

SYNTHESIS OF ARITHMETIC CIRCUITS

FPGA, ASIC, and Embedded Systems

JEAN-PIERRE DESCHAMPS

University Rovira i Virgili

GÉRY JEAN ANTOINE BIOUL

National University of the Center of the Province of Buenos Aires

GUSTAVO D. SUTTER

University Autonoma of Madrid



A JOHN WILEY & SONS, INC., PUBLICATION

SYNTHESIS OF ARITHMETIC CIRCUITS

SYNTHESIS OF ARITHMETIC CIRCUITS

FPGA, ASIC, and Embedded Systems

JEAN-PIERRE DESCHAMPS

University Rovira i Virgili

GÉRY JEAN ANTOINE BIOUL

National University of the Center of the Province of Buenos Aires

GUSTAVO D. SUTTER

University Autonoma of Madrid



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2006 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey. Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400, fax 978-646-8600, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008 or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

Library of Congress Cataloging-in-Publication Data:

Deschamps, Jean-Pierre, 1945-

Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems/Jean-Pierre Deschamps, Gery Jean Antoine Bioul, Gustavo D. Sutter.
p. cm.

ISBN-13 978-0471-68783-2 (cloth)

ISBN-10 0-471-68783-9 (cloth)

1. Computer arithmetic and logic units. 2. Digital electronics. 3. Embedded computer systems.
I. Bioul, Gery Jean Antoine. II. Sutter, Gustavo D. III. Title.

TK7895.A65D47 2006

621.39'5 - - dc22

2005003237

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To Marc

CONTENTS

Preface	xvii
About the Authors	xix
1 Introduction	1
1.1 Number Representation, 1	
1.2 Algorithms, 2	
1.3 Hardware Platforms, 2	
1.4 Hardware–Software Partitioning, 3	
1.5 Software Generation, 3	
1.6 Synthesis, 3	
1.7 A First Example, 3	
1.7.1 Specification, 3	
1.7.2 Number Representation, 6	
1.7.3 Algorithms, 6	
1.7.4 Hardware Platform, 8	
1.7.5 Hardware–Software Partitioning, 8	
1.7.6 Program Generation, 9	
1.7.7 Synthesis, 10	
1.7.8 Prototype, 12	
1.8 Bibliography, 14	

2	Mathematical Background	15
2.1	Number Theory, 15	
2.1.1	Basic Definitions, 15	
2.1.2	Euclidean Algorithms, 17	
2.1.3	Congruences, 19	
2.2	Algebra, 25	
2.2.1	Groups, 25	
2.2.2	Rings, 27	
2.2.3	Fields, 27	
2.2.4	Polynomial Rings, 27	
2.2.5	Congruences of Polynomial, 32	
2.3	Function Approximation, 35	
2.4	Bibliography, 36	
3	Number Representation	39
3.1	Natural Numbers, 39	
3.1.1	Weighted Systems, 39	
3.1.2	Residue Number System, 42	
3.2	Integers, 42	
3.2.1	Sign-Magnitude Representation, 42	
3.2.2	Excess- E Representation, 43	
3.2.3	B 's Complement Representation, 44	
3.2.4	Booth's Encoding, 47	
3.3	Real Numbers, 51	
3.4	Bibliography, 54	
4	Arithmetic Operations: Addition and Subtraction	55
4.1	Addition of Natural Numbers, 55	
4.1.1	Basic Algorithm, 55	
4.1.2	Faster Algorithms, 57	
4.1.3	Long-Operand Addition, 66	
4.1.4	Multioperand Addition, 67	
4.1.5	Long-Multioperand Addition, 70	
4.2	Subtraction of Natural Numbers, 71	
4.3	Integers, 71	
4.3.1	B 's Complement Addition, 71	
4.3.2	B 's Complement Sign Change, 72	
4.3.3	B 's Complement Subtraction, 74	

4.3.4	B 's Complement Overflow Detection, 74	
4.3.5	Excess- E Addition and Subtraction, 78	
4.3.6	Sign-Magnitude Addition and Subtraction, 79	
4.4	Bibliography, 80	
5	Arithmetic Operations: Multiplication	81
5.1	Natural Numbers Multiplication, 82	
5.1.1	Introduction, 82	
5.1.2	Shift and Add Algorithms, 83	
5.1.2.1	Shift and Add 1, 83	
5.1.2.2	Shift and Add 2, 84	
5.1.2.3	Extended Shift and Add Algorithm: $XY + C + D$, 86	
5.1.2.4	Cellular Shift and Add, 86	
5.1.3	Long-Operand Algorithm, 90	
5.2	Integers, 91	
5.2.1	B 's Complement Multiplication, 91	
5.2.1.1	Mod B^{n+m} B 's Complement Multiplication, 92	
5.2.1.2	Signed Shift and Add, 93	
5.2.1.3	Postcorrection B 's Complement Multiplication, 93	
5.2.2	Postcorrection 2's Complement Multiplication, 96	
5.2.3	Booth Multiplication for Binary Numbers, 97	
5.2.3.1	Booth- r Algorithms, 97	
5.2.3.2	<i>Per Gelosia</i> Signed-Digit Algorithm, 98	
5.2.4	Booth Multiplication for Base- B Numbers (Booth- r Algorithm in Base B), 102	
5.3	Squaring, 104	
5.3.1	Base- B Squaring, 104	
5.3.1.1	Cellular Carry-Save Squaring Algorithm, 104	
5.3.2	Base-2 Squaring, 106	
5.4	Bibliography, 107	
6	Arithmetic Operations: Division	109
6.1	Natural Numbers, 110	
6.2	Integers, 117	
6.2.1	General Algorithm, 117	
6.2.2	Restoring Division Algorithm, 121	
6.2.3	Base-2 Nonrestoring Division Algorithm, 121	
6.2.4	SRT Radix-2 Division, 126	
6.2.5	SRT Radix-2 Division with Stored-Carry Encoding, 131	
6.2.6	P - D Diagram, 139	

6.2.7	SRT-4 Division, 142	
6.2.8	Base- B Nonrestoring Division Algorithm, 148	
6.3	Convergence (Functional Iteration) Algorithms, 155	
6.3.1	Introduction, 155	
6.3.2	Newton–Raphson Iteration Technique, 155	
6.3.3	MacLaurin Expansion—Goldschmidt’s Algorithm, 159	
6.4	Bibliography, 161	
7	Other Arithmetic Operations	165
7.1	Base Conversion, 165	
7.2	Residue Number System Conversion, 173	
7.2.1	Introduction, 173	
7.2.2	Base- B to RNS Conversion, 173	
7.2.3	RNS to Base- B Conversion, 177	
7.3	Logarithmic, Exponential, and Trigonometric Functions, 180	
7.3.1	Taylor–MacLaurin Series, 181	
7.3.2	Polynomial Approximation, 183	
7.3.3	Logarithm and Exponential Functions Approximation by Convergence Methods, 184	
7.3.3.1	Logarithm Function Approximation by Multiplicative Normalization, 184	
7.3.3.2	Exponential Function Approximation by Additive Normalization, 188	
7.3.4	Trigonometric Functions—CORDIC Algorithms, 194	
7.4	Square Rooting, 198	
7.4.1	Digit Recurrence Algorithm—Base- B Integers, 198	
7.4.2	Restoring Binary Shift-and-Subtract Square Rooting Algorithm, 202	
7.4.3	Nonrestoring Binary Add-and-Subtract Square Rooting Algorithm, 204	
7.4.4	Convergence Method—Newton–Raphson, 208	
7.5	Bibliography, 208	
8	Finite Field Operations	211
8.1	Operations in Z_m , 211	
8.1.1	Addition, 212	
8.1.2	Subtraction, 213	
8.1.3	Multiplication, 213	
8.1.3.1	Multiply and Reduce, 214	
8.1.3.2	Modified Shift-and-Add Algorithm, 214	

- 8.1.3.3 Montgomery Multiplication, 216
- 8.1.3.4 Specific Ring, 220
- 8.1.4 Exponentiation, 221
- 8.2 Operations in $GF(p)$, 222
- 8.3 Operations in $Z_p[x]/f(x)$, 224
 - 8.3.1 Addition and Subtraction, 224
 - 8.3.2 Multiplication, 225
- 8.4 Operations in $GF(p^n)$, 228
- 8.5 Bibliography, 236
- Appendix 8.1 Computation of f_{ki} , 236

9 Hardware Platforms

239

- 9.1 Design Methods for Electronic Systems, 239
 - 9.1.1 Basic Blocks of Integrated Systems, 240
 - 9.1.2 Recurring Topics in Electronic Design, 241
 - 9.1.2.1 Design Challenge: Optimizing Design Metrics, 241
 - 9.1.2.2 Cost in Integrated Circuits, 242
 - 9.1.2.3 Moore's Law, 243
 - 9.1.2.4 Time-to-Market, 243
 - 9.1.2.5 Performance Metric, 244
 - 9.1.2.6 The Power Dimension, 245
- 9.2 Instruction Set Processors, 245
 - 9.2.1 Microprocessors, 247
 - 9.2.2 Microcontrollers, 248
 - 9.2.3 Embedded Processors Everywhere, 248
 - 9.2.4 Digital Signal Processors, 249
 - 9.2.5 Application-Specific Instruction Set Processors, 250
 - 9.2.6 Programming Instruction Set Processors, 251
- 9.3 ASIC Designs, 252
 - 9.3.1 Full-Custom ASIC, 252
 - 9.3.2 Semicustom ASIC, 253
 - 9.3.2.1 Gate-Array ASIC, 253
 - 9.3.2.2 Standard-Cell-Based ASIC, 254
 - 9.3.3 Design Flow in ASIC, 255
- 9.4 Programmable Logic, 256
 - 9.4.1 Programmable Logic Devices (PLDs), 256
 - 9.4.2 Field Programmable Gate Array (FPGA), 258
 - 9.4.2.1 Why FPGA? A Short Historical Survey, 258
 - 9.4.2.2 Basic FPGA Concepts, 258

9.4.3	Xilinx™ Specifics, 260	
9.4.3.1	Configurable Logic Blocks (CLBs), 262	
9.4.3.2	Input/Output Blocks (IOBs), 262	
9.4.3.3	RAM Blocks, 262	
9.4.3.4	Programmable Routing, 264	
9.4.3.5	Arithmetic Resources in Xilinx FPGAs, 264	
9.4.4	FPGA Generic Design Flow, 264	
9.5	Hardware Description Languages (HDLs), 267	
9.5.1	Today's and Tomorrow's HDLs, 267	
9.6	Further Readings, 268	
9.7	Bibliography, 268	
10	Circuit Synthesis: General Principles	271
10.1	Resources, 272	
10.2	Precedence Relation and Scheduling, 277	
10.3	Pipeline, 281	
10.4	Self-Timed Circuits, 282	
10.5	Bibliography, 288	
11	Adders and Subtractors	289
11.1	Natural Numbers, 289	
11.1.1	Basic Adder (Ripple-Carry Adder), 289	
11.1.2	Carry-Chain Adder, 292	
11.1.3	Carry-Skip Adder, 294	
11.1.4	Optimization of Carry-Skip Adders, 298	
11.1.5	Base- B^s Adder, 301	
11.1.6	Carry-Select Adder, 303	
11.1.7	Optimization of Carry-Select Adders, 307	
11.1.8	Carry-Lookahead Adders (CLAs), 310	
11.1.9	Prefix Adders, 318	
11.1.10	FPGA Implementation of Adders, 322	
11.1.10.1	Carry-Chain Adders, 322	
11.1.10.2	Carry-Skip Adders, 323	
11.1.10.3	Experimental Results, 326	
11.1.11	Long-Operand Adders, 327	
11.1.12	Multioperand Adders, 328	
11.1.12.1	Sequential Multioperand Adders, 328	
11.1.12.2	Combinational Multioperand Adders, 330	

- 11.1.12.3 Carry-Save Adders, 333
- 11.1.12.4 Parallel Counters, 337
- 11.1.13 Subtractors and Adder-Subtractors, 344
- 11.1.14 Termination Detection, 346
- 11.1.15 FPGA Implementation of the Termination Detection, 348
- 11.2 Integers, 350
 - 11.2.1 B 's Complement Adders and Subtractors, 350
 - 11.2.2 Excess- E Adders and Subtractors, 352
 - 11.2.3 Sign-Magnitude Adders and Subtractors, 355
- 11.3 Bibliography, 357

12 Multipliers

359

- 12.1 Natural Numbers, 360
 - 12.1.1 Basic Multiplier, 360
 - 12.1.2 Sequential Multipliers, 363
 - 12.1.3 Cellular Multiplier Arrays, 363
 - 12.1.3.1 Ripple-Carry Multiplier, 365
 - 12.1.3.2 Carry-Save Multiplier, 368
 - 12.1.3.3 Figures of Merit, 370
 - 12.1.4 Multipliers Based on Dissymmetric $B^r \times B^s$ Cells, 370
 - 12.1.5 Multipliers Based on Multioperand Adders, 378
 - 12.1.6 Per Gelsia Multiplication Arrays, 383
 - 12.1.6.1 Introduction, 383
 - 12.1.6.2 Adding Tree for Base- B Partial Products, 384
 - 12.1.7 FPGA Implementation of Multipliers, 386
- 12.2 Integers, 388
 - 12.2.1 B 's Complement Multipliers, 388
 - 12.2.2 Booth Multipliers, 390
 - 12.2.2.1 Booth-1 Multiplier, 390
 - 12.2.2.2 Booth-2 Multiplier, 392
 - 12.2.2.3 Signed-Digit Multiplier, 397
 - 12.2.3 FPGA Implementation of the Booth-1 Multiplier, 404
- 12.3 Bibliography, 406

13 Dividers

407

- 13.1 Natural Numbers, 407
- 13.2 Integers, 415
 - 13.2.1 Base-2 Nonrestoring Divider, 415
 - 13.2.2 Base- B Nonrestoring Divider, 421

13.2.3	SRT Dividers, 424	
13.2.3.1	SRT-2 Divider, 424	
13.2.3.2	SRT-2 Divider with Carry-Save Computation of the Remainder, 428	
13.2.3.3	FPGA Implementation of the Carry-Save SRT-2 Divider, 434	
13.2.4	SRT-4 Divider, 435	
13.2.5	Convergence Dividers, 439	
13.2.5.1	Newton–Raphson Divider, 439	
13.2.5.2	Goldschmidt Divider, 441	
13.2.5.3	Comparative Data Between Newton–Raphson (<i>NR</i>) and Goldschmidt (<i>G</i>) Implementations, 444	
13.3	Bibliography, 444	
14	Other Arithmetic Operators	447
14.1	Base Conversion, 447	
14.1.1	General Base Conversion, 447	
14.1.2	BCD to Binary Converter, 449	
14.1.2.1	Nonrestoring 2^p Subtracting Implementation, 449	
14.1.2.2	Shift-and-Add BCD to Binary Converter, 450	
14.1.3	Binary to BCD Converter, 452	
14.1.4	Base- B to RNS Converter, 455	
14.1.5	CRT RNS to Base- B Converter, 456	
14.1.6	RNS to Mixed-Radix System Converter, 458	
14.2	Polynomial Computation Circuits, 463	
14.3	Logarithm Operator, 467	
14.4	Exponential Operator, 468	
14.5	Sine and Cosine Operators, 470	
14.6	Square Rooters, 472	
14.6.1	Restoring Shift-and-Subtract Square Rooter (Naturals), 472	
14.6.2	Nonrestoring Shift-and-Subtract Square Rooter (Naturals), 475	
14.6.3	Newton–Raphson Square Rooter (Naturals), 477	
14.7	Bibliography, 479	
15	Circuits for Finite Field Operations	481
15.1	Operations in Z_m , 481	
15.1.1	Adders and Subtractors, 481	
15.1.2	Multiplication, 484	
15.1.2.1	Multiply and Reduce, 484	

15.1.2.2	Shift and Add, 485	
15.1.2.3	Montgomery Multiplication, 487	
15.1.2.4	Modulo $(B^k - c)$ Reduction, 490	
15.1.2.5	Exponentiation, 494	
15.2	Inversion in $GF(p)$, 497	
15.3	Operations in $Z_p[x]/f(x)$, 500	
15.4	Inversion in $GF(p^n)$, 504	
15.5	Bibliography, 510	
16	Floating-Point Unit	513
16.1	Floating-Point System Definition, 513	
16.2	Arithmetic Operations, 515	
16.2.1	Addition of Positive Numbers, 515	
16.2.2	Difference of Positive Numbers, 517	
16.2.3	Addition and Subtraction, 518	
16.2.4	Multiplication, 520	
16.2.5	Division, 521	
16.2.6	Square Root, 522	
16.3	Rounding Schemes, 524	
16.4	Guard Digits, 525	
16.5	Adder-Subtractor, 527	
16.5.1	Alignment, 527	
16.5.2	Additions, 529	
16.5.3	Normalization, 530	
16.5.4	Rounding, 530	
16.6	Multiplier, 537	
16.7	Divider, 542	
16.8	Square Root, 546	
16.9	Comments, 548	
16.10	Bibliography, 548	
	Index	549

PREFACE

From the beginnings of digital electronic science, the synthesis of circuits carrying out arithmetic operations has been a central topic. As a matter of fact, it is an activity directly related to computer development. From then on, a well-known technical discipline was born: computer arithmetic. Traditionally, the study of arithmetic circuits has been oriented toward applications to general-purpose computers, which provide the most important applications of digital circuits. However, the electronic market share corresponding to specific systems (embedded systems) is significant. It is important to point out that the huge business volume that corresponds to general-purpose computers (personal computers, servers, main frames) is distributed among a relatively reduced number of different models. Therefore the number of designers involved in general-purpose computer development is not as big as it might seem and is much less than the number of engineers dedicated to production and sales. The case of embedded systems is different. Embedded systems are circuits designed for specific applications (special-purpose devices), so a great diversity of products exist in the market, and the design effort per fabricated unit can be a lot bigger than in the case of general-purpose computers. In consequence, the design of specific computers is an activity in which numerous engineers are involved, in all type of companies—even small ones—within numerous countries.

In this book methods and examples for synthesis of arithmetic circuits are described with an emphasis somewhat different from the classic texts on computer arithmetic.

- It is not limited to the description of the arithmetic units of computers.
- Descriptions of computation algorithms are presented in a section apart from the one dedicated to their materialization or implementation by digital circuits. The development of an embedded system is an operation of hardware–software codesign for which it is not known beforehand what tasks will be executed by a microprocessor and what other tasks by specific coprocessors. For this reason, it

appeared useful to describe the algorithms in an independent manner, without any assumption on subsequent executions by an existent processor (software) or by a new customized circuit (hardware).

- A special, although not exclusive, importance has been given to user programmable devices (field programmable devices such as FPGAs), especially to the families Spartan II and Virtex. Those devices are very commonly used for the realization of specific systems, mainly in the case of small series and prototypes. The particular architecture of those components leads the designer to use synthesis techniques somewhat different from the ones applied for ASICs (application-specific integrated circuits) for which standard cell libraries exist.
- In what concern circuits description, logic schemes are presented, sometimes with some VHDL models, in such a way that the corresponding circuits can easily be simulated and synthesized.

After an introductory chapter, the book is divided in two parts. The first one is dedicated to mathematical aspects and algorithms: mathematical background (Chapter 2), number representation (Chapter 3), addition and subtraction (Chapter 4), multiplication (Chapter 5), division (Chapter 6), other arithmetic operations (Chapter 7), and operations in finite fields (Chapter 8). The second part is dedicated to the central topic—the synthesis of arithmetic circuits: hardware platforms (Chapter 9), general principles of synthesis (Chapter 10), adders and subtractors (Chapter 11), multipliers (Chapter 12), dividers (Chapter 13), other arithmetic primitives (Chapter 14), operators for finite fields (Chapter 15), and floating-point unit.

Numerous VHDL models, and other source files, can be downloaded from <http://www.ii.uam.es/~gsutter/arithmetic/>. This will be indicated in the text (e.g., *complete VHDL source code available*). As regards the VHDL models, they are of two types: some of them have been developed for simulation purposes only, so the working of the corresponding circuit can be observed; others are synthesizable models that have been implemented within commercial programmable components (FPGA's).

The authors thank the people who have helped them in developing this book, especially Dr. Tim Bratten, for correcting the text, and Paula Mirón, for the cover design. They are grateful to the following universities for providing them the means for carrying this work through to a successful conclusion: University Rovira i Virgili (Tarragona, Spain), University Rey Juan Carlos (Madrid, Spain), State University UNCPBA (Tandil, Argentina), University FASTA (Mar del Plata, Argentina), and Autonomous University of Madrid (Spain).

JEAN-PIERRE DESCHAMPS
University Rovira i Virgili

GÉRY JEAN ANTOINE BIOUL
National University of the Center of the Province of Buenos Aires

GUSTAVO D. SUTTER
University Autònoma of Madrid

ABOUT THE AUTHORS

Jean-Pierre Deschamps received a MS degree in electrical engineering from the University of Louvain, Belgium, in 1967, a PhD in computer science from the Autonomous University of Barcelona, Spain, in 1982, and a PhD degree in electrical engineering from the Polytechnic School of Lausanne, Switzerland, in 1983. He has worked in several companies and universities. He is currently a professor at the University Rovira i Virgili, Tarragona, Spain. His research interests include ASIC and FPGA design, digital arithmetic, and cryptography. He is the author of six books and about a hundred international papers.

Géry Jean Antoine Bioul received a MS degree in physical aerospace engineering from the University of Liège, Belgium. He worked in digital systems design with PHILIPS Belgium and in computer-aided industrial logistics with several Fortune-100 U.S. companies in the United States, and Africa. He has been a professor of computer architecture in several universities mainly in Africa and South America. He is currently a professor at the State University UNCPBA of Tandil (Buenos Aires), Argentina, and a professor consultant at the Saint Thomas University FASTA of Mar del Plata (Buenos Aires), Argentina. His research interests include logic design and computer arithmetic algorithms and implementations. He is the author of about 50 international papers and patents on fast arithmetic units.

Gustavo D. Sutter received a MS degree in Computer Science from the State University UNCPBA of Tandil (Buenos Aires), Argentina, and a PhD degree from the Autonomous University of Madrid, Spain. He has been a professor at the UNCPBA, Argentina and is currently a professor at the University Autonoma of Madrid, Spain. His research interests include ASIC and FPGA design, digital arithmetic, and development of embedded systems. He is the author of about 30 international papers and communications.

1

INTRODUCTION

The design of embedded systems, that is, circuits designed for specific applications, is based on a series of decisions as well as on the use of several types of development techniques. For example:

- Selection of the data representation
- Generation or selection of algorithms
- Selection of hardware platforms
- Hardware–software partitioning
- Program generation
- New hardware synthesis
- Cosimulation, coemulation, and prototyping

Some of these activities have a close relationship with the study of arithmetic algorithms and circuits, especially in the case of systems including a great amount of data processing (e.g., ciphering and deciphering, image processing, digital signature, biometry).

1.1 NUMBER REPRESENTATION

When using general-purpose equipment, the designer has few possible choices concerning the internal representation of data. He must conform to some fixed

and predefined data types such as *integer*, *floating-point*, *double precision*, and *character*. On the contrary, if a specific system is under development, the designer can choose, for each data, the most convenient type of representation. It is no longer necessary to choose some standard fixed-point or floating-point numeration system. Nonstandard specific formats can be used. In Chapter 3 the main number representation methods will be defined.

1.2 ALGORITHMS

Every complex data processing operation must be decomposed into simpler operations — the computation primitives — executable either by the main processor or by some specific coprocessor. The way the computation primitives are used in order to perform the complex operation is what is meant by *algorithm*. Obviously, knowledge of algorithms is of fundamental importance for developing arithmetic procedures (software) and circuits (hardware). It is the topic of Chapters 4–8.

1.3 HARDWARE PLATFORMS

The selection of a hardware platform is based on the answer to the following question. How do we get the desired behavior at the lowest cost, while fulfilling some additional constraints? As a matter of fact, the concept of cost must be carefully defined in each particular case. It can cover several aspects: for example, the unit production cost, the nonrecurring engineering costs, and the implicit cost for a late introduction of the product to the market. Some examples of additional technical constraints are the size of the system, its power consumption, and its reliability and maintainability.

For systems requiring little data processing capability, *microcontrollers* and low-range *microprocessors* can be the best choice. If the computation needs are greater, more powerful microprocessors, or even *digital signal processors* (DSPs), should be considered. This type of solution (microprocessors and DSPs) is very flexible as the development work mainly consists in generating programs.

For getting higher performances, it may be necessary to develop specific circuits. A first option is to use a programmable device, for example, a *field-programmable gate array* (FPGA). It could be an interesting option for prototypes and small series. For greater series, an *application-specific integrated circuit* (ASIC) should be developed. ASIC vendors offer several types of products: for example, *gate arrays*, with relatively small prototyping costs, or *standard cell libraries*, integrating a complete *system-on-chip* (SOC) including processors, program memories, data memories, logic, macrocells, and analog interfaces.

A brief presentation of the most common hardware platforms is given in Chapter 9.

1.4 HARDWARE–SOFTWARE PARTITIONING

The hardware–software partitioning consists of deciding which operations will be executed by the central processing unit (the software) and which ones by specific coprocessors (the hardware). As a matter of fact, the platform selection and the hardware–software partitioning are tightly related operations. For systems requiring little data processing capability, the whole system is implemented in software. If higher performances are necessary, the noncritical operations, as well as control of the operation sequence, are executed by the central processing unit, while the critical ones are implemented within specific coprocessors.

1.5 SOFTWARE GENERATION

The operations belonging to the software block of the chosen partition must be programmed. In Chapters 4–8 the algorithms are presented in an Ada-like language that can easily be translated to C or even to the assembly language of the chosen microprocessor.

1.6 SYNTHESIS

Once the hardware–software partition has been defined, all the tasks assigned to the specific hardware (FPGA, ASIC) must be translated into circuit descriptions. Some important synthesis principles and methods are described in Chapter 10. The synthesis of arithmetic circuits, based on the algorithms of Chapters 4–8, is the topic of Chapters 11–15, and an additional chapter (16) is dedicated to the implementation of floating-point arithmetic.

1.7 A FIRST EXAMPLE

Common examples of application fields resorting to embedded solutions are cryptography, access control, smart cards, automotive, avionics, space, entertainment, and electronic sales outlets. In order to illustrate the main steps of the design process, a small digital signature system will now be developed (complete assembly language and VHDL code available).

1.7.1 Specification

The system under development (Figure 1.1) has three inputs,

- `character` is an 8-bit vector.
- `new_character` is a signal used for synchronizing the input of successive characters.
- `sign` is a control signal ordering the computation of a *digital signature*.

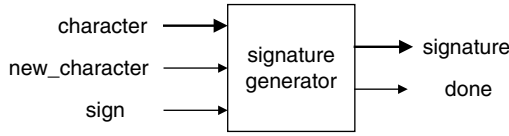


Figure 1.1 System under development.

and two outputs,

- done is a status variable indicating that the signature computation has been completed,
- signature is a 32-bit vector, namely, the signature of the message.

The working of the system is shown in Figure 1.2: a sequence c_1, c_2, \dots, c_n of any number n of characters (the message), synchronized by the signal new_character, is inputted. When the sign control signal goes high, the done flag is lowered and the signature of the message is computed. The done flag will be raised as soon as the signature s is available.

In order to sign the message two functions must be defined:

- a hash function associating a 32-bit vector (the summary) to every message, whatever its length;
- an encode function computing the signature corresponding to the summary.

The following (naive) hash function is used:

Algorithm 1.1 Hash Function

```

summary:=0;
while not(end_of_message) loop
  get(character);
  a:=(summary(7 downto 0)+character) mod 256;
  summary(23 downto 16):=summary(31 downto 24);
  summary(15 downto 8):=summary(23 downto 16);
end while

```

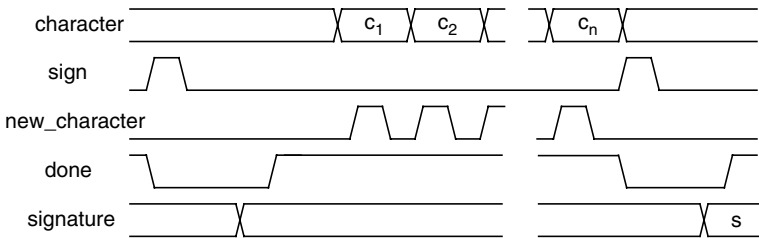


Figure 1.2 Input and output signals.

- Dot
 - component, *see* Dot operator
 - diagram, 381
 - operation, 58
 - operator, 302, 310
 - procedure, 60, 62
- DSP, *see* Digital signal processor
- Embedded
 - processor, 248, 250
 - system, 1, 251, 268
- Encoder
 - Booth, 397
 - stored-carry, 333, 338
- Equivalence
 - class, 20, 32
 - relation, 32
- Error
 - maximum, 51, 52, 53
 - maximum relative, 52, 53
 - relative, 51
- Euclidean algorithm, 17
 - extended, 18, 22
 - extended for polynomials, 31
 - for polynomials, 28
- Euler phi function, 24
- Excess- E
 - adder, 352
 - addition, 78
 - representation, 43
 - sign change, 79
 - subtraction, 78
 - subtractor, 352
- Excess- k , 166
- Exponent, 182, 514
- Exponential, 165, 180, 181, 463, 468
 - additive normalization, 188, 191
 - binary computation, 468
 - computation circuit, 469
 - convergence methods, 184
- Exponentiation, 6
 - modulo m , 221, 494
- Exponentiator, 10, 12
- Extended Booth, 359

- Fermat's little theorem, 24, 33
- Field, 27, 32
 - finite field operations, 211, 481
- Field programmable gate array (FPGA), 2, 8, 82, 258, 271, 359, 366, 386, 387, 404, 434
 - arithmetic resources in Xilinx, 264
 - basic concepts, 258
 - configurable logic blocks (CLB), 262
 - generic design flow, 264
 - input/output blocks (IOB), 262
 - logic block, 260
 - look-up table (LUT), 259
 - programming methods, 258
 - Xilinx specifics, 260
- Fixed-point numeration system, 51
- Flag
 - done, 284
 - stat-done, 347
- Floating-point numbers
 - adder, 527
 - adder-subtractor, 537
 - addition, 515, 518
 - arithmetic operations, 515
 - chopping, 524
 - difference, 517
 - divider, 542
 - division, 521
 - multiplication, 520
 - multiplier, 537
 - multiplier with stored-carry encoding, 541
 - normalization, 515, 523
 - normalization circuit, 530
 - overflow, 524
 - rounding, 515
 - rounding circuit, 530
 - rounding schemes, 524
 - square root, 522, 546
 - subtraction, 518
 - subtractor, 527
 - truncation, 524
 - underflow, 524
 - unit, 513
- Floating-point numeration system, 51, 52, 513, 514
 - ANSI/IEEE, 44, 53
 - unbiased, 525
- FPGA, *see* Field-programmable gate array
- Frobenius numbers, 236
- Full adder
 - augmented, 316
 - cell, 289
 - generalized, 370
 - decimal, 291
- Full subtractor cell, 344
- Function approximation, 35

- Galois field, 33
 - operations, 222
- Garner's algorithm, 22, 23, 180
- Gate array, 2. *See also* Field programmable gate array; Integrated circuit
- Gcd, *see* Greatest common divisor

- Generalized full adder (GFA), 370
- Generalized generate function, 58, 301
- Generalized half adder (GHA), 370
- Generalized Hörner expansion (GHE), 184, 463, 466
- Generate function, 56, 292, 293
 - generalized, 58, 301
- Generator, 25, 26
- $GF(p)$
 - inversion, 497
 - operations, 222
- $GF(p^n)$
 - exponentiation, 510
 - inversion, 229, 230, 231, 233, 235
 - inverter, 504, 506
 - multiplier, 501
 - operations, 228
- GFA, *see* Generalized full adder
- GHA, *see* Generalized half adder
- GHE, *see* Generalized Hörner expansion
- Goldschmidt, 441, 444
 - algorithm, 109, 159
- Greatest common divisor (gcd), 16, 17, 18, 28
- Group, 25
 - Abelian, 26
 - commutative, 26
 - cyclic, 25, 26, 33
 - multiplicative, 24
- Guard digits, 525

- Half adder cell, 291
 - generalized, 370
- Hardware description language, 267
 - Verilog, 267
 - VHDL, 267
- Hardware platform, 2, 8, 239
- Hash function, 4
- HDL, *see* Hardware description language
- Hexadecimal system, 6
- Hörner, 166, 170, 181
 - expansion, 84, 360
 - generalized expansion, 184, 463, 466
 - scheme revisited, 183

- IC, *see* Integrated circuit
- Identity element
 - additive, 27
 - multiplicative, 27
- Input/output modules, 241
- Integrated circuit, 239–255
 - application specific, *see* ASIC
- Integrated system
 - basic blocks, 240
 - cost, 242
 - design metrics, 241
 - performance metric, 244
- Intellectual property (IP), 240
- Inverse, multiplicative, 20
- Inversion
 - Itoh–Tsujii algorithm, 231
 - in $GF(p)$, 497
 - in $GF(p^n)$, 229–231, 233, 235
 - in Z_p , 223
- Inverter
 - $GF(p)$, 497
 - $GF(p^n)$, 504, 506
 - modulo p , 498
- IP, *see* Intellectual property

- Ladner–Fischer carry chain, 321, 322
- Latchless pipelining technique, 282
- Latency, 271, 281, 283
 - average, 284
- Logarithm, 165, 180, 182, 463, 467
 - computation circuit, 468
 - convergence methods, 184
 - multiplicative normalization, 184, 187
- Long-operand adder, 327
- Look-up tables, 82, 165. *See also* Field programmable gate array, look up tables
- LUT, *see* Look-up tables

- MacLaurin expansion, 159
- MacLaurin series, 35
- Manchester carry chain, 293
- Mantissa, 182
- Market window, 244
- Memory, 240
- Mersenne, 458
- Microcontroller, 2, 248
- Microprocessor, 2, 247
- Mixed numeration system, 178
- Mixed-radix, 459
 - unsigned digit system, 178
- Modulo (B^k-c) reduction, 490
- Modulo m
 - adder, 481
 - addition, 212
 - exponentiation, 221, 494
 - multiplier, 484. *See also* Modulo m multiplication
 - reduction, 220
 - subtraction, 213
 - subtractor, 481

- Modulo m multiplication, 213
 - Montgomery product, 216, 218
 - multioperand, 220
 - multiply and reduce, 214, 484
 - shift and add, 214, 485
- Modulo p inverter, 498
- Montgomery product, 211, 216, 218, 487
 - multioperand, 220
- Moore's law, 243
- Multioperand, 384
 - adder combinational, 330
 - adder sequential, 328, 329
 - adders, 378
 - addition array, 331
 - addition basic algorithm, 67
 - addition tree, 332
- Multiplexer carry-skip, 324, 325
- Multiplicand, 360
- Multiplication
 - array, 359, 363
 - Booth, 96
 - Booth for base- B numbers, 102
 - Booth- r , 97
 - B 's complement, 91
 - cellular carry-save, 88
 - cellular ripple-carry, 86
 - cellular shift and add, 86
 - dissymmetric cell, 370
 - extended shift and add, 86
 - floating point numbers, 520
 - Hörner shift and add, 84
 - integer(s), 91
 - long-operand, 90
 - mod B^{n+m} B 's complement, 92
 - modulo m , 7, 213
 - Montgomery product, 211
 - natural, 7
 - natural numbers, 82
 - Per Gelosia, 359, 383
 - Per Gelosia signed-digit, 98
 - polynomials, 227
 - post-correction 2's complement, 96
 - post-correction B 's complement, 93, 389
 - shift and add, 83
 - signed shift and add, 93
- Multiplicative normalization, 467
- Multiplicator, 360
- Multiplier
 - base- B , 361
 - based on multioperand adders, 378
 - Booth, 390
 - Booth-1, 390, 404
 - Booth-2, 392
 - Booth-3, 395
 - Booth- r , 395
 - B 's complement, 388
 - carry-save, 368, 388
 - cellular, 363
 - floating point, 537
 - FPGA implementation, 386, 387, 404
 - modulo m Montgomery, 487
 - modulo m , 484
 - ripple-carry, 360, 365, 367
 - sequential, 363
- Newton–Raphson, 109, 155, 180, 208, 439, 444, 477
 - convergence graph, 156
- Nonrecurring engineering cost (NRE), 242, 252
- Normalization
 - additive, 468
 - circuit, 530
 - exponential additive, 188, 191
 - floating point numbers, 515, 523
 - multiplicative, 467
- NRE, *see* Nonrecurring engineering cost
- Number
 - integer, 15, 42
 - natural, 15, 39
 - representation, 1, 6
- Number representation system(s), 39
 - positional, 40
 - weighted, 39
- Number theory, 15
- Numeration system
 - base- B , 40
 - binary, 40
 - decimal, 40
 - fixed-point, 51
 - hexadecimal, 40
 - mixed-radix, 22, 40
 - octal, 40
- Order of an element, 25, 26
- Overflow
 - detection, 74
 - floating-point, 524
- Parallel (3,2)-counter, 337
- Parallel counter, 379
- Partial remainder–divisor plot diagram, *see* P - D diagram
- Partitioning hardware–software, 3, 8
- P - D diagram, 139, 140
- Pentium bug, 148
- Per Gelosia, 98, 383, 401

- $\phi(n)$, 24
- PicoBlaze, 8, 9, 12
- Pipeline, 281, 283
 - latchless technique, 282
- (p,k) -counter, 378
- PLD, *see* Programmable logic device
- Polynomial, 27
 - approximation, 180, 183
 - computation, 463
 - constant, 27
 - irreducible, 28
 - monic, 28
 - multiplication, 227
 - zero, 27
 - 0-degree, 33
- Power consumption, 242, 245
- Precedence graph, 277
 - relation, 277
 - scheduled, 278
- Precision
 - absolute, 51
 - relative, 51
- Prefix carry chain, 318
- Prescaling, 185
- Prime, 16
 - relatively, 16, 24
- Primitive element, 25
- Private key, 5
- Processor, 245. *See also* Application specific
 - instruction set processors; Complex instruction set computers; Digital signal processor; Reduced instruction set computer; Very long instruction word
 - embedded, 248, 250
 - general-purpose (GPP), 246
 - harvard architecture, 246
 - microcontroller, 2, 248
 - microprocessor, 2, 247
 - programming instruction-set, 251
 - superscalar, 247
 - von Neumann architecture, 246
- Product, modular, 8
- Programmable logic, 256. *See also* Field programmable gate array
- Programmable logic device (PLD), 256
- Propagate function, 56, 292
 - generalized, 58, 301
- Prototype, 12
- Prototyping board, 8
- Q-select, 143
- Quotient, 16, 28, 110
- Quotient-digit
 - nonredundant, 148
 - redundant, 149
- Range
 - of positive numbers, 52
 - of represented numbers, 52
 - relative, 53
- Rate, sample, 281
- Reciprocal computation, 157
- Reduced instruction set computer (RISC), 245, 248
- Redundant
 - base- B coding, 166
 - set of digits, 50
 - systems, 166
- Remainder, 16, 28, 110
 - carry-stored representation, 132
- Req signal, 284
- Residue number system (RNS), 39, 42, 173, 455, 458
 - to mixed-radix, 458
 - representation, 42
- Residues modulo m_i , 173
- Resource
 - computation, 272
 - connection, 272
 - memory, 272
- Ring, 27
 - commutative, 32
 - finite, 211
 - polynomial, 27, 211
- Ripple-carry
 - adder, 289, 360
 - array, 366, 372
 - B -ary adder, 384
 - multioperand adders, 378
 - multiplier, 360, 365
- RISC, *see* Reduced instruction set computer
- RNS, *see* Residue number system
- Robertson diagram, 119, 139
- Rounding
 - circuit, 530
 - floating-point numbers, 515
 - schemes, 524
- Sample rate, 281
- Scheduling, 277
- SDFA, *see* Signed-digit, full adder
- Self-timed
 - adder, 285
 - circuit, 282
 - pipelined circuit, 284
- Self-timing, 282
- Semigroup, 26