moore's law as a guide to its own technology improvements [16] perhaps introduces something of a self-fulfilling prophecy into the picture. The semiconductor industry is very inventive, and there have been several major breakthroughs that have maintained the exponential advances, often against the predictions of top people in the field. The first microprocessors were built with p-channel metal oxide semiconductor (PMOS) and then n-channel MOS (NMOS) technologies, but the star technology is still complementary metal oxide semiconductor (CMOS). CMOS became the dominant MOS digital technology in the 1980s, and has maintained this position in spite of other technologies, such as Bipolar-CMOS (BiCMOS) and GaAs, that, for a time, offered

some advantages, e.g., speed, over CMOS. In reality, CMOS is a set of logic families that make use of the ability to produce complementary devices in the same process, and we have seen a wide variety of static and dynamic CMOS logic families that have been used to implement computation. The first commercial CMOS processes used lithography (the optical technique of patterning photosensitive material on semiconductor substrates) with dimensions measured in microns. At the time of writing this book, 22 nm processes are starting to come on line, and 45 nm ICs are in the marketplace. An experimental (in 2009) Intel microprocessor family has 48 cores (individual processors) on a 1.3 billion transistor single chip (called the cloud computer). The computational power of such chips is measured in tera-FLOPS.

1.5 Arithmetic and DSP

Concurrent with the commercial development of integrated circuits in the 1960s, a new field of research was beginning to emerge: digital signal processing (DSP). In DSP, signals are sampled both in time (discrete time) and in magnitude (digital). Although the basic theoretical techniques were quite well known before the commercial development took place, the use of DSP only took a firm foothold with the advent of semiconductor electronics and, in particular, IC fabrication. We will spend a few pages discussing this subject because the search for efficient computational techniques for DSP algorithms led to an increased interest in alternative arithmetic techniques, including the double-base number system. In this section, for completeness, we review the mathematical bases for replacing classical analog processing with DSP.

1.5.1 Data Stream Processing

One of the features of many DSP algorithms is the requirement for data stream processing, that is, the continuous application of a computational algorithm on an input digital data stream to produce a continuous output data stream. This is exactly what analog signal processing does, but analog computers operate on continuous waveforms, and instead of digital logic, analog processors use amplifiers, resistors, capacitors, and inductors. Replacing continuous (analog) signal processing with DSP has some limitations, but these are overcome in practice. This represents a fundamental difference between the use of computers to solve a problem, where a program is written to input numerical data and to output the solution to the problem as another set of data. Once the solution has been obtained, the computer has done its job! In data streaming, however, the processor is continuously fed with a stream of numbers (e.g., a regularly sampled

analog signal), and outputs a stream of numbers at the same rate as the input. The stream of input numbers may be essentially infinite (e.g., samples of the output of a microphone that is never turned off), so the processor is never finished. The throughput rate (the inverse of the time between adjacent samples of the waveform) is determined by the application, and early DSP hardware was woefully inadequate to keep up with even the lowest-bandwidth applications. During the final three decades of the 20th century, many different processing techniques, including alternative arithmetic techniques and special architectures, were developed to squeeze higher and higher throughput rates out of the available hardware. As IC technology advances, mainstream computer arithmetic solutions can operate at sufficiently high-throughput rates to satisfy many of the ubiquitous applications, such as multimedia streaming, with acceptable results. However, as the definition of *acceptable* becomes more and more severe, as new applications take hold, and as we approach a potential brick wall for current technologies, it will be very useful to maintain interest in alternative techniques in order to increase the degrees of freedom to the designers of future systems.

1.5.2 Sampling and Conversion

The sampling operation allows continuous waveforms to be captured at regular intervals as streams of digital data. Figure 1.6 demonstrates the effect of sampling continuous signals by looking at the time and frequency domain of a sampled band-limited time domain waveform using the Fourier transform (FT) [17]. The figure shows time domain waveforms on the left and frequency domain waveforms on the right.

In Figure 1.6 we make use of the *multiplication* ⇔ *convolution* FT mapping property. The FT is shown in Equation (1.3), where $f(t)$ is an integrable function, t is the variable in the time domain, and ω is the variable in the radian frequency domain; i is the complex operator.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{1.3}$$

Equation (1.4) shows the inverse FT.

$$f(t) = \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \tag{1.4}$$

The FT is used to analyze the frequency domain components of a time domain waveform. The multiplication ⇔ convolution mapping property operates in both the forward and inverse transform directions; Figure 1.6 uses the forward mapping direction. We also use the property that a periodic

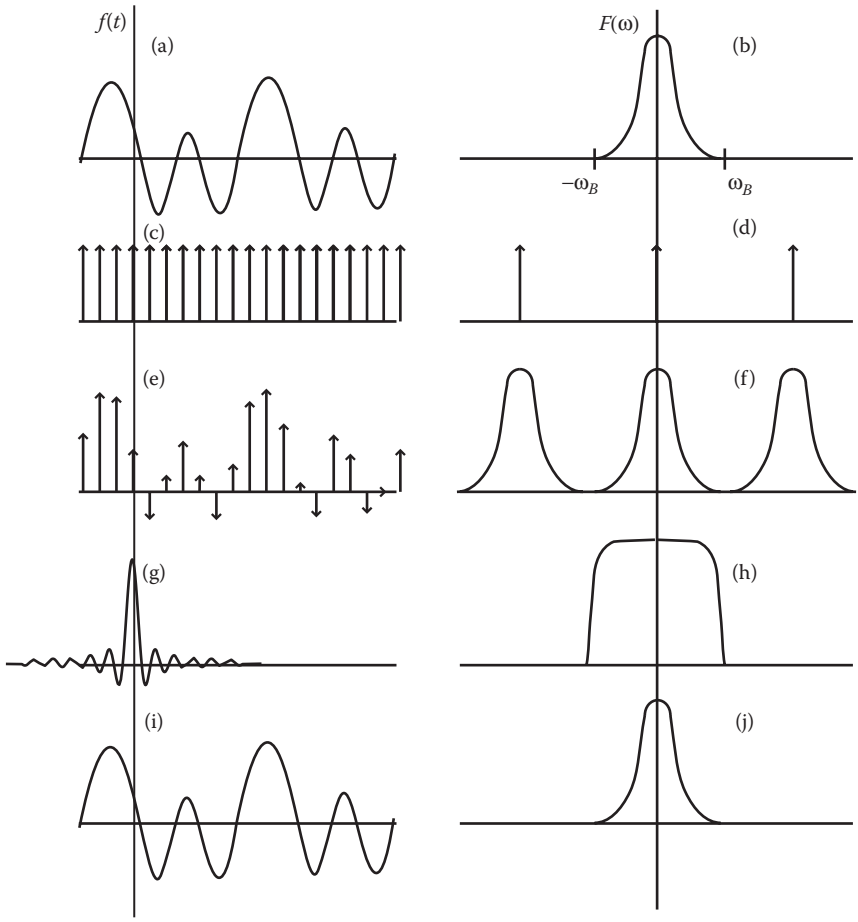

FIGURE 1.6
Processing continuous signals using digital signal processing (DSP).

train of Dirac delta functions (impulse train) in the time domain maps to another impulse train in the frequency domain [17]. A Dirac delta function, $\delta(x)$, is defined as in Equation (1.5):

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \tag{1.5}$$

where $\int_{-\infty}^{\infty} \delta(x)\,dx = 1$.