

THIRD EDITION

BeBOP

TO THE BOOLEAN

BOOGIE

An Unconventional
Guide to Electronics



Clive "Max" Maxfield



Bebop to the Boolean Boogie

This page intentionally left blank

Bebop to the Boolean Boogie

An Unconventional Guide

to Electronics

Third Edition

Clive "Max" Maxfield



AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier



Newnes is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Copyright © 2009, Elsevier Inc. All rights reserved.

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, or otherwise, without the prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK; phone: (+44) 1865 843830, fax: (+44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (<http://www.elsevier.com>), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

- ⊗ Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

Library of Congress Cataloging-in-Publication Data
Application submitted.

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

ISBN: 978-1-85617-507-4

For information on all Newnes publications,
visit our web site at: www.books.elsevier.com

08 09 10 11 12 13 10 9 8 7 6 5 4 3 2 1

Printed in Canada.

**Working together to grow
libraries in developing countries**

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

*This book is dedicated to my Auntie Barbara, whose assiduous scrubbing in my younger years has
left me the proud owner of the cleanest pair of knees in the known universe!*

This page intentionally left blank

Contents

ABOUT THE AUTHOR	xvii
FOREWORD	xix
ABOUT THIS BOOK	xxi
ACKNOWLEDGMENTS	xxv

SECTION 1 • Fundamentals

CHAPTER 1	Analog Versus Digital.....	3
	It Was a Dark and Stormy Night	3
	Analog Versus Digital Views of the World.....	4
	Multi-Value Digital Systems.....	5
	Experiments with Bricks.....	6
CHAPTER 2	Atoms, Molecules, and Crystals.....	11
	Protons, Neutrons, and Electrons.....	11
	Quantum Levels and Electron Shells	13
	Making Molecules	13
	Crystals and Other Structures	15
CHAPTER 3	Conductors, Insulators, and Other Stuff	17
	Conductors and Insulators.....	17
	Voltage, Current, and Resistance.....	18
	Resistance and Resistors.....	19
	Capacitance and Capacitors.....	21
	Inductance and Inductors.....	23
	Memristance and Memristors	28
	Impedance and Reactance.....	28
	Admittance, Conductance, and Susceptance	29
	Unit Qualifiers	30
CHAPTER 4	Semiconductors (Diodes and Transistors)	33
	Herding Wild Electrons.....	33
	The Electromechanical Relay.....	33
	The First Vacuum Tubes	35
	Semiconductors.....	36
	Semiconductor Diodes	37

Bipolar Junction Transistors (BJTs).....	39
Metal-Oxide Semiconductor Field-Effect Transistors (MOSFETs)	41
The Transistor as a Switch	43
Gallium Arsenide Semiconductors.....	44
Light-Emitting Diodes (LEDs).....	45
Organic LEDs (OLEDs)	46
Active Versus Passive and Electric Versus Electronic	47
 CHAPTER 5	
Primitive Logic Functions	49
Switch Representations of AND and OR Functions	49
FALSE and TRUE Versus OPEN and CLOSED.....	50
BUF and NOT Functions.....	51
“Connect the NOTs”	52
AND, OR, and XOR Functions.....	52
NAND, NOR, and XNOR Functions.....	53
Not a Lot	55
Functions Versus Gates	56
 CHAPTER 6	
Using Transistors to Build Logic Gates	57
NMOS, PMOS, and CMOS	57
Using 0s and 1s Instead of Fs and Ts.....	57
NOT and BUF Gates	58
NAND and AND Gates.....	60
NOR and OR Gates.....	61
XNOR and XOR Gates.....	62
XNOR and XOR Gates: Pass-Transistor Implementations.....	63
Pass-Transistor Logic.....	65
 CHAPTER 7	
Alternative Number Systems	67
Fingers, Toes, and Pebbles	67
Bones with Notches	67
Tally Sticks: The Hidden Dangers	68
The Abacus.....	69
Roman Numerals.....	69
Decimal (Base-10).....	70
Duo-Decimal (Base-12).....	71
Sexagesimal (Base-60).....	73
The Concepts of Zero and Negative Numbers	74
Vigesimal (Base-20).....	76

Jobs Abound for Time-Travelers	76
Quinary (Base Five)	77
Binary (Base-2).....	78
Octal (Base-8) and Hexadecimal (Base-16).....	80
Way Back in the Mists of Time.....	82
Representing Numbers Using Powers	82
Lucky and Unlucky Numbers.....	84
Tertiary Logic.....	85
CHAPTER 8	
Binary Arithmetic	87
Before We Start	87
Unsigned Binary Numbers.....	87
Adding Unsigned Binary Numbers.....	88
Nines' and Ten's Complements.....	89
Subtracting Unsigned Binary Numbers.....	91
Sign-Magnitude Binary Numbers	93
Signed Binary Numbers.....	94
Adding Signed Binary Numbers	95
Subtracting Signed Binary Numbers	96
Binary Multiplication	97
Binary Division.....	98
CHAPTER 9	
Boolean Algebra	99
Cabbages, Parrots, and Buckets of Burning Oil	99
Primitive Logic Functions.....	100
Combining a Single Variable with Logic 0 or Logic 1	102
The Idempotent Rules	102
The Complementary Rules.....	102
The Involution Rule.....	103
The Commutative Rules	104
The Associative Rules.....	104
Precedence of Operators.....	105
The First Distributive Rule.....	105
The Second Distributive Rule	105
The Simplification Rules.....	106
DeMorgan Transformations.....	106
Minterms and Maxterms	112
Sum-of-Products and Product-of-Sums	112
Canonical Forms.....	114
An Interesting Conundrum.....	114

CHAPTER 10	Karnaugh Maps.....	117
	The Tree of Porphyry.....	117
	John Venn and his Venn Diagrams	117
	Allan Marquand and Lewis Carroll	117
	Maurice Karnaugh and Karnaugh Maps	118
	Minimization Using Karnaugh Maps.....	119
	Grouping Minterms	120
	Incompletely Specified Functions.....	122
	Populating Maps Using 0s Versus 1s.....	123
CHAPTER 11	Slightly More Complex Functions.....	125
	First Gather a Bucket of Logic Gates.....	125
	Scalar Versus Vector Notation.....	125
	Equality Comparators	126
	Multiplexers	127
	Decoders	129
	Tri-State Functions	130
	Combinational Versus Sequential Functions	132
	RS Latch (NOR Implementation).....	132
	RS Latch (NAND Implementation).....	137
	D-Type Latches	138
	D-Type Flip-flops.....	139
	Implementing a D-Type Flip-flop	142
	JK and T Flip-flops.....	143
	Shift Registers.....	144
	Counters	146
	Setup and Hold Times.....	148
	Brick by Brick	149
CHAPTER 12	State Machines	151
	"Is That a Gizmo in Your Pocket, Or . . ."	151
	State Diagrams.....	152
	State Tables	153
	State Machines.....	154
	State Assignment	155
	Don't Care States, Unused States, and Latch-Up Conditions....	158
CHAPTER 13	Analog-to-Digital and Vice Versa.....	161
	Setting the Scene	161
	Analog-to-Digital	162
	Digital-to-Analog	164
	DSP Versus DSP.....	165

Analog Signal Processing (ASP)	165
Digital Signal Processing (DSP)	166
DSP Examples.....	167
What Implements the Digital Signal Processing?.....	167

SECTION 2 • Components and Processes

CHAPTER 14

Integrated Circuits (ICs)	173
The First Integrated Circuits	173
An Overview of the Fabrication Process	175
A Slightly More Detailed Look at the Fabrication Process	176
An Introduction to the Packaging Process.....	181
Integrated Circuits Versus Discrete Components.....	185
Different Types of ICs	186
TTL, ECL, and CMOS	187
Core Supply Voltages	187
Equivalent Gates	188
Device Geometries	188
What Comes After Optical Lithography?.....	190
How Many Transistors?.....	192
Moore's Law.....	192

CHAPTER 15

Memory ICs	193
RAMs and ROMs.....	193
Cells, Words, and Arrays.....	195
Addressing a Word in Memory.....	196
Kilo, Mega, Giga, Tera, Etc.....	196
Bits and Bytes.....	197
ROM Control Decoding.....	197
RAM with Separate Data In and Data Out Busses.....	199
RAM with Single Bidirectional Bus.....	200
Increasing Width and Depth.....	201
Mask-Programmed ROMs.....	202
PROMs	203
EPROMs	205
EEPROMs/E ² PROMs	207
FLASH	207
SRAMs and DRAMs	208
SDRAMs.....	208
DDR, DDR2, DDR3, QDR, RAMBUS, Etc.	210
SIMMs, DIMMs, and RIMMs.....	210

ECC Memory.....	211
MRAMs.....	211
nvRAMs, FRAMs, PRAMs, RRAMs, CBRAMs, SONOS, Etc	211
CHAPTER 16	213
A Simple Programmable Function.....	213
Fusible-Link Technologies.....	214
Antifuse Technologies	215
EPROM, E ² PROM, FLASH, and SRAM Technologies	217
The First Programmable Logic Devices (PLDs)	217
PROMs	218
PLAs.....	221
PALs and GALs	223
Additional Programmable Options.....	224
Introducing CPLDs	224
Introducing FPGAs.....	227
Alternative FPGA Architectures.....	229
Alternative FPGA Configuration Technologies.....	232
Mixed-Signal FPGAs, CSSPs, and	233
Summary	233
CHAPTER 17	235
Introducing ASICs.....	235
Full Custom Devices.....	236
Gate Arrays.....	236
High-Level View of the Gate Array Design Flow	238
Standard Cell Devices.....	240
High-Level View of the Standard Cell Design Flow	241
1T Versus 6T SRAM	241
Structured ASICs	242
Input/Output (I/O) Cells and Pads	245
ASICs Versus ASSPs.....	246
Who Are All the Players?	246
Summary	248
CHAPTER 18	251
Not Much Fun	251
The First Circuit Boards	251
PCBs and PWBs.....	252
RoHS and Lead-Free Solder	252
Subtractive Processes.....	253

Additive Processes.....	255
Single-Sided Boards	257
Lead Through-Hole (LTH).....	259
Wave Soldering	259
Surface Mount Technology (SMT).....	260
Double-Sided Boards.....	262
Holes Versus Vias	264
Multilayer Boards.....	265
Through-Hole, Blind, and Buried Vias.....	266
Power and Ground Planes.....	267
High Density Interconnect (HDI) and Microvia Technologies ...	270
Backplanes and Motherboards.....	271
Conductive Ink Technology	272
Chip-on-Board (COB)	273
Flexible Printed Circuits (FPCs).....	274
CHAPTER 19	
Hybrids.....	277
The Offspring Resulting from Crossbreeding	277
Hybrid Substrates.....	277
The Thick-Film Process.....	278
Creating Tracks.....	279
Creating Resistors.....	280
Laser Trimming	281
Creating Capacitors and Inductors.....	282
Double-sided Thick-Film Hybrids	283
Subtractive Thick-Film Technology	283
The Thin-Film Process.....	283
Laser Trimming	285
The Assembly Process	286
Attaching the Die.....	286
Wire Bonds	287
Tape-Automated Bonding	288
Flipped-Chip Techniques.....	289
Advantages of Using Bare Die.....	290
The Packaging Process.....	290
CHAPTER 20	
Advanced Packaging Techniques	293
Sliding Down the Rabbit Hole.....	293
Wire Bonds Versus Flip-Chip	293
Wire Bonding and Flip-Chip.....	294

Chip-Scale Package (CSP) Technology	294
3-D Die Stacking	295
System-in-Package (SiP), PiP, and PoP	296
A Positive Plethora of Substrates	297
An Example SiP Based on Cofired Ceramics	298
Low-Fired Cofired Ceramics	301
Assembly and Packaging	301
Pin Grid Arrays	302
Pad, Ball, and Column Grid Arrays	302
Fuzz-Buttons	304
Populating the Die	304
The Mind Boggles	305
CHAPTER 21	
Alternative and Future Technologies	307
A Smorgasbord of Technologies	307
Reconfigurable Computing	307
Elemental Computing Arrays (ECAs)	310
Optical Interconnect	314
Fiber-Optic Interconnect	314
It Pays to Keep Your Eyes Open	317
Free-Space Interconnect	317
Guided-Wave Interconnect	318
Optical Memories	320
Protein Switches and Memories	321
Electromagnetic Transistor Fabrication	324
Heterojunction Transistors	325
Buckyballs and Nanotubes	328
Diamond Substrates	331
Chemical Vapor Deposition	331
Chemical Vapor Infiltration	332
Ubiquitous Laser Beams	332
The Maverick Inventor	333
The Requirement for Single-Crystal Diamond	333
Conductive Adhesives	334
Superconductors	335
Nanotechnology	337
Back to the Water Molecule	337
Imagine a Soup	339
Once Again, the Mind Boggles	341
Summary	342

SECTION 3 • Design Tools and Stuff

CHAPTER 22	General Concepts.....	345
	Stuff, More Stuff, and Yet More Stuff.....	345
	The Origins of EDA.....	345
	Computer-Aided Design (CAD)	346
	Computer-Aided Engineering (CAE).....	346
	Designers Versus Engineers.....	347
	Electronic Design Automation (EDA)	347
	Automation.....	347
	Embedded Systems	348
	Programming Versus Hardware Design Languages.....	349
	Netlists.....	350
	Transistor-Level	350
	Gate-Level	351
	Component-Level.....	351
	Different Levels of Abstraction.....	351
	Transistor-Level	352
	Switch-Level	352
	Gate-Level	353
	Structural	353
	Functional (Boolean, RTL)	353
	Behavioral.....	354
	Algorithmic.....	354
	Different Languages	354
	Programming Languages.....	354
	Scripting Languages.....	355
	Hardware Description Languages (Digital).....	355
	Hardware Description Languages (Analog).....	358
	Verification Languages (General)	358
	Verification Languages (Formal)	358
	Electronic System Level (ESL)	359
CHAPTER 23	Design and Verification Tools.....	361
	Weasel Words	361
	Design Capture	361
	Transistor-Level and Gate-Level Netlists.....	361
	Schematic Capture.....	362
	Higher Levels of Abstraction	363
	Graphical Design Entry Lives On.....	363

Functional Verification (Simulation)	364
Formal Verification	365
Logic Synthesis	366
Layout (Place-and-Route)	367
Parasitic Extraction.....	367
Timing Analysis.....	368
Static Timing Analysis (STA)	368
Statistical Static Timing Analysis (SSTA)	369
Design for Manufacturability (DFT).....	370
And So Much More	371
Schematic Synthesis.....	371
Analog Synthesis	371
RF/Microwave Design Tools.....	372
Hardware Simulation Acceleration and Emulation.....	372
Mixed-Signal Simulation	373
Physical Verification (DRC, ERC, LVS).	373
Signal Integrity (SI) Analysis.....	374
Thermal Analysis	374
Power Analysis	374
Electromagnetic Interference and Compliance (EMI and EMC).....	374
SCAN, BIST, JTAG, etc.....	375
Automatic Test Pattern Generation (ATPG)	376
Fault Simulation	376
Turn That Frown Upside Down.....	376
APPENDIX A Assertion-Level Logic	377
APPENDIX B Positive Versus Negative Logic.....	383
APPENDIX C Reed-Müller Logic	389
APPENDIX D Gray Codes	393
APPENDIX E Linear Feedback Shift Registers (LFSRs).....	407
APPENDIX F Pass-Transistor Logic	423
APPENDIX G More on Semiconductors	427
APPENDIX H Rounding Algorithms 101	435
APPENDIX I An Interesting Conundrum.....	455
APPENDIX J A No-Holds Barred Seafood Gumbo	459
Glossary	465
Index.....	525

About the Author

Clive "Max" Maxfield is six feet tall, outrageously handsome, English and proud of it. In addition to being a hero, a trendsetter, and a leader of fashion, he is widely regarded as an expert in all aspects of electronics (at least by his mother).

After receiving his BS in Control Engineering in 1980 from Sheffield Polytechnic (now Sheffield Hallam University) in England, Max began his career as a designer of central processing units for mainframe computers. In those days of yore, Max and the rest of the team were designing silicon chips using pencil and paper, because they didn't have access to any computer-aided tools. Over the following years, Max meandered his way through most of the tools used to design chips, circuit boards, and electronic systems. These tools are now gathered under the umbrella name of *Electronic Design Automation* (EDA).

To cut a long story short, Max now finds himself President of TechBites Interactive (<http://www.techbites.com/index.html>). A marketing consultancy, TechBites specializes in communicating the value of technical products and services to nontechnical audiences through such mediums as websites, advertising, technical documents, brochures, collaterals, and multimedia.

In his spare time (Ha!), Max is the editor of the Programmable Logic DesignLine site (<http://www.pldesignline.com/>) and the executive editor of the iDesign portion of *Chip Design Magazine* (<http://www.chipdesignmag.com/>). Max is also the coeditor and copublisher of the web-delivered electronics and computing hobbyist magazine *EPE Online* (<http://www.epemag3.com/>).

In addition to numerous technical articles and papers appearing in magazines and at conferences around the world, Max is also the author and coauthor of a number of books, including *Designus Maximus Unleashed (Banned in Alabama)*, *Bebop BYTES Back (An Unconventional Guide to Computers)*, *EDA: Where Electronics Begins, The Design Warrior's Guide to FPGAs*, and *How Computers Do Math* (which features the pedagogical and phantasmagorical virtual DIY Calculator).

On the off-chance that you're still not impressed, Max was once referred to as an "*industry notable*" and a "*semiconductor design expert*" by someone famous who wasn't prompted, coerced, or remunerated in any way!

This page intentionally left blank

Foreword

My first exposure to the unique writing style of Clive (call me "Max") Maxfield was a magazine article that he cowrote with an associate. The article was technically brilliant (he paid me to say that) and very informative, but it was the short biography at the end of the piece that I enjoyed the most. I say enjoyed the most because, as you will soon learn, Max does not necessarily follow the herd or dance to the same drummer as the masses. Trade journals have a reputation for being informative and educational, but also as dry as West Texas real estate.

Anyway, Max's personally submitted biography not only included a message from his mom, but also made mention of the fact that he (Max) is taller than his coauthor, who just happened to be his boss at the time. Now to some people this may seem irrelevant, but to our readers (and Max's boss), these kind of things—trivial as they may seem to the uninitiated—are what helps us to maintain our off-grid sense of the world. Max has become, for better or worse, a part of that alternate life experience.

So now it's a couple of years later, and Max has asked me to write a few words by way of introduction to his magnum opus. Personally, I think that the title of this tome alone (hmmm, a movie?) should provide some input as to what you can expect. But, for those who require a bit more: be forewarned, dear reader, you will probably learn far more than you could hope to expect from *Bebop to the Boolean Boogie*, just because of the unique approach Max has to technical material. The author will guide you from the basics through a minefield of potentially boring theoretical mish-mash, to a Nirvana of understanding. You will not suffer that fate familiar to every reader: rereading paragraphs over and over wondering what in the world the author was trying to say. For a limey, Max shoots amazingly well and from the hip, but in a way that will keep you interested and amused. If you are not vigilant, you may not only learn something, but you may even enjoy the process. The only further advice I can give is to "expect the unexpected."

—Pete Waddell, Publisher, *Printed Circuit Design*
Literary genius (so says his mom), and taller than Max by 1½ inches

This page intentionally left blank

About this Book

NOTE FROM THE AUTHOR WITH REGARD TO THE SECOND EDITION

I awoke one Saturday morning in July 1992 with the idea that it would be “sort of cool” to stroll into a bookshop and see something I’d written on the shelves. So with no clue as to what this would actually entail, I started penning the first edition of *Bebop to the Boolean Boogie*, which eventually hit the streets in 1995.

Much to my surprise, *Bebop* quickly found a following at Yale University as part of an introductory electronics course (it was subsequently adopted by a number of other universities around the world), and it soon became required reading for sales and marketing groups at a number of high-tech companies in Silicon Valley and across the USA.

Time passed (as is its wont), and suddenly it was seven years later and we were in a new millennium! Over these last few years, electronics and computing technology has progressed in leaps and bounds. In 1995, for example, an integrated circuit containing around 14 million transistors was considered to be relatively state-of-the-art. By the summer of 2002, however, Intel had announced a test chip containing 330 million transistors!

And it’s not just improvements to existing technologies, because over the last few years entirely new materials like carbon nanotubes have made their appearance on the scene. Therefore, by popular demand, I’ve completely revamped *Bebop* from cover to cover, revising the nitty-gritty details to reflect the latest in technology, and adding a myriad of new facts, topics, and nuggets of trivia. Enjoy!

xxi

NOTE FROM THE AUTHOR WITH REGARD TO THE THIRD EDITION

Give me strength! Where does the time go? It’s now 2008 as I write this note—16 years after I started work on the first edition of this tome. As you may have noticed, things are racing along in technology space (where no one can hear you scream). Looking back at my notes to the second edition, I see mention of Intel introducing a chip containing “330 million transistors!” Observe the exclamation mark. As you can see, I was quite excited

about this. Well, I happen to know that, in just a few days as I pen these words—on May 19, 2008—Altera will be introducing a new family of FPGAs (see *Chapter 16: Programmable ICs*) at the 40-nm technology node, and the largest of these devices will comprise 2.5 billion transistors!!! (I know, multiple exclamation marks are the sign of a deranged mind, but I've reached the age where I no longer care.)

Similarly, in 2003 I was thrilled by the possibilities for nanotubes. These are still jolly exciting, but the latest “buzz on the street” pertains to a newly isolated form of carbon called *graphene*, which may offer a myriad of properties that are of interest to the designers of integrated circuits.

On the other side of the coin, some technologies that were once in vogue have simply faded away with barely a whimper, while others that initially seemed to be “red hot” never actually came to fruition and disappeared without trace. But we need not become despondent, because new ideas and technologies are popping up all the time.

I tell you, things just keep on getting better and better and more and more interesting, and I for one can't wait to see what's hiding just around the corner. So, until next time (and I have no doubt that there will be a *next time*), I hope all goes well with you and yours and that you enjoy reading this book as much as I've enjoyed writing it.

This outrageously interesting book has two namesakes: Bebop, a jazz style known for its fast tempos and agitated rhythms, and Boolean Algebra, a branch of mathematics that is the mainstay of the electronics designer's tool chest. *Bebop to the Boolean Boogie* meets the expectations set by both, because it leaps from topic to topic with the agility of a mountain goat, and it will become your key reference guide to understanding the weird and wonderful world of electronics.

Bebop to the Boolean Boogie provides a wide-ranging but comprehensive introduction to the electronics arena, roaming through the fundamental concepts, and rampaging through electronic components and the processes used to create them. As a bonus, nuggets of trivia are included with which you can amaze your family and friends; for example, Greenland Eskimos have a base-20 number system, because they count using both fingers and toes.

Section 1: Fundamentals starts by considering the differences between analog and digital views of the world. We then proceed rapidly through atomic theory and semiconductor switches to primitive logic functions and their electronic implementations. The concepts of alternative numbering systems are presented, along with binary arithmetic, Boolean algebra, and Karnaugh map representations.

Finally, the construction of more complex logical functions is considered, along with their applications.

Section 2: Components and Processes is where we consider the components from which electronic systems are formed and the processes required to manufacturer them. The construction of integrated circuits ("silicon chips") is examined in some detail, followed by introductions to memory devices, programmeable devices, and application-specific devices. The discussion continues with hybrids, printed circuit boards, and a variety of advanced packaging techniques. We close with an overview of some alternative and future technologies.

Section 3: Design Tools and Stuff, which is new to the third edition, is where we discuss how electronics engineers actually go about designing and implementing components, circuit boards, and electronic systems, including the tools they use and the way in which these tools play together.

This book is of particular interest to electronics students. Additionally, by clarifying the techno-speech used by engineers, the book is of value to anyone who is interested in understanding more about electronics but lacks a strong technical background.

Except where such interpretation is inconsistent with the context, the singular shall be deemed to include the plural, the masculine shall be deemed to include the feminine, and the spelling (and the punctuation) shall be deemed to be correct!

This page intentionally left blank

Acknowledgments

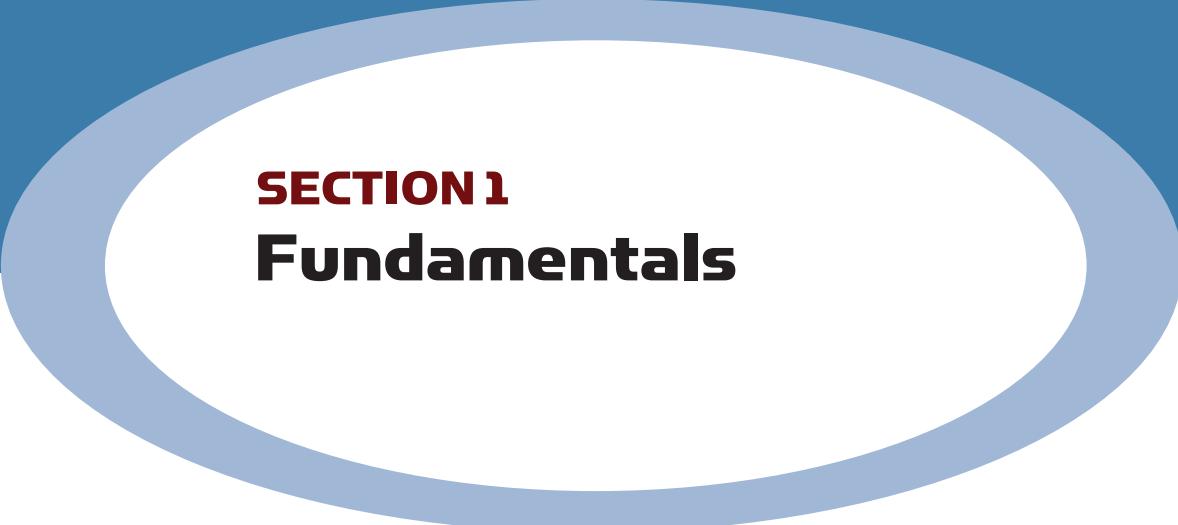
Special thanks for technical advice go to Alvin Brown, Alon Kfir, Don Kuk, and Preston Jett, the closest thing to living encyclopedic reference manuals one could hope to meet. (The reason that the text contains so few bibliographic references is due to the fact that I never had to look anything up—I simply asked the relevant expert for the definitive answer.)

I would also like to thank Dave Thompson from Mentor Graphics, Tamara Snowden and Robert Bielby from Xilinx, Stuart Hamilton from NEC, Richard Gordon and Gary Smith from Gartner Dataquest (Gary is now with Gary Smith EDA at <http://www.garysmitheda.com/>), Richard Goering from *EE Times* (Richard is now with SCD Source at <http://www.scdsource.com/>), high-speed design expert Lee Ritchey from Speeding Edge, and circuit board technologist Happy Holden from Westwood Associates, all of whom helped out with critical nuggets of information just when I needed them the most.

Actually, when I come to think about it, over the years I've been fortunate enough to come into contact with an incredible number of people spanning all areas of electronics and computing, and I've learned something new and interesting from every one of them, so my thanks also go out to all of these folks (you know who you are).

—Clive "Max" Maxfield, May 2008

This page intentionally left blank



SECTION 1

Fundamentals

This page intentionally left blank

CHAPTER 1

Analog Versus Digital

IT WAS A DARK AND STORMY NIGHT...

Ah, the classic opening: “*It was a dark and stormy night...*” which was made famous by the Peanuts cartoon character, Snoopy (see Box). I always wanted to start a book this way myself, and this is as good a time as any...

The phrase “*It was a dark and stormy night...*” is actually the opening sentence of an 1830 book by the British author Edward George Earl Bulwer-Lytton. A legend in his own lunchtime, Bulwer-Lytton became renowned for penning exceptionally bad prose, of which the opening to his book *Paul Clifford* set the standard for others to follow.

3

For your delectation and delight, the complete opening sentence of Bulwer-Lytton’s masterpiece was: “*It was a dark and stormy night; the rain fell in torrents—except at occasional intervals, when it was checked by a violent gust of wind which swept up the streets (for it is in London that our scene lies), rattling along the housetops, and fiercely agitating the scanty flame of the lamps that struggled against the darkness.*”

Actually, Bulwer-Lytton (1803–1873) was a very popular writer in his day, coining such phrases as “*the great unwashed*,” “*pursuit of the almighty dollar*,” and “*the pen is mightier than the sword*.” However, he may well have fallen into obscurity along with so many of his contemporaries if it were not for the annual Bulwer-Lytton Fiction Contest sponsored by the English Department of San Jose State University.

“*It was a dark and stormy night ...*” is now generally understood to represent an extravagantly florid style with redundancies and run-on sentences, and the Bulwer-Lytton Fiction Contest was formed to “celebrate” the worst extremes of this general style of writing. Over the years, the contest has gained international attention and now attracts 10,000 or more entries a year. In fact, I myself have submitted an entry for the 2008 competition, but all I’ve heard so far is an e-mail message saying that I can be assured that my offering “*will be given the consideration it deserves*.” On the off-chance that you’re interested, I’ve included a copy of my humble submission at the end of this chapter.

ANALOG VERSUS DIGITAL VIEWS OF THE WORLD

Now sit up and pay attention because this bit is important. Electronic engineers split their world into two views, called *analog*¹ and *digital*, and it's necessary to understand the difference between these views to make much sense out of the rest of this book.

At this point, even though we've barely dipped our toes in the water, I can imagine you rolling your eyes saying to yourself: "*Good grief, I already know all of this stuff!*" Well, I'm hoping that by the time we reach the end of this first chapter you'll be thinking: "*Hmmm, maybe Max is not as daft as he looks (but, there again, who could be?) I actually learned something here. I can't wait to read the next chapter. And just as soon as I get a spare moment I will rush out and buy some of Max's other books!*"

In the context of electronics, an analog device or system is one that uses continuously variable signals to represent information for input, processing, output, and so forth. A very simple example of an analog system would be a light controlled by a dimmer switch.

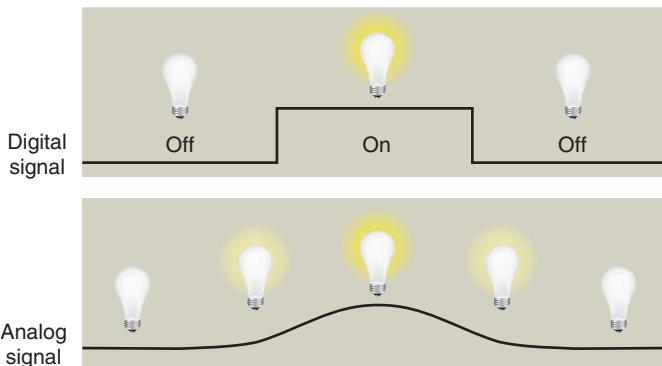
By comparison, a digital device or system is one that uses discrete (that is, discontinuous) values to represent information for input, processing, storage, output, and so forth. A digital quantity is one that can be represented as being in one of a finite number of states, such as 0 and 1, ON and OFF, UP and DOWN, etc. As an example of a simple digital system, consider a light switch in a house. When the switch is UP, the light is ON, and when the switch is DOWN, the light is OFF.²

We can illustrate the differences in the way these two systems work by means of a graph-like diagram (Figure 1.1). Time is considered to progress from left to right, and the solid lines—which engineers often refer to as *waveforms*—indicate what is happening.

In this illustration, the digital waveform commences in its OFF state. After some time it changes to its ON state, and sometime later it returns to its OFF state. By comparison, in the case of the analog waveform, we typically don't think in terms of ON and OFF. Rather, we tend to regard things as being more ON or more OFF with an infinite number of values between these two extremes.

¹In England, *analog* is spelled *analogue*, and it's pronounced with a really cool accent.

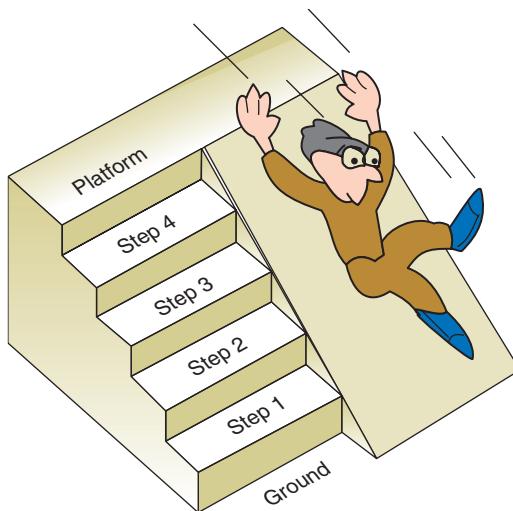
²At least, that's the way they work in America. It's the opposite way round in England, and you take your chances in the rest of the world.

**FIGURE 1.1**

Digital versus analog waveforms.

MULTI-VALUE DIGITAL SYSTEMS

One interesting point about digital systems is that they can have more than two states. These states are called *quanta* (from the Latin *quantus*, meaning “how much” or “how great”), and the accuracy or resolution of a digital value is dependent on the number of quanta employed to represent it. For example, consider a fun-loving fool sliding down a ramp mounted alongside a staircase (Figure 1.2).

**FIGURE 1.2**

Staircase and ramp.

In order to accurately determine this person’s position on the ramp, an independent observer would require the use of a tape measure. (Of course, everyone

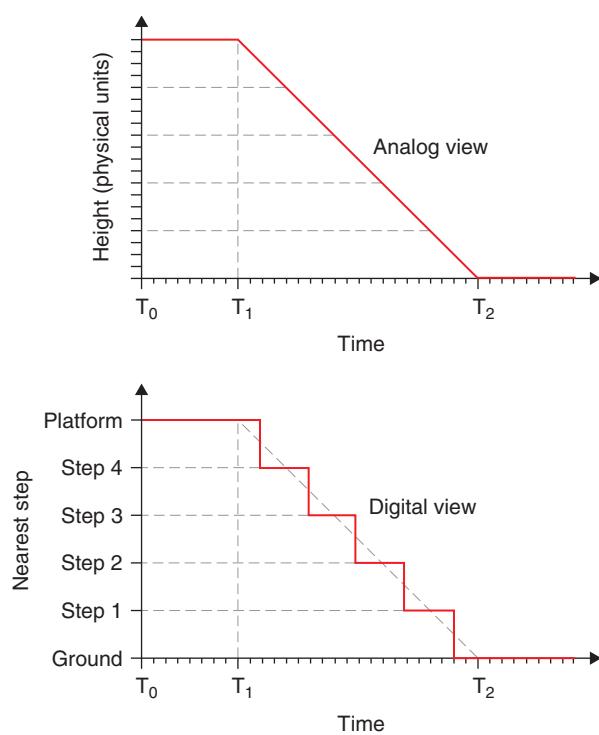
involved would have to be in agreement as to the specific end points being measured, such as the tip of the ramp-slider's nose or his belly button, for example.) The exact position on the ramp, as measured using the tape measure, would be considered to be an analog value. In this case, the analog value most closely represents the real world and can be as precise as the measuring technique allows.

Alternatively, an observer could estimate the ramp-slider's approximate location in relation to the nearest stair. Such an estimation would be considered to be a digital value.

Assume that at some starting time we'll call T_0 (time zero), our thrill-seeker is balanced at the top of the ramp preparing to take the plunge. He commences sliding at time T_1 and reaches the bottom of the ramp at time T_2 . If the ramp-slider were to be holding a pen against the wall during his descent, we would see a graphical depiction of his descent on the wall. (My mother used to hate it when I performed experiments of this nature. "Clive," she would say, because that's my real name, "*you've got to stop doing this sort of thing, you're 30 years old for goodness sake!*") Similarly, analog and digital waveforms can be plotted representing the location of the person on the ramp as a function of time (Figure 1.3).

FIGURE 1.3

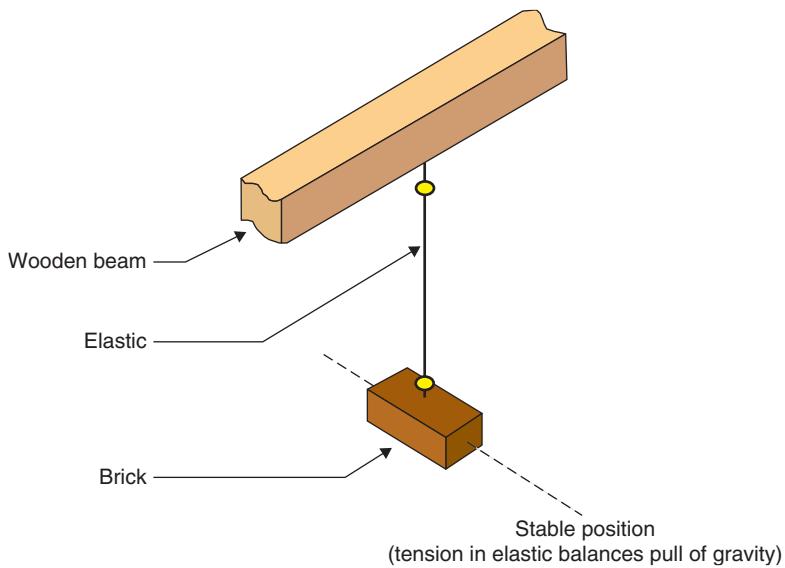
Waveforms showing the ramp-slider's position.



Once again, the horizontal axis in both waveforms represents the passage of time, which is considered to progress from left to right. In the case of the analog waveform, the vertical axis is used to represent our thrill-seeker's exact location in terms of height above the ground, and is therefore annotated with real, physical units. By comparison, the vertical axis for the digital waveform is annotated with abstract labels, which do not have any units associated with them.

EXPERIMENTS WITH BRICKS

To examine the differences between analog and digital views in more detail, let's consider a brick suspended from a wooden beam by a piece of elastic. If the brick is left to its own devices, it will eventually reach a stable state in which the pull of gravity is balanced by the tension in the elastic (Figure 1.4).

**FIGURE 1.4**

Brick suspended by elastic.

Let's assume that at time T_0 the system is in its stable state. The system remains in this state until time T_1 , when an inquisitive passerby grabs hold of the brick and pulls it down, thereby increasing the tension on the elastic. It takes some amount of time to pull the brick down, and it reaches its lowest point at time T_2 .

The passerby hangs around for a while looking somewhat foolish, releases the brick at time T_3 , and thereafter exits from our story, never to cross our paths again.³ The brick's resulting motion may be illustrated using an analog waveform (Figure 1.5).

Now consider a digital view of the brick's motion represented by two quanta labeled LOW and HIGH. The LOW quanta may be taken to represent any height *less than or equal to* the system's stable position, and the HIGH quanta therefore represents any height *greater than* the stable position (Figure 1.6). (Our original analog waveform is shown as a dotted line.)

Although it is apparent that the digital view is a subset of the analog view, digital representations often provide extremely useful approximations of the real world. If the only requirement in the above example is to determine whether the brick is above or below the stable position, for example, then the digital view is the most appropriate.

³Alright, alright—just after releasing the brick, our passerby walked around a corner and bumped into a beautiful girl. They fell in love, got married, bought a small cottage in the country, had three children (two girls and a boy), and lived happily ever after.

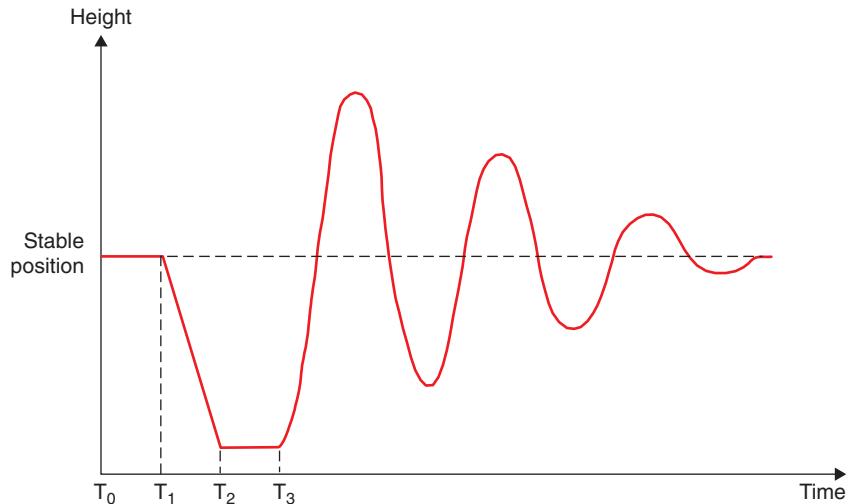


FIGURE 1.5
Brick on elastic: Analog waveform.

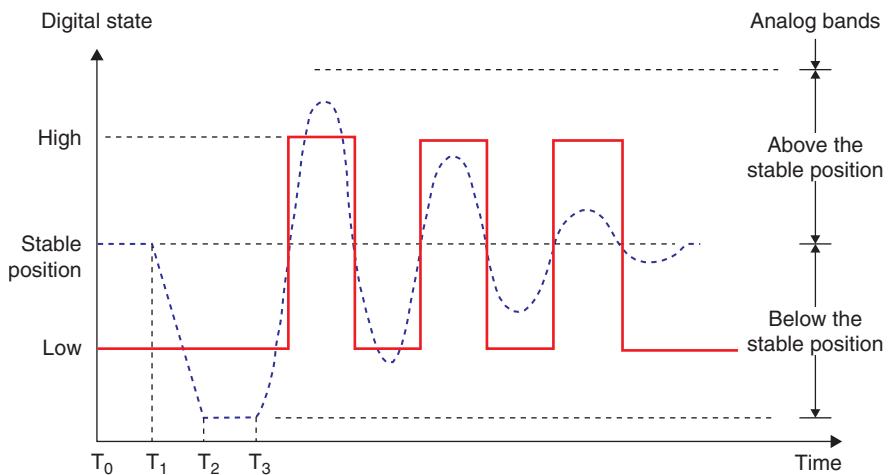
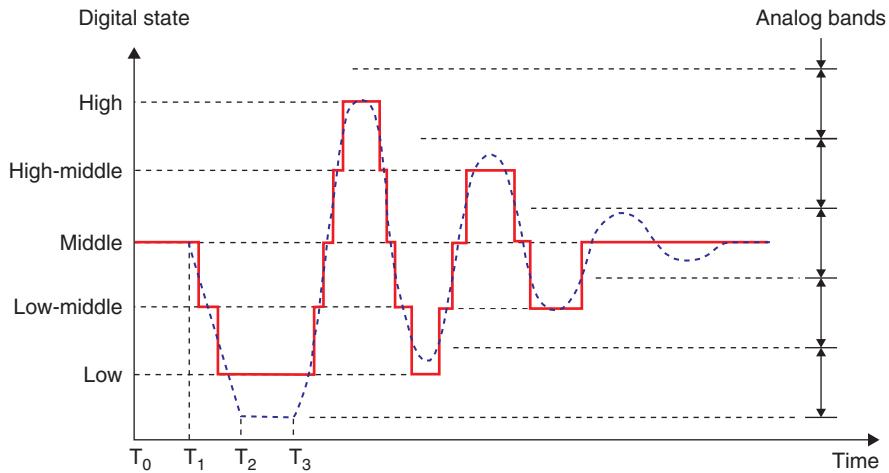


FIGURE 1.6
Brick on elastic: Two-quanta digital system.

The accuracy or resolution of a digital view can be improved by adding more quanta. For example, consider a digital view with five quanta: LOW, LOW-MIDDLE, MIDDLE, HIGH-MIDDLE, and HIGH. As the number of quanta increases, the digital view more closely approximates its analog counterpart (Figure 1.7).⁴

⁴We consider this concept of digital resolution in a little more detail in *Chapter 13: Analog-to-Digital and Vice Versa*.

**FIGURE 1.7**

Brick on elastic: Five-quanta digital system.

In the real world, every electronic component behaves in an analog fashion. However, these components can be connected together so as to form functions whose behavior is amenable to digital approximations. This book predominantly concentrates on the digital view of electronics, although certain aspects of analog designs and the effects associated with them are discussed where appropriate.

MAX'S ENTRY TO THE 2008 BULWER-LYTTON FICTION CONTEST

Before you peruse my humble offering, let's remind ourselves that the point of this contest is to write a single sentence representing the worst possible (although grammatically correct) opening sentence for a hypothetical fictional book. Now read on ...

As the hours passed, the expressions on the partygoers' faces became increasingly bemused and bewildered as my mother—having grabbed the conversational reins with gusto and abandon using one of her classic opening gambits of “I bumped into Mrs. Forteske-Smythe at the fishmonger’s the other day ...”—proceeded to inundate the gathered throng with a myriad of seemingly innocuous and unrelated details: “... you remember, she was the oldest of three sisters; the youngest, Beryl, was a slut, while the middle girl eloped with a transsexual Australian taxidermist and they had two sons who couldn’t bring themselves to touch any form of fruit, and ...” and I could see the question forming in everyone’s minds: “Can she possibly tie all of these tidbits of trivia together and somehow bring this tortuous tale to a meaningful close?”... and I cowered against the wall wearing a tight, grim smile because I knew, to my cost, that she could.

This page intentionally left blank

CHAPTER 2

Atoms, Molecules, and Crystals

PROTONS, NEUTRONS, AND ELECTRONS

Matter, the stuff that everything is made of, is formed from *atoms*. The heart of an atom, the *nucleus*, is composed of *protons* and *neutrons* and is surrounded by a *cloud* of *electrons*.¹ For example, consider an atom of the gas helium, which consists of two protons, two neutrons, and two electrons (Figure 2.1).

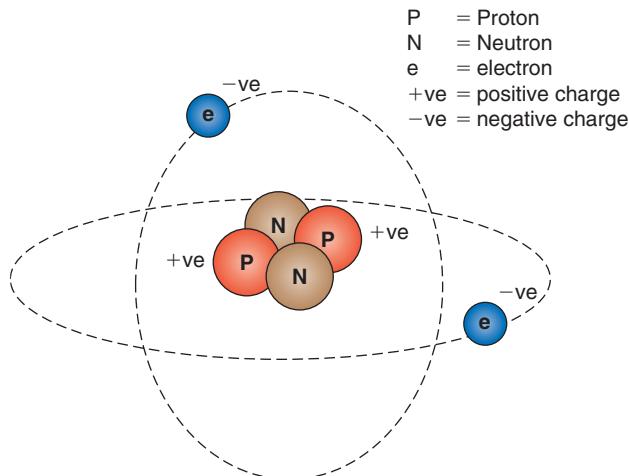


FIGURE 2.1

Helium atom.

¹We now know that protons, neutrons, and electrons are formed from fundamental particles called *quarks*, of which there are six flavors: *up*, *down*, *charm*, *strange*, *top* (or *truth*), and *bottom* (or *beauty*). Quarks are so weird that they have been referred to as "*The dreams that stuff is made from*," and they are way beyond the scope of this book.

It may help to visualize the electrons as orbiting the nucleus in the same way that the moon orbits the earth. Things aren't this simple in the real world (the location of each electron at any particular point in time is best described by a probability function), but the concept of orbiting electrons serves our purpose here.

Each proton carries a single positive (+ve) charge, and each electron carries a single negative (-ve) charge. The neutrons are neutral and act like glue, holding the nucleus together and resisting the natural tendency of the protons to repel each other. Protons and neutrons are approximately the same size, while electrons are very much smaller. If a basketball were used to represent the nucleus of a helium atom, then on the same scale, softballs could represent the individual protons and neutrons, while large garden peas could represent the electrons. In this case, the diameter of an electron's orbit would be approximately equal to the length of 250 American football fields (excluding the end zones)! Thus, the majority of an atom consists of empty space. If all the empty space were removed from the atoms that form a camel, it *would* be possible for the little rascal to pass through the eye of a needle!^{2,3,4}

The number of protons determines the type of the element; for example, hydrogen has one proton, helium two, lithium three, etc. Atoms vary greatly in size, from hydrogen with its single proton to those containing hundreds of protons. The number of neutrons does not necessarily equal the number of protons. There may be several different flavors, or *isotopes*, of the same element differing only in their number of neutrons. For example, hydrogen has three isotopes with zero, one, and two neutrons; these isotopes are called *hydrogen*, *deuterium*, and *tritium*, respectively.

Left to its own devices, each proton in the nucleus will have a complementary electron. If additional electrons are forcibly added to an atom, the result is a *negative ion* of that atom; if electrons are forcibly removed from an atom, the result is a *positive ion*.

²I am, of course, referring to the Bible verse: "It is easier for a camel to go through the eye of a needle than for a rich man to enter the kingdom of God." (Mark 10:25; New International Version).

³In fact, the "needle" was a small, man-sized gate located next to the main entrance to Jerusalem. Camels, wagons, and so forth were obliged to pass through the main gate, while people could choose to go through the smaller opening if the main gate was closed.

⁴Be warned! The author has discovered to his cost that if you call a zoo to ask the cubic volume of the average adult camel, they treat you as if you are a complete idiot.

QUANTUM LEVELS AND ELECTRON SHELLS

In an atom where each proton is balanced by a complementary electron, one would assume that the atom would be stable and content with its own company, but things are not always as they seem. Although every electron contains the same amount of negative charge, electrons orbit the nucleus at different levels, known as *quantum levels* or *electron shells* (the term *quantum* comes from the Latin *quantus*, meaning “how much” or “how great”).

Each electron shell requires a specific number of electrons to fill it; the first shell requires two electrons, the second requires eight, etc. Thus, although a hydrogen atom contains both a proton and an electron and is therefore electrically balanced, it is still not completely happy with its lot. Given a choice, hydrogen would prefer to have a second electron to fill its first electron shell. However, simply adding a second electron is not the solution; although the first electron shell would now be filled, the extra electron would result in an electrically unbalanced negative ion.

MAKING MOLECULES

The previous point could present a bit of a poser, but the maker of the universe came up with a solution: atoms can use the electrons in their outermost shell to form bonds with other atoms. The atoms share each other’s electrons, thereby forming more complex structures. One such structure is called a *molecule*; for example, two hydrogen atoms (chemical symbol: H), each comprising a single proton and electron, can bond together and share their electrons to form a hydrogen molecule (chemical symbol: H₂; Figure 2.2).

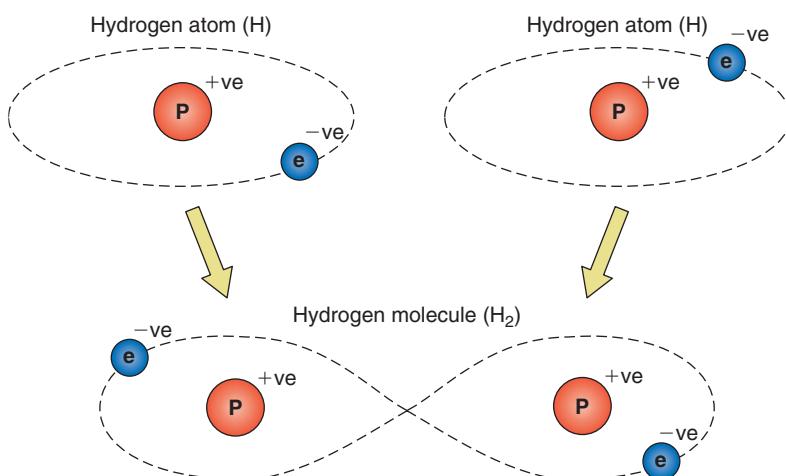
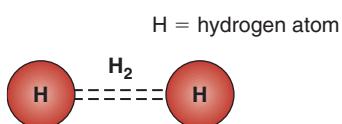


FIGURE 2.2

Two hydrogen atoms bonding to form a hydrogen molecule.

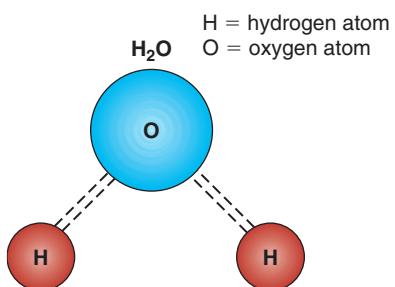
The electrons contained in the outermost (*valence*) electron shell of an atom are known as *valence electrons* (from the Latin *valentia*, meaning “strength,” or “worth”), so these types of bonds are called *valence bonds*. The resulting hydrogen molecule contains two protons and two electrons from its constituent atoms and so remains electrically balanced. However, each atom lends its electron to its partner and, at the same time, borrows an electron from its partner. This can be compared to two circus jugglers passing objects between each other—the quickness of the hand deceives the eye. The electrons are passing backwards and forwards between the atoms so quickly that each atom is fooled into believing it has two electrons. The first electron shell of each atom appears to be completely filled and the hydrogen molecule is therefore stable.

**FIGURE 2.3**

Alternative representation of a hydrogen molecule.

Even though the hydrogen molecule is the simplest molecule of all, the previous illustration demanded a significant amount of time, space, and effort. Molecules formed from atoms containing more than a few protons and electrons would be well-nigh impossible to represent in this manner. A simpler form of representation is therefore employed, with two dashed lines indicating the sharing of two electrons (Figure 2.3).

Now, let's contrast the case of hydrogen with that of helium. Helium atoms each have two protons and two electrons and are therefore electrically balanced. Additionally, as helium's two electrons completely fill its first electron shell, this atom is very stable.⁵ This means that under normal circumstances, helium atoms do not go around casually making molecules with every other atom they meet.⁶

**FIGURE 2.4**
Water molecule.

Molecules can also be formed by combining different types of atoms. An oxygen atom (chemical symbol: O) contains eight protons and eight electrons. Two of the electrons are used to fill the first electron shell, which leaves six left over for the second shell. Unfortunately for oxygen, its second shell would ideally prefer eight electrons to fill it. Each oxygen atom can therefore form two bonds with other atoms—for example, with two hydrogen atoms to form a water molecule (chemical symbol: H_2O ; Figure 2.4). (The reason the three atoms in the water

⁵Because helium is so stable, it is known as an *inert*, or *noble*, gas, where the latter appellation presumably comes from the fact that helium doesn't mingle with the commoners (grin).

⁶Noble gases rarely react with other elements since they are already stable. Under normal conditions, they occur as odorless, colorless, monatomic gases. The six known noble gases are: helium (He), neon (Ne), argon (Ar), krypton (Kr), xenon (Xe), and radon (Rn).

molecule are not shown as forming a straight line is discussed in the section on nanotechnology in *Chapter 21: Alternative and Future Technologies*).

Each hydrogen atom lends its electron to the oxygen atom and—at the same time—borrows an electron from the oxygen atom. This leads both of the hydrogen atoms to believe they have two electrons in their first electron shell. Similarly, the oxygen atom lends two electrons (one to each hydrogen atom) and borrows two electrons (one from each hydrogen atom).

When the two borrowed electrons are added to the original six in the oxygen atom's second shell, this shell appears to contain the eight electrons necessary to fill it. Thus, all the atoms in the water molecule are satisfied and the molecule is stable.

CRYSTALS AND OTHER STRUCTURES

Structures other than molecules may be formed when atoms bond; for example, *crystals*. Carbon, silicon, and germanium all belong to the same family of elements. Each has only four electrons in its outermost electron shell. Silicon has 14 protons and 14 electrons; two electrons are required to fill the first electron shell and eight to fill the second shell. Thus, only four remain for the third shell, which would ideally prefer eight. Under the appropriate conditions, each silicon atom will form bonds with four other silicon atoms, resulting in a three-dimensional silicon crystal⁷ (Figure 2.5).

The electrons used to form the bonds in crystalline structures such as silicon are tightly bound to their respective atoms. Yet another structure is presented by metals, such as copper, silver, and gold. Metals have an *amorphous crystalline structure* in which their shared electrons have relatively weak bonds and may easily migrate from one atom to another.

Apart from the fact that atoms are the basis of life, the universe, and everything as we know it, they are also fundamental to the operation of the components used in electronic designs. Electricity may be considered to be vast herds of electrons migrating from one place to another, while electronics is the art and science of controlling these herds: starting them, stopping them, deciding where they can roam, and determining the activities the little scamps are going to perform while on their way.

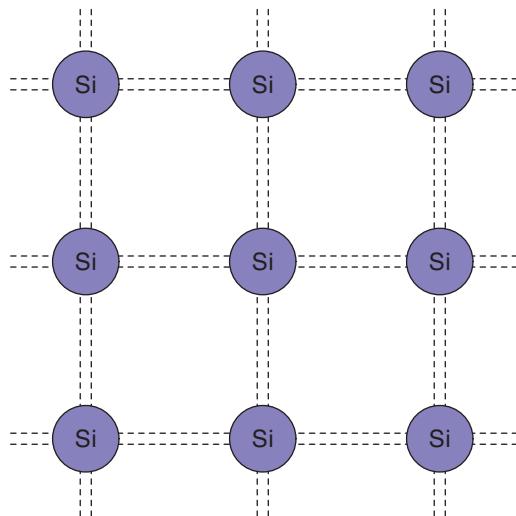


FIGURE 2.5

Simplified (two-dimensional) representation of the three-dimensional structure of crystalline silicon.

⁷An equivalent structure formed from carbon atoms is known as *diamond*.

This page intentionally left blank

CHAPTER 3

Conductors, Insulators, and Other Stuff

CONDUCTORS AND INSULATORS

A substance that conducts electricity easily is called a *conductor*. Metals such as copper are very good conductors because the bonds in their amorphous crystalline structures are relatively weak, which means that the bonding electrons can easily migrate from one atom to another. If a piece of copper wire is used to connect a source with an excess of electrons to a target with too few electrons, the wire will conduct electrons between them (Figure 3.1).

If we consider electricity to be the migration of electrons from one place to another, then we may also say that it flows from the more negative source to the more positive target. As an electron jumps from the negative source into the wire, it “pushes” (repels) the nearest electron in the wire out of the way. This electron in turn pushes another, and the effect ripples down the wire until an electron at the far end of the wire is ejected into the more positive target. When an electron arrives in the positive target, it neutralizes one of the positive charges.

An individual electron can take a surprisingly long time to migrate from one end of the wire to the other. However, the time between an electron entering

17

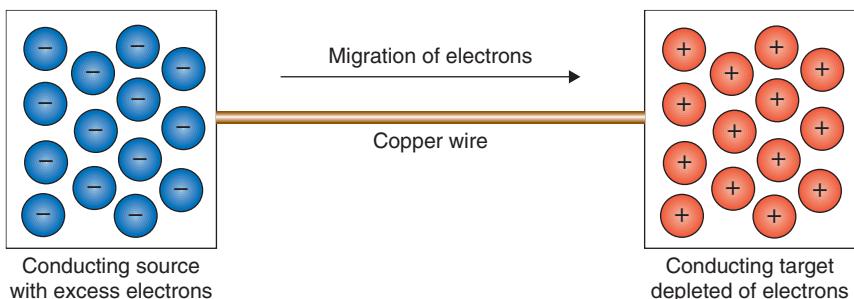


FIGURE 3.1
Electrons flowing through a copper wire.

one end of the wire and causing an equivalent electron to be ejected from the other end is extremely fast.¹

As opposed to a conductor, a substance that does not conduct electricity easily is called an *insulator*. Materials such as rubber are very good insulators because the electrons used to form bonds are tightly bound to their respective atoms.²

VOLTAGE, CURRENT, AND RESISTANCE

One measure of whether a substance is a conductor or an insulator is how much it resists the flow of electricity. In order to visualize this, let's take a step back and imagine a tank of water to which two pipes are connected at different heights; the water ejected from the pipes is caught in two buckets, A and B (Figure 3.2).

Assume that the contents of the tank are magically maintained at a constant level. The water pressure at the end of a pipe inside the tank depends on the depth of the pipe with respect to the surface level of the water. The difference in pressure between the ends of a pipe inside and outside the tank causes water

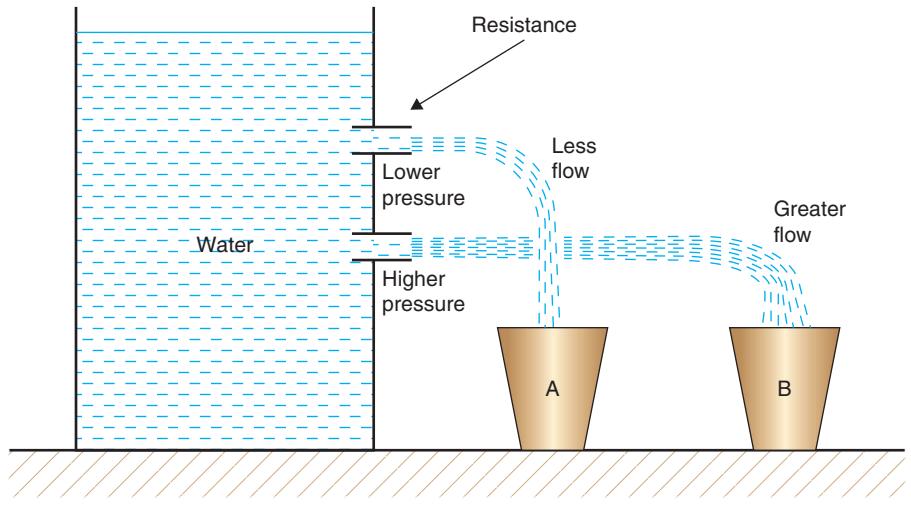


FIGURE 3.2

Water tank representation of voltage, current, and resistance.

¹For a copper wire isolated in a vacuum, the speed of a signal propagating through the wire is only fractionally less than the speed of light. However, the speed of a signal is modified by a variety of factors, including any materials surrounding or near the conductor. Signal speeds in electronic circuits vary, but are typically in the vicinity of half the speed of light.

²In reality, everything conducts if presented with a sufficiently powerful electric potential. For example, if you don a pair of rubber boots and then fly a kite in a thunderstorm, your rubber boots won't save you when the lightning comes racing down the kite string! (Bearing in mind that litigation is a national sport in America, do NOT try this at home unless you are a professional.)

to flow. The amount of water flowing through a pipe depends on the water pressure and on the resistance to that pressure determined by the pipe's cross-sectional area. A thin pipe with a smaller cross-sectional area will present more resistance to the water than will a thicker pipe with a larger cross-sectional area. Thus, if both pipes have the same cross-sectional area, bucket B will fill faster than bucket A.

In electronic systems, the flow of electricity is called *current*, measured in units of *amperes* or *amps*;^{3,4} the resistance to electrical flow is simply called *resistance*, measured in units of *ohms*;⁵ and the electrical equivalent to pressure is called *voltage*, or electric potential, measured in units of *volts*.⁶

One other concept we should perhaps mention here is that of *power*. Consider the effect of a barrel of water being slowly dripped on top of you. Apart from getting wet, you wouldn't really notice any ill effect. By comparison, imagine what it would feel like if a small cupful of water travelling at 1000 miles per hour were to hit you in the face; this would certainly attract your attention. In the context of our water tank representation shown in Figure 3.2, the "power" associated with the water is a function of both the quantity of flowing water and the speed at which it flows. Similarly, in the context of an electronic circuit, the amount of power, which is measured in units of *watts*,⁷ is determined by the equation: $P = V \times I$ (Power = Voltage multiplied by Current). But we digress ...

RESISTANCE AND RESISTORS

The materials used to connect components in electronic circuits are typically selected to have low resistance values. In some cases, however, engineers make

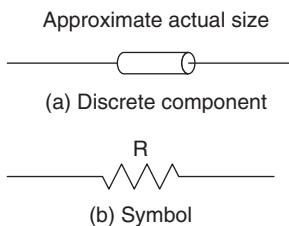
³The terms *amp* and *ampere* are named after the French mathematician and physicist André-Marie Ampère (1775–1836), who formulated one of the basic laws of electromagnetism in 1820.

⁴An amp corresponds to approximately 6,250,000,000,000,000 electrons per second flowing past a given point in an electrical circuit. (Not that the author counted them himself, you understand; this little nugget of information is courtesy of Microsoft's multimedia encyclopedia, *Encarta*.)

⁵The term *ohm* is named after the German physicist Georg Simon Ohm (1789–1854), who defined the relationship between voltage, current, and resistance in 1827. (We now call this *Ohm's Law*.)

⁶The term *volt* is named after the Italian physicist Count Alessandro Giuseppe Antonio Anastasio Volta (1745–1827), who invented the electric battery in 1800. (Having said this, some people believe that an ancient copper-lined jar found in an Egyptian pyramid was in fact a primitive battery ... but, there again, some people will believe anything. Who knows for sure?)

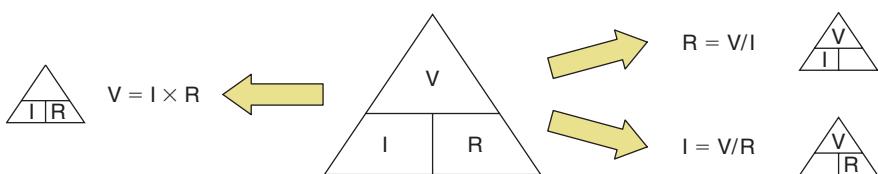
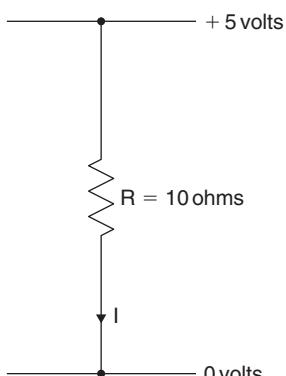
⁷The term *watt* is named after the Scottish inventor and engineer James Watt (1736–1819), whose improvements to the steam engine were fundamental to the changes brought by the Industrial Revolution.

**FIGURE 3.3**

Resistor: Component and symbol.

use of special resistive components called *resistors*. The value of resistance (R) depends on the resistor's length, cross-sectional area, and the resistivity of the material from which it is formed. Resistors come in many shapes and sizes; a common example could be as shown in Figure 3.3.⁸

In a steady-state system where everything is constant, the voltage, current, and resistance are related by a rule called *Ohm's Law*, which states that voltage (V) equals current (I) multiplied by resistance (R). An easy way to remember Ohm's Law is by means of a diagram known as *Ohm's Triangle* (Figure 3.4).

**FIGURE 3.4**
Ohm's Triangle.

$$\begin{aligned}\text{Ohm's law: } & V = I \times R \\ & I = V/R \\ & I = 5 \text{ volts}/10 \text{ ohms} \\ & I = 0.5 \text{ amps}\end{aligned}$$

FIGURE 3.5

Current flowing through a resistor.

Consider a simple electrical circuit comprising two wires with electrical potentials of 5 volts and 0 volts, connected by a resistor of 10 ohms (Figure 3.5).

With regard to Figure 3.5, instead of writing "5 volts," engineers would typically abbreviate this to "5 V." Similarly, instead of writing "0.5 amps," engineers would typically abbreviate this to "0.5 A." Also, the Greek letter omega (Ω) is used to represent resistance, so instead of writing "10 ohms," engineers would typically abbreviate this to "10 Ω ."

Observe that no space is used in the case of a single letter unit qualifier like 0.5 A (or 10 Ω), but a space is required when using a multi-letter unit qualifier such as 500 mA (where "m" stands for *milli*, meaning "thousandth").

⁸In addition to the simple two-terminal resistor illustrated in Figure 3.3, there are also variable resistors (sometimes called *potentiometers*), in which a third "center" connection is made via a conducting slider. Changing the position of the slider (perhaps by turning a knob) alters the relative resistance between the center connection and the two ends. There are also a variety of sensor resistors, including light-dependent resistors (LDRs) whose value depends on the amount of light falling on them, heat-dependent resistors called *thermistors*, and voltage-dependent resistors called VDRs or *varistors*. See also the *Memristor* topic toward the end of this chapter.

We should also note that engineers use the International System of Units (abbreviated to SI from the French “*Le Système International d’Unités*”), which is the modern form of the metric system. In the case of an SI unit that is derived from the proper name of a person, the first (often only) letter of its symbol must be in uppercase (V, A, W, Hz, etc.). By comparison, when this type of SI unit is spelled out in English, it should always begin with a lowercase letter (*volt*, *amp*, *watt*, *hertz*, etc.), except when used at the beginning of a sentence or in capitalized material such as a title.

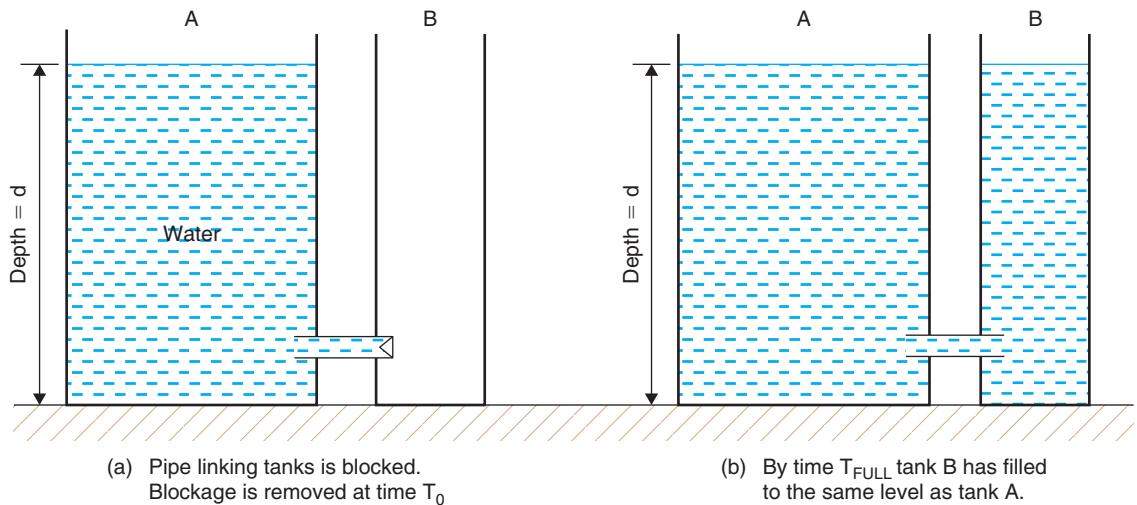
Observe that Figure 3.5 shows the direction of current flow as being from the more positive (+5 volts) to the more negative (0 volts). This may seem strange as, from our previous discussions, we know that current actually consists of electrons migrating from a negative source to a positive target.

The reason for this inconsistency is that the existence of electricity was discovered long before it was fully understood. Electricity as a phenomenon was known for quite some time, but it wasn’t until the early part of the 20th century that British physicist Sir Joseph John (J.J.) Thomson (1856–1940) proved the existence of the electron at the University of Aberdeen, Scotland. The men who established the original electrical theories had to make decisions about things they didn’t fully understand. The direction of current flow is one such example; for a variety of reasons (some of them being philosophical), it was originally believed that current flowed from positive to negative. As you may imagine, this inconsistency can, and does, cause endless problems.

CAPACITANCE AND CAPACITORS

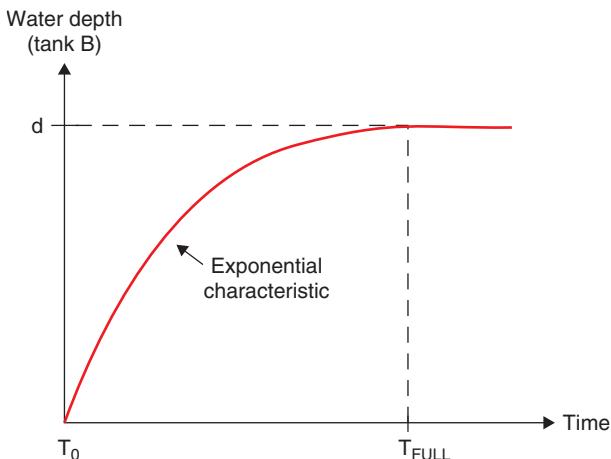
Now imagine a full water tank A connected by a blocked pipe to an empty water tank B [Figure 3.6(a)]. Assume that the contents of tank A are magically maintained at the same level regardless of the amount of water that is removed. At some time T_0 (“time zero”), the pipe is unblocked and tank B starts to fill. By the time we’ll call T_{FULL} , tank B will have reached the same level as tank A [Figure 3.6(b)].

The speed with which tank B fills depends on the rate at which water flows between the two tanks. In turn, the rate of water flow depends on the difference in pressure between the two ends of the pipe and any resistance to the flow caused by the pipe. When the water starts to flow between the tanks at time T_0 , there is a large pressure differential between the end of the pipe in tank A and the end in tank B, so the water will flow quickly. As tank B fills, however, the pressure differential between the tanks will be reduced correspondingly. This

**FIGURE 3.6**

Water tank representation of capacitance.

means that tank B fills faster at the beginning of the process than it does at the end. The rate at which tank B fills has an *exponential characteristic* that is best illustrated by a graph (Figure 3.7).

**FIGURE 3.7**

Graphical representation of the rate at which a water capacitor fills.

The electronic equivalent of water tank B stores electrical charge. This ability to store charge is called *capacitance*, measured in units of *farads*.⁹ Capacitance effects occur naturally in electronic circuits, and engineers generally try to ensure that their values are as low as possible. In some cases, however, designers may make use of special capacitive components called *capacitors*. One type of capacitor is formed from two metal plates separated by a layer of insulating material; the resulting capacitance (C) depends on the surface area of the plates,

the size of the gap between them, and the non-conducting (dielectric) material used to fill the gap. Capacitors come in many shapes and sizes; a common example could be as shown in Figure 3.8.

⁹The term *farad* is named after the British scientist Michael Faraday (1791–1867), who constructed the first electric motor in 1821.

Now consider a simple circuit comprising a resistor, a capacitor, and a switch. Initially the switch is in the OPEN (OFF) position, the capacitor voltage V_{CAP} is 0 volts, and no current is flowing [Figure 3.9(a)].

When the switch is CLOSED (turned ON), any difference in potential between V_{POS} (a positive voltage source) and V_{CAP} will cause current to flow through the resistor [Figure 3.9(b)]. As usual, the direction of current flow is illustrated in the classical (positive-to-negative) rather than the actual (negative-to-positive) sense. The current flowing through the resistor causes the capacitor to charge towards V_{POS} [Figure 3.10(a)]. But as the capacitor charges, the difference in voltage between V_{POS} and V_{CAP} decreases, and consequently so does the current [Figure 3.10(b)].

The maximum current I_{MAX} occurs at time T_0 when there is the greatest difference between V_{POS} and V_{CAP} . From Ohm's Law, we know that $I_{MAX} = V_{POS}/R$. The capacitor is considered to be completely charged by time T_{FULL} , at which point the flow of current falls to zero.

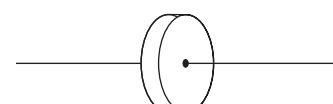
The time T_{RC} equals $R \times C$ and is known as the *RC time constant*. With R in ohms and C in farads, the resulting T_{RC} is in units of seconds. The RC time constant is approximately equal to the time taken for V_{CAP} to achieve around 63% of its final value and for the current to fall to around 37% of its initial value.¹⁰

Once a circuit has reached a steady-state condition where nothing is changing, any capacitors act like OPEN (OFF) switches; the effects of these components only come into play when signals are transitioning between different values.

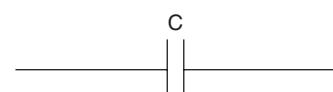
INDUCTANCE AND INDUCTORS

This is the tricky one. The author has yet to see a water-based analogy for inductance that didn't leak like a sieve (pun intended). One useful analogy is to consider two

Approximate actual size



(a) Discrete component



(b) Symbol

FIGURE 3.8

Capacitor: Component and symbol.

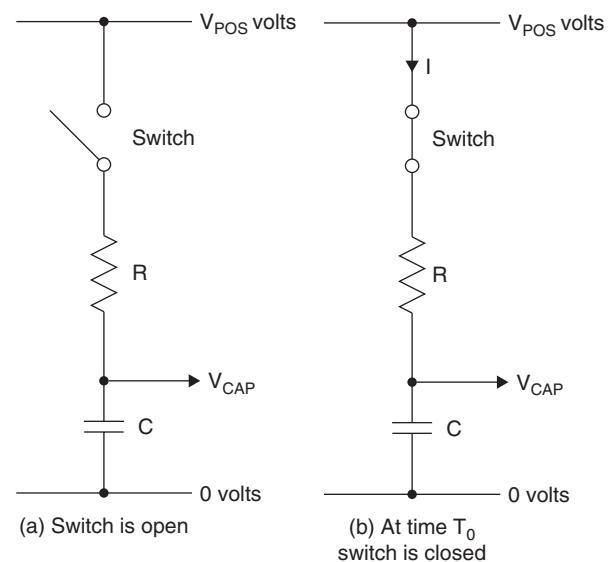
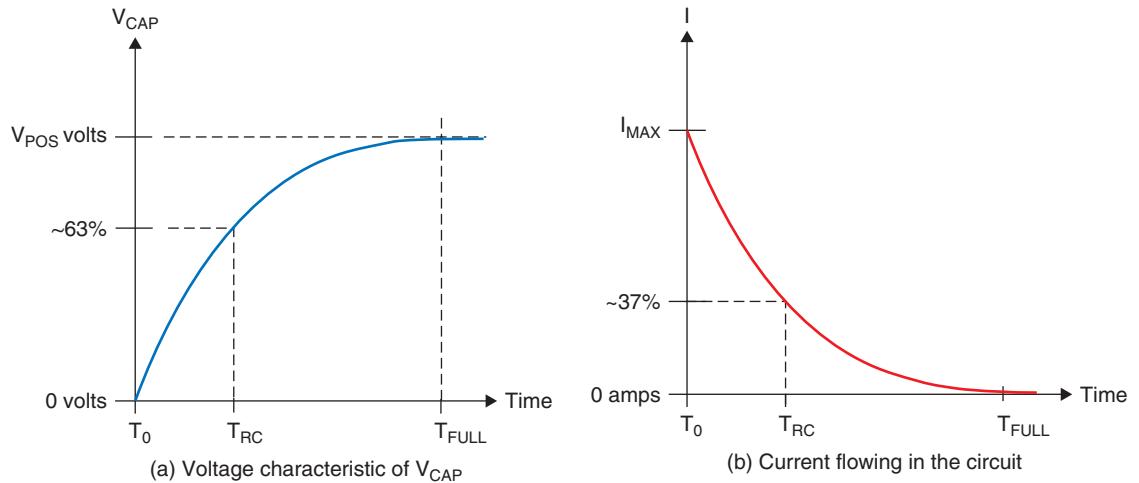


FIGURE 3.9

Resistor-capacitor-switch circuit.

¹⁰During each successive T_{RC} time constant, the capacitor will charge 63% of the *remaining* distance to the maximum voltage level. A capacitor is generally considered to be fully charged after five time constants.

**FIGURE 3.10**

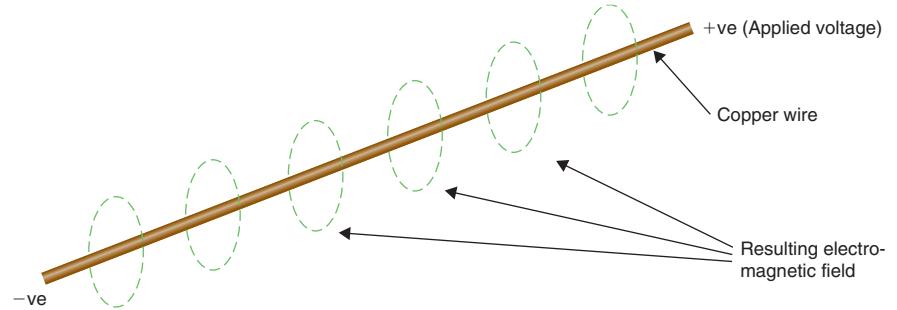
Voltage and current characteristics of the resistor-capacitor-switch circuit.

electric fans facing each other on a desk. If you turn one of the fans on, the other will start to spin in sympathy. In this case, we might say that the first fan *induces* an effect in the second. Well, electrical inductance is just like this, but different.

What, you want more? Oh well, how about this then: a difference in electrical potential between two ends of a conducting wire causes current to flow, and current flowing through a wire causes an electromagnetic field to be generated around that wire (Figure 3.11).

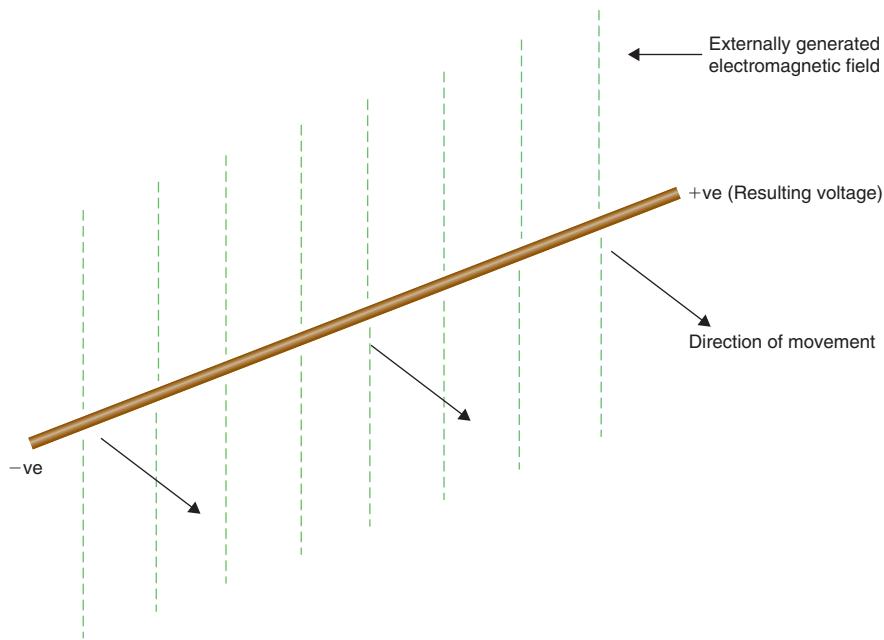
FIGURE 3.11

Current flowing through a wire generates an electromagnetic field.



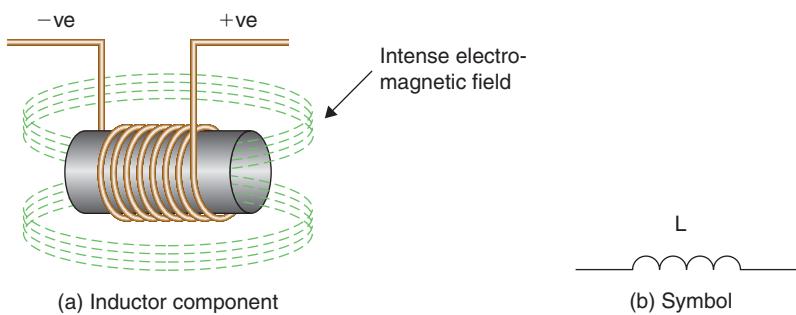
Correspondingly, if a piece of wire is moved through an externally generated electromagnetic field, it cuts the lines of electromagnetic flux, resulting in an electrical potential being generated between the two ends of the wire (Figure 3.12).

Engineers sometimes make use of components called *inductors*, which may be formed by winding a wire into a coil around a rod of iron or some other

**FIGURE 3.12**

A conductor cutting through an electromagnetic field generates an electrical potential.

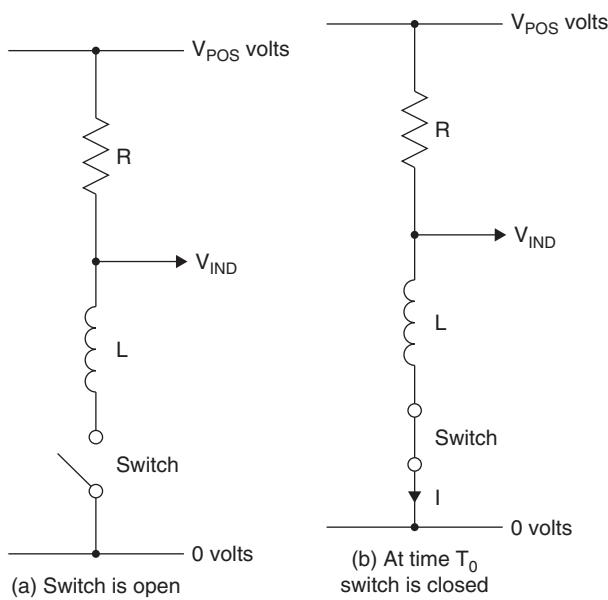
ferromagnetic material (the wire would be coated by a layer of insulating material to prevent coil windings forming electrical connections with each other or with the rod). When a current is passed through the coil, the result is an intense electromagnetic field (Figure 3.13).

**FIGURE 3.13**

Inductor: Component and symbol.

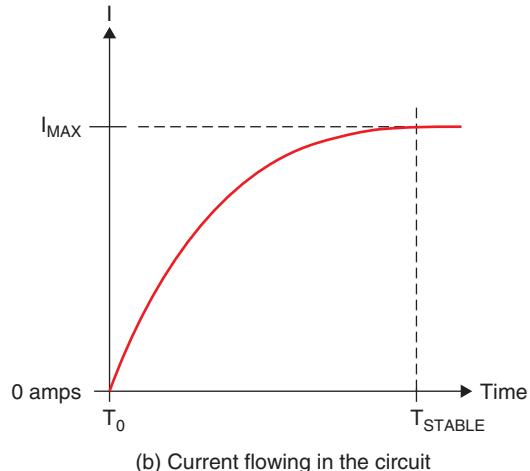
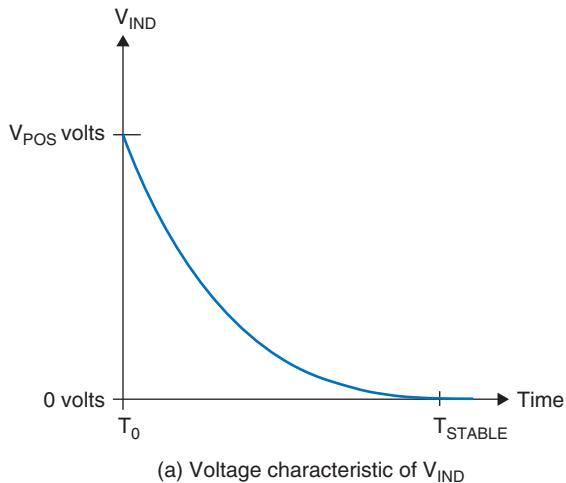
Now consider a simple circuit comprising a resistor, an inductor, and a switch. Initially the switch is in the OPEN (OFF) position, the inductor voltage V_{IND} is at V_{POS} volts, and no current is flowing [Figure 3.14(a)].

As the inductor is formed from a piece of conducting wire, one might expect that closing the switch at time T_0 [Figure 3.14(b)] would immediately

**FIGURE 3.14**

Resistor-inductor-switch circuit.

is fully established. The resulting voltage and current curves are illustrated in Figures 3.15(a) and 3.15(b), respectively.

**FIGURE 3.15**

Voltage and current characteristics of the resistor-inductor-switch circuit.

cause V_{IND} to drop to 0 volts, but this is not the case. When the switch is CLOSED (turned ON) and current begins to flow, the inductor's electromagnetic field starts to form. As the field grows in strength, the lines of flux are pushed out from the center, and in the process they cut through the loops of wire forming the coil. This has the same effect as moving a conductor through an electromagnetic field and a voltage differential is created between the ends of the coil. This generated voltage is such that it attempts to oppose the changes causing it. This effect is called *inductance*, the official unit of which is the *henry*.¹¹

As time progresses, the coil's inductance is overcome and its electromagnetic field

¹¹The term *henry* is named after the American inventor Joseph Henry (1797–1878), who discovered inductance in 1832.

Thus, by the time we'll call T_{STABLE} , the inductor appears little different from any other piece of wire in the circuit (except for its concentrated electromagnetic field). This will remain the case until some new event occurs to disturb the circuit—for example, opening the switch again.

Once a circuit has reached a steady-state condition where nothing is changing, any inductors act like simple pieces of wire; the effects of these components only come into play when signals are transitioning between different values.

As we noted above, inductors are typically formed by coiling insulated wire around a rod of iron or some other ferromagnetic material. When you strike a musical tuning fork, it rings with a certain frequency depending on its physical characteristics (size, material, etc.). Similarly, an inductor has a natural resonant frequency depending on the diameter of the rod, the material used to form the rod, the number of coils, etc. For this reason, inductors may form part of the tuning circuits used in radios.

Another common use of the inductive effect is to create components called *transformers*. Consider Figure 3.16 in which two ferromagnetic rods are connected by two ferromagnetic straps. Each rod has a coil of insulated wire wrapped round it. For the purposes of this example, let's assume that the input to the coil on the left (V_{IN}) is an oscillating signal in the form of a sine wave.

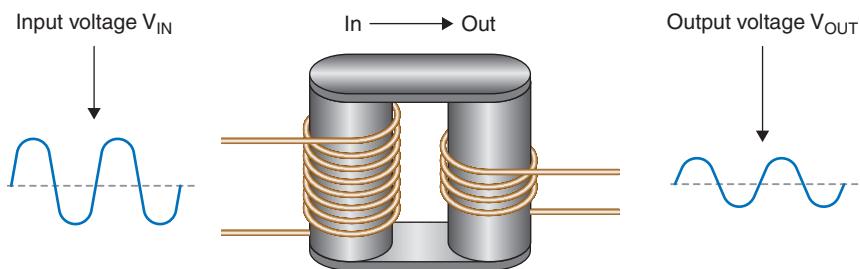


FIGURE 3.16
A simple step-down transformer.

Suppose the output coil on the right comprises half the number of turns of the input coil on the left. In this case, and assuming an ideal transformer with no losses, the output voltage (V_{OUT}) will be *half* the amplitude of the input voltage and the output current will be *twice* that of the input current.

This example would be referred to as a *step-down* transformer, because the output voltage is lower than the input voltage. By comparison, if the output coil on the right contained twice the number of turns of the input coil on the left, then this would be a *step-up* transformer, because the output voltage would be *twice* the amplitude of the input voltage and the output current would be *half* that of the input current.

MEMRISTANCE AND MEMRISTORS

The three fundamental *passive*¹² electronic components are: the capacitor, which stores energy in an electrical field; the resistor, which resists the flow of electricity; and the inductor, which resists any change to the flow of electrical current going through it.

In 1971, Leon O. Chua (1936–) postulated a component called a *memristor* (“memory resistor”). A professor in the electrical engineering and computer sciences department at the University of California, Berkeley, Chua strongly believed that this fourth component should exist to provide conceptual symmetry with the resistor, inductor, and capacitor.

As its name might suggest, the simplest way to think about this is as a resistor with memory. Among many other potential uses, a memristor (if such a beast existed) could be used to implement computer memory; a direct current applied in the component could adjust its apparent resistance, which could subsequently be observed (“read”) using alternating current.

In April 2008, almost four decades after Chua’s original paper, a team of engineers from Hewlett Packard (HP) led by the scientist R. Stanley Williams published a paper describing the construction of a working memristor based on a thin film of titanium dioxide. Being much simpler than currently popular MOSFET transistor switches [see *Chapter 4: Semiconductors (Diodes and Transistors)* and *Chapter 15: Memory ICs*] and also able to implement one bit of memory in a single device, memristors are believed to offer some very exciting possibilities for the future.

IMPEDANCE AND REACTANCE

This is where things start to get a little tricky, but I’ll try to be gentle (if your brain starts to overheat, jump immediately to the *Unit Qualifiers* topic at the end of this chapter and return to ponder this later when your thinking organ has cooled down).

The term *Direct Current* (DC), also known as *continuous current*, refers to the unidirectional flow of electric charge. Direct current is produced by sources such as batteries or solar cells. By comparison, the term *Alternating Current* (AC) refers to an electrical current whose magnitude and direction vary cyclically.

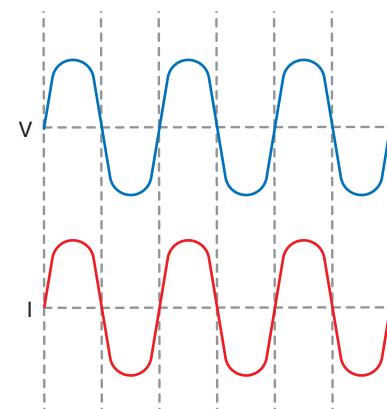
¹²We’ll consider the difference between passive and active components in *Chapter 4: Semiconductors (Diodes and Transistors)*.

Let's assume that we have an AC signal in the form of a sine wave, and that this signal is being fed into a simple electrical circuit.¹³ Purely for the sake of discussion, let's first assume that we have an "ideal circuit" comprising only resistors and zero-resistance wires. In particular, let's assume that there are no capacitors (or capacitive effects) or inductors (or inductive effects) in this circuit. In this case, the voltages and currents associated with any resistors in the circuit will always be in phase [Figure 3.17(a)].

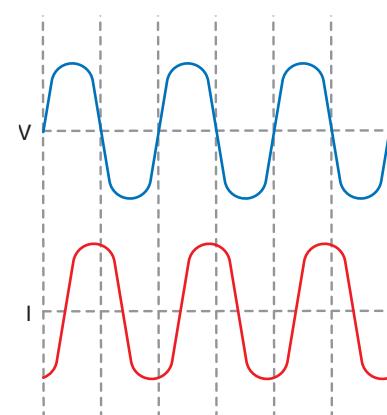
By comparison, in the case of a circuit that also contains capacitors (or capacitive effects) and/or inductors (or inductive effects), voltage and current values across these elements will be out of phase [Figure 3.17(b)]. Specifically, the voltage across a capacitor will *lag* the current passing through that capacitor, while the voltage across an inductor will *lead* the current passing through that inductor.

And so we come to the concept of *electrical impedance*, or simply *impedance*, which describes a measure of opposition to a sinusoidal alternating current. Electrical impedance extends the concept of resistance to AC circuits, describing not only the relative amplitudes of the voltage and current, but also the relative phases.

Impedance (Z) is described as a complex quantity¹⁴ using the equation $Z = R \times iX$, where the *real* part of the impedance is the resistance (R) and the imaginary part is the *reactance* (X). The SI unit of impedance, resistance, and reactance is the ohm.



(a) Voltage and current in phase



(b) Voltage and current out of phase

FIGURE 3.17

Voltage and current waveforms in and out of phase.

ADMITTANCE, CONDUCTANCE, AND SUSCEPTANCE

Just for the sake of completeness (on the off-chance your aunt asks you about this during your next family get-together, for example), let's briefly define a few

¹³See Chapter 4: *Semiconductors (Diodes and Transistors)* for a discussion on the difference between the terms *electrical* and *electronic*.

¹⁴If you aren't familiar with the concept of complex numbers, just take it from me that they have a real part and an imaginary part, where the imaginary part is prefixed by the letter "i." On the off-chance you're interested, "i" represents the square root of negative 1. This has to be an imaginary number, because anything multiplied by itself (irrespective of whether it is positive or negative) will result in a positive value.

more terms. Before we start, I might point out that I personally have never had occasion to use any of these little rascallions in day-to-day conversation, but you never know when something like this might come in handy.

Consider a resistor with a resistance value of R . As we previously discussed, R is measured in ohms and is represented by the capital Greek letter Omega " Ω ." Now, engineers often need to use the reciprocal of R in their calculations. This is, of course, $1/R$, and you would think that they would be happy to leave it at that. Sadly not; instead, they call the resulting value *conductance*, which they measure in *mohs* (that's "ohms" spelled backwards) and represent with an upside-down Omega character.

But wait, there's more. Do you recall the equation from the previous topic: $Z = R \times iX$, where Z is the impedance, R is the resistance, and X is the reactance? Well, the reciprocal of the impedance (Z) is called the *admittance* (Y) and the reciprocal of the reactance (X) is called the *susceptance* (B).

Historically, admittance, conductance, and susceptance were all measured in mohs. More recently, the official SI unit for all three is the *siemens* (S), which is named after the German inventor and industrialist Ernst Werner von Siemens (popularly known as Werner von Siemens) (1816–1892).

UNIT QUALIFIERS

Engineers often work with very large or very small values of voltage, current, resistance, capacitance, and inductance. As an alternative to writing endless zeros, electrical quantities can be annotated with the qualifiers given in Table 3.1. For example, $15\text{ M}\Omega^{15}$ (15 megaohms) means fifteen million ohms; 4 mA (4 milliamps) means four one-thousandths of an amp; and 20 fF (20 femtofarads) means a very small capacitance indeed.

One last point that's worth noting is that the qualifiers *kilo*, *mega*, *giga*, *tera*, and so forth mean slightly different things when we use them to describe the size of computer memory. The reasoning behind this (and many other mysteries) is revealed in *Chapter 15: Memory ICs*.

¹⁵Remember that the Greek letter omega " Ω " is used to represent resistance.

Table 3.1**Unit Qualifiers**

Qualifier	Symbol	Factor	Name (U.S.)
yotta	Y	10^{24}	septillion (one million million million million)
zetta	Z	10^{21}	sextrillion (one thousand million million million)
exa	E	10^{18}	quintillion (one million million million)
peta	P	10^{15}	quadrillion (one thousand million million)
tera	T	10^{12}	trillion (one million million)
giga	G	10^9	billion (one thousand million) ¹⁶
mega	M	10^6	million
kilo	k	10^3	thousand
milli	m	10^{-3}	thousandth
micro	μ or u	10^{-6}	millionth
nano	n	10^{-9}	billionth (one thousandth of one millionth)
pico	p	10^{-12}	trillionth (one millionth of one millionth)
femto	f	10^{-15}	quadrillionth (one thousandth of one millionth of one millionth)
atto	a	10^{-18}	quintillionth (one millionth of one millionth of one millionth)
zepto	z	10^{-21}	sextillionth (one thousandth of one millionth of one millionth of one millionth)
yocto	y	10^{-24}	septillionth (one millionth of one millionth of one millionth of one millionth)

¹⁶In Britain, “billion” traditionally used to mean “a million million” (10^{12}). However, for reasons unknown, the Americans decided that “billion” should mean “a thousand million” (10^9). Generally speaking, in order to avoid the confusion that would otherwise ensue, most countries in the world (including Britain) have decided to go along with the Americans.

This page intentionally left blank

CHAPTER 4

Semiconductors (Diodes and Transistors)

HERDING WILD ELECTRONS

As we noted in *Chapter 2: Atoms, Molecules, and Crystals*, electricity may be considered to be vast herds of electrons migrating from one place to another, while electronics is the art and science of controlling these herds: starting them, stopping them, deciding where they can roam, and determining the activities they are going to perform while on their way. Ever since humans discovered electricity (as opposed to electricity—in the form of lightning—discovering us), taming the little rascal and bending it to our will has occupied a lot of thought and ingenuity.

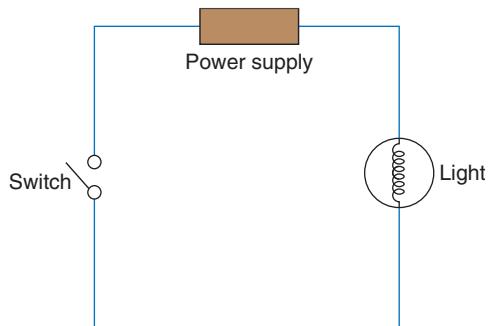
33

The first, and certainly the simplest, form of control is the humble mechanical switch. Consider a circuit consisting of a switch, a power supply (say a battery), and a light bulb (Figure 4.1).

When the switch is CLOSED, the light is ON; when the switch is OPEN, the light is OFF. As we'll see in *Chapter 5: Primitive Logic Functions*, we can actually implement some interesting logical functions by connecting switches together in different ways. If mechanical switches were all we had to play with, however, the life of an electronics engineer would be somewhat boring. The folks in the dim-and-distant past agreed, so they decided that something with a little more "zing" was required ...

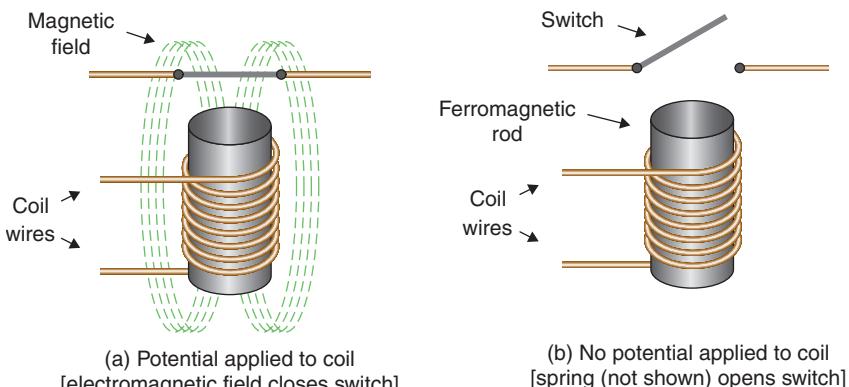
FIGURE 4.1

The simplest control device is a switch.



THE ELECTROMECHANICAL RELAY

By the end of the 19th century, when Queen Victoria still held sway over all she surveyed, the most sophisticated form of control for electrical systems was the electromechanical relay (Figure 4.2). This device consisted of a rod of iron (or some other ferromagnetic

**FIGURE 4.2**

The electromechanical relay.

material) wrapped in a coil of wire (the wire would be coated by a layer of insulating material to prevent the coil windings forming electrical connections with each other or with the rod). Applying an electrical potential across the ends of the coil caused the iron rod to act like a magnet. The magnetic field could be used to attract another piece of iron acting as a switch [Figure 4.2(a)]. Removing the potential from the coil caused the iron bar to lose its magnetism, and a small spring would return the switch to its inactive state [Figure 4.2(b)].

The relay is a digital component, because it is either ON or OFF. Although simple in concept, these devices were to prove enormously important with regard to early control systems. This is because the outputs from one or more relays can be used to control other relays, and the outputs from these relays can be used to control yet more relays, and so on and so forth. In fact, by connecting relays together in different ways it's possible to create all sorts of things.

Perhaps the most ambitious use of relays was to build gigantic electro-mechanical computers, such as the Harvard Mark 1. Constructed between 1939 and 1944, the Harvard Mark 1 was 50 feet long, 8 feet tall, and contained over 750,000 individual components.

The problem with relays (especially the types that were around in the early days) is that they can only switch a limited number of times a second. This severely limits the performance of a relay-based computer. For example, the *Harvard Mark 1* took approximately six seconds to multiply two numbers together, so engineers started to look around for something that could switch faster ...

THE FIRST VACUUM TUBES

In 1879, the legendary American inventor Thomas Alva Edison (1847–1931) publicly demonstrated his incandescent electric light bulb for the first time.¹ This is the way it worked. A filament was mounted inside a glass bulb. Then all the air was sucked out, leaving a vacuum. When electricity was passed through the filament, it began to glow brightly (the vacuum stopped it from bursting into flames).

A few years later in 1883, one of Edison's assistants discovered that he could detect electrons flowing through the vacuum from the lighted filament to a metal plate mounted inside the bulb. Unfortunately, Edison didn't develop this so-called *Edison Effect* any further. In fact, it wasn't until 1904 that the English physicist Sir John Ambrose Fleming (1849–1945) used this phenomenon to create the first vacuum tube.² This device, known as a *diode*, had two terminals and conducted electricity in only one direction (a feat that isn't as easy to achieve as you might think).

In 1906, the American inventor Lee de Forest (1873–1961) introduced a third electrode into his version of a vacuum tube. The resulting *triode* could be used as both an amplifier and a switch. De Forest's triodes revolutionized the broadcasting industry (he presented the first live opera broadcast and the first news report on radio). Furthermore, their ability to act as switches was to have a tremendous impact on digital computing.

One of the most famous early electronic digital computers is the *Electronic Numerical Integrator and Calculator* (ENIAC), which was constructed at the University of Pennsylvania between 1943 and 1946. Occupying 1000 square feet, weighing in at 30 tons, and employing 18,000 vacuum tubes, ENIAC was a monster ... but it was a monster that could perform fourteen multiplications or 5000 additions a second, which was way faster than the relay-based Harvard Mark 1.

However, in addition to requiring enough power to light a small town, ENIAC's vacuum tubes were horrendously unreliable, so researchers started looking for a smaller, faster, and more dependable alternative that didn't demand as much power ...

¹Contrary to popular belief, this wasn't the world's first incandescent bulb. In 1878, the English physicist and electrician, Sir Joseph Wilson Swan (1828–1914) successfully demonstrated a true incandescent bulb, a year earlier than Edison.

²Vacuum tubes are known as *valves* in England. This is based on the fact that they can be used to control the flow of electricity, similar in concept to the way in which their mechanical namesakes are used to control the flow of fluids.

SEMICONDUCTORS

Most materials are conductors, insulators, or something in-between, but a special class of materials known as *semiconductors* can be persuaded to exhibit both conducting and insulating properties.

The first semiconducting material to undergo serious evaluation was the element germanium (chemical symbol: Ge). However, for a variety of reasons, silicon (chemical symbol: Si) replaced germanium as the semiconductor of choice. As silicon is the main constituent of sand and one of the most common elements on earth (silicon accounts for approximately 28% of the earth's crust), we aren't in any danger of running out of it in the foreseeable future.

Pure crystalline silicon acts as an insulator; however, scientists at Bell Laboratories in the United States found that by inserting certain impurities into the crystal lattice, they could make silicon act as a conductor. The process of inserting the impurities is known as *doping*, and the most commonly used *dopants* are boron atoms (chemical symbol: B) with three electrons in their outermost electron shells, and phosphorus atoms (chemical symbol: P) with five.

If a piece of pure silicon is surrounded by a gas containing boron or phosphorus and heated in a high-temperature oven, the boron or phosphorus atoms will permeate the crystal lattice and displace some silicon atoms without disturbing other atoms in the vicinity. This process is known as *diffusion*. Boron-doped silicon is called *P-type silicon* and phosphorus-doped silicon is called *N-type silicon* (Figure 4.3).³

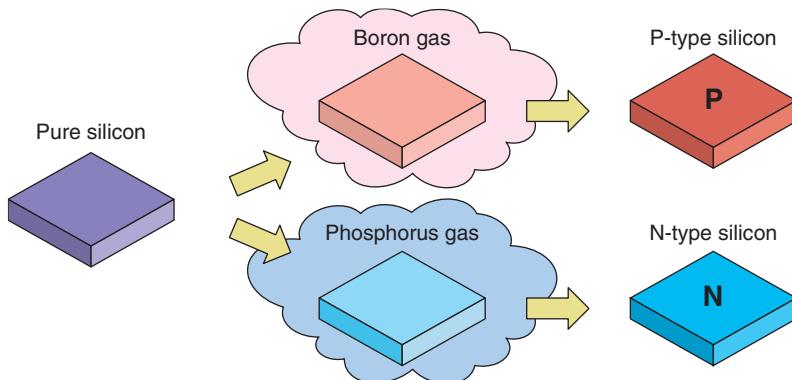


FIGURE 4.3

Creating P-type and N-type silicon.

³If you ever happen to run across a "full-up" illustration of an integrated circuit, as shown in *Chapter 14: Integrated Circuits (ICs)*, you may see annotations like *n*, *n+*, *n++*, *p*, *p+*, and *p++*. In this case, the *n* and *p* stand for standard N-Type and P-type material (which we might compare to an average guy); the *n+* and *p+* indicate a heavier level of doping (say a bodybuilder or the author flexing his rippling muscles on the beach); and the *n++* and *p++* indicate a really high level of doping (like a weightlifter on steroids).

Due to the fact that boron atoms have only three electrons in their outermost electron shells, they can only make bonds with three of the silicon atoms surrounding them. This leaves the fourth silicon atom un-sated and eager to fill its outermost electron shell. Thus, the site (location) occupied by a boron atom in the silicon crystal will accept a free electron with relative ease and is therefore known as an *acceptor* (it's also called a *hole*). So, why do we call this "P-Type silicon?" Well (unofficially), due to the fact that each site of a boron atom is happy to accept an electron, we can visualize this site as being "sort-of" positive. However, the more "official" reason is that we can regard holes as being positive charge carriers (sort of the opposite of electrons).

By comparison, as phosphorous atoms have five electrons in their outermost electron shells, the site of a phosphorous atom in the silicon crystal will donate an electron with relative ease and is therefore known as a *donor*. Due to the fact that it is happy to donate an electron, we can visualize this site as being "sort-of" negative. But the real reason we call this "N-Type silicon" is that the conducting electrons are, of course, negative charge carriers.

SEMICONDUCTOR DIODES

As was noted above, pure crystalline silicon acts as an insulator. By comparison, both P-type and N-type silicon are reasonably good conductors (Figure 4.4).

The point when things start to become really interesting, however, is when a piece of silicon is doped such that part is P-type and part is N-type. In order to wrap our brains around this, consider what would happen if we were to take a wooden toothpick, dip half of it in a pool of melted wax, remove it from the wax and let it harden, briefly immerse the whole thing in a cup of colored dye, take it out of the dye, dry it off, and scrape away the wax. This leaves one-half of our toothpick in its original state while the other half has been colored.

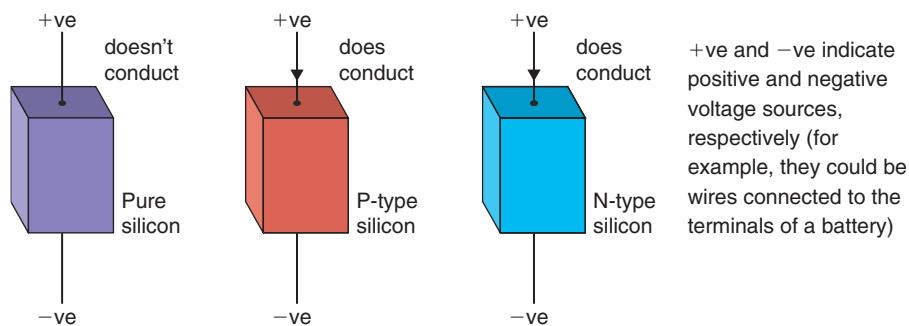


FIGURE 4.4

Pure silicon compared to P-type and N-type silicon.

Next, we flip the toothpick upside down, dip the half of the toothpick we just tinted in melted wax, remove it from the wax and let it harden, briefly immerse the whole thing in a cup containing a different colored dye, take it out of the dye, dry it off, and scrape away the wax. Now, one-half of our toothpick will be one color, while the other half is a different color.

Although the actual process is much more complex, we can do something similar with a piece of silicon to make part P-type and part N-type as illustrated in Figure 4.5. We'll return to consider this process in more detail in *Chapter 14: Integrated Circuits (ICs)*. The result is known as a *p-n junction*.

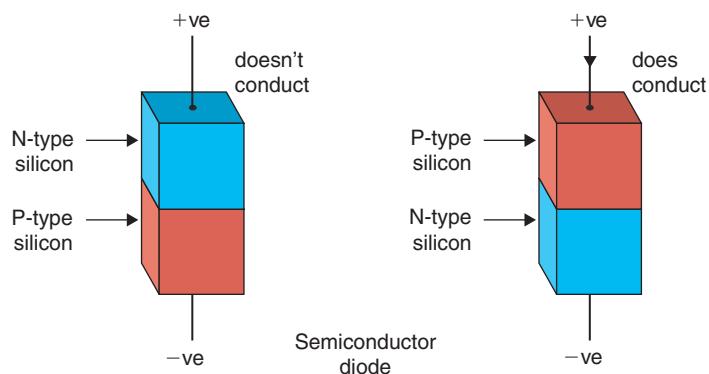


FIGURE 4.5

Mixing P-type and N-type silicon.

Approximate actual size



(a) Diode component



(b) Symbol

FIGURE 4.6

Diode: Component and symbol.

As we see, the silicon with both P-type and N-type material conducts electricity in only one direction; in the other direction it behaves like an OPEN (OFF) switch.⁴ These structures, known as *semiconductor diodes*,⁵ come in many shapes and sizes; an example could be as shown in Figure 4.6.

If the triangular body of the symbol is pointing in the classical direction of current flow (more positive to more negative), the diode will conduct. An individually packaged diode (which would be referred to as a *discrete component*) consists of a piece of silicon with connections to external leads, all encapsulated in a protective package (the silicon is typically smaller than a grain of sand). The package protects the silicon from moisture and other impurities and, when the diode is operating, helps to conduct heat away from the silicon.

⁴If you want to know more about how this works at the nitty-gritty level, including terms like *depletion zones*, then bounce over to Appendix G: More on Semiconductors.

⁵The *semiconductor* portion of *semiconductor diode* was initially used to distinguish these components from their vacuum tube-based cousins. As semiconductors took over, however, everyone started to refer to them simply as *diodes*.

Due to the fact that diodes (and transistors, as discussed below) are formed from solids—as opposed to vacuum tubes, which are largely formed from empty space—people started to refer to components formed from semiconductors as *solid-state electronics*.

BIPOLAR JUNCTION TRANSISTORS (BJTS)

More complex components called *transistors* can be created by forming a sandwich out of three regions of doped silicon. The transistor and subsequently the integrated circuit must certainly qualify as two of the greatest inventions of the 20th century.

Unfortunately, serious research on semiconductors didn't really commence until World War II. At that time, it was recognized that devices formed from semiconductors had potential as amplifiers and switches, and could therefore be used to replace the prevailing technology of vacuum tubes, but that they would be much smaller, lighter, and would require less power. All these factors were of interest to the designers of electronic systems such as radar, which were to play a large role in the war.

Bell Laboratories in the United States began research into semiconductors in 1945, and physicists William Shockley (1910–1989), Walter Brattain (1902–1987), and John Bardeen (1908–1991) succeeded in creating the first point-contact germanium transistor on December 23, 1947. (They took a break for the Christmas holidays before publishing their achievement, which is why some reference books state that the first transistor was created in 1948.)

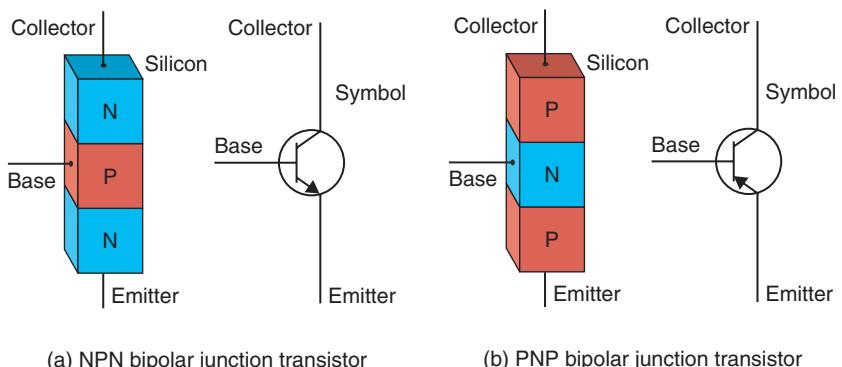
In 1950, Shockley invented a new device called a *Bipolar Junction Transistor* (BJT),^{6,7} which was more reliable, easier and cheaper to build, and gave more consistent results than point-contact devices. BJTs are formed from three pieces of doped silicon, called the *collector*, *base*, and *emitter*. There are two basic types of bipolar transistors, called NPN and PNP,⁸ where these names relate to the manner in which the silicon is doped (Figure 4.7).

In the analog world, a transistor can be used as a voltage amplifier, a current amplifier, or a switch; in the digital world, a transistor is primarily considered to be a switch. The structure of a transistor between the collector and emitter

⁶In conversation, the term *BJT* is spelled out as “B-J-T.”

⁷Apropos of nothing at all, the first TV dinner was marketed by the C.A. Swanson company three years later.

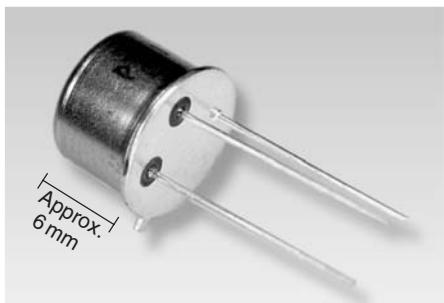
⁸In conversation, the terms *NPN* and *PNP* are spelled out as “N-P-N” and “P-N-P,” respectively.

**FIGURE 4.7**

Bipolar junction transistors (BJTs).

FIGURE 4.8

Individually packaged transistor. (Photo courtesy of Alan Winstanley)



terminals is similar to that of two diodes connected back-to-back. Two diodes connected in this way would typically not conduct; however, when signals are applied to the base terminal, the transistor can be turned ON or OFF. If the transistor is turned ON, it acts like a CLOSED switch and allows current to flow between the collector and the emitter; if the transistor is turned OFF, it acts like an OPEN switch and no current flows. We may think of the collector and emitter as *data* terminals, and the base as the *control* terminal.

As for a diode, an individually packaged transistor consists of the silicon, with connections to external leads, all encapsulated in a protective package (the silicon is typically smaller than a grain of sand). The package protects the silicon from moisture and other impurities and helps to conduct heat away from the silicon when the transistor is operating. Transistors may be packaged in plastic or in little metal cans about a quarter of an inch in diameter, with three leads sticking out of the bottom (Figure 4.8).

By the late 1950s, bipolar transistors were being manufactured out of silicon rather than germanium (although germanium had certain electrical advantages, silicon was cheaper and easier to work with). The original bipolar transistors were manufactured using the *mesa process*, in which a doped piece of silicon called the *mesa* (or base) was mounted on top of a larger piece of silicon forming the collector, while the emitter was created from a smaller piece of silicon embedded in the base.

In 1959, the Swiss physicist Jean Hoerni (1924–1997) invented the *planar process*, in which optical lithographic techniques were used to diffuse the base into the collector and then to diffuse the emitter into the base. One of Hoerni's colleagues, Robert Noyce (1927–1990), invented a technique for growing an insulating layer of silicon dioxide over the transistor, leaving small areas over the base and emitter exposed and diffusing thin layers of aluminum into these areas to create wires. The processes developed by Hoerni and Noyce led directly to modern integrated circuits. These techniques are discussed in more detail in *Chapter 14: Integrated Circuits (ICs)*.

METAL-OXIDE SEMICONDUCTOR FIELD-EFFECT TRANSISTORS (MOSFETS)

In reality, the properties of semiconductors did not start to become well understood until the 1950s. Having said this, as far back as 1925, the Austro-Hungarian scientist Dr. Julius Edgar Lilienfeld (1881–1963) proposed the basic principles behind what we would now recognize as a semiconductor device called a *Metal-Epitaxial Semiconductor Field-Effect Transistor* (MESFET) being used as an amplifier.⁹

In 1926, Dr. Lilienfeld immigrated to America and applied for a patent for this device. On the off-chance you're interested; the title of this little scamp (US Patent 1,745,175) was "*Method and apparatus for controlling electric currents*." Two years later, in 1928 (US Patent 1,900,018), he described what we would now recognize as a depletion-mode MOSFET.¹⁰

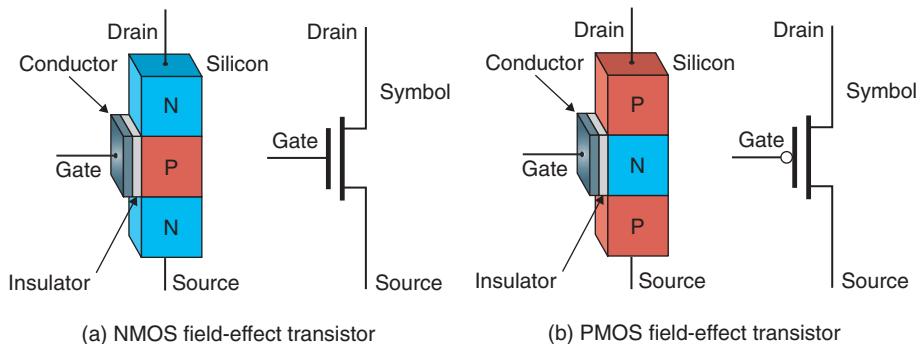
The term MOSFET (or MOS-FET or MOS FET) stands for *Metal-Oxide Semiconductor Field-Effect Transistor* (we'll explain what this mouthful means in a moment).^{11,12} In 1960, Dawon Kahng and Martin M. (John) Atalla at Bell Labs fabricated the first successful MOSFET. In 1962, Steven Hofstein and Fredric Heiman at the RCA research laboratory in Princeton, New Jersey, created an experimental integrated circuit comprising 16 MOSFETs.

⁹As an aside, during the early 1900s, Lilienfeld did some work with Count Ferdinand von Zeppelin (1838–1917) on designing hydrogen-filled dirigibles.

¹⁰Nothing is simple. The MOSFETs discussed in this chapter are *enhancement-type* devices, which are OFF unless a control signal is applied to the gate terminal to turn them ON. By comparison, *depletion-type* devices are ON unless a control signal is applied to turn them OFF. And then there are *Junction FETs* (JFETs) and MESFETs. See also *Appendix G: More on Semiconductors*, for more details on all of these little ragamuffins.

¹¹In conversation, the term *MOSFET* is pronounced as a single word, where "MOS" rhymes with "boss" and "FET" rhymes with "bet."

¹²These may also be referred to as *Insulated Gate Field-Effect Transistors* (IGFETs).

**FIGURE 4.9**

Metal-oxide semiconductor field-effect transistors (MOSFETs).

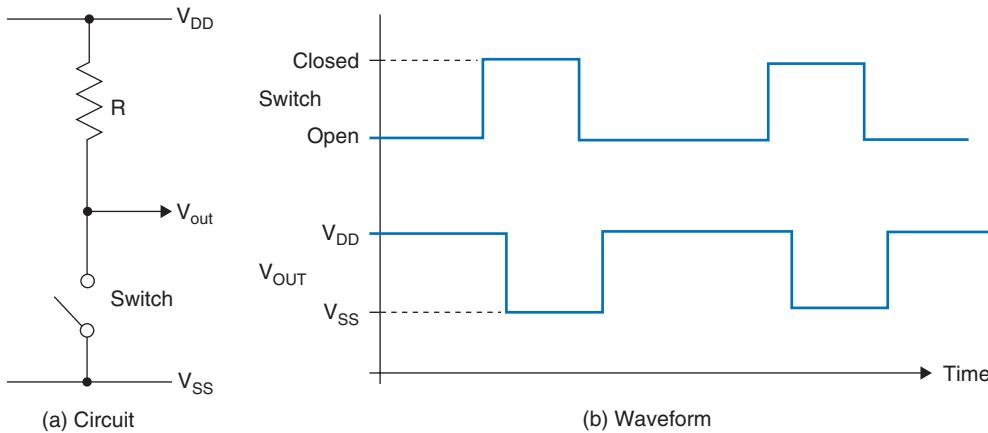
Although the original MOSFETs were somewhat slower than their bipolar transistors, they were cheaper, smaller, and used less power. Also of interest was the fact that modified metal-oxide semiconductor structures could be made to act as capacitors or resistors.

There are two basic types of MOSFETs, called *n-channel* and *p-channel*; once again these names relate to the way in which the silicon is doped (Figure 4.9).

In the case of these devices, the *drain* and *source* form the *data* terminals and the *gate* acts as the *control* terminal. Unlike bipolar devices, the control terminal is connected to a conducting plate, which is insulated from the silicon by a layer of nonconducting oxide. In the original devices the conducting plate was metal—hence, the term *metal-oxide*—but this is now something of a misnomer because modern versions tend to use a layer of *polycrystalline silicon (polysilicon)*. When a signal is applied to the gate terminal, the plate, insulated by the oxide, creates an electromagnetic field, which turns the transistor ON or OFF—hence, the term *field-effect*.

Now, this is the bit that always confuses the unwary, because the term *channel* refers to the piece of silicon under the gate terminal; that is, the piece linking the drain and source regions. But the channel in the n-channel device is formed from P-type material, while the channel in the p-channel device is formed from N-type material.

At first glance, this would appear to be totally counterintuitive, but there is reason behind the madness. Let's consider the n-channel device. In order to turn this ON, a positive voltage is applied to the gate. This positive voltage attracts any negative electrons in the P-type material and causes them to accumulate beneath the oxide layer where they form a negative channel—hence, the term *n-channel*. In fact, saying “n-channel” and “p-channel” is a bit of a mouthful,

**FIGURE 4.10**

Resistor-switch circuit.

so instead we typically just refer to these as *NMOS* and *PMOS transistors*, respectively.¹³

This book concentrates on MOSFETs, because their symbols, construction, and operation are relatively easy to understand as compared to their BJT cousins.

THE TRANSISTOR AS A SWITCH

To illustrate the application of a transistor as a switch, first consider a simple circuit comprising a resistor and a real switch (Figure 4.10).

We'll consider the meaning behind the V_{DD} and V_{SS} power supply labels in a moment. For our purposes here, let's simply assume that V_{DD} is more positive than V_{SS} .

When the switch is OPEN (OFF), V_{OUT} is connected via the resistor to V_{DD} ; when the switch is CLOSED (ON), V_{OUT} is connected via the switch directly to V_{SS} . In this latter case, V_{OUT} takes the value V_{SS} because, like people, electricity takes the path of least resistance, and the resistance to V_{SS} through the closed switch is far less than the resistance to V_{DD} through the resistor. Observe that the waveforms in Figure 4.10 show a delay between the switch operating and V_{OUT} responding. Although this delay is extremely small, it is important to note that there will always be some elements of delay in any physical system.

Now consider the case where the switch is replaced with an NMOS transistor (Figure 4.11). Let's assume that there's a wire connected to the control input of

¹³In conversation, *NMOS* and *PMOS* are pronounced “N-MOS” and “P-MOS,” respectively. That is, by spelling out the first letter followed by “MOS” to rhyme with “boss.”

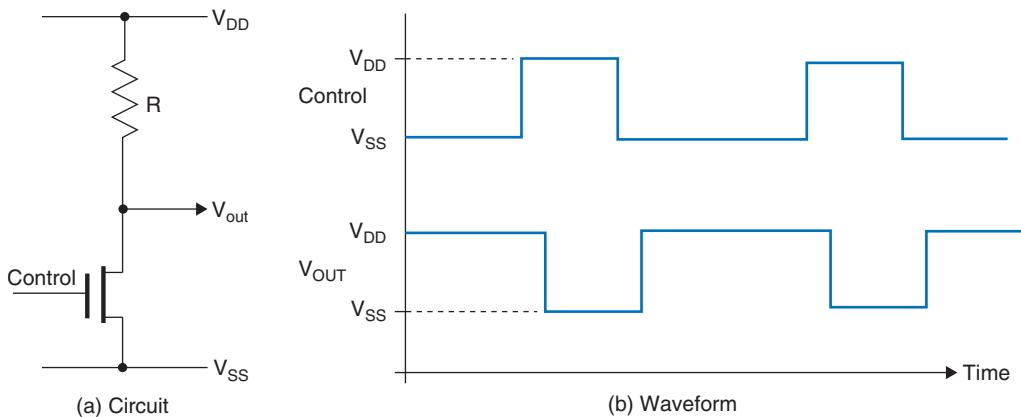


FIGURE 4.11

Resistor-NMOS transistor circuit

the transistor (this wire isn't shown here for simplicity), and that the other end of this wire can be switched back and forth between V_{DD} and V_{SS} .

When the control input to an NMOS transistor is connected to V_{ss} , the transistor is turned OFF and acts like an OPEN switch; when the control input is connected to V_{dd} , the transistor is turned ON and acts like a closed switch. Thus, the transistor functions in a similar manner to the mechanical switch. However, a mechanical switch controlled by hand can only be operated a few times a second, but a transistor's control input can be driven by other transistors, allowing it to be operated hundreds of millions of times a second.

Returning to the labels V_{SS} and V_{DD} , these are commonly used in circuits employing MOSFETs. At this point we have little interest in their actual values, and for the purpose of these examples, need only assume that the V_{DD} supply rail¹⁴ is more positive than the V_{SS} rail. Why do we use these labels? Well, if you cast your mind back to *Chapter 3: Conductors, Insulators, and Other Stuff*, you will recall that current—in the form of electrons—flows from the more negative source to the more positive target. In the case of the circuit shown in Figure 4.11, the current flows from the V_{SS} rail (the “source”) into the transistor’s source terminal, through the transistor, and “drains away” out of its drain terminal into the V_{DD} rail.

GALLIUM ARSENIDE SEMICONDUCTORS

Silicon is known as a *four-valence semiconductor* because it has four electrons available to make bonds in its outermost electron shell. Although silicon is the most

¹⁴I don't know where the term *rail* comes from in this context, but engineers say "supply rail" or "power supply rail" all the time.

commonly used semiconductor, there is another that requires some mention. The element gallium (chemical symbol: Ga) has three electrons available in its outermost shell, and the element arsenic (chemical symbol: As) has five. A crystalline structure of gallium arsenide (GaAs) is known as a III-V valence semiconductor¹⁵ and can be doped with impurities in a similar manner to silicon.

In a number of respects, GaAs is preferable to silicon, not the least of which is that GaAs transistors can switch approximately eight times faster than their silicon equivalents. However, GaAs is hard to work with, which results in GaAs transistors being more expensive than their silicon cousins.

LIGHT-EMITTING DIODES (LEDS)

On February 9, 1907, one of Marconi's engineers, Mr. H.J. Round of New York, New York, had a letter published in *Electrical World* magazine as follows:

To the editors of Electrical World:

Sirs: During an investigation of the unsymmetrical passage of current through a contact of carborundum and other substances a curious phenomenon was noted. On applying a potential of 10 volts between two points on a crystal of carborundum, the crystal gave out a yellowish light.

Mr. Round went on to note that some crystals gave out green, orange, or blue light. This is quite possibly the first documented reference to the effect upon which special components called *light-emitting diodes* (LEDs) are based.¹⁶

Sad to relate, no one seemed particularly interested in Mr. Round's discovery, and nothing really happened until 1922, when the same phenomenon was observed by O. V. Losov in Leningrad. Losov took out four patents between 1927 and 1942, but he was killed during World War II and the details of his work were never discovered.

In fact, it wasn't until 1951, following the discovery of the bipolar transistor, that researchers really started to investigate this effect in earnest. They found that by creating a semiconductor diode from a compound semiconductor formed from two or more elements—such as gallium arsenide (GaAs) as mentioned

¹⁵In conversation, the Roman Numerals "III-V" are pronounced "three-five."

¹⁶In conversation, the term LED may be spelled out as "L-E-D" (in which case you would say "an L-E-D") or pronounced as a single word to rhyme with "bed" (in which case you would say "a LED").

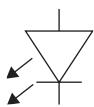


FIGURE 4.12
Symbol for a LED.

in the previous topic—light is emitted from the p-n junction, that is, the junction between the P-type and N-type doped materials.

As for a standard diode, an LED conducts electricity in only one direction (and it emits light only when it's conducting). Thus, the symbol for an LED is similar to that for a normal diode, but with two arrows to indicate light being emitted (Figure 4.12).

An LED formed from pure gallium arsenide emits infrared light, which is useful for sensors, but which is invisible to the human eye. It was discovered that adding aluminum to the semiconductor to give aluminum gallium arsenide (AlGaAs) resulted in red light humans could see. Thus, after much experimentation and refinement, the first red LEDs started to hit the streets in the late 1960s.

LEDs are interesting for a number of reasons, not the least of which is that they are extremely reliable, they have a very long life (typically 100,000 hours as compared to 1000 hours for an incandescent light bulb), they generate very pure, saturated colors, and they are extremely energy efficient (LEDs use up to 90% less energy than an equivalent incandescent bulb).

Over time, more materials were discovered that could generate different colors. For example, gallium phosphide gives green light, and aluminum indium gallium phosphite can be used to generate yellow and orange light. For a long time, the only color missing was blue. This was important because blue light has the shortest wavelength of visible light, and engineers realized that if they could build a blue laser diode they could quadruple the amount of data that could be stored on, and read from, a CD-ROM or DVD.

However, although semiconductor companies spent hundreds of millions of dollars desperately trying to create a blue LED, the little rascal remained elusive for more than three decades. In fact, it wasn't until 1996 that the Japanese Electrical Engineer Shuji Nakamura demonstrated a blue LED based on gallium nitride. Quite apart from its data storage applications, this discovery also makes it possible to combine the output from a blue LED with its red and green cousins to generate white light. Many observers believe that this may ultimately relegate the incandescent light bulb to the museum shelf.

ORGANIC LEDs (OLEDS)

The traditional LEDs discussed in the previous topic are based on semiconductors formed from metalloidal materials such as silicon. When current flows through the LED, positive and negative charges combine and light is emitted.

An *Organic Light-Emitting Diode* (OLED) performs the same trick, but it is based on thin layers of organic molecules (the term *organic* is used in this context because these molecules have a “backbone” formed from carbon atoms, and carbon is the key element for organic life as we know it).

When used to produce displays, OLED technology produces self-luminous displays that do not require backlighting. These properties result in compact displays that require very little power and are much thinner and brighter than their *Liquid Crystal Display* (LCD) counterparts.

ACTIVE VERSUS PASSIVE AND ELECTRIC VERSUS ELECTRONIC

In this context, the term *active* is used to refer to a component that can use an electrical signal to control the current passing through it; for example, transistors are classed as *active devices*. By comparison, components that are incapable of controlling current by means of another electrical signal are referred to as *passive devices*. On this basis, resistors, capacitors, inductors, and even diodes—all of which simply respond (in a “passive” sort of way) to whatever electrical signals life throws at them—are therefore all classed as passive devices.

Some purists would say that in order for a circuit to be properly called “electronic,” it must contain one or more active devices. On this basis, a circuit comprising only resistors, capacitors, and inductors would be considered to be an “electric circuit” rather than an “electronic circuit.” (Personally, I don’t care what other people think and I would still call it an electronic circuit, but you can make your own decision on this point.)

This page intentionally left blank

CHAPTER 5

Primitive Logic Functions

SWITCH REPRESENTATIONS OF AND AND OR FUNCTIONS

Before we leap into the fray, let's first perform a couple of simple thought experiments involving switches and light bulbs. Consider an electrical circuit consisting of a power supply, a light, and two switches connected in *series*, one after the other [Figure. 5.1(a)].

The switches are the inputs to the circuit and the light is the output. A truth table provides a convenient way to represent the operation of the circuit [Figure. 5.1(b)]. As the light is only ON when both the a and b switches are CLOSED (ON), this circuit could be said to perform a 2-input AND function.¹

49

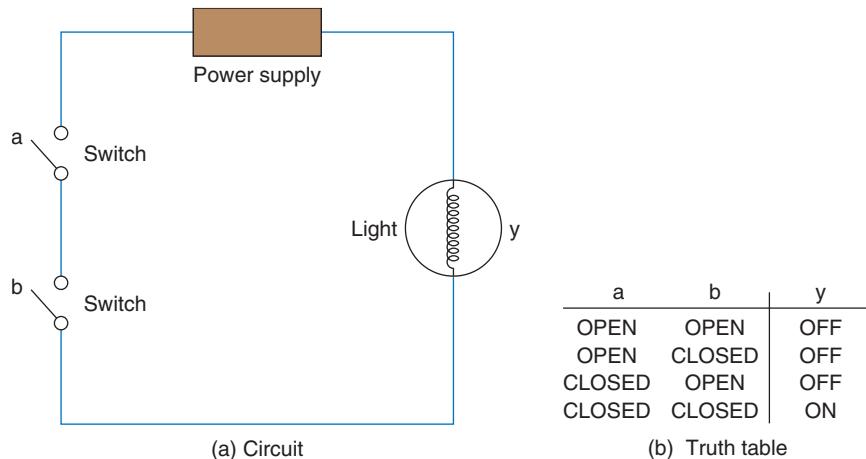
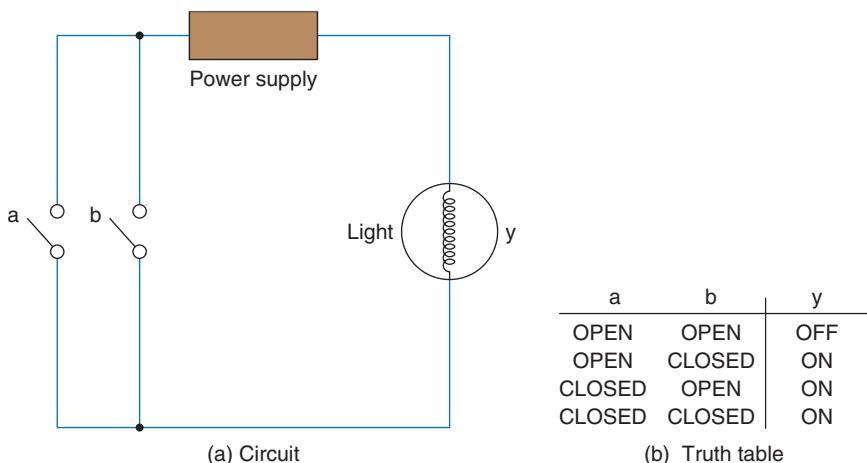


FIGURE 5.1

Switch representation of a 2-input AND function.

¹A 3-input version could be constructed by adding a third switch in *series* with the first two.

**FIGURE 5.2**

Switch representation of a 2-input OR function.

Not surprisingly, the way in which the circuit functions depends on the way in which the switches are connected. Consider another circuit in which two switches are connected in *parallel* (side by side) [Figure. 5.2(a)].

Think about how this circuit will work, and then compare your musings to its truth table [Figure. 5.2(b)]. In this case, as the light is ON when either a or b are CLOSED (ON), this circuit could be said to perform a 2-input OR function.²

In a limited respect, we might consider that these circuits are making simple logical decisions; two switches offer four combinations of OPEN (OFF) and CLOSED (ON), but only certain combinations cause the light to be turned ON.

FALSE AND TRUE VERSUS OPEN AND CLOSED

Logic functions such as AND and OR are generic concepts that can be implemented in a variety of ways, including switches as illustrated above, transistors for use in computers, and even pneumatic devices for deployment in hostile environments (close to the core of a nuclear reactor, for example).³

Thus, instead of drawing circuits using light switches, it is preferable to make use of more abstract forms of representation. This permits designers to specify the function of systems with minimal consideration as to their final physical realization. To facilitate this, special symbols are employed to represent logic functions, and truth table assignments are specified using the abstract terms

²A 3-input version could be constructed by adding a third switch in *parallel* with the first two.

³I've also seen logic functions implemented in some very strange ways, such as falling/not-falling dominos or marbles interacting with pivoting/not-pivoting strips of wood.

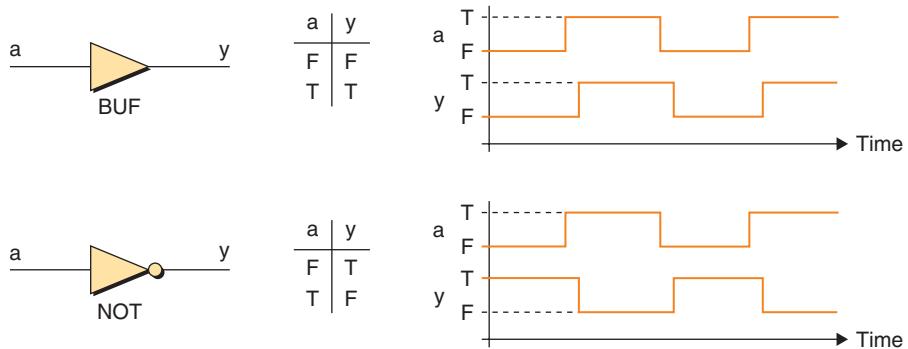


FIGURE 5.3
BUF and NOT functions.

false and true (of course, these labels are the reason why “truth tables” are so-named). Using the abstract terms false and true is preferable because assignments such as open and closed, on and off, up and down, and so forth may imply a particular implementation.

BUF AND NOT FUNCTIONS

The simplest of all the logic functions are known as BUF and NOT. The symbols, truth tables, and waveform diagrams for these functions are presented in Figure 5.3. The F and T values in the truth tables are shorthand for FALSE and TRUE, respectively.

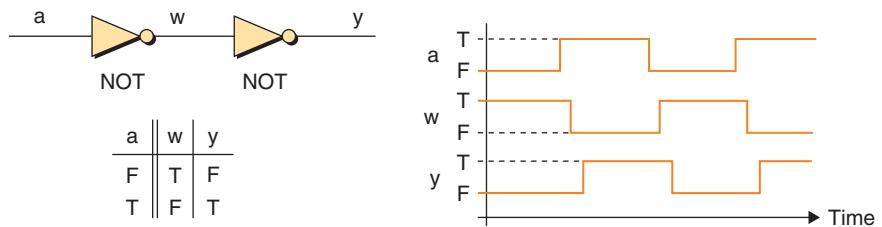
The output of the BUF function has the same value as the input to the function; if the input is FALSE the output is FALSE, and if the input is TRUE the output is TRUE. By comparison, the small circle, or *bubble*,⁴ on the output of the NOT symbol indicates an inverting function; if the input is FALSE the output is TRUE, and if the input is TRUE the output is FALSE.⁵

Actually, it can be a little tricky to wrap one’s brain around the concept of a NOT function the first time you see it,⁶ so let’s briefly regress to the world of our switch examples. Suppose we have a single light switch on the wall and we tell a user that when the switch is flicked UP the light will turn ON, and when the switch is returned to its DOWN position the light will turn OFF. Assume that the user plays with the switch and determines that we’ve been telling the truth. This would correspond to a BUF function.

⁴Some engineers use the term *bubble*, others say *bobble*, and the rest try to avoid mentioning it at all.

⁵A commonly used alternative name for a NOT function is INV (short for inverter).

⁶This function will make more sense in *Chapter 6: Using Transistors to Build Logic Gates*, when we implement it using transistors.

**FIGURE 5.4**

Two NOT functions connected in series.

Now suppose that we introduce the user to another switch. Once again we explain that the switch being UP corresponds to an ON state, while the switch being DOWN corresponds to an OFF condition. Let's assume, however, that we've deliberately wired (or mounted) the switch the "wrong way round" such that flicking it UP (ON) will turn the light OFF and pressing it DOWN (OFF) will turn the light ON. In this case, we've created the physical equivalent of a NOT function.

Finally, as a reminder that these abstract functions will eventually have physical realizations, the waveforms show delays between transitions on the inputs and corresponding responses at the outputs. The actual values of these delays depend on the technology used to implement the functions, but it is important to note that in any physical implementation there will always be some element of delay.

“CONNECT THE NOTS”

I was wracking my brain for a cunning title for this tasty tidbit of a topic, when my wife (Gina the Gorgeous) jokingly suggested “*connect the NOTs*” as a play on the kids’ game “*connect the dots*” (she will be jolly surprised when she eventually sees this in print). But we digress ... Consider the effect of connecting two NOT functions in *series* (one after the other) as shown in Figure. 5.4.

The first NOT gate inverts the value from the input, and the second NOT gate inverts it back again. Thus, the two negations cancel each other out (sort of like “*two wrongs do make a right*”). The end result is equivalent to that of a BUF function, except that each NOT contributes an element of delay.

AND, OR, AND XOR FUNCTIONS

Three slightly more complex functions are known as AND, OR, and XOR (Figure. 5.5).⁷ The AND and OR representations shown here are the abstract

⁷In conversation, the term XOR is pronounced “X-OR”; that is, spelling the letter ‘X’ followed by “OR” to rhyme with “door.”

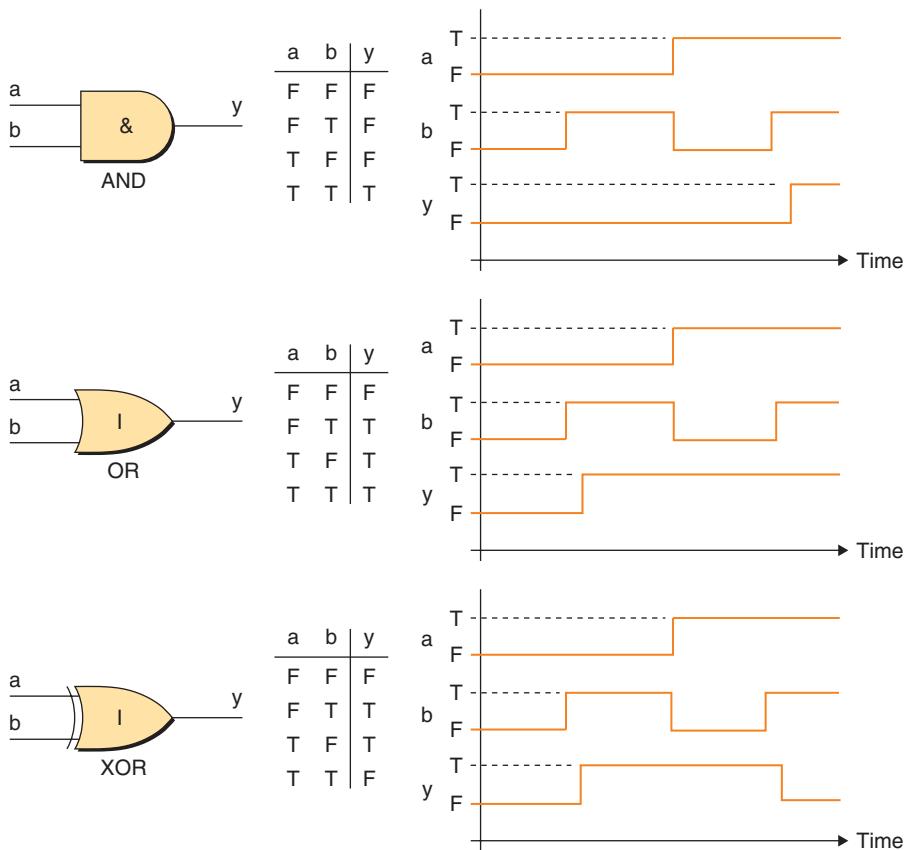


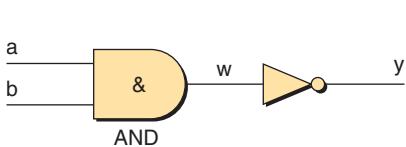
FIGURE 5.5
AND, OR, and XOR
functions.

equivalents of our original switch examples. In the case of the AND, the output is only TRUE if both a and b are TRUE; in the case of the OR, the output is TRUE if either a or b are TRUE. In fact, the OR should more properly be called an *inclusive-OR*, because the TRUE output cases include the case when both inputs are TRUE. Contrast this with the *exclusive-OR*, or XOR, where the TRUE output cases *exclude* the case when both inputs are TRUE.

NAND, NOR, AND XNOR FUNCTIONS

Now consider the effect of appending a NOT function to the output of the AND function as illustrated in Figure 5.6.

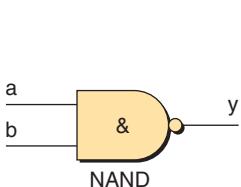
This combination of functions occurs frequently in designs. Similarly, the outputs of the OR and XOR functions are often inverted with NOT functions. This leads to three more primitive functions called NAND (NOT-AND), NOR



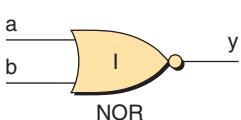
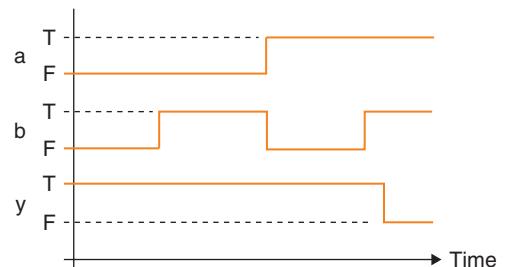
a	b	w	y
F	F	F	T
F	T	F	T
T	F	F	T
T	T	T	F

FIGURE 5.6

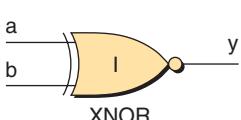
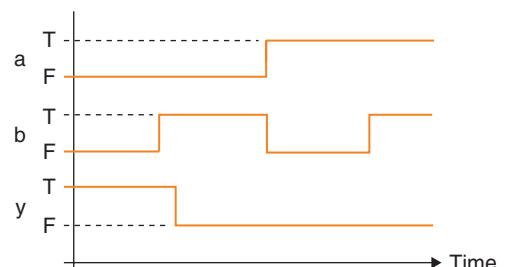
AND function followed by a NOT function.



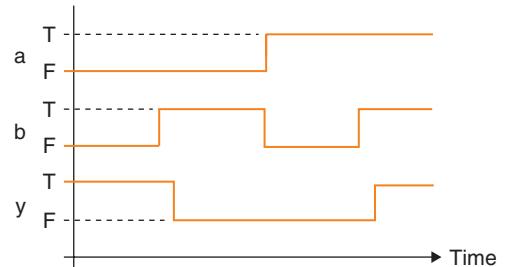
a	b	y
F	F	T
F	T	T
T	F	T
T	T	F



a	b	y
F	F	T
F	T	F
T	F	F
T	T	F



a	b	y
F	F	T
F	T	F
T	F	F
T	T	T

**FIGURE 5.7**

NAND, NOR, and XNOR functions.

(NOT-OR),⁸ and NXOR (NOT-XOR) as illustrated in Figure 5.7. In practice, the NXOR is almost always referred to as an XNOR (exclusive-NOR).⁹

The bubbles on the outputs of the NAND, NOR, and XNOR symbols indicate that these are inverting functions. One way to visualize this is that the symbol

⁸In conversation, the terms *NAND* and *NOR* are pronounced as single words to rhyme with “band” and “door,” respectively.

⁹In conversation, the term *XNOR* is pronounced “X-NOR”; that is, spelling the letter “X” followed by “NOR” to rhyme with “door.”

for the NOT function has been forced back into the preceding symbol until only its bobble remains visible.

If the XOR and XNOR functions seem to be a little esoteric, consider a real-world example from your home, such as two light switches mounted at opposite ends of a hallway controlling the same light. If both of the switches are UP or DOWN the light will be ON; for any other combination the light will be OFF. Constructing a truth table reveals a classic example of an XNOR function.

Of course, if we appended a NOT function to the output of a NAND, we'd end up back with our original AND function again. Similarly, appending a NOT to a NOR or an XNOR will result in an OR and XOR, respectively.

NOT A LOT

And that's about it. In reality there are only eight simple functions (BUF, NOT, AND, NAND, OR, NOR, XOR, and XNOR) from which everything else is constructed. In fact, some might argue that there are only seven core functions because you can construct a BUF out of two NOTs, as was discussed earlier.

Actually, if you want to go down this path, you can construct all of the above functions using one or more NAND gates (or one or more NOR gates). For example, if you connect the two inputs of a NAND gate together, you end up with a NOT as shown in Figure 5.8 (you can achieve the same effect by connecting the two inputs of a NOR gate together).

As the a and b inputs to the NAND function are connected together, we know that they have to be carrying identical values, so we end up showing only two rows in the truth table. We also know that if we invert the output from a NAND, we end up with an AND. So we could append a NAND configured as a NOT to the output of another NAND to generate an AND (Figure 5.9).

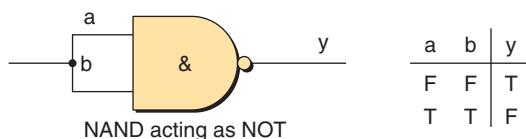
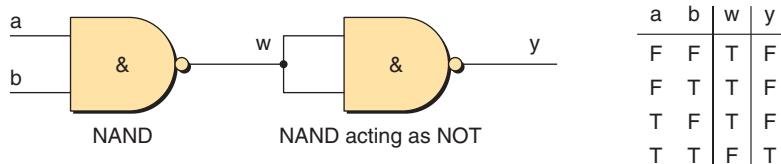


FIGURE 5.8

Forming a NOT from a NAND.

FIGURE 5.9

Forming an AND from two NANDs.



In *Chapter 9: Boolean Algebra*, we'll discover how to transform functions formed from ANDs into equivalent functions formed from ORs and vice versa. Coupled with what we've just seen here, this would allow us to build pretty much anything we desired out of a bunch of 2-input NAND (or NOR) functions.

FUNCTIONS VERSUS GATES

Simple functions such as BUF, NOT, AND, NAND, OR, NOR, XOR, and XNOR are often known as *primitive gates*, *primitives*, *logic gates*, or simply *gates*.¹⁰ Strictly speaking, the term *logic function* implies an abstract mathematical relationship, while *logic gate* implies an underlying physical implementation. In practice, however, these terms are often used interchangeably.

More complex functions can be constructed by combining primitive gates in different ways. A complete design—say a computer—employs a great many gates connected together to achieve the required result. When the time arrives to translate the abstract representation into a particular physical implementation, the logic symbols are converted into appropriate equivalents such as switches, transistors, or pneumatic valves. Similarly, the FALSE and TRUE logic values are mapped into appropriate equivalents such as switch positions, voltage levels, or air pressures.

The majority of designs end up being translated into a single technology. However, one of the advantages of abstract representations is that they allow designers to implement different portions of a single design in dissimilar technologies with relative ease. Throughout the remainder of this book we will be concentrating on electronic implementations.

¹⁰The reasoning behind using the term *gate* is that these functions serve to control electronic signals in much the same way that farmyard gates can be used to control animals.

CHAPTER 6

Using Transistors to Build Logic Gates

NMOS, PMOS, AND CMOS

There are several different families of transistors available to designers and, although the actual implementations vary, each can be used to construct primitive logic gates. This book concentrates on the enhancement-mode *Metal-Oxide Semiconductor Field-Effect Transistors* (MOSFETs) introduced in *Chapter 4: Semiconductors (Diodes and Transistors)*. There are two reasons for this. First, their symbols, construction, and operation are easier to understand than are their *Bipolar Junction Transistor* (BJT) cousins. Second, enhancement-mode MOSFETs are the most widely used transistors in the world, finding application in both analog and digital circuits, especially in today's digital integrated circuits, the largest of which may literally contain billions of these little scamps.

57

Logic gates can be created using only NMOS (n-channel) or only PMOS (p-channel) MOSFET transistors; however, a popular implementation called *Complementary Metal-Oxide Semiconductor* (CMOS)¹ makes use of both NMOS and PMOS transistors connected in a complementary manner.

CMOS gates operate from two voltage levels. As we discussed in *Chapter 4*, these are usually given the labels V_{DD} and V_{SS} . To a large extent, the actual values of V_{DD} and V_{SS} are irrelevant as long as V_{DD} is sufficiently more positive than V_{SS} ; purely for the purpose of these discussions (and to make you feel happier), however, we will assume that V_{DD} is +5V and V_{SS} is 0V.

USING 0s AND 1s INSTEAD OF Fs AND Ts

There are two conventions known as *positive logic* and *negative logic*.² Under the positive logic convention used throughout this book, the more positive V_{DD} is

¹In conversation, the term CMOS is pronounced "C-MOS"; that is, spelling the letter "C" followed by "MOS" to rhyme with "boss."

²These conventions are discussed in excruciating detail in *Appendix B: Positive Logic Versus Negative Logic*.

assigned the value of logic 1, and the more negative V_{SS} is assigned the value of logic 0.

In *Chapter 5: Primitive Logic Functions* it was noted that truth table assignments can be specified using the abstract values FALSE and TRUE. However, for reasons that are more fully examined in *Appendix B: Positive Logic Versus Negative Logic*, electronics designers usually represent FALSE (F) and TRUE (T) as 0 and 1, respectively.

The reason for this is that digital functions can be considered to represent either logical or arithmetic operations. Often, the same bunch of gates may be used to perform different operations at different times. Not surprisingly, therefore, it is preferable to employ a single consistent nomenclature to cover both cases, and it is easier to view logical operations in terms of 0s and 1s than it is to view arithmetic operations in terms of Fs and Ts.

NOT AND BUF GATES

The simplest logic function to implement in CMOS is a NOT gate (Figure 6.1). The small circle, or bobble, on the control input of transistor Tr_1 , indicates a PMOS transistor. The bobble is used to indicate that this transistor has an *active-low*³ control, which means that a logic 0 applied to the control input turns the transistor ON and a logic 1 turns it OFF.

By comparison, the lack of a bobble on the control input of transistor Tr_2 indicates an NMOS transistor. The lack of a bobble says that this transistor has an *active-high*⁴ control, which means that a logic 1 applied to the control input turns the transistor ON and a logic 0 turns it OFF.

Now, assume that we have a piece of wire and we attach one end to input a ; we can connect the other end to V_{SS} (logic 0) or V_{DD} (logic 1) as we wish. When we

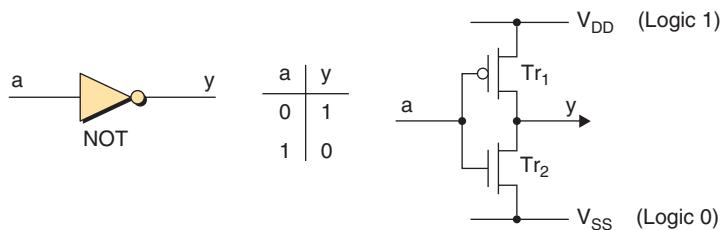


FIGURE 6.1
CMOS implementation of a NOT gate.

³The “low” comes from the fact that, under the commonly used positive-logic system, logic 0 is more negative (conceptually “lower”) than logic 1.

⁴The “high” comes from the fact that, under the commonly used positive-logic system, logic 1 is more positive (conceptually “higher”) than logic 0.

connect our wire to V_{SS} (thereby applying a logic 0 to input a), transistor Tr_1 is turned ON, transistor Tr_2 is turned OFF, and output y is connected to V_{DD} (logic 1) via Tr_1 . Similarly, when we connect our wire to V_{DD} (thereby applying a logic 1 to input a), transistor Tr_1 is turned OFF, transistor Tr_2 is turned ON, and output y is connected to V_{SS} (logic 0) via Tr_2 .

Don't worry if all this seems a bit confusing at first. The main points to remember are that a logic 0 applied to its control input turns the PMOS transistor ON and the NMOS transistor OFF, while a logic 1 turns the PMOS transistor OFF and the NMOS transistor ON. It may help to visualize the NOT gate's operation in terms of switches rather than transistors (Figure 6.2).

Surprisingly, a noninverting BUF gate is more complex than an inverting NOT gate. This is due to the fact that a BUF gate is constructed from two NOT gates connected in *series* (one after the other), which means that it requires four transistors (Figure 6.3).

The first NOT gate is formed from transistors Tr_1 and Tr_2 , while the second is formed from transistors Tr_3 and Tr_4 . In this case, a logic 0 applied to input a is inverted to a logic 1 on internal signal w, which is subsequently inverted back again to a logic 0 on output y. Similarly, a logic 1 on a is inverted to a logic 0 on w, which is then inverted back again to a logic 1 on y.

Around this stage, it is not unreasonable to question the need for BUF gates in the first place—after all, their logical function could be achieved using a simple

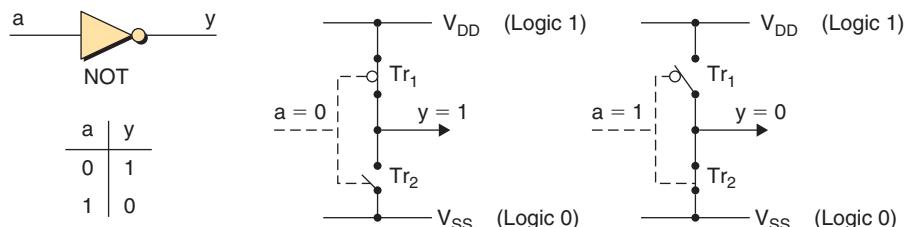


FIGURE 6.2
NOT gate's operation
represented using
switches.

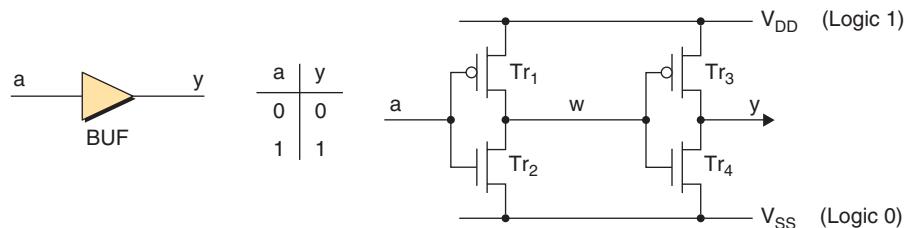


FIGURE 6.3
CMOS implementation
of a BUF gate.

piece of wire. But there's method to our madness, because BUF gates may actually be used for a number of purposes: for example, to isolate signals, to provide increased drive capability, or to add an element of delay.

NAND AND AND GATES

The implementations of the NOT and BUF gates shown above illustrate an important point, which is that it is generally easier to implement an inverting function than its noninverting equivalent. In the same way that a NOT is easier to implement than a BUF, a NAND is easier to implement than an AND, and a NOR is easier to implement than an OR. More significantly, inverting functions typically require fewer transistors and operate faster than their noninverting counterparts, which can obviously be an important design consideration. Consider a 2-input NAND gate, which requires only four transistors (Figure 6.4).⁵

When both a and b are presented with logic 1s, transistors Tr_1 and Tr_2 are turned OFF, transistors Tr_3 and Tr_4 are turned ON, and output y is connected to logic 0 via Tr_3 and Tr_4 . Any other combination of inputs results in one or both of Tr_3 and Tr_4 being turned OFF, one or both of Tr_1 and Tr_2 being turned ON, and output y being connected to logic 1 via Tr_1 and/or Tr_2 . Once again, it may help to visualize the gate's operation in terms of switches rather than transistors (Figure 6.5).

Now consider an AND gate. This is formed by inverting the output of a NAND (see Figure 6.4) with a NOT (see Figure 6.1), which means that a 2-input AND requires six transistors (Figure 6.6).⁶

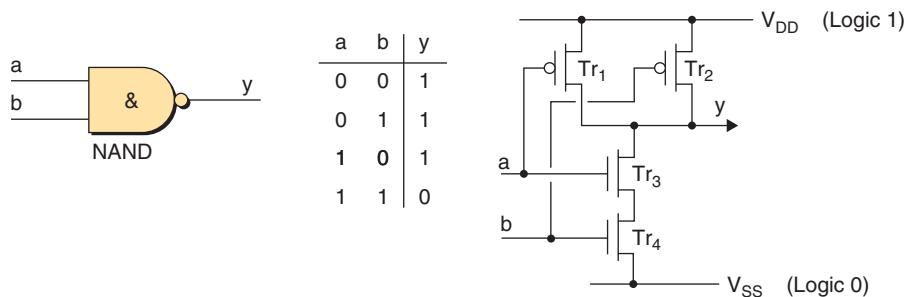
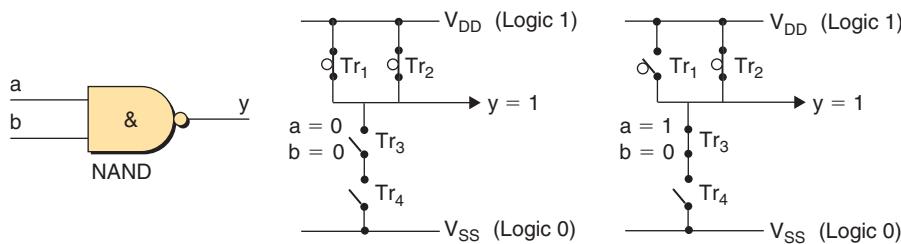


FIGURE 6.4
CMOS implementation
of a 2-input NAND gate.

⁵A 3-input version could be constructed by adding an additional PMOS transistor in parallel with Tr_1 and Tr_2 , and an additional NMOS transistor in series with Tr_3 and Tr_4 .

⁶Remember that electronics designers are cunning little rascals with lots of tricks up their sleeves. In fact, it's possible to create an AND gate using only one transistor and a resistor (see the discussions on *Pass-Transistor Logic* later in this chapter).



a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

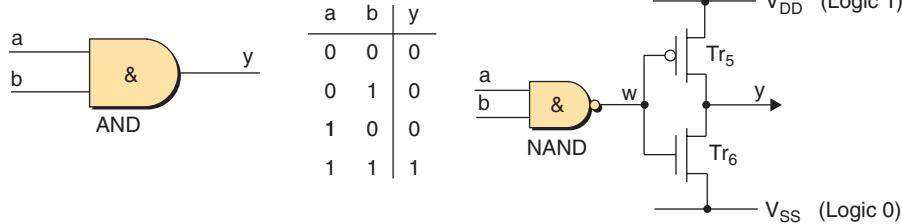


FIGURE 6.5
NAND gate's operation represented using switches.

FIGURE 6.6
CMOS implementation of a 2-input AND gate.

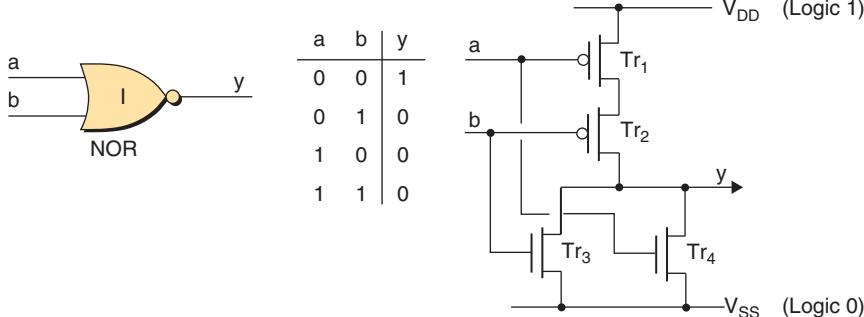
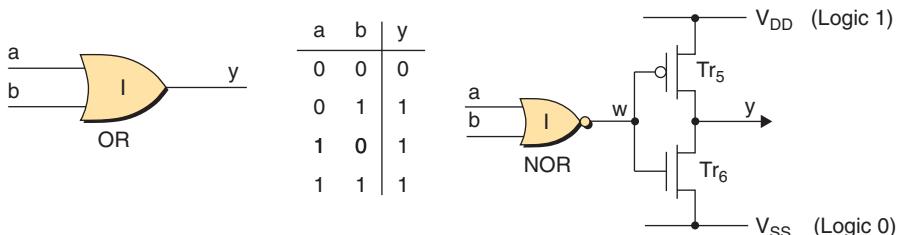


FIGURE 6.7
CMOS implementation of a 2-input NOR gate.

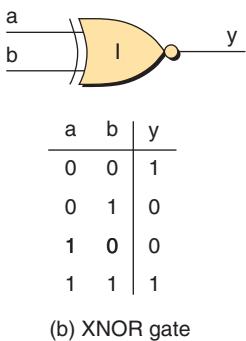
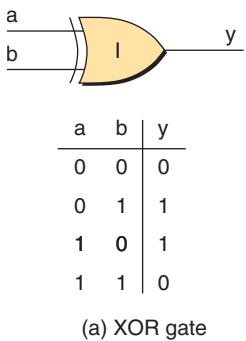
NOR AND OR GATES

A similar story occurs in the case of NOR gates and OR gates. First, consider a 2-input NOR, which requires only four transistors (Figure 6.7).⁷

⁷A 3-input version could be constructed by adding an additional PMOS transistor in series with Tr₁ and Tr₂, and an additional NMOS transistor in parallel with Tr₃ and Tr₄.

**FIGURE 6.8**

CMOS implementation of a 2-input OR gate.

**FIGURE 6.9**

XOR and XNOR gate symbols and truth tables.

When both a and b are set to logic 0, transistors Tr_3 and Tr_4 are turned OFF, transistors Tr_1 and Tr_2 are turned ON, and output y is connected to logic 1 via Tr_1 and Tr_2 . Any other combination of inputs results in one or both of Tr_1 and Tr_2 being turned OFF, one or both of Tr_3 and Tr_4 being turned ON, and output y being connected to logic 0 via Tr_3 and/or Tr_4 . (Refer back to Figure 6.5, and then try sketching out your own diagram representing this NOR gate's operation in terms of switches rather than transistors.)

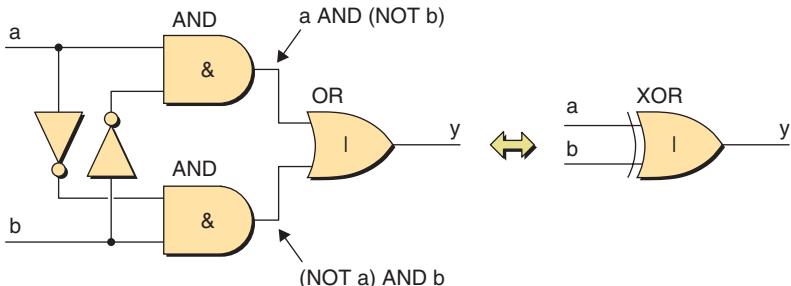
Once again, an OR gate is formed by inverting the output of a NOR with a NOT, which means that a 2-input OR requires six transistors (Figure 6.8).

XNOR AND XOR GATES

The concepts of NAND, AND, NOR, and OR are relatively easy to understand because they map onto the way we think in everyday life. For example, a textual equivalent of a NOR could be: *If it's windy or if it's raining then I'm not going out.*

By comparison, the concepts of XOR and XNOR can be a little harder to grasp because we don't usually consider things in these terms. For example, a textual equivalent of an XOR could be: *If it is windy and it's not raining, or if it's not windy and it is raining, then I will go out.* Although this does make sense in a strange sort of way, we don't often find ourselves making decisions in this manner. However, these gates are full of surprises and they find many uses (see Appendix D: Gray Codes, for example).

Consider the XOR and XNOR symbols and truth tables shown in Figure 6.9. From our earlier discussions in *Chapter 5: Primitive Logic Functions*, we know that the output from the XOR will be logic 1 if inputs a and b are presented with complementary values (that is, if a is 0 and b is 1, or vice versa). However, if inputs a and b are presented with the same values (both 0 or both 1), then the output from the XOR will be logic 0 [Figure 6.9(a)]. And, of course, the output from an XNOR is the negated version of the output from an XOR [Figure 6.9(b)].

**FIGURE 6.10**

Implementing a 2-input XOR from NOT, AND, and OR gates.

In the case of the XOR, we could state its function verbally as: “*Either a or b, but not both.*” Now, there are several ways in which we can realize an XOR; for example, we can easily implement this functionality using a combination of NOT, AND, and OR gates as illustrated in Figure 6.10.⁸

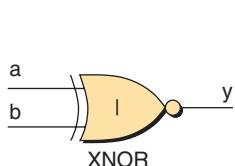
Take a few moments to work this out in your mind. The output from the upper left-hand AND gate will only ever be 1 if a is 1 and b is 0; the output from the lower left-hand AND gate will only ever be 1 if a is 0 and b is 1; and the output from the OR gate will only be 1 if at least one of the AND gates is generating a 1.

Now, in order to implement an XNOR gate, we could use a NOT gate to invert the output from the OR gate shown in Figure 6.10. Why would this *not* be a good idea? Well, if you cast your mind back to Figure 6.8, you will recall that an OR gate (with six transistors) is actually formed from the combination of a NOR gate (with four transistors) and a NOT gate (with two transistors). Thus, if we were to add another NOT gate to the output of the OR in Figure 6.10, we’d end up with a NOR gate formed from eight transistors! Thus, it would be much better to simply replace our OR gate (with six transistors) with a primitive NOR gate (with only four transistors).

XNOR AND XOR GATES: PASS-TRANSISTOR IMPLEMENTATIONS

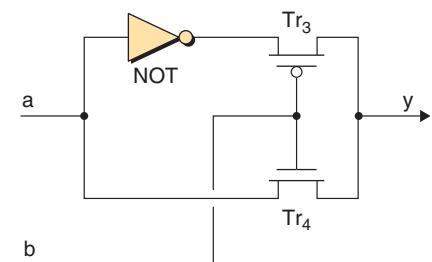
The 2-input XOR shown in Figure 6.10 requires 22 transistors (two each for the two NOT gates, six each for the two AND gates, and six more for the OR gate).

⁸As we can see from Figure 6.10, there are no such beasts as XOR and XNOR primitive gates, per se; instead, these functions are constructed from collections of other primitive gates. Thus, unlike AND, NAND, OR, and NOR gates, there are no such beasts as XNOR or XOR primitives with more than two inputs. However, equivalent functions with more than two inputs can be formed by connecting a number of 2-input XOR or XNOR functions together.

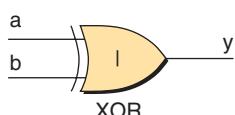
**FIGURE 6.11**

Pass-transistor implementation of a 2-input XNOR.

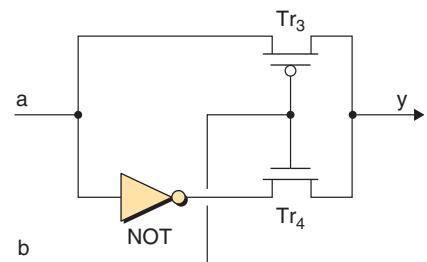
a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

**FIGURE 6.12**

Pass-transistor implementation of a 2-input XOR gate.



a	b	y
0	0	0
0	1	1
1	0	1
1	1	0



And, as we discussed in the previous topic, we could implement a 20-transistor XNOR function by replacing the 6-transistor OR gate with a 4-transistor NOR gate.

Now, just to keep the old mental juices flowing, consider how a 2-input XNOR can be implemented using only four transistors, as illustrated in Figure 6.11.

The NOT gate would be constructed in the standard way using two transistors as described above, but the XNOR differs from the previous gates in the way that transistors Tr_3 and Tr_4 are utilized. First, consider what happens when input b is presented with a logic 0. In this case, transistor Tr_4 is turned OFF, transistor Tr_3 is turned ON, and output y is connected to the output of the NOT gate via Tr_3 . Thus, when input b is logic 0, output y is the *inverse* of the value on input a . Now consider what happens when input b is presented with a logic 1. In this case, transistor Tr_3 is turned OFF, transistor Tr_4 is turned ON, and output y is connected to input a via Tr_4 . Thus, when input b is logic 1, output y has the *same* value as input a . Therefore, the end result of all these machinations is a function that satisfies the requirements of the XNOR truth table.

The interesting point is that, unlike the other complementary gates, it is not necessary to use a NOT gate to invert the output of this XNOR to form an XOR (although we could if we wanted to, of course). A little judicious rearranging of the components results in a 2-input XOR that also requires only four transistors (Figure 6.12).

First, consider what happens when input b is presented with a logic 0. In this case, transistor Tr_4 is turned OFF, transistor Tr_3 is turned ON, and output y is connected to input a via Tr_3 . Thus, when input b is logic 0, output y has the *same* value as input a. Now consider what happens when input b is presented with a logic 1. In this case, transistor Tr_3 is turned OFF, transistor Tr_4 is turned ON, and output y is connected to the output of the NOT gate via Tr_4 . Thus, when input b is logic 1, output y is the *inverse* of the value on input a. Thus, the end result is a function that satisfies the requirements of the XOR truth table.

PASS-TRANSISTOR LOGIC

In the BUF, NOT, AND, NAND, OR, and NOR gates described earlier (and also the XOR and XNOR gates formed from NOT, AND, and OR, gates, etc.), the input signals and internal data signals are used only to drive *control* (gate) terminals on the transistors. By comparison, in the case of the XOR and XNOR gates presented in the previous topic, transistors TR_3 and TR_4 are connected in such a way that input and internal data signals pass between their *data* (source and drain) terminals. This technique, which is known as *pass-transistor logic*, can be attractive in that it minimizes the number of transistors required to implement a function. However, pass-transistor logic is not necessarily the best approach because strange and unexpected effects can ensue if you're not careful and you don't know what you're doing.

An alternative solution for an XOR is to invert the output of our pass-transistor XNOR with a regular NOT gate. Similarly, an XNOR can be constructed by inverting the output of our pass-transistor XOR with a NOT. Although these new implementations each now require six transistors rather than four, they are more robust because the NOT gates buffer the outputs and provide a higher drive capability. And, as we previously discussed, XOR and XNOR functions can be constructed from combinations of the other primitive gates. This increases the transistor count still further but, once again, it results in more robust solutions.

Having said all this, pass-transistor logic can be applicable in certain situations for designers who do know what they're doing. In the discussions above, it was noted that it is possible to implement an AND function using a single transistor and a resistor. Similarly, it's possible to implement an OR function using a single transistor and a resistor, and to implement XOR and XNOR functions using only two transistors and a resistor. If you're feeling brave, try to work out how to achieve these minimal implementations for yourself (solutions are given in Appendix F: Pass-Transistor Logic).

This page intentionally left blank

CHAPTER 7

Alternative Number Systems

FINGERS, TOES, AND PEBBLES

The first tools used as aids to calculation were almost certainly man's own fingers, and it is not simply a coincidence that the word "digit" is used to refer to a finger (or toe) as well as a numerical quantity. As the need to represent larger numbers grew, early man employed readily available materials for the purpose. Small stones or pebbles could be used to represent larger numbers than fingers and toes and had the added advantage of being able to easily store intermediate results for later use. Thus, it is also no coincidence that the word "calculate" is derived from the Latin word for pebble.

67

BONES WITH NOTCHES

The oldest objects known to represent numbers are bones with notches carved into them (Figure 7.1). These bones, which were discovered in Western Europe, date from the Aurignacian period 20,000 to 30,000 years ago and correspond to the first appearance of Cro-Magnon man.¹ Of special interest is a wolf's jawbone more than 20,000 years old with 55 notches in groups of five, which was discovered in Czechoslovakia in 1937. This is the first evidence of the *tally system*, which is still used occasionally to the present day and could therefore qualify as one of the most enduring of human inventions.

Also of interest is a piece of bone dating from around 8500 BC, which was discovered in Africa and which appears to have notches representing the prime numbers 11, 13, 17, and 19. Prime numbers are those that are wholly divisible only by the number one and themselves, so it is not surprising that early man would have attributed a special significance to them. What is surprising

¹The term *Cro-Magnon* comes from caves of the same name in Southern France, in which the first skeletons of this race were discovered in 1868.



FIGURE 7.1
Notches on bones.
(Courtesy Clive "Max" Maxfield and Alvin Brown)

is that someone of that era had the mathematical sophistication to recognize this quite advanced concept and took the trouble to write it down—not to mention that prime numbers would appear to have had little relevance to the everyday problems of the time, such as gathering food and staying alive.

TALLY STICKS: THE HIDDEN DANGERS

The practice of making marks on things or cutting notches into them to represent numbers has survived to the present day, especially among schoolchildren making tally marks on their desks to signify the days of their captivity.

In the not-so-distant past, storekeepers (who often could not read or write) used a similar technique to keep track of their customer's debts. For example, a baker might make cuts across a stick of wood equal to the number of loaves in the shopper's basket. This stick was then split lengthwise, with the baker and the customer each keeping half, so that both could remember how many loaves were owed for and neither of them could cheat.

Similarly, the British government used wooden tally sticks until the early 1780s. These sticks had notches cut into them to record financial transactions and to act as receipts. Over the course of time, these tally sticks were replaced by paper records, which left the cellars of the Houses of Parliament full to the brim with pieces of old wood. Rising to the challenge with the inertia common to governments around the world, Parliament dithered until 1834 before finally getting around to ordering the destruction of the tally sticks.

There was some discussion about donating the sticks to the poor as firewood; but wiser heads prevailed, pointing out that the sticks actually represented "top secret" government transactions. The fact that the majority of the poor couldn't read or write and often couldn't count was obviously of no great significance, and it was finally decreed that the sticks should be burned in the courtyard of the Houses of Parliament. However, fate is usually more than willing to enter the stage with a pointed jape—gusting winds caused the fire to break out of control and burn the House of Commons to the ground (although they did manage to save the foundations)!

THE ABACUS

The first actual calculating mechanism known to us is the *abacus*, which is thought to have been invented by the Babylonians sometime between 1000 and 500 BC (although some pundits are of the opinion that it was actually invented by the Chinese).

The word *abacus* comes to us by way of Latin as a mutation of the Greek word *abax*. In turn, the Greeks may have adopted the Phoenician word *abak*, meaning “sand,” although some authorities lean toward the Hebrew word *abhaq*, meaning “dust.” Irrespective of the source, the original concept referred to a flat stone covered with sand (or dust) into which numeric symbols were drawn. The first abacus was almost certainly based on such a stone, with pebbles being placed on lines drawn in the sand.

Over time, the stone was replaced by a wooden frame supporting thin sticks, braided hair, or leather thongs, onto which clay beads or pebbles with holes were threaded. A variety of different types of abacus were developed, but the most popular became those based on the *bi-quinary system*, which utilizes a combination of two bases (base-2 and base-5) to represent decimal numbers. Although the abacus does not qualify as a mechanical calculator, it certainly stands proud as one of the first mechanical aids to calculation.

ROMAN NUMERALS

In the next topic we’re going to introduce the concept of *place-value number systems*. In order to understand why these are so efficacious, let’s first briefly consider the concept of Roman Numerals, in which I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000, and so forth.

Using this scheme, XXXV represents 35 (three tens and a five). One problem with this type of number system is that over time, as a civilization develops, it tends to become necessary to represent larger and larger quantities. This means that mathematicians either have to keep on inventing new symbols or start using lots and lots of their old ones. But the biggest disadvantage of this approach is that it’s painfully difficult to work with (try multiplying CLXXX by DDCV and it won’t take you long to discover what I mean).

Actually, it’s easy for us to rest on our laurels and smugly criticize ideas of the past with the benefit of hindsight (the one exact science). In fact, Roman numerals were used extensively in England until the middle of the 17th century, and are still used to some extent to this day; for example, the copyright

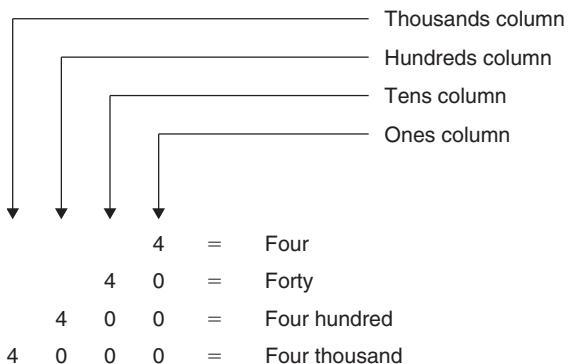


FIGURE 7.2
Decimal is a place-value number system.

notice on films and television programs often indicates the year in Roman numerals!

DECIMAL (BASE-10)

The commonly used *decimal numbering system* is based on 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The name decimal comes from the Latin *decem*, meaning “ten.” The symbols used to represent these digits arrived in Europe around the 13th century from the Arabs, who in turn borrowed

them from the Hindus (and never gave them back). As the decimal system is based on 10 digits, it is said to be *base-10* or *radix-10*, where the term *radix* comes from the Latin word meaning “root.”

With the exception of specialist requirements such as computing, base-10 numbering systems have been adopted almost universally—this is almost certainly due to the fact that humans happen to have 10 fingers.² If mother nature had dictated six or eight fingers on each hand, for example, the outcome would most probably have been the common usage of base-12 or base-16 numbering systems, respectively (see also the discussions on the *Duo-Decimal* and *Hexadecimal* systems).

The decimal system is a *place-value system*, which means that the value of a particular digit depends both on the digit itself and on its position within the number (Figure 7.2).

Every column in a place-value number has a “weight” associated with it, and each digit is combined with its column’s weight to determine the final value of the number (Figure 7.3).

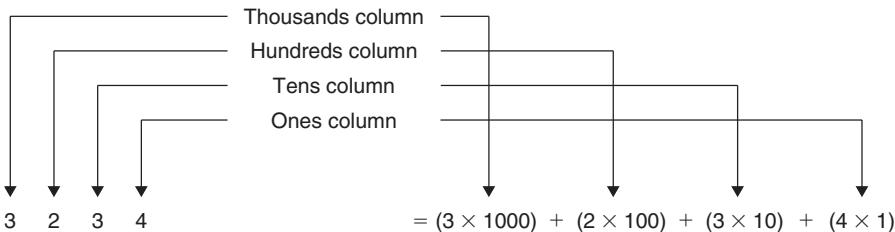


FIGURE 7.3
Combining digits with column weights in decimal.

²Including thumbs.

Counting in decimal commences at 0 and progresses up to 9, at which point all of the available digits have been used. Thus, the next count causes the first column to be reset to 0 and the second column to be incremented, resulting in 10. Similarly, when the count reaches 99, the next count causes the first column to be reset to zero and the second column to be incremented. But, as the second column already contains a 9, this causes it to be reset to 0 and the third column to be incremented resulting in 100 (Figure 7.4).

Although base-10 systems are anatomically convenient, they have few other advantages to recommend them. In fact, depending on your point of view, almost any other base (with the possible exception of nine) would be as good as, or better than, base-10, which is only wholly divisible by 2 and 5. For many arithmetic operations, the use of a base that is wholly divisible by many numbers, especially the smaller values, conveys certain advantages. An educated layman may well prefer a base-12 system on the basis that 12 is wholly divisible by 2, 3, 4, and 6. For their own esoteric purposes, some mathematicians would ideally prefer a system with a prime number as a base; for example, base-7 or base-11.

DUO-DECIMAL (BASE-12)

Number systems with bases other than 10 have sprouted up like weeds throughout history. Some cultures made use of *duo-decimal* (base-12) systems; instead of counting fingers they counted finger-joints. Each of our fingers has three joints (at least they do in my branch of the family), so if you use your thumb to point to the joints of the other fingers on the same hand, you can count *one-two-three* on the first finger, *four-five-six* on the next, and so on up to *twelve* on your little finger (Figure 7.5).

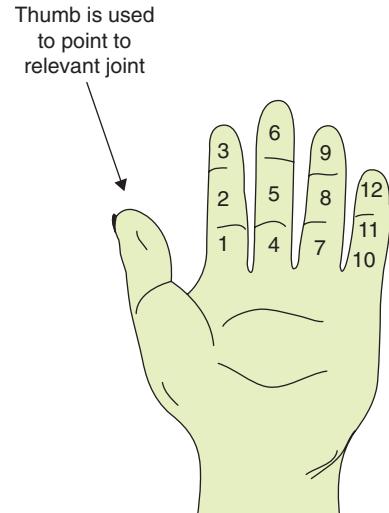
This system is particularly useful if one wishes to count up to 12 while still maintaining a free hand to throw a spear at someone whom, we may assume, is not a close friend.

This form of counting may explain why the ancient Sumerians, Assyrians, and Babylonians divided their days into 12 periods: six for day and six for night. The lengths of the periods were

0	10	20	100	etc.
1	11	21	101	
2	12	22	102	
.	.	.	.	
.	.	.	.	
.	.	.	.	
8	18	98	998	
9	19	99	999	

FIGURE 7.4
Counting in decimal.

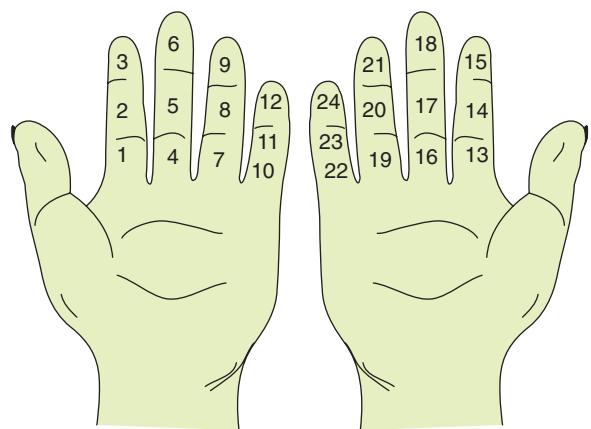
FIGURE 7.5
Using finger joints to count in duo-decimal.



11:00 PM	⇒	1:00 AM	≡	Hour of the Rat
1:00 AM	⇒	3:00 AM	≡	Hour of the Ox
3:00 AM	⇒	5:00 AM	≡	Hour of the Tiger
5:00 AM	⇒	7:00 AM	≡	Hour of the Hare
7:00 AM	⇒	9:00 AM	≡	Hour of the Dragon
9:00 AM	⇒	11:00 AM	≡	Hour of the Snake
11:00 AM	⇒	1:00 PM	≡	Hour of the Horse
1:00 PM	⇒	3:00 PM	≡	Hour of the Ram
3:00 PM	⇒	5:00 PM	≡	Hour of the Monkey
5:00 PM	⇒	7:00 PM	≡	Hour of the Cock
7:00 PM	⇒	9:00 PM	≡	Hour of the Dog
9:00 PM	⇒	11:00 PM	≡	Hour of the Boar

FIGURE 7.6

The Chinese 12-hour day.

**FIGURE 7.7**

Using finger joints to count to 24.

adjusted to the seasons (since the length of daylight compared to nighttime varies throughout the year), but were approximately equal to two of our hours. In fact, the Chinese use a form of this system to the present day (Figure 7.6).

If a similar finger-joint counting strategy is applied to both hands, the counter can represent values from 1 through 24 (Figure 7.7). This may explain why the ancient Egyptians divided their days into 24 periods, which is, in turn, why we have 24 hours in a day. Strangely enough, an Egyptian hour was only approximately equal to one of our hours. This was because the Egyptians liked things to be nice and tidy, so they decided to have 12 hours of daylight and 12 hours of nighttime. Unfortunately, as the amount of daylight varies throughout the year, they were obliged to adjust the lengths of their hours according to the seasons.

One of the methods used by the Egyptians to measure time was the water clock, or *clepsydra*,³ which consisted of a container of water with a small hole in the bottom through which the water escaped. Units of time were marked on the side of the container, and the length of the units corresponding to day and night could be adjusted by varying the distance between the markings or by modifying the shape of the container (by having the top wider than the bottom, for example).

In addition to their base-12 system, the Egyptians also experimented with a sort-of base-10 system. In this system, the numbers 1 through 9 were drawn using the appropriate number of vertical lines; 10 was represented by a circle; 100 was a

³The term *clepsydra* is derived from the Greek *klepto*, meaning “thief,” and *hydro*, meaning “water.” Thus, clepsydra literally means “water thief.”

coiled rope; 1000 a lotus blossom; 10,000 a pointing finger; 100,000 a tadpole; and 1,000,000 a picture of a man with his arms spread wide in amazement. (This is similar in concept to the Roman Numerals we discussed earlier.)

Thus, in order to represent a number like 2,327,685, they would have been obliged to use pictures of two amazed men, three tadpoles, two pointing fingers, seven lotus blossoms, six coiled ropes, eight circles, and five vertical lines. It requires very few attempts to divide tadpoles and lotus blossoms by pointing fingers and coiled ropes to appreciate why this scheme didn't exactly take the world by storm.

SEXAGESIMAL (BASE-60)

Previously we noted that, for many arithmetic operations, the use of a number system whose base is wholly divisible by many numbers—especially the smaller values—conveys certain advantages. And so we come to the Babylonians, who were famous for their astrological observations and calculations, and who used a *sexagesimal* (base-60) numbering system.

While there is no definite proof as to the origins of the sexagesimal base, it's possible that this was an extension of the finger-joint counting schemes discussed above (as illustrated in Figure 7.8).

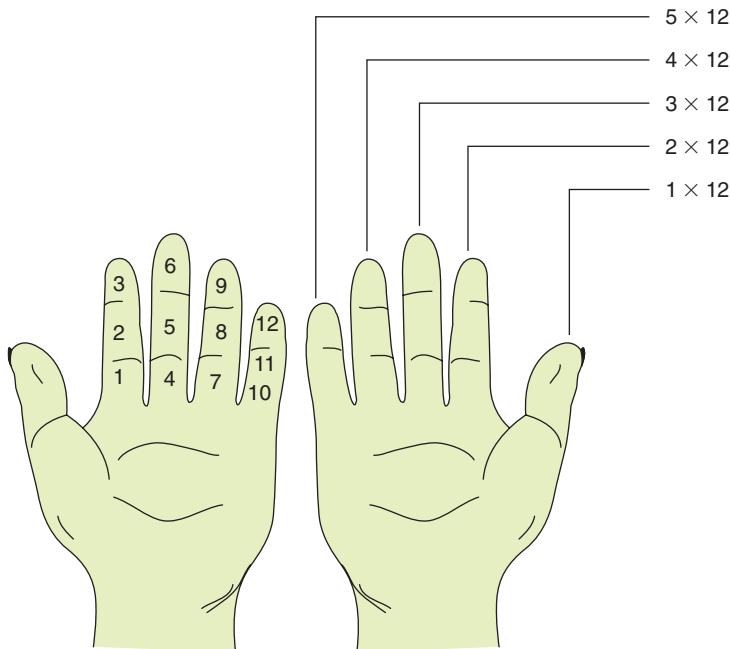


FIGURE 7.8

Using fingers and finger joints to count to 60.

In this scenario, the finger joints of the left hand are still used to represent the values 1 through 12; however, instead of continuing directly with the finger joints of the right hand, the thumb and fingers on the right hand are used to keep track of each count of 12. When all of the right hand digits are extended the count is 60 ($5 \times 12 = 60$).

Although 60 may appear to be a large value to have as a base, it does convey certain advantages. Sixty is the smallest number that can be wholly divided by each of the numbers 2 through 6; in fact, 60 can be wholly divided by 2, 3, 4, 5, 6, 10, 12, 15, 20, and 30. (Just to increase their fun, in addition to using base-60 the Babylonians also made use of 6 and 10 as subbases.)

The Babylonians' sexagesimal system, which first appeared around 1900 to 1800 BC, is also credited with being the first known place-value number system, in which the value of a particular digit depends on both the digit itself and its position within the number. This was an extremely important development, because—prior to place-value systems—people were obliged to use different symbols to represent different powers of a base. As was illustrated by the Egyptian and Roman systems we mentioned earlier, having unique symbols for 10, 100, 1000, and so forth makes even rudimentary calculations very difficult to perform.

And, before we move on, we should note that although the Babylonians' base-60 system may seem a tad unwieldy to us, one cannot help but feel that it was an improvement on the Sumerians who came before them. The reason I say this is that the Sumerians had three distinct counting systems to keep track of land, produce, and animals, and—on the basis that these were three different “things”—they used a completely different set of symbols for each system!

THE CONCEPTS OF ZERO AND NEGATIVE NUMBERS

Interestingly enough, the idea of numbers like one, two, and three developed a long time before the concept of zero. This was largely because the requirement for a number “zero” was less than obvious in the context of the calculations that early men and women were trying to perform.

For example, suppose that a young man’s father had instructed him to stroll up to the top field to count their herd of goats and, on arriving, the lad discovered the gate wide open and no goats to be seen.

First, the lad’s task on the counting front had effectively been done for him. Second, on returning to his aged parent, he probably wouldn’t feel the need to say: *“Oh revered one, I regret to inform you that the result of my calculations leads me*

“126904”	would be written as	“1269 4”
“102056”	would be written as	“1 2 56”
“160014”	would be written as	“16 14”

FIGURE 7.9

Representing zeros using spaces.

<i>Original Amount</i>	<i>– Fish Distributed</i>	<i>= Fish Remaining</i>
1Δ52 fish	– 45 fish to Gina	= 1ΔΔ7 fish
	– 2Δ fish to Max	= 987 fish
	– 4Δ7 fish to Joseph	= 58Δ fish
	– 176 fish to Drew	= 4Δ4 fish
	– 4Δ4 fish to Henry	= “No fish left”

FIGURE 7.10

Representing zeros using a placeholder symbol.

to believe that we are the proud possessors of zero goats.” Instead, he would be far more inclined to proclaim something along the lines of: “Father, some drongo left the gate open and all of our goats have wandered off into the sunset.”

As a result, in the original Babylonian system, for example, a zero was simply represented by a space; the decimal equivalent would be as shown in Figure 7.9.

It is easy to see how this can lead to a certain amount of confusion, especially when attempting to portray multiple zeros next to each other. The problems can only be exacerbated if one is using a sexagesimal system and writing on tablets of damp clay in a thunderstorm.

After more than 1500 years of potentially inaccurate calculations, the Babylonians finally began to use a special sign for zero. Some say that this concept, which first appeared around 300 BC, was one of the most significant inventions in the history of mathematics. However, the Babylonians only used the zero symbol as a placeholder to separate digits—they didn’t have the concept of zero as an actual value. If we were to use the “Δ” character to represent the Babylonian placeholder, the decimal equivalent to the clay tablet accounting records of the time would read something like that shown in Figure 7.10.

The last entry in the “Fish Remaining” column is particularly revealing. Because the Babylonian zero was only a placeholder and not a value, the accounting records had to say “No fish left” rather than “Δ fish.”

In fact, the use of zero as an actual value, along with the concept of negative numbers, first appeared in India around 600 AD. Although negative numbers appear reasonably obvious to us today, they were not well understood until modern times. As recently as the 18th century, the great Swiss mathematician Leonhard Euler (1707–1783; pronounced “oiler” in America) believed that negative numbers were greater than infinity, and it was common practice to ignore any negative results returned by equations on the assumption that they were meaningless!

VIGESIMAL (BASE-20)

The Mayans, Aztecs, and Celts developed *vigesimal* (*base-20*) systems by counting using both fingers and toes. The Eskimos of Greenland, the Tamanas of Venezuela, and the Ainu of northern Japan are three of the many other groups of people who also make use of vigesimal systems. For example, to say “fifty-three,” the Greenland Eskimos would use the expression “*Inup pinga-jugsane arkanek-pingasut*,” which translates as “*Of the third man, three on the first foot.*”⁴ This means that the first two men contribute 20 each (10 fingers and 10 toes), and the third man contributes 13 (10 fingers and 3 toes).

JOBs ABOUND FOR TIME-TRAVELERS

To this day, we bear the legacies of almost every number system our ancestors experimented with. From the duo-decimal systems we have 24 hours in a day (2×12), 12 inches in a foot, and special words such as *dozen* (meaning 12) and *gross* (meaning $12 \times 12 = 144$). Similarly, the Chinese have 12 hours in a day (each equal to two of our hours) and 24 seasons in a year (each approximately equal to two of our weeks).

From the sexagesimal system we have 60 seconds in a minute, 60 minutes in an hour, and 360 degrees in a circle, where 360 degrees is derived from the product of the Babylonian’s main base (60) and their subbase (6); that is, $60 \times 6 = 360$. And from the vigesimal systems we have special words like *score* (meaning 20), as in Lincoln’s famous Gettysburg Address, in which he proclaimed: “*Four score and seven years ago, our fathers brought forth on this continent a new nation ...*” This all serves to illustrate that number systems with bases other than 10 are not only possible, but positively abound throughout history.

Because we’re extremely familiar with using numbers, we tend to forget the tremendous amounts of mental effort that have been expended to raise us to our present level of understanding. In the days of yore when few people knew how to count, anyone who was capable of performing relatively rudimentary mathematical operations could easily achieve a position of power. For example, if you could predict an eclipse (especially one that actually came to pass) you were obviously someone to be reckoned with.

Similarly, if you were a warrior chieftain, it would be advantageous to know how many fighting men and women you had at your command, and the person who

⁴George Ifrah: *From One to Zero (A Universal History of Numbers)*.

could provide you with this information would obviously rank highly on your summer-solstice card list.⁵ So, should you ever be presented with the opportunity to travel back through time, you can bask in the glow of the knowledge that there are numerous job opportunities awaiting your arrival. But we digress ...

QUINARY (BASE FIVE)

One system that is relatively easy to understand is *quinary* (*base-5*), which uses the digits 0, 1, 2, 3, and 4. This system is particularly interesting in that a quinary finger-counting scheme is still in use today by merchants in the Indian state of Maharashtra, near Bombay.

As with any place-value system, each column in a quinary number has a weight associated with it, where the weights are derived from the base. Each digit is combined with its column's weight to determine the final value of the number (Figure 7.11).

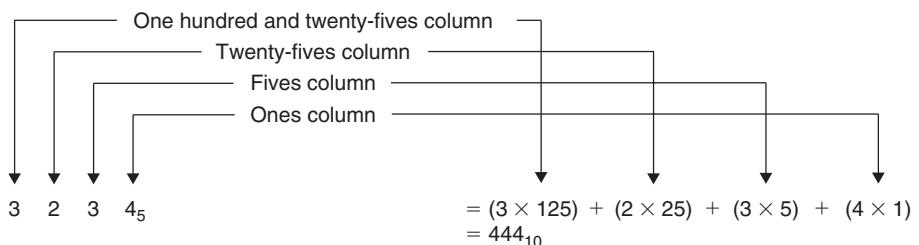


FIGURE 7.11
Combining digits with column weights in quinary.

When using systems with bases other than 10, subscripts are used to indicate the relevant base; for example, $3234_5 = 444_{10}$ ($3234_{\text{QUINARY}} = 444_{\text{DECIMAL}}$). By convention, any value without a subscript is assumed to be in decimal.

Counting in quinary commences at 0 and progresses up to 4_5 , at which point all the available digits have been used. Thus, the next count causes the first column to be reset to 0 and the second column to be incremented, resulting in 10_5 . Similarly, when the count reaches 44_5 , the next count causes the first column to be reset to zero and the second column to be incremented. But, as the second column already contains a 4, this causes it to be reset to 0 and the third column to be incremented resulting in 100_5 (Figure 7.12).

⁵You wouldn't have a Christmas card list, because the concept of Christmas cards wasn't invented until 1843.

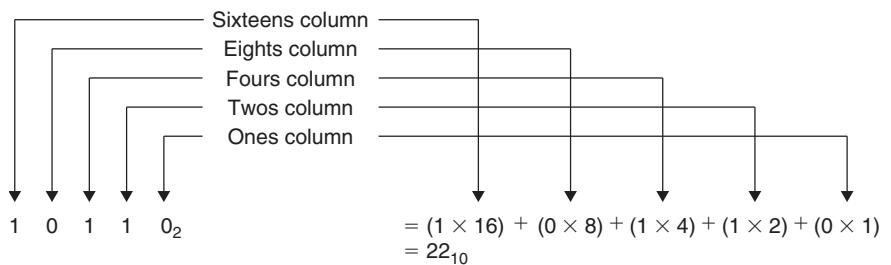
0_5 (0)	20_5 (10)	100_5 (25)	1000_5 (125)	etc.
1_5 (1)	21_5 (11)	101_5 (26)	1001_5 (126)	
2_5 (2)	22_5 (12)	102_5 (27)	1002_5 (127)	
3_5 (3)	.	.	.	
4_5 (4)	.	.	.	
10_5 (5)	.	.	.	
11_5 (6)	.	.	.	
12_5 (7)	42_5 (22)	442_5 (122)	4442_5 (622)	
13_5 (8)	43_5 (23)	443_5 (123)	4443_5 (623)	
14_5 (9)	44_5 (24)	444_5 (124)	4444_5 (624)	

FIGURE 7.12

Counting in quinary (values shown in parentheses are decimal equivalents).

BINARY (BASE-2)

Digital systems are constructed out of logic gates that can only represent two states; thus, computers are obliged to make use of a number system comprising only two digits. Base-2 number systems are called *binary* and use the digits 0 and 1. As usual, each column in a binary number has a weight derived from the base, and each digit is combined with its column's weight to determine the final value of the number (Figure 7.13). Once again, subscripts are used to indicate the relevant base; for example, $10110_2 = 22_{10}$ ($10110_{\text{BINARY}} = 22_{\text{DECIMAL}}$).

**FIGURE 7.13**

Combining digits with column weights in binary.

Sometime in the late 1940s, the American chemist-turned-topologist-turned-statistician John Wilder Tukey (1915–2000) realized that computers and the binary number system were destined to become important. In addition to coining the word *software*, Tukey decided that saying “binary digit” was a bit of a mouthful, so he started to look for an alternative. He considered a variety of

options like *binit* and *bigit*, but he eventually settled on *bit*, which is elegant in its simplicity and is used to this day.

Based on this, the binary value 10110_2 would be said to be 5 bits wide. Additionally, a group of 4 bits is known as a *nybble* (sometimes called a *nibble*), and a group of 8 bits is known as a *byte*. The idea that “*two nybbles make a byte*” is in the way of being an engineer’s idea of a joke, which shows that they do have a sense of humor (it’s just not a particularly sophisticated one).⁶

Counting in binary commences at 0 and rather quickly progresses up to 1_2 , at which point all the available digits have been used. Thus, the next count causes the first column to be reset to 0 and the second column to be incremented, resulting in 10_2 . Similarly, when the count reaches 11_2 , the next count causes the first column to be reset to zero and the second column to be incremented. But, as the second column already contains a 1, this causes it to be reset to 0 and the third column to be incremented resulting in 100_2 (Figure 7.14).

Although binary mathematics is fairly simple, humans tend to find it difficult at first because the numbers are inclined to be long and laborious to manipulate. For example, the binary value 11010011_2 is relatively difficult to conceptualize, while its decimal equivalent of 211 is comparatively easy. On the other hand, working in binary has its advantages. For example, if you can remember ...

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

... then you’ve just memorized the entire binary multiplication table!

⁶Continuing the theme, there have been sporadic attempts to construct terms for bit-groups of other sizes; for example, *tayste* or *crumb* for a 2-bit group; *playte* or *chawmp* for a 16-bit group; *dynner* or *gawble* for a 32-bit group; and *tayble* for a 64-bit group. But you can only take a joke so far, and using anything other than the standard terms *nybble* and *byte* is extremely rare. Having said this, the term *word* is commonly used as discussed in Chapter 15: *Memory ICs*.

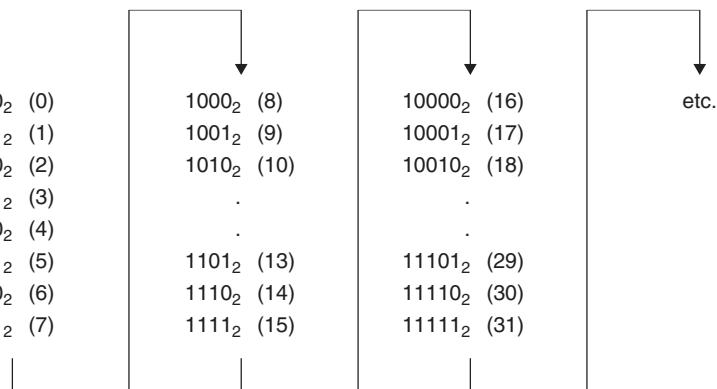


FIGURE 7.14
Counting in binary
(values shown in
(parentheses are
decimal equivalents).

OCTAL (BASE-8) AND HEXADECIMAL (BASE-16)

Any number system having a base that is a power of two (2, 4, 8, 16, 32, etc.) can be easily mapped into its binary equivalent, and vice versa. For this reason, electronics engineers typically make use of either the *octal* (base-8) or *hexadecimal* (base-16) systems.

As a base-8 system, octal requires eight individual symbols to represent all of its digits. This isn't a problem because we can simply use the symbols 0 through 7 that we know and love so well. In the case of the base-16 hexadecimal system, however, we require 16 individual symbols to represent all of the digits. This does pose something of a problem because there are only 10 Hindu-Arabic symbols available (0 through 9). One solution would be to create some new symbols, but some doubting Thomases (and Thomasinas) regard this as less than optimal because it would necessitate the modification of existing typewriters and computer keyboards. As an alternative, the first six letters of the alphabet are brought into play (Figure 7.15).

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

FIGURE 7.15

The 16 hexadecimal digits.

The rules for counting in octal and hexadecimal are the same as for any other place-value system—when all the digits in a column are exhausted, the next count sets that column to zero and increments the column to the left (Figure 7.16).

Although not strictly necessary, binary, octal, and hexadecimal numbers are often prefixed by leading zeros to pad them to whatever width is required. This padding is usually performed to give some indication as to the physical number of bits used to represent the various values within a computer.

Each octal digit can be directly mapped onto three binary digits, and each hexadecimal digit can be directly mapped onto four binary digits (Figure 7.17).

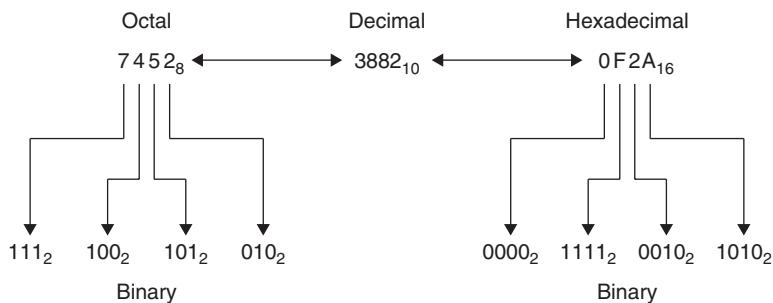
Similarly, it's easy to convert a binary number such as 000011100101010_2 into its hexadecimal equivalent. All we have to do is to split the binary value into 4-bit nibbles, and to then map each nibble to its corresponding hexadecimal digit.

In the original digital computers, data paths were often 9 bits, 12 bits, 18 bits, or 24 bits wide, which provides one reason for the original popularity of the

<i>Decimal</i>	<i>Binary</i>	<i>Octal</i>	<i>Hexadecimal</i>
0	00000000	000	00
1	00000001	001	01
2	00000010	002	02
3	00000011	003	03
4	00000100	004	04
5	00000101	005	05
6	00000110	006	06
7	00000111	007	07
8	00001000	010	08
9	00001001	011	09
10	00001010	012	0A
11	00001011	013	0B
12	00001100	014	0C
13	00001101	015	0D
14	00001110	016	0E
15	00001111	017	0F
16	00010000	020	10
17	00010001	021	11
18	00010010	022	12
:	:	:	:
etc.	etc.	etc.	etc.

FIGURE 7.16

Counting in octal and hexadecimal.

**FIGURE 7.17**

Mapping octal and hexadecimal to binary.

octal system. Due to the fact that each octal digit maps directly to three binary bits, these data-path values were easily represented in octal. More recently, digital computers have standardized on data-path widths that are integer multiples of 8 bits; for example, 8 bits, 16 bits, 32 bits, 64 bits, and so forth. Because each hexadecimal digit maps directly to four binary bits, these data-path values are more easily represented in hexadecimal. This may explain the decline in popularity of the octal system and the corresponding rise in popularity of the hexadecimal system.

WAY BACK IN THE MISTS OF TIME

With regard to the previous topic and as an additional nugget of trivia, the term *tetrapod* refers to an animal that has four limbs, along with hips and shoulders and fingers and toes. In the mid-1980s, paleontologists discovered *Acanthostega* who, at approximately 350 million years old, is one of the most primitive tetrapods known—so primitive in fact, that these creatures still lived exclusively in water and had not yet ventured onto land.

After the dinosaurs (who were also tetrapods) exited the stage, humans were one branch of the tetrapod tree that eventually inherited the earth (along with hippopotami, hedgehogs, aardvarks, frogs ... and all of the other vertebrates). Ultimately, we're all descended from *Acanthostega* or one of her cousins. The point is that *Acanthostega* had eight fully evolved fingers on each hand (Figure 7.18).

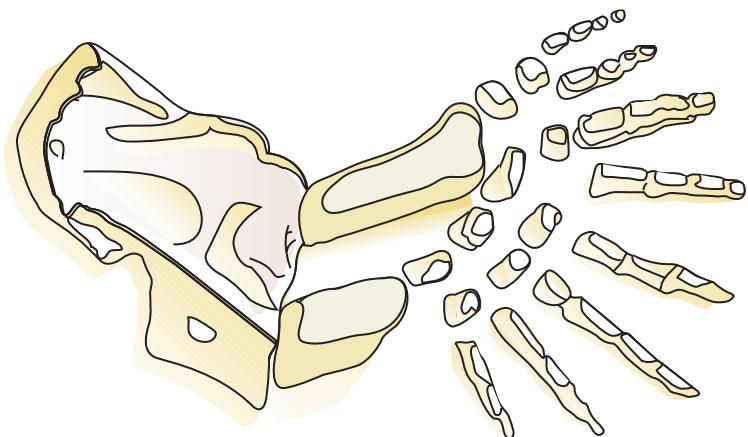


FIGURE 7.18

The first tetrapods had eight fingers on each hand. (Courtesy Clive “Max” Maxfield and Alvin Brown)

So ... if evolution hadn't taken a slight detour, we'd probably have ended up using a base-16 numbering system (which would have been jolly handy when we finally got around to inventing computers, let me tell you).⁷

REPRESENTING NUMBERS USING POWERS

An alternative way of representing numbers is by means of *powers*; for example, 10^3 , where 10 is the base value and the superscripted 3 is known as the *power*

⁷As another point of interest, the Irish hero Cuchulain was reported as having seven fingers on each hand (but this would have been no help in computing whatsoever).

or *exponent*.⁸ We read 10^3 as “ten to the power of three.” The power specifies how many times the base value must be multiplied by itself; thus, 10^3 represents $10 \times 10 \times 10$. Any value can be used as a base (Figure 7.19).

<i>Decimal (Base-10)</i>				
$10^0 = 1$	=			1_{10}
$10^1 = 10$	=			10_{10}
$10^2 = 10 \times 10$	=			100_{10}
$10^3 = 10 \times 10 \times 10$	=			1000_{10}

<i>Binary (Base-2)</i>				
$2^0 = 1$	=	1_2	=	1_{10}
$2^1 = 2$	=	10_2	=	2_{10}
$2^2 = 2 \times 2$	=	100_2	=	4_{10}
$2^3 = 2 \times 2 \times 2$	=	1000_2	=	8_{10}

<i>Octal (Base-8)</i>				
$8^0 = 1$	=	1_8	=	1_{10}
$8^1 = 8$	=	10_8	=	8_{10}
$8^2 = 8 \times 8$	=	100_8	=	64_{10}
$8^3 = 8 \times 8 \times 8$	=	1000_8	=	512_{10}

<i>Hexadecimal (Base-16)</i>				
$16^0 = 1$	=	1_{16}	=	1_{10}
$16^1 = 16$	=	10_{16}	=	16_{10}
$16^2 = 16 \times 16$	=	100_{16}	=	256_{10}
$16^3 = 16 \times 16 \times 16$	=	1000_{16}	=	4096_{10}

FIGURE 7.19

Representing numbers using powers.

Any base to the power of one is equal to itself; for example, $8^1 = 8$. Strictly speaking, a power of zero is not part of the series, but by convention, any base to the power of zero equals one; for example, $8^0 = 1$. Powers provide a convenient way to represent column-weights in place-value systems, as illustrated in Figures 7.20, 7.21, and 7.22.

⁸Rather than talking about using powers, some mathematicians prefer to refer to this type of representation as an exponential form.

FIGURE 7.20
Using powers to represent column weights in decimal.

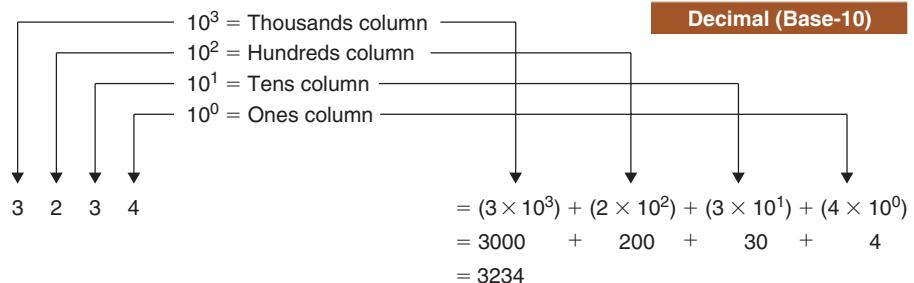


FIGURE 7.21
Using powers to represent column weights in binary.

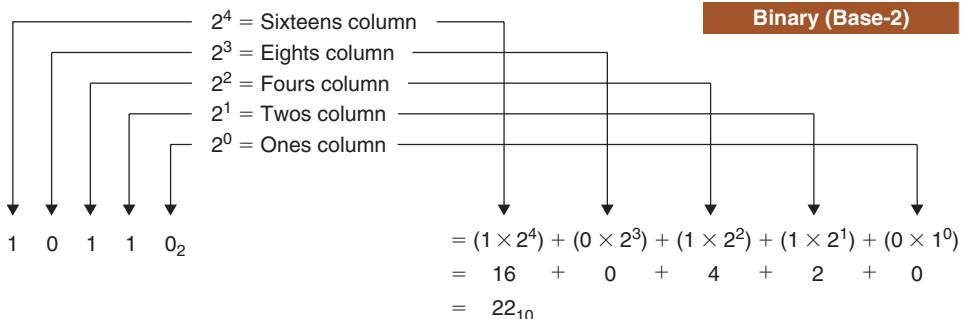
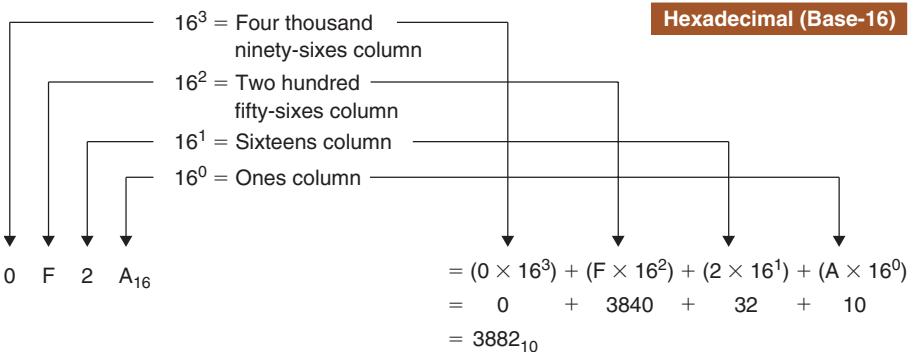


FIGURE 7.22
Using powers to represent column weights in hexadecimal.



LUCKY AND UNLUCKY NUMBERS

Apropos of nothing at all, man has been interested in the properties of odd numbers since antiquity, often ascribing mystical and magical properties to them, for example, “lucky seven” and “unlucky thirteen.” As Pliny the Elder (23–79 AD) is reported to have said, “*Why is it that we entertain the belief that for every purpose odd numbers are the most effectual?*”

TERTIARY LOGIC

And finally, for reasons that have become a little involved, communications theory tells us that optimal data transfer rates can be achieved if each data element represents three states (see Box). Now engineers wouldn't be the guys and gals they are if they weren't prepared to accept a challenge, and some experiments have been performed with *tertiary logic*. This refers to logic gates that are based on three distinct voltage levels. In this case, the three voltages are used to represent the *tertiary (base-3)* values of 0, 1, and 2, and their logical equivalents FALSE, TRUE, and MAYBE.⁹

However, while it's relatively easy to use transistors to generate two distinct voltage levels, it's harder to generate a consistent intermediate voltage that can represent a third value. Similarly, while it's relatively easy to use transistors to interpret (detect) two voltage levels, it's harder to interpret a third, intermediate value.¹⁰

Additionally, creating and using an equivalent to Boolean Algebra¹¹ that works with the three logic states FALSE, TRUE, and MAYBE is enough to make even the strongest among us quail. Thankfully, tertiary logic is currently of academic interest only (otherwise this book might have been substantially longer). Still, there's an old saying: "*What goes around comes around,*" and it's not beyond the realm of possibility that tertiary logic (or an even bigger relative) will rear its ugly head sometime in the future.

In this context, the term "data" refers to numerical, logical, or other digital information presented in a form suitable for processing by an electronic system such as a digital computer.

Actually, "data" is the plural of the Latin *datum*, meaning "something given." The plural usage is still common, especially among scientists, so it's not unusual to see expressions like "*These data are ...*"

However, it is becoming increasingly common to use "data" to refer to a singular group entity such as information. Thus, an expression like "*This data is ...*" would also be acceptable to a modern audience.

⁹A *tertiary digit* is known as a *trit*.

¹⁰Actually, if the truth were told, it's really not too difficult to generate and detect three voltage levels using modern components. The problem is that you can't achieve this without using so many transistors that any advantages of using a tertiary system are lost.

¹¹Boolean Algebra is introduced in *Chapter 9: Boolean Algebra*.

This page intentionally left blank

CHAPTER 8

Binary Arithmetic

BEFORE WE START ...

Because digital computers are constructed from logic gates that can represent only two states, they are obliged to make use of the *binary (base-2)* number system with its two digits: 0 and 1 (see *Chapter 7: Alternative Numbering Systems*, for more details on binary).

Unlike calculations on paper where both decimal and binary numbers can be of any size—limited only by the size of your paper, the endurance of your pencil, and your stamina—the numbers manipulated within a computer have to be mapped onto a physical system of logic gates and wires. Thus, the maximum value of a number inside a computer is dictated by the width of its data path; that is, the number of bits used to represent that number.¹

87

UNSIGNED BINARY NUMBERS

As their name might suggest, *unsigned binary numbers* don't have the concept of a sign (plus or minus), which means they can be used to represent only positive values. Consider the range of numbers that can be represented using 8 bits (Figure 8.1).

Each "x" character represents a single bit; the right-hand bit is known as the *Least Significant Bit* (LSB) because it represents the smallest value. Similarly, the left hand bit is known as the *Most Significant Bit* (MSB) because it represents the largest value.

In computing, it is usual to commence indexing things from zero, so the least significant bit is referred to as bit 0, and the most significant bit (of an 8-bit value) is referred to as bit 7. Every bit can be individually assigned a value of

¹Actually, this isn't strictly true, because there are tricks we can use to represent large numbers by splitting them into smaller "chunks" and reusing the same bits over and over again, but that's beyond the scope of what we're looking at here.

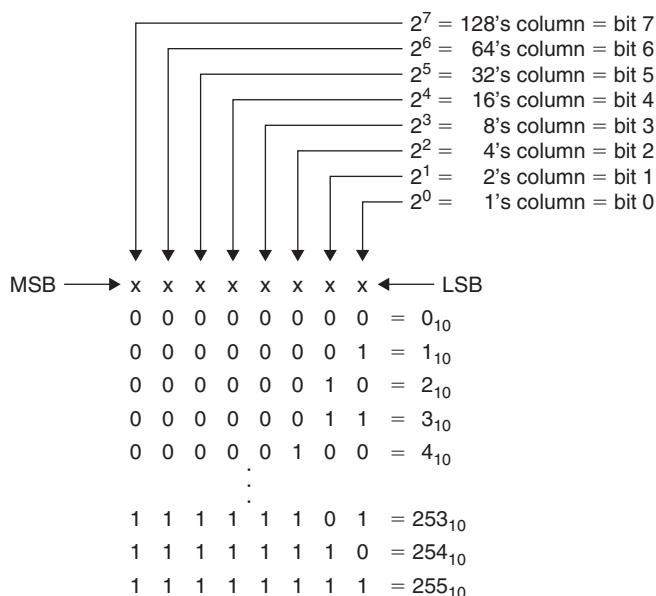


FIGURE 8.1
Unsigned binary numbers.

For each of the remaining bits, there may be a *carry-in* from the previous stage and a *carry-out* to the next stage. To fully illustrate this process, consider the step-by-step addition of two 8-bit binary numbers, as illustrated in Figure 8.2.

$ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\ \hline & & & & & & & 1 \end{array} $ <p>(a) Bit 0, $1 + 0 = 1_2$</p> $ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\ \hline & & & & 0 & 1 & 1 \end{array} $ <p>(c) Bit 2, $0 + 0 = 0_2$</p> $ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0 \\ \hline & 1 & 0 & 0 & 1 & 1 \end{array} $ <p>(e) Bit 4, $1 + 1 + \text{carry_in} = 11_2$</p> $ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \\ \hline & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array} $ <p>(g) Bit 6, $0 + 0 + \text{carry_in} = 1_2$</p>	$ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\ \hline & & & & & & & 1\ 1 \end{array} $ <p>(b) Bit 1, $0 + 1 = 1_2$</p> $ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\ \hline & & & & 0 & 0 & 1 & 1 \end{array} $ <p>(d) Bit 3, $1 + 1 = 10_2$</p> $ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \\ \hline & 0 & 1 & 0 & 0 & 1 & 1 \end{array} $ <p>(f) Bit 5, $1 + 0 + \text{carry_in} = 10_2$</p> $ \begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ + 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \\ \hline & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array} $ <p>(h) Bit 7, $0 + 0 = 0_2$</p>
	\longleftrightarrow 57_{10} $+ 26_{10}$ $= 83_{10}$

FIGURE 8.2
Binary addition.

We commence with the least significant bits in step (a), $1 + 0 = 1$. In step (b) we see $0 + 1 = 1$, followed by $0 + 0 = 0$ in step (c). The first carry-out occurs in step (d), where $1 + 1 = 2$ (that's $1_2 + 1_2 = 10_2$ in binary); thus, in this instance, the sum is 0 and there is a carry-out of 1 to the next stage.

The second carry-out occurs in step (e), where the carry-in from the previous stage results in $1 + 1 + 1 = 3$ (that's $1_2 + 1_2 + 1_2 = 11_2$ in binary); thus, in this instance, the sum is 1 and there is a carry-out of 1 to the next stage. The third and final carry-out occurs in step (f), and it's plain sailing from there on in.

NINES' AND TEN'S COMPLEMENTS

There are two forms of complement associated with every number system, the *radix complement* and the *diminished radix complement*, where the term *radix* refers to the base of the number system. Under the decimal (base-10) system, the radix complement is also known as the *ten's complement* and the diminished radix complement is known as the *nines' complement*.

There's a lot of confusion in this area: should one say *nines*, *nine's*, or *nines' complement*? Similarly, should one say *tens*, *ten's*, or *tens' complement*? The problem is that you can run across all of these variants.

Some folks recommend using the placement of the apostrophe to distinguish between the radix complement and the diminished radix complement. In this usage, for example, the term *three's complement* would refer to the *radix complement* of a value in base-3, while the term *threes' complement* would indicate *diminished radix complement* of a value in base-4.

Other folks (if they've thought about it at all) are of the view that the distinction is not important when the radix is apparent, which is nearly always the case. Thus, many writers use *nine's complement* and *ten's complement* [or *one's complement* and *two's complement* in the case of the binary (base-2) equivalents]. In fact, many style manuals leave out the apostrophe completely, recommending *nines complement* and *tens complement* (or *ones complement* and *twos complement* in binary).

First, let's consider a decimal subtraction performed using the nines' complement technique (Figure 8.3).

The standard way of performing the operation would be to subtract the *subtrahend* (283) from the *minuend* (647), which, as in this example, may require the use of one or more *borrow operations*. To perform the equivalent operation using a nines' complement approach, each of the digits of the subtrahend is

Standard subtraction

$$\begin{array}{r} 6 \ 4 \ 7 \\ - 2 \ 8 \ 3 \\ \hline = 3 \ 6 \ 4 \end{array}$$

Nines' complement equivalent

$$\begin{array}{r} 9 \ 9 \ 9 \\ - 2 \ 8 \ 3 \\ \hline = 7 \ 1 \ 6 \end{array} \quad \begin{array}{r} 6 \ 4 \ 7 \\ + 7 \ 1 \ 6 \\ \hline = 1 \ 3 \ 6 \ 3 \end{array}$$

Take nines' complement Add nines' complement to minuend

} End-around-carry

FIGURE 8.3

Nines' complement decimal subtraction.

first subtracted from a 9. The resulting nines' complement value is added to the minuend, and then an *end-around-carry operation* is performed. The advantage of the nines' complement technique is that it is *never necessary* to perform a borrow operation.²

Now consider the same subtraction performed using the ten's complement technique (Figure 8.4).

Standard subtraction

$$\begin{array}{r} 6 \ 4 \ 7 \\ - 2 \ 8 \ 3 \\ \hline = 3 \ 6 \ 4 \end{array}$$

Ten's complement equivalent

$$\begin{array}{r} 1 \ 0 \ 0 \ 0 \\ - 2 \ 8 \ 3 \\ \hline = 7 \ 1 \ 7 \end{array} \quad \begin{array}{r} 6 \ 4 \ 7 \\ + 7 \ 1 \ 7 \\ \hline = 1 \ 3 \ 6 \ 4 \end{array}$$

Take ten's complement Add ten's complement to minuend

} Drop any carry

FIGURE 8.4

Ten's complement decimal subtraction.

The advantage of the ten's complement approach is that it is not necessary to perform an end-around-carry; any carry-out resulting from the addition of the most significant digits is simply dropped from the final result. The disadvantage is that, during the process of creating the ten's complement, it is necessary to perform a borrow operation for *every nonzero digit* in the subtrahend. This problem could be solved by first taking the nines' complement of the subtrahend, adding one to the result, and then performing the remaining operations as for the ten's complement.

²The fact that one doesn't have to perform any borrow operations when using nines' complements made this technique extremely popular in the days of yore when the math skills of the general populace weren't particularly high.

SUBTRACTING UNSIGNED BINARY NUMBERS

Unsigned binary numbers may be subtracted from each other using an identical process to that used for decimal subtraction. However, for reasons of efficiency, computers rarely perform subtractions in this manner; instead, these operations are typically performed by means of *complement techniques*.

In the case of binary (base-2) numbers, the radix complement is known as the *two's complement* and the diminished radix complement is known as the *ones' complement*. First, consider a binary subtraction performed using the ones' complement technique (Figure 8.5).

Standard subtraction

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ - 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \\ \hline = 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \end{array} \quad \longleftrightarrow \quad 57_{10} - 30_{10} = 27_{10}$$



Ones' complement equivalent

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ - 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \\ \hline = 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \end{array}$$

Take ones' complement

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ + 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \\ \hline 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \end{array}$$

End-around-carry

$$\begin{array}{r} 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

Add ones' complement to minuend

FIGURE 8.5
Ones' complement binary subtraction.

Once again, the standard way of performing the operation would be to subtract the subtrahend (0001110_2) from the minuend (00111001_2), which may require the use of one or more borrow operations. To perform the equivalent operation in ones' complement, each of the digits of the subtrahend is first subtracted from a 1. The resulting ones' complement value is added to the minuend, and then an end-around-carry operation is performed. The advantage of the ones' complement technique is that it is *never necessary* to perform a borrow operation. In fact, it isn't even necessary to perform a subtraction, because the ones' complement of a binary number can be simply generated by inverting all of its bits; that is, by exchanging all the 0s with 1s and vice versa.

Now consider the same binary subtraction performed using the two's complement technique (Figure 8.6).

Standard subtraction

$$\begin{array}{r}
 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\
 - 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \\
 \hline
 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1
 \end{array}
 \quad \Rightarrow 57_{10} - 30_{10} = 27_{10}$$



Two's complement equivalent

$$\begin{array}{r}
 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
 - 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \\
 \hline
 = 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0
 \end{array}
 \quad \Rightarrow \quad \begin{array}{r}
 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\
 + 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0 \\
 \hline
 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1
 \end{array}$$

Take two's complement

Drop any carry

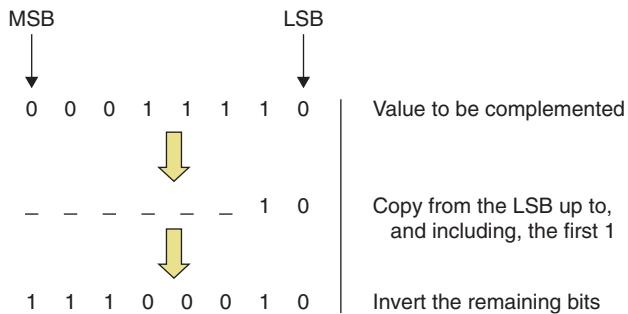
Add two's complement
to minuend

FIGURE 8.6
Two's complement binary subtraction.

As before, the advantage of the two's complement is that it is not necessary to perform an end-around-carry; any carry-out resulting from the addition of the two most significant bits is simply dropped from the final result. The disadvantage is that, during the process of creating the two's complement, it is necessary to perform a borrow operation for *every nonzero digit* in the subtrahend. This problem can be solved by first taking the ones' complement of the subtrahend, adding one to the result, and then performing the remaining operations as for the two's complement.

As fate would have it, there is a short-cut approach available to generate the two's complement of a binary number. Commencing with the least significant bit of the value to be complemented, each bit up to and including the first 1 is copied directly, and the remaining bits are inverted (Figure 8.7).

Unfortunately, the binary subtraction examples presented in Figures 8.5 and 8.6 would return incorrect results if we were to swap the values of the minuend

**FIGURE 8.7**

Shortcut for generating a two's complement.

and the subtrahend (the values on the top and the bottom, respectively). This is because, for the purposes of these examples, we made certain that the minuend was larger than the subtrahend, thereby ensuring that the result would be a positive value.

To put this another way, in order for these techniques to work, the final result must be greater than or equal to zero. The reason is clear: subtracting a larger value from a smaller value results in a negative value, but we're currently using unsigned binary numbers and—by definition—an unsigned binary number can only be used to represent a positive value.

Now, it would obviously be somewhat inconvenient if computers could only be used to generate positive values, so we need to come up with some way to represent both positive *and* negative numbers ...

SIGN-MAGNITUDE BINARY NUMBERS

In standard decimal arithmetic, numbers are typically represented in a form known as *sign-magnitude*,³ which means prefixing values with plus or minus signs (by default, numbers without signs are assumed to represent positive values). For example, values of plus and minus 27 would be shown as +27 and -27 (or just 27 and -27), respectively.

If we so desired, we could use the same technique in binary; for example, we could say that the most significant bit was to be considered to be the *sign bit*; also that a 0 in this bit would indicate a positive value while a 1 would indicate a negative value. In the case of an 8-bit value, for example, 00000001_2 would represent +1, 00000010_2 would represent +2, 00000011_2 would represent +3, and so forth, while 10000001_2 would represent -1, 00000010_2 would represent -2, 00000011_2 would represent -3, and so on.

Using this binary sign-magnitude approach with an 8-bit value would allow us to represent numbers in the range -127 to +127 (11111111_2 to 01111111_2). One slightly awkward outcome of this scheme is that we would also end up with both negative and positive versions of zero; that is, -0 and +0 (10000000_2 and 00000000_2). Why would this be a problem? Well, if we were comparing two values, would +0 be considered to be larger than -0? In fact, for reasons of efficiency, computers rarely employ the sign-magnitude form, and instead use the *signed binary* format discussed in the next topic.

³This is sometimes written as *sign + magnitude*.

SIGNED BINARY NUMBERS

Perhaps not surprisingly, *signed binary numbers* have the concept of a sign, and they can be used to represent both positive and negative values. Once again, the most significant bit is also called the *sign bit*, but signed binary numbers use this bit in a rather cunning way (Figure 8.8).

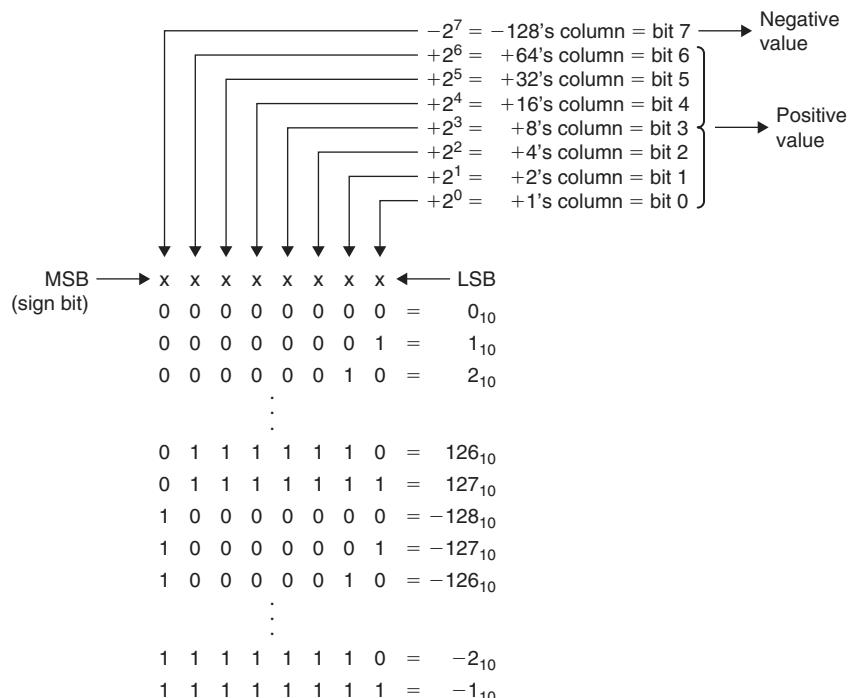
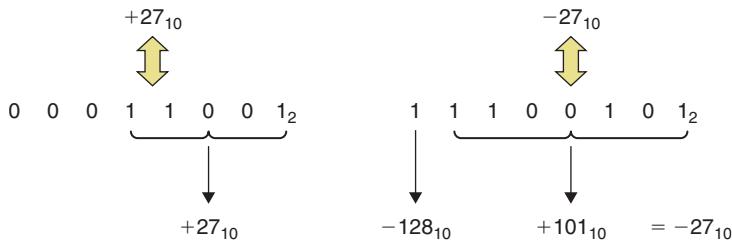


FIGURE 8.8

Signed binary numbers.

As we see, the least significant bits continue to represent the same positive quantities as for unsigned binary numbers, but the sign bit is used to represent a *negative quantity*. In the case of a signed 8-bit number, a 1 in the sign bit represents -2^7 (that's -128 in decimal), and the remaining bits are used to represent positive values in the range 0_{10} through $+127_{10}$. Thus, when the value represented by the sign bit is combined with the values represented by the remaining bits, an 8-bit signed binary number can be used to represent values in the range -128_{10} through $+127_{10}$ (also, we only have one bit-pattern representing zero).

Now, this can be tricky to wrap one's brain around the first time you see it. So, to further illustrate the differences between the sign-magnitude and signed binary formats, let's briefly consider a positive sign-magnitude decimal number and its negative equivalent; for example, $+27$ and -27 . As we see, the digits are identical for both cases and only the sign changes. Now consider the same values represented as signed binary numbers (Figure 8.9).

**FIGURE 8.9**

Comparison of positive and negative signed binary numbers.

In this case, the bit patterns of the two binary numbers are very different. This is because the sign bit represents an actual quantity (-128_{10}) rather than a simple plus or minus; thus, the signed equivalent of -27_{10} is formed by combining -128_{10} with $+101_{10}$.

ADDING SIGNED BINARY NUMBERS

At this point you're probably muttering to yourself: "*This is insane, what kind of raving lunatic would come up with a scheme as convoluted as this?*" Well, sit up and pay attention, because this is the clever part: closer investigation of Figure 8.9 reveals that each bit pattern is in fact the two's complement of the other! To put this another way, taking the two's complement of a positive signed binary value returns its negative equivalent, and vice versa. And why does this make the world a better place? Well ...

... the end result of using signed binary numbers is to greatly reduce the complexity of operations within a computer. To illustrate why this is so, let's first consider one of the simplest arithmetic operations, that of addition. Compare the additions of positive and negative decimal values in sign-magnitude form with their signed binary counterparts (Figure 8.10).

Examine the standard decimal calculations on the left. The one at the top is easy to understand because it's a straightforward addition of two positive values. However, even though we are all extremely familiar with decimal addition, you probably found the other three a little harder because you had to decide exactly what to do with the negative values. By comparison, the signed binary calculations on the right are all simple additions, regardless of whether the individual values are positive or negative.

This really has huge implications. If computers were forced to use a binary version of the sign-magnitude form, they would have to perform a relatively complicated sequence of operations in order to achieve the simplest addition (Figure 8.11).

Decimal sign-magnitude		Signed binary
$\begin{array}{r} 57 \\ + 30 \\ \hline = 87 \end{array}$	↔	$\begin{array}{r} 001110001 \\ + 000111100 \\ \hline 010101111 \end{array}$
$\begin{array}{r} 57 \\ + -30 \\ \hline = 27 \end{array}$	↔	$\begin{array}{r} 001110001 \\ + 111000010 \\ \hline 000011011 \end{array}$
$\begin{array}{r} -57 \\ + 30 \\ \hline = -27 \end{array}$	↔	$\begin{array}{r} 110001111 \\ + 000111100 \\ \hline 111000101 \end{array}$
$\begin{array}{r} -57 \\ + -30 \\ \hline = -87 \end{array}$	↔	$\begin{array}{r} 110001111 \\ + 111000010 \\ \hline 101010001 \end{array}$

FIGURE 8.10

Comparison of sign-magnitude decimal versus signed binary additions.

First compare the signs of the two values.	
IF THE SIGNS ARE THE SAME: Add the values. Note that the result will always have the same sign as the original values.	IF THE SIGNS ARE DIFFERENT: Subtract the smaller value from the larger value, then attach the correct sign to the result.

FIGURE 8.11

Steps required for sign-magnitude additions.

As well as being time consuming, performing all these operations would require a substantial number of logic gates. Thus, the advantages of using the signed binary format for addition operations are apparent: signed binary numbers can always be directly added together to provide the correct result in a single operation, regardless of whether they represent positive or negative values. That is, the operations $a + b$, $a + (-b)$, $(-a) + b$, and $(-a) + (-b)$ are all performed in exactly the same way, by simply adding the two values together. This results in fast computers that can be constructed using a minimum number of logic gates.

SUBTRACTING SIGNED BINARY NUMBERS

Let's move on to consider the case of subtraction. We all know that $10 - 3 = 7$ in decimal arithmetic, and that the same result can be obtained by negating the

right-hand value and inverting the operation from a subtraction to an addition: that is, $10 + (-3) = 7$.

This technique is also true for signed binary arithmetic, although the negation of the right hand value is performed by taking its two's complement rather than by changing its sign. For example, consider a generic signed binary subtraction represented by $a - b$.⁴ Generating the two's complement of b results in $-b$, allowing the operation to be performed as an addition: $a + (-b)$. This means that computers do not require different blocks of logic to add and subtract; instead, they require only an *adder* and a *two's completer*, where the two's completer requires significantly fewer logic gates than a subtractor.

BINARY MULTIPLICATION

One technique for performing multiplication in any number base is by means of repeated addition; for example, in decimal, $6 \times 4 = 6 + 6 + 6 + 6 = 24$. However, even though computers can perform millions of operations every second, the repeated addition approach is time-consuming when the values to be multiplied are large. As an alternative, binary numbers may be multiplied together by means of a *shift-and-add* technique, as illustrated in Figure 8.12.

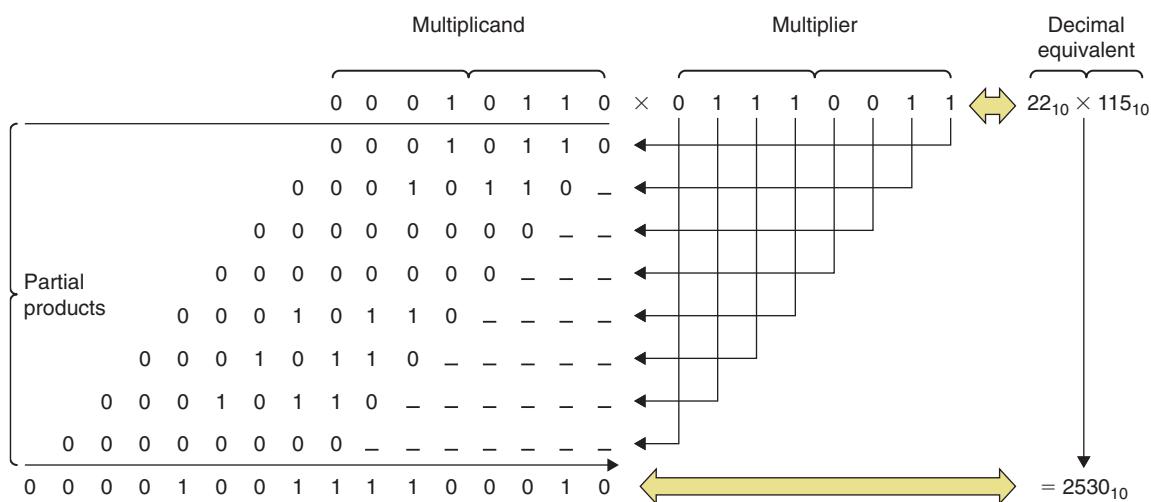


FIGURE 8.12
Binary multiplication using a shift-and-add technique.

⁴For the sake of simplicity, only the case of $a - b$ is discussed here. However, the operations $a - b$, $a - (-b)$, $(-a) - b$, and $(-a) - (-b)$ are all performed in exactly the same way, by simply taking the two's complement of b and adding the result to a , regardless of whether a and/or b represent positive or negative values.

Using this approach, a *partial product* is generated for every bit in the multiplier. If the value of the multiplier bit is 0, its corresponding partial product consists only of 0s; if the value of the bit is 1, its corresponding partial product is a copy of the multiplicand. Additionally, each partial product is left-shifted as a function of the multiplier bit with which it is associated; for example, the partial product associated with bit 0 in the multiplier is left-shifted zero bits, the partial product associated with bit 1 is left-shifted one bit, etc. All of the partial products are then added together to generate the result, whose width is equal to the sum of the widths of the two values being multiplied together.

Unfortunately, this algorithm works only with unsigned binary values. However, this problem can be overcome by taking the two's complement of any negative values before feeding them into the multiplier. If the signs of the two values are the same, both positive or both negative, then no further action need be taken.⁵ Alternatively, if the signs are different, then the result returned from the multiplier must be negated by transforming it into its two's complement.

There are several ways to construct a multiplier based on this shift-and-add technique. In one implementation, for example, all of the partial products are generated simultaneously and then added together. This requires a lot of logic gates, but the resulting multiplication is extremely fast. As another option, we can cycle round generating each of the partial products one-at-a-time and adding them into an accumulated result. This cuts down on the number of logic gates required, but it increases the time taken to achieve the final result.

BINARY DIVISION

As you might imagine, long division is just about as much fun in binary as it is in decimal, which is to say “*Not a lot!*” For this reason, binary division is best left to computers because they are in no position to argue about it.⁶

⁵This is because a positive multiplied by a positive and a negative multiplied by a negative both return positive results; for example, $(-3) \times (-4) = +12$.

⁶If you are interested in learning more about this topic, may I be so bold as to recommend another book that I coauthored with a friend, called *How Computers Do Math*. (That's what the book is called, not my friend—his name is Alvin.) This little rascal (again, the book, not Alvin), ISBN-13: 978-0471732785, comes equipped with a CD-ROM containing a virtual 8-bit computer-calculator that runs on your PC.

CHAPTER 9

Boolean Algebra

CABBAGES, PARROTS, AND BUCKETS OF BURNING OIL

One of the most significant mathematical tools available to electronics designers was actually invented for quite a different purpose. Around the 1850s, a British mathematician, George Boole (1815–1864), developed a new form of mathematics that is now known as *Boolean Algebra*. Boole's intention was to use mathematical techniques to represent and rigorously test logical and philosophical arguments. His work was based on the idea that a *statement* is a sentence that asserts or denies an attribute about an object or group of objects:

99

Statement: *Your face resembles a cabbage.*

Depending on how carefully you choose your friends, they may either agree or disagree with the sentiment expressed, but this is subjective and this statement cannot therefore be proved to be either *True* or *False*.

By comparison, a *proposition* is a statement that is either *True* or *False* with no ambiguity:

Proposition: *I just tipped a bucket of burning oil into your lap.*

This proposition may be *True* or it may be *False*, but it is definitely one or the other and there is no ambiguity about it.

Propositions can be combined together in several ways; a proposition combined with an AND operator is known as a *conjunction*:

Conjunction: *You have a parrot on your head AND you have a fish in your ear.*

The result of a conjunction is *True* if all of the propositions comprising that conjunction are *True*.

A proposition combined with an OR operator is known as a *disjunction*:

Disjunction: *You have a parrot on your head OR you have a fish in your ear.*

The result of a disjunction is *True* if at least one of the propositions comprising that disjunction is *True*.

From these humble beginnings, Boole established a new mathematical field known as symbolic logic, in which logical relationship between propositions can be represented symbolically by such means as equations or truth tables. Sadly, this work found little application outside the school of symbolic logic for almost one hundred years.

In fact, the significance of Boole's work was not fully appreciated until the late 1930s, when a graduate student at MIT, Claude Elwood Shannon (1916–2001), submitted a master's thesis that revolutionized electronics. In this thesis, Shannon showed that Boolean Algebra offered an ideal technique for representing the logical operation of digital systems. Shannon had realized that the Boolean concepts of *False* and *True* could be mapped onto the binary digits 0 and 1, and that both could be easily implemented by means of electronic circuits.^{1,2}

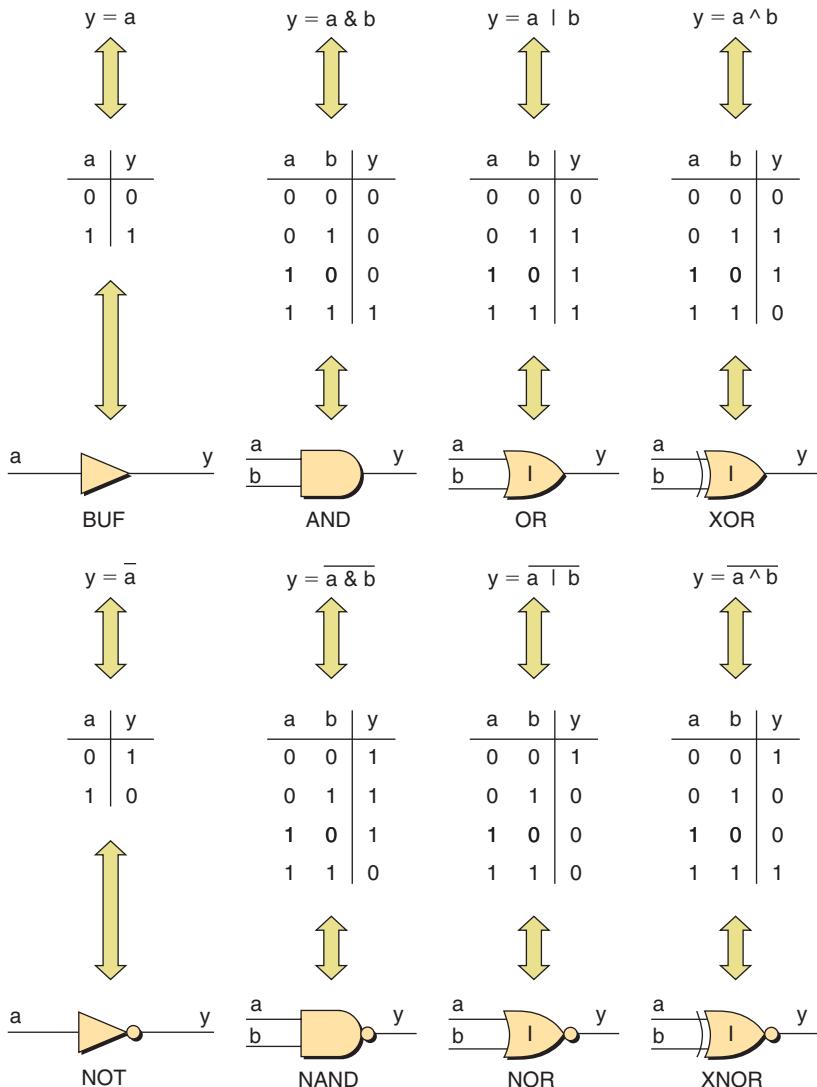
PRIMITIVE LOGIC FUNCTIONS

Logical functions can be represented using graphical symbols, equations, or truth tables, and these views can be used interchangeably (Figure 9.1).

There are a variety of ways to represent Boolean equations. In this book, the symbols $\&$, $|$, and \wedge are used to represent AND, OR, and XOR respectively; a

¹In addition to recognizing the application of Boolean Algebra to electronic design, Shannon is also credited with the invention of the rocket-powered Frisbee, and is famous for riding down the corridors at Bell Laboratories on a unicycle while simultaneously juggling four balls.

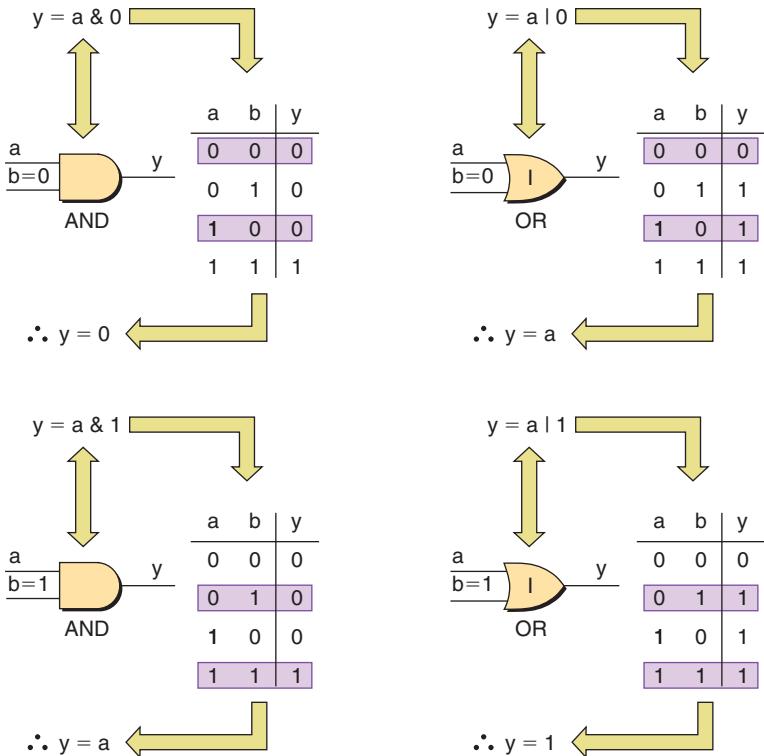
²The author can juggle with five china plates ... but only for a very short time indeed!

**FIGURE 9.1**

Summary of primitive logic functions.

negation, or NOT, is represented by a horizontal line, or *bar*, over the portion of the equation to be negated.

For the remainder of this chapter, we will typically use AND and OR functions in our examples. As we know from *Chapter 5: Primitive Logic Functions*, and *6: Using Transistors to Build Logic Gates*, the results from NAND and NOR functions would be the logical inverse of those from AND and OR functions, respectively.

**FIGURE 9.2**

Combining a single variable with a logic 0 or a logic 1.

COMBINING A SINGLE VARIABLE WITH LOGIC 0 OR LOGIC 1

A set of simple but highly useful rules can be derived from the combination of a single variable with a logic 0 or logic 1 (Figure 9.2).³

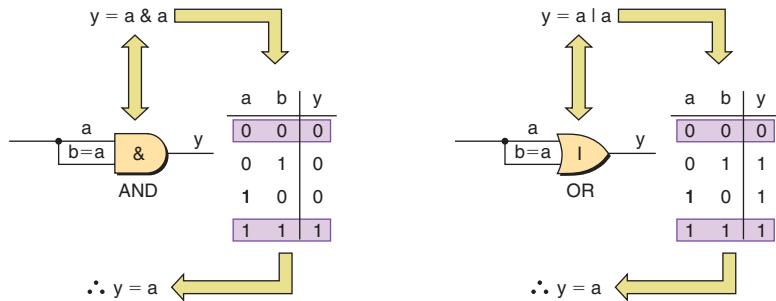
THE IDEMPOTENT RULES

The rules derived from the combination of a single variable with *itself* are known as the *idempotent rules* (Figure 9.3).

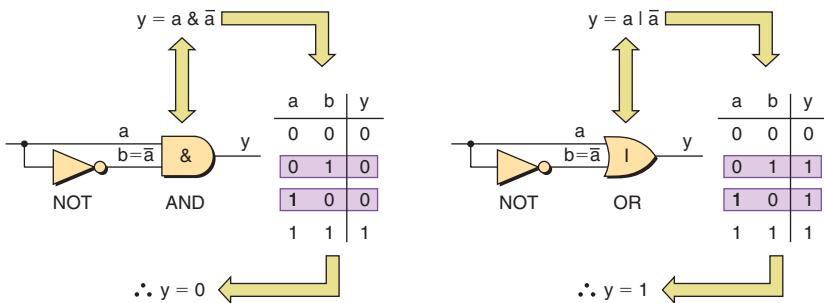
THE COMPLEMENTARY RULES

The rules derived from the combination of a single variable with the *inverse of itself* are known as the *complementary rules* (Figure 9.4).

³The symbol \therefore shown in the equations in Figures 9.2, 9.3, and 9.4 means “therefore.”

**FIGURE 9.3**

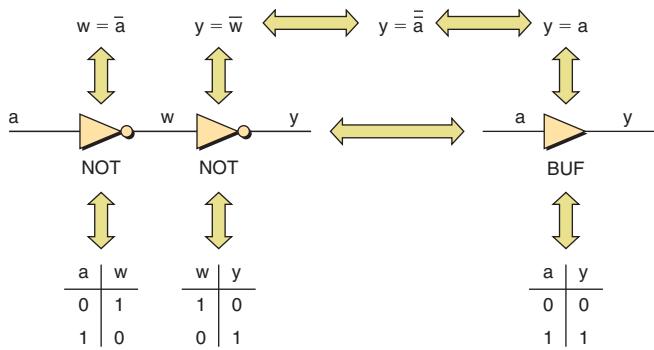
The idempotent rules.

**FIGURE 9.4**

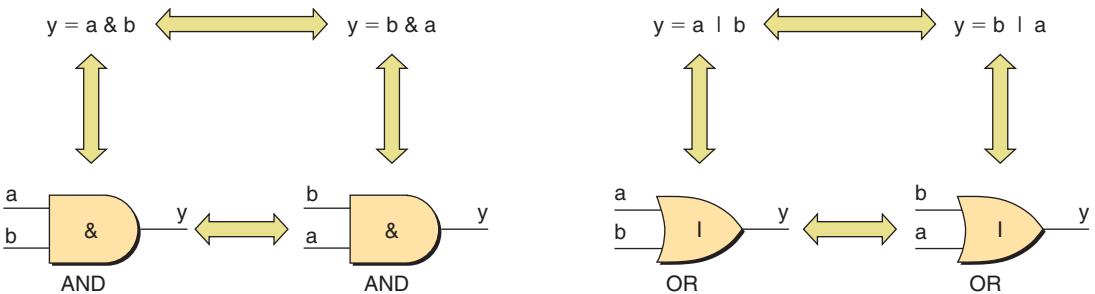
The complementary rules.

THE INVOLUTION RULE

The *involution rule* states that an even number of inversions cancel each other out; for example, two NOT functions connected in series generate an identical result to that of a BUF function (Figure 9.5).

**FIGURE 9.5**

The involution rule.

**FIGURE 9.6**

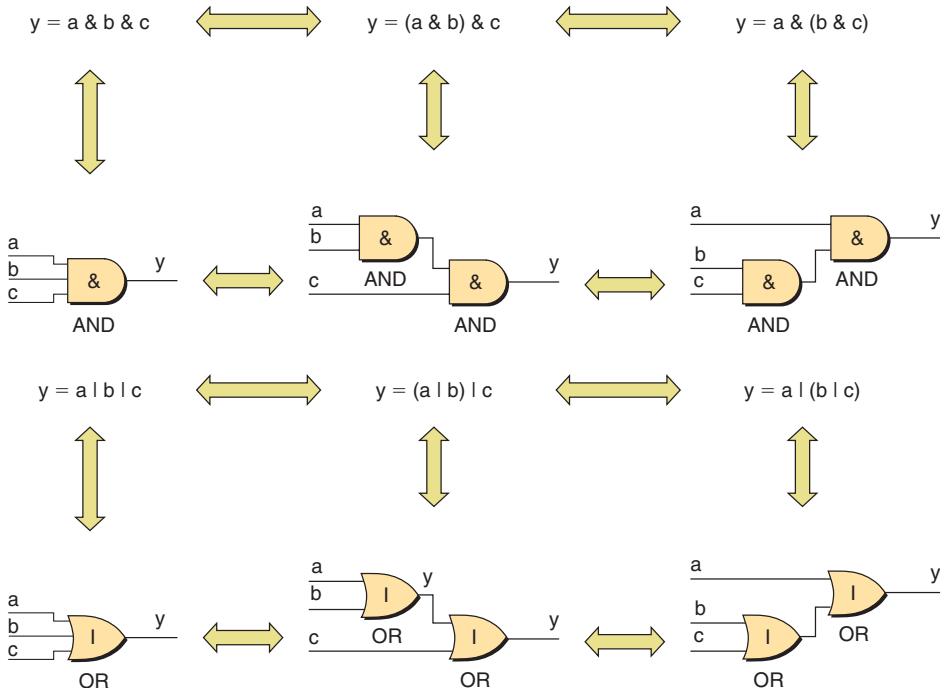
The commutative rules.

THE COMMUTATIVE RULES

The *commutative rules* state that the order in which variables are specified will not affect the result of an AND or OR operation (Figure 9.6).

THE ASSOCIATIVE RULES

The *associative rules* state that the order in which pairs of variables are associated together will not affect the result of multiple AND or OR operations (Figure 9.7).

**FIGURE 9.7**

The associative rules.

PRECEDENCE OF OPERATORS

In standard arithmetic, the multiplication operator is said to have a higher *precedence* than the addition operator. This means that, if an equation contains both multiplication and addition operators without parenthesis, then the multiplication is performed before the addition.⁴ For example:

$$6 + 2 \times 4 \equiv 6 + (2 \times 4)$$

Similarly, in Boolean Algebra, the & (AND) operator has a higher precedence than the | (OR) operator:

$$a | b \& c \equiv a | (b \& c)$$

Due to the similarities between these arithmetic and logical operators, the & (AND) operator is known as a *logical multiplication* or *product*, while the | (OR) operator is known as a *logical addition* or *sum*. To avoid any confusion as to the order in which logical operations will be performed, this book will always make use of parentheses.

THE FIRST DISTRIBUTIVE RULE

In standard arithmetic, the multiplication operator will *distribute* over the addition operator because it has a higher precedence. For example:

$$6 \times (5 + 2) \equiv (6 \times 5) + (6 \times 2)$$

Similarly, in Boolean Algebra, the & (AND) operator will distribute over an | (OR) operator because it has a higher precedence; this is known as the *first distributive rule* (Figure 9.8).

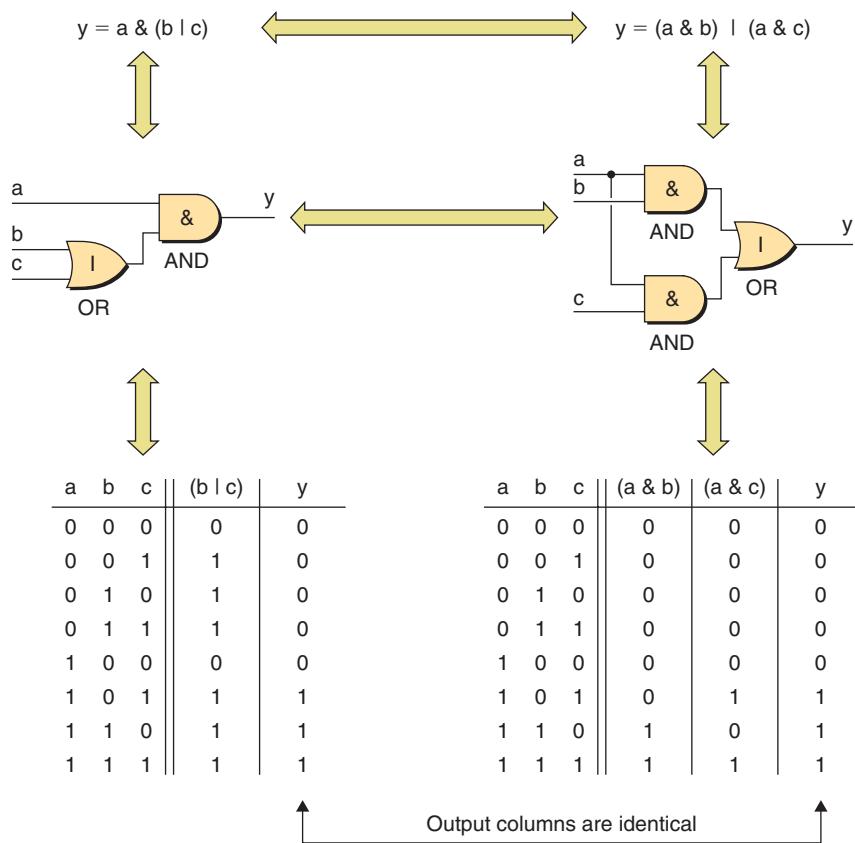
THE SECOND DISTRIBUTIVE RULE

In standard arithmetic, the addition operator will *not distribute* over the multiplication operator because it has a lower precedence.⁵ For example:

$$6 + (5 \times 2) \neq (6 + 5) \times (6 + 2)$$

⁴Note that the symbol \equiv shown in these equations indicates "is equivalent to" or "is the same as."

⁵Note that the symbol \neq shown in the equation indicates "is not equal to."

**FIGURE 9.8**

The first distributive rule.

However, Boolean Algebra is special in this case. Even though the $|$ (OR) operator has lower precedence than the $\&$ (AND) operator, it will still distribute over the $\&$ operator; this is known as the *second distributive rule* (Figure 9.9).

THE SIMPLIFICATION RULES

There are a number of *simplification rules* that can be used to reduce the complexity of Boolean expressions. As the end result is to reduce the number of logic gates required to implement the expression, the process of simplification is also known as *minimization* (Figure 9.10).

DEMORGAN TRANSFORMATIONS

A contemporary of Boole's, another British mathematician, Augustus DeMorgan (1806–1871), also made significant contributions to the field of

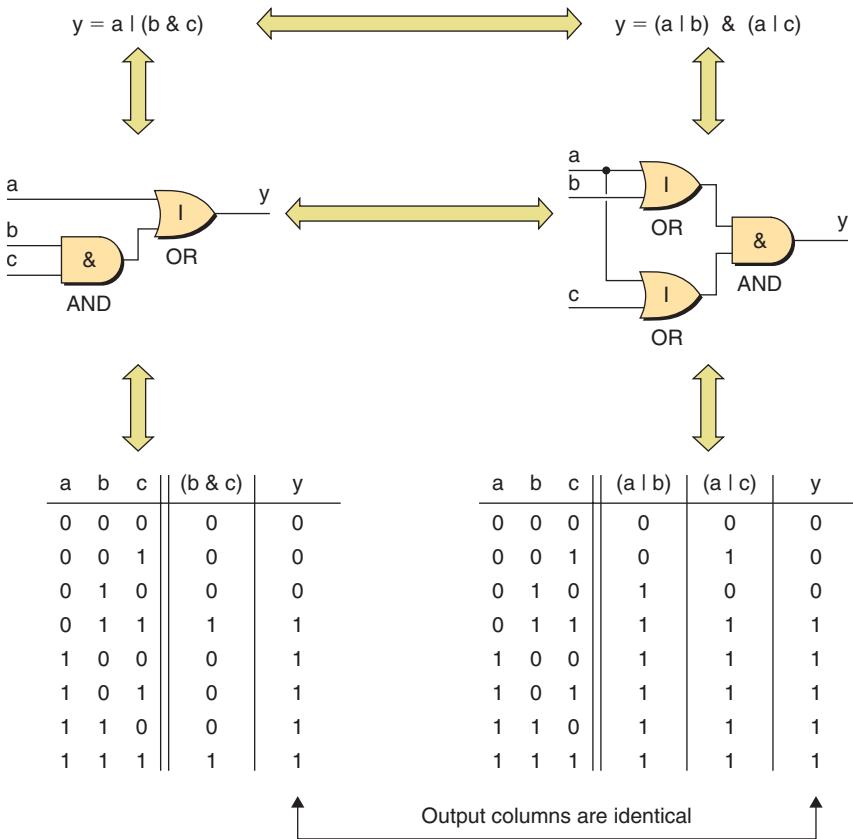


FIGURE 9.9
The second distributive rule.

symbolic logic; most notably, a set of rules called *DeMorgan Transformations*, which facilitate the conversion of Boolean expressions into alternate and often more convenient forms. As the *Encyclopedia Britannica* says: “A renascence of logical studies came about almost entirely because of Boole and DeMorgan.”⁶

A DeMorgan Transformation comprises four steps:

1. Exchange all of the $\&$ (AND) operators for \mid (OR) operators and vice versa.
2. Invert all the variables; also exchange 0s for 1s and vice versa.
3. Invert the entire function.
4. Reduce any multiple inversions.

⁶In fact, the rules we now attribute to DeMorgan were known in a more primitive form by William of Ockham (1280–1349) in the 14th century. To celebrate Ockham’s position in history, the OCCAM computer programming language was named in his honor. (OCCAM was the native programming language for the British-developed INMOS transputer.)

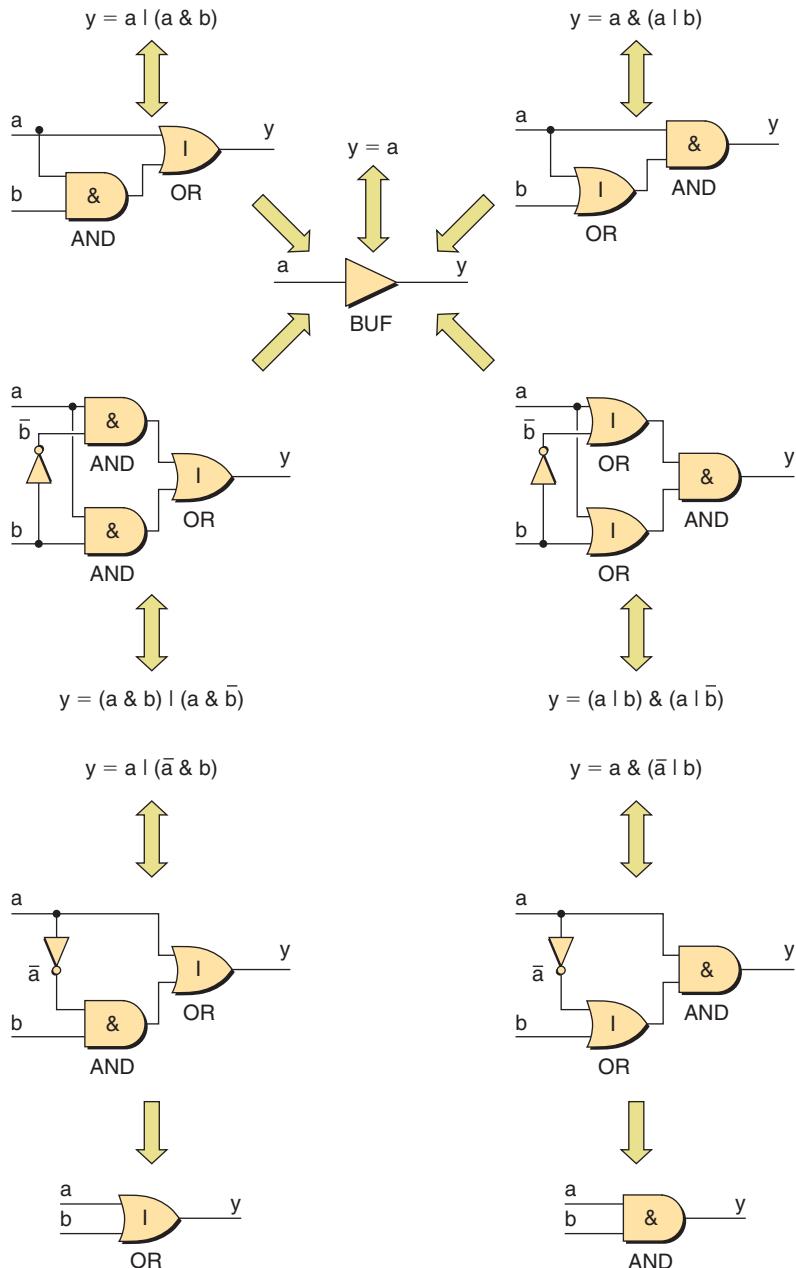
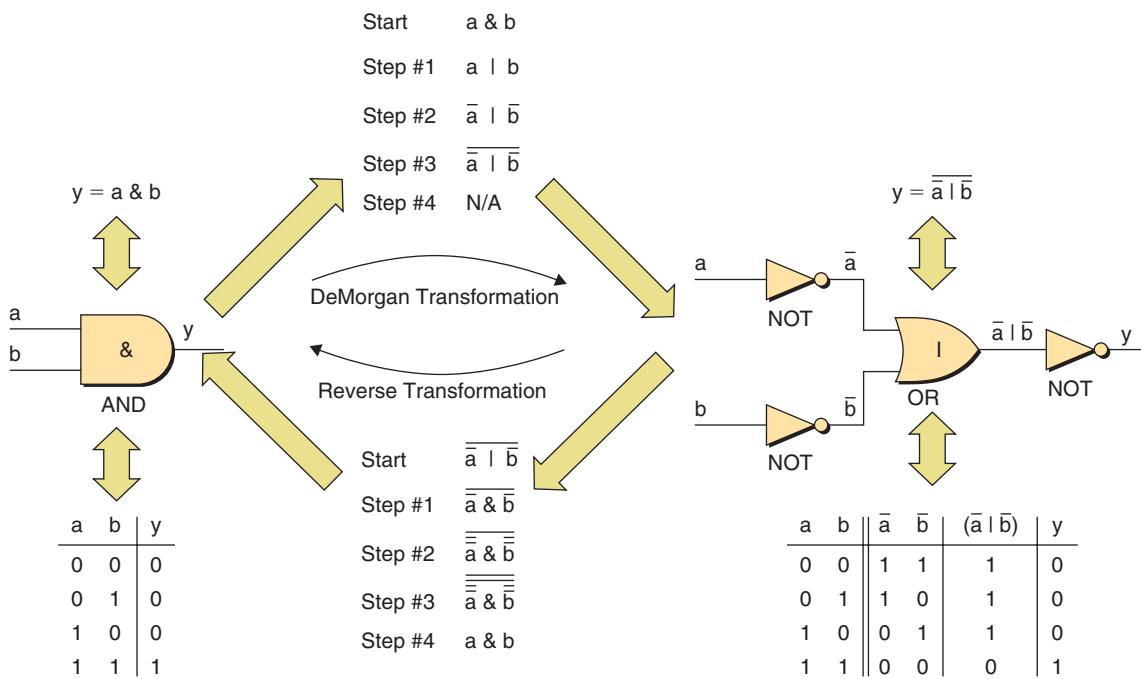


FIGURE 9.10
The simplification rules.

**FIGURE 9.11**

DeMorgan Transformation of an AND function.

Consider the DeMorgan Transformation of a 2-input AND function (Figure 9.11). Note that the NOT gate on the output of the new function can be combined with the OR to form a NOR.

Similar transformations can be performed on all of the primitive functions, as illustrated in Figure 9.12.

This is another one of those times when, at a first glance, the output result appears to be more complex than what we started with. On this basis, it's legitimate to say: *"Why would anyone bother to perform this rigmarole?"* Well, consider a simple circuit that implements the function $y = (a \& b) \mid (c \& d)$, as illustrated in Figure 9.13.

As we know from our discussions in *Chapter 6: Using Transistors to Build Logic Gates*, each AND and OR gate will require 6 transistors (assuming CMOS implementations), so this circuit will consume $3 \times 6 = 18$ transistors. Now, suppose that we perform a DeMorgan Transformation on the OR gate; this will result in the circuit shown in Figure 9.14.

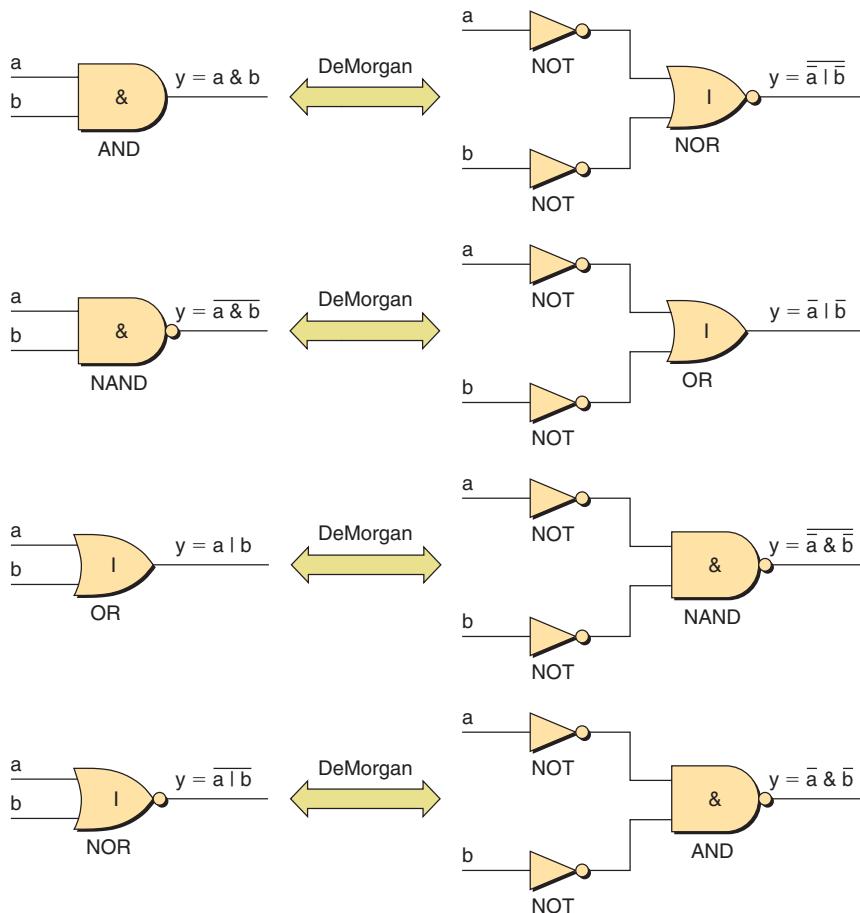


FIGURE 9.12
DeMorgan
Transformations of
AND, OR, NAND, and
NOR functions.

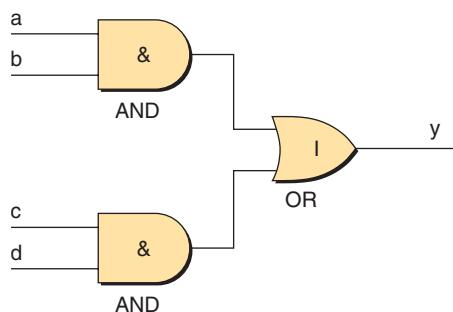
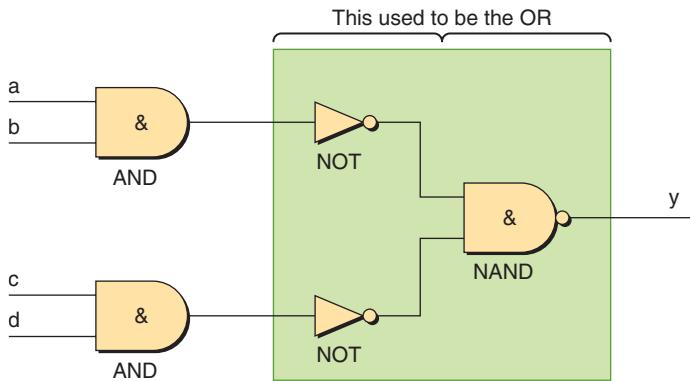


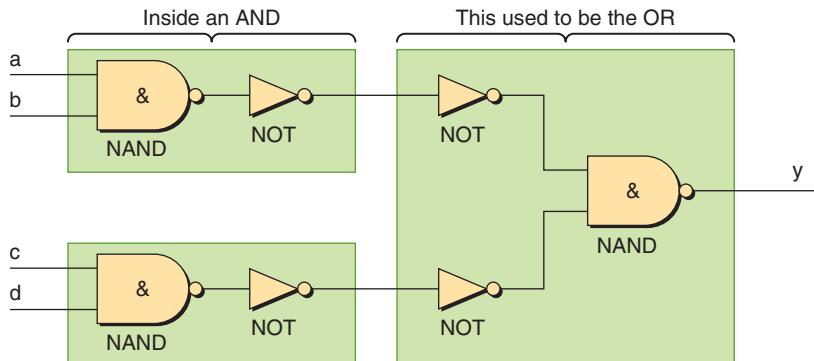
FIGURE 9.13
A simple function.

Now, at first glance, this may not seem to be too much of an advantage. On the one hand our new NAND gate has only 4 transistors (as compared to the 6 transistors required by the OR); but we've also added two NOT gates, each of which comprises 2 transistors, so our new implementation now consumes $(2 \times 6) + (2 \times 2) + 4 = 20$ transistors. Eeeek!

But wait a moment, remembering that we're talking about CMOS implementations of our logic gates, we know that an AND gate (with 6 transistors) is actually formed from a NAND gate (with 4 transistors) whose output is inverted by a NOT gate (with 2 transistors).

**FIGURE 9.14**

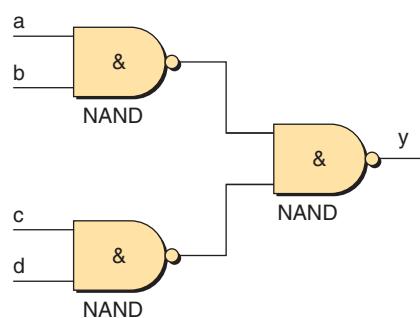
The same function following a DeMorgan Transformation on the OR.

**FIGURE 9.15**

Our AND gates are actually formed from NAND and NOT gates.

If we were to draw this out, the result would be as illustrated in Figure 9.15.

Now, as we know from the *involution rule* we introduced earlier in this chapter, an even number of inversions cancel each other out; for example, two NOT functions connected in series generate an identical result to that of a BUF function, which is (logically) the same as not having anything there at all. Thus, by cancelling out the pairs of NOT gates, we end up with a circuit comprising only three NAND gates (each containing 4 transistors) shown in Figure 9.16.

**FIGURE 9.16**
The final circuit.

Thus, by means of a DeMorgan Transformation, we've taken our original circuit comprising two ANDs and an OR (with 18 transistors) and transformed it into a new version comprising three NANDs (with 14 transistors). This equates to around a 20% reduction in transistors. This may not seem a lot in this case, but it becomes extremely significant when you're talking about a silicon chip containing hundreds of millions of transistors.

Of course, it can be a pain doing this sort of thing by hand (although this is the way we all used to do things when I was a bright-eyed, bushy-tailed young engineer). Fortunately, electronics designers now have computer-aided tools—such as *Logic Synthesis*—that do this sort of thing for us.

a	b	c	Minterms	Maxterms
0	0	0	($\bar{a} \& \bar{b} \& \bar{c}$)	($a \mid b \mid c$)
0	0	1	($\bar{a} \& \bar{b} \& c$)	($a \mid b \mid \bar{c}$)
0	1	0	($\bar{a} \& b \& \bar{c}$)	($a \mid \bar{b} \mid c$)
0	1	1	($\bar{a} \& b \& c$)	($a \mid \bar{b} \mid \bar{c}$)
1	0	0	($a \& \bar{b} \& \bar{c}$)	($\bar{a} \mid b \mid c$)
1	0	1	($a \& \bar{b} \& c$)	($\bar{a} \mid b \mid \bar{c}$)
1	1	0	($a \& b \& \bar{c}$)	($\bar{a} \mid \bar{b} \mid c$)
1	1	1	($a \& b \& c$)	($\bar{a} \mid \bar{b} \mid \bar{c}$)

FIGURE 9.17

Minterms and maxterms.

MINTERMS AND MAXTERMS

For each combination of inputs to a logical function, there is an associated *minterm* and an associated *maxterm*. Consider a truth table with three inputs: a, b, and c (Figure 9.17).

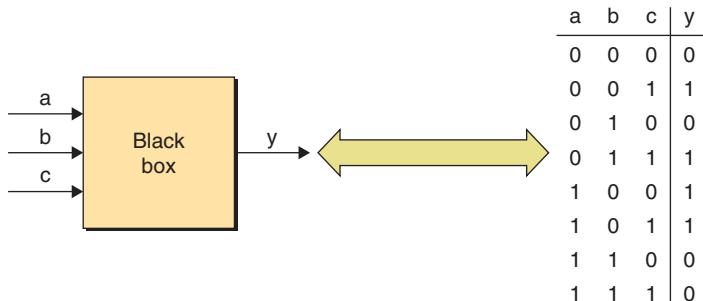
The *minterm* associated with each input combination is the & (AND), or *product*, of the input variables, while the *maxterm* is the | (OR), or *sum*, of the inverted input variables. Minterms and maxterms are useful for deriving Boolean equations from truth tables, as discussed in the following topic.

SUM-OF-PRODUCTS AND PRODUCT-OF-SUMS

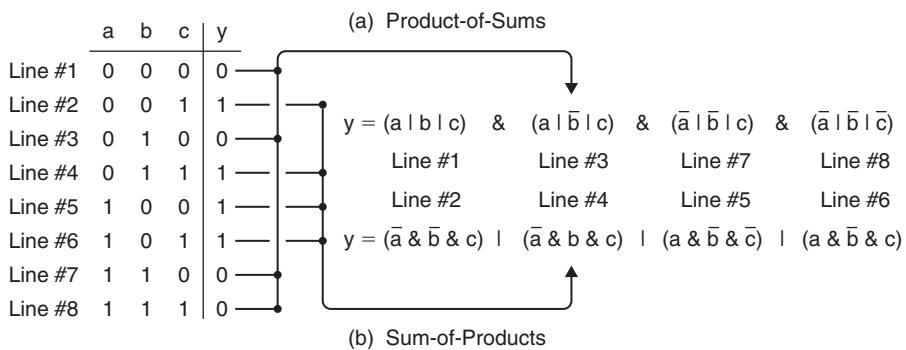
A designer will often specify portions of a design using truth tables, and determine how to implement these functions as logic gates later. The designer may start by representing a function as a *black box*⁷ with an associated truth table (Figure 9.18). Note that the values assigned to the output y in the truth table shown in Figure 9.18 were selected randomly, and have no significance beyond the purposes of this example.

There are two commonly used techniques for deriving Boolean equations from a truth table. In the first technique, the minterms corresponding to each line in the truth table for which the output is a logic 1 are extracted and combined using | (OR) operators; this method results in an equation said to be in *sum-of-products* form [Figure 9.19(a)]. In the second technique, the maxterms

⁷A *black box* is so-called because, initially, we don't know exactly what's going to be in it.

**FIGURE 9.18**

A black box with associated truth table.

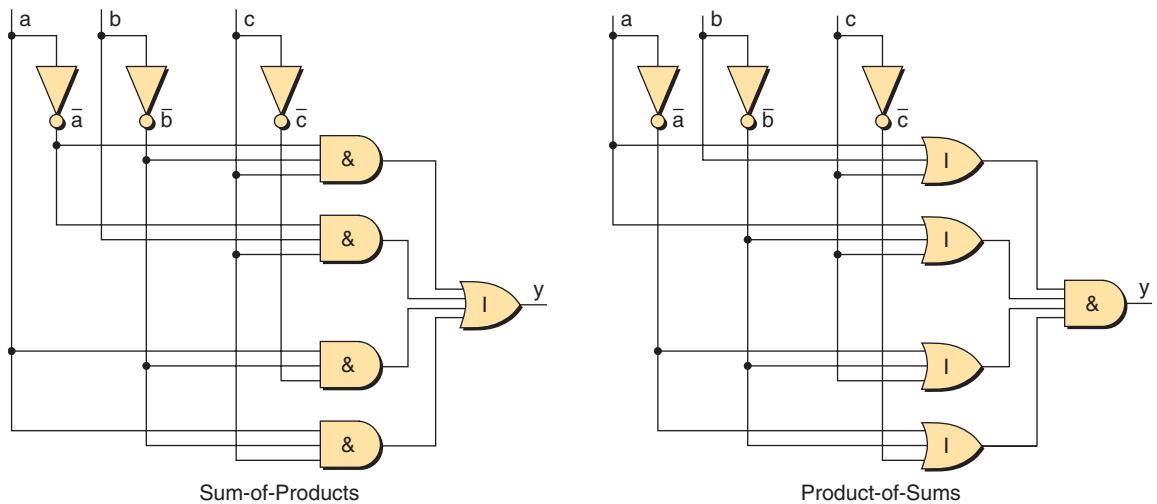
**FIGURE 9.19**

Sum-of-products versus products-of-sums (equations).

corresponding to each line in the truth table for which the output is a logic 0 are combined using $\&$ (AND) operators; this method results in an equation said to be in *product-of-sums* form [Figure 9.19(b)].

For a function whose output is logic 1 fewer times than it is logic 0, it is generally easier to extract a sum-of-products equation. By comparison, if the output is logic 0 fewer times than it is logic 1, it is generally easier to extract a product-of-sums equation. The sum-of-products and product-of-sums forms complement each other and return identical results. Furthermore, an equation in either form can be transformed into its alternative form by means of the appropriate DeMorgan Transformation.

Once an equation has been obtained in the required form, the designer would typically make use of the appropriate simplification rules to minimize the number of logic gates required to implement the function. However, neglecting any potential minimization, the equations above could be translated directly into their logic gate equivalents (Figure 9.20).

**FIGURE 9.20**

Sum-of-products versus products-of-sums (implementations).

CANONICAL FORMS

In a mathematical context, the term *canonical form* is taken to mean a generic or basic representation. Canonical forms provide the means to compare two expressions without falling into the trap of trying to compare “apples” with “oranges.” The sum-of-products and product-of-sums representations are different canonical forms. Thus, in order to compare two Boolean equations, both must first be coerced into the same canonical form; either sum-of-products or product-of-sums.

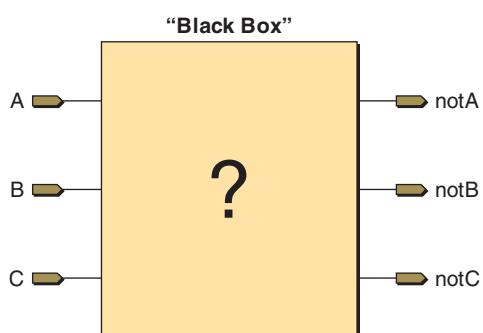


FIGURE 9.21
A black box with
three inputs and three
outputs.

AN INTERESTING CONUNDRUM

If you are new to Boolean Algebra, you may well feel that it's horribly complicated. In this case, you may be surprised to hear that engineers actually enjoy posing (and solving) logical problems. For example, here's an interesting conundrum that should get the old brain cells “firing on all six cylinders.” Consider a *black box* with three inputs (A, B, C) and three outputs (notA, notB, notC) as illustrated in Figure 9.21.

The notA output is the logical negation of the A input (if A is 0, then notA will be 1; if A is 1, then notA will be 0). Similarly, the notB and notC outputs are the logical negations of the B and C inputs, respectively.

Now, here's the clever part ... you have to decide how to implement the contents of the black box. You can use as many AND and OR gates as you wish, but you can use only two NOT gates, and you can't use any NAND, NOR, XOR, or XNOR gates (when you've had all the fun you can stand, visit *Appendix I: An Interesting Conundrum* to discover a solution).

This page intentionally left blank

CHAPTER 10

Karnaugh Maps

THE TREE OF PORPHYRY

Diagrams used to represent logical concepts have been around in one form or another for a very long time. For example, Aristotle (384–322 BC) was certainly familiar with the idea of using a stylized tree figure to represent the relationships between (and successive subdivisions of) such things as different species. Diagrams of this type, which are known as the *Tree of Porphyry*, are often to be found in medieval pictures.

JOHN VENN AND HIS VENN DIAGRAMS

Following the Tree of Porphyry, there seems to have been a dearth of activity on the logic diagram front until 1761, when the brilliant Swiss mathematician Leonhard Euler (whom we first met in *Chapter 7: Alternative Numbering Systems*) introduced a geometric system that could generate solutions for problems in class logic.

Unfortunately, Euler's work in this area didn't really catch on because it was somewhat awkward to use, and it was eventually supplanted in the 1890s by a more polished scheme proposed by the English logician John Venn (1834–1923). Venn was heavily influenced by the work of the English mathematician George Boole (whom we introduced in *Chapter 9: Boolean Algebra*), and his *Venn diagrams* very much complemented Boolean Algebra.

117

ALLAN MARQUAND AND LEWIS CARROLL

Venn diagrams were strongly based on the interrelationships between overlapping circles or ellipses. The first logic diagrams based on squares or rectangles were introduced in 1881 by Allan Marquand (1853–1924). A lecturer in logic and ethics at John Hopkins University, Marquand's diagrams spurred interest by a number of other contenders, including one offering by an English logician and author, the Reverend Charles Lutwidge Dodgson (1832–1898).

Dodgson's diagrammatic technique first appeared in his book *The Game of Logic*, which was published in 1886, but he is better known to us by his pen name, Lewis Carroll, and as being the author of *Alice's Adventures in Wonderland*. Apart from anything else, these rectangular diagrams are of interest to us because they were the forerunners of a more modern form known as *Karnaugh maps* ...

MAURICE KARNAUGH AND KARNAUGH MAPS

In 1953, the American physicist Maurice Karnaugh (pronounced "car-no," 1924–) invented a form of logic diagram called a *Karnaugh map*, which provides an alternative technique for representing Boolean functions; for example, consider the Karnaugh map for a 2-input AND function (Figure 10.1).

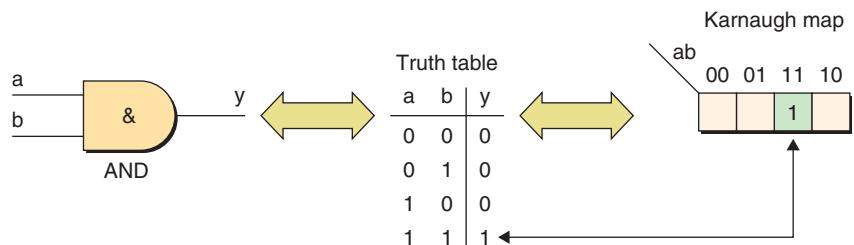


FIGURE 10.1

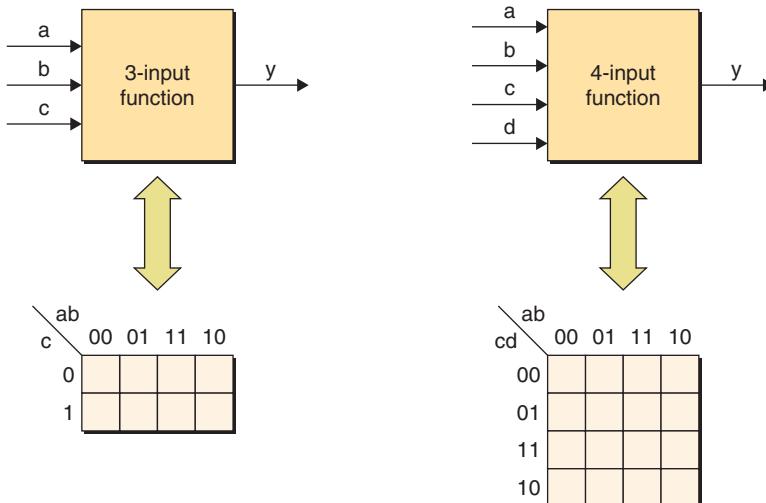
Karnaugh map for a 2-input AND function.

The Karnaugh map comprises a box for every line in the truth table. The binary values above the boxes are those associated with the a and b inputs. Unlike a truth table, in which the input values typically follow a binary sequence, the Karnaugh map's input values must be ordered such that the values for adjacent columns vary by only a single bit: for example, 00_2 , 01_2 , 11_2 , and 10_2 . This ordering is known as a *Gray code*,¹ and it is a key factor with regard to the way in which Karnaugh maps work.

The y column in the truth table shows all the 0 and 1 values associated with the gate's output. Similarly, all of the output values could be entered into the Karnaugh map's boxes. For clarity, however, it is common for only a single set of values to be used (typically the 1s).

Similar maps can be constructed for 3-input and 4-input functions. In the case of a 4-input map, the values associated with the c and d inputs must also be ordered as a Gray code: that is, they must be ordered in such a way that the values for adjacent rows vary by only a single bit (Figure 10.2).

¹Gray codes are introduced in more detail in Appendix D: *Gray Codes*.

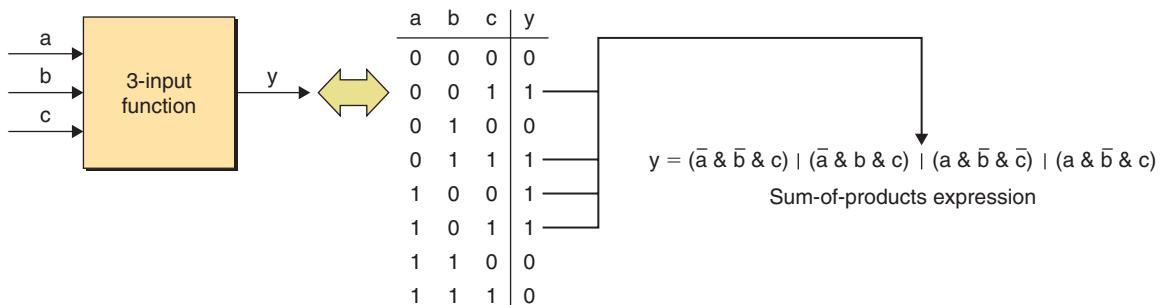
**FIGURE 10.2**

Generic Karnaugh maps for 3- and 4-input functions.

MINIMIZATION USING KARNAUGH MAPS

Karnaugh maps often prove useful in the simplification and minimization of Boolean functions. Consider an example 3-input function represented as a black box with an associated truth table (Figure 10.3).²

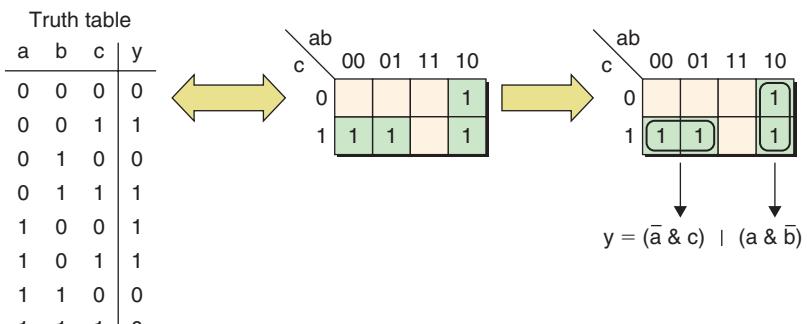
The equation extracted from the truth table in sum-of-products form contains four minterms,³ one for each of the 1s assigned to the output. Algebraic simplification techniques could be employed to minimize this equation, but this

**FIGURE 10.3**

Example 3-input function.

²The values assigned to output y in the truth table were selected randomly and have no significance beyond the purposes of this example.

³The concepts of minterms and maxterms were introduced in *Chapter 9: Boolean Algebra*.

**FIGURE 10.4**

Karnaugh map minimization of example 3-input function.

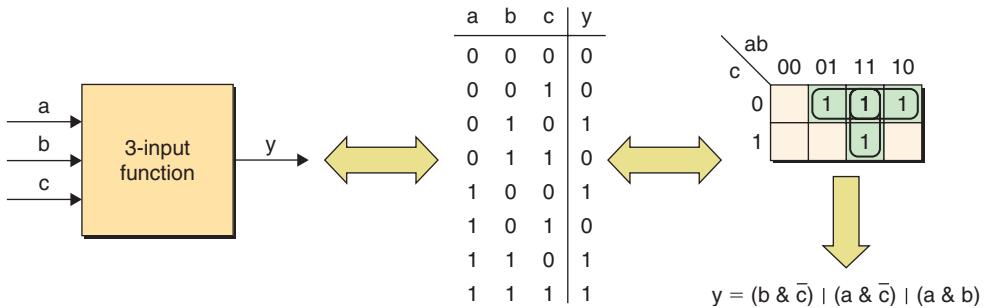
would necessitate every minterm being compared to each of the others, which can be somewhat time-consuming.

This is where Karnaugh maps leap onto the stage with a fanfare of trumpets. The 1s assigned to the Karnaugh map's boxes represent the same minterms as the 1s in the truth table's output column; however, as the input values associated with each row and column in the map differ by only one bit, any pair of horizontally or vertically adjacent boxes corresponds to minterms that differ by only a single variable. Such pairs of minterms can be grouped together and the variable that differs can be discarded, leaving a much-simplified equation (Figure 10.4).

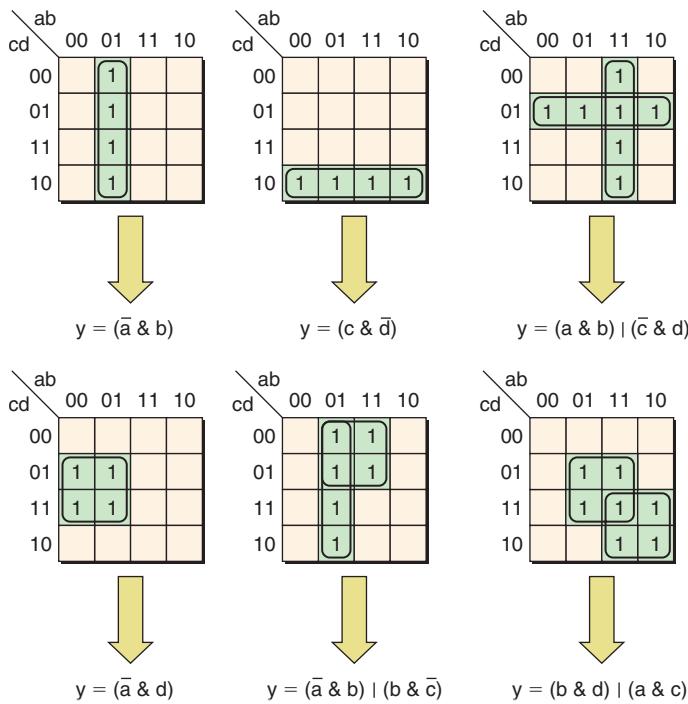
In the case of the horizontal group, input a is 0 for both boxes, input c is 1 for both boxes, and input b is 0 for one box and 1 for the other. Thus, for this group, changing the value on b does not affect the value of the output. This means that b is redundant and can be discarded from this group. Similarly, in the case of the vertical group, input a is 1 for both boxes, input b is 0 for both boxes, and input c is 0 for one box and 1 for the other. Thus, input c is redundant for this group and can be discarded.

GROUPING MINTERMS

In the case of a 3-input Karnaugh map, any two horizontally or vertically adjacent minterms, each composed of three variables, can be combined to form a new product term composed of only two variables. Similarly, in the case of a 4-input map, any two adjacent minterms, each composed of four variables, can be combined to form a new product term composed of only three variables. Additionally, the 1s associated with the minterms can be used to form multiple groups. For example, consider the 3-input function shown in Figure 10.5, in which the minterm corresponding to a = 1, b = 1, and c = 0 is common to three groups.

**FIGURE 10.5**

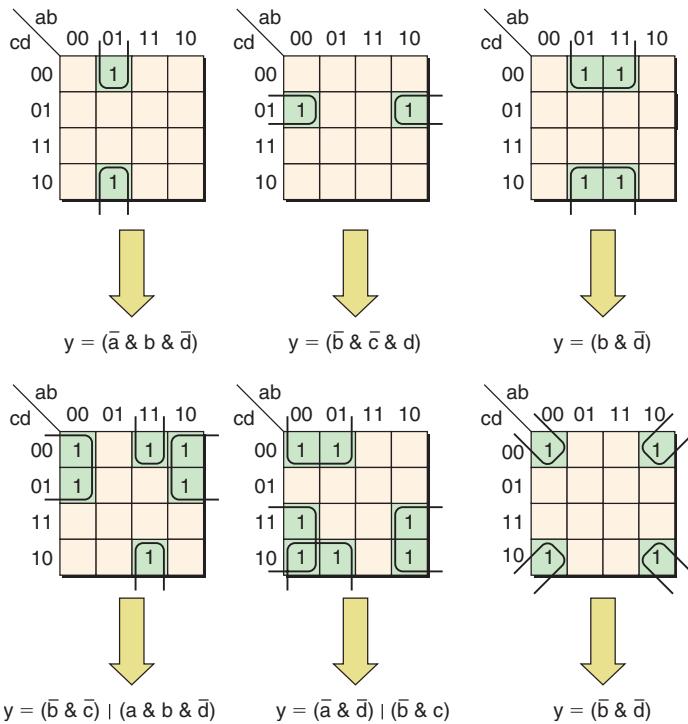
Using the same minterm to form multiple Karnaugh map groups.

**FIGURE 10.6**

Karnaugh map groupings of four adjacent minterms.

Groupings can also be formed from four adjacent minterms, in which case two redundant variables can be discarded. Consider some 4-input Karnaugh Map examples (Figure 10.6).

In fact, any group of 2^n adjacent minterms can be gathered together (where n is a positive integer). For example, $2^1 =$ two minterms, $2^2 = 2 \times 2 =$ four minterms, $2^3 = 2 \times 2 \times 2 =$ eight minterms, and so forth.

**FIGURE 10.7**

Some additional Karnaugh map grouping possibilities.

As was noted earlier, Karnaugh map input values are ordered so that the values associated with adjacent rows and columns differ by only a single bit. One result of this ordering is that the top and bottom rows are also separated by only a single bit (it may help to visualize the map rolled into a *horizontal cylinder* such that the top and bottom edges are touching). Similarly, the left and right columns are separated by only a single bit (in this case it may help to visualize the map rolled into a *vertical cylinder* such that the left and right edges are touching). This leads to some additional groupings, a few of which are shown in Figure 10.7.

Note especially the last example. Diagonally adjacent minterms generally cannot be used to form a group; however, remembering that the left-right columns and the top-bottom rows are logically adjacent, this means that the four corner minterms are also logically adjacent, which in turn means that they can be used to form a single group.

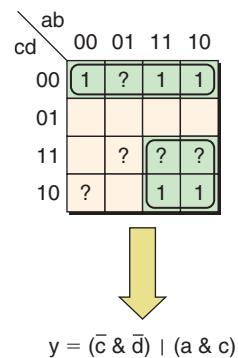
INCOMPLETELY SPECIFIED FUNCTIONS

In certain cases a function may be incompletely specified: that is, the output may be undefined for some of the input combinations. If, for example,

the designer knows that certain input combinations will never occur, then the value assigned to the output for these combinations is irrelevant. Alternatively, for some input combinations, the designer may simply not care about the value on the output. In both cases, the designer can represent the output values associated with the relevant input combinations as question marks in the Karnaugh map (Figure 10.8).

The ? characters indicate *don't care* states, which can be considered to represent either 0 or 1 values at the designer's discretion. In the example shown in Figure 10.8, we have no interest in the ? character in the box referenced by $a = 0, b = 0, c = 1, d = 0$ or the ? character in the box at $a = 0, b = 1, c = 1, d = 1$, because neither of these can be used to form a larger group. However, if we decide that the other three ? characters are going to represent 1 values, then they can be used to form larger groups, which allows us to minimize the function to a greater degree than would otherwise be possible.

It should be noted that many electronics references use X characters to represent *don't care* states. Unfortunately, this may lead to confusion, as some computer-aided design tools (such as logic simulators) use X characters to represent *don't know* states. Unless otherwise indicated, this book will use ? and X to represent *don't care* and *don't know* states, respectively.

**FIGURE 10.8**

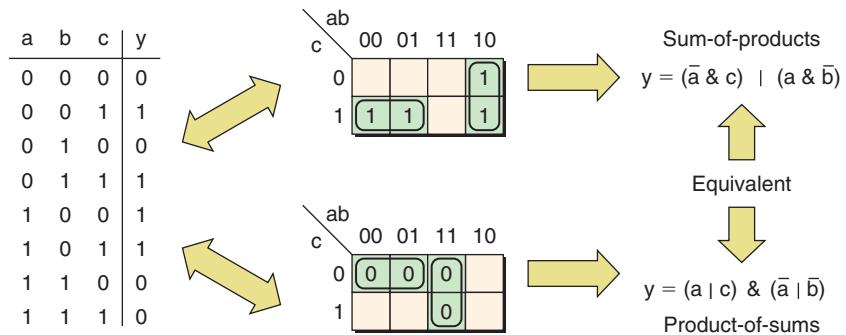
Karnaugh map for an incompletely specified function.

POPULATING MAPS USING 0s VERSUS 1s

When we were extracting Boolean equations from truth tables in the previous chapter, we noted that in the case of a function whose output is logic 1 fewer times than it is logic 0, it is generally easier to extract an equation in the *sum-of-products* form. By comparison, if the output is logic 0 fewer times than it is logic 1, it is generally easier to extract an equation in the *product-of-sums* form.

The same thing applies to a Karnaugh map. If the output is logic 1 fewer times than it is logic 0, then it's probably going to be a lot easier to populate the map using logic 1s. Alternatively, if the output is logic 0 fewer times than it is logic 1, then populating the map using logic 0s may be a better idea.

When a Karnaugh Map is populated using the 1s assigned to the truth table's output, the resulting Boolean expression is extracted from the map in *sum-of-products* form. By comparison, if the Karnaugh Map is populated using the 0s assigned to the truth table's output, then the groupings of 0s are used to generate expressions in *product-of-sums* form (Figure 10.9).

**FIGURE 10.9**

Populating Karnaugh maps with 0s versus 1s.

Although the *sum-of-products* and *product-of-sums* expressions appear to be somewhat different, they do produce identical results. The expressions can be shown to be equivalent using algebraic means, or by constructing truth tables for each expression and comparing the outputs.

Karnaugh maps are most often used to represent 3-input and 4-input functions. It is possible to create similar maps for 5-input and 6-input functions, but these maps can quickly become unwieldy and difficult to use. Thus, the Karnaugh technique is generally not considered to have any application for functions with more than six inputs.

CHAPTER 11

Slightly More Complex Functions

FIRST GATHER A BUCKET OF LOGIC GATES

The primitive logic functions NOT, AND, OR, NAND, NOR, XOR, and XNOR can be connected together to build slightly more complex functions; in turn, these may be used as building blocks in yet more sophisticated systems. The examples introduced in this chapter were selected because they occur commonly in designs, are relatively simple to understand, and will prove useful in later discussions.

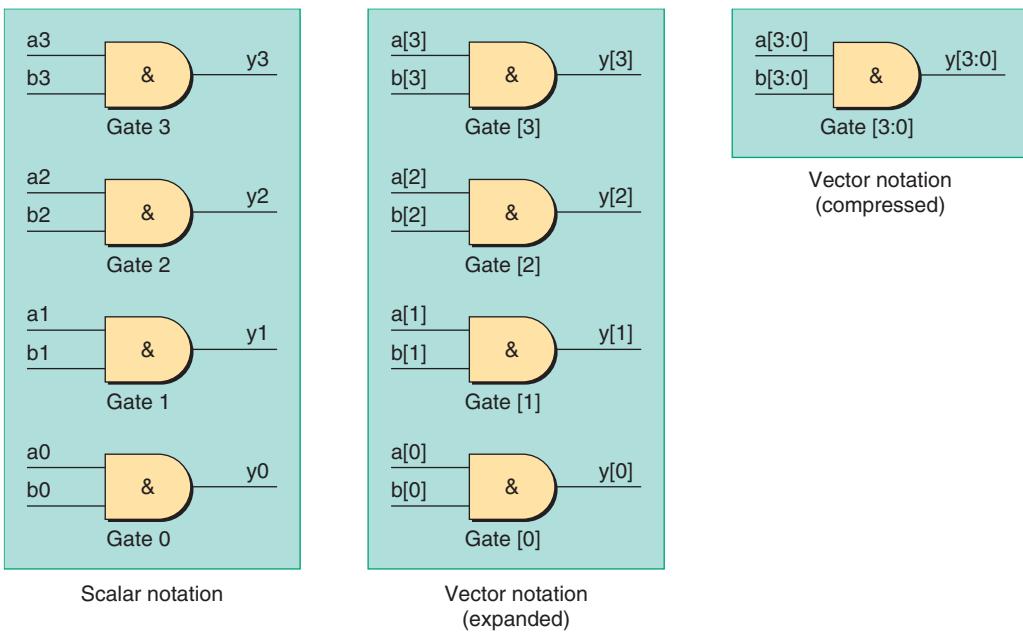
125

SCALAR VERSUS VECTOR NOTATION

A single signal carrying one bit of binary data is known as a *scalar* entity. A set of signals carrying similar data can be gathered together into a group known as a *vector* (see also the glossary definition of *vector*).

Consider the circuit fragments shown in Figure 11.1. Each of these fragments represents four 2-input AND gates. In the case of the scalar notation, each signal is assigned a unique name: for example, a₃, a₂, a₁, and a₀. By comparison, when using vector notation, a single name is applied to a group of signals, and individual signals within the group are referenced by means of an index: for example, a[3], a[2], a[1], and a[0]. This means that if we were to see a schematic (circuit) diagram containing two signals called a₃ and a[3], we would understand this to represent two completely different signals (the former being a scalar named “a₃” and the latter being an element of a vector named “a”).

A key advantage of vector notation is that it allows all of the signals comprising the vector to be easily referenced in a single statement: for example, a[3:0]. Thus, vector notation can be used to reduce the size and complexity of a circuit diagram while at the same time increasing its clarity.

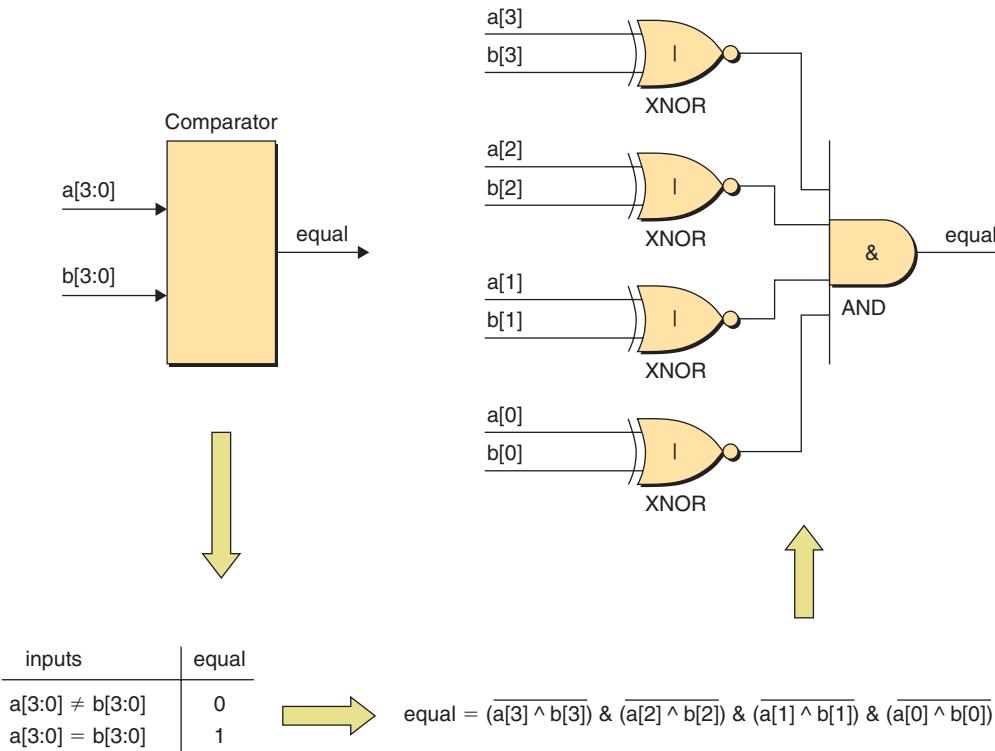
**FIGURE 11.1**

Scalar versus vector notations.

EQUALITY COMPARATORS

In some designs it may be necessary to compare two sets of binary values to see if they contain the same data. Consider a function used to compare two 4-bit vectors: $a[3:0]$ and $b[3:0]$. A scalar output called *equal* is to be set to logic 1 if each bit in $a[3:0]$ is equal to its corresponding bit in $b[3:0]$. That is, the vectors are equal if $a[3] = b[3]$, $a[2] = b[2]$, $a[1] = b[1]$, and $a[0] = b[0]$ (Figure 11.2).

The values on $a[3]$ and $b[3]$ are compared using a 2-input XNOR gate. As we know from *Chapter 5: Primitive Logic Functions*, and *Chapter 6: Using Transistors to Build Logic Gates*, if the values on the inputs to an XNOR are the same (both 0s or both 1s), then its output will be 1; but if the values on its inputs are different, the output will be 0. Similar comparisons are performed between the other inputs: $a[2]$ with $b[2]$, $a[1]$ with $b[1]$, and $a[0]$ with $b[0]$. The final AND gate is used to gather the results of the individual comparisons. If all the inputs to the AND gate are 1, the two vectors are the same and the output of the AND gate will be 1. Correspondingly, if any of the inputs to the AND gate are 0, the two vectors are different and the output of the AND gate will be 0.

**FIGURE 11.2**

4-bit equality comparator.

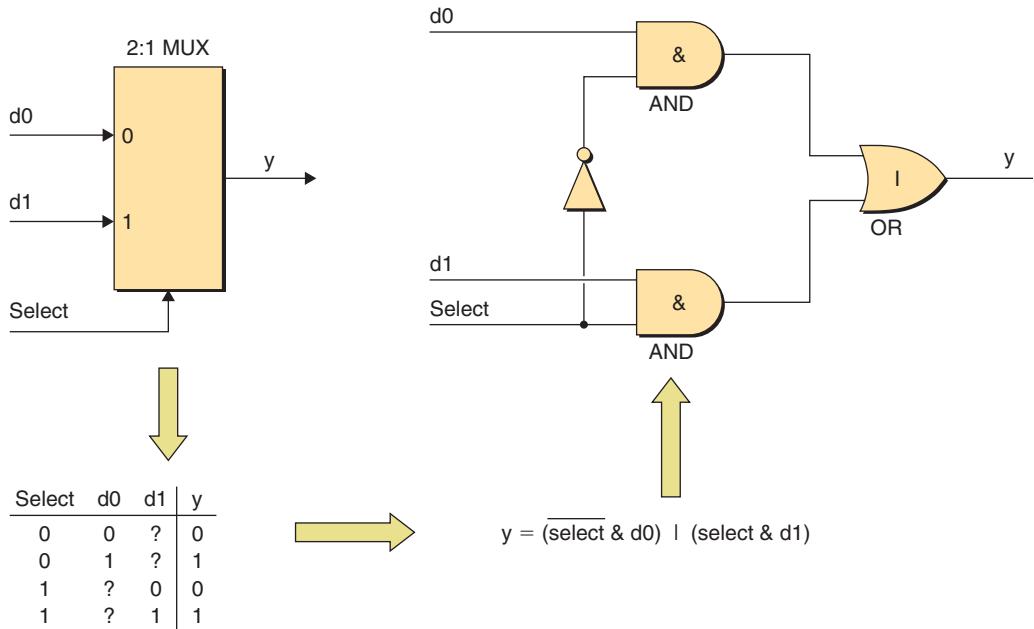
Of course, a similar result could have been obtained by replacing the XNORs with XORs and the AND with a NOR, and either of these implementations could be easily extended to accommodate input vectors of greater width.

MULTIPLEXERS

A multiplexer uses a binary value, or *address*, to select between a number of inputs and to convey the data from the selected input to the output. For example, consider a 2:1 ("two-to-one") multiplexer as illustrated in Figure 11.3.

The 0 and 1 annotations on the multiplexer symbol represent the possible values of the select input and are used to indicate which data input will be selected.

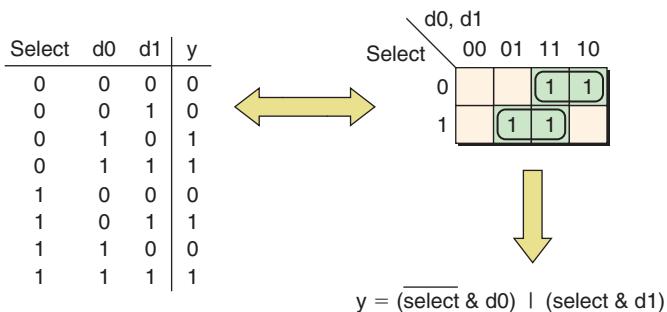
The ? characters in the truth table indicate *don't care* states. When the select input is presented with a logic 0, the output from the function depends only on the value of the d0 data input, and we don't care about the value on the d1 input. Similarly, when select is presented with a logic 1, the output from the function depends only on the value of the d1 data input, and we don't care

**FIGURE 11.3**

2:1 multiplexer.

about the value on the d_0 input. The use of don't care states reduces the size of the truth table, better represents the operation of this particular function, and simplifies the extraction of the *sum-of-products* expression describing this function because the don't cares can be ignored.

Of course, an identical result could have been achieved using a full truth table combined with a Karnaugh map minimization (Figure 11.4).¹

**FIGURE 11.4**

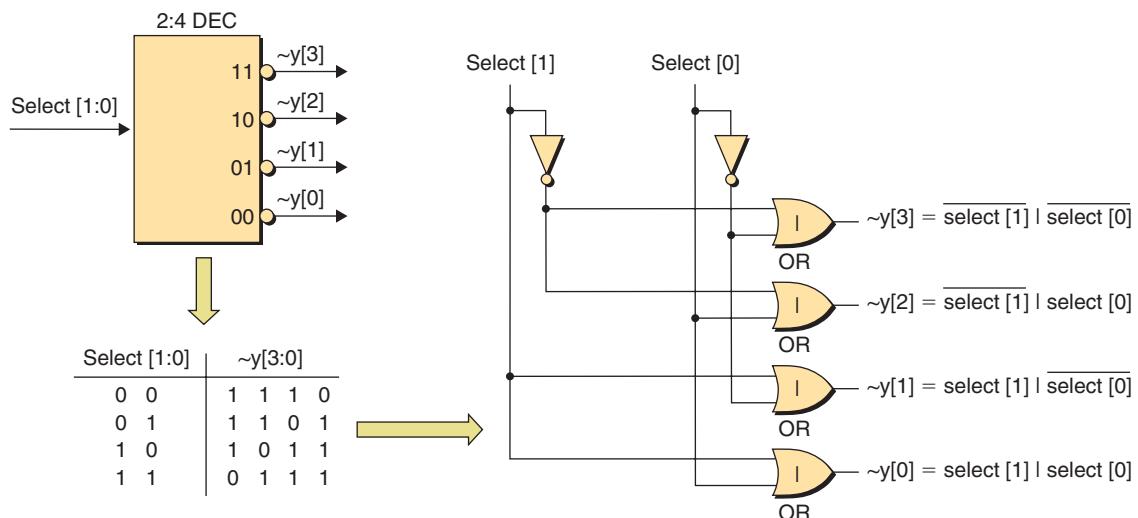
Using a Karnaugh map to derive the 2:1 multiplexer equation.

¹Karnaugh map minimization techniques were introduced in *Chapter 10: Karnaugh Maps*.

Larger multiplexers are also common in designs: for example, 4:1 multiplexers with four data inputs feeding one output, and 8:1 multiplexers with eight data inputs feeding one output. In the case of a 4:1 multiplexer, we will require two select inputs to choose between the four data inputs (using binary patterns of 00, 01, 10, and 11). Similarly, in the case of an 8:1 multiplexer, we will require three select inputs to choose between the eight data inputs (using binary patterns of 000, 001, 010, 011, 100, 101, 110, and 111).

DECODERS

A decoder uses a binary value, or *address*, to select between a number of outputs and to assert the selected output by placing it in its *active state*. For example, consider a 2:4 ("two-to-four") decoder as illustrated in Figure 11.5.



The 00, 01, 10, and 11 annotations on the decoder symbol represent the possible values that can be applied to the select [1:0] inputs and are used to indicate which output will be asserted.

The truth table shows that when a particular output is selected, it is asserted to a 0, and when that output is not selected, it returns to a 1. Because the outputs are asserted to 0s, this device is said to have *active-low* outputs. An *active-low* signal is one whose active state is considered to be logic 0.²

FIGURE 11.5
2:4 decoder with active-low outputs.

²Similar functions can be created with *active-high outputs*, which mean that when an output is selected it is asserted to a logic 1.

The active-low nature of this particular function is also indicated by the bubbles (small circles) associated with the symbol's outputs. Also, the tilde “~” characters prefixing the output names $\sim y[3]$, $\sim y[2]$, $\sim y[1]$, and $\sim y[0]$ are used to indicate that these signals are active-low (the use of tilde characters is discussed in more detail in *Appendix A: Assertion-Level Logic*).

Additionally, from our discussions in *Chapters 9: Boolean Algebra* and *10: Karnaugh Maps*, we know that as each output is 0 for only one input combination, it is simpler to extract these equations in *product-of-sums* form. (Observe the horizontal bars shown over some of the signal names in the equations in Figure 11.5. These bars indicate that their associated signals have been logically inverted; that is, that each of these signals is coming from the output of the relevant NOT gate.)

Larger decoders are also commonly used in designs: for example, 3:8 decoders with three select inputs and eight outputs, 4:16 decoders with four select inputs and sixteen outputs, etc.

TRI-STATE FUNCTIONS

There is a special category of gates called *tri-state functions* whose outputs can adopt three states: 0, 1, and Z. Let's first consider a simple tri-state buffer (Figure 11.6).

The tri-state buffer's symbol is based on a standard buffer with an additional control input known as the *enable*. The active-low nature of this particular function's enable is indicated by the bubble associated with this input on the symbol, and by the tilde “~” character in its name, $\sim \text{enable}$. (Similar functions with active-high enables are also commonly used in designs.)

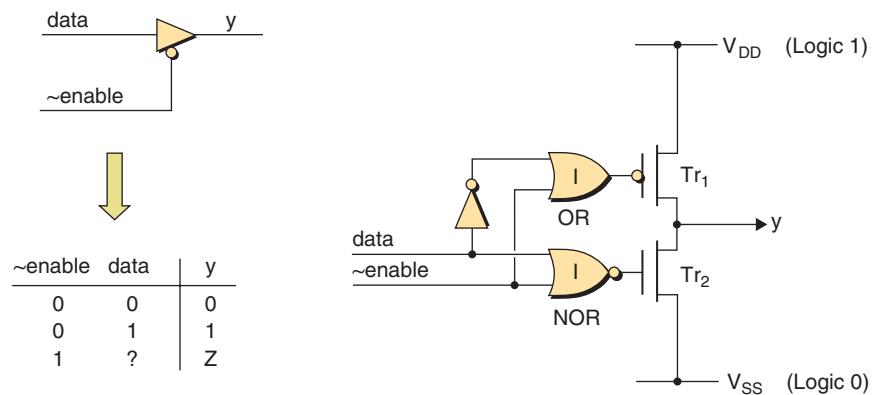


FIGURE 11.6
Tri-state buffer with active-low enable.

The Z character in the truth table represents a state known as *high-impedance* in which the gate is not driving either of the standard 0 or 1 values. In fact, in the high-impedance state the gate is effectively disconnected from its output.

Although Boolean Algebra is not well equipped to represent the Z state, the implementation of the tri-state buffer is relatively easy to understand. When the ~enable input is presented with a logic 1 (its inactive state), the output of the OR gate is forced to logic 1 and the output of the NOR gate is forced to logic 0, thereby turning both the T_{r_1} and T_{r_2} transistors OFF, respectively. With both transistors turned OFF, the output y is disconnected from V_{DD} and V_{SS} , and is therefore in the high-impedance state.

By comparison, when the ~enable input is presented with a logic 0 (its active state), the outputs of the OR and NOR gates are determined by the value on the data input. The circuit is arranged so that only one of the T_{r_1} and T_{r_2} transistors can be ON at any particular time. If the data input is presented with a logic 1, transistor T_{r_1} is turned ON, thereby connecting output y to V_{DD} (which equates to logic 1). By comparison, if the data input is presented with a logic 0, transistor T_{r_2} is turned ON, thereby connecting output y to V_{SS} (which equates to logic 0).

Tri-state buffers can be used in conjunction with additional control logic to allow the outputs of multiple devices to drive a common signal. For example, consider the simple circuit shown in Figure 11.7.

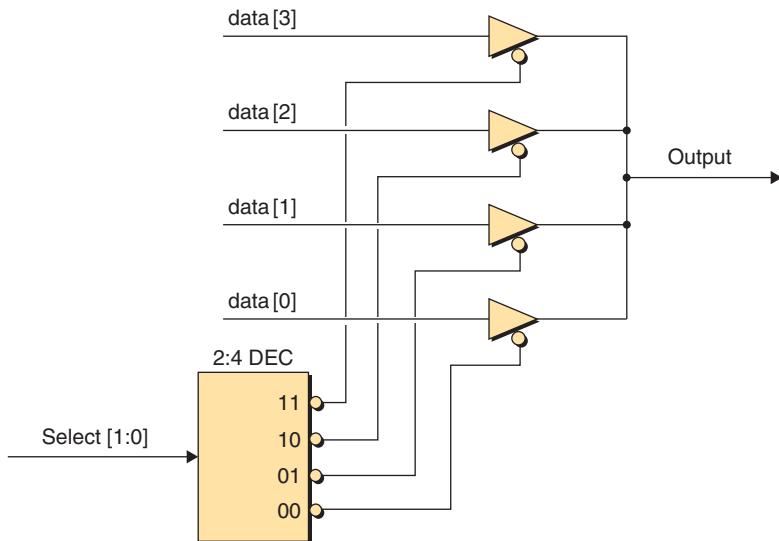


FIGURE 11.7

Using tri-state buffers to allow multiple devices to drive a common signal.

The use of a 2:4 decoder with active-low outputs ensures that only one of the tri-state buffers is enabled at any time. The enabled buffer will propagate the data on its input to the common output, while the remaining buffers will be forced into their tri-state condition.

In hindsight, it now becomes obvious that the standard primitive gates (AND, OR, NAND, NOR, etc.) depend on internal Z states to function (when any transistor is turned OFF, its output effectively goes to a Z state). However, the standard primitive gates are constructed in such a way that at least one of the transistors connected to the output is turned ON, which means that the output of a standard gate is always driving either a logic 0 or a logic 1.

COMBINATIONAL VERSUS SEQUENTIAL FUNCTIONS

Logic functions are categorized as being either *combinational* (sometimes referred to as *combinatorial*) or *sequential*. In the case of a *combinational* function, the logic values on that function's outputs are directly related to the current *combination* of values on its inputs. All of the previous example functions have been of this type.

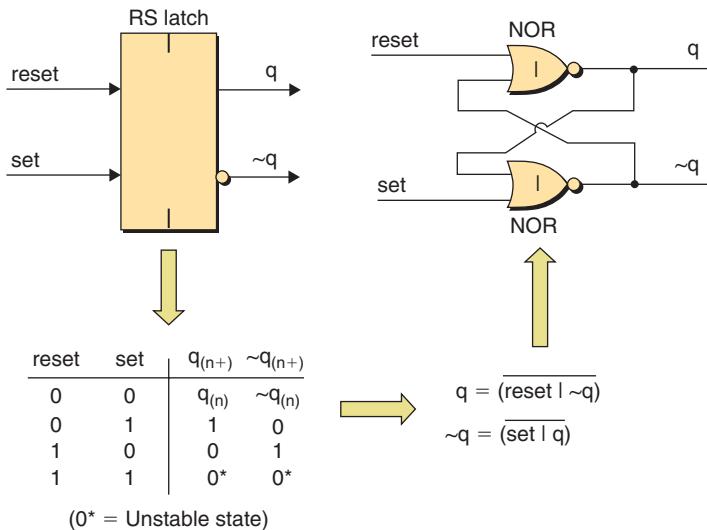
By comparison, in the case of a *sequential* function, the logic values on that function's outputs depend not only on its current input values, but also on previous input values. That is, the output values depend on a *sequence* of input values. Because sequential functions remember previous input values, they may also be referred to as *memory elements*.

RS LATCH (NOR IMPLEMENTATION)

One of the simpler sequential functions is that of an *RS latch* ("Reset-Set latch"), which can be implemented using two NOR gates connected in a back-to-back configuration (Figure 11.8). In this NOR implementation, both reset and set inputs are *active-high* (this is indicated by the lack of bubbles associated with these inputs on the symbol). The names of these inputs indicate the effect they have on the q output; when reset is active q is reset to logic 0, and when set is active q is set to logic 1.

The q and $\sim q$ outputs are known as the *true* and *complementary* outputs, respectively.³ In the latch's normal mode of operation, the value on $\sim q$ is the inverse, or complement, of the value on q. This is also indicated by the bubble associated

³In this case, the tilde “~” character prefixing the output name, $\sim q$ is used to indicate that this signal is a complementary output. Once again, the use of tilde characters is discussed in detail in *Appendix A: Assertion-Level Logic*.

**FIGURE 11.8**

RS latch: NOR-based implementation.

with the $\sim q$ output on the symbol. The only time $\sim q$ is not the inverse of q occurs when both reset and set are active at the same time (this unstable state is discussed in more detail below).

The truth table column labels $q_{(n+)}$ and $\sim q_{(n+)}$ indicate that these columns refer to the *future values* on the outputs. The $n+$ subscripts represent some future time, or “now-plus.” By comparison, the labels $q_{(n)}$ and $\sim q_{(n)}$ used in the body of the truth table indicate the *current values* on the outputs. In this case the n subscripts represent the current time, or “now.” Thus, the first row in the truth table indicates that when both reset and set are in their inactive states (logic 0s), the future values on the outputs will be the same as their current values.

The secret of the RS latch’s ability to remember previous input values is based on a technique known as *feedback*. This refers to the feeding back of the outputs as additional inputs into the function. In order to see how this works, let’s assume that both the reset and set inputs are initially in their inactive states, but that some previous input sequence placed the latch in its *set condition*; that is, q is logic 1 and $\sim q$ is logic 0. Now consider what occurs when the reset input is placed in its active state and then returns to its inactive state (Figure 11.9).

As a reminder, if *any* input to a NOR gate is logic 1, its output will be forced to logic 0; it’s only if both inputs to the NOR are 0 that the output will be 1. Thus, when reset is placed in its active (logic 1) state , the q output from the first gate is forced to 0 . This 0 on q is fed back into the second gate

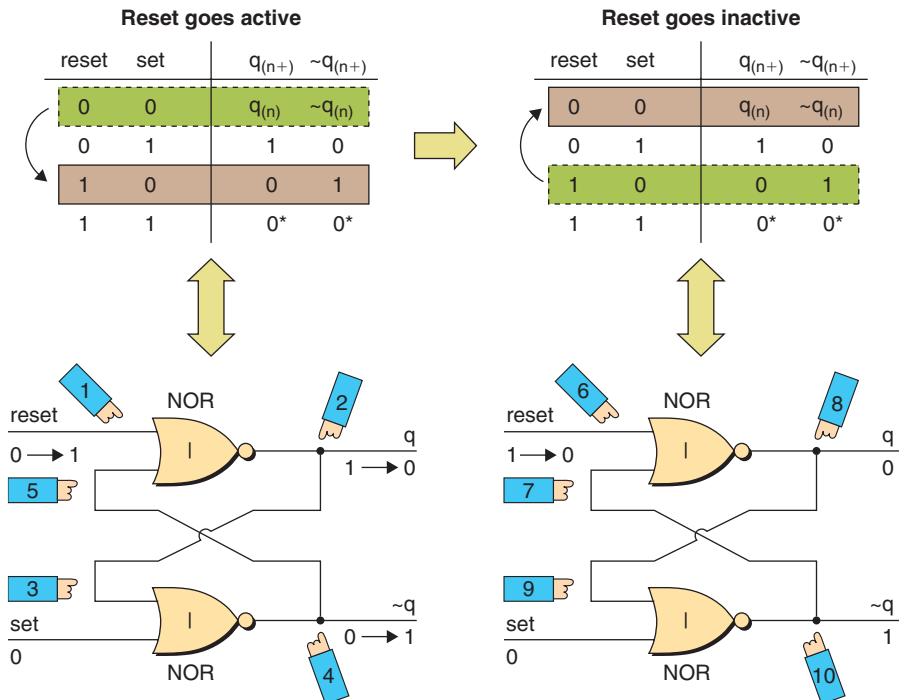


FIGURE 11.9
RS latch: Reset input goes active then inactive.

3 and, as *both* inputs to this gate are now 0, the $\sim q$ output is forced to 1 4 . The key point to note here is that the 1 on $\sim q$ is now fed back into the first gate 5 .

When the **reset** input returns to its inactive (logic 0) state 6 , the 1 from the $\sim q$ output continues feeding back into the first gate 7 , which means that the **q** output continues to be forced to 0 8 . Similarly, the 0 on **q** continues feeding back into the second gate 9 , and as both of this gate's inputs are now at 0, the $\sim q$ output continues to be forced to 1 10 . The end result is that the 1 from 7 causes the 0 at 8 which is fed back to 9 , and the 0 on the **set** input combined with the 0 from 9 causes the 1 at 10 which is fed back to 7 . (Phew!)

Thus, the latch⁴ has now been placed in its *reset condition*, and a self-sustaining loop has been established. Even though both the **reset** and **set** inputs are now inactive, the **q** output remains at 0, thereby indicating that **reset** was the last input to be in its active state. Once the function has been placed in its *reset condition*,

⁴The term *latch*—which is commonly associated with a fastening for a door or gate—comes from the Middle English *lacchen*, which comes from the Old English *loeccan*, meaning “to seize” or “to take hold of.”

any subsequent activity on the reset input will have no effect on the outputs, which means that the only way to affect the function is by means of its set input.

Now consider what occurs when the set input is placed in its active state and then returns to its inactive state (Figure 11.10).

When the set input is placed in its active (logic 1) state **11**, the $\sim q$ output from the second gate is forced to 0 **12**. This 0 on $\sim q$ is fed back into the first gate **13** and, as both inputs to this gate are now 0, the q output is forced to 1 **14**. The key point to note is that the 1 on q is now fed back into the second gate **15**.

When the set input returns to its inactive (logic 0) state **16**, the 1 from the q output continues feeding back to the second gate **17**, whose $\sim q$ output continues to be forced to 0 **18**. Similarly, the 0 on the $\sim q$ output continues feeding back into the first gate **19**, and the q output continues to be forced to 1 **20**. The end result is that the 1 at **17** causes the 0 at **18** that is fed back to **19**, and the 0 on the reset input combined with the 0 at **19** causes the 1 at **20** which is fed back to **17**. (Are we having fun yet?)

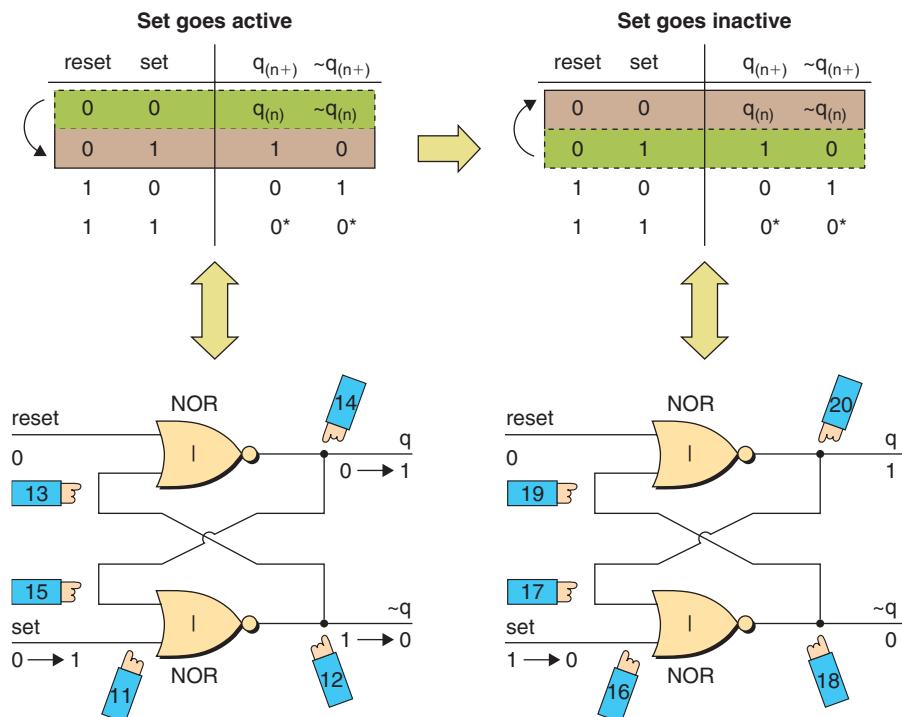


FIGURE 11.10
RS latch: Set input goes active then inactive.

Thus, the latch has been returned to its *set condition* and, once again, a self-sustaining loop has been established. Even though both the reset and set inputs are now inactive, the q output remains at 1, thereby indicating that set was the last input to be in its active state. Once the function has been placed in its *set condition*, any subsequent activity on the set input will have no effect on the outputs, which means that the only way to affect the function is by means of its reset input.

The unstable condition indicated by the fourth row of the RS latch's truth table occurs when both the reset and set inputs are active at the same time. In this case, problems may occur when both reset and set return to their inactive states simultaneously or too closely together (Figure 11.11).

When both reset and set are active at the same time, the 1 on reset [21] forces the q output to 0 [22] and the 1 on set [23] forces the $\sim q$ output to 0 [24]. The 0 on q is fed back to the second gate [25], and the 0 on $\sim q$ is fed back to the first gate [26].

Now, consider what occurs when reset and set go inactive simultaneously ([27] and [28], respectively). When the new 0 values on reset and set are combined with the 0 values fed back from q [29] and $\sim q$ [30], each gate

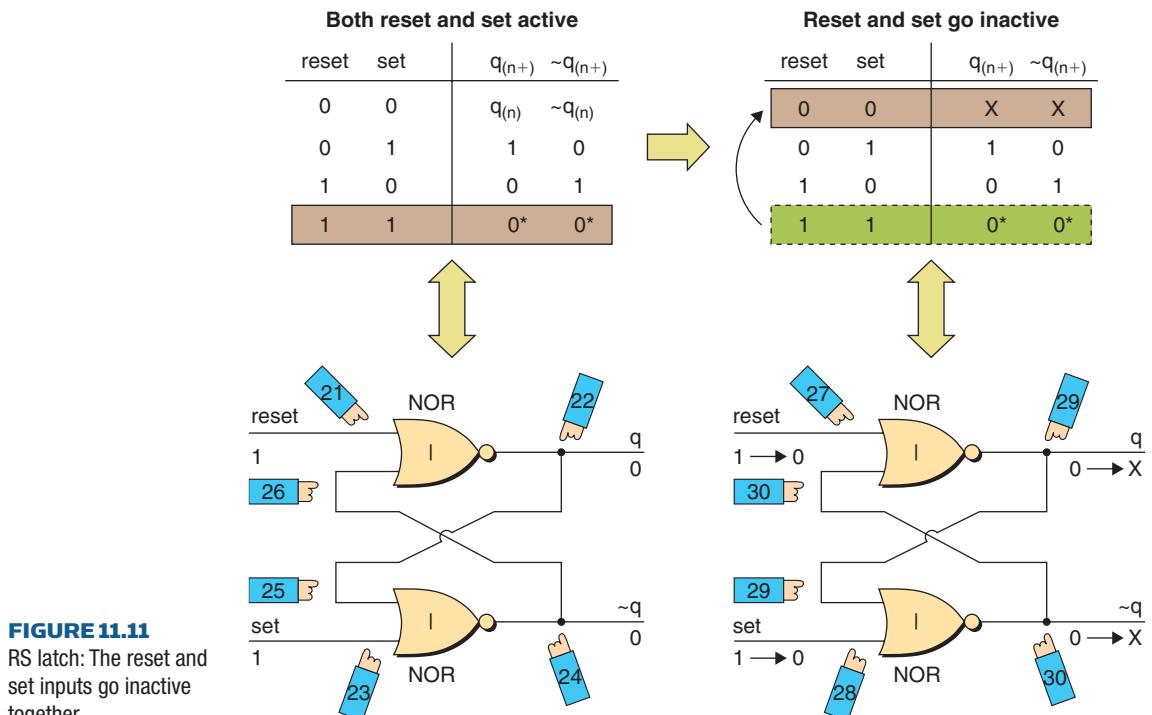


FIGURE 11.11
RS latch: The reset and set inputs go inactive together.

initially sees both of its inputs at 0 and therefore both gates attempt to drive their outputs to 1. After any delays associated with the gates have been satisfied, both of the outputs will indeed go to 1.

When the output of the first gate goes to 1, this value is fed back to the input of the second gate. At the same time this is happening, the output of the second gate goes to 1, and this value is fed back to the input of the first gate. Each gate now has its fed-back input at 1, and both gates therefore attempt to drive their outputs to 0. As we see, the circuit has entered a *metastable condition* in which the outputs oscillate between 0 and 1 values.

If both halves of the function were exactly the same, these metastable oscillations would continue indefinitely. But there will always be some differences (no matter how small) between the gates and their delays, and the function will eventually collapse into either its *reset condition* or its *set condition*. As there is no way to predict the final values on the q and $\sim q$ outputs, they are indicated as being in X ("don't know") states (29 and 30). These X states will persist until a valid input sequence occurs on either the reset or set inputs.

RS LATCH (NAND IMPLEMENTATION)

An alternative implementation for an RS latch can be realized using two NAND gates connected in a back-to-back configuration (Figure 11.12).

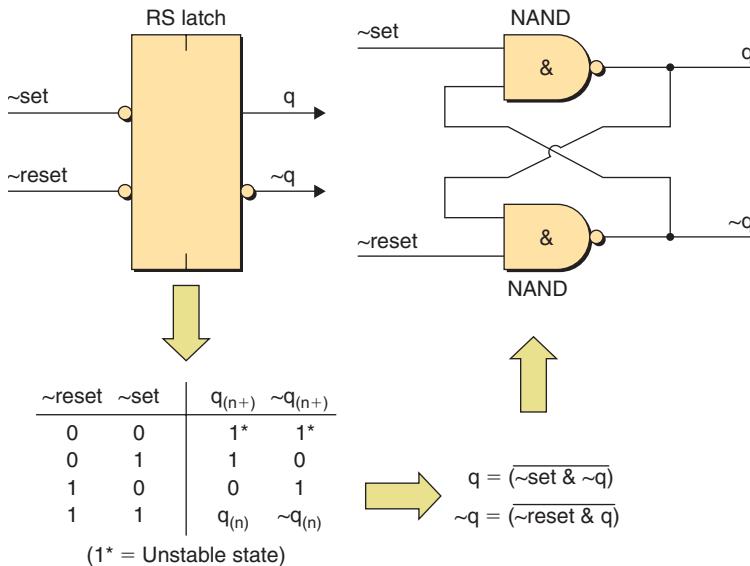


FIGURE 11.12

RS latch: NAND-based implementation.

In a NAND implementation, both the $\sim\text{reset}$ and $\sim\text{set}$ inputs are *active-low* (this is indicated by the bubbles associated with these inputs on the symbol and by the tilde “ \sim ” characters in their names). As a reminder, if *any* input to a NAND is 0, the output will be forced to 1; it’s only if *both* inputs to a NAND are 1 that the output will be 0. Working out how this version of the latch works is left as an exercise to the reader.⁵

D-TYPE LATCHES

A more sophisticated function called a *D-type* (“data-type”) *latch* can be constructed by attaching two AND gates and a NOT gate to the front of an RS latch (Figure 11.13).

The enable input is *active-high* for this configuration, as is indicated by the lack of a bubble on the symbol. When the enable input is placed in its active (logic 1) state, the true and inverted versions of the data input are allowed to propagate through the AND gates and are presented to the back-to-back NOR gates. If the data input changes while enable is still active, the outputs will respond to reflect the new value.

When the enable input returns to its inactive (logic 0) state, it forces the outputs of both ANDs to 0, and any further changes on the data input have no effect.

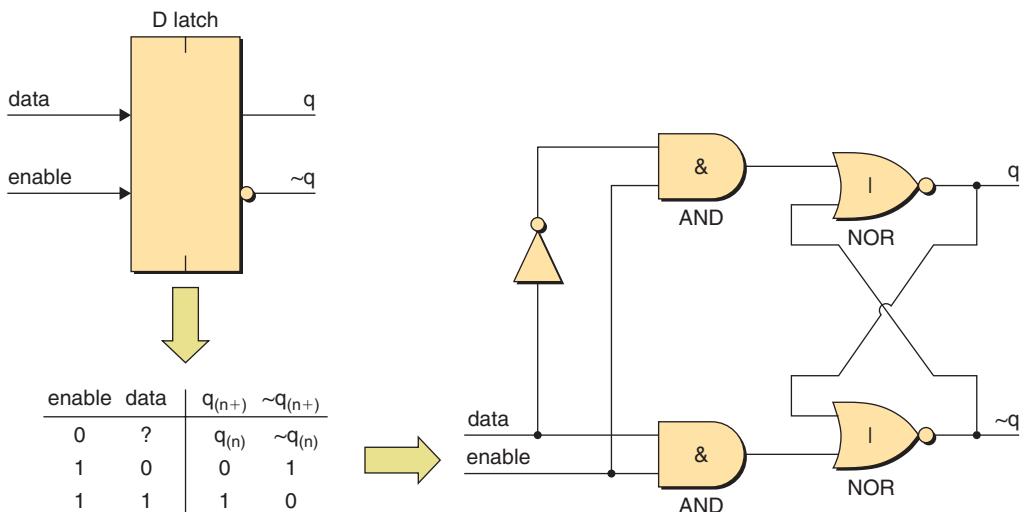
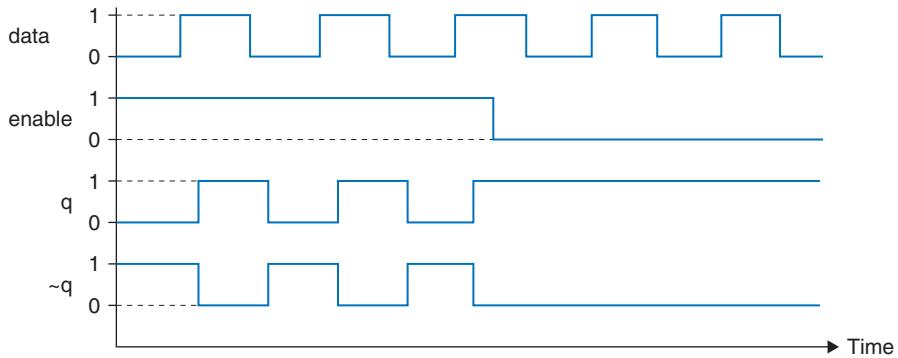


FIGURE 11.13
D-type latch with active-high enable.

⁵This is where we see if you’ve been paying attention (grin).

**FIGURE 11.14**

D-type latch: Waveform for active-high enable.

Thus, the back-to-back NOR gates remember the last value they saw from the data input prior to the enable input going inactive (see also the *Setup and Hold Times* topic at the end of this chapter).

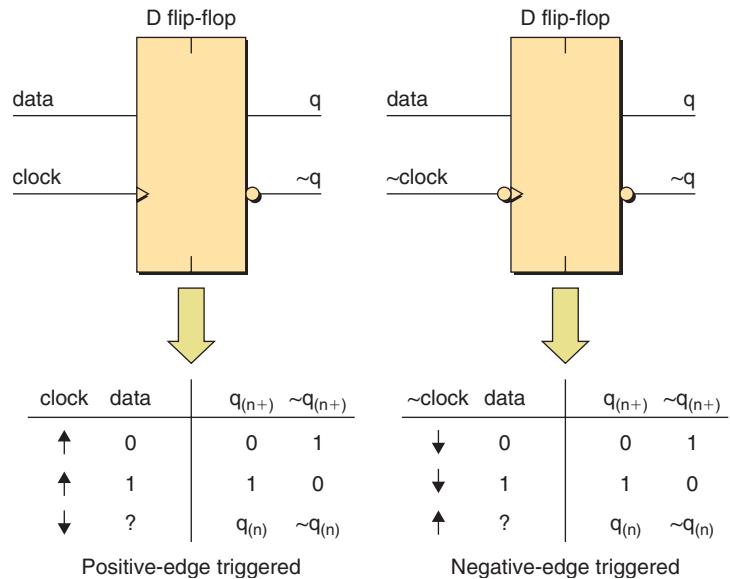
Consider an example waveform (Figure 11.14). While the enable input is in its active state, whatever value is presented to the data input appears on the q output and an inverted version appears on the ~q output. As usual, there will always be some element of delay between changes on the inputs and corresponding responses on the outputs. When enable goes inactive, the outputs remember their previous values and no longer respond to any changes on the data input. As the operation of the device depends on the logic value, or level, on enable, this input is said to be *level-sensitive*.

D-TYPE FLIP-FLOPS

In the case of a *D-type flip-flop* (which may also be referred to as a *register*), the data *appears* to be loaded when a *transition*, or *edge*, occurs on the clock input, which is therefore said to be *edge-sensitive*. (The reason we say “*appears to be loaded when an edge occurs*” is discussed in the following topic.)

A transition from logic 0 to logic 1 is known as a *rising-edge* or a *positive-edge*, while a transition from logic 1 to logic 0 is known as a *falling-edge* or a *negative-edge*. Depending on the implementation, a D-type flip-flop’s clock input may be *positive-edge* or *negative-edge* triggered (Figure 11.15).

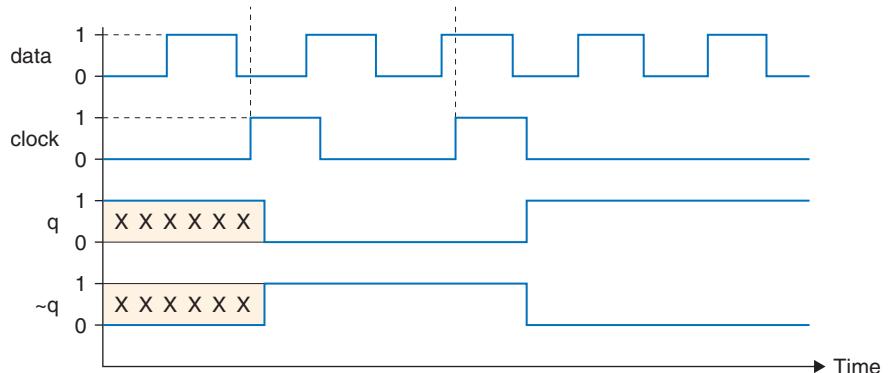
The chevrons (arrows “>”) associated with the clock inputs on the symbols indicate that these are edge-sensitive inputs. A chevron without an associated bobble indicates a *positive-edge clock*, while a chevron with a bobble indicates a *negative-edge clock*. The last rows in the truth tables show that an inactive edge on the clock leaves the contents of the flip-flops unchanged (these cases are often omitted from the truth tables).

**FIGURE 11.15**

D-type flip-flops: Positive and negative-edge triggered.

Consider an example waveform for a positive-edge triggered D-type flip-flop (Figure 11.16). As the observer initially has no knowledge as to the contents of the flop-flop, the q and $\sim q$ outputs commence with X ("don't know") values.

The first rising edge of the clock loads the 0 on the data input into the flip-flop, which (after a small delay) causes q to change to 0 and $\sim q$ to change to 1. The second rising edge of the clock loads the 1 on the data input into the flip-flop; q goes to 1 and $\sim q$ goes to 0.

**FIGURE 11.16**

D-type flip-flop: Waveform for active-high enable

Some flip-flops have an additional input called $\sim\text{clear}$ or $\sim\text{reset}$ which forces q to 0 and $\sim q$ to 1, irrespective of the value on the data input (Figure 11.17). Similarly, some flip-flops have a $\sim\text{preset}$ or $\sim\text{set}$ input, which forces q to 1 and $\sim q$ to 0; and some have both $\sim\text{clear}$ and $\sim\text{preset}$ inputs.

The examples shown in Figure 11.17 reflect active-low $\sim\text{clear}$ inputs, but active-high equivalents are also available. Furthermore, as is illustrated in Figure 11.17, these inputs may be either *asynchronous* or *synchronous*. In the more common asynchronous case, the effect of the $\sim\text{clear}$ input going active is immediate and overrides both the clock and data inputs (the “asynchronous” qualifier reflects the fact that the effect of this input is *not* synchronized to the clock). By comparison, in the synchronous case the effect of the $\sim\text{clear}$ input is synchronized to the active edge of the clock.⁶

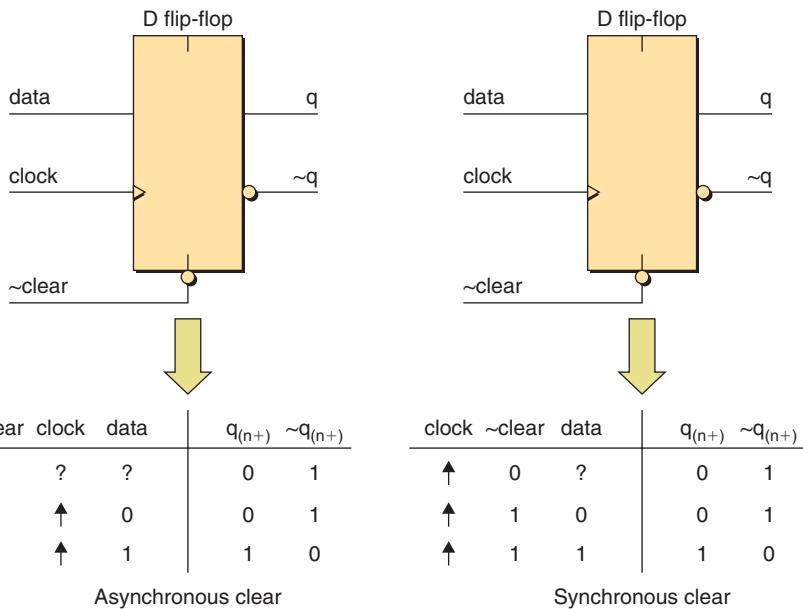


FIGURE 11.17

D-type flip-flops: Asynchronous and synchronous $\sim\text{clear}$ inputs.

⁶The component symbols used in this book are relatively traditional and simple. One disadvantage of this is that, as illustrated in Figure 11.17, there's no way to tell if a clear or preset input is synchronous or asynchronous without also looking at its truth table. There are more modern and sophisticated symbol standards that do cover all eventualities, but their complexity is beyond the scope of this book to explain.

IMPLEMENTING A D-TYPE FLIP-FLOP

There are a number of ways to implement a D-type flip-flop. The most understandable from our point of view would be to use two D-type latches in series (one after the other) as illustrated in Figure 11.18. This is known as a *master-slave* relationship, where the first latch is the “master” and the second is the “slave.” For the purposes of this example we’ll assume that the first latch has an active-low enable and the second has an active-high enable. When both of these signals are connected together, they are presented to the outside world as the clock input.⁷

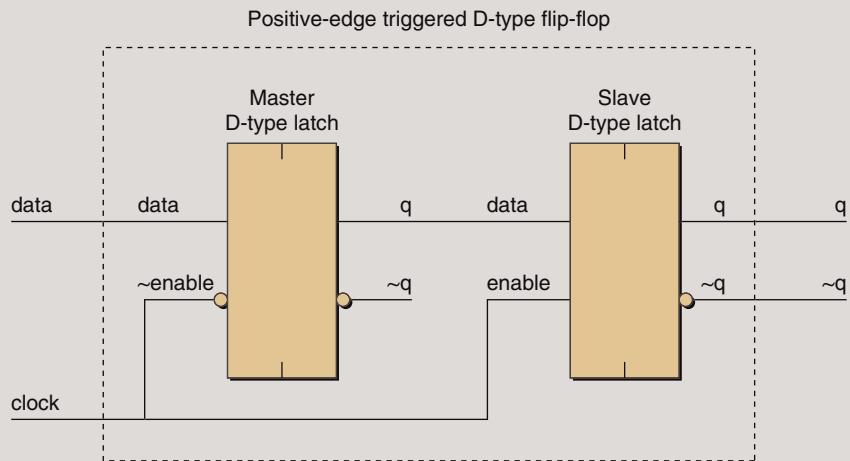


FIGURE 11.18

D-type flip-flop: Positive-edge triggered implementation.

When the clock input is a logic 0, the master latch is enabled and passes whatever value is presented to its data input through to its outputs (only its q output is actually used in this example). Meanwhile, the slave latch is disabled and continues to store (and to output) its existing contents.

When the clock input is subsequently driven to a logic 1, the master latch is disabled and continues to store (and to output) its existing contents. Meanwhile the slave latch is now enabled and passes whatever value is presented to its data input (the value from the output of the master latch) through to its outputs.

Thus, everything is really controlled by voltage levels, but from the outside world it appears that the flip-flop was loaded by a rising-edge on its clock input.

⁷If the first latch and second latches had active-high and active-low enables, respectively, then when we connected these signals together they would be presented to the outside world as the ~clock input.

JK AND T FLIP-FLOPS

The majority of examples in this book are based on D-type flip-flops. However, for the sake of completeness, it should be noted that there are several other flavors of flip-flops available. Two common types are the *JK* and *T* (for “Toggle”) flip-flops (Figure 11.19).

The first row of the JK flip-flop’s truth table shows that when both the *j* and *k* (data) inputs are 0, an active edge on the *clock* input leaves the contents of the flip-flop unchanged. The two middle rows of the truth table show that if the *j* and *k* inputs have opposite values, an active edge on the *clock* input will effectively load the flip-flop (the *q* output) with the value on the *j* input (the $\sim q$ output will take the complementary value). The last line of the truth table shows that when both the *j* and *k* inputs are 1, an active edge on the *clock* causes the outputs to toggle to the inverse of their previous values.⁸ By comparison, the *T* flip-flop doesn’t have any data inputs; the outputs simply toggle to the inverse of their previous values on each active edge of the *clock* input.

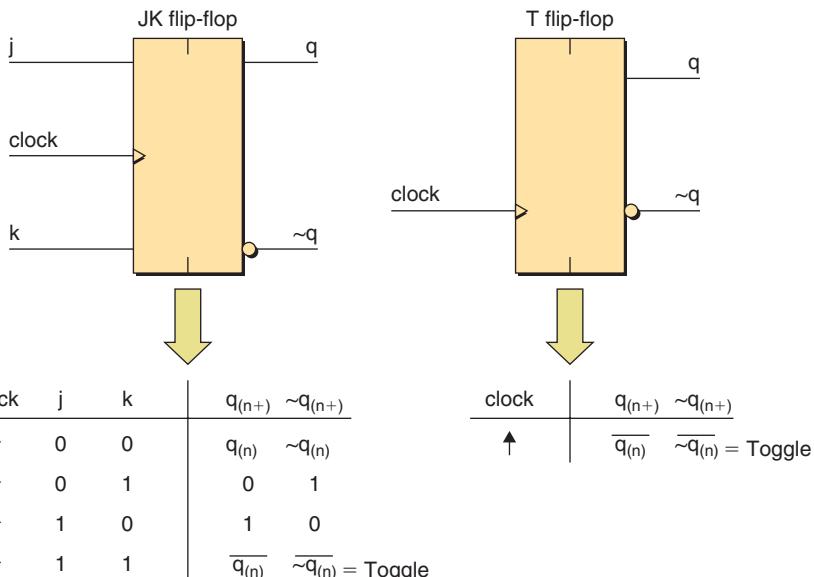


FIGURE 11.19
JK and T flip-flops.

⁸This may be the origin of the term *flip-flop*, because the outputs “flip” and “flop” back and forth.

SHIFT REGISTERS

As was previously noted, another term for a flip-flop is *register*. Functions known as *shift registers*—which facilitate the shifting of binary data one bit at a time—are commonly used in digital systems. Consider a simple 4-bit shift register constructed using D-type flip-flops (Figure 11.20).

This particular example is based on positive-edge triggered D-type flip-flops with active-low \sim clear inputs (in this case we're only using each flip-flop's q output). Also, this example is classed as a *Serial-In-Parallel-Out* (SIPO) shift register, because data is loaded in *serially* (one bit after the other) and read out in *parallel* (side by side).

When the \sim clear input is presented with a logic 0 (its active state), all of the q outputs are forced to 0. When the \sim clear input is set to 1 (its inactive state), a positive-edge on the clock input loads the value on the serial_in input into the first flip-flop, dff[0]. At the same time, the value that used to be in dff[0] is loaded into dff[1], the value that used to be in dff[1] is loaded into dff[2], and the value that used to be in dff[2] is loaded into dff[3].

This may seem a tad weird-and-wonderful the first time you see it, but the way in which this works is actually quite simple (and, of course, capriciously cunning). Each flip-flop exhibits a delay between seeing an active edge on its clock

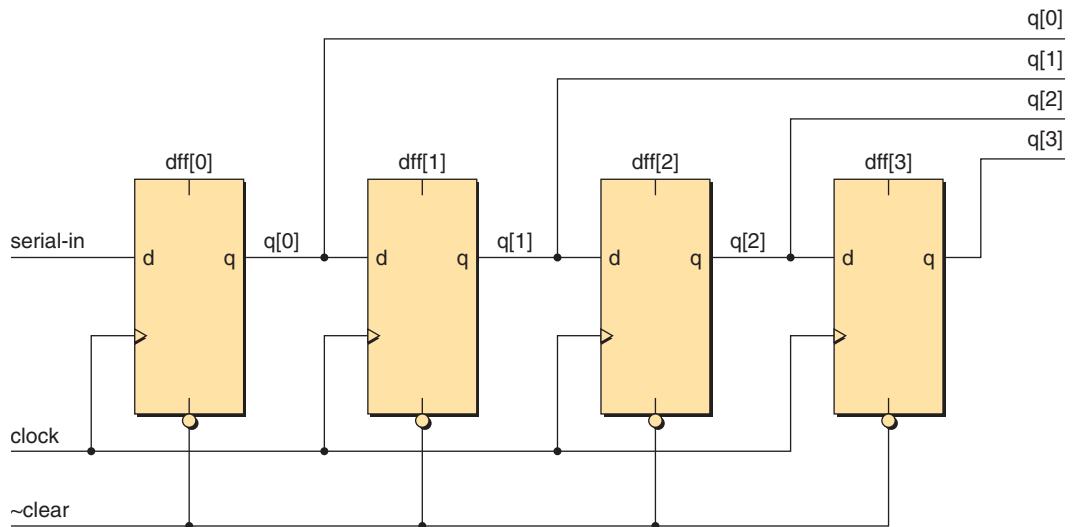
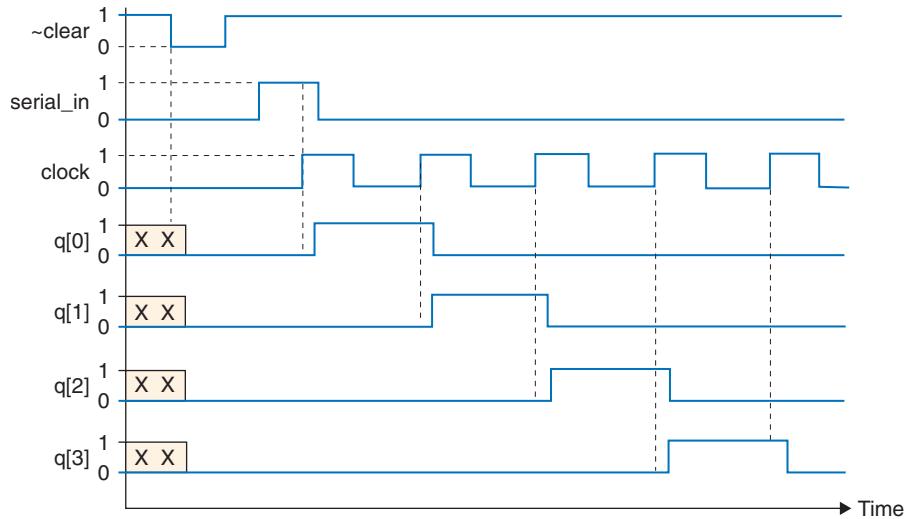


FIGURE 11.20
SIPO shift register.

**FIGURE 11.21**

Waveform for SIPO shift register.

input and the ensuing response on its q output. These delays provide sufficient time for the next flip-flop in the chain to load the value from the previous stage before that value changes. Consider an example waveform where a single logic 1 value is migrated through the shift register (Figure 11.21).

When we start, all of the flip-flops contain X (“don’t know”) values. When the $\sim\text{clear}$ input is placed in its active state (logic 0), all of the flip-flops are cleared to 0. When the first active edge occurs on the clock input, the serial_in input is 1, so this is the value that’s loaded into the first flip-flop. At the same time, the original 0 value from the first flip-flop is loaded into the second, the original 0 value from the second flip-flop is loaded into the third, and the original 0 value from the third flip-flop is loaded into the fourth.

When the next active edge occurs on the clock input, the serial_in input is 0, so this is the value that’s loaded into the first flip-flop. At the same time, the original 1 value from the first flip-flop is loaded into the second, the 0 value from the second flip-flop is loaded into the third, and the 0 value from the third flip-flop is loaded into the fourth.

Similarly, when the next active edge occurs on the clock input, the serial_in input is still 0, so this is the value that’s loaded into the first flip-flop. At the same time, the 0 value from the first flip-flop is loaded into the second, the 1

value from the second flip-flop is loaded into the third, and the 0 value from the third flip-flop is loaded into the fourth. And so it goes ...

Other common shift register variants are the *Parallel-In-Serial-Out* (PISO), and the *Serial-In-Serial-Out* (SISO). For example, consider a 4-bit SISO shift register (Figure 11.22).

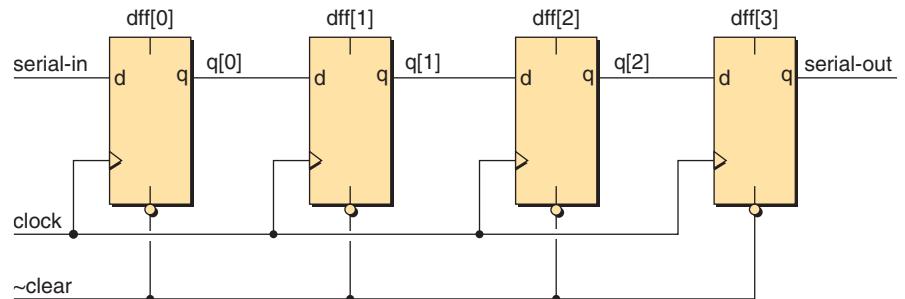


FIGURE 11.22
SISO shift register.

COUNTERS

Counter functions are also commonly used in digital systems. The number of states that the counter will sequence through before returning to its original value is called the *modulus* of the counter. For example, a function that counts from 0000_2 to 1111_2 in binary (or 0 to 15 in decimal) has a modulus of 16 and would be called a *modulo-16*, or *mod-16*, counter. Consider a modulo-16 binary counter implemented using D-type flip-flops (Figure 11.23).

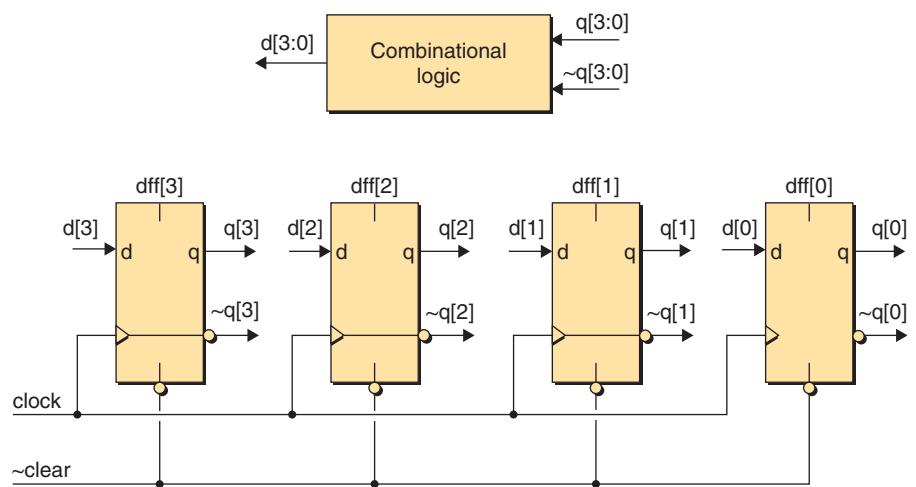
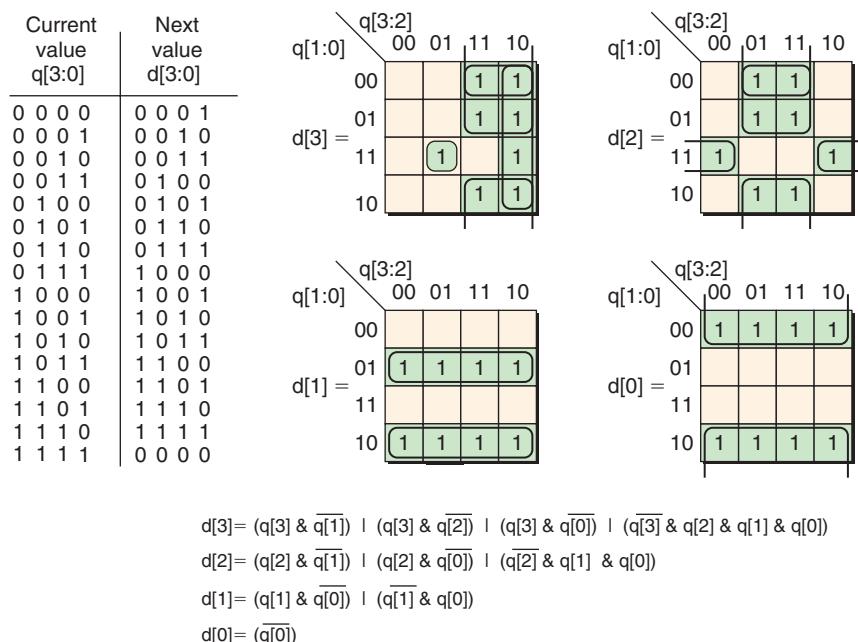


FIGURE 11.23
Modulo-16 binary counter.

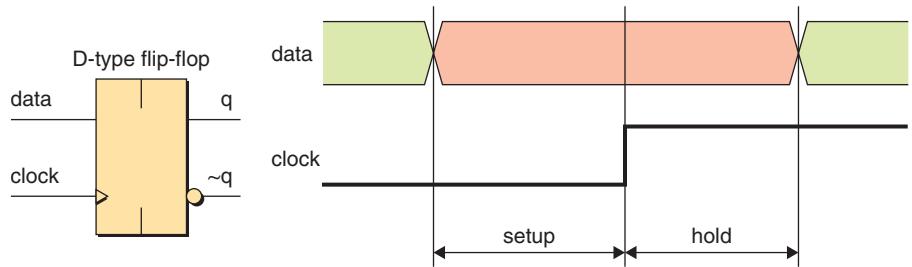
This particular example is based on positive-edge triggered D-type flip-flops with active-low $\sim\text{clear}$ inputs. The four flip-flops are used to store the current count value that is displayed on their $q[3:0]$ outputs. When the $\sim\text{clear}$ input is set to 1 (its inactive state), a positive-edge on the clock input causes the counter to load the next value in the count sequence.

A block of combinational logic is used to generate the next value, $d[3:0]$, which is based on the current value $q[3:0]$ (Figure 11.24). Note that there is no need to create the inverted versions of $q[3:0]$ used in these equations (these are the signals with the horizontal bars drawn over them), because these signals are already available from the flip-flops in the form of their $\sim q[3:0]$ outputs.

**FIGURE 11.24**

Generating the next count value.

It's important to note that we don't need to implement a binary counter, per se (by which we mean a counter that counts in a binary sequence). For example, a 4-bit binary counter would follow the sequence 0000_2 , 0001_2 , 0010_2 , 0011_2 , ... 1110_2 , 1111_2 , at which point it would cycle back to 0000_2 and start all over again. If we wished, our counter could follow a *Gray code* sequence (see Appendix D: *Gray Codes*), or we could use a *Linear Feedback Shift Register* (LFSR) approach [see Appendix E: *Linear Feedback Shift Registers (LFSRs)*].

**FIGURE 11.25**

Setup and hold times.

SETUP AND HOLD TIMES

One point we've glossed over thus far is the fact that there are certain timing requirements associated with flip-flops. In particular, there are two parameters called the *setup* and *hold times* that describe the relationship between the flip-flop's data and clock inputs (Figure 11.25).

The waveform shown here is a little different from those we've seen before. What we're trying to indicate is that when we start (on the left-hand side), the value presented to the data input may be a 0 or a 1, and it can change back and forth as often as it pleases. However, it must settle one way or the other before the *setup* time; otherwise when the active edge occurs on the clock we can't guarantee what will happen. Similarly, the value presented to the data input must remain stable for the *hold* time following the clock or, once again, we can't guarantee what will happen. In our illustration, the period for which the value on the data input must remain stable is shown in pink.

The setup and hold times shown in Figure 11.24 are reasonably understandable. However, things can sometimes become a little confusing, especially in the case of today's Deep Submicron (DSM) integrated circuit technologies.⁹ The problem is that we may sometimes see so-called *negative setup* or *negative hold* times (Figure 11.26).

Once again, the periods for which the value on the data input must remain stable are shown in pink. These effects, which may seem a little strange at first, are caused by internal delay paths inside the flip-flop.

Last but not least, we should note that there will also be setup and hold times between the clear (or reset) and preset (or set) inputs and the clock input. Also, there will be corresponding setup and hold times between the data, clear (or reset), and preset (or set) inputs and the enable input on D-type latches (phew!).

⁹Integrated circuits (and DSM technologies) are introduced in *Chapter 14: Integrated Circuits (ICs)*.

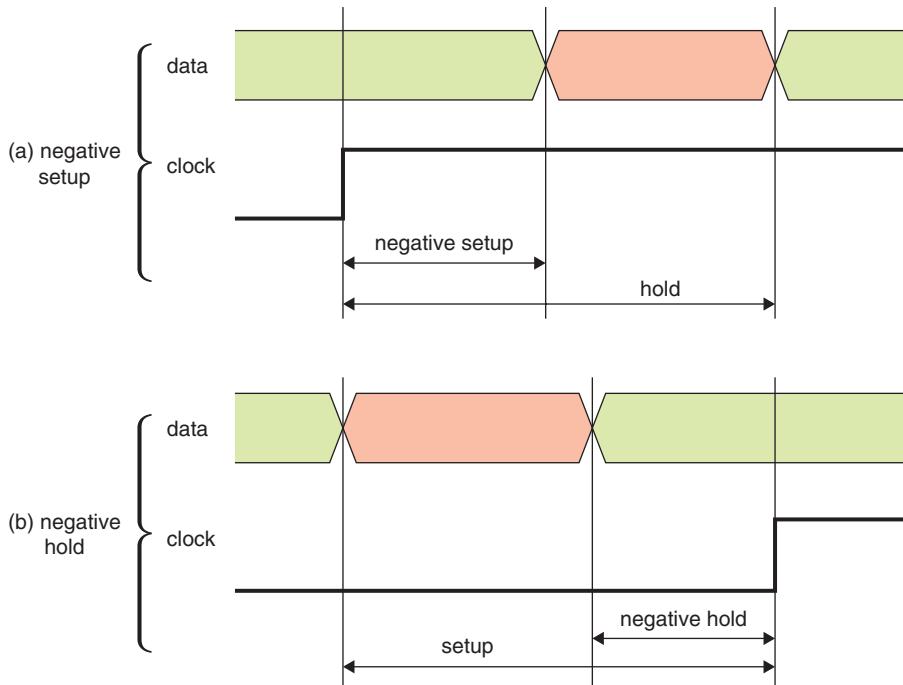


FIGURE 11.26
Negative setup and hold times.

BRICK BY BRICK

Let us pause here for a brief philosophical moment. Consider, if you will, a brick formed from clay. Now, there's not a lot you can do with a single brick, but when you combine thousands and thousands of bricks together you can create the most tremendous structures. At the end of the day, the Great Wall of China is no more than a pile of bricks, molded by man's imagination.¹⁰

In the world of the electronics engineer, silicon is the clay, primitive logic gates are the bricks, and the functions described above are simply building blocks.¹¹ Any digital system, even one as complex as a supercomputer, is constructed from building blocks like comparators, multiplexers, shift registers, and counters. Once you understand the building blocks, there are no ends to the things you can achieve!

¹⁰But at approximately 2400 km in length, it's a very impressive pile of bricks (I've walked—and climbed—a small portion of the beast and it fair took my breath away).

¹¹We might also note that transistors and clay share something in common—they both consist predominantly of silicon!

This page intentionally left blank

CHAPTER 12

State Machines

“IS THAT A GIZMO IN YOUR POCKET, OR . . .”

Consider a coin-operated machine that accepts nickels and dimes¹ and, for the princely sum of fifteen cents, dispenses some useful article called a “gizmo” that the well-dressed man-about-town could not possibly be without.² We may consider such a machine to comprise three main blocks: a *receiver* that accepts money, a *dispenser* that dispenses the “gizmo” along with any change, and a *controller* that oversees everything and makes sure things function as planned (Figure 12.1).

The connections marked nickel, dime, dispense, change, and acknowledge represent digital signals carrying logic 0 and 1 values. The user can deposit nickels and dimes into the receiver in any order, but may only deposit one coin at a time. When a coin is deposited, the receiver determines its type and sets the corresponding signal (nickel or dime) to a logic 1.

The operation of the controller is synchronized by the clock signal. On a rising edge of the clock, the controller examines the nickel and dime inputs to see

151

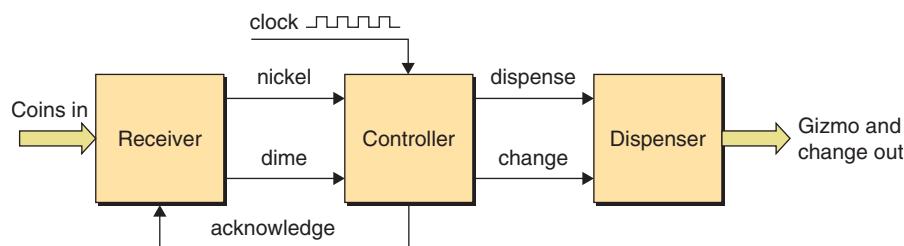


FIGURE 12.1
Block diagram of a coin-operated machine.

¹For the benefit of those readers who do not reside in the United States, nickels and dimes are American coins worth five and ten cents, respectively.

²I always carry two in case of an emergency.

if any coins have been deposited. The controller keeps track of the amount of money deposited and determines if any actions are to be performed.

Every time the controller inspects the nickel and dime signals, it sends an acknowledge signal back to the receiver. The acknowledge signal informs the receiver that the coin has been accounted for, and the receiver responds by resetting the nickel and dime signals to logic 0 and awaiting the next coin. The acknowledge signal can be generated in a variety of ways that are not particularly relevant here.

When the controller decides that sufficient funds have been deposited, it instructs the dispenser to deal out a “gizmo” and any change (if necessary) by setting the dispense and change signals to logic 1, respectively.

STATE DIAGRAMS

A useful level of abstraction for a function such as the controller is to consider it as consisting of a set of *states* through which it sequences. The *current state* depends on the *previous state* combined with the *previous values* on the nickel and dime inputs. Similarly, the *next state* depends on the *current state* combined with the *current values* on the nickel and dime inputs. The operation of the controller may be represented by means of a *state diagram*, which offers a way to view the problem and to describe a solution (Figure 12.2).

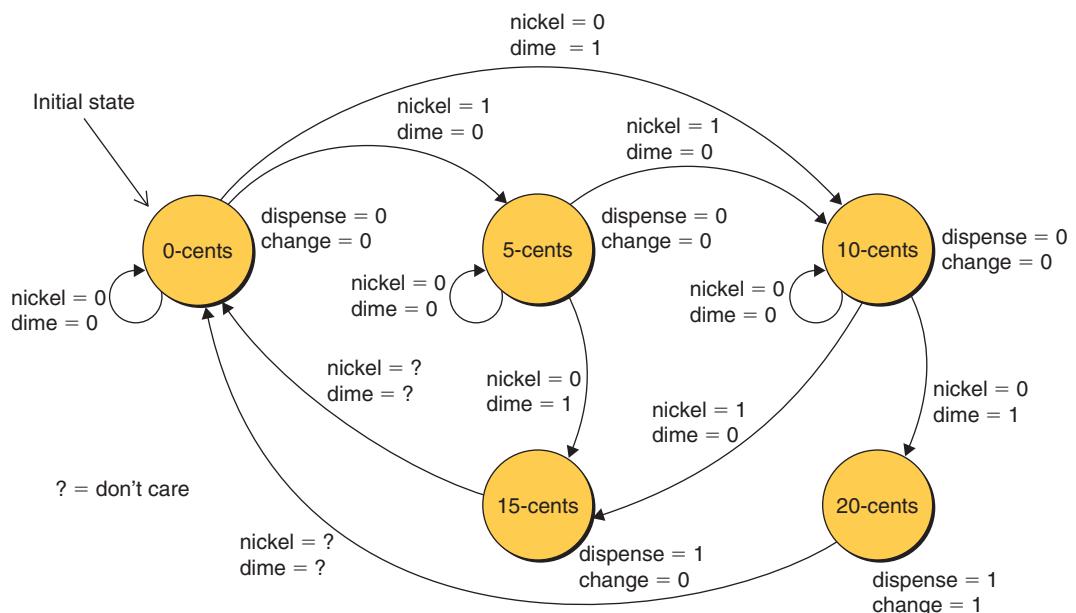


FIGURE 12.2
State diagram for the controller.

The states are represented by the circles labeled 0-cents, 5-cents, 10-cents, 15-cents, and 20-cents, and the values on the dispense and change outputs are associated with these states. The arcs connecting the states are called *state transitions* and the values of the nickel and dime inputs associated with the state transitions are called *guard conditions*. The controller will only sequence between two states if the values on the nickel and dime inputs match the guard conditions.

Let's assume that the controller is in its initial state of 0-cents. The values of the nickel and dime inputs are tested on every rising edge on the clock.³ As long as no coins are deposited, the nickel and dime signals remain at 0 and the controller remains in the 0-cents state. Once a coin is deposited, the next rising edge on the clock will cause the controller to sequence to the 5-cents or the 10-cents states depending on the coin's type. It is at this point that the controller sends an acknowledge signal back to the receiver, instructing it to reset the nickel and dime signals back to logic 0 and to await the next coin.

Observe that the 0-cents, 5-cents, and 10-cents states have state transitions that loop back into themselves (the ones with associated nickel = 0 and dime = 0 guard conditions). These indicate that the controller will stay in whichever state it is currently in until a new coin is deposited.

So, at this stage of our discussions, the controller is either in the 5-cents or the 10-cents state depending on whether the first coin was a nickel or a dime, respectively. What happens when the next coin is deposited? Well, this depends on the state we're in and the type of the new coin. If the controller is in the 5-cents state, then a nickel or dime will move it to the 10-cents or 15-cents states, respectively. Alternatively, if the controller is in the 10-cents state, then a nickel or dime will move it to the 15-cents or 20-cents states, respectively.

When the controller reaches either the 15-cents or 20-cents states, the next clock will cause it to dispense a "gizmo" and return to its initial 0-cents state (in the case of the 20-cents state, the controller will also dispense a nickel in change).

STATE TABLES

Another form of representation is that of a *state table*. This is similar to a truth table (inputs on the left and corresponding outputs on the right), but it also includes the controller's *current state* as an input and its *next state* as an output (Figure 12.3).

³The controller is known to sequence between states only on the rising edge of the clock, so displaying this signal on every transition in the state diagram would be redundant.

<i>Current state</i>	<i>clock</i>	<i>nickel</i>	<i>dime</i>	<i>dispense</i>	<i>change</i>	<i>Next state</i>
0-cents	↑	0	0	0	0	0-cents
0-cents	↑	1	0	0	0	5-cents
0-cents	↑	0	1	0	0	10-cents
5-cents	↑	0	0	0	0	5-cents
5-cents	↑	1	0	0	0	10-cents
5-cents	↑	0	1	0	0	15-cents
10-cents	↑	0	0	0	0	10-cents
10-cents	↑	1	0	0	0	15-cents
10-cents	↑	0	1	0	0	20-cents
15-cents	↑	?	?	1	0	0-cents
20-cents	↑	?	?	1	1	0-cents

FIGURE 12.3

State table for the controller.

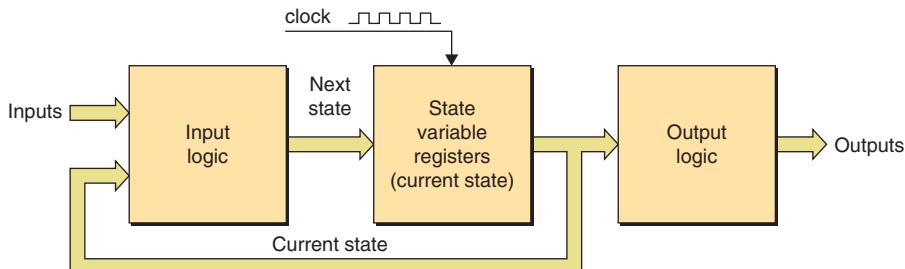
In this instance, the *clock* signal has been included for purposes of clarity (it's only when there's a rising edge on the *clock* that the outputs are set to the values shown in that row of the table). As for the state diagram introduced in the previous topic, however, displaying this signal is somewhat redundant and it is often omitted.

STATE MACHINES

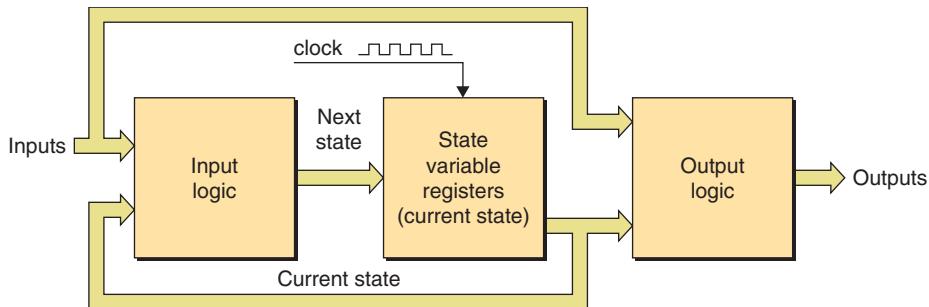
The actual implementation of a function such as the controller is called a *state machine*. In fact, when the number of states is constrained and finite, this is more usually called a *Finite State Machine* (FSM). The heart of a state machine consists of a set of registers⁴ known as the *state variables*. Each state, 0-cents, 5-cents, 10-cents, etc. is assigned a unique binary pattern of 0s and 1s, and the pattern representing the current state is stored in the state variables.

The two most common forms of synchronous, or clocked, state machines are known as Moore and Mealy machines after the men who formalized them. A *Moore machine* is distinguished by the fact that the outputs are derived only from the values in the state variables (Figure 12.4). The controller function featured in this discussion is a classic example of a Moore machine.

⁴For the purposes of these discussions, we'll assume that these registers are D-type flip-flops as were introduced in *Chapter 11: Slightly More Complex Functions*.

**FIGURE 12.4**

Block diagram of a generic Moore machine.

**FIGURE 12.5**

Block diagram of a generic Mealy machine.

By comparison, the outputs from a *Mealy machine* may be derived from a combination of the values in the state variables and one or more of the inputs (Figure 12.5).

In both of the Moore and Mealy forms, the input logic consists of primitive gates such as AND, NAND, OR, and NOR. These combine the values on the inputs with the *current state* (which is fed back from the state variables) to generate the pattern of 0s and 1s representing the *next state*. This new pattern of 0s and 1s is presented to the inputs of the state variables and will be loaded into them on the next rising edge of the clock.

The output logic also consists of standard primitive logic gates that decode the values stored in the state variables (representing the *current state*) and generate the appropriate values on the outputs.

STATE ASSIGNMENT

A key consideration in the design of a state machine is that of *state assignment*, which refers to the process by which the states are assigned to the binary patterns of logic 0s and 1s that are to be stored in the state variables.

One common form of state assignment requiring the minimum number of registers is known as *binary encoding*. Each register can only contain a single binary digit, so it can only be assigned a value of 0 or 1. Two registers can be assigned four binary values (00, 01, 10, and 11), three registers can be assigned

eight binary values (000, 001, 010, 011, 100, 101, 110, and 111), and so forth. The controller used in our coin-operated machine consists of five unique states, and therefore requires a minimum of three state variable registers.

The actual process of binary-encoded state assignment is a nontrivial problem. In the case of our controller function, there are 6720 possible combinations⁵ by which five states can be assigned to the eight binary values provided by three registers. Each of these solutions may require a different arrangement of primitive gates to construct the input and output logic, which in turn affects the maximum frequency that can be used to drive the system clock. Additionally, the type of registers used to implement the state variables also affects the supporting logic (the discussions here are based on the use of D-type flip-flops).

Assuming that full use is made of *don't care* states, an analysis of the various binary-encoded solutions for our controller yields the following ...

138	solutions requiring	7	product terms
852	solutions requiring	8	product terms
1876	solutions requiring	9	product terms
3094	solutions requiring	10	product terms
570	solutions requiring	11	product terms
190	solutions requiring	12	product terms

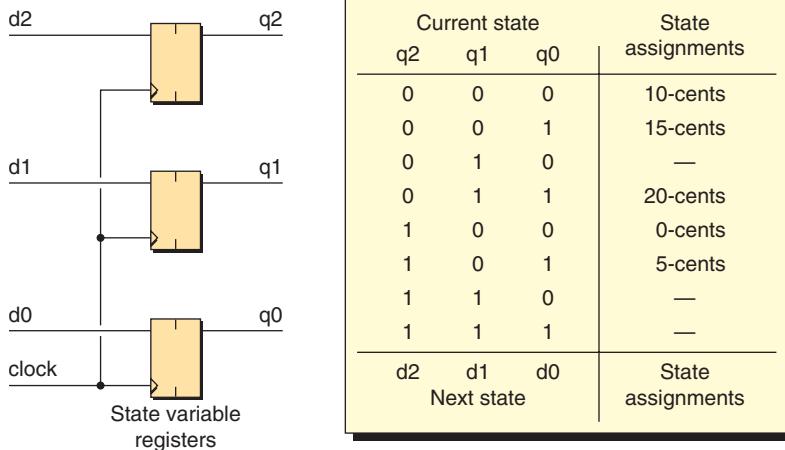
... where a *product term* is a group of *literals* linked by & (AND) operators—for example, (a & b & c)—and a *literal* is any true or inverted variable. Thus, the product term (a & b & c) contains three literals (a, b, and c).

But wait, there's more! A further analysis of the 138 solutions requiring only seven product terms yields the following:

66	solutions requiring	17	literals
24	solutions requiring	18	literals
48	solutions requiring	19	literals

Thus, the chances of a random assignment resulting in an optimal solution are relatively slight. Fortunately, there are computer programs available to aid

⁵This number would be somewhat reduced if all the “mirror-image” combinations were taken into account, but that would not significantly lessen the complexity of determining the optimal combination.

**FIGURE 12.6**

Example binary-encoded state assignment.

designers in this task.⁶ One solution resulting in the minimum number of product terms and literals is shown in Figure 12.6.

A truth table for the controller function can now be derived from the state table shown in Figure 12.3 by replacing the assignments in the *current state* column with the corresponding binary patterns for the state variable outputs (q_2 , q_1 , and q_0), and replacing the assignments in the *next state* column with the corresponding binary patterns for the state variable inputs (d_2 , d_1 , and d_0). The resulting equations can then be derived from the truth table using standard algebraic or Karnaugh map techniques. As an alternative, a computer program can be used to obtain the same results in less time with far fewer opportunities for error.⁷ Whichever technique is employed, the state assignments above lead to the following minimized Boolean equations:

$$d_0 = (\overline{q_0} \& \overline{q_2} \& \text{dime}) | (q_0 \& q_2 \& \overline{\text{nickel}}) | (\overline{q_0} \& \text{nickel})$$

$$d_1 = (\overline{q_0} \& \overline{q_2} \& \text{dime})$$

$$d_2 = (q_0 \& \overline{q_2}) | (q_2 \& \overline{\text{nickel}} \& \overline{\text{dime}}) | (\overline{q_0} \& q_2 \& \overline{\text{dime}})$$

$$\text{dispense} = (q_0 \& \overline{q_2})$$

$$\text{change} = (q_1)$$

⁶I used the program BOOL, which was created by my friend Alon Kfir (a man with a size-16 brain if ever there was one).

⁷Once again, I used BOOL ("What's the point of barking if you have a dog?" as they say in England).

The product terms shown in bold appear in multiple equations. However, regardless of the number of times a product term appears, it is only counted once because it only has to be physically implemented once. Similarly, the literals used to form product terms that appear in multiple equations are only counted once.

Another common form of state assignment is known as *one-hot encoding*, in which each state is represented by an individual register. In this case, our controller with its five states would require five register bits. The one-hot technique typically requires a greater number of logic gates than does binary encoding. However, as the logic gates are used to implement simpler equations, the one-hot method results in faster state machines that can operate at higher clock frequencies.

Last but not least, some designs require the use of state assignments based on *Gray codes*, such that transitioning between any pair of states involves changing the value of only a single state variable (see also Appendix D: *Gray Codes* for more discussion).

DON'T CARE STATES, UNUSED STATES, AND LATCH-UP CONDITIONS

It was previously noted that the analysis of the binary-encoded state assignment made full use of *don't care* states.⁸ This allows us to generate a solution that uses the least number of logic gates, but there are additional considerations that must now be discussed in more detail.

The original definition of our coin-operated machine stated that it is only possible for a single coin to be deposited at a time. Assuming this to be true, then the nickel and dime signals will never be assigned logic 1 values simultaneously. Thus, the designer (or a computer program) can use this information to assign *don't care* states to the outputs for any combination of inputs that includes a logic 1 on both of the nickel and dime signals.

Additionally, the three binary-encoded state-variable registers provide eight possible binary patterns, of which only five were used. The analysis above was based on the assumption that *don't care* states can be assigned to the outputs for any combination of inputs that includes one of the unused patterns on the state variables. This assumption also requires further justification.

⁸The concept of don't care states was introduced in Chapter 10: *Karnaugh Maps*.

When the coin-operated machine is first powered-up, each state variable register can potentially initialize with a random logic 0 or 1 value. The controller could therefore power-up with its state variables containing any of the eight possible patterns of 0s and 1s. For some state machines this would not be an important consideration, but this is not true in the case of our coin-operated machine. Suppose the controller were to power-up in the 20-cents state, for example, in which case it would immediately dispense a “gizmo” and five cents change. The owner of such a machine may well be of the opinion that this was a less-than-ideal feature.

Alternatively, the controller could power-up with its state variables in one of the unused combinations. Subsequently, the controller could sequence directly—or via one or more of the other unused combinations—to any of the defined states. In a worst-case scenario, the controller could remain in the current unused combination indefinitely or it could sequence endlessly between unused combinations; these worst-case scenarios are known as *latch-up conditions*.

One method of avoiding latch-up conditions is to assign additional, dummy states to each of the unused combinations and to define state transitions from each of these dummy states to the controller’s initialization state of 0-cents. Unfortunately, in the case of our coin-operated machine, this technique would not affect the fact that the controller could “wake up” in a valid state other than 0-cents. An alternative is to provide some additional circuitry to generate a *Power-on Reset* (PoR) signal—for example, a single pulse that occurs only when the power is first applied to the machine. The power-on reset can be used to force the state variable registers into the pattern associated with the 0-cents state. The state assignment analysis presented in the previous topic assumed the use of such a power-on reset.

This page intentionally left blank

CHAPTER 13

Analog-to-Digital and Vice Versa

SETTING THE SCENE

As we began our discussions in *Chapter 1: Analog Versus Digital* by separating the analog and digital views of the world, it seems appropriate to close this section of the book by reuniting them.

In the early days of electronics, systems were predominantly analog in nature.¹ Today, analog functions are still used for a wide variety of tasks, but digital implementations are preferable in many cases.

Why might a digital system be preferable to its analog counterpart? Well, as a simple example, let's cast our minds back into the mists of time and consider VHS video tapes, which were analog in nature.² By this we mean that video data was stored on the tape as an analog (constantly varying) signal. The problem is that analog representations of this type are susceptible to noise effects.

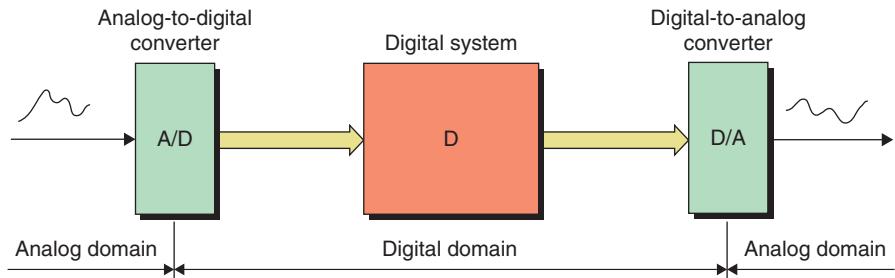
Let's say that you took a VHS video of Granny's birthday and that you made a copy of this video for your Auntie Barbara. Unfortunately, the process of copying analog signals is not perfect. Small errors (*noise*) will creep in, resulting in unwanted visual artifacts. Now, the first copy might not be too bad, but suppose Auntie Barbara makes a copy of *her* copy to send to Uncle Frank, and that he subsequently makes a copy of *his* copy to send to Cousin Bob.

The result is that we now have noise on top of noise on top of noise, and it doesn't take long before downstream copies start to look absolutely horrible.

161

¹I remember playing with analog computers at college, for example.

²VHS systems were launched in 1976. DVDs came along in the latter half of the 1990s. By the early 2000s, DVDs had become the more popular media. At the time of this writing, a lot of folks still have VHS tapes and players lying around the home (especially those who have a lot of home movies), but you can't get mainstream movies on VHS tapes anymore.

**FIGURE 13.1**

Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters.

The alternative that we use today (in the form of CDs and DVDs, for example) is to take real-world analog signals (music, video, etc.) and convert them into digital representations. Once we have our video of Granny's birthday in the digital domain—stored as a series of numbers with associated error correcting codes and suchlike—we can make copies of copies of copies ad infinitum without suffering any degradation.

The point of all of this is that, while some systems operate solely on digital data, others have to interact with the analog world. It may be necessary to convert an analog input into a form that can be manipulated by the digital system, or to transform an output from a digital system into the analog realm. As is illustrated in Figure 13.1, these tasks are performed by *Analog-to-Digital* (A/D) and *Digital-to-Analog* (D/A) converters, respectively (these may also be referred to as ADCs and DACs).

In this example, we commence on the left-hand side with a signal in the analog domain. This signal is passed through an A/D converter, which translates it into a digital equivalent. We can now process this data to our heart's content using digital techniques; this is known as *Digital Signal Processing* (DSP). Finally, we take the output from the digital system, pass it through a D/A converter, and return the result to the real world in which we live. Let's consider all of this in a little more detail ...

ANALOG-TO-DIGITAL

A *transducer* is a device that converts input energy of one form into output energy of another. Analog effects can manifest themselves in a variety of different ways such as heat and pressure. In order to be processed by a digital system, the analog quantity must be detected and converted into a suitable form by means of an appropriate transducer called a *sensor*. For example, a microphone is a sensor that detects sound and converts it into a corresponding

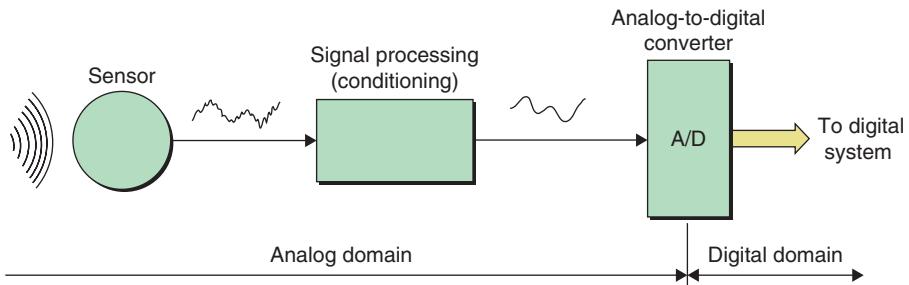


FIGURE 13.2
The analog-to-digital conversion process.

voltage. The analog-to-digital conversion process can be represented as shown in Figure 13.2.

The output from the sensor typically undergoes some form of signal processing, such as filtering and amplification, before being passed to the A/D converter. This signal processing is generically referred to as *conditioning*. The A/D converter accepts the conditioned analog voltage and converts it into a series of equivalent digital values by *sampling* and *quantization* (Figure 13.3).

The sampling usually occurs at regular time intervals and is triggered by the digital part of the system. The complete range of values that the analog signal can assume is divided into a set of discrete bands or quanta. At each sample time, the A/D converter determines which band the analog signal falls into

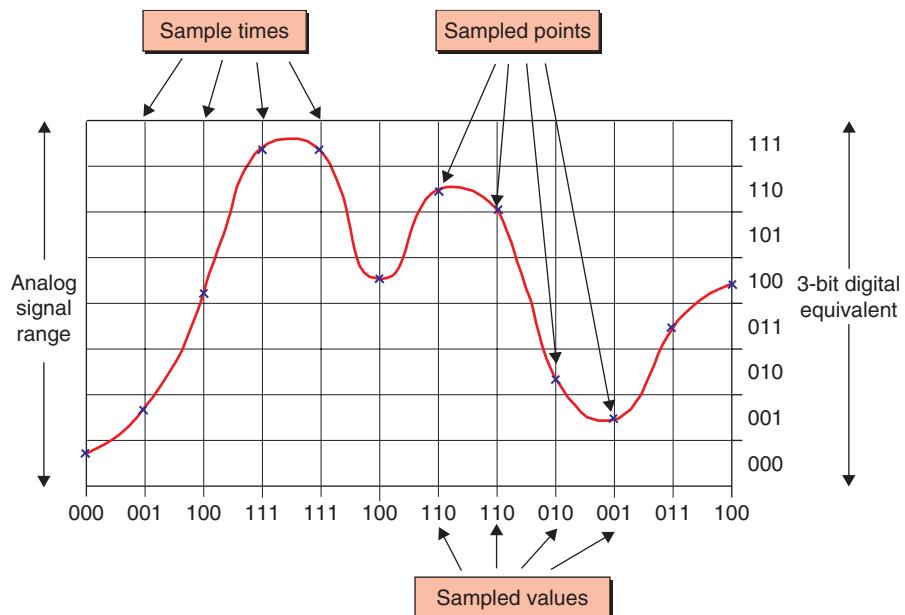


FIGURE 13.3
The sampling and quantization of an analog signal.

(this is the “quantization” part of the process) and outputs the equivalent binary code for that band.

The main factor governing the accuracy of the conversion is the number of bands used. For example, a 3-bit code can represent only eight bands, each encompassing 12.5% of the analog signal’s range, while a 12-bit code can represent 4096 bands, each encompassing 0.025% of the signal’s range. This means that we have a classical engineering trade-off. In the case of a music CD, for example, we want the best sound we can get, but the more bits we use to represent each sample, the more data³ we have to store and the more processing we have to perform. This also leads us to the concept of *quantization noise* or *quantization error*, which refers to the difference between the original real-world analog signal and the quantized digital value caused by *rounding*⁴ (or *truncating*) the analog signal to the nearest digital quanta.

DIGITAL-TO-ANALOG

A D/A converter accepts a digital code and transforms it into a useful corresponding analog current or voltage by means of an appropriate transducer called an *actuator*. For example, a loudspeaker is an actuator that converts an electrical signal into sound. The digital-to-analog conversion process can be represented as shown in Figure 13.4.

The conversions usually occur at regular time intervals and are triggered by a clock signal from the digital part of the system. The output from the D/A converter typically undergoes some form of conditioning before being passed to the actuator. For example, in the case of an audio system, the “staircase-like”

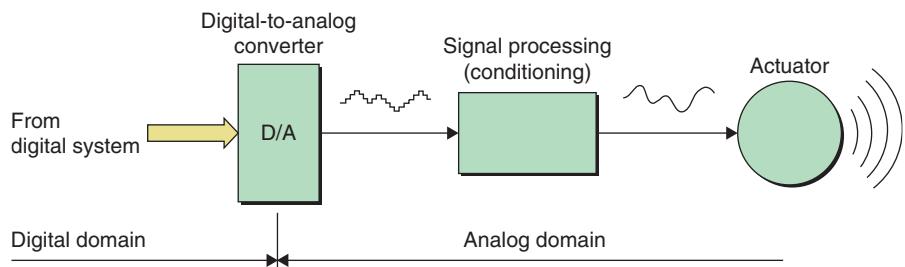


FIGURE 13.4

The digital-to-analog conversion process.

³As we discussed in Chapter 7, the term *data* is the plural of the Latin *datum*, meaning “something given.” The plural usage is still common, especially among scientists, so it’s not unusual to see expressions like “These data are ...” However, it is becoming increasingly common to use *data* to refer to a singular group entity such as information; thus, an expression like “This data is ...” would also be acceptable to a modern audience.

⁴See Appendix H: *Rounding Algorithms 101*, for discussions on different rounding algorithms.

signal coming out of the D/A converter will be “smoothed” before being passed to an amplifier (not shown in Figure 13.4) and, ultimately, to the loudspeaker.

DSP VERSUS DSP

It's hard to read an electronics-related article these days without running across the term *DSP*, but what does this actually mean? Depending on your background, your knee-jerk reaction may be “*DSP means Digital Signal Processing*,” or “*DSP refers to a Digital Signal Processor*.” In fact, if you were feeling particularly bold, you may even have said: “*DSP refers to a Digital Signal Processor performing Digital Signal Processing*.” (This latter option would be very clever of you but, as we shall see, this is not necessarily the case.)

Actually, considering the fact that the term *DSP* is so ubiquitous these days, there's a surprising amount of confusion about all of this. For example, do you actually need a digital signal processor to perform digital signal processing? It may surprise you to hear that the short answer is “*No!*” (Of course, the longer answer is: “*Well, it all depends on what you mean by a ‘Digital Signal Processor,’ doesn't it?*”)

Let's take a step back. First we'll set the scene by considering the concept of *Analog Signal Processing* (ASP); next we'll ponder what we mean by *Digital Signal Processing* (DSP); and finally we'll consider how we might go about actually doing it.

ANALOG SIGNAL PROCESSING (ASP)

As we discussed in *Chapter 1: Analog Versus Digital*, in the context of electronics, an analog device or system is one that uses continuously variable signals to represent information for input, processing, output, and so forth. On this basis, *Analog Signal Processing* (ASP) involves the processing of signals in the analog domain.

A really simple form of *Analog Signal Processor* (ASP) would be a basic analog amplifier. Suppose we took an analog signal from a guitar, for example, fed it into the input to the amplifier, and used the amplified output to drive a loudspeaker, as illustrated in Figure 13.5.

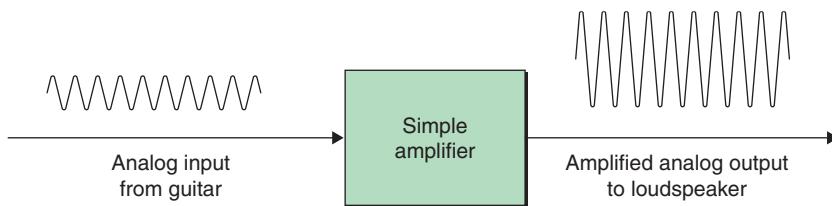


FIGURE 13.5

Simple analog signal processing scenario.

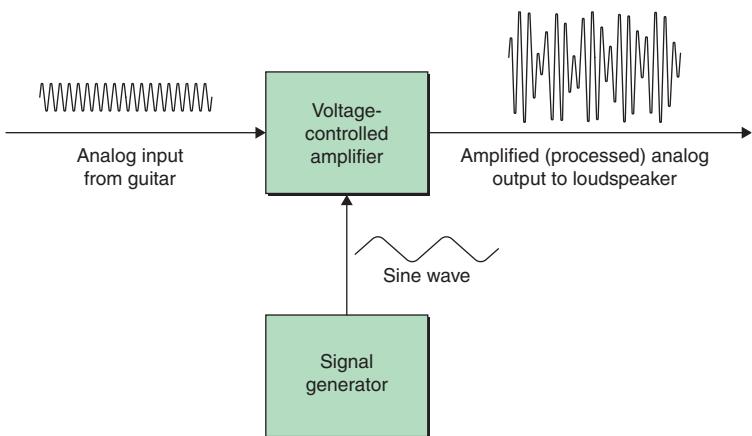


FIGURE 13.6

A slightly more complex analog signal processing implementation.

In this case, our analog signal processing simply involves generating a larger version of our input signal. A slightly more complex form of analog signal processing might be to use a signal generator to output a sine wave, and to use the amplitude of this sine wave to drive a voltage-controlled amplifier, as illustrated in Figure 13.6.

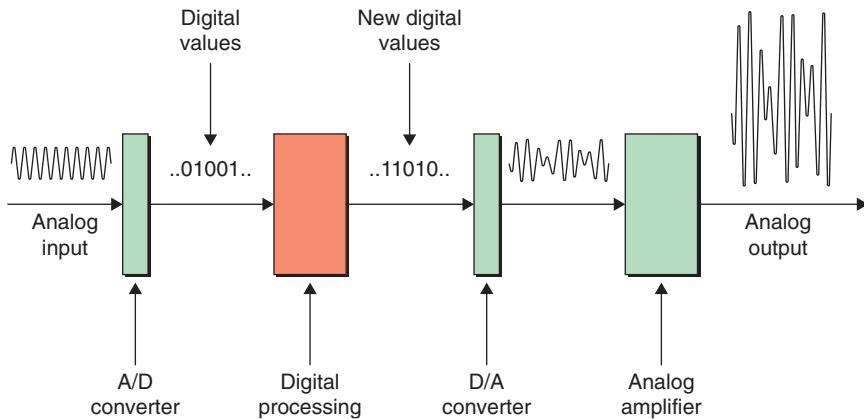
Analog signal processing can be very effective for a variety of tasks, but it tends to be limited to things like amplification, filtering, signal conditioning, and similar activities. One classic example would be guitar sound-effect generators such as *reverb* and *echo*. Another classic example would be *Amplitude Modulation* (AM) radio.

Analog techniques can achieve large results using relatively few components, but only for relatively simple (conceptually speaking) tasks. If we want to perform more complex signal processing activities, it's generally easier to do so in the digital domain.

DIGITAL SIGNAL PROCESSING (DSP)

Perhaps not surprisingly, the term *digital signal processing* refers to processing data (signals) in the digital domain. Let's stick with our guitar-based examples for a while. As we discussed earlier in this chapter, we can take an analog signal such as the output from an electric guitar and pass it through an *Analog-to-Digital* (A/D) converter, which translates it into a series of digital values (Figure 13.7).

We can now process these digital values in the digital domain. For example, we might apply a variety of signal processing algorithms to simulate reverb, delay, or distortion effects to the signal. Following this processing, we can pass our modified digital values through a *Digital-to-Analog* (D/A) converter, amplify the resulting analog signal, and use it to drive a loudspeaker.

**FIGURE 13.7**

Simple digital signal processing scenario.

DSP EXAMPLES

These days we are surrounded by devices that are performing digital signal processing activities. For example, the *Graphics Processing Unit* (GPU) driving my computer display is performing DSP as I pen these words.

Do you have an MP3 player? If so, have you ever taken one of your music CDs and “ripped” it onto your player? In this case, your computer took the uncompressed digital audio signal from the CD and performed digital signal processing to compress it into an MP3 file (where MP3 is a digital audio encoding and compression format). Similarly, when you decide to play a track on your MP3 player, the player performs digital signal processing to uncompress the MP3 data into a form suitable for listening.

As another obvious example, do you own a digital camera? If so, whenever you take a picture, the camera performs a variety of digital signal processing tasks, such as automatically locating and focusing on any human faces in the frame. Later, in addition to compressing the image to make it consume less memory (so you can store more pictures), the camera may use digital signal processing algorithms to adjust the image for brightness, contrast, color balance, and so forth.

And the list goes on, and on, and on ...

WHAT IMPLEMENTS THE DIGITAL SIGNAL PROCESSING?

When you come to think about it, the complicated part is deciding on the algorithms we wish to apply to our digital data. Once we’ve decided on these algorithms, all we have to do is to implement them in some way.

Whatever algorithms we decide to use, they ultimately break down into large numbers of relatively simple tasks, such as multiplying digital values, adding values together, and similar operations. Purely for the sake of an example, let's assume that one small part of a digital-signal processing-algorithm looks like the following:

$$y = (a \times b) + (c \times d) + (e \times f) + (g \times h);$$

Note that all of these variables (y , a , b , c , ...) represent multi-bit values; for example, a through h might each be 16-bits wide while y might be 32-bits wide.

Now, we could use a general-purpose microprocessor to perform this task. But this would be relatively slow and painful. First, the processor would have to load a into one of its internal registers; then it would have to load b into another register; then it would have to multiply them together and store the result in yet another register. Next, the processor would have to repeat these operations for variables c and d , but this time it would also have to add this product to the original product from the multiplication of a and b ; and so it goes ...

An alternative is to use a special-purpose microprocessor called a *Digital Signal Processor* (DSP). This may be presented as a discrete silicon chip (integrated circuit) or as a *core*⁵ that is built into a larger chip. If you were to look inside a cell phone, for example, you might find a small general-purpose processor that is used to monitor the keyboard and suchlike coupled with a DSP that is used to process your voice, play music, and display images on the screen.

The architecture of a digital signal processor is targeted toward performing a certain set of digital signal processing tasks. It may, for example, contain one or more *Multiply-and-Accumulate* (MAC) units that can be used to multiply two values together and add the result to a "running total."

Having said this, DSPs are still *von Neumann machines*⁶ that have to fetch data and process it in small "chunks." Thus, yet another approach is to use a *Field Programmable Gate Array* (FPGA). As discussed in *Chapter 16: Programmable ICs*, these devices may be visualized as containing hundreds of thousands of small "islands" of programmable logic in a "sea" of programmable interconnect.

⁵The concept of cores is discussed in more detail in *Chapter 17: Application-Specific Integrated Circuits (ASICs)*.

⁶The term *von Neumann machine* refers to a computer architecture in which data and program memory are mapped into the same address space. Named after Hungarian-born American mathematician John von Neumann (1903–1957), this is the de facto standard architecture for the majority of today's computers.

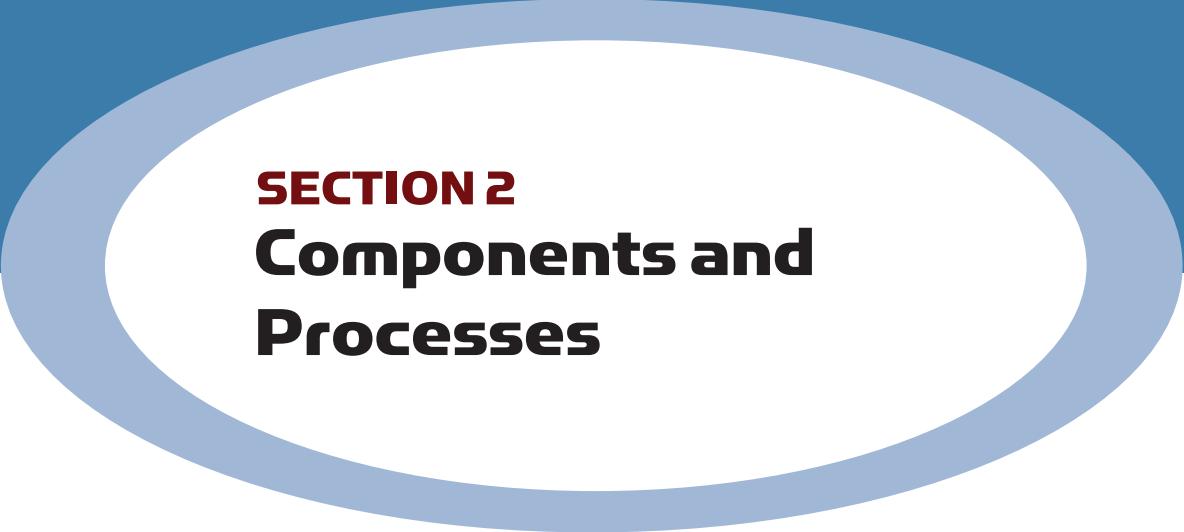
The point is that an FPGA can be configured/programmed on-the-fly to implement whatever task we wish it to perform. In the case of our example equation ...

$$y = (a \times b) + (c \times d) + (e \times f) + (g \times h);$$

... we could configure the FPGA to contain four 16×16 multipliers coupled with a four-input 32-bit adder. Since a high-end FPGA package may contain hundreds or thousands of pins, we could bring all of the a through h inputs in simultaneously; we could perform all four multiplications in a single clock cycle, and we could add all four products on a second clock cycle.

Thus, by performing multiple operations in parallel, an FPGA can perform digital-signal processing tasks much faster than a special-purpose *Digital Signal Processor* (DSP). Having said this, a DSP chip may be the preferred solution depending on the target application.

This page intentionally left blank



SECTION 2

Components and Processes

This page intentionally left blank

CHAPTER 14

Integrated Circuits (ICs)

THE FIRST INTEGRATED CIRCUITS

In the early days of semiconductors, transistors and other electronic components were available only in individual packages. These *discrete components* were laid out on a circuit board and connected by hand using separate wires. At that time, an electronic memory element capable of storing a single binary bit of data cost more than \$2. By comparison, in the early 1990s, enough logic gates to store 5000 bits of data cost less than a penny.¹ This vast reduction in price was primarily due to the invention of the *Integrated Circuit* (IC).²

A functional electronic circuit requires transistors, resistors, diodes, etc., and the connections between them. A *monolithic integrated circuit* (the *monolithic* qualifier is usually omitted) has all of these components formed on the surface layer of a sliver, or chip, of a single piece of semiconductor; hence, the term *monolithic*, meaning “seamless.” Although a variety of semiconductor materials are available, the most commonly used is silicon, and integrated circuits are popularly known as *silicon chips*. (Unless otherwise noted, the remainder of these discussions will assume integrated circuits based on silicon as the semiconductor.)

To a large extent, the demand for miniaturization was driven by the demands of the American space program. For some time, people had been thinking that it would be a good idea to be able to fabricate entire circuits on a single piece of semiconductor. The first public discussion of this idea is credited to a British radar expert, Geoffrey William Arnold (G.W.A.) Dummer (1909–2002), in a paper presented in 1952. However, it was not until the summer of 1958 that Jack St. Clair Kilby (1923–2005), working for Texas Instruments, succeeded in

173

¹I just did a search on the web as I pen these words. I found a 1-gigabyte memory stick for \$9.99, which equates to more than 8.5 million bits of memory per penny.

²In conversation, IC is pronounced by spelling it out as “I-C”.

fabricating multiple components on a single piece of semiconductor. Kilby's first prototype was a phase shift oscillator comprising five components on a piece of germanium, half an inch long and thinner than a toothpick. Although manufacturing techniques subsequently took different paths from those used by Kilby, he is still credited with the creation of the first true integrated circuit.

Around the same time that Kilby was working on his prototype, two of the founders of Fairchild Semiconductors—the Swiss physicist Jean Hoerni (1924–1997) and the American physicist Robert Noyce (1927–1990)—were working on more efficient processes for creating these devices. Between them, Hoerni and Noyce invented the *planar process*, in which optical lithographic techniques are used to create transistors, insulating layers, and interconnections on integrated circuits.

By 1961, Fairchild and Texas Instruments had announced the availability of the first commercial planar integrated circuits comprising simple logic functions. This announcement marked the beginning of the mass production of integrated circuits. In 1963, Fairchild produced a device called the 907 containing two logic gates, each of which consisted of four bipolar transistors and four resistors. The 907 also made use of *isolation layers* and *buried layers*, both of which were to become common features in modern integrated circuits.

During the mid-1960s, Texas Instruments introduced a large selection of basic “building block” ICs called the 54xx (“fifty-four-hundred”) series and the 74xx (“seventy-four-hundred”) series, which were specified for military and commercial use, respectively. These *jelly bean* devices each contained small amounts of simple logic. For example, a 7400 device contained four 2-input NAND gates, a 7402 contained four 2-input NOR gates, and a 7404 contained six NOT (inverter) gates.

TI’s 54xx and 74xx series were implemented in *Transistor-Transistor Logic* (TTL). By comparison, in 1968, RCA introduced a somewhat equivalent CMOS-based library of parts called the 4000 (“four thousand”) series.

In 1967, Fairchild introduced a device called the *Micromosaic*, which contained a few hundred transistors. The key feature of the Micromosaic was that the transistors were not initially connected to each other. A designer used a computer program to specify the function the device was required to perform, and the program determined the necessary transistor interconnections and constructed the photo-masks required to complete the device. The Micromosaic is credited as the forerunner of the modern *Application-Specific Integrated Circuit* (ASIC),³ and also as the first real application of computer-aided design. In 1970,

³ASICS are discussed in more detail in Chapter 17: *Application-Specific Integrated Circuits (ASICs)*.

Fairchild introduced the first 256-bit static RAM, called the 4100, while Intel announced the first 1024-bit dynamic RAM, called the 1103, in the same year.⁴

One year later, in 1971, Intel introduced the world's first *microprocessor* (μ P), the 4004, which was conceived and created by Marcian "Ted" Hoff, Stan Mazor, and Federico Faggin. Also referred to as a *computer-on-a-chip*, the 4004 contained only around 2300 transistors and could execute 60,000 operations per second.

AN OVERVIEW OF THE FABRICATION PROCESS

The construction of integrated circuits requires one of most exacting production processes ever developed. The environment must be at least a thousand times cleaner than that of an operating theater, and impurities in materials have to be so low as to be measured in parts per billion.⁵ The process begins with the growing of a single crystal of pure silicon in the form of a cylinder with a diameter that can be anywhere up to 300 mm.⁶ The cylinder is cut into paper-thin slices called *wafers*, which are approximately 0.2 mm thick (Figure 14.1).

The thickness of the wafer is determined by the requirement for sufficient mechanical strength to allow it to be handled without damage. The actual thickness necessary for the creation of the electronic components is less than 1 μ m (one-millionth of a meter). After the wafers have been sliced from the cylinder, they are polished to a smoothness rivaling the finest mirrors.

The most commonly used fabrication process is *optical lithography*, in which *Ultraviolet Light* (UV) is passed through a stencil-like⁷ object called a *photo-mask*, or just *mask* for short. This square or rectangular mask carries patterns formed by areas that are either transparent or opaque to ultraviolet frequencies (similar in concept to a black-and-white photographic negative) and the resulting image is projected

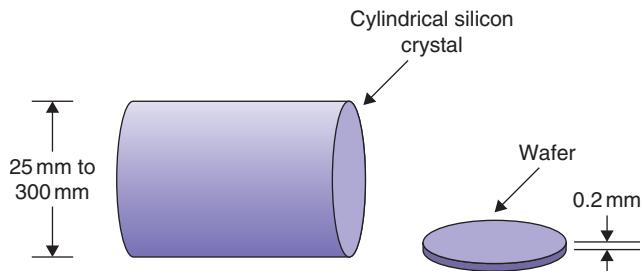


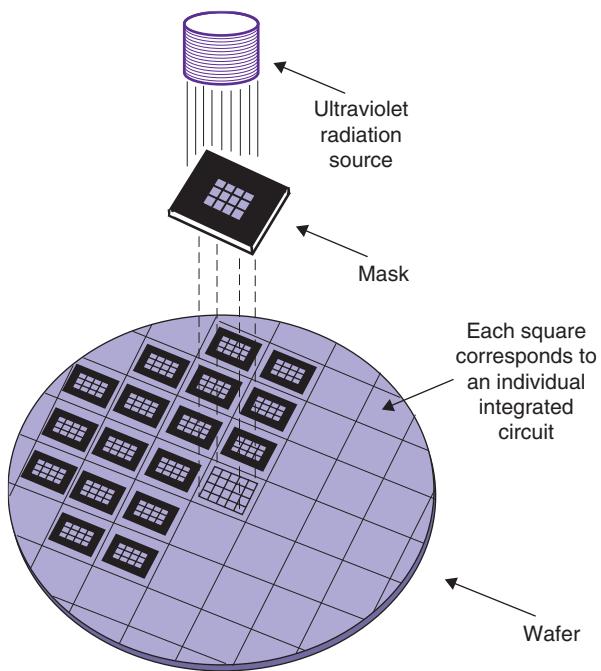
FIGURE 14.1
Creating silicon wafers.

⁴Memory devices are discussed in Chapter 15: *Memory ICs*.

⁵If you took a bag of flour and added a grain of salt, this would be impure by comparison.

⁶When the first edition of this tome hit the streets in 1995, the maximum wafer diameter was 200 mm. By 2002, leading manufacturers were working with 300 mm diameter wafers, which are relatively standard at the time of this writing. By 2012, it is expected that a shift will have started to use 450 mm diameter wafers.

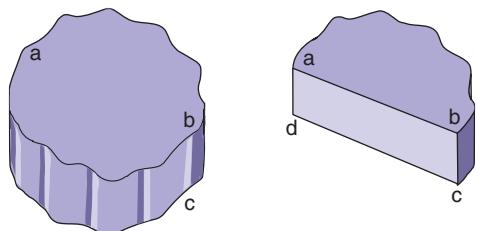
⁷Just in case you were wondering, the term *stencil* comes from the Middle English word *stanseld*, meaning "adorned brightly."

**FIGURE 14.2**

The opto-lithographic step-and-repeat process.

tern has been replicated across the whole of the wafer's surface. This technique for duplicating the pattern is called a *step-and-repeat process*.

As we shall see, multiple layers are required to construct the transistors (and other components), where each layer requires its own unique mask. Once all of the transistors have been created, similar techniques are used to lay down the tracking (wiring) layers that connect the transistors together.

**FIGURE 14.3**

Small area of silicon somewhere on the wafer.

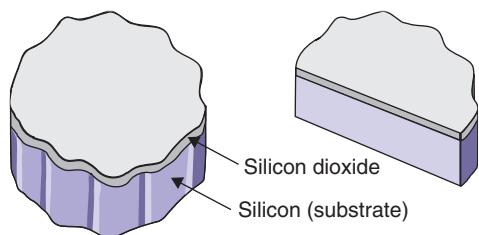
Assume that the small area of silicon shown here is sufficient to accommodate a single transistor in the middle of one of the integrated circuits residing

onto the surface of the wafer. By means of some technical wizardry that we'll consider in the next topic, we can use the patterns of ultraviolet light to "grow" corresponding structures in the silicon. The simple patterns shown in the following diagrams were selected for reasons of clarity; in practice, a mask can contain hundreds of millions (sometimes billions) of fine lines and geometric shapes (Figure 14.2).

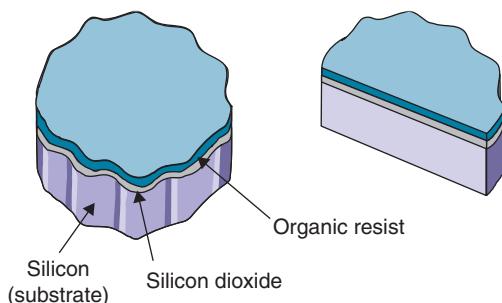
Each wafer can contain hundreds or thousands of identical integrated circuits. The pattern projected onto the wafer's surface corresponds to a single integrated circuit, which is typically in the region of $1\text{ mm} \times 1\text{ mm}$ to $10\text{ mm} \times 10\text{ mm}$, but some chips are $15\text{ mm} \times 15\text{ mm}$, and some are even larger. After the area corresponding to one integrated circuit has been exposed, the wafer is moved and the process is repeated until the same pat-

A SLIGHTLY MORE DETAILED LOOK AT THE FABRICATION PROCESS

To illustrate the manufacturing process in more detail, we will consider the construction of a single NMOS transistor occupying an area far smaller than a speck of dust. For reasons of electronic stability, the majority of processes begin by lightly doping the entire wafer to form either N-type or, more commonly, P-type silicon. However, for the purposes of this discussion, we will assume a process based on a pure silicon wafer (Figure 14.3).

**FIGURE 14.4**

Grow or deposit a layer of silicon dioxide.

**FIGURE 14.5**

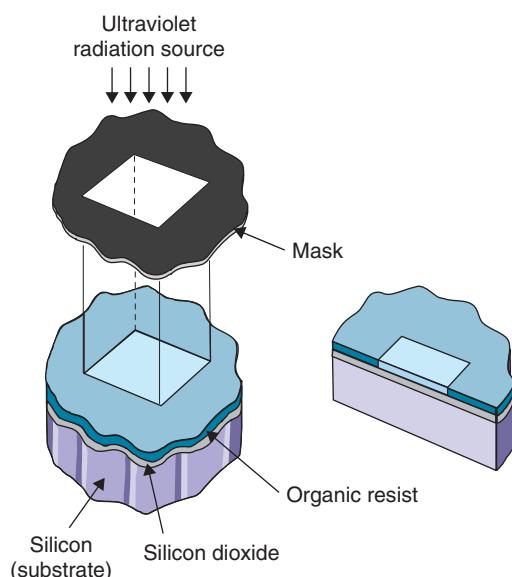
Add a layer of organic resist.

somewhere on the wafer. During the fabrication process, the wafer is often referred to as the *substrate*, meaning “base layer.” A common first stage is to either grow or deposit a thin layer of silicon dioxide (glass) across the entire surface of the wafer by exposing it to oxygen in a high-temperature oven (Figure 14.4).

After the wafer has cooled, it is coated with a thin layer of organic resist,⁸ which is first dried and then baked to form an impervious layer (Figure 14.5).

A mask is created and ultraviolet light is applied. The ionizing ultraviolet radiation passes through the transparent areas of the mask into the resist, silicon dioxide, and silicon. The ultraviolet breaks down the molecular structure of the resist, but does not have any effect on the silicon dioxide or the pure silicon (Figure 14.6).

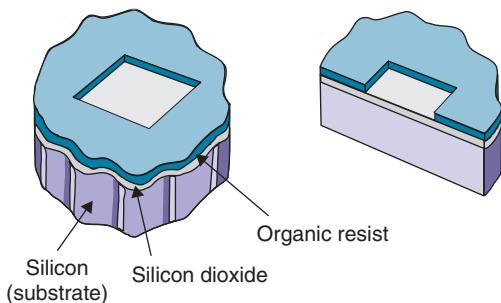
As was previously noted, the small area of the mask shown here is associated with a single transistor. The full mask for a high-end integrated circuit can comprise hundreds of millions (sometimes billions) of similar patterns.⁹ After the area under the mask has

**FIGURE 14.6**

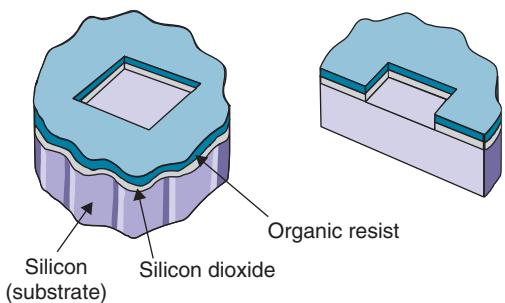
Use ultraviolet light to degrade exposed resist.

⁸The term *organic* is used because this type of resist is a carbon-based compound, and carbon is the key element for life as we know it.

⁹For the purpose of these discussions, I’m thinking of high-end digital integrated circuits containing tens or hundreds of millions of logic gates. It is, of course, possible to have simpler devices containing much fewer elements. In fact, only a couple of days ago as I pen these words, I was chatting with an analog designer who had just designed a chip containing only nine painstakingly handcrafted transistors.

**FIGURE 14.7**

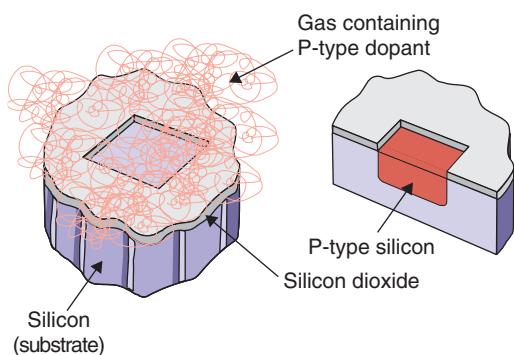
Dissolve the degraded resist with an organic solvent.

**FIGURE 14.8**

Etch the exposed silicon dioxide.

been exposed, the wafer is moved, and the process is repeated until the pattern has been replicated across the entire surface of the wafer, once for each integrated circuit. The wafer is then bathed in an organic solvent to dissolve the degraded resist. Thus, the pattern on the mask has been transferred to a series of corresponding patterns in the resist (Figure 14.7).

A process by which ultraviolet light passing through the transparent areas of the mask causes the resist to be degraded is known as a *positive-resist* process; *negative-resist* processes are also available. In a negative-resist process, the ultraviolet radiation passing through the transparent areas of the mask is used to cure (harden) the resist, and the remaining uncured areas are then removed using an appropriate solvent.

**FIGURE 14.9**

Dope the exposed silicon.

After the unwanted resist has been removed, the wafer undergoes a process known as *etching*, in which an appropriate solvent is used to dissolve any exposed silicon dioxide without having any effect on the organic resist or the pure silicon (Figure 14.8).

The remaining resist is removed using an appropriate solvent. Next, the wafer is placed in a high-temperature oven where it is exposed to a gas containing the selected dopant (a P-type dopant in this case). The atoms in the gas diffuse into the substrate, resulting in a region of doped silicon (Figure 14.9).¹⁰

The remaining silicon dioxide layer is removed by means of an appropriate solvent that doesn't affect the silicon substrate (including the doped regions).

¹⁰In some processes, diffusion is augmented with *ion implantation techniques*, in which beams of ions (which were introduced in *Chapter 2: Atoms, Molecules, and Crystals*) are fired at the wafer to alter the type and conductivity of the silicon in selected regions.

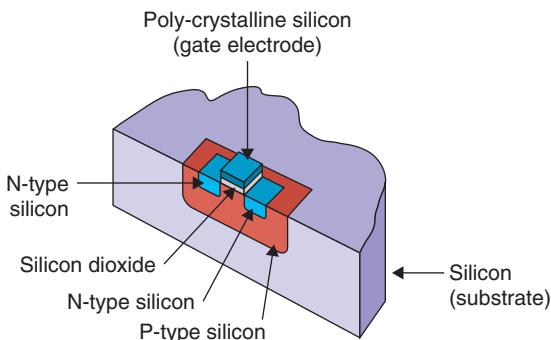


FIGURE 14.10
Add N-type diffusion regions and the gate electrode.

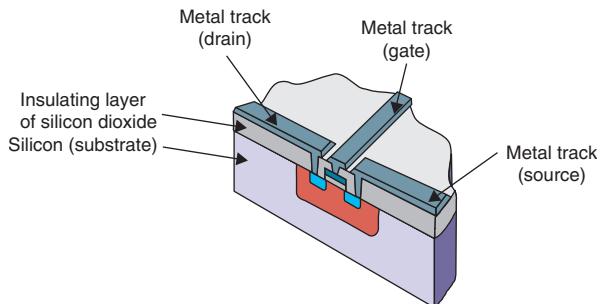


FIGURE 14.11
Add a layer of metal tracks.

Additional masks and variations on the process are used to create two N-type diffusion regions, a gate electrode, and a layer of insulating silicon dioxide between the substrate and the gate electrode (Figure 14.10).

In the original MOS technologies, the gate electrode was metallic: hence, the “metal-oxide semiconductor” appellation. In modern processes, however, the gate electrode is formed from *poly-crystalline silicon* (often abbreviated to *poly-silicon* or even just *poly*), which is also a good conductor.

The N-type diffusions form the transistor’s *source* and *drain* regions (you might wish to refer back to *Chapter 4: Semiconductors (Diodes and Transistors)* to refresh your memory at this point). The gap between the source and drain is called the *channel*. To provide a sense of scale, the length of the channel in one of today’s state-of-the-art technologies is measured in a few tens of billionths of a meter (see also the discussions on *device geometries* later in this chapter).

Another layer of insulating silicon dioxide is now grown or deposited across the surface of the wafer. Using lithographic techniques similar to those described above, holes are etched through the silicon dioxide in areas in which it is desired to make connections, and a *metallization layer* of interconnections called *tracks* (you can think of them as wires) is deposited (Figure 14.11).^{11,12}

¹¹In the early days, the tracks were formed out of aluminum, because this didn’t react with (diffuse into) the insulating silicon dioxide. As the structures on chips got smaller and the tracks got thinner and narrower, their resistance increased; eventually, aluminum could no longer do the job. Thus, modern chips use copper interconnect; this requires more steps to isolate the copper from the silicon dioxide, but it’s worth it because copper is a much better conductor than aluminum.

¹²Silver is the best electrical and thermal conductor of any metal (followed by copper and then by gold), but it’s even harder to work with than copper when it comes to building silicon chips.

The end result is an NMOS transistor; a logic 1 on the track connected to the *gate* terminal will turn the transistor ON, thereby enabling current to flow between its *source* and *drain* terminals. An equivalent PMOS transistor could have been formed by exchanging the P-type and N-type diffusion regions. By varying the structures created by the masks, components such as resistors and diodes can be fabricated at the same time as the transistors. The tracks are used to connect groups of transistors to form primitive logic gates and to connect groups of these gates to form more complex functions.

An integrated circuit contains three distinct levels of conducting material: the *diffusion* layer at the bottom, the *polysilicon* layers in the middle, and the *metallization* layers at the top. In addition to forming components, the diffusion layer may also be used to create embedded wires. Similarly, in addition to forming gate electrodes, the polysilicon may also be used to interconnect components. There may be several layers of polysilicon and multiple layers of metallization, with each pair of adjacent layers separated by an insulating layer of silicon dioxide. The layers of silicon dioxide are selectively etched with holes that are filled with conducting metal and are known as *vias*; these allow connections to be made between the various tracking layers.

Early integrated circuits typically supported only two layers of metallization. The tracks on the first layer predominantly ran in a “North-South” direction, while the tracks on the second predominantly ran “East-West.”¹³ As the number of transistors increased, engineers required more and more tracking layers. The problem is that when a layer of insulating silicon dioxide is deposited over a tracking layer, you end up with slight “bumps” where the tracks are (like snow falling over a snoozing polar bear—you end up with a bump).

After a few tracking layers, the bumps are pronounced enough that you can’t continue. The answer is to re-planarize the wafer (smooth the bumps out) after each tracking and silicon dioxide layer combo has been created. This is achieved by means of a process called *Chemical Mechanical Polishing* (CMP), which returns the wafer to a smooth, flat surface before the next tracking layer is added. Using this process, high-end silicon chips can support up to ten tracking layers.

¹³In 2001, a group of companies announced a new chip interconnect concept called *X Architecture* in which logic functions on chips are wired together using diagonal tracks (as opposed to traditional North-South and East-West tracking layers). It is claimed that this diagonal interconnect strategy can increase chip performance by 10% and reduce power consumption by 20%. I know of a couple of chips that have been fabricated using this technology, but it has not yet gained widespread adoption.

AN INTRODUCTION TO THE PACKAGING PROCESS

There are a wide variety of different packaging techniques. We'll start by considering one of the simplest packaging styles (one that's easy to understand) and then we'll consider some slightly more complex techniques. (We'll also look at some really sophisticated packaging technologies in *Chapter 20: Advanced Packaging Techniques*.)

In the case of our simple packaging technique, relatively large areas of aluminum or copper called *pads* are constructed at the edges of each integrated circuit for testing and connection purposes. Some of the pads are used to supply power to the device, while the rest are used to provide input and output signals to the components in the chip (Figure 14.12).

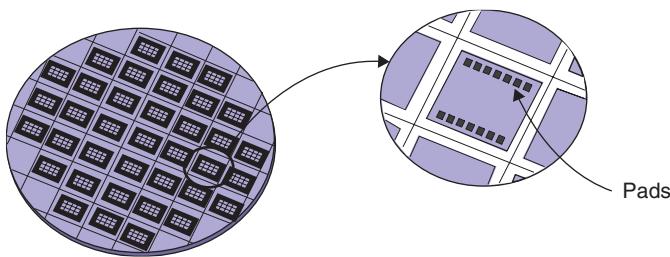


FIGURE 14.12

Power and signal pads.

In a step known as *overglossing*, the entire surface of the wafer is coated with a final *barrier* layer (or *passivation* layer) of silicon dioxide or silicon nitride, which provides physical protection for the underlying circuits from moisture and other contaminants. One more lithographic step is required to pattern holes in the barrier layer to allow connections to be made to the pads. In some cases, additional metallization may be deposited on the pads to raise them fractionally above the level of the barrier layer. Augmenting the pads in this way is known as *silicon bumping*.

The entire fabrication process requires numerous lithographic steps, each involving an individual mask and layer of resist to selectively expose different parts of the wafer.

The individual integrated circuits are tested while they are still part of the wafer in a process known as *wafer probing*. An automated tester places probes on the device's pads, applies power to the power pads, injects a series of signals into the input pads, and monitors the corresponding signals returned from

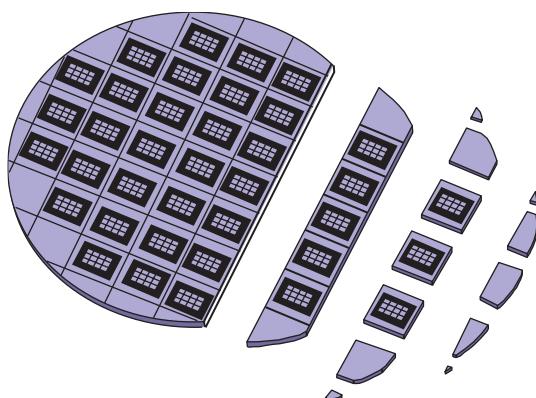


FIGURE 14.13

Die separation.

the output pads. Each integrated circuit is tested in turn, and any device that fails the tests is automatically tagged with a splash of dye for subsequent rejection. The *yield* is the number of devices that pass the tests as a percentage of the total number fabricated on that wafer.

The completed circuits, known as *die*,¹⁴ are separated by marking the wafer with a diamond scribe and fracturing it along the scribed lines (much like cutting a sheet of glass or breaking up a Kit Kat® bar) (Figure 14.13).

Following separation, the majority of the die are packaged individually. Since there are almost as

many packaging technologies as there are device manufacturers, we will initially restrain ourselves to a low-end “cheap-and-cheerful” process. First, the die is attached to a metallic lead frame using an adhesive (Figure 14.14).

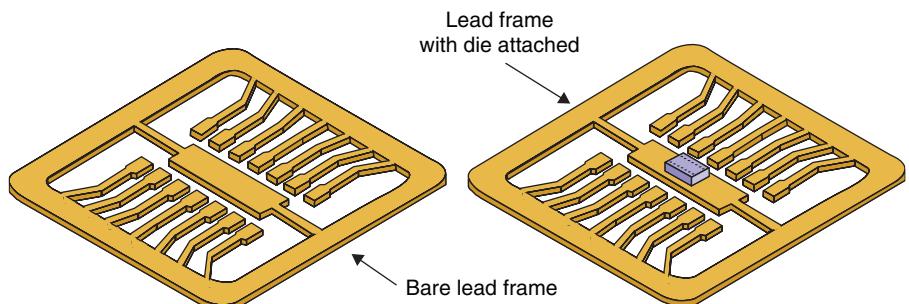


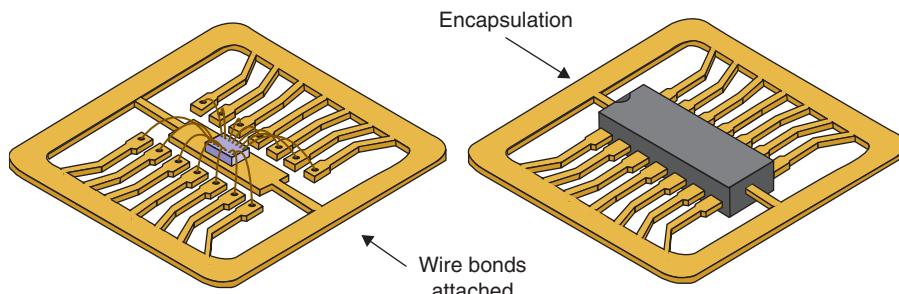
FIGURE 14.14

The die is attached to a lead frame.

One of the criteria used when selecting the adhesive is its ability to conduct heat away from the die when the device is operating. An automatic wire-bonding tool connects the pads on the die to the leads on the lead frame with wire bonds finer than a human hair.¹⁵ The whole assembly is then encapsulated in a block of plastic or epoxy (Figure 14.15).

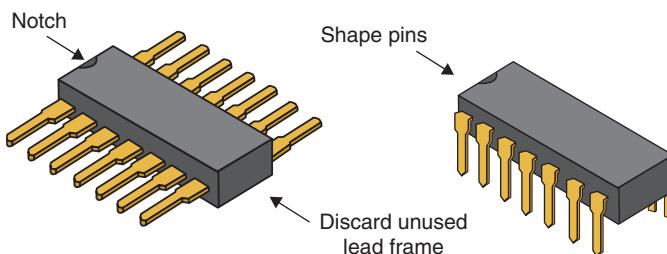
¹⁴The plural of *die* is also *die* (in much the same way that *herring* is the plural of *herring* as in “a shoal of herring”).

¹⁵Human hairs range in thickness from around 0.07 mm to 0.1 mm. A hair from a typical blond lady’s head is approximately 0.075 mm (three-quarters of one-tenth of a millimeter) in diameter. By comparison, integrated circuit bonding wires are typically one-third this diameter, and they can be even thinner.

**FIGURE 14.15**

Wire bonding and encapsulation.

A dimple or notch is formed at one end of the package so that the users will know which end is which. The unused parts of the lead frame are cut away and the device's leads, or pins, are shaped as required; these operations are usually performed at the same time (Figure 14.16).

**FIGURE 14.16**

Discard unused lead frame and shape pins.

An individually packaged integrated circuit consists of the die and its connections to the external leads, all encapsulated in the protective package. The package protects the silicon from moisture and other impurities and helps to conduct heat away from the die when the device is operating.

As was previously noted, there is tremendous variety in the size and shape of packages. A rectangular device with pins on two sides, as illustrated here, is called a *Dual In-Line* (DIL) package or a DIP.¹⁶ A standard 14-pin packaged device is approximately 18 mm long by 6.5 mm wide by 2.5 mm deep, and has 2.5-mm spaces between pins. An equivalent *Small Outline Package* (SOP) could be as small as 4 mm long by 2 mm wide by 0.75 mm deep, and have 0.5-mm

¹⁶Invented at Fairchild in 1965, DIL packages were the mainstay of the industry through the 1970s and 1980s. Their use started to decline in the 1990s as *Surface Mount Technology* (SMT) gained in popularity, but you can still find new components from some manufacturers in these packages to this day; for example, Microchip Technology (<http://www.microchip.com>) continue to provide many of their latest-and-greatest PIC® microcontrollers in both DIL and SMT packages.

spaces between pins. Other packages can be square and have pins on all four sides, and some have an array of pins protruding from the base.

The shapes into which the pins are bent depend on the way the device is intended to be mounted on a circuit board. The package described above has pins that are intended to go all the way through the circuit board using a mounting technique called *Lead Through Hole* (LTH). By comparison, the packages associated with a technique called *Surface Mount Technology* (SMT) have pins that are bent out flat, and which attach to only one side (surface) of the circuit board [an example of this is shown in *Chapter 18: Printed Circuit Boards (PCBs)*].

It's important to remember that the example discussed above reflects a very simple packaging strategy for a device with very few pins. By 2002, some integrated circuits (and their packages) had as many as 1000 pins; by 2008, some high-end components were available with 2000 pins or more. This multiplicity of pins requires a very different packaging approach. For example, the pads on the die are no longer restricted to its periphery, but are instead located over the entire face of the die. A minute ball of solder is then attached to each pad, and the die is flipped over and attached to the package substrate (this is referred to as a *flip-chip* technique). Each pad on the die has a corresponding pad on the package substrate, and the package-die combo is heated so as to melt the solder balls and form good electrical connections between the die and the substrate (Figure 14.17).

Eventually, the die will be encapsulated in some manner to protect it from the outside world. The package's substrate itself may be made out of the same

material as a printed circuit board, or out of ceramic, or out of some even more esoteric material. Whatever its composition, the substrate will contain multiple internal wiring layers that connect the pads on the upper surface to pads (or pins) on the lower surface. The pads (or pins) on the lower surface (the side that actually connects to the circuit board) will be spaced much further apart—relatively speaking—than the pads that connect to the die. At some stage the package will have to be attached to a circuit board. In one technique known as a *Ball Grid Array* (BGA), the package has an array of pads on its bottom surface, and a small ball of solder

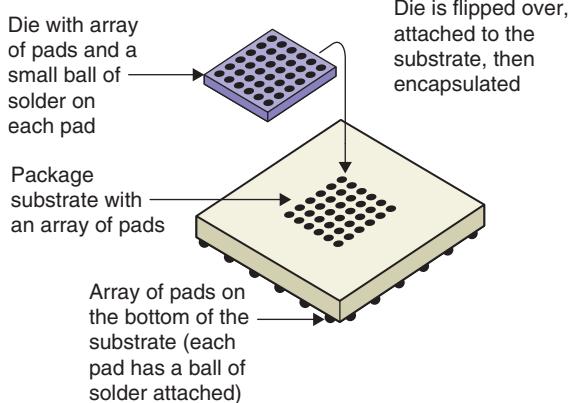


FIGURE 14.17

A flip-chip ball grid array packaging technique.

is attached to each of these pads. Each pad on the package will have a corresponding pad on the circuit board, and heat is used to melt the solder balls and form good electrical connections between the package and the board.

Modern packaging technologies are extremely sophisticated. Some ball grid arrays have pins spaced 0.3 mm (one-third of a millimeter) apart! In the case of *Chip-Scale Package* (CSP) technology, the package is barely larger than the die itself. In the early 1990s, some specialist applications began to employ a technique known as *die stacking*, in which several bare die are stacked on top of each other to form a sandwich. The die are connected together and then packaged as a single entity.

As was previously noted, there are a wide variety of integrated packaging styles. There are also many different ways in which the die can be connected to the package. We will introduce a few more of these techniques in future chapters.¹⁷

INTEGRATED CIRCUITS VERSUS DISCRETE COMPONENTS

The tracks that link components inside an integrated circuit have widths measured in fractions of a millionth of a meter, and lengths measured in millimeters. By comparison, the tracks that link components on a circuit board are orders of magnitude wider, and have lengths measured in tens of centimeters. Thus, the transistors used to drive tracks inside an integrated circuit can be much smaller than those used to drive their circuit board equivalents, and smaller transistors use less power. Additionally, signals take a finite time to propagate down a track, so the shorter the track, the faster the signal.

A single integrated circuit can contain hundreds of millions of transistors (see also the topic titled *How Many Transistors?* later in this chapter). A similar design based on discrete components would be tremendously more expensive in terms of price, size, operating speed, power requirements, and the time and effort required to design and manufacture the system. Additionally, every solder joint on a circuit board is a potential source of failure, which affects the reliability of the design. Integrated circuits reduce the number of solder joints and, hence, improve the reliability of the system.

¹⁷Additional packaging styles and alternative mounting strategies are presented in *Chapter 18: Printed Circuit Boards (PCBs)*, *Chapter 19: Hybrids*, and *Chapter 20: Advanced Packaging Techniques*.

In the past, an electronic system was typically composed of a number of integrated circuits, each with its own particular function (say a microprocessor, some peripheral functions, some memory devices, etc.). For many of today's high-end applications, however, electronics engineers are combining all of these functions on a single device, which may be referred to as a *System-on-Chip* (SoC).

DIFFERENT TYPES OF ICS

The first integrated circuit—a simple phase-shift oscillator—was constructed in 1958. Since that time, a plethora of different device types have appeared on the scene. There are far too many different integrated circuit types for us to cover in this book, but some of the main categories—along with their approximate dates of introduction—are shown in Figure 14.18.¹⁸

Memory devices (in particular SRAMs and DRAMs) are introduced in *Chapter 15: Memory ICs*; programmable integrated circuits (PLDs, CPLDs, FPGAs, and CSSPs) are presented in *Chapter 16: Programmable ICs*; and *Application-Specific Integrated Circuits* (ASICs) are discussed in *Chapter 17: Application-Specific Integrated Circuits (ASICs)*.

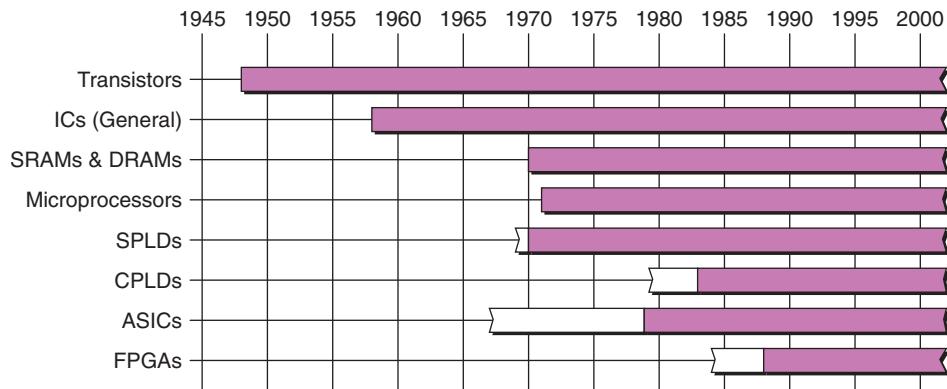


FIGURE 14.18

Timeline of device introductions (dates are approximate).

¹⁸The white portions of the timeline bars in Figure 14.18 indicate that although early incarnations of these technologies may have been available, they perhaps hadn't been enthusiastically received during this period. For example, Xilinx introduced the world's first FPGA as early as 1984, but many design engineers didn't really become interested in these little rascallions until the late 1980s.

TTL, ECL, AND CMOS

Transistors are available in a variety of flavors called *families* or *technologies*. One of the first to be invented was the *Bipolar Junction Transistor* (BJT), which was the mainstay of the industry for many years. If bipolar transistors are connected together in a certain way, the resulting logic gates are classed as *Transistor-Transistor Logic* (TTL). An alternative method of connecting the same transistors results in logic gates classed as *Emitter-Coupled Logic* (ECL).

Another family called *Metal-Oxide Semiconductor Field-Effect Transistors* (MOSFETs) were invented some time after bipolar junction transistors. *Complementary Metal-Oxide Semiconductor* (CMOS) logic gates are based on NMOS and PMOS MOSFETs connected together in a complementary manner.

Some integrated circuits use a combination of technologies; in the case of *Bipolar CMOS* (BiCMOS), for example, the function of every primitive logic gate is implemented in low-power CMOS, but the output stage of each gate uses high-drive bipolar transistors.

Finally, gates fabricated using the gallium arsenide (GaAs) semiconductor as a substrate are faster than their silicon equivalents, but they are expensive to produce, and so are used for specialized applications only.

CORE SUPPLY VOLTAGES

Toward the end of the 1980s and the beginning of the 1990s, the majority of digital integrated circuits were based on a 5.0-volt supply. However, increasing usage of portable personal electronics, such as notebook computers and cellular telephones, began to drive the requirement for devices that consume and dissipate less power.

One way to reduce power consumption is to lower the supply voltage; thus, by the mid to late 1990s, the most common supplies were 3.3 volts for portable computers and 3.0 volts for communication systems.

The core supply voltage continued to fall over the years [by *core* we mean the supply voltage used to drive the internals of the silicon chip; the input/output (I/O) pins may use higher voltages]. Leading-edge devices were using 2.5 volts by 1999, 1.8 volts by 2000, 1.5 volts by 2001, and 1.2 volts by 2003. At the time of this writing in 2008, some cutting-edge devices have core supply voltages as low as 0.9 volts.

EQUIVALENT GATES

One common metric used to categorize a digital integrated circuit is the number of logic gates it contains. However, difficulties may arise when comparing devices, as each type of logic function requires a different number of transistors. This leads to the concept of an *equivalent gate*, whereby each type of logic function is assigned an equivalent gate value, and the relative complexity of an integrated circuit is judged by summing its equivalent gates.

Unfortunately, the definition of an equivalent gate can vary, depending on whom one is talking to. A reasonably common convention is for a 2-input NAND to represent one equivalent gate. A more esoteric convention defines an ECL equivalent gate as being “*one-eleventh the minimum logic required to implement a single-bit full-adder*,” while some vendors define an equivalent gate as being equal to an arbitrary number of transistors based on their own particular technology. The best policy is to establish a common frame of reference before releasing a firm grip on your hard-earned lucre.

The acronyms SSI, MSI, LSI, VLSI, and ULSI represent *Small-, Medium-, Large-, Very-Large-, and Ultra-Large-Scale Integration*, respectively. By one convention, the number of gates represented by these terms are: SSI (1–12), MSI (13–99), LSI (100–999), VLSI (1000–999,999), and ULSI (1,000,000 or more).¹⁹

DEVICE GEOMETRIES

Integrated circuits are also categorized by their *geometries*, meaning the size of the structures created on the substrate. For example, a 1-μm²⁰ CMOS device would have structures that measure one-millionth of a meter. The structures typically embraced by this description are the width of the tracks and the length of the channel between the source and drain diffusion regions; the dimensions of other features are derived as ratios of these structures.²¹

¹⁹Truth to tell, not many folks use these terms anymore. They've become increasingly irrelevant as designers have moved away from using simple *jelly bean* components and chips have increased in complexity into the hundreds of millions (or billions) of transistors.

²⁰The “μ” symbol stands for *micro* from the Greek *micros*, meaning “small,” (hence, the use of μP and μC as abbreviations for microprocessor and microcontroller, respectively). In the metric system, μ stands for “one millionth part of,” so 1 μm means “one-millionth of a meter.”

²¹I'm simplifying things here; there are different ways of measuring things depending on the type of component; for example, memory chips versus devices containing general-purpose logic gates.

Each new geometry may be referred to as a *technology node* or a *process node*. Geometries are continuously shrinking as fabrication processes improve. In 1990, devices with 1- μm geometries were considered to be state of the art, and many observers feared that the industry was approaching the limits of manufacturing technology, but geometries continued to shrink regardless as illustrated in Table 14.1.

Around the time we reached the 0.5- μm technology node, it became common to refer to anything below this point as *Deep-Submicron* (DSM). Later, at some point that wasn't particularly well defined (or was defined differently by different people) we moved into the realm of *Ultra-Deep-Submicron* (UDSM). However, no one tends to use these terms anymore, because we've now travelled so deep into the rabbit hole that we've run out of meaningful qualifying words.

Table 14.1 Evolution of Technology Nodes

Year	Node
1990	1.00 μm
1992	0.80 μm
1994	0.50 μm
1996	0.35 μm
1997–1998	0.25 μm
1999–2000	180 nm
2001	130 nm
2001	100 nm
2003	90 nm
2005–2006	65 nm
2008	45 nm
2008	40 nm
2009–2010*	32 nm
2011–2012*	22 nm
2013–2014*	16 nm

*“Finger-in-the-air” predicted dates

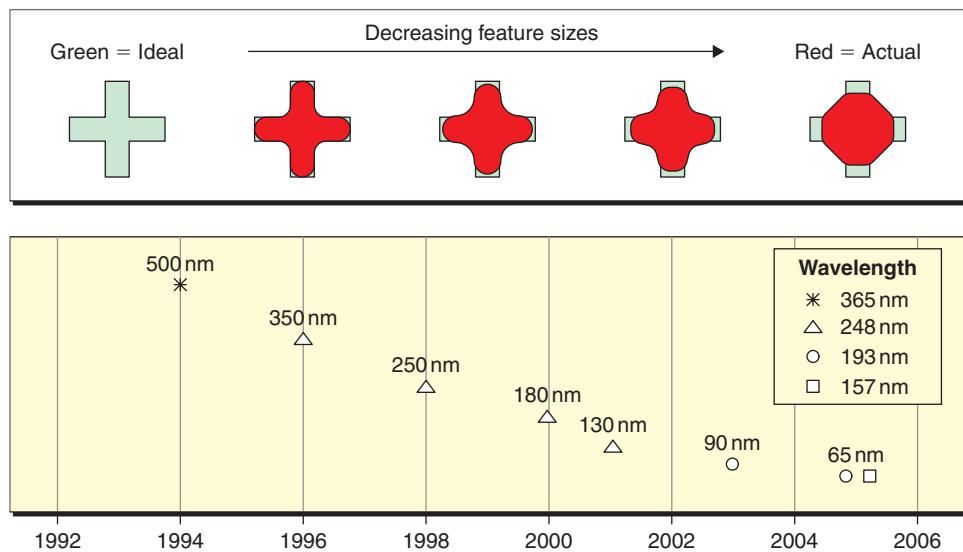
With devices whose geometries were $1\text{ }\mu\text{m}$ and higher, it was relatively easy to talk about them in conversation. For example, one might say "*I'm working with a one-micron technology.*" But things started to get a little awkward when we dropped below $1\text{ }\mu\text{m}$, because it's a bit of a pain to have to keep on saying things like "*zero-point-one-three microns.*" For this reason, sometime around the end of the 20th century, it became common to talk in terms of "*nano*," where one *nano* (short for *nanometer*) equates to one one-thousandth of a micron; that is, one-thousandth of one-millionth of a meter. Thus, when referring to a 130-nm ($0.13\text{ }\mu\text{m}$) technology, instead of mumbling "*zero-point-one-three microns,*" you could now proudly proclaim "*one-hundred and thirty nano.*" Of course, both of these mean exactly the same thing, but if you want to talk about this sort of stuff, it's best to use the vernacular of the day and present yourself as hip and trendy as opposed to an old fuddy-duddy from the last millennium.

While smaller geometries result in lower power consumption and higher operating speeds, these benefits do not come without a price. Logic gates implemented in nanometer technologies exhibit extremely complex timing effects, which make corresponding demands on designers and design tools. Additionally, all materials are naturally radioactive to some extent, and the materials used to package integrated circuits can spontaneously release alpha particles. Devices with smaller geometries are more susceptible to the effects of noise, and the alpha decay in packages can cause corruption of the data being processed by deep-submicron logic gates. Nanometer technologies also suffer from a phenomenon known as *subatomic erosion* or, more correctly, *electromigration*, in which the structures in the silicon are eroded by the flow of electrons in much the same way as land is eroded by a river.

WHAT COMES AFTER OPTICAL LITHOGRAPHY?

Although new techniques are constantly evolving, technologists can foresee the limits of miniaturization that can be practically achieved using optical lithography. These limits are ultimately dictated by the wavelength of ultraviolet radiation.

In fact, the features (structures) on a silicon chip are now smaller than the wavelength of the light used to create them (Figure 14.19). If we assume that the green geometric shape shown in this illustration is the ideal (desired) form, then this is the shape that would be generated by the design tools. The problem is that, if this shape were to be replicated as-is in the photo-mask, then the corresponding form appearing on the silicon would drift farther and farther from the ideal with the decreasing feature sizes associated with the newer technology nodes.

**FIGURE 14.19**

Technology nodes versus the ultraviolet wavelengths used to create them.

The way this is currently addressed in conventional design flows is for the manufacturing group to post-process the design files with a variety of *Resolution Enhancement Techniques* (RET), such as *Optical Proximity Correction* (OPC) and *Phase Shift Mask* (PSM). For example, they may modify the original design files by augmenting existing features or adding new features—known as *Sub-Resolution Assist Features* (SRAF)—so as to obtain better printability. One way to visualize this is that if the manufacturing group (actually, their tools) knows that the features will be distorted by the printing (imaging) process in certain ways, they can add their own distortions in the “opposite direction” in an attempt to make the two distortions cancel each other out.

The technology has now passed from using *Standard Ultraviolet* (UV) to *Extreme Ultraviolet* (EUV), which is just this side of soft X-rays in the electromagnetic spectrum. One potential alternative is true X-ray lithography, but this requires an intense X-ray source and is considerably more expensive than optical lithography. Another possibility is *electron beam lithography* (often abbreviated to *e-beam lithography*), in which fine electron beams are used to draw extremely high-resolution patterns directly into the resist without a mask. Electron beam lithography is sometimes used for custom and prototype devices, but it is much slower and more expensive than optical lithography. Thus, for the present, it would appear that optical lithography will continue to be the mainstay of mass-produced integrated circuits.

HOW MANY TRANSISTORS?

The first integrated circuits typically contained around six transistors. By the latter half of the 1960s, devices containing around 100 transistors were reasonably typical.

In the first half of 2002, Intel announced its McKinley microprocessor—an integrated circuit based on a 130-nano ($0.13\text{-}\mu\text{m}$) process, containing more than 200 million transistors. And by the summer of 2002, Intel had announced a test chip based on a 90-nano ($0.09\text{-}\mu\text{m}$) process that contained 330 million transistors.

On May 19, 2008 (a few days ago at the time of this writing) a company called Altera (<http://www.altera.com/>) announced a new family of FPGAs (see *Chapter 16: Programmable ICs*), the largest member of which contains 2.5 billion transistors! At this level of processing power, we will soon have the capabilities required to create Star Trek-style products, like a universal real-time language translator.

MOORE'S LAW

You can't read a technical paper these days without blundering into some mention of Moore's Law, so it behooves us to explain this here. In 1965, Gordon Moore (1929–) (who was to cofound Intel Corporation in 1968) was preparing a speech that included a graph illustrating the growth in the performance of memory ICs. While plotting this graph, Moore realized that new generations of memory devices were released approximately every 18 months, and that each new generation of devices contained roughly twice the capacity of its predecessor.

This observation subsequently became known as *Moore's Law*, and it has been applied to a wide variety of electronics trends. These include the number of transistors that can be constructed in a certain area of silicon (the number doubles approximately every 18 months), the price per transistor (which follows an inverse Moore's Law curve and halves approximately every 18 months), and the performance of microprocessors (which again doubles approximately every 18 months).

CHAPTER 15

Memory ICs

RAMS AND ROMS

Memory devices are a special class of integrated circuits that are used to store binary data for later use. There are two main categories of semiconductor memories: *Read-Only Memory* (ROM) and *Read-Write Memory* (RWM).¹ Other components in the system can read (extract) data from ROM devices, but cannot write (insert) new data into them. By comparison, data can be read out of RWM devices and, if required, new data can be written back into them. The act of reading data out of a RWM does not affect the master copy of the data stored in the device.² For a number of reasons, mainly historical, RWMs are more commonly known as *Random-Access Memories* (RAMs).

193

ROM and RAM devices find a large number of diverse applications in electronic designs, but their predominant usage is as memory in computer systems, so that's what we'll focus on here (Figure 15.1).³

The brain of the computer is the *Central Processing Unit* (CPU), which is where all of the number crunching and decision making are performed. The CPU uses a set of signals called the *address bus* to point to the memory location in which it is interested. The address bus is said to be *unidirectional* because it conveys information in only one direction: from the CPU to the memory. By means of control signals, the CPU either reads data from, or writes data to, the selected

¹There's also an engineering joke called a *Write-Only Memory* (WOM). The idea behind this mythical beast is that you can write data into it, but you can't read that data back out again.

²This is true of modern semiconductor memories. In the case of the original magnetic core store memories, the act of reading a word of data out of the core set that word to all logic 0s or all logic 1s (this was called a *destructive read*), so extra circuitry had to be employed to automatically rewrite the data back in.

³In conversation, ROM is pronounced as a single word to rhyme with "bomb," while RAM is pronounced to rhyme with "ham."

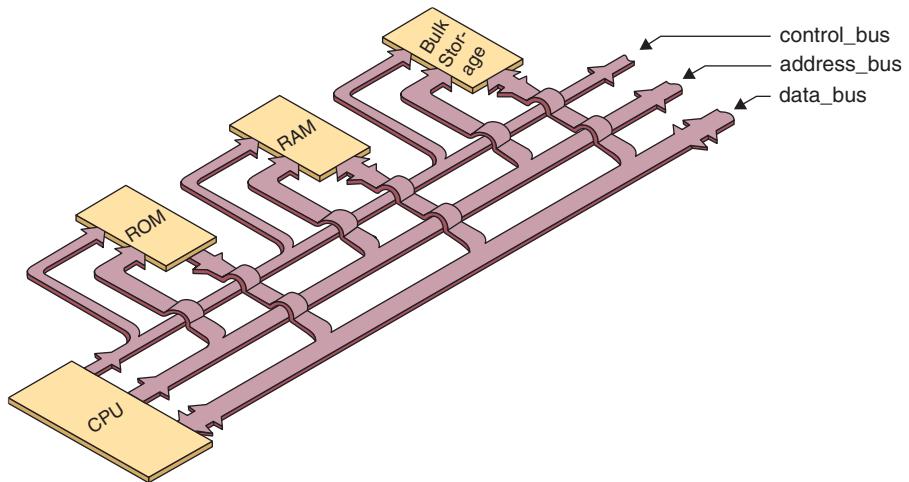


FIGURE 15.1
ROM and RAM in a computer system.

memory location. The data is transferred on a set of signals called the *data bus*, which is said to be *bidirectional* because it can convey information in two directions: from the CPU to the memory and vice versa.

The bulk storage is often based on magnetic media such as a hard disk drive, which can be used to store a large quantity of information relatively cheaply. Because magnetic media maintains its data when power is removed from the system it is said to be *nonvolatile*.

One of the major disadvantages of currently available bulk storage units is their relatively slow speed.⁴ The CPU can process data at a much higher rate than the bulk storage can supply or store it. Semiconductor memories are significantly more expensive than bulk storage, but they are also a great deal faster.

ROMs are also classed as being *nonvolatile*, because their data remains when power is removed from the system. By comparison, RAM devices initialize containing random logic 0 or logic 1 values when power is first applied to a system. Thus, any meaningful data stored inside a RAM must be written into it by other components in the system after it has been powered-up. Additionally, RAMs are said to be *volatile*, because any data they contain is lost when power is removed from the system.

When a computer system is first powered-up, it doesn't know much about anything. The CPU is hard-wired so that the first thing it does is read an instruction

⁴Note the use of the "relatively" qualifier; modern bulk storage is actually amazingly fast, but not as fast as the rest of the system.

from a specific memory address: for example, address zero. The components forming the system are connected together in such a way that this hard-wired address points to the first location in a block of ROM.⁵ The ROM contains a sequence of instructions that are used by the CPU to initialize both itself and other parts of the system. This initialization is known as *boot-strapping*, which is derived from the phrase “pulling yourself up by your boot-straps.” At an appropriate point in the initialization sequence, instructions in the ROM cause the CPU to copy a set of master programs, known collectively as the *Operating System* (OS), from the bulk storage into the RAM. Finally, the instructions in the ROM direct the CPU to transfer its attention to the operating system instructions in the RAM, at which point the computer is ready for the user to enter the game.

CELLS, WORDS, AND ARRAYS

The smallest unit of memory, called a *cell*, can be used to store a single bit of data: that is, a logic 0 or a logic 1. A number of cells physically grouped together are classed as a *word*, and all the cells in a word are typically written to, or read from, at the same time. The core of a memory device is made up of a number of words arranged as an *array* (Figure 15.2).

The “width” (w) of a memory is the number of bits used to form a *word*,⁶ where the bits are usually numbered from 0 to $(w - 1)$. Similarly, the “depth” (d) of a

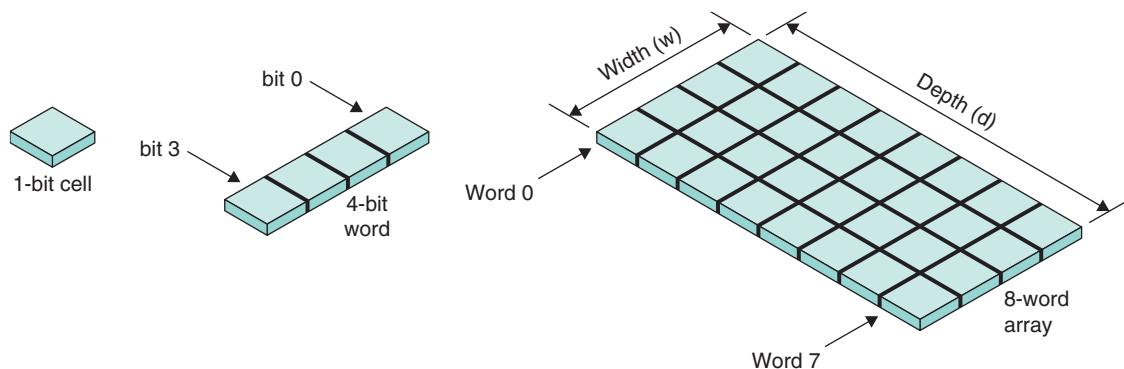


FIGURE 15.2

Memory cells, words, and arrays.

⁵If the truth be told, early computers used ROM, but these days they tend to employ another form of memory called FLASH, which is nonvolatile like ROM, but which can be reprogrammed (if necessary) like RAM (FLASH memory is introduced later in this chapter).

⁶Note that there is no official definition as to the width of a word: this is always system-dependent.

memory is the number of words used to form the array, where the words are usually numbered from 0 to $(d - 1)$. The following examples assume a memory array that is four bits wide and eight words deep; of course, real devices can be much wider and deeper and can contain humungous amounts of data.

ADDRESSING A WORD IN MEMORY

For external components (such as a CPU) to reference a particular word in the memory, they must specify that word's address by placing appropriate values onto the address bus. The address is decoded inside the device, and the contents of the selected word are made available as outputs from the array (Figure 15.3).

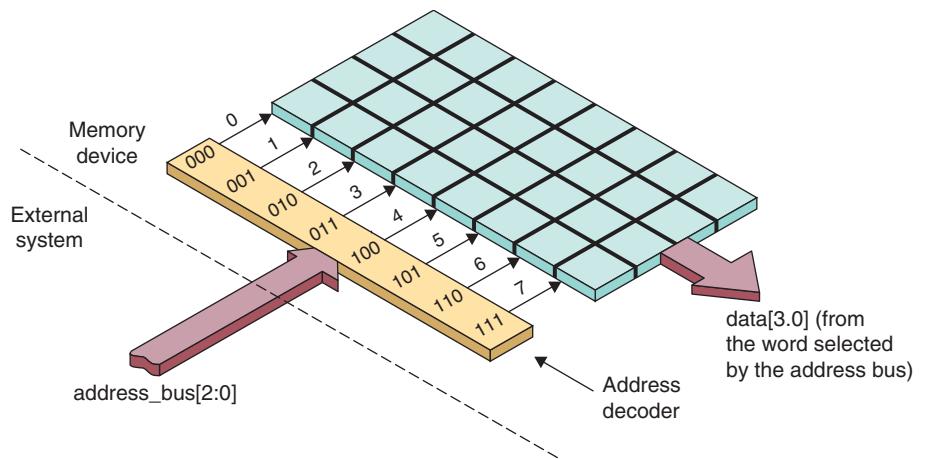


FIGURE 15.3

Address bus decoding.

Standard memory devices are constrained to have a depth of 2^n words, where n is the number of bits used to form the address bus. For example, the 3-bit address bus illustrated in Figure 15.3 can be decoded to select one of eight words ($2^3 = 8$) using a 3:8 decoder.⁷

KILO, MEGA, GIGA, TERA, ETC.

The fact that standard memory devices are constrained to have a depth of 2^n words leads to an interesting quirk when referring to the size of a memory device. In SI units,⁸ the qualifier "K" (kilo)⁹ represents one thousand (1000),

⁷Decoders were introduced in *Chapter 11: Slightly More Complex Functions*.

⁸The original metric system of measurement was developed during the French Revolution and its use was legalized in the United States in 1866. The International System of Units (SI) is a modernized version of the metric system.

⁹The term *kilo* comes from the Greek *khilioi*, meaning "thousand" (strangely enough, this is the only prefix with an actual numerical meaning).

but the closest power of two to one thousand is 2^{10} , which equals 1024. Therefore, a 1-kilobit (1-kb or 1-Kb)¹⁰ memory actually refers to a device containing 1,024 bits.

Similarly, the qualifier "M" (mega)¹¹ is generally taken to represent one million (1,000,000), but the closest power of two to one million is 2^{20} , which equals 1,048,576. Therefore, a 1-megabit (1-Mb) memory actually refers to a device containing 1,048,576 bits. In the case of the qualifier "G" (giga),¹² which is now generally taken to represent one billion (1,000,000,000),¹³ the closest power of two is 2^{30} , which equals 1,073,741,824. Therefore, a 1-gigabit (1-Gb) memory actually refers to a device containing 1,073,741,824 bits.

And we can keep on going:

$$1 \text{ terabit (1 tb)} = 2^{40} = 1,099,511,627,776 \text{ bits}$$

$$1 \text{ petabit (1 Pb)} = 2^{50} = 1,125,899,906,842,624 \text{ bits}$$

$$1 \text{ exabit (2 Eb)} = 2^{60} = 1,152,921,504,606,846,976 \text{ bits}$$

$$1 \text{ zettabit (1 Zb)} = 2^{70} = 1,180,591,620,717,411,303,424 \text{ bits}$$

$$1 \text{ yottabit (1 Yb)} = 2^{80} = 1,208,925,819,614,629,174,706,176 \text{ bits}$$

and so forth.

BITS AND BYTES

If the width of the memory is equal to a byte or a multiple of bytes, then the size of the memory may also be referred to in terms of bytes. For example, a memory containing 1024 words, each 8 bits wide, may be referred to as being either an 8-kilobit (8-Kb) or a 1-kilobyte (1-KB) device (note the use of "b" and "B" to represent *bit* and *byte*, respectively).

ROM CONTROL DECODING

Because multiple memory devices are usually connected to a single data bus, the data coming out of the internal array is typically buffered from the external

¹⁰The official SI symbol for kilo is a lowercase "k" (for example, 10 kg represents ten kilograms). When referring to computer memories, however, it is usual to use an uppercase "K" (for example 1 Kb); this is because the other symbols used for memory, like mega (M) and giga (G), are uppercase.

¹¹The term *mega* comes from the Greek *megas*, meaning "great" (hence, the fact that Alexander the Great was known as Megas Alexandros in those days).

¹²The term *giga* comes from the Latin *gigas*, meaning "giant."

¹³See the discussions in *Chapter 3: Conductors, Insulators, and Other Stuff*, with regard to the fact that we now take "one billion" to represent one thousand million, rather than one million million.

system by means of tri-state gates.¹⁴ Enabling the tri-state gates allows the device to drive data onto the data bus, while disabling them allows other devices to drive the data bus.

In addition to its address and data buses, a ROM requires a number of control signals, the two most common being $\sim\text{chip_select}$ and $\sim\text{read}$. (The $\sim\text{read}$ control is sometimes called $\sim\text{output_enable}$. Alternatively, some devices have both $\sim\text{read}$ and $\sim\text{output_enable}$ controls.) These control signals are commonly active-low; that is, they are considered to be ON when a logic 0 is applied.¹⁵ The $\sim\text{chip_select}$ signal indicates to the ROM that its attention is required, and it is combined with the $\sim\text{read}$ signal to form an internal signal (called $\sim\text{rd}$ in this example), which is used to control the tri-state gates (Figure 15.4).

When the $\sim\text{rd}$ signal is active (logic 0), the tri-state gates are enabled and the data stored in the word selected by the address bus is driven out onto the data bus. When $\sim\text{rd}$ is inactive, the tri-state gates are disabled and the outputs from the device are placed into high-impedance Z states.

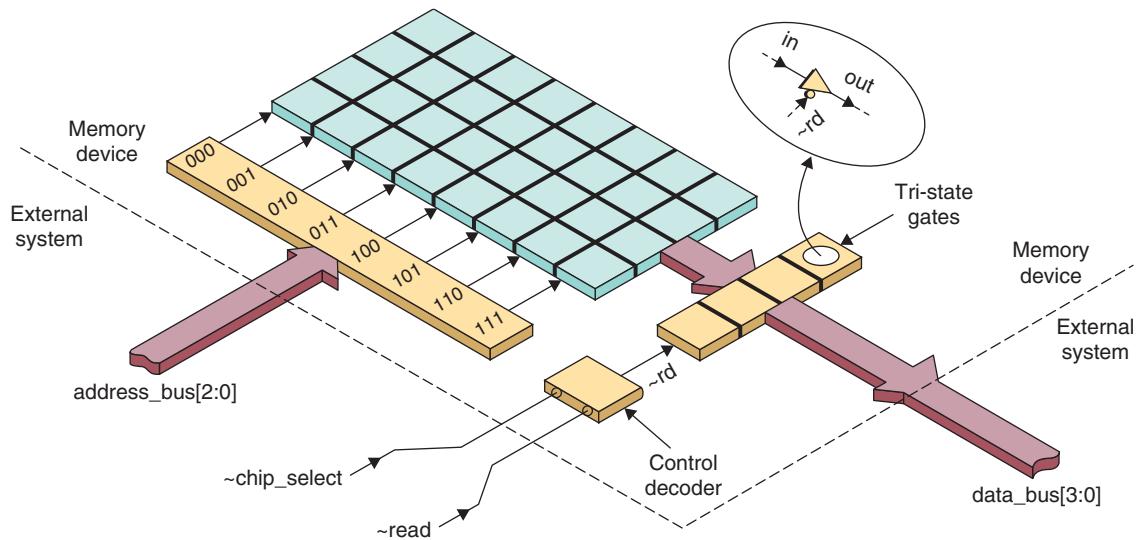


FIGURE 15.4
ROM control decoding and data bus.

¹⁴Tri-state gates were introduced in Chapter 11: Slightly More Complex Functions.

¹⁵Tilde “~” characters prefixing signal names are used to indicate that these signals are active-low. The use of tilde characters is discussed in detail in Appendix A: Assertion-Level Logic.

RAM WITH SEPARATE DATA IN AND DATA OUT BUSSES

In addition to the control signals used by ROMs, RAMs require a mechanism to control the writing of new data *into* the device. These components usually have an additional control signal called something like `~write`, which is also active-low. Once again, the `~chip_select` signal indicates to the RAM that its attention is required, and it is combined with the `~read` signal to form an internal `~rd` signal that is used to control the tri-state gates. Additionally, the `~chip_select` signal is combined with the `~write` signal to form an internal `~wr` signal. First, let's consider a device that employs separate buses for writing data into the memory array and reading data out of the array (Figure 15.5).

When $\sim wr$ is active, the data present on the data in bus is written into the word selected by the address bus. The contents of the word pointed to by the address bus are always available at the output of the array, irrespective of

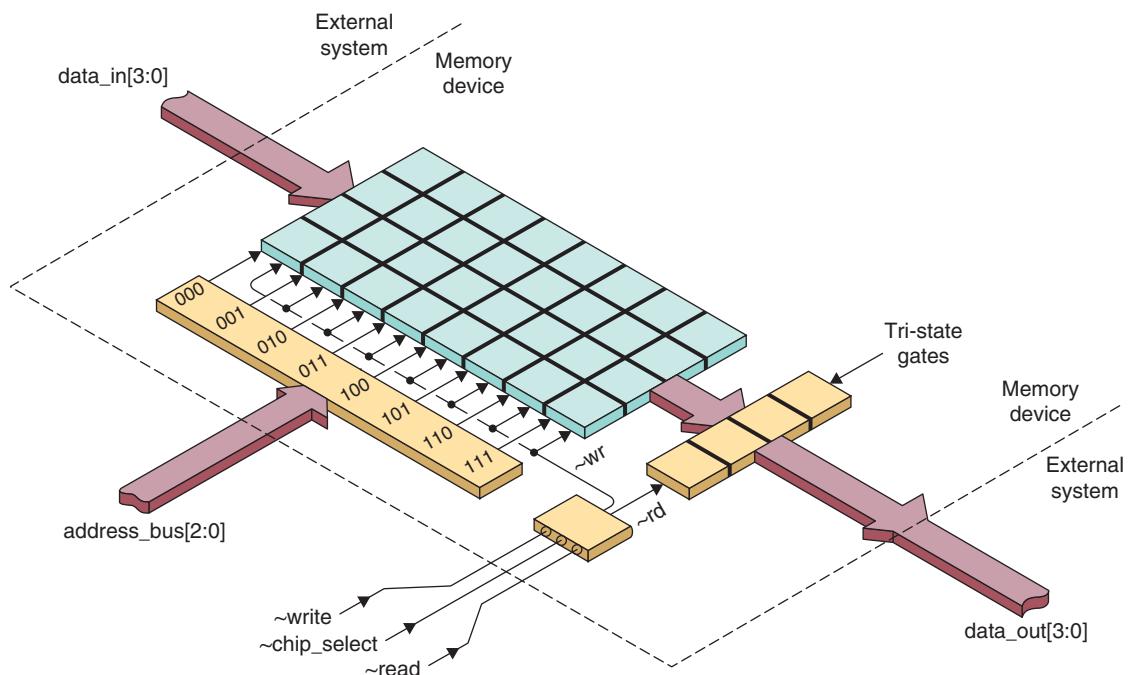


FIGURE 15.5

RAM with separate data in and data out busses.

the value on $\sim\text{wr}$. Therefore, devices of this type may be written to and read from simultaneously.¹⁶

RAM WITH SINGLE BIDIRECTIONAL BUS

In contrast to those devices with separate data in and data out buses as discussed above, the majority of RAMs use a common bus for both writing and reading. In this case, the $\sim\text{read}$ and $\sim\text{write}$ signals are usually combined into a single control input called something like $\text{read}\sim\text{write}$ (the name $\text{read}\sim\text{write}$ indicates that a logic 1 on this signal is associated with a read operation, while a logic 0 is associated with a write). When the $\sim\text{chip_select}$ signal indicates to the RAM that its attention is required, the value on $\text{read}\sim\text{write}$ is used to determine the type of operation to be performed. If $\text{read}\sim\text{write}$ carries a logic 1, the internal $\sim\text{rd}$ signal is made active and a read operation is initiated; if $\text{read}\sim\text{write}$ carries a logic 0, the internal $\sim\text{wr}$ signal is made active and a write operation is executed (Figure 15.6).

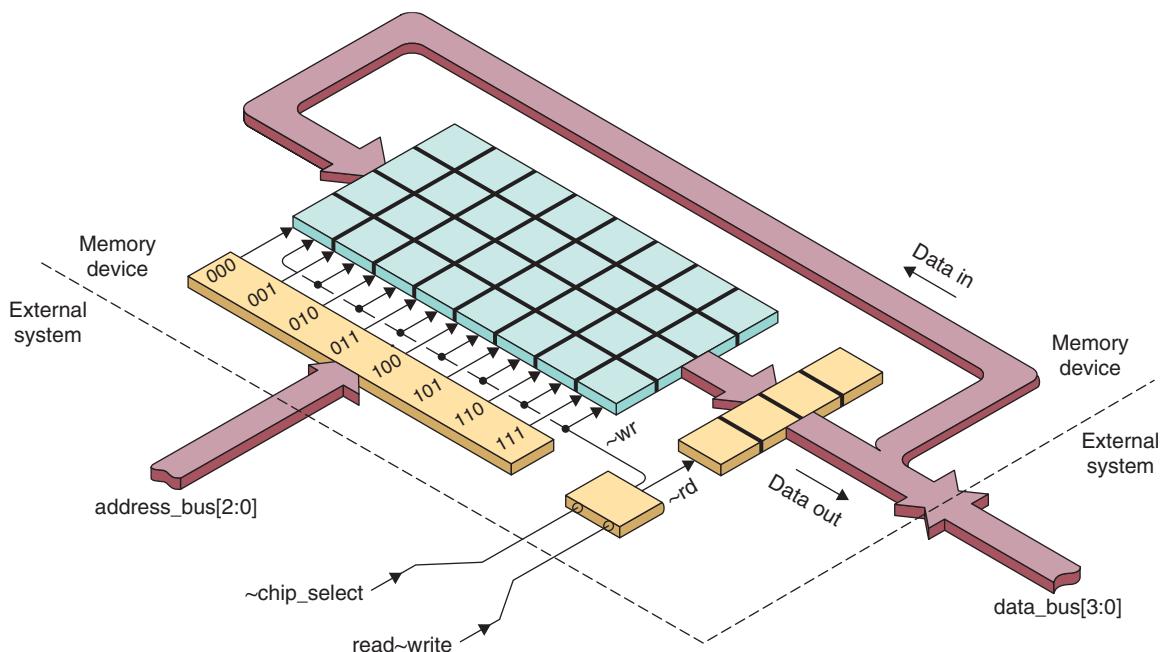


FIGURE 15.6
RAM with a single
bidirectional data bus.

¹⁶Note that we're simplifying everything here for clarity. In the case of a RAM with separate data in and data out busses, we would also typically have two address buses—one to specify the read address and the other to specify the write address. Known as a *dual-port* RAM, this allows a word of data to be read out of one location while simultaneously writing a new word of data into another location.

When $\sim\text{rd}$ is active (and therefore $\sim\text{wr}$ is inactive), the tri-state gates are enabled and the data from the word selected by the address bus is driven onto the data bus. When $\sim\text{wr}$ is active (and therefore $\sim\text{rd}$ is inactive), the tri-state gates are disabled. This allows external devices to place data onto the data bus to be written into the word selected by the address bus.

If the value on the address bus changes while $\sim\text{rd}$ is active, the data associated with the newly selected word will appear on the data bus. However, it is not permissible for the value on the address bus to change while $\sim\text{wr}$ is active because the contents of multiple locations may be corrupted.

INCREASING WIDTH AND DEPTH

Individual memory devices can be connected together to increase the width and depth of the total memory as seen by the rest of the system. For example, two 1024-word devices, each 8 bits wide, can be connected so as to appear to be a single 1024-word device with a width of 16 bits. An address bus containing 10 bits is required to select between the 1024 words (Figure 15.7).

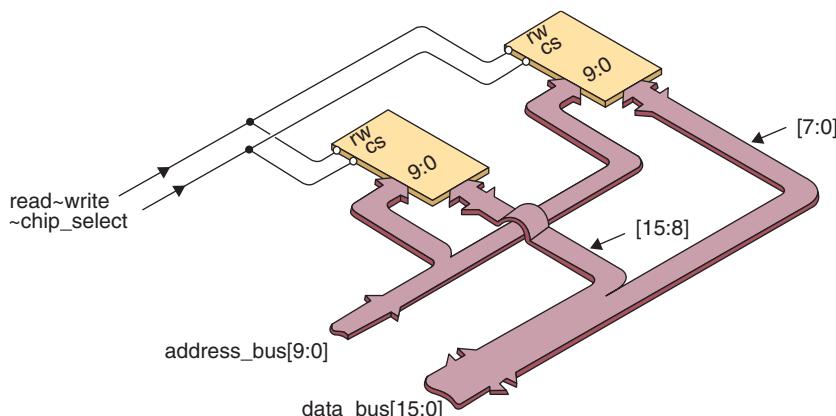
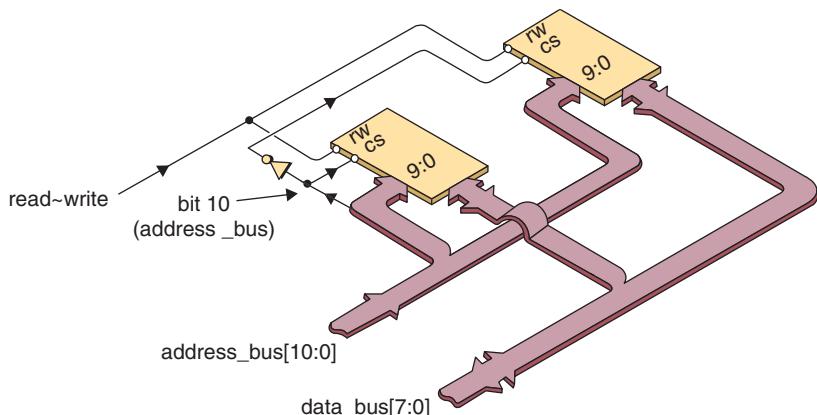


FIGURE 15.7

Connecting memory devices to increase the width.

In this case, the address bus and control signals are common to both memories, but each device handles a subset of the signals forming the data bus. Additional devices can be added to further increase the total width of the data bus as required. Alternatively, two 1024-word devices, each 8 bits wide, can be connected so as to appear to be a single device with a width of 8 bits and a depth of 2048 words. An address bus containing 11 bits is required to select between the 2048 words (Figure 15.8).

**FIGURE 15.8**

Connecting memory devices to increase the depth.

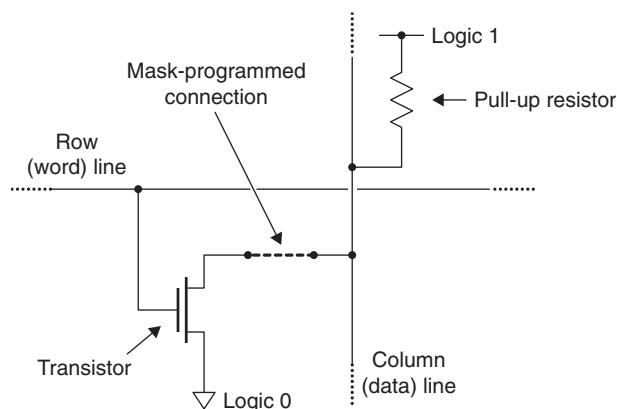
In this case, the data bus, the ten least-significant bits of the address bus, and the `read~write` control are common to both memories. However, the *Most Significant Bit* (MSB) of the address bus is decoded to generate two `~chip_select` signals. A logic 0 on the most-significant bit of the address bus selects the first device and deselects the second, while a logic 1 deselects the first device and selects the second.

Additional address bits can be used to further increase the total depth as required. If the address bus in the previous example had contained 12 bits, the two most-significant bits could be passed through a 2:4 decoder to generate four `~chip select` signals. These signals could be used to control four 1024-word devices, making them appear to be a single memory with a width of 8 bits and a depth of 4096 words.

MASK-PROGRAMMED ROMS

ROM devices are said to be *mask-programmed* (or sometimes *mask-programmable*) because the data they contain is hard-coded into them during their construction (using photo-masks as was discussed in the previous chapter).

For example, consider a transistor-based ROM cell that can hold a single bit of data (Figure 15.9). The entire ROM consists of a number of *row* (word) and *column* (data) lines forming an array. Each column has a single pull-up resistor attempting to hold that column to a weak logic 1 value, and every row-column intersection has an associated transistor and—potentially—a mask-programmed connection.

**FIGURE 15.9**

A transistor-based mask-programmed ROM cell.

The majority of the ROM can be preconstructed and the same underlying architecture can be used for multiple customers. When it comes to customizing the device for use by a particular customer, a single photo-mask is used to define which cells are to include mask-programmed connections and which cells are to be constructed without such connections.

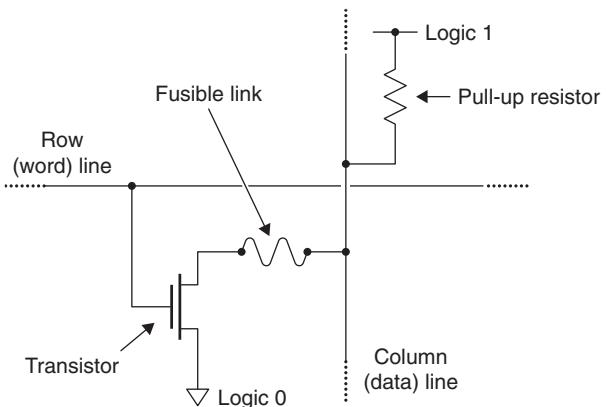
Now consider what happens when a row line is placed in its active state, thereby attempting to activate all of the transistors connected to that row. In the case of a cell that includes a mask-programmed connection, activating that cell's transistor will connect the column line through the transistor to logic 0, so the value appearing on that column as seen from the outside world will be a logic 0. By comparison, in the case of a cell that doesn't have a mask-programmed connection, that cell's transistor will have no effect, so the pull-up resistor associated with that column will hold the column line at logic 1, which is the value that will be presented to the outside world.

PROMS

The problem with mask-programmed devices is that creating them is a very expensive pastime unless you intend to produce them in extremely large quantities. Furthermore, such components are of little use in a development environment in which you often need to modify their contents.

For this reason, the first *Programmable Read-Only Memory* (PROM)¹⁷ devices were developed at Harris Semiconductor in 1970. These devices were created

¹⁷In conversation, PROM is pronounced just like the high school dance of the same name.

**FIGURE 15.10**

A transistor-and-fusible-link-based PROM cell.

using a nichrome-based fusible-link¹⁸ technology. As a generic example, consider a somewhat simplified representation of a transistor-and-fusible-link-based PROM cell (Figure 15.10).

In its unprogrammed state as provided by the manufacturer, all of the fusible links in the device are present. In this case, placing a row line in its active state will turn on all of the transistors connected to that row, thereby causing all of the column lines to be pulled-down to logic 0 via their respective transistors. However, design engineers can selectively remove (*blow*) undesired fuses by applying pulses of relatively high voltage and current to the device's inputs. Wherever a fuse is removed ("blown"), that cell will appear to contain a logic 1.

PROMs were initially intended to be used as memories to store computer programs and constant data values (hence, the ROM portion of their appellation). However, design engineers also found them useful for implementing simple logical functions such as lookup tables and state machines. The fact that PROMs were relatively cheap meant that these functions could be quickly modified to fix bugs or test new implementations by simply burning a new device and plugging it into the system.

PROMs are said to be *One-Time Programmable* (OTP), because once you've programmed one and blown its fuses there's no going back. PROMs are slightly slower than their ROM equivalents, but are significantly cheaper for small- to medium-sized production runs.

¹⁸See also the discussions on the Fusible Link and Antifuse technologies in *Chapter 16: Programmable ICs*.

EPROMS

As was previously noted, one consideration associated with devices based on fusible-link technologies is that they can only be programmed a single time—once you've blown a fuse it's too late to change your mind. (In rare cases it's possible to incrementally modify devices by blowing additional fuses, but the fates have to be smiling in your direction.) For this reason, people started to think that it would be nice if there were some way to create devices that could be programmed, erased, and reprogrammed with new data.

One alternative is a technology known as *Erasable Programmable Read-Only Memory* (EPROM); the first such device, the 1702, was introduced by Intel in 1971. An EPROM transistor has the same basic structure as a standard MOSFET transistor, but with the addition of a second polysilicon *floating gate* isolated by layers of oxide (Figure 15.11).

In its unprogrammed state, the floating gate is uncharged and doesn't affect the normal operation of the control gate. In order to program the transistor, a relatively high voltage¹⁹ is applied between the control gate and drain terminals. This causes the transistor to be turned hard on, and energetic electrons force their way through the oxide into the floating gate in a process known as *hot electron injection* (or *high energy injection*).

When the programming signal is removed, a negative charge remains on the floating gate. This charge is very stable and will not dissipate for more than

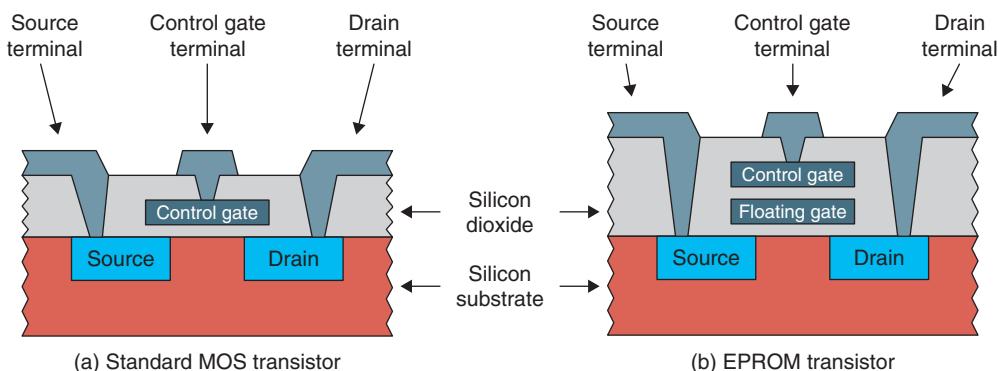
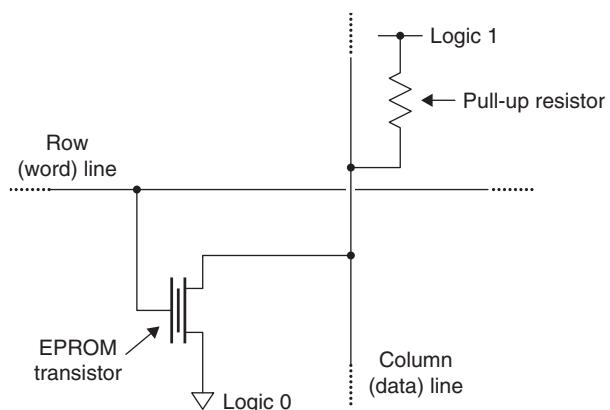


FIGURE 15.11

Standard MOSFET versus EPROM transistors.

¹⁹Not outrageously high; just higher than the typical signal levels. There's no point in our specifying the programming voltage here, because it keeps on falling with the advent of new technology nodes.

**FIGURE 15.12**

An EPROM transistor-based memory cell.

a decade under normal operating conditions. The stored charge on the floating gate inhibits the normal operation of the control gate, and, thus, distinguishes those cells that have been programmed from those which have not. This means we can use such a transistor to form a memory cell (Figure 15.12).

Observe that this cell no longer requires a fusible link or mask-programmed connection. In its unprogrammed state as provided by the manufacturer, all of the floating gates in the

EPROM transistors are uncharged. In this case, placing a row line in its active state will turn on all of the transistors connected to that row, thereby causing all of the column lines to be pulled-down to logic 0 via their respective transistors. In order to program the device, engineers can use the inputs to the device to charge the floating gates associated with selected transistors, thereby disabling those transistors. In these cases, the cells will appear to contain logic 1 values.

As they are an order of magnitude smaller than fusible links, EPROM cells are efficient in terms of silicon real estate. Their main claim to fame, however, is that they can be erased and reprogrammed. An EPROM cell is erased by discharging the electrons on that cell's floating gate. The energy required to discharge the electrons is provided by a source of *Ultraviolet* (UV) radiation. An EPROM device is delivered in a ceramic or plastic package with a small quartz window in the top, which is usually covered with a piece of opaque sticky tape. In order for the device to be erased, it is first removed from its host circuit board, its quartz window is uncovered, and it is placed in an enclosed container with an intense ultraviolet source.

The main problems with EPROM devices are their expensive packages with quartz windows and the time it takes to erase them, which is in the order of 20 minutes. A foreseeable problem with future devices is paradoxically related to improvements in the process technologies that allow transistors to be made increasingly smaller. As the structures on the device become smaller and the density (number of transistors and interconnects) increases, a larger percentage of the surface of the die is covered by metal. This makes it difficult for the EPROM cells to absorb the ultraviolet light and increases the required exposure time.

When EPROMs first appeared on the scene, engineers found them ideal for prototyping and other applications that required regular changes to the stored

data. It was common practice for EPROMs to be used during the development of a design and then subsequently replaced by equivalent PROM devices in cheaper plastic packages after the design had stabilized.

EEPROMS/E²PROMS

A further technology, called *Electrically-Erasable Read-Only Memory* (EEPROM or E²PROM) was developed towards the end of the 1970s. An E²PROM cell is approximately 2.5 times larger than an equivalent EPROM cell because it comprises two transistors and the space between them (Figure 15.13).

The E²PROM transistor is similar to that of an EPROM transistor in that it contains a floating gate, but the insulating oxide layers surrounding this gate are very much thinner. The second transistor can be used to erase the cell electrically.

Thus, E²PROM devices are electrically programmable by the designer and are non-volatile, but can be electrically erased and reprogrammed should the designer so desire. Unlike an EPROM device that must be erased and reprogrammed in its entirety, an E²PROM can be erased and reprogrammed on a word-by-word basis. Additionally, by means of additional circuitry, an E²PROM can be erased and reprogrammed while remaining resident on the circuit board, in which case it may be referred to as *In-System Programmable* (ISP).

FLASH

Yet another technology called *FLASH* is generally regarded as an evolutionary step that combines the best features from EPROM and E²PROM. The name FLASH was originally coined to reflect its fast reprogramming time compared to EPROM. FLASH has been under development since the end of the 1970s, and was officially described in 1985, but the technology did not initially receive a great deal of interest. Toward the end of the 1980s, however, the demand for portable computer and communication systems increased dramatically, and FLASH began to attract the attention of designers.

All variants of FLASH are electrically erasable like E²PROMs. Some devices are based on a single transistor cell, which provides a greater capacity than an E²PROM, but which must be erased and reprogrammed on a device-wide basis similar to an EPROM. Other devices are based on a dual transistor cell and can be erased and reprogrammed on a word-by-word or block-by-block basis.

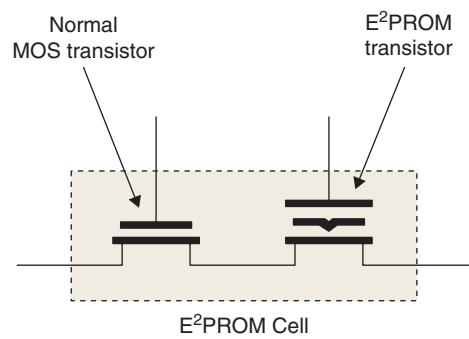


FIGURE 15.13
An E²PROM cell.

FLASH is considered to be of particular value when the designer requires the ability to reprogram a system in the field or via a communications link while the devices remain resident on the circuit board.

SRAMS AND DRAMS

In the case of *Dynamic RAMs* (DRAMs), each cell is formed from a transistor-capacitor pair. The term *dynamic* is applied because a capacitor loses its charge over time and each cell must be periodically recharged to retain its data. This operation, known as *refreshing*, requires the contents of each cell to be read out and then rewritten. Some types of DRAM require external circuitry to supervise the refresh process, in which case a special independent controller device is employed to manage a group of DRAMs. In other cases, a DRAM may contain its own internal self-refresh circuitry.

In the case of *Static RAMs* (SRAMs), each cell is formed from four or six transistors configured as a latch or a flip-flop.²⁰ The term *static* is applied because, once a value has been loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.

SRAMs are much faster than DRAMs, but each SRAM cell requires significantly more silicon real estate and consumes much more power than a corresponding DRAM cell. The first SRAM (256-bit) and DRAM (1024-bit) devices were both created in 1970. For the next few decades, both types of memory quadrupled their capacities approximately every three years, but by the beginning of the 21st century this had slowed to a doubling every two to three years.

These days, DRAM (in its SDRAM incarnation as discussed in the next topic) is used to satisfy the bulk of a system's RAM requirements, while SRAM is used where speed is of the essence; for example, the cache memory inside a CPU.

SDRAMS

Until the latter half of the 1990s, DRAM-based computer memories were *asynchronous*, which means they weren't synchronized to the system clock. Every new generation of computers used a new trick to boost speed, and even the engineers started to become confused by the plethora of names, such as *Fast Page Mode* (FPM), *Extended Data Out* (EDO), and *Burst EDO* (BEDO). In reality, these were all based on core

²⁰See also Chapter 17: Application-Specific Integrated Circuits (ASICs) for discussions on 1T versus 6T SRAM.

DRAM concepts; the differences were largely in how one wired the chips together on the circuit board (OK, sometimes there was a bit of tweaking inside the chips also).

Over time, the industry migrated to *Synchronous DRAM* (SDRAM), which refers to a memory subsystem that is synchronized to the system clock, thereby making everyone's lives much easier. Once again, SDRAM is based on underlying DRAM concepts—it's all in the way you tweak the chips and connect them together.

There are lots of nitty-picky details that we won't go into here; all we need to know for the purpose of these discussions is that SDRAM is based on DRAM arranged in multiple *banks* that are interleaved together and accessed via a multiplexer. For example, consider a scenario with four banks as illustrated in Figure 15.14.

Now, this is obviously an extremely oversimplified representation; for example, we haven't shown the clock signal coming from the main system, or any registers that would be used to latch the outputs, or any mechanism by which we could write data into the memory subsystem; on the other hand, this illustration will serve the purposes of these discussions.

Now, suppose that we previously loaded a big, multi-word "chunk" of data into our SDRAM subsystem. As we know, each DRAM bank is relatively slow, compared to an SRAM equivalent. Thus, if we had stored our block of data in contiguous locations in the same bank of DRAM, it would take us a relatively long time to read it out again (which is, of course, why we don't do it that way).

Instead, the first word of data would be stored in bank 0, the second word in bank 1, the third word in bank 2, and the fourth word in bank 3; then we continue with the fifth word in bank 0, the sixth word in bank 1, the seventh word in bank 2 ... and so forth. Now, when we wish to retrieve our data, we can flick from bank to bank. In this case, we would use the multiplexer to select the output from bank 0 (which contains the first word of data), latch this data, and present it to the main system. We would then use the multiplexer to select the output from bank 1 (which contains the second word of data), latch *this* data, and present it to the main system. And so on for bank 2 (containing the third word) and bank 3 (containing the fourth word).

The point is that as soon as the multiplexer has turned its attention away from bank 0, the subsystem can direct it to start retrieving the fifth word of data;

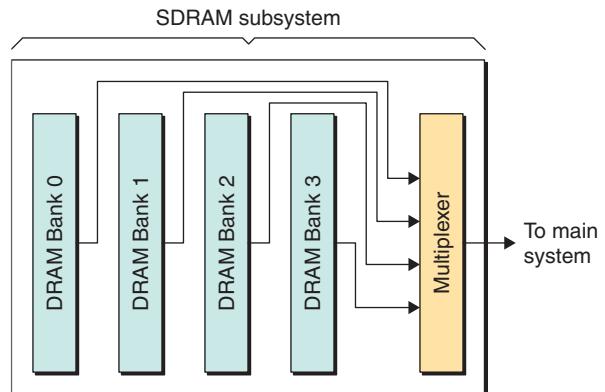


FIGURE 15.14

An example SDRAM subsystem with four banks of DRAM.

similarly, as soon as the multiplexer moves on from bank 1, the subsystem can direct it to start retrieving the sixth word of data. Thus, when the multiplexer eventually cycles back to bank 0, it already has the fifth word of data ready and waiting. By this means, the interleaved banks of DRAM can be used to store and retrieve data much faster than would be possible using a single bank.

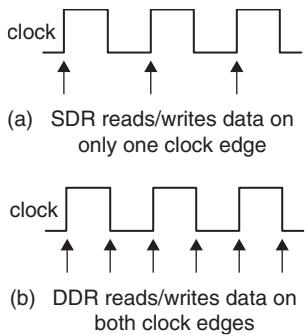


FIGURE 15.15

SDR versus
DDR SDRAM
implementations.

DDR, DDR2, DDR3, QDR, RAMBUS, ETC.

The original *Single Data Rate* (SDR) SDRAM specification was based on using only one of the clock edges (say the rising edge) to read/write data out-of/into the memory, as illustrated in Figure 15.15(a).

By comparison, *Double Data Rate* (DDR) SDRAM uses a technique called *double pumping* to double the bandwidth (the amount of data that can be transferred in a given amount of time) without increasing the clock frequency. It achieves this by transferring data on both the rising and falling edges of the clock signal, as illustrated in Figure 15.15(b) (this sounds simple if you say it quickly, but actually making this work is trickier than it may at first appear).

DDR2 is essentially a later generation of DDR that provides higher performance than its ancestor; similarly, DDR3 uses cunning tricks to out-perform DDR2. [This is probably as good a time as any to mention the concept of *Quad Data Rate* (QDR) memory, which has separate data in and data out busses, both of which can be used on both edges of the clock.]

Today, virtually all SDRAM implementations are manufactured in compliance with standards established by JEDEC, which is an electronics industry association that adopts open standards to facilitate interoperability of electronic components. By comparison, *Direct Rambus DRAM* or DRDRAM (sometimes just called *Rambus DRAM* or RDRAM) is a proprietary type of SDRAM designed by the Rambus Corporation. *Extreme Data Rate* (XDR) DRAM was the successor to RDRAM, while XDR2 DRAM is the successor to XDR DRAM.

SIMMS, DIMMS, AND RIMMS

Whichever flavor of DRAM you are using, a single device can only contain a limited amount of data, so a number of DRAMs are gathered together onto a small circuit board called a *memory module*. Each memory module has a line of gold-plated pads on both sides of one edge of the board. These pads plug into a corresponding connector on the main computer board.

A *Single In-Line Memory Module* (SIMM) has the same electrical signal on corresponding pads on the front and back of the board (that is, the pads on opposite sides of the board are “tied together”). By comparison, in the case of a *Dual In-Line Memory Module* (DIMM), the pads on opposite sides of the board are electrically isolated from each other and form two separate contacts.

Last but not least, we have the RIMM, which really doesn’t stand for anything per se, but which is the trademarked name for a Rambus Memory Module. (RIMMs are similar in concept to DIMMs, but have a different pin count and configuration.)

ECC MEMORY

Computer systems are very complicated and there’s always the chance that an error will occur when reading or writing to the memory (a stray pulse of *noise* may flip a logic 0 into a logic 1, or vice versa, while your back is turned). Thus, serious computers use *Error-Correcting Code* (ECC) memory, which includes extra bits and special circuitry that tests the accuracy of data as it passes in and out of memory and corrects any (simple) errors.

MRAMS

A technology that continues to attract a great deal of interest for the future is *Magnetoresistive Random Access Memory* (MRAM), which may be able to store more data, read and write data faster, and use less power than any of the current memory technologies. In fact, the seeds of MRAM were laid as far back as 1974, when IBM developed a component called a *Magnetic Tunnel Junction* (MTJ), which comprises a sandwich of two ferromagnetic layers separated by a thin insulating layer. A memory cell is created by the intersection of two wires (say a *row* line and a *column* line) with an MTJ sandwiched between them. MRAMs have the potential to combine the high speed of SRAM, the storage capacity of DRAM, and the nonvolatility of FLASH, while consuming very little power.

Thus far, ongoing improvements in conventional memory technologies have kept MRAM in a niche role, but lots of folks believe that that MRAM has so many advantages that it will eventually take center stage.

NVRAMS, FRAMS, PRAMS, RRAMS, CBRAMS, SONOS, ETC.

Last but not least, we have a class of devices known as *Nonvolatile RAMs* (nvRAMs). Historically, these little rascals were formed from an SRAM die

mounted in a package with a very small battery, or as a mixture of SRAM and E²PROM (later SRAM and FLASH) cells fabricated on the same die.

Over recent years, a lot of experimental nvRAM technologies have popped-up to shout a cheery “Hello!” These include (but are not limited to) *Ferroelectric RAM* (FeRAM or FRAM), *Phase-Change Memory* (also known as PCM, PRAM, or PCRAM), *Resistive Random Access Memory* (RRAM), *Conductive-Bridging RAM* (CBRAM), and SONOS (short for *Silicon-Oxide-Nitride-Oxide-Silicon*, which describes the layers of materials forming the memory cell).

CHAPTER 16

Programmable ICs

A SIMPLE PROGRAMMABLE FUNCTION

Before we move on to consider some actual devices, let's first set the scene with some fundamental concepts. As a basis for these discussions, let's start by considering a very simple programmable function with two inputs called a and b , and a single output called y (Figure 16.1).

The inverting (NOT) gates associated with the inputs mean that each input is available in both its *true* (unmodified) and *complemented* (inverted) form. Observe the locations of the "potential links." In the absence of any of these links, all of the inputs to the AND gate are connected via pull-up resistors to a logic 1 value. In turn, this means that the output y will always be driving a logic 1, which makes this circuit a very boring one in its current state. In order to make our function more interesting, we need some mechanism that allows us to establish one or more of the potential links ...

213

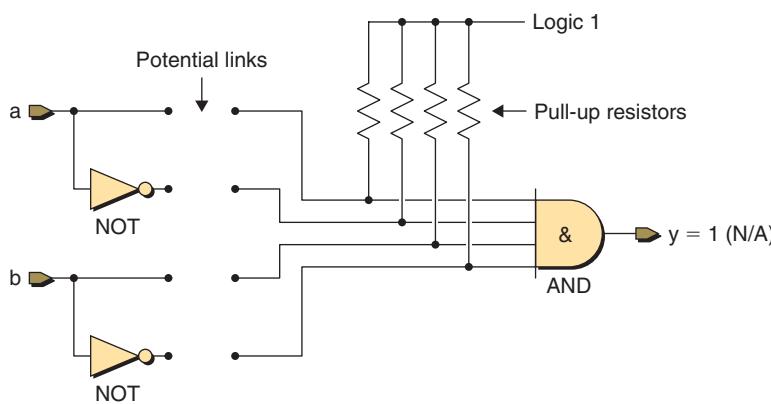
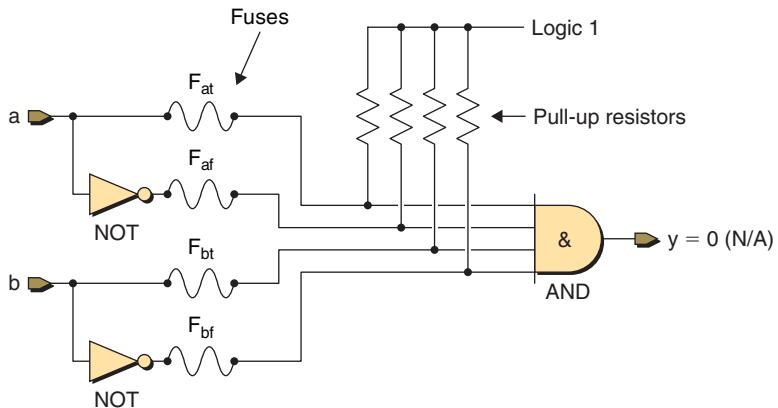


FIGURE 16.1

A simple programmable function.

**FIGURE 16.2**

Augmenting our example function with unprogrammed fusible links.

FUSIBLE-LINK TECHNOLOGIES

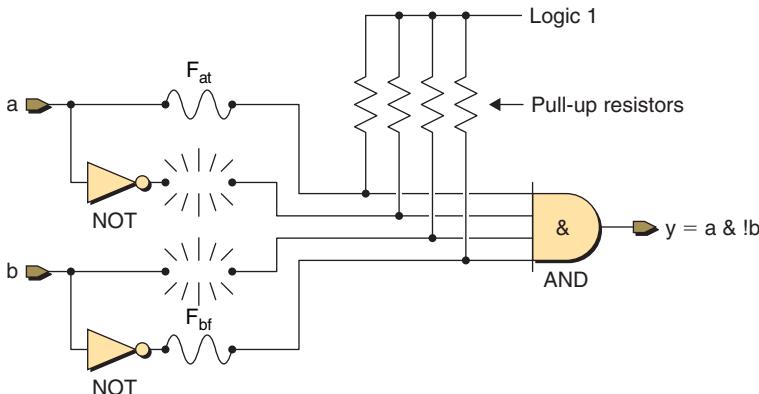
One of the first techniques that allowed users to program their own devices was—and still is—known as *fusible-link* technology. In this case, the device is manufactured with all of the links in place, where each link is referred to as a *fuse* (Figure 16.2).

These fuses are similar in concept to the fuses you find in household products like a television. If anything untoward occurs, such that the television starts consuming too much power, its fuse will burn out. This results in an open circuit (a break in the wire), which protects the rest of the unit from harm. Of course, the fuses in a silicon chip are formed using the same processes that are employed to create the transistors and wires on the chip, so they are microscopically small.

When an engineer purchases a programmable device based on fusible links, all of the fuses are initially intact. This means that, in its unprogrammed state, the output from our example function will always be logic 0. This is because any 0 presented to the input of an AND gate will cause its output to be 0, so if input a is 0, the output from the AND will be 0; alternatively, if input a is 1, then the output from its NOT gate (which we shall call $\text{!}a$, meaning “not a ”)¹ will be 0, and once again, the output from the AND will be 0. A similar situation occurs in the case of input b .

The point is that design engineers can selectively remove undesired fuses by applying pulses of relatively high voltage and current to the device’s inputs. For example, consider what happens if we remove fuses F_{at} and F_{bt} (Figure 16.3).

¹Actually, this isn’t a particularly great naming convention because (for reasons that are more fully discussed in Appendix A: Assertion-Level Logic) it could lead to confusion in everyday usage; however, it will serve our purpose here.

**FIGURE 16.3**

Programming (blowing) some of the fusible links.

Removing these fuses disconnects the complementary version of input a and the true version of input b from the AND gate (the pull-up resistors associated with these signals cause their associated inputs to the AND to be presented with logic 1 values). This leaves the device to perform its new function, which is $y = a \& !b$. (The “ $\&$ ” character in this equation is used to represent the AND, while the “ $!$ ” character is used to represent the NOT.) Previously, we used a horizontal bar over a signal name to indicate a negation, this is just another representation. This process of removing fuses is typically referred to as “programming the device,” but it may also be referred to as “blowing the fuses” or “burning the device.”

Devices based on fusible-link technologies are said to be *One-Time Programmable* (OTP), because once a fuse has been blown it cannot be replaced and there's no going back.

ANTIFUSE TECHNOLOGIES

As a diametric alternative to fusible-link technologies, we have their *antifuse* counterparts, in which each configurable path has an associated link called an *antifuse*. In its unprogrammed state, an antifuse has such a high resistance that it may be considered to be an open circuit (a break in the wire) as illustrated in Figure 16.4.

This is the way the device appears when it is first purchased. However, antifuses can be selectively “grown” (programmed) by applying pulses of relatively high voltage and current to the device’s inputs. For example, if we add the antifuses associated with the complementary version of input a and the true version of input b , our device will now perform the function $y = !a \& b$ (Figure 16.5).

An antifuse commences life as a microscopic column of amorphous (noncrystalline) silicon linking two metal tracks. In its unprogrammed state, the amorphous

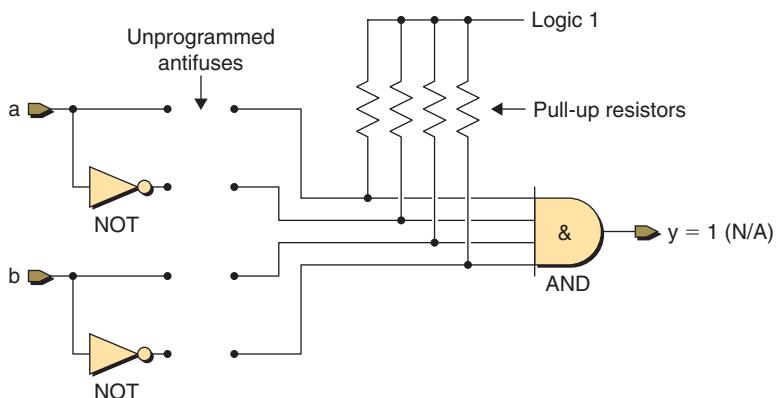


FIGURE 16.4
Unprogrammed antifuses.

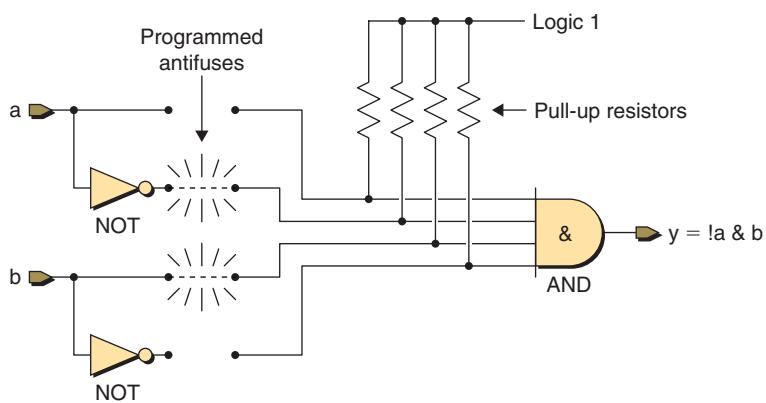


FIGURE 16.5
Programmed antifuse links.

silicon acts as an insulator with a very high resistance, in excess of one billion ohms [Figure 16.6(a)].

The act of programming this particular element effectively “grows” a link—also known as a *via*—by converting the insulating amorphous silicon into conducting polysilicon [Figure 16.6(b)].

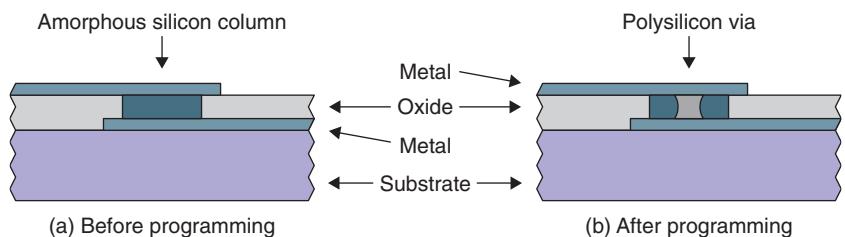


FIGURE 16.6
Growing an antifuse.

Not surprisingly, devices based on antifuse technologies are *One-Time Programmable* (OTP), because once an antifuse has been grown it cannot be removed and there's no changing your mind.

EPROM, E²PROM, FLASH, AND SRAM TECHNOLOGIES

In *Chapter 15: Memory ICs*, we introduced a variety of memory technologies, including EPROM, E²PROM, and FLASH. Although we don't want to go into the nitty-gritty details here, the point is that we could take our example function illustrated in Figure 16.1 and use EPROM transistors to implement its programmable links. Alternatively, we could create its programmable links using E²PROM cells or FLASH technology.

As yet one final alternative, we could realize programmable links based on SRAM cells, as illustrated in Figure 16.7. In this case, the entire cell comprises a multi-transistor SRAM storage element whose output drives an additional control transistor. Depending on the contents of the storage element (logic 0 or logic 1), the control transistor will either be OFF (disabled) or ON (enabled).

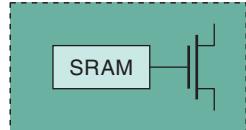


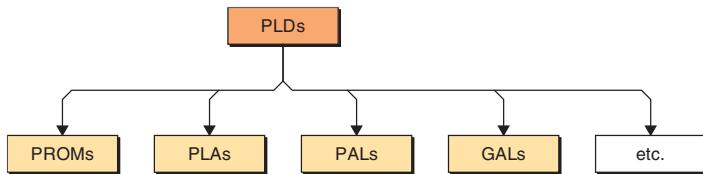
FIGURE 16.7
An SRAM-based programmable cell.

One disadvantage of having a programmable device based on SRAM cells is that each cell consumes a significant amount of silicon real estate, because these cells are formed from four or six transistors configured as a latch or a flip-flop. Another disadvantage is that the device's configuration data (programmed state) will be lost when power is removed from the system. In turn, this means that these devices always have to be reprogrammed when the system is powered-on. However, such devices have the corresponding advantage that they can be reprogrammed quickly and repeatedly as required.

THE FIRST PROGRAMMABLE LOGIC DEVICES (PLDS)

The first programmable integrated circuits were generically referred to as *Programmable Logic Devices* (PLDs). The original components—which started arriving on the scene in 1970 in the form of PROMs—were rather simple, but everyone was too polite to mention it.

Just to increase our fun and frivolity, engineers love to use the same acronym to mean different things or different acronyms to mean the same thing (listening to a gaggle of engineers regaling each other in conversation can make even the strongest mind start to “throw a wobbly”). In the case of the early PLDs, for example, there is a multiplicity of underlying architectures, many of which

**FIGURE 16.8**

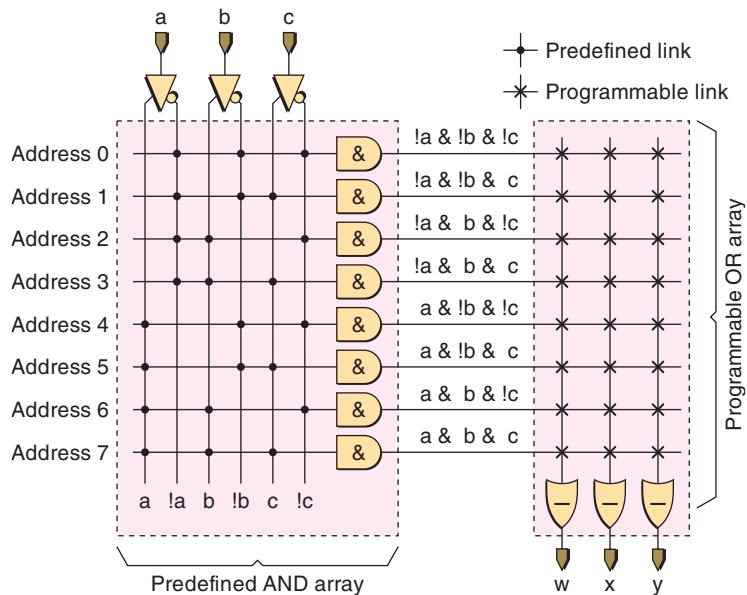
A snapshot of the early days of PLDs technology.

have acronyms formed from different combinations of the same three or four letters. Let's take a snapshot of the early days as illustrated by Figure 16.8, and then we'll ponder these devices in a little more detail.

Of course there are also EPLD, E²PLD, and FLASH versions of these devices (EPROMs and E²PROMs, for example), but these variants have been omitted from Figure 16.8 for purposes of simplicity.

PROMS

The first of the simple PLDs were *Programmable Read-Only Memories* (PROMs), which jumped into the limelight in 1970. One way to visualize the manner in which these devices perform their magic is to consider them as consisting of a fixed array of AND functions driving a programmable array of OR functions. For the purposes of these discussions, let's consider a hypothetical 3-input, 3-output PROM as illustrated in Figure 16.9.

**FIGURE 16.9**

Unprogrammed PROM (predefined AND array, programmable OR array).

Depending on the whim of the manufacturer, the programmable links in the OR array might be implemented as fusible links, as EPROM transistors, E²PROM cells, or FLASH.² It is important to note that this illustration is intended only to provide a high-level view of the way in which our example device works—it does *not* represent an actual circuit diagram. In reality, each AND function in the AND array has three inputs provided by the appropriate true or complemented versions of the a, b, and c device inputs. Similarly, each OR function in the OR array has eight inputs provided by the outputs from the AND array.

As we discussed in *Chapter 15: Memory ICs*, PROMs were originally intended for use as computer memories in which to store program instructions and constant data values. However, design engineers also used them to implement simple logical functions such as lookup tables and state machines. In fact, a PROM can be used to implement any block of *combinational*³ logic so long as it doesn't have too many inputs or outputs. The simple 3-input, 3-output PROM shown in Figure 16.9, for example, can be used to implement any combinatorial function with up to three inputs and three outputs. In order to illustrate the way in which this works, consider the small block of logic shown in Figure 16.10 (this circuit has no significance beyond the purposes of this example).

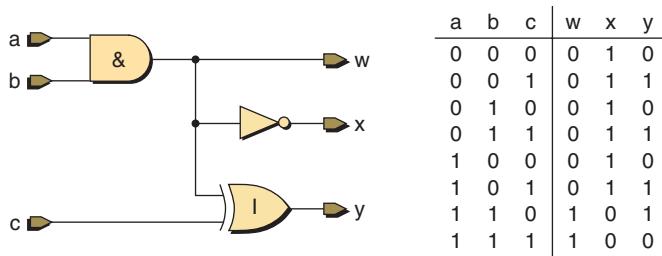


FIGURE 16.10

A small block of combinational logic.

We could replace this block of logic with our 3-input, 3-output PROM. All that would be required would be to program the appropriate links in the OR array (Figure 16.11).

²Simple PLDs didn't (and don't) use antifuse or SRAM programming technologies.

³Some folks prefer the term *combinatorial*.

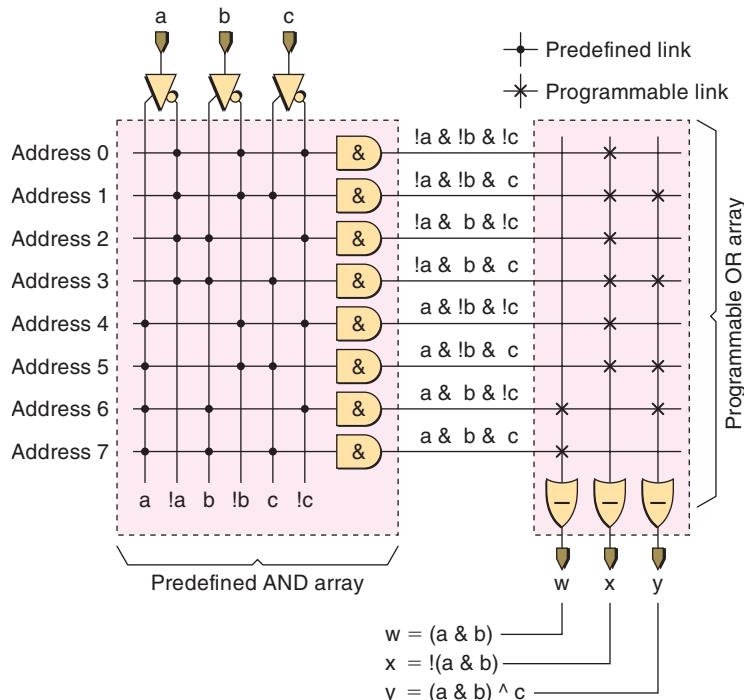


FIGURE 16.11
Programmed PROM.

With regard to the equations shown in this figure, “ $\&$ ” represents AND, “ $|$ ” represents OR, “ \wedge ” represents XOR, and “ $!$ ” represents NOT.^{4,5,6,7} This syntax (or numerous variations thereof) was very common in the early days of PLDs, because it allowed logical equations to be easily and concisely represented in text files using standard computer keyboard characters.

The example shown above is, of course, very simple. Real PROMs can have significantly more inputs and outputs, and can therefore be used to implement larger blocks of combinational logic. From the mid-1960s until the mid-1980s (or later), combinational logic was commonly implemented by means of jelly bean ICs such as the Texas Instruments 74xx series devices. The fact that quite a large number of these jelly bean chips could be replaced with a single PROM resulted in circuit

⁴The “ $\&$ ” (ampersand) character is commonly spoken of as an “amp” or “amper.”

⁵The “ $|$ ” (vertical line) character is commonly spoken of as a “bar,” “or,” or “pipe.”

⁶The “ \wedge ” (circumflex) character is commonly spoken of as a “hat,” “control,” “up-arrow,” or “caret.”

⁷More rarely it may be referred to as a “chevron,” “power of” (as in “to the power of”), or “shark-fin.”

⁸The “ $!$ ” (exclamation mark) character is commonly spoken of as a “bang,” “ping,” or “shriek.”

boards that were smaller, lighter, cheaper, and less prone to error (each solder joint on a circuit board provides a potential failure mechanism). Furthermore, if any logic errors were subsequently discovered in this portion of the design (if the design engineer had inadvertently used an AND function instead of a NAND, for example), then these slip-ups could easily be fixed by blowing a new PROM (or erasing and reprogramming an EPROM or E²PROM). This was much more preferable to the ways in which errors had to be addressed on boards based on jelly bean ICs, which included adding new devices to the board, cutting existing tracks with a scalpel, and adding wires by hand to connect the new devices into the rest of the circuit.

As we discussed in *Chapter 9: Boolean Algebra*, in logical terms, the AND ("&") operator is known as a *logical multiplication* or *product*, while the OR ("|") is known as a *logical addition* or *sum*. Furthermore, when we have a logical equation in the form ...

$$y = (a \& !b \& c) | (!a \& b \& c) | (a \&!b \&c) | (a \&!b \& c)$$

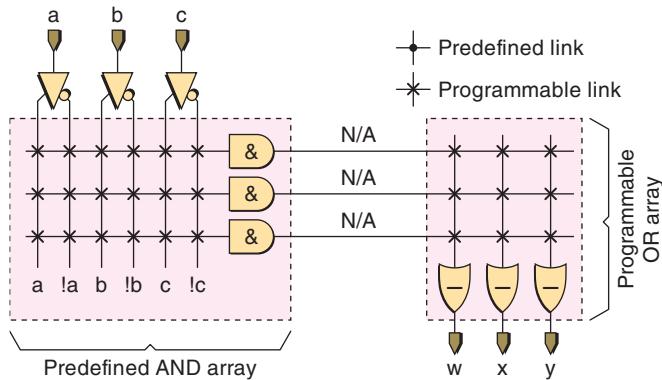
... then the term *literal* refers to any true or inverted variable (a, !a, b, !b, etc.), and a group of literals linked by "&" operators is referred to as a *product term*. Thus, the product term (a & !b & c) contains three literals (a, !b, and c) and the above equation is said to be in *sum-of-products* form.

The point is that, when PROMs are employed to implement combinational logic as illustrated in Figures 16.10 and 16.11, they are useful for equations requiring a large number of product terms, but they can only support relatively few inputs because every input combination is always decoded and used. This led engineers to start considering alternative architectures ...

PLAs

In order to address the limitations imposed by the PROM architecture, the next step up the PLD evolutionary ladder was that of *Programmable Logic Arrays* (PLAs), which first became available circa 1975. These were the most user-configurable of the simple PLDs, because both the AND and OR arrays were programmable. First, consider a simple 3-input, 3-output PLA in its unprogrammed state (Figure 16.12).

Unlike a PROM, the number of AND functions in the AND array is independent of the number of inputs to the device. Additional ANDs can be formed by simply introducing more rows into the array. Similarly, the number of OR functions in the OR array is independent of both the number of inputs to the device *and* the number of AND functions in the AND array. Additional ORs can be formed by simply introducing more columns into the array.

**FIGURE 16.12**

Unprogrammed PLA (programmable AND and OR arrays).

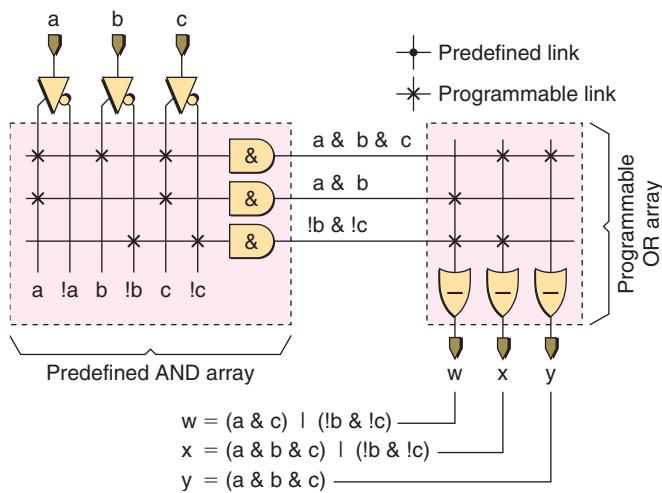
Now, assume that we wish our example PLA to implement the three equations shown below. We can achieve this by programming the appropriate links as illustrated in Figure 16.13.

$$w = (a \& c) | (!b \& !c)$$

$$x = (a \& b \& c) | (!b \& !c)$$

$$y = (a \& b \& c)$$

As fate would have it, PLAs never achieved any significant level of market presence, but several vendors experimented with different flavors of these devices

**FIGURE 16.13**

Programmed PLA.

for a while. For example, PLAs were not obliged to have AND arrays feeding OR arrays, and some alternative architectures such as AND arrays feeding NOR arrays were occasionally seen “strutting their stuff.” However, while it would be theoretically possible to field architectures such as OR-AND, NAND-OR, and NAND-NOR, these variations were relatively rare or nonexistent. One reason the designers of these devices tended to stick to AND-OR⁸ (and AND-NOR) architectures was that the *sum-of-products* representations most often used to specify logical equations could be directly mapped onto these structures. Other equation formats (like *product-of-sums*) could be accommodated using standard algebraic operations and transformations (these tasks were typically performed by software programs that could perform these techniques with their metaphorical hands tied behind their backs).

PLAs were touted as being particularly useful for large designs, whose logical equations featured a lot of common product terms that could be used by multiple outputs [for example, the product term ($\neg b \wedge c$) is used by both the w and x outputs in Figure 16.13]. This feature is referred to as *product-term sharing*.

On the downside, signals take a relatively long time to pass through programmable links as opposed to their predefined counterparts. Thus, the fact that both their AND and OR arrays were programmable meant that PLAs were significantly slower than PROMs.

PALS AND GALS

In order to address the speed problems posed by PLAs, a new class of device called *Programmable Array Logic* (PAL)⁹ was introduced in the late 1970s. Conceptually, a PAL is almost the exact opposite of a PROM, because it has a programmable AND array and a predefined OR array. As an example, consider a simple 3-input, 3-output PAL in its unprogrammed state (Figure 16.14).

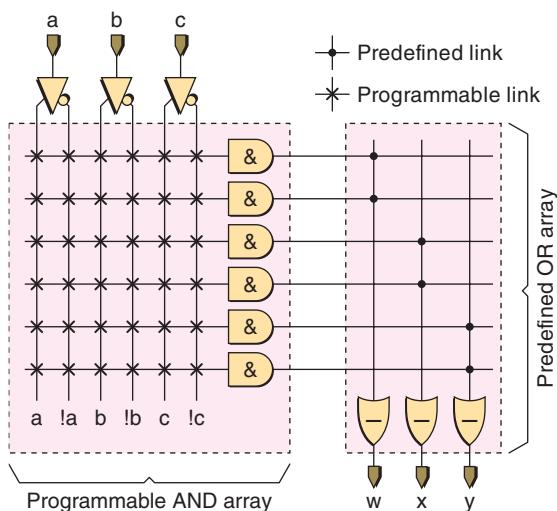


FIGURE 16.14
Unprogrammed PAL
(programmable AND array, predefined OR array).

⁸Actually, one designer I talked to told me that his team created a NOT-NOR-NOR-NOT architecture (this apparently offered a slight speed advantage) but they told their customers it was an AND-OR architecture (which is how it appeared to the outside world): “...because that was what they were expecting.” Even today, what device vendors say they build and what they actually build are not necessarily the same thing.

⁹PAL is a registered trademark of Monolithic Memories Inc.

In 1983, Lattice Semiconductor Corporation introduced a suite of *Generic Array Logic* (GAL) devices, which provided sophisticated CMOS-based electrically-erasable (E²) variations on the PAL concept.

The advantage of PALs and GALs (as compared to PLAs) is that they are faster because only one of their arrays is programmable. On the downside, PALs and GALs are more limited because they only allow a restricted number of product terms to be “OR-ed” together (but engineers are cunning fellows, and we have lots of tricks up our sleeves that—to a large extent—allow us to get around this sort of thing).

ADDITIONAL PROGRAMMABLE OPTIONS

The programmable device examples shown in the previous topics were small and rudimentary for the purposes of simplicity. In addition to being a lot larger (having more inputs, outputs, and internal signals), real devices can offer a variety of additional programmable options, such as the ability to invert the outputs and/or have tri-statable outputs.

Furthermore, some devices support registered and/or latched outputs (with associated programmable multiplexers that allow the user to specify whether to use the registered or nonregistered version of the output on a pin-by-pin basis). And some devices provide the ability to configure certain pins to act as either outputs or additional inputs. The list of options goes on ...

The problem here is that different devices may provide different subsets of the various options, which makes selecting the optimum device for a particular application something of a challenge. Engineers typically work around this by (a) restricting themselves to a limited selection of devices and then tailoring their designs to these devices, or (b) using a software program to help them decide which devices best fit their requirements on an application-by-application basis.

INTRODUCING CPLDS

The one truism in electronics is that everyone is always looking for things to get bigger (in terms of functional capability), smaller (in terms of physical size), faster, more powerful, and cheaper—surely that’s not *too* much to ask, is it? Thus, the tail end of the 1970s and the early 1980s began to see the emergence of more sophisticated PLD devices. In order to distinguish these little scamps from their less-sophisticated ancestors (which still find use to this day), these new devices were referred to as *Complex PLDs* (CPLDs). Perhaps not surprisingly, it subsequently became common practice to refer to the original, less-pretentious versions as *Simple PLDs* (SPLDs).

Just to make life more confusing, some people understand the terms PLD and SPLD to be synonymous, while others regard PLD as being a superset that encompasses both SPLDs and CPLDs (unless otherwise noted, we shall embrace this latter interpretation; Figure 16.15).

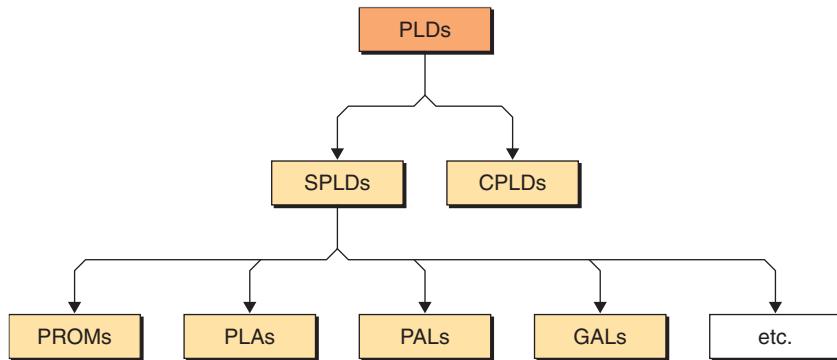


FIGURE 16.15

The introduction of CPLDs.

Leading the fray were the inventors of the original PAL devices—the guys and gals at Monolithic Memories Inc. (MMI)—who introduced a component they called a MegaPAL. This was an 84-pin device that essentially comprised four standard PALs with some interconnect linking them together. Unfortunately, the MegaPAL consumed a disproportionate amount of power and it was generally perceived to offer little advantage compared to using four individual devices.

The big leap forward occurred in 1984, when newly formed Altera Corporation introduced a CPLD based on a combination of CMOS and EPROM technologies. Using CMOS allowed Altera to achieve tremendous functional density and complexity while consuming relatively little power. And basing the programmability of these devices on EPROM cells made them ideal for use in development and prototyping environments.

Having said this, Altera's claim to fame wasn't due only to the combination of CMOS and EPROM. When engineers started to grow SPLD architectures into larger devices like the MegaPAL, it was originally assumed that the central interconnect array (also known as the *programmable interconnect matrix*) linking the individual SPLD blocks required 100% connectivity to the inputs and outputs associated with each block. The problem was that a 2x increase in the size of the SPLD blocks (equating to 2x the inputs and 2x the outputs) resulted in a 4x increase in the size of the interconnect array. In turn, this resulted in a huge decrease in speed coupled with higher power dissipation and component costs.

Altera made the conceptual leap to using a central interconnect array with less than 100% connectivity (see the discussions associated with Figure 16.17 for a tad more information on this concept). This increased the complexity of the software design tools, but it kept the speed, power, and cost of these devices scalable.

Although every CPLD manufacturer fields its own unique architecture, a generic device might consist of a number of SPLD macrocell blocks (typically PALs) sharing a common programmable interconnection matrix (Figure 16.16). As opposed to SPLD macrocell blocks, some modern CPLDs are based on *Look-Up Tables* (LUTs), which we'll discuss in more detail when we come to consider *Field Programmable Gate Arrays* (FPGAs).

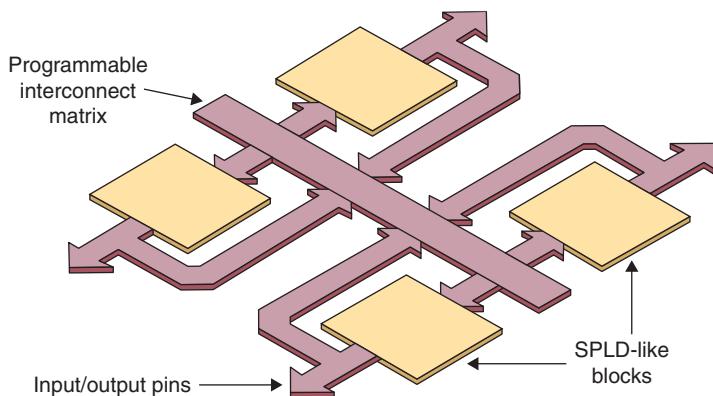
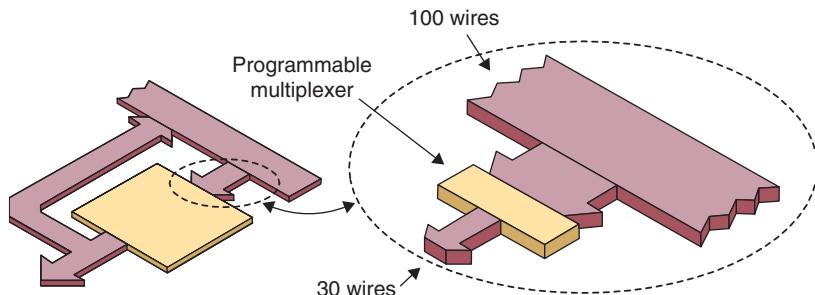


FIGURE 16.16
A generic CPLD structure.

In addition to programming the individual SPLD blocks, the connections between the blocks can be programmed using the programmable interconnect matrix.

Of course, Figure 16.16 is an abstract representation. In reality, all of these structures are formed on the same piece of silicon, and there are a variety of additional features not shown here. For example, the programmable interconnect matrix may contain a lot of wires (say 100), but this is more than can be handled by the individual SPLD blocks, which might only be able to accommodate a limited number of signals (say 30). Thus, the SPLD blocks are interfaced to the interconnect matrix using some form of programmable multiplexer (Figure 16.17).

Depending on the manufacturer and the device family, the CPLD's programmable switches may be based on EPROM, E²PROM, FLASH, or SRAM cells. In the case of SRAM-based devices, some variants increase their versatility by allowing the SRAM cells associated with each SPLD block to be used either as programmable switches or as an actual chunk of memory.

**FIGURE 16.17**

Using programmable multiplexers.

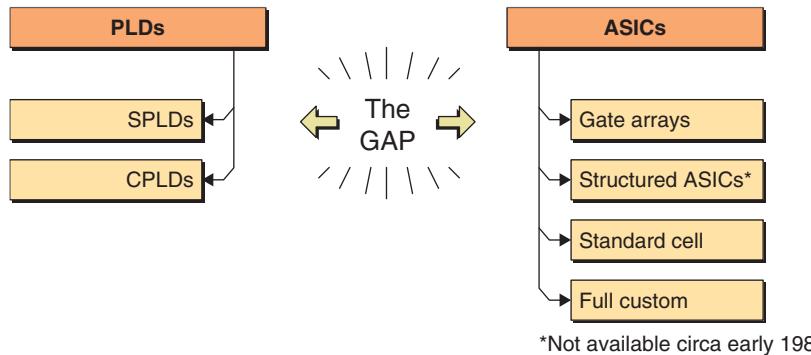
CPLDs find use in a wide range of applications. Flash-based versions are particularly useful in the case of portable (handheld) products, which require low power consumption, small packages, and low price. Many CPLDs support multiple banks of configurable input/output (I/O) pins, where different banks can be programmed to use a variety of I/O standards and voltages. Thus, one use for CPLDs is to act as an interface between other components that use different I/O standards.

Sometimes a system's application processor might be limited in the number of I/O pins it supports. In this case, a CPLD can be used in a pin-expansion role; the application processor "talks" to the CPLD, which in turn "talks" to a number of other devices. Another use is to off-load system tasks from the power-hungry application processor to a power-frugal CPLD.

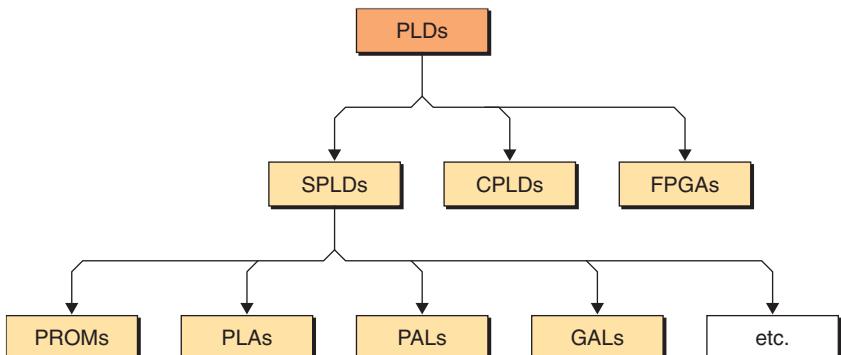
INTRODUCING FPGAS

Around the beginning of the 1980s, it became apparent that there was a "gap" in the digital integrated circuit continuum (Figure 16.18). On the one hand there were programmable devices like SPLDs and CPLDs, which were highly configurable and had fast design and modification times, but which couldn't support large or complex functions. At the other end of the spectrum were ASICs (these devices are presented in more detail in *Chapter 17: Application-Specific Integrated Circuits (ASICs)*).

As we shall see, ASICs can support extremely large and complex functions, but they are expensive and time-consuming to design. Furthermore, once a design is implemented as an ASIC, it's effectively "frozen in silicon." In order to address this gap (Figure 16.19), in 1984 a company called Xilinx introduced a new class of integrated circuit to the market; this new component was called a *Field Programmable Gate Array (FPGA)*.

**FIGURE 16.18**

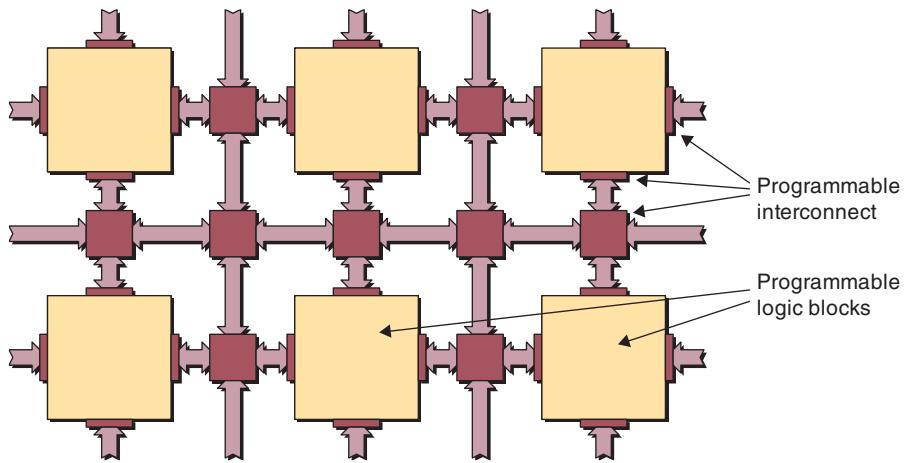
The gap between PLDs and ASICs.

**FIGURE 16.19**

FPGAs join the PLD fraternity.

One way to visualize an FPGA is as a large number of programmable logic block “islands” surrounded by a “sea” of programmable interconnects, as illustrated in Figure 16.20. Only a few logic blocks are shown here, but a modern device can contain hundreds of thousands of such blocks. As we’ll see in the next topic, each of the logic blocks can be configured (programmed) to perform a specific function. Furthermore, the interconnect can be configured to connect the inputs and outputs of the various logic blocks together as required.

As usual, this high-level illustration is merely an abstract representation; in reality, all of the logic and interconnect would be implemented on the same piece of silicon using standard integrated circuit creation techniques. In addition to the local interconnect shown in Figure 16.20, there would also be global (high-speed) interconnection paths that can transport signals across the chip without having to go through multiple local switching elements.

**FIGURE 16.20**

One way to visualize a generic FPGA architecture.

The device will also include primary input/output (I/O) pins and pads (not shown here). By means of its configuration cells, the interconnect can be programmed so that the primary inputs to the device are connected to the inputs of one or more programmable logic blocks, and the outputs from any logic block can be used to drive the inputs to any other logic block and/or the primary outputs from the device.

The end result is that FPGAs successfully bridge the gap between PLDs and ASICs. On the one hand, they are highly configurable and have the fast design and modification times associated with PLDs. On the other hand, they can be used to implement large and complex functions that had previously been achievable only using ASICs.

ALTERNATIVE FPGA ARCHITECTURES

Now, sit down and take a deep breath, because in a moment I'm going to use a scary word: "fabric."¹⁰ In the context of a silicon chip, this is used to refer to the underlying structure of the device (sort of like the phrase "*The fabric of civilization ...*"). When you first hear someone using *fabric* in this way, it might sound a little "snoopy" or pretentious. Truth to tell, however, once you get used to it, you'll find that this can be a jolly useful word.

¹⁰On the off-chance you were wondering, the word *fabric* comes from the Middle English *fabryke*, meaning "something constructed."

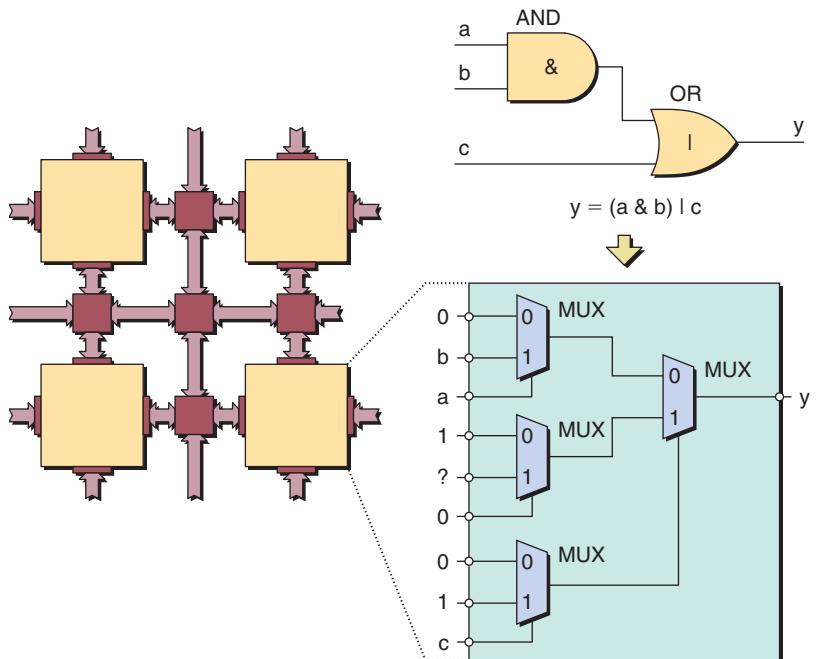
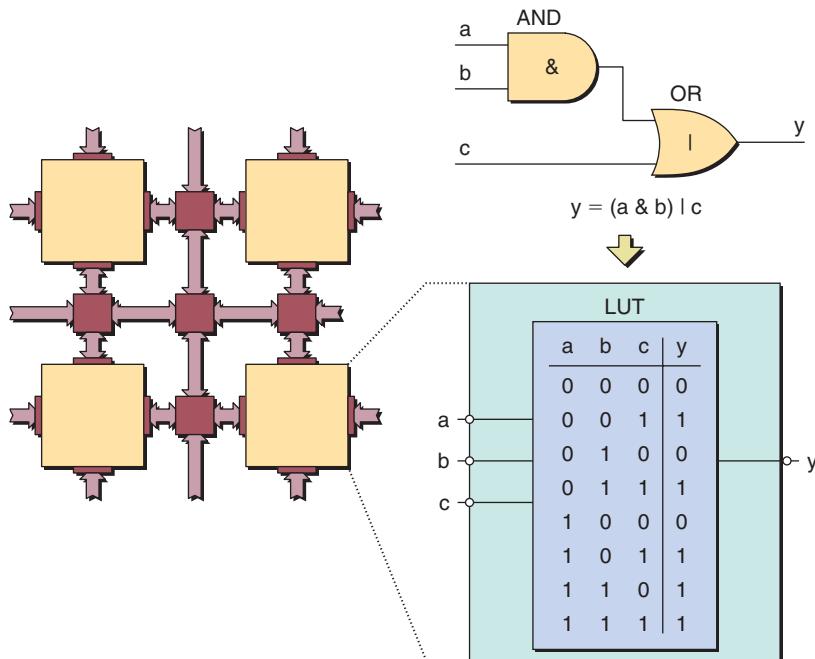


FIGURE 16.21
MUX-based logic blocks.

With regard to the logic blocks that we saw in Figure 16.20, there are a number of ways in which these may be implemented. One possibility is to use a multiplexer (MUX) approach. As an example, consider one way in which the 3-input function $y = (a \& b) | c$ could be implemented using a block containing only multiplexers (Figure 16.21). (Note that in a real device, each logic block would contain additional components such as one or more registers.)

The device can be configured (programmed) such that each input to the block is presented with a logic 0, a logic 1, or the true or inverse version of a signal ("a," "b," or "c" in this case) coming from another block or from a primary input to the device. This allows each block to be configured in myriad ways to implement a plethora of possible functions. (The "?" shown on the input to the central multiplexer in Figure 16.21 indicates that we don't care whether this input is connected to a 0 or a 1.)

Although some folks really like MUX-based FPGAs, the vast majority of today's FPGA fabrics are based on a *Look-Up Table* (LUT) architecture as illustrated in Figure 16.22. (Once again, in the real world, each logic block would contain additional components such as registers, by-pass multiplexers, and such-like.)

**FIGURE 16.22**

LUT-based logic blocks.

The underlying concept behind a LUT is relatively simple. The input signals to the block are used as an index (pointer) into a lookup table. The FPGA is configured (programmed) so that the cell pointed to by each input combination contains the desired output value.

The first FPGAs used 3-input LUTs. After a couple of years, 4-input LUTs became standard, and this persisted until around 2006. At the time of this writing, the smaller FPGA families still use 4-input LUTs, but the really high-end devices may use 6- or 8-input LUTs. (These big-boys may be used as a single large LUT or split into smaller versions such as two 4-input LUTs, or a 3-input and a 5-input LUT.)

In addition to their underlying programmable fabric, different FPGAs will contain different combinations of hard macro blocks, including blocks of on-chip RAM, adders, multipliers, *Digital Signal Processing* (DSP) functions, processor cores, and so forth.¹¹ (Actually, relatively few FPGAs support processor cores in

¹¹For example, a recently introduced FPGA at the time of this writing offers 680K logic blocks, 22.4 megabits of internal RAM, and 1360 18 × 18 multipliers.

the form of dedicated hard macro blocks; in most cases, we use one or more *soft cores* by configuring a portion of the programmable fabric to function as a microprocessor or microcontroller.)

High-end FPGAs can contain hundreds or thousands of input/output (I/O) pins. These pins are arranged in multiple banks, and each bank can be configured (programmed) to use variety of I/O standards and voltages. Also, FPGAs may contain hard-macro serial-gigabit transceiver blocks that support high-speed chip-to-chip communication using a variety of standards like Fibre Channel, InfiniBand®, PCI Express®, RapidIO™, SkyRail™, and 10-gigabit Ethernet.

ALTERNATIVE FPGA CONFIGURATION TECHNOLOGIES

Some FPGAs are configured using antifuse technology; others are programmed using FLASH configuration cells. Antifuse-based devices are programmed outside of the system in which they will ultimately reside; FLASH-based devices may be programmed outside or inside the system [in this latter case they are said to be *In-System Programmable* (ISP)]. Two big advantages of antifuse and FLASH-based fabrics are their relatively low power consumption and their nonvolatile nature, which provides “instant-on” capability.

The majority of FPGAs are based on SRAM configuration cells, which allows them to be quickly and easily reprogrammed. In addition to their versatility, a big advantage of these devices is the fact that they use only standard CMOS processes, which means they can ride the current latest and greatest technology wave. The main disadvantages are that they consume a lot of power (relative to their antifuse and FLASH cousins) and that they require an external configuration device (this is typically presented in the form of a FLASH memory chip).

Some FPGAs use a hybrid combination of SRAM and FLASH on the same chip.¹² Each programmable element in the device has an associated FLASH bit and an SRAM cell. The idea here is that when the device powers up, the contents of each FLASH bit are copied into its corresponding SRAM cell. Such a device is essentially “instant on,” but it can also be easily reprogrammed as required. Furthermore, once the device has been powered-up and is running under its initial configuration, the chip can continue running while its

¹²One family combines an SRAM-based FPGA die with a FLASH memory die in a single package.

FLASH is reprogrammed with a new configuration. This new configuration can subsequently be copied over into the SRAM configuration cells in a fraction of a second.

MIXED-SIGNAL FPGAS, CSSPS, AND ...

In addition to their digital fabric, some FPGAs also include configurable analog functions, such as analog inputs and outputs, analog-to-digital and digital-to-analog converters, operational amplifiers, and so forth. This combination of digital and analog functionality is referred to as *mixed-signal*. Of particular interest is the fact that you can configure a part of the digital fabric to act as a soft microprocessor core, and then program this core to monitor the analog signals and to reconfigure (“fine tune”) the analog functions on-the-fly.

Another class of components called *Customer-Specific Standard Products* (CSSPs) combine traditional FPGA programmable fabric with collections of hard macros. These hard macros include functions that can interface to different types of external memory devices, provide various communications functions, the ability to drive display devices, and so forth.

In *Chapter 17: Application-Specific Integrated Circuits (ASICs)*, we will introduce the concept of a *System-on-Chip* (SoC), which is generally regarded to be an ASIC that includes one or more processor cores, memory, and other functions on a single chip. Actually, when you come to think about it, an FPGA is really an ASIC that has been constructed in such a way as to be configurable. On this basis, some folks would refer to an FPGA containing memory whose fabric has been configured to implement a processor core and other functions as being an SoC. Other folks would prefer to use the term *System-on-a-Programmable-Chip* (SoPC). Of course, all of this is just playing with words; this is one of those times when “*you pays your money and you makes your choice*,” as the old saying goes.

SUMMARY

The problem with programmable logic devices in the form of SPLDs, CPLDs, and FPGAs is that there are exceptions to every rule. There is a baffling and bewildering mixture of architectures and configuration technologies, so the best we can hope for here is to provide a very general overview.

In the case of the underlying programmable fabric, we have three main types: Macrocell (AND-OR or similar arrays), MUX-based, or LUT-based (see Table 16.1).

Table 16.1

Device types versus programmable fabrics

Device Type	Underlying Fabric		
	Macrocell	MUX	LUT
SPLD	Yes	No	No
CPLD	Some	Some	Some
FPGA	No	Some	Most

Similarly, in the case of the various configuration (programming) technologies, the best we can hope for is to illustrate which technologies are predominantly associated with which type of components while remembering that there are all sorts of oddball devices roaming the world (see Table 16.2).

Table 16.2

Device types versus configuration technologies

Configuration Technology	Symbol	Predominantly associated with ...
Fusible-link	—~—	SPLDs
Antifuse	—□—	Some FPGAs
EPROM	— —	SPLDs and CPLDs
E ² PROM/FLASH	— —	SPLDs and CPLDs, some FPGAs
SRAM		Most FPGAs, some CPLDs
Hybrid SRAM/Flash		Some FPGAs

CHAPTER 17

Application-Specific Integrated Circuits (ASICs)

INTRODUCING ASICS

As its name might suggest, an *Application-Specific Integrated Circuit* (ASIC) is a device whose function is determined by design engineers to satisfy the requirements of a particular application.

In 1967, Fairchild Semiconductor introduced a device called the *Micromosaic*, which contained a few hundred transistors. The key feature of the Micromosaic was that the transistors were not initially connected together. Design engineers used a computer program to specify the function they wished the device to perform. This program determined the necessary interconnections required to link the transistors and generated the masks required to complete the device. Although relatively simple, the Micromosaic is credited as being the forerunner of the modern ASIC and the first real application of *Computer-Aided Design* (CAD).

235

A formal classification of ASICs tends to become a bit fluffy around the edges. However, it is generally accepted that there are four major categories of these little rascallions: *Gate Array* (GA) devices (including sea-of-gates), *Structured ASICs*, *Standard Cell* (SC) components, and *Full Custom* parts (Figure 17.1).

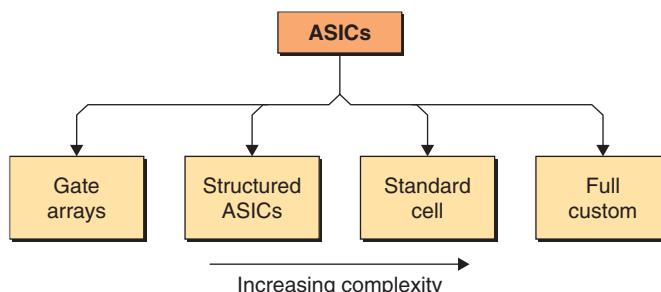


FIGURE 17.1

The four main ASIC categories.

In the following topics we will consider these devices in the order in which they appeared on the scene.

FULL CUSTOM DEVICES

In the early days of digital integrated circuits, there were really only two main classes of devices (excluding memory chips and simple programmable logic devices). The first were relatively simple building-block-type components that were created by companies like TI and Fairchild and sold as standard off-the-shelf parts to anyone who wanted to use them. The second were full-custom ASICs like microprocessors, which were designed and built to order for use by a specific company.

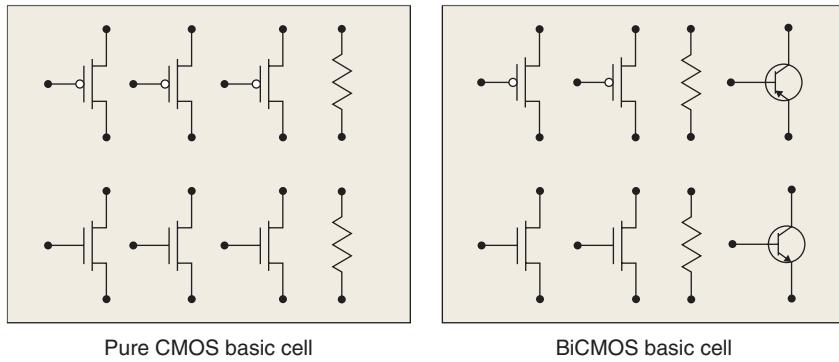
In the case of *full custom* devices, the design engineers have complete control over every mask layer used to fabricate the silicon chip. The ASIC vendor does not prefabricate any components on the silicon and does not provide any cell libraries. With the aid of appropriate design tools, engineers can “handcraft” the dimensions of individual transistors, and then create higher-level logic functions based on these components. For example, if the engineers require a slightly faster logic gate, they can alter the dimensions of the transistors used to build that gate. The design tools used for full custom devices are often created in-house.

Full custom devices may also include analog circuitry such as comparators, amplifiers, filters, and digital-to-analog and analog-to-digital converters. The design of full custom devices is highly complex and time consuming, but the resulting chips contain the maximum amount of logic that consumes the minimum amount of power with minimal waste of silicon real estate.

GATE ARRAYS

A silicon chip may be considered to consist of two major facets: the components such as transistors and resistors and the tracks connecting the components together. In the case of *gate arrays*, the ASIC vendor prefabricates wafers of silicon chips containing unconnected transistors and resistors.¹ Gate arrays are based on the concept of a *basic cell*, which consists of a selection of components; each ASIC vendor determines the numbers and types of components provided in their particular basic cell (Figure 17.2).

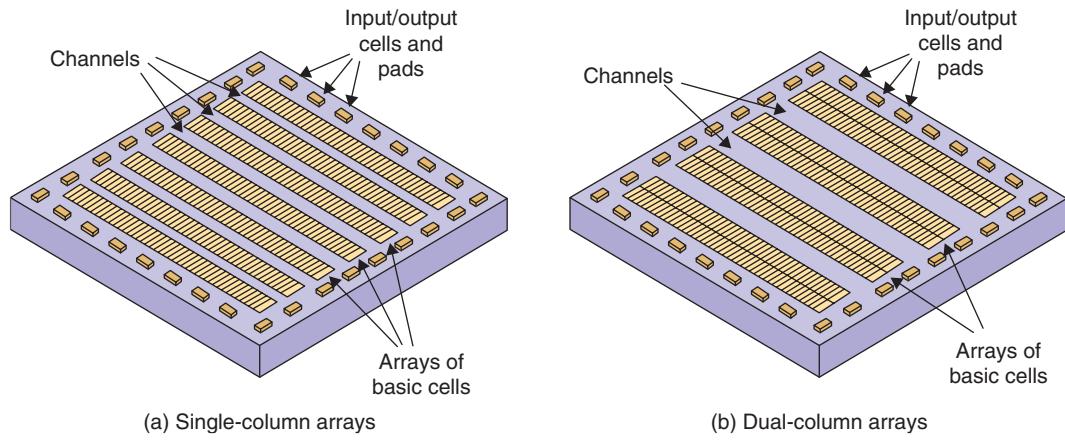
¹In the early days, gate arrays were also known as *Uncommitted Logic Arrays* (ULAs), but this term has largely fallen into disuse.

**FIGURE 17.2**

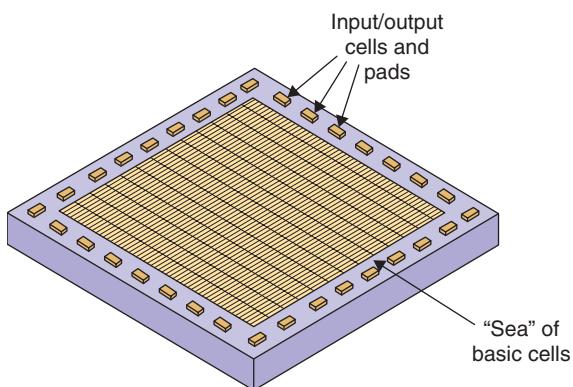
Examples of gate array basic cells.

The first types of gate arrays were called *channeled gate arrays*. In this case, the basic cells were typically presented as either single-column or dual-column arrays, where the free areas between the arrays were known as the *channels* (Figure 17.3). (Note that, although these diagrams feature individual silicon chips, at this stage in the process these chips would still be embedded in the middle of a wafer.)

By comparison, in the case of *channel-less* or *channel-free* devices, the basic cells are presented as a single large array. The surface of the device is covered in a

**FIGURE 17.3**

Channeled gate arrays.

**FIGURE 17.4**

Channel-less gate arrays.

was no longer available to the user. More recent processes, which can have up to eight or ten metallization layers, overcome this problem.

"sea" of basic cells, and there are no dedicated channels for the interconnections. Thus, these devices are popularly referred to as *sea-of-gates* or *sea-of-cells* (Figure 17.4).

The channels in channeled devices are used for the tracks that connect the logic gates together. By comparison, in channel-less devices, the connections between logic gates have to be deposited over the top of other basic cells. In the case of early processes based on only two layers of metallization, any basic cell overlaid by a track

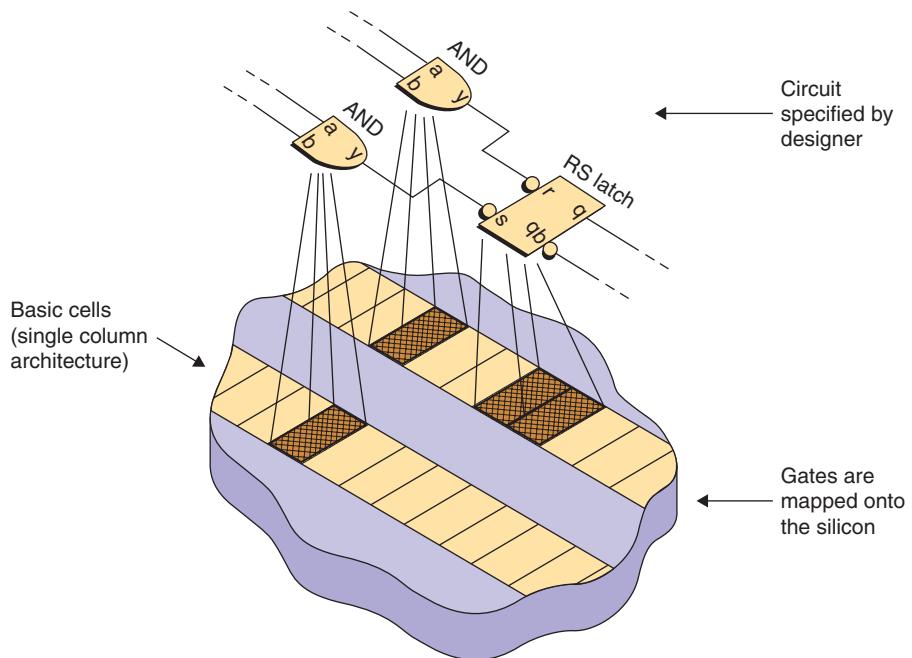
HIGH-LEVEL VIEW OF THE GATE ARRAY DESIGN FLOW

ASIC design flows are discussed in more detail in *Section 3: Design Tools and Stuff*, but providing a brief, high-level overview here will make things a lot clearer.

Although the transistors from one or more basic cells can theoretically be connected together to implement practically any digital function, design engineers using gate arrays do not work at the transistor level. Instead, each ASIC vendor selects a set of logic functions such as primitive gates, multiplexers, and registers that they wish to make available to the engineers. The vendor also determines how each of these functions can be implemented using the transistors from one or more basic cells.

A primitive logic gate may only require the transistors from a single basic cell, while a more complex function like a D-type flip-flop or a multiplexer may require the use of several basic cells. Each of these "building block" functions is referred to as a *cell*—not to be confused with a *basic cell*—and the set of functions provided by the ASIC vendor are known collectively as the *cell library*. The number of functions in a typical cell library can range from 50 to 250 or more.

Aided by a suite of design tools (design capture, synthesis, simulation, etc.), design engineers end up with a gate-level *netlist*. This netlist describes the cells to be used and the connections that have to be made between them in order for the gate array to perform its desired function. This netlist is then passed to the layout portion of the flow. First, the *place engine* or *placer* assigns the cells selected by the engineers to basic cells on the silicon (Figure 17.5).

**FIGURE 17.5**

Cells (logic gates and functions) in the netlist are mapped onto basic cells.

Next, the *route engine* or *router* determines the optimal way to connect the cells together (these tools operate hand-in-hand and are referred to collectively as *place-and-route*). Further tools are used to create the masks required to implement the final metallization layers. These layers are used to connect the transistors in the basic cells to form logic gates, and to connect the logic gates to form the complete device.

Functions in gate-array cell libraries are generally fairly simple, ranging from primitive logic gates to the level of registers. The ASIC vendor may also provide libraries of more complex logical elements called *hard macros* (or *macro-cells*) and *soft macros* (or *macro-functions*).² In the case of gate arrays, the functions represented by the hard-macros and soft-macros are usually along the lines of comparators, shift-registers, counters, and adders. Hard macros and soft macros are both constructed using cells from the cell library. In the case of a hard macro, the ASIC vendor predetermines how the cells forming the macro will be assigned to the basic cells and how the connections between the basic cells will

²The term *macro* was inherited from the software guys. In software terms, a macro is like a subroutine. So a single computer instruction (to call the macro) initiates a series of additional instructions for the computer to perform. Similarly, in hardware terms, a single macro represents a large number of primitive logic gates.

be realized. By comparison, in the case of a soft macro, the assignment of cells to basic cells is performed at the same time, and by the same tool, as it is for the simple cells specified by the design engineers.

Gate arrays are classed as *semi-custom devices*. The definition of a semi-custom device is that it has one or more customizable mask layers, but not all the layers are customizable. Additionally, the design engineers can only utilize the predefined logical functions provided by the ASIC vendor in the cell and macro libraries. (More recent gate array incarnations may include pre-created hard macros such as blocks of memory, processor cores, and peripheral functions. These are not implemented using basic cells; instead, they are fabricated as custom blocks directly on the chip.)

The main disadvantages of gate arrays (as compared to other ASIC implementations) are their somewhat lower density, and their performance. Having said this, gate arrays often offer a viable approach for relatively low-volume production runs. Gate arrays “ruled the ASIC roost” until the introduction of standard cell devices, which are presented in the next topic. Ever since standard cell components started to take off, industry observers have been predicting the demise of gate arrays, but these little scamps continue to hold onto a small niche in the market.

STANDARD CELL DEVICES

Standard cell devices bear many similarities to gate arrays. Once again, each ASIC vendor decides which logic functions they wish to make available to the design engineers. Some vendors supply both gate array and standard cell devices, in which case the majority of the logic functions in the cell libraries will be identical and the main differences will be in their timing attributes.

Standard cell vendors also supply hard macro and soft macro libraries, which include elements such as processors, controllers, and communication functions. Additionally, these macro libraries typically include a selection of RAM and ROM functions, which were implemented inefficiently in early gate array devices. Last but not least, the design engineers may decide to reuse previously designed functions and/or to purchase blocks of *Intellectual Property (IP)*.

There are several different flavors of intellectual property. In the generic sense, if you have a capriciously cunning idea, then that idea is your intellectual property and no one else can use it unless they pay you for it. When a team of electronics engineers is tasked with designing a complex integrated circuit, rather than “reinventing the wheel,” they may decide to purchase the plans for one or more

functional blocks that have already been created by someone else. The plans for these functional blocks are known as *Intellectual Property* (IP). IP blocks can range all the way up to sophisticated communications functions and microprocessors. The more complex functions (like microprocessors) may be referred to as *cores*.

HIGH-LEVEL VIEW OF THE STANDARD CELL DESIGN FLOW

Once again, aided by a suite of design tools, the design engineers determine which elements they wish to use from the cell and macro libraries, and how they require these elements to be connected together. Unlike gate arrays, however, standard cell devices do not use the concept of a basic cell, and no components are prefabricated on the silicon chip. The ASIC vendor creates custom masks for every stage of the device's fabrication. This allows each logic function to be created using the minimum number of transistors required to implement that function with no redundant components. Additionally, the cells and macro functions can be located anywhere on the chip, there are no dedicated interconnection areas, and the functions are placed so as to facilitate any connections made between them. Standard cell devices therefore provide a closer-to-optimal utilization of the silicon than do gate arrays (Figure 17.6).

1T VERSUS 6T SRAM

The vast majority of ASICs include some amount of memory. By 2008, 50% or more of the silicon might be consumed by memory in certain devices; and some folks believe that memory will account for 95% or more of a large proportion of designs by 2016.

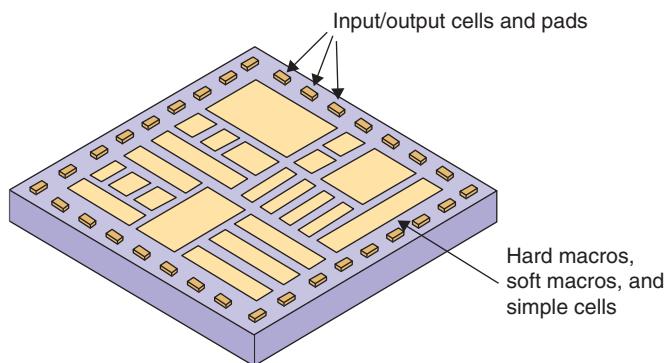


FIGURE 17.6

Logic functions can be placed anywhere on a standard cell device.

So, what's the big deal? Well, if you use standard 6T SRAM (that's SRAM in which each memory cell is formed from six transistors), this certainly provides the speed you need, but it takes a relatively large amount of space and consumes a relatively large amount of power.

The alternative is to use something called 1T SRAM, in which each cell is implemented using a single transistor. But isn't this DRAM? Well, yes and no. It's certainly true to say that if we look under the hood this looks very much like DRAM on a single-cell basis. As we know from *Chapter 15: Memory ICs*, a DRAM cell is significantly smaller than its SRAM equivalent, and it consumes less power, but it's also significantly slower. However, as we also know from *Chapter 15*, in the case of Synchronous DRAM (SDRAM), we can arrange banks of DRAM in such a way as to increase the speed of the memory subsystem as a whole.

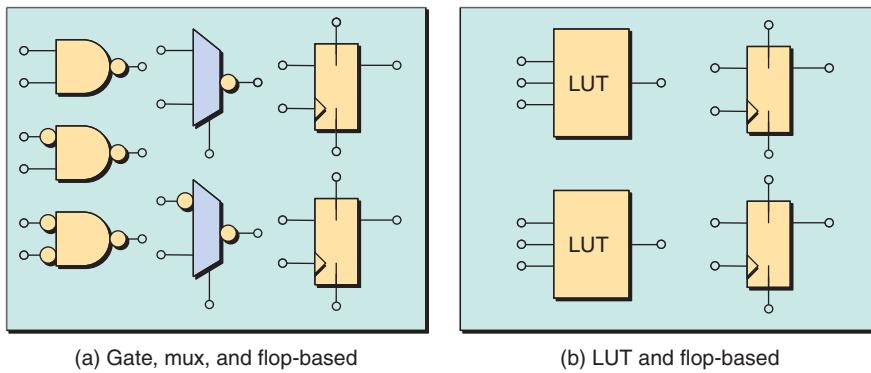
Similarly, the way in which a block of 1T SRAM is arranged (with its own internal refresh circuitry and interface circuitry and suchlike) means that it actually appears to be a block of (very dense) high-speed SRAM to the outside world (the rest of your design). Until recently, this form of memory required extra process steps to augment the underlying CMOS process, but modern forms can be implemented using standard CMOS technology.

STRUCTURED ASICS

The concept of “*Structured ASICs*” (although they weren’t called that at the time) spluttered into life around the beginning of the 1990s, slouched around for a while, and then returned to the nether regions from whence it came. A decade later—circa 2001 to 2002—a number of ASIC manufacturers started to investigate innovative ways of reducing ASIC design costs and development times. Not wishing to be associated with traditional gate arrays, everyone was happy when someone came up with the “*Structured ASIC*” moniker somewhere around the middle of 2003.

As usual, of course, every vendor has their own proprietary architecture, so our discussions here will provide only a generic view of these components. Each device commences with a fundamental element that is called a *module* by some and a *tile* by others. This element may contain a mixture of prefabricated generic logic (implemented either as gates, multiplexers, or as a lookup table), one or more registers, and possibly a little local RAM (Figure 17.7).

An array (sea) of these elements is then prefabricated across the face of the chip. Alternatively, some fabrics commence with a *base cell* (or *base tile* or *base*

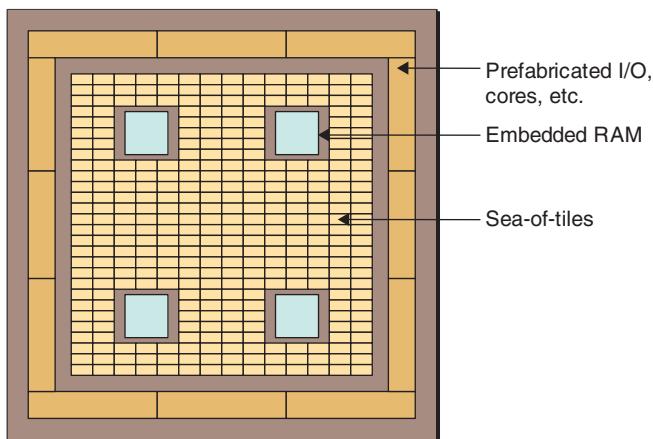
**FIGURE 17.7**

Examples of simple structured ASIC tiles.

module or ...) containing only generic logic in the form of prefabricated gates, multiplexers, and/or lookup tables. In conjunction with some special units containing registers, small memory elements, and other logic, an array of these base units (say 4×4 , 8×8 , or 16×16) then make up a *master cell* (or *master tile* or *master module* or ...). Once again, an array (sea) of these master units is then prefabricated across the face of the chip.

Also prefabricated are functions like RAM blocks, clock generators, boundary scan logic, and so forth (Figure 17.8).

The idea is that the device can be customized using only the metallization layers (just like a standard gate array). The difference is that—due to the higher level of sophistication of the structured ASIC tile—the majority of the metallization

**FIGURE 17.8**

A generic structured ASIC architecture.

layers are also predefined. Thus, many structured ASIC architectures require the customization of only two or three metallization layers (in one case, it is necessary to customize only a single via layer). This dramatically reduces the time and costs associated with creating the remaining photo-masks used to complete the device.

Although it's difficult to assign an exact value, the predefined and prefabricated logic associated with structured ASICs results in an overhead compared to standard cell devices in terms of power consumption, performance, and silicon real estate.

So what are the advantages with regard to structured ASICs? Well, although structured ASICs can't attain the high performance and low power consumption of full-blown ASICs, they come reasonably close. But the real advantages of structured ASICs are reduced design time, effort, and cost (for medium-size production runs). This is because the majority of the layers forming the device (silicon, polysilicon, and metallization) are prefabricated. All that is required is to customize relatively few of the metallization layers to complete the device. This significantly reduces its cost (the development expenses associated with the underlying fabric can be amortized across all of the end users) and also dramatically reduces the time required to get working prototypes back from the foundry.

Furthermore, the act of designing structured ASICs is much simpler than for a full-blown ASIC, because the majority of the signal integrity, power grid, and clock-tree distribution issues have already been addressed by the structured ASIC vendor. This means that the design team doesn't require extreme amounts of expertise in these areas, and also that they don't have to spend huge amounts of money on the incredibly sophisticated analysis tools that are a prominent feature in full-blown ASIC design flows.

Now, one point to consider is that a lot of structured ASIC fabrics are based on proprietary cells (logic functions). This means that when a foundry introduces a new technology node, these cells have to be re-implemented from the ground up, which is expensive and—more importantly—time-consuming. In order to address this issue, some structured ASIC vendors base their fabric on cells from the foundry's standard libraries. Using these library cells allows the structured ASIC vendor to dramatically increase the availability of its offering (and reduce risk) by leverage all of the work performed by the foundry in qualifying the new process.

Last but not least, at least one structured ASIC is based on the concept of *standard metal*. The idea here is that all of the tracks on the device are pre-created

in the form of track segments. All that is required is for the customization of one or two via layers to make the required connections between different track segments. The advantage of this approach is that all of the tracks can be pre-characterized in terms of their capacitance, resistance, inductance, signal delays, and so forth; also that the vast majority of signal integrity issues have already been resolved by the vendor.

INPUT/OUTPUT (I/O) CELLS AND PADS

Around the periphery of the silicon chip are power and signal pads, used to interface the device to the outside world. The cell-library data books for gate array and standard cell devices include a set of functions known as *Input/Output* (I/O) cells. The signal pads contain a selection of transistors, resistors, and diodes necessary to implement *input*, *output*, or *bidirectional buffers*, and the design engineers can decide how each pad will be configured. The masks and metallization used to interconnect the internal logic are also used to configure the components in the input/output cells and to connect the internal logic to these cells (Figure 17.9).

The ASIC vendor may permit the design engineers to individually specify whether each input/output cell should present CMOS, TTL, or ECL characteristics to the outside world. The input/output cells also contain any circuitry required to provide protection against *Electrostatic Discharge* (ESD). After the

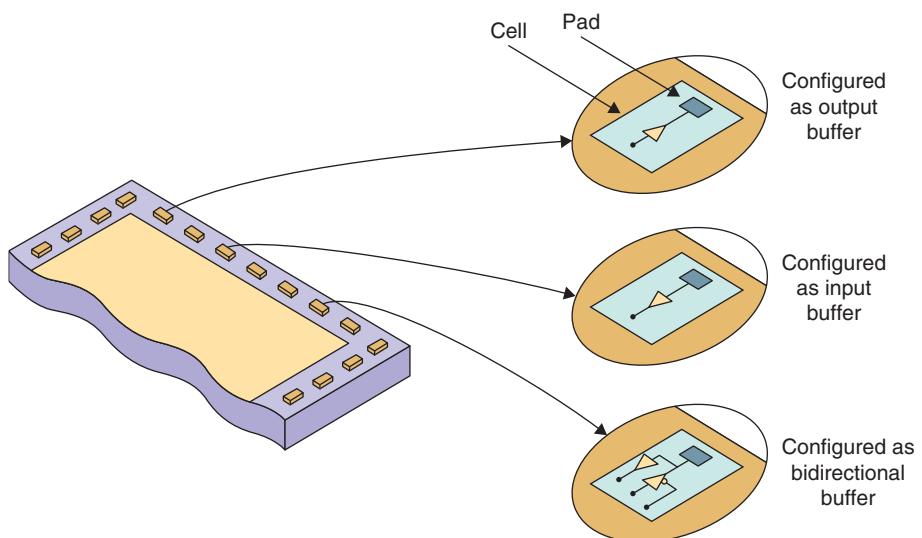


FIGURE 17.9
Input/Output (I/O) cells and pads.

metallization layers have been added, the chips are separated and packaged using the same techniques as for standard integrated circuits.³

ASICS VERSUS ASSPS

Generally speaking, an *Application-Specific Integrated Circuit* (ASIC) is a component that is designed by and/or used by a single company in a specific system. By comparison, an *Application-Specific Standard Product* (ASSP) is a more general-purpose device that is created using ASIC tools and technologies, but that is intended for use by multiple system design houses. Meanwhile, a *System-on-Chip* (SoC) is an ASIC or ASSP that acts as an entire subsystem including a microprocessor or microcontroller, memory, peripherals, custom logic, and so forth.

WHO ARE ALL THE PLAYERS?

It can be useful to have a high-level view as to who all of the players are in this complicated game (understanding the technology is the easy part; it's when you try to work out who does what to whom that things start to get hairy).

First of all, we have the folks who create the tools (software programs) that the engineers use to design integrated circuits, circuit boards, and electronic systems. For historical reasons, the tools used to capture and verify a design (either an integrated circuit or a circuit board) were classed as *Computer-Aided Engineering* (CAE). By comparison, the layout (place-and-route) tools used to actually implement the design were classed as *Computer-Aided Design* (CAD). Sometime during the 1980s, all of the CAE and CAD tools came to be referred to by the "umbrella" name of *Electronic Design Automation* (EDA), and everyone was happy (apart from the ones who weren't, but they don't count).

If you say things the wrong way when talking to someone in the industry, you immediately brand yourself as an outsider (one of "them" instead of one of "us"). For historical reasons that are based on the origins of the terms CAE and CAD, the term *design engineer* or simply *engineer* is typically used to refer to someone who conceives and describes the functionality of an integrated circuit, printed circuit board, or electronic system (what it does and how it does it). By comparison, the term *layout designer* or simply *designer* is typically used to refer to someone who lays out a circuit board or integrated circuit (determines the locations of the components and the routes of the tracks connecting them together).

³See Chapters 14: *Integrated Circuits (ICs)* and 20: *Advanced Packaging Techniques*, for more discussions on integrated circuit packaging technologies.

Now, this is where things start to get interesting. Let's start with the *System House A* block in the middle of Figure 17.10. These are the folks who design and build the system-level products (from cell phones to televisions to computers) that eventually wend their way into our hands.

When these system house folks are creating a new product, they may decide it requires one or more ASICs, which they also design. Once they've created the design for an ASIC, they may pass it over to an ASIC vendor to be implemented and fabricated. In this case, the ASIC vendor is in charge of creating the masks and constructing and packaging the final devices.

As opposed to a full-line ASIC vendor, a *fabless semiconductor company* is one that designs and sells integrated circuits, but does not have the ability to manufacture them. By comparison, a *foundry* is a company that manufactures integrated circuits, but doesn't actually do any designs of their own (these are also known as *fabs* because they fabricate the integrated circuits).

Initially, we described a system house (represented as *System House A* in Figure 17.10) as passing its ASIC designs over to an ASIC vendor for implementation and fabrication. However, some system houses (represented as *System House B* in Figure 17.10) perform both the design and implementation, and then hand the masks over to a foundry for fabrication. In yet another scenario, a system house (represented as *System House C* in Figure 17.10) may have its own fab capability, in which case it will perform the entire process—design, implementation, and fabrication—"in-house."

And just to confuse the issue even further, we have *Integrated Device Manufacturers* (IDMs). These are companies that are very similar to system houses, except that IDMs focus on designing, manufacturing, and selling integrated circuits as opposed to complete electronic systems.

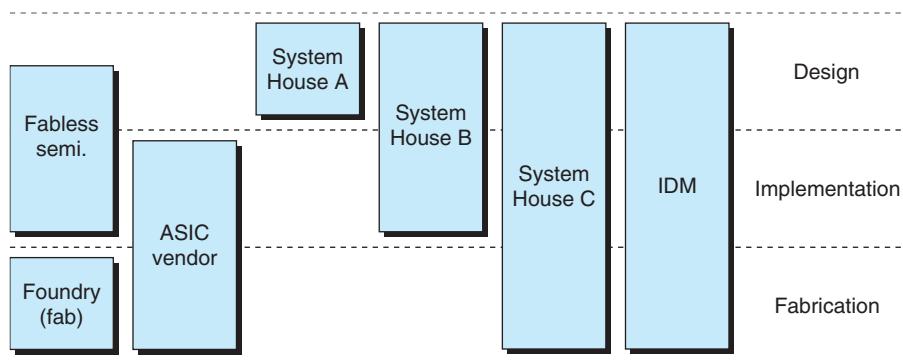


FIGURE 17.10
Some of the key players.

SUMMARY

Electronic designs typically contain a selection of highly complex functions—such as processors, controllers, and memory—interfaced to each other by a plethora of simple functions such as primitive logic gates. The simple interfacing functions are often referred to as the *glue logic* because they “glue” everything together.

Both simple and complex functions may be available as standard, off-the-shelf integrated circuits. Devices containing small numbers of simple logic functions (for example, four 2-input AND gates) are sometimes referred to as *jelly bean parts*. When glue logic is implemented using jelly bean parts, these components may require a disproportionate amount of the circuit board’s real estate. A few complex devices containing the vast majority of the design’s logic gates may occupy a small area, while a large number of jelly bean devices containing relatively few logic gates can occupy the bulk of the board’s surface.

Early ASICs contained only a few hundred logic gates. These were mainly used to implement glue logic, and it was possible to replace fifty or more jellybean devices with a single ASIC. This greatly reduced the size of circuit boards, increased speed and reliability, and reduced power consumption. As time progressed, application-specific devices with tens or hundreds of millions of gates and thousands of pins became available. Thus, today’s ASICs can be used to implement the most complex functions.

Both gate array and standard cell devices essentially consist of building blocks designed and characterized by the manufacturer and connected together by the designer. Gate arrays are mask-programmable with a predefined number of transistors, and different designs are essentially just changes in the interconnect. This means that gate arrays require the customization of fewer layers than standard cell devices. Gate arrays are therefore faster to implement than their standard cell equivalents, but the latter can contain significantly more logic gates.

In certain respects, gate arrays have moved toward having similar capabilities to those of standard cells, and some support complex functions such as memory and processor cores. The design engineers generally know the memory and processing requirements in advance of the rest of the logic, and the gate array manufacturer may supply devices with pre-built processor and memory functions surrounded by arrays of basic cells. Some gate array devices support simple analog cells in addition to the digital cells, and standard cell devices can be constructed with complex analog functions, if required.

Thus far, structured ASICs haven't really gained the high-level of market penetration many folks anticipated, but it's getting harder and harder to design full-blown standard cell ASICs, as timing, power, signal integrity, and other design considerations become increasingly complex with each new technology node. Thus, it may be that structured ASICs will leap onto the center stage with gusto and abandon in the not-so-distant future.

Last but not least, creating full-custom ASIC devices (like high-end microprocessors from Intel and AMD) is so mind-bogglingly complex as to make one's brains leak out of one's ears, and we don't want that to happen, so instead let's bounce over to *Chapter 18: Printed Circuit Boards (PCBs)*, to see what exciting topics are waiting to be discovered.

This page intentionally left blank

CHAPTER 18

Printed Circuit Boards (PCBs)

NOT MUCH FUN

Electronic components are rarely useful in isolation, and it is usually necessary to connect a number of them together in order to achieve a desired effect. Early electronic circuits were constructed using discrete (individually packaged) components such as transistors, resistors, capacitors, and diodes. These were mounted on a nonconducting board and connected using individual pieces of insulated copper wire. The thankless task of wiring the boards by hand was time-consuming, boring, prone to errors, and expensive, and it was generally agreed that this wasn't much fun.

251

THE FIRST CIRCUIT BOARDS

The great American inventor Thomas Alva Edison (1847–1931) had some ideas about connecting electronic circuits together. In a note to Frank Sprague (1857–1934), founder of Sprague Electric, Edison outlined several concepts for printing additive traces on an insulating base. He even talked about the possibility of using conductive inks (it was many decades before this technology—which is introduced later in this chapter—came to fruition).

In 1903, Albert Hanson (a Berliner living in London) obtained a British patent for a number of processes for forming electrical conductors on an insulating base material. One of these described a technique for cutting or stamping traces out of copper foil and then sticking them to the base. Hanson also came up with the idea of double-sided boards and through-holes (which were selectively connected by wires).

In 1913, Arthur Berry filed a British patent for covering a substrate with a layer of copper and selectively etching parts of it away to leave tracks. In another British patent issued in 1925, Charles Ducas described etching, plating-up, and

even multilayer circuit boards (including the means of interconnecting the layers). For the next few decades, however, it was easier and cheaper to wire boards manually. The real push into circuit boards only came with the invention of the transistor and later, with the integrated circuit.

PCBS AND PWBS

By the 1950s, the interconnection technology now known as *Printed Wire Boards* (PWBs) or *Printed Circuit Boards* (PCBs) had gained commercial acceptance. Both terms are synonymous, but the former is more commonly used in America, while the latter is predominantly used in Europe and Asia.¹ These circuit boards are often referred to as *laminates* because they are constructed from thin layers or sheets. In the case of the simpler boards, an insulating base layer has conducting tracks formed on one or both sides. The base layer may technically be referred to as the *substrate*, but this term is rarely used in the circuit board world.²

The original board material was *Bakelite*, but modern boards are predominantly made from woven glass fibers that are bonded together with an epoxy. The board is cured using a combination of temperature and pressure, which causes the glass fibers to melt and bond together, thereby giving the board strength and rigidity. These materials may also be referred to as *organic substrates*, because epoxies are based on carbon compounds as are living creatures. The most commonly used board material of this type is known as FR4, where the first two characters stand for “flame retardant,” and you can count the number of people who know what the “4” stands for on the fingers of one hand.

To provide a sense of scale, a fairly representative board might be 15cm × 20cm in area and in the region of 1.5mm to 2.0mm in thickness, but they can range in size from 2cm × 2cm or smaller (and thinner) to 50cm × 50cm or larger (and thicker).³

ROHS AND LEAD-FREE SOLDER

*Solders*⁴ are fusible metal alloys that have relatively low melting points in the range of 90 to 450°C (200 to 840°F). The reason solders are of interest to us

¹Having said this, I currently live in America and I don't recall hearing anyone say “printed wire board” for quite some time, so maybe the “printed circuit board” appellation has won the day.

²See also the glossary definition of *substrate*.

³These dimensions are not intended to imply that circuit boards must be square or even rectangular. In fact, a circuit board may be constructed with whatever outline is necessary to meet the requirements of a particular enclosure: for example, the shape of a car dashboard.

⁴The word *solder* comes (in a roundabout route from Middle English via Old French) from the Latin *solidare*, meaning “to make solid.”

here is they are used to attach electronic components to circuit boards in a process called *soldering*.

Until recently, solders used in electronic applications were made from tin-lead alloys. In 2003, however, the European Union adopted the *Restriction of Hazardous Substances Directive (RoHS)*. Although many folks think of RoHS as being “the lead-free directive,” it actually restricts the use of six hazardous materials (including lead, mercury, and cadmium) in the manufacture of electronic and electrical equipment.

The RoHS Directive led to the development of a variety of lead-free solders that may contain tin, copper, silver, bismuth, indium, zinc, antimony, and traces of other metals. Typical solders used for the majority of electronics applications are based on mixtures of tin-silver-copper with traces of other metals or metallic elements.

RoHS started to be enforced in Europe in 2006, but some application segments like military, aerospace, and medical applications have been granted exceptions. Many other countries around the world are adopting their own RoHS-like regulations. Although full-blown RoHS-type regulation in the United States is considered to be unlikely at the Federal level in the near- to medium-term, several states have already enacted legislation that requires electronic equipment manufacturers to comply with RoHS requirements. Also, anyone who wishes to sell products into the European Union Markets has to ensure that those products are RoHS compliant.

SUBTRACTIVE PROCESSES

In a subtractive process, a thin layer of copper foil in the order of 0.02-mm thick is bonded to the surface of the board. The copper’s surface is coated with an *organic resist*, which is cured in an oven to form an impervious layer (Figure 18.1). (Note that the “wobbly” edges in this illustration are intended to imply that we are considering only a very small area in the middle of the board.)

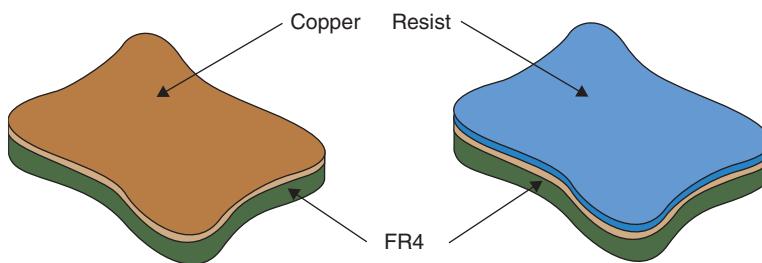
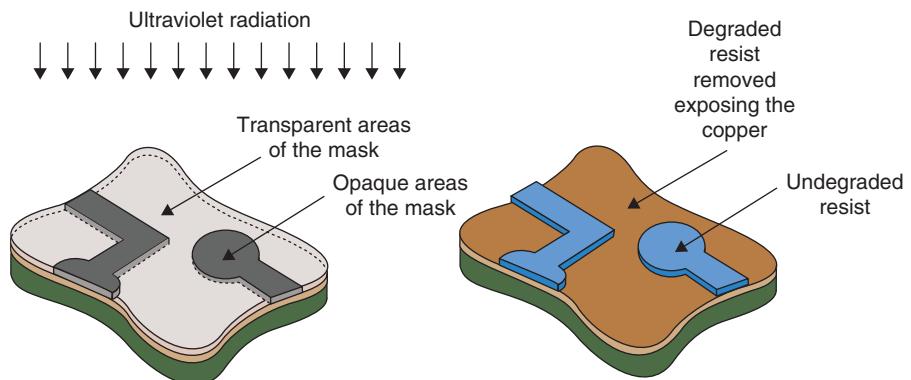


FIGURE 18.1

Subtractive process: Applying resist to copper-clad board.

**FIGURE 18.2**

Subtractive process:
Degrading the resist
and exposing the
copper.

Next, an optical mask is created with areas that are either transparent or opaque to ultraviolet light. The mask is usually the same size as the board, and can contain hundreds of thousands of fine lines and geometric shapes.

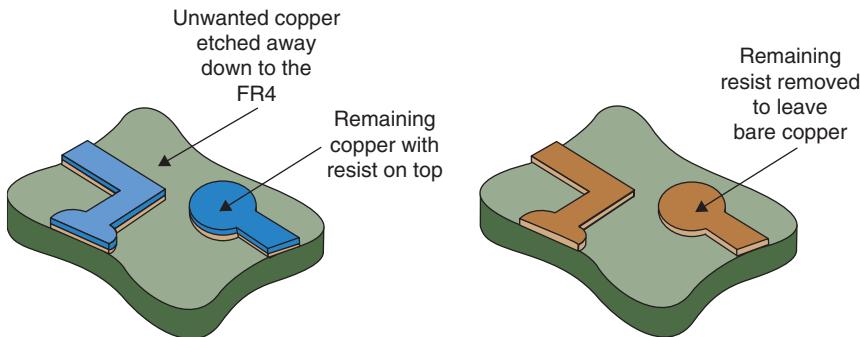
The mask is placed over the surface of the board, which is then exposed to *Ultraviolet* (UV) light. This ionizing radiation passes through the transparent areas of the mask to break down the molecular structure of the resist. After the board has been exposed, it is bathed in an organic solvent, which dissolves the degraded resist. Thus, the pattern on the mask has been transferred to a corresponding pattern in the resist (Figure 18.2).

A process in which ultraviolet light passing through the transparent areas of the mask causes the resist to be degraded is known as a *positive-resist* process; *negative-resist* processes are also available.⁵ The following discussions assume positive-resist processes, unless otherwise noted.

After the unwanted resist has been removed, the board is placed in a bath containing a cocktail based on sulfuric acid, which is agitated and aerated to make it more active. The sulfuric acid dissolves any exposed copper not protected by the resist in a process known as *etching*. The board is then washed to remove the remaining resist. Thus, the pattern in the mask has now been transferred to a corresponding pattern in the copper (Figure 18.3).

This type of process is classed as *subtractive* because the board is first covered with the conductor and any unwanted material is then removed, or *subtracted*.

⁵In a *negative-resist* process the ultraviolet radiation passing through the transparent areas of the mask is used to cure the resist. The remaining uncured areas are then removed using an appropriate solvent. Thus, a mask used in a negative-resist process is the photographic negative of one used in a positive-resist process to achieve the same effect; that is, the transparent areas are now opaque, and vice versa.

**FIGURE 18.3**

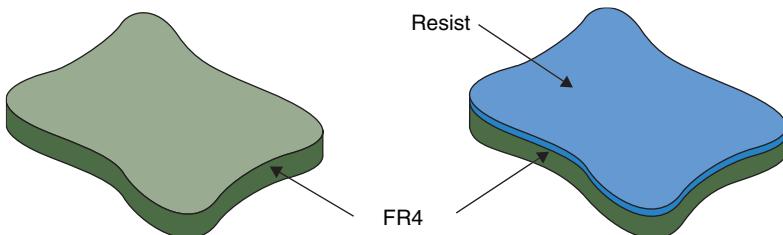
Subtractive process:
Removing the unwanted
copper.

As a point of interest, much of this core technology predates the modern electronics industry. The process of copper etching was well known by the printing industry in the 1800s, and opto-lithographic techniques involving organic resists were used to create printing plates as early as the 1920s. These existing processes were readily adopted by the fledgling electronics industry.

By today's standards, early printed circuit boards had humongously wide tracks. As process technologies improved, it became possible to achieve ever-finer features. By 2002, for example, reasonably high-end boards were using *lines* and *spaces* of 5 mils or 4 mils, where one mil is one one-thousandth of an inch. (In this context, the term *lines* refers to the widths of the tracks, while *spaces* refers to the gaps between adjacent tracks.) At the time of this writing, in 2008, lines and spaces of 4 and 4 mils or 3 and 3 mils are not uncommon, while really high-end boards may be as low as 1 and 1 mil.

ADDITIVE PROCESSES

An *additive process* does not involve any copper foil being bonded to the board. Instead, the coating of organic resist is applied directly to the board's surface (Figure 18.4).

**FIGURE 18.4**

Additive process:
Applying resist to bare
board.

Once again the resist is cured, an optical mask is created, ultraviolet light is passed through the mask to break down the resist, and the board is bathed in an organic solvent to dissolve the degraded resist (Figure 18.5).⁶

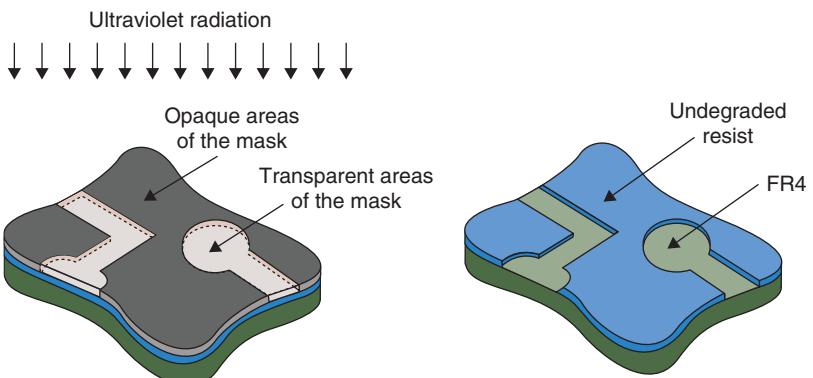


FIGURE 18.5
Additive process:
Degrading the resist
and exposing the FR4.

After the unwanted resist has been removed, the board is placed in a bath containing a cocktail based on copper sulfate where it undergoes a process known as *electroless plating*. Tiny crystals of copper grow on the exposed areas of the board to form copper tracks. The board is then washed in an appropriate solvent to remove the remaining resist (Figure 18.6).

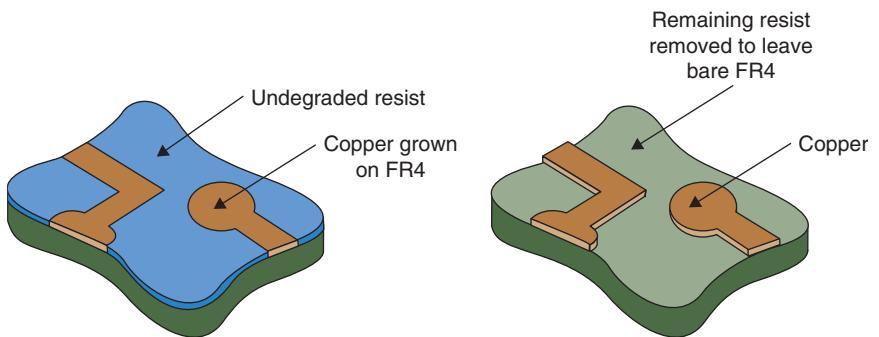


FIGURE 18.6
Additive process:
Adding the desired
copper.

A process of this type is classed as *additive* because the conducting material is only grown on, or *added to*, specific areas of the board. Additive processes are increasing in popularity because they require less processing and result in less wasted material. Additionally, fine tracks can be grown more accurately in additive processes than they can be etched in their subtractive counterparts.

⁶Note that a mask used in an additive process is the photographic negative of one used in a subtractive process to achieve the same effect; that is, the transparent areas are now opaque, and vice versa.

These processes are of particular interest for high-speed designs and microwave applications, in which conductor thicknesses and controlled impedances are critical. Groups of tracks, individual tracks, or portions of tracks can be built up to precise thicknesses by iterating the process multiple times with selective masking.

SINGLE-SIDED BOARDS

It probably comes as no great surprise to hear that *single-sided boards* have tracks on only one side. These tracks, which may be created using either subtractive or additive processes, are terminated with areas of copper known as *pads*. The shape of the pads and other features are, to some extent, dictated by the method used to attach components to the board. Initially, these discussions will assume that the components are to be attached using a technique known as *through-hole*, which is described in more detail below. The pads associated with the through-hole technique are typically circular, and holes are drilled through both the pads and the board using a computer-controlled drilling machine (Figure 18.7).⁷

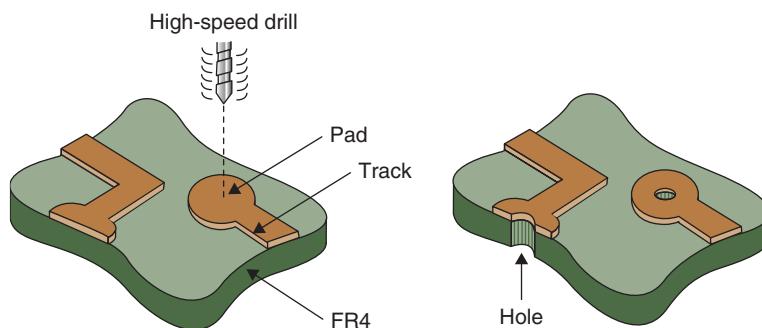
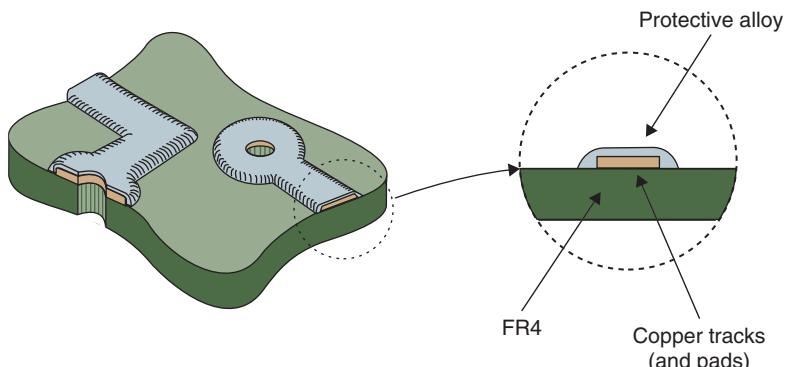


FIGURE 18.7
Single-sided boards:
Drilling the holes.

Once the holes have been drilled, an electroless plating process referred to as *tinning* (in England) or *plating* (in America) is used to coat the tracks and pads with a layer of alloy. This used to be a tin-lead alloy (hence, the name *tinning*) but there's an increasing use of RoHS-compliant materials. This alloy is used to prevent the copper from oxidizing and to provide protection against contamination. The deposited alloy has a rough surface composed of vertical crystals called *spicules*, which, when viewed under a microscope, resemble a bed of nails. To prevent oxygen from reaching the copper through pinholes in the

⁷This is usually referred to as an *NC drilling machine*, where NC stands for *numerically controlled*.

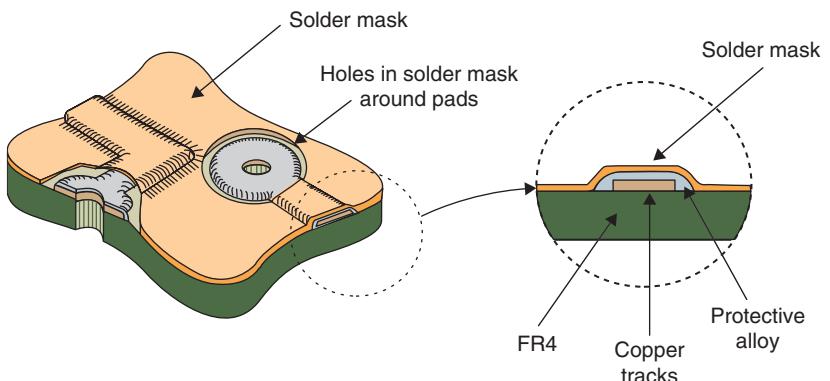
**FIGURE 18.8**

Single-sided boards:
Coating the copper with
a layer of alloy.

alloy, the board is placed in a *reflow oven* where it is heated by either *Infrared* (IR) radiation or hot air. The reflow oven causes the alloy to melt and form a smooth surface (Figure 18.8).

After the board has cooled, a layer known as the *solder mask* is applied to the surface carrying the tracks (the purpose of this layer is discussed in the next section). One common technique is for the solder mask to be screen-printed onto the board. In this case, the screen used to print the mask has patterns that leave the areas around the pads exposed, and the mask is then cured in an oven. In an alternative technique, the solder mask is applied across the entire surface of the board as a film with an adhesive on one side. In this case, a further opto-lithographic stage is used to cure the film using ultraviolet light. The optical mask used in this process contains opaque patterns, which prevent the areas around the pads from being cured; these areas are then removed using an appropriate solvent (Figure 18.9).

Beware! Although there are relatively few core processes used to manufacture circuit boards, there are almost endless variations and techniques. For example, the tracks and pads may be created before the holes are drilled, or vice versa. Similarly,

**FIGURE 18.9**

Single-sided boards:
Adding the solder mask.

the protective alloy may be applied before the solder mask, or vice versa. This latter case, known as *Solder Mask Over Bare Copper* (SMOBC), prevents solder from leaking under the mask when the protective alloy melts during the process of attaching components to the board. Thus, the protective alloy is applied only to any areas of copper that are left exposed by the solder mask, such as the pads at the end of the tracks. As there are so many variations and techniques, these discussions can hope to offer only an overview of the main concepts.

LEAD THROUGH-HOLE (LTH)

Prior to the early 1980s, the majority of integrated circuits were supplied in packages that were attached to a circuit board by inserting their leads through holes drilled in the board. This technique, which is still used to some extent, is known as *Lead Through-Hole* (LTH), *Plated Through-Hole* (PTH), or, more concisely, *through-hole*. In the case of a single-sided board, any components that are attached to the board in this fashion are mounted on the opposite side to the tracks. This means that any masks used to form the tracks are actually created as mirror-images of the required patterns (Figure 18.10).

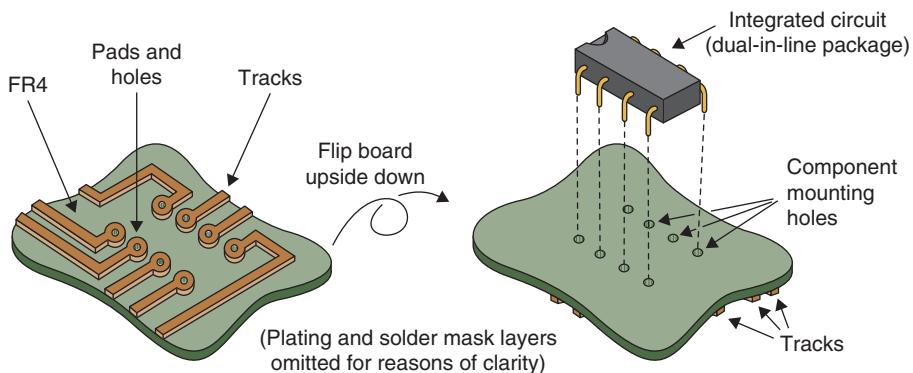


FIGURE 18.10
Populating the board:
Lead through-hole
(LTH).

The act of attaching components is known as *populating the board*, and the area of the board occupied by a component is known as its *footprint*. Early manufacturing processes required the boards to be populated by hand, but modern processes make use of automatic insertion machines. Similarly, component leads used to be hand-soldered to the pads, but modern processes employ automatic techniques, such as *wave soldering*, *reflow ovens*, or *vapor-phase soldering* as discussed in the following topics.

WAVE SOLDERING

Wave soldering and the *Lead Through-Hole* (LTH) population technique go hand-in-hand. A wave soldering machine is based on a tank containing hot, liquid

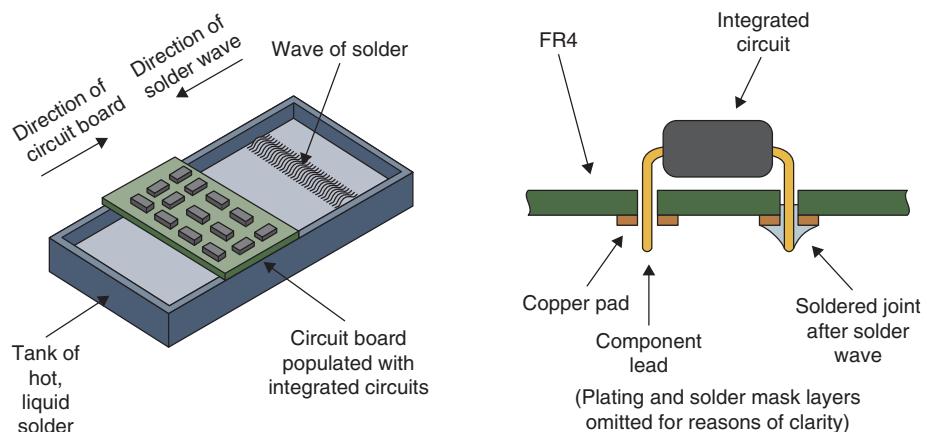


FIGURE 18.11
Wave soldering.

solder (Figure 18.11). The machine creates a wave (actually, a large ripple) of solder that travels across the surface of the tank. Circuit boards populated using the through-hole technique are passed over the machine on a conveyer belt. The system is carefully controlled and synchronized such that the solder wave brushes across the bottom of the board only once.

The solder mask (which was introduced earlier in this chapter) prevents the solder from sticking to anything except the exposed pads and component leads. Because the solder is restricted to the area of the pads, surface tension causes it to form good joints between the pads and the component leads. Additionally, capillary action causes the solder to be drawn up the hole, thereby forming reliable, low-resistance connections. If the solder mask were omitted, the solder would run down the tracks away from the component leads. In addition to forming bad joints, the amount of heat absorbed by the tracks would cause them to separate from the board (this is not considered to be a good thing to happen).

SURFACE MOUNT TECHNOLOGY (SMT)

In the early 1980s, new techniques for packaging integrated circuits and populating boards began to appear. The most popular is *Surface Mount Technology* (SMT), in which component leads are attached directly to pads on the surface of the board. Components with packages and lead shapes suitable for this technology are known as *Surface Mount Devices* (SMDs). One example of a package that achieves a high lead count in a small area is the *Quad Flat Pack* (QFP), in which leads are present on all four sides of a thin, square package.⁸

⁸Other packaging styles such as *Pad Grid Arrays* (PGAs) and *Ball Grid Arrays* (BGAs)—which are also suitable for surface mount technology—are introduced in more detail in *Chapter 20: Advanced Packaging Techniques*.

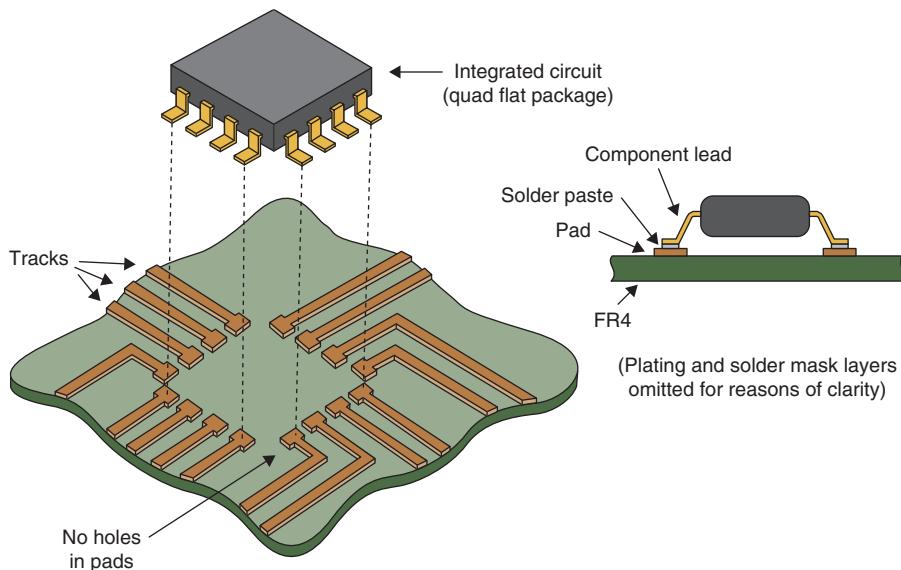


FIGURE 18.12
Populating the board:
Surface mount
technology (SMT).

Boards populated with surface mount devices are fabricated in much the same way as their through-hole equivalents (except that the pads used for attaching the components are typically square or rectangular and do not have any holes drilled through them).⁹ However, the processes begin to diverge after the solder mask and plating layers have been applied. A layer of solder paste is screen-printed onto the pads, and the board is populated by an automatic *pick-and-place machine*, which pushes the component leads into the paste (Figure 18.12).

Thus, in the case of a single-sided board, the components would be mounted on the same side as the tracks. When all of the components have been attached, the solder paste is melted to form good conducting bonds between their leads and the board's pads. The solder paste can be melted by placing the board in a *reflow oven*, where it is heated by *Infrared* (IR) radiation or by hot air. Alternatively, the solder may be melted using *vapor-phase soldering*, in which the board is lowered into the vapor-cloud above a tank containing boiling hydrocarbons.¹⁰

Surface mount technology is well suited to automated processes. Because the components are attached directly to the surface of the board, they can be constructed with leads that are finer and more closely spaced than their through-hole

⁹Actually, the pads may have holes, in the case of the HDI/microvia technologies discussed later in this chapter.

¹⁰Vapor-phase soldering is becoming increasingly less popular due to environmental concerns.

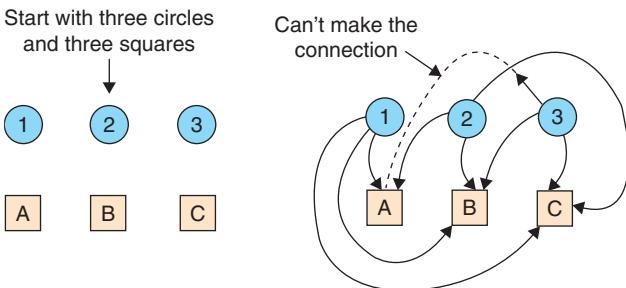


FIGURE 18.13
Circles and squares game.

equivalents. This results in smaller and lighter packages which can be mounted closer together and occupy less of the board's surface area, which is referred to as *real estate*. This, in turn, results in smaller, lighter, and faster circuit boards. The fact that the components do not require holes for their leads is also advantageous, because drilling holes is a time-consuming and expensive process. Additionally, if any holes are required to make connections through the board (as discussed below), they can be made much smaller because they do not have to accommodate component leads.

DOUBLE-SIDED BOARDS

There is a simple game played by children all over the world. The game commences by drawing three circles and three squares on a piece of paper, and then trying to connect each circle to each square without any of the connecting lines crossing each other (Figure 18.13).

Children can devote hours to this game, much to the delight of their parents.¹¹ Unfortunately, there is no solution, and one circle-square pair will always remain unconnected. This simple example illustrates a major problem with single-sided boards, which may have to support large numbers of component leads and tracks. If any of the tracks cross, an undesired electrical connection will be made and the circuit will not function as desired. One solution to this dilemma is to use wire links called *jumpers* (Figure 18.14).

Unfortunately, the act of inserting a jumper is as expensive as for any other component. At some point it becomes more advantageous to employ a *double-sided board*, which has tracks on both sides.

Initially, the construction of a double-sided board is similar to that for a single-sided board. Assuming a subtractive process, copper foil is bonded to

¹¹Hang on a moment, so that's why my dad taught me this game!

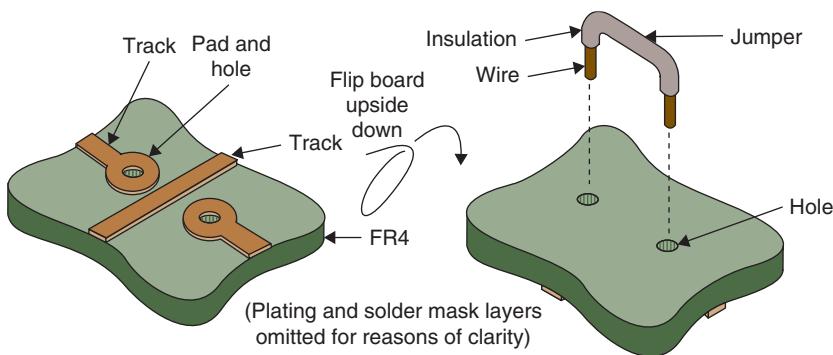


FIGURE 18.14
Using jumpers.

both sides of the board, and then organic resist is applied to both surfaces and cured. Separate masks are created for each side of the board, and ultraviolet light is applied to both sides. The ultraviolet radiation that is allowed to pass through the masks degrades the resist, which is then removed using an organic solvent. Any exposed copper that is not protected by resist is etched, the remaining resist is removed, and holes are drilled. However, a double-sided board now requires an additional step. After the holes have been drilled, a plating process is used to line them with copper (Figure 18.15).

Instead of relying on jumpers, a track can now pass from one side of the board to the other by means of these copper-plated holes, which are known as *vias*.^{12,13} The tracks on one side of the board usually favor the Y-axis (North-South), while

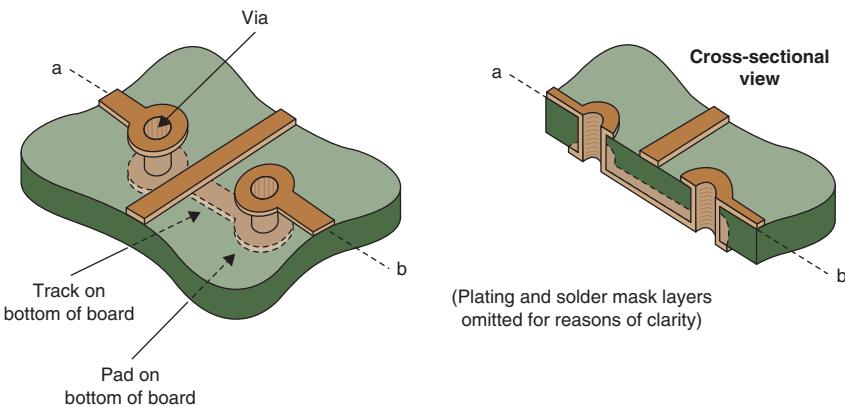


FIGURE 18.15
Double-sided boards:
Creating vias.

¹²The term *via* is taken to mean a conducting path linking two or more conducting layers, but does not include a hole accommodating a component lead (see also the *Holes Versus Vias* topic).

¹³There are a number of alternative techniques that may be used to create circuit board vias. By default, however, the term is typically understood to refer to holes plated with copper, as described here.

the tracks on the other side favor the X-axis (East-West). The inside of the vias and the tracks on both sides of the board are plated with protective alloy, and solder masks are applied to both surfaces (or vice versa in the case of the SMOBC-based processes, which were introduced earlier).

Some double-sided boards are populated with through-hole or surface mount devices on only one side. Some boards may be populated with through-hole devices on one side and surface mount devices on the other. And some boards may have surface mount devices attached to both sides. This latter case is of particular interest, because surface mount devices do not require holes to be drilled through the pads used to attach their leads (they typically have separate fan-out vias as discussed below). Thus, when using surface mount technology, it is possible to place two devices directly facing each other on opposite sides of the board without making any connections between them.

Having said this, in certain circumstances it may be advantageous to form connections between surface mount devices directly facing each other on opposite sides of the board. The reason for this is that a through-board connection can be substantially shorter than an equivalent connection between adjacent devices on the same side of the board. Thus, this technique may be of use for applications such as high-speed data buses, because shorter connections result in faster signals.

HOLES VERSUS VIAS

Manufacturers of circuit boards are very particular about the terminology they use, and woe betide anyone caught mistakenly referring to a *hole* as a *via*, or vice versa. Figure 18.16 should serve to alleviate some of the confusion.

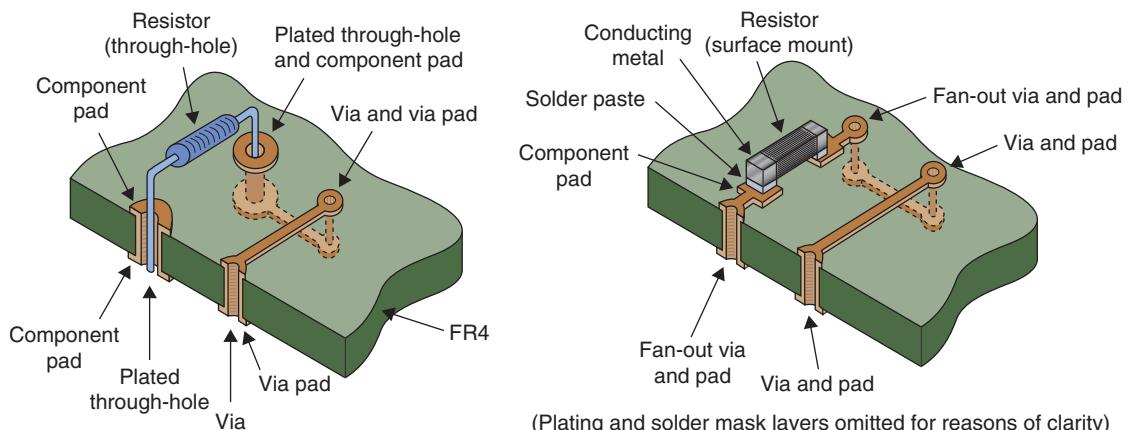


FIGURE 18.16
Holes versus vias.

In the case of a single-sided board (as illustrated in Figures 18.7 and 18.14), a hole that is used to accommodate a through-hole component lead is simply referred to as a *hole*. By comparison, in the case of double-sided boards (or multilayer boards as discussed below), a hole that is used to accommodate a through-hole component lead is plated with copper, and is referred to as a *plated through-hole*. Additionally, a hole that is only used to link two or more conducting layers, but does not accommodate a component lead, is referred to as a *via* (or, for those purists among us, an *interstitial via*). The qualification attached to this latter case is important, because even if a hole that is used to accommodate a component lead is also used to link two or more conducting layers, it is still referred to as a *plated through-hole* and not a *via*. (Phew!)

Because vias do not have to accommodate component leads, they can be created with smaller diameters than plated through-holes, thereby occupying less of the board's real estate. To provide a sense of scale, the diameters of plated through-holes and their associated pads are usually on the order of 24 mils (0.6 mm) and 48 mils (1.2 mm), respectively, while the diameters of vias and via pads are typically 12 mils (0.3 mm) and 24 mils (0.6 mm), respectively.

Finally, in the case of surface mount devices attached to double-sided boards (or multilayer boards as discussed below), each component pad is usually connected by a short length of track to a via, known as a *fan-out via*,¹⁴ which forms a link to other conducting layers (an example of a fan-out via is shown in Figure 18.16). However, if this track exceeds a certain length (it could meander all the way around the board), an otherwise identical via at the end would be simply referred to as a via. Unfortunately, there is no standard length of track that differentiates a fan-out via from a standard via, and any such classification depends solely on the in-house design rules employed by the designer and board manufacturer.

MULTILAYER BOARDS

It is not unheard of for a circuit board to support thousands of components and tens of thousands of tracks and vias. Double-sided boards can support a higher population density than single-sided boards, but there quickly comes a point when even a double-sided board reaches its limits. A common limiting factor is lack of space for the necessary number of vias. In order to overcome this limitation, designers may move onwards and upwards to *multilayer boards*.

¹⁴Some folks attempt to differentiate vias that fall inside the device's footprint (under the body of the device) from vias that fall outside the device's footprint by referring to the former as *fan-in vias*, but this is not a widely used term.

A multilayer board is constructed from a number of single-sided or double-sided *sub-boards*.¹⁵ The individual sub-boards can be very thin, and multilayer boards with four or six conducting layers are usually around the same thickness as a standard double-sided board. Multilayer boards may be constructed using a double-sided sub-board at the center with single-sided sub-boards added to each side [Figure 18.17(a)].¹⁶ Alternatively, they may be constructed using only double-sided sub-boards separated by nonconducting layers of semi-cured FR4 known as *prepreg* [Figure 18.17(b)].

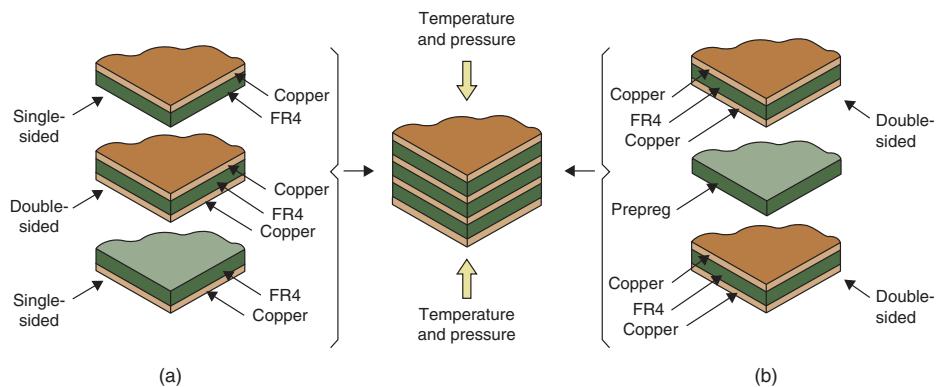


FIGURE 18.17
Multilayer boards:
Alternative structures.

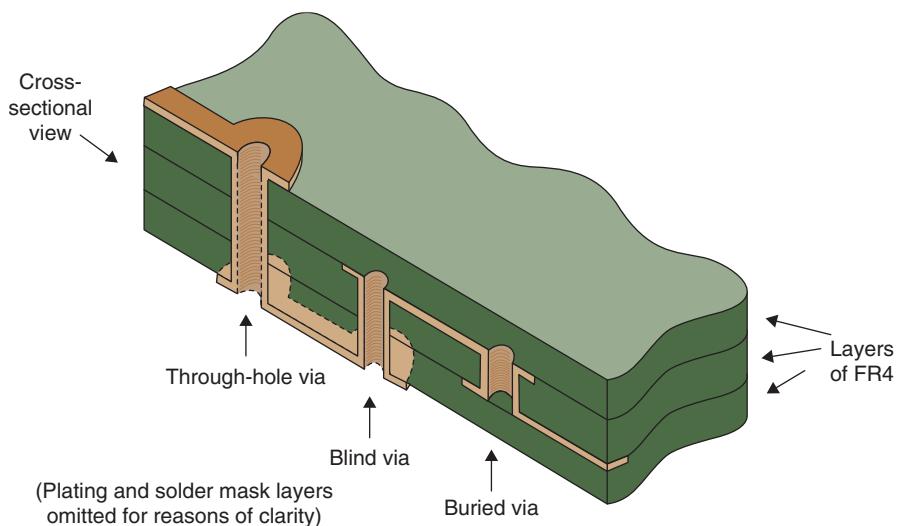
After all of the layers have been etched to form tracks and pads, the sub-boards and prepreg are bonded together using a combination of temperature and pressure. This process also serves to fully cure the prepreg. Boards with four conducting layers are typical for designs intended for large production runs. The majority of multilayer boards have less than ten conducting layers, but boards with twenty-four conducting layers or more are not outrageously uncommon, and some specialized boards like *backplanes* (as discussed later in this chapter) may have 60 layers or more!

THROUGH-HOLE, BLIND, AND BURIED VIAS

To overcome the problem of limited space, multilayer boards may make use of *through-hole*, *blind*, and *buried vias*. A through-hole via passes all the way through the board, a blind via is visible from only one side of the board, and a buried via is used to link internal layers and is not visible from either side of the board (Figure 18.18).

¹⁵The term *sub-board* is not an industry standard, and it is used in these discussions only to distinguish the individual layers from the completed board.

¹⁶This technique is usually reserved for boards that carry only four conducting layers.

**FIGURE 18.18**

Through-hole versus blind versus buried vias.

Unfortunately, although they help to overcome the problem of limited space, blind and buried vias significantly increase the complexity of the manufacturing process. When these vias are only used to link both sides of a single sub-board, that board must be drilled individually and a plating process used to line its vias with copper. Similarly, in the case of vias that pass through a number of sub-boards, those boards must be bonded together, drilled, and plated as a group. Finally, after all of the sub-boards have been bonded together, any holes that are required to form plated through-holes and vias are drilled and plated. Blind and buried vias can greatly increase the number of tracks that a board can support but, in addition to increasing costs and fabrication times, they can also make it an absolute swine to test.

POWER AND GROUND PLANES

The layers carrying tracks are known as the *signal layers*. In a multilayer board, the signal layers are typically organized so that each pair of adjacent layers favors the Y-axis (North-South) and the X-axis (East-West), respectively. Additionally, two or more conducting layers are typically set aside to be used as *power* and *ground planes*. The power and ground planes usually occupy the central layers, but certain applications have them on the board's outer surfaces. This latter technique introduces a number of problems, but it also increases the board's protection from external sources of noise, such as electromagnetic radiation.

Unlike the signal layers, the bulk of the copper on the power and ground planes remains untouched. The copper on these layers is etched away only in

those areas where it is not required to make a connection. As a simple example, consider a through-hole device with eight leads. Assume that leads 4 and 8 connect to the ground and power planes, respectively, while the remaining leads are connected into various signal layers (Figure 18.19).

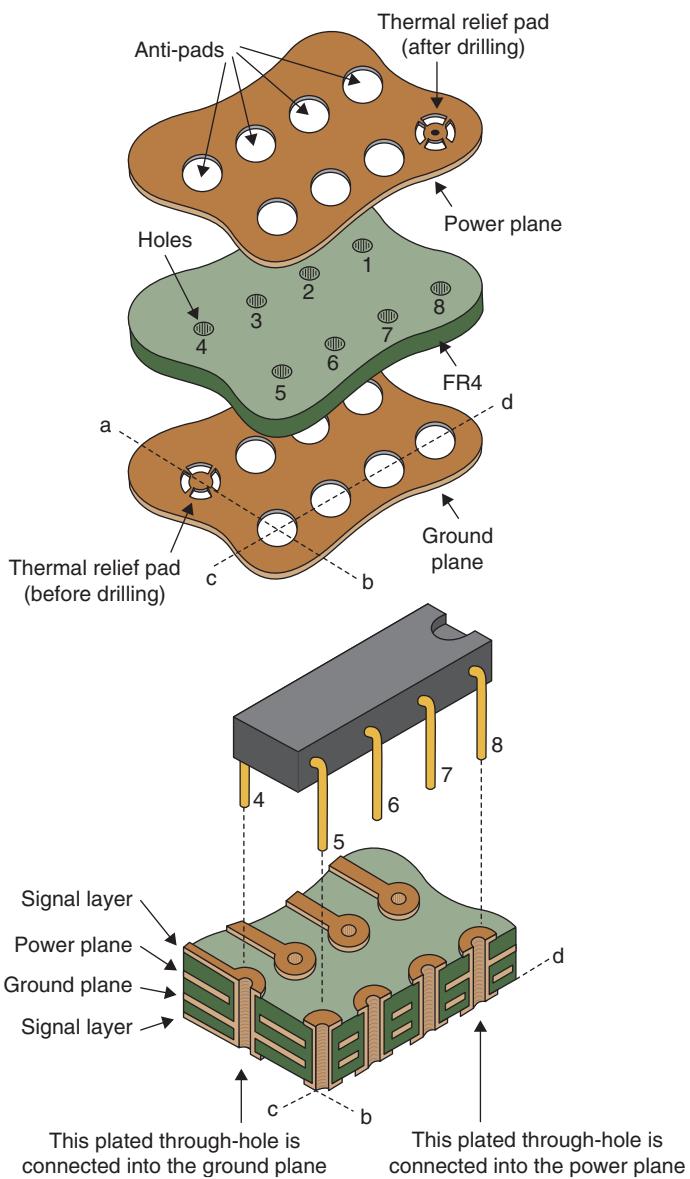


FIGURE 18.19
Power and ground planes; thermal relief and anti-pads.

For the sake of simplicity, the exploded view in Figure 18.19 shows only the central sub-board carrying the power planes; the sub-boards carrying the signal layers would be bonded to either side. Also note that the holes shown in the prepreg in the exploded view would not be drilled and plated until all of the sub-boards had been bonded together.

In the case of component leads 1, 2, 3, 5, 6, and 7, both the power and ground planes have copper removed around the holes. These etched-away areas, which are referred to as *anti-pads*, are used to prevent connections to the planes when the holes are plated. Similarly, the power plane has an anti-pad associated with lead 4 (the ground lead), and the ground plane has an anti-pad associated with lead 8 (the power lead).

The power plane has a special pattern etched around the hole associated with lead 8 (the power lead), and a similar pattern is present on the ground plane around the hole associated with lead 4 (the ground lead). These patterns, which are referred to as *thermal relief pads*,¹⁷ are used to make electrical connections to the power and ground planes. The spokes in the thermal relief pads are large enough to allow sufficient current to flow, but not so large that they will conduct too much heat.

Thermal relief pads are necessary to prevent excessive heat from being absorbed into the ground and power planes when the board is being soldered.¹⁸ When the solder is applied, a surface-tension effect known as capillary action sucks it up the vias and plated through-holes. The solder must be drawn all the way through to form reliable, low-resistance connections. The amount of copper contained in the power and ground planes can cause problems because it causes them to act as *thermal heat sinks*. The use of thermal relief pads ensures good electrical connections, while greatly reducing heat absorption. If the thermal relief pads were not present, the power and ground planes would absorb too much heat too quickly. This would cause the solder to cool and form plugs in the vias resulting in unreliable, high-resistance connections. Additionally, in the case of wave soldering, so much heat would be absorbed by the power and ground planes that all of the layers forming the board could separate, in a process known as *delamination*.

¹⁷The pattern of a thermal relief pad is often referred to as a *wagon wheel*, because the links to the plated-through hole or via look like the spokes of a wheel. Depending on a number of factors, a thermal relief pad may have anywhere from one to four spokes.

¹⁸For future reference, the term *pad-stack* refers to any pads, anti-pads, and thermal relief pads associated with a particular via or plated-through hole as it passes through the board.

A special flavor of multilayer boards known as *Padcap* (or *Pads-Only-Outer-Layers*) are sometimes used for high-reliability military applications. Padcap boards are distinguished by the fact that the outer surfaces of the board only carry pads, while any tracks are exclusively created on inner layers and connected to the pads by vias. Padcap technology offers a high degree of protection in hostile environments because all of the tracks are inside the board.

HIGH DENSITY INTERCONNECT (HDI) AND MICROVIA TECHNOLOGIES

One exciting (and relatively recent) circuit board development is that of *High Density Interconnect* (HDI). This features the use of extremely thin tracks and tiny vias called *microvias*. HDI officially refers to any vias and via pads with diameters of 6 mils (0.15 mm) and 12 mils (0.3 mm)—or smaller—respectively. By 2002, boards using 4-mil and 3-mil diameter microvias were reasonably common; by 2008 folks were using 2-mil and 1-mil microvias.

In a typical implementation, one or two microvia layers are added to the outer faces of a standard multiplayer board; thus, the term *buildup technology* is also commonly used when referring to high density interconnect. In fact, the terms *HDI*, *microvia*, and *buildup technology* are often used interchangeably.

Microvias—which are actually blind vias that just pass through one or more of the buildup layers on the outer faces of the main board—may be created using a variety of techniques. One common method is to use a laser, which can “drill” 20,000+ microvias per second.

The reason HDI with microvias is so necessary is largely tied to recent advances in device packaging technologies. It’s now possible to get devices with 2000 pins (or pads or leads or connections or whatever you want to call them), and packages with more pins are on the way. These pins are presented as an array across the bottom of the device. The pin pitch (the distance between adjacent pins) has shrunk to the extent that it simply isn’t possible to connect the package to a board using conventional via diameters and line widths [there just isn’t enough space to squeeze in all of the fan-out vias and route (lay down the paths of) all of the tracks]. The use of microvia technology alleviates this problem, making it possible to implement a *breakout pattern* that brings tracks from the pins outside the device’s footprint and fans them out so that they can be connected to the board’s regular signal layers and power planes.

Although using microvia technology isn’t cheap, *per se*, it can actually end up being very cost-effective. In one example of which the author is aware, the use

of microvias enabled an 18-layer board to be reduced to 10 layers, made the board smaller, and halved the total production cost.

BACKPLANES AND MOTHERBOARDS

Another flavor of multilayer boards, known as *backplanes*, have their own unique design constraints and form a subject in their own right. Backplanes usually have a number of connectors into which standard circuit boards are plugged (Figure 18.20).

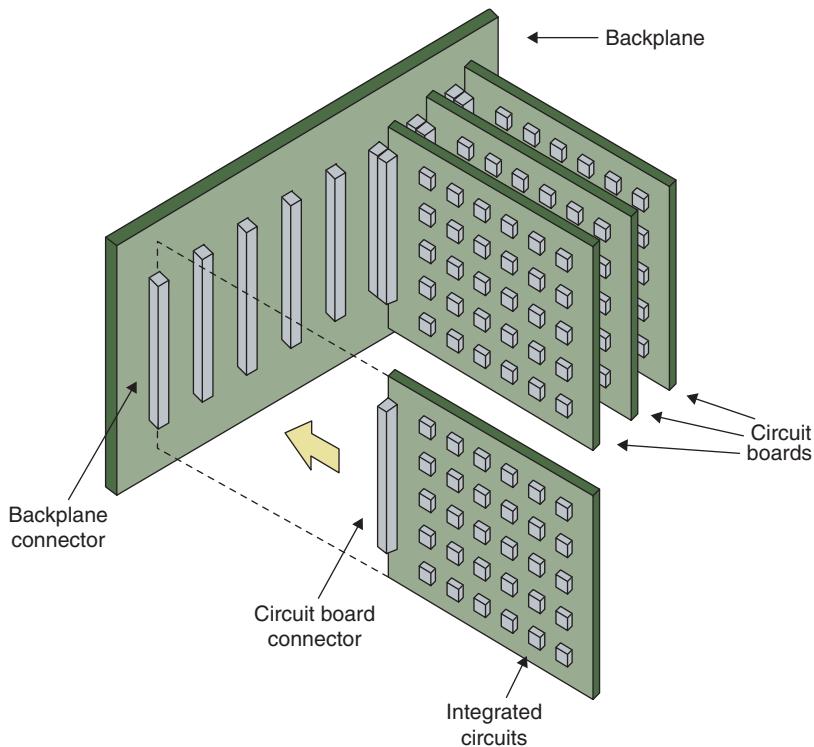


FIGURE 18.20

Backplanes.

Backplanes typically do not carry any active components, such as integrated circuits, but they often carry passive components, such as resistors and capacitors. If a backplane does contain active components, then it is usually referred to as a *motherboard*. In this case, the boards plugged into it are referred to as *daughter boards* or *daughter cards*.

Because of the weight that they have to support, backplanes for large systems can be 1 cm thick or more. Their conducting layers have thicker copper plating

than standard boards and the spaces between adjacent tracks are wider to reduce noise caused by inductive effects.

Backplanes may also support a lot of hardware, in the form of bolts, earth straps, and power cables. It is not unusual for a backplane to have multiple power planes, such as +5 volts, -5 volts, +12 volts, -12 volts, +24 volts, -24 volts, and so on. Each power plane typically has an independent ground plane associated with it to increase noise immunity. So, our hypothetical backplane would require 12 conducting layers for the power and ground planes alone. Additionally, systems containing both analog and digital circuits often require independent power and ground planes for purposes of noise immunity. Backplanes also require excellent thermal tolerance, because some of the more heroic systems can consume upwards of 200 amps and generate more heat than a rampaging herd of large electric radiators.

CONDUCTIVE INK TECHNOLOGY

The underlying concept of *conductive ink technology* is relatively simple. Tracks are screen-printed onto a bare board using a conducting ink, which is then cured in an oven. Next, a dielectric, or insulating, layer is screen-printed over the top of the tracks. The screen used to print the dielectric layer is patterned so as to leave holes over selected pads on the signal layer. After the dielectric layer has been cured, the cycle is repeated to build a number of signal layers separated by dielectric layers. The holes patterned into the dielectric layers are used to form vias between the signal layers (Figure 18.21).

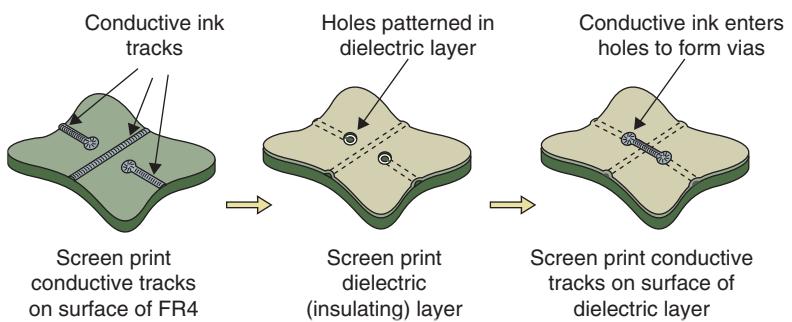


FIGURE 18.21
Conductive ink technology.

Finally, plated through-holes and vias can be created, and components can be mounted, using the standard processes described previously. The apparent simplicity of the conductive ink technique hides an underlying sophistication in

materials technology. Early inks were formed from resin pastes loaded with silver or copper powder. These inks required high firing temperatures to boil off the paste and melt the powder to form conducting tracks. Additionally, the end product was not comparable to copper foil for adhesion, conductivity, or solderability.

In the early 1990s, new inks were developed based on pastes containing a mixture of two metal or alloy powders. One powder has a relatively low melting point, while the other has a relatively high melting point. When the board is cured in a reflow oven at temperatures as low as 200°C, a process called *sintering* occurs between the two powders, resulting in an alloy with a high melting point and good conductivity.

Conductive ink technology has not yet achieved track widths as fine as traditional circuit board processes, but it does have a number of attractive features, not the least of which is that it uses commonly available screen-printing equipment. Modern inks have electrical conductivity comparable to copper and they work well with both wave soldering and reflow soldering techniques. Additionally, these processes generate less waste and are more cost-effective and efficient than the plating and etching of copper tracks.

CHIP-ON-BOARD (COB)

Chip-on-Board (COB) is a relatively modern process that only began to gain widespread recognition in the early 1990s, but which is now accepted as a common and cost-effective die attachment technique. As the name implies, unpackaged integrated circuits are mounted directly onto the surface of the board. The integrated circuits are mechanically and electrically connected using similar *wire bonding*, *tape-automated bonding*, and *flip-chip* techniques to those used for *hybrids* and *System-in-Package* (SiP) assemblies.¹⁹ The final step is *encapsulation*, in which the integrated circuits and their connections are covered with “globs” of epoxy resin, plastic, or some other material; these “globs” are then cured to form hard protective covers (Figure 18.22).²⁰

There are a number of variations of *Chip-on-Board*. For example, the designer may wish to maintain an extremely low profile for applications such as intelligent credit cards. One way to achieve this is to form cavities in the board into which the

¹⁹Hybrid and System-in-Package (SiP) concepts are introduced in Chapters 19: *Hybrids*, and 20: *Advanced Packaging Techniques*, respectively.

²⁰In addition to mechanical and environmental protection, the encapsulating material is also used to block out light.

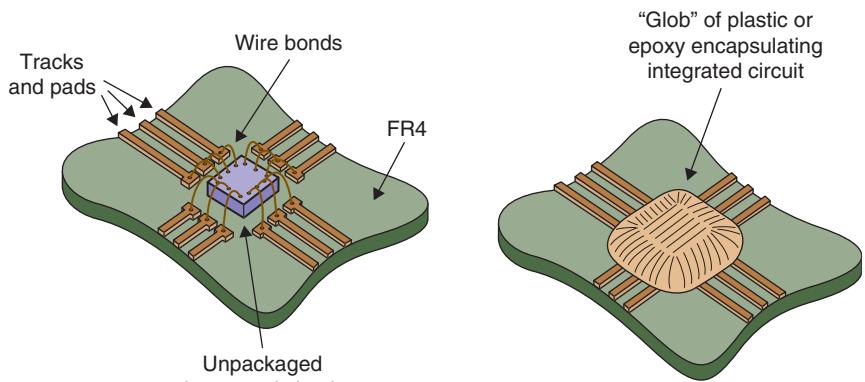


FIGURE 18.22
Chip-on-Board (COB).

integrated circuits are inserted. Compared to surface mount technology, and especially to through-hole technology, chip-on-board offers significant reductions in size, area, and weight. Additionally, this technique boosts performance because the chips can be mounted closer together, resulting in shorter tracks and faster signals.

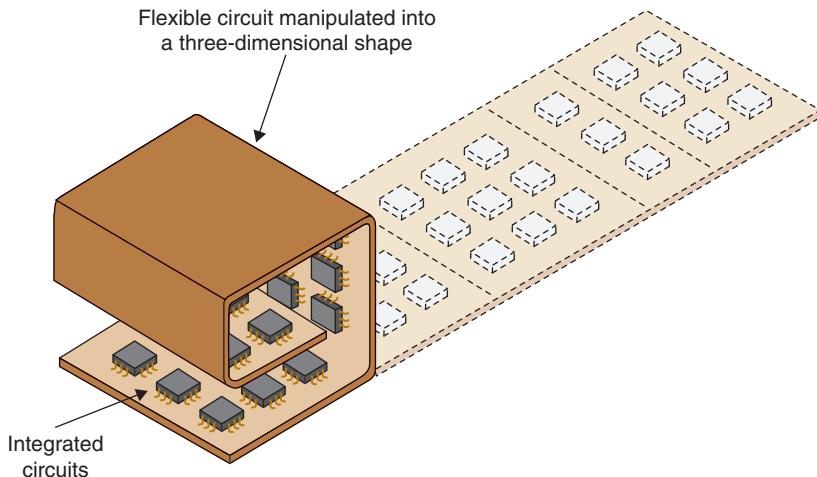
FLEXIBLE PRINTED CIRCUITS (FPCs)

Last, but not least, are *Flexible Printed Circuits* (FPCs), often abbreviated to *flex*, in which patterns of conducting tracks are printed onto flexible materials. Surprisingly, flexible circuits are not a recent innovation: they can trace their ancestry back to 1904 when conductive inks were printed on linen paper. However, modern flexible circuits are made predominantly from organic materials such as *polyesters* and *polyimides*. These base layers can be thinner than a human hair, yet still withstand temperatures up to approximately 700°C without decomposing.

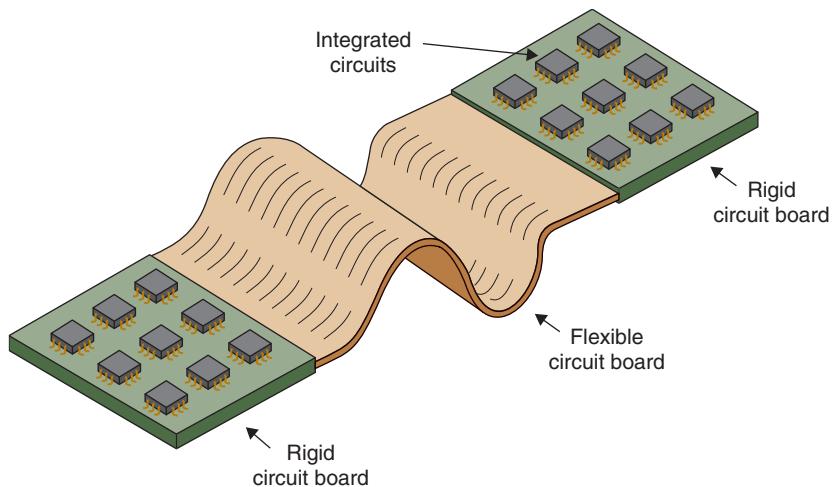
There are many variants of flexible circuits, not the least being *flexing*, or *dynamic flex*, and *nonflexing*, or *static flex*. Dynamic flex is used in applications that are required to undergo constant flexing, such as ribbon cables in printers, while static flex can be manipulated into permanent three-dimensional shapes for applications such as calculators and high-tech cameras, requiring efficient use of volume and not just area (Figure 18.23).

As well as single-sided flex, there are also double-sided and multilayer variants. Additionally, unpackaged integrated circuits can be mounted directly onto the surface of the flexible circuits in a similar manner to chip-on-board discussed above. However, in this case, the process is referred to as *Chip-on-Flex* (COF).

A common manifestation of flex technology is found in hybrid constructions known as rigid flex, which combine standard rigid circuit boards with flexible printed circuits (Figure 18.24).

**FIGURE 18.23**

Flexible printed circuits: Static flex.

**FIGURE 18.24**

Flexible printed circuits: Rigid flex.

In this example, the flexible printed circuit linking two standard (rigid) boards eliminates the need for connectors on the boards (which would have to be linked by cables), thereby reducing the component count, weight, and susceptibility to vibration of the circuit, and greatly increasing its reliability.

The use of flexible circuits is increasing for a number of reasons. These include the ongoing development of miniaturized, lightweight, portable electronic

systems such as cellular phones, and the maturing of surface mount technology, which has been described as the ideal packaging technology for flexible circuits. Additionally, flexible circuits are amenable to being produced in the form of a continuous roll, which can offer significant manufacturing advantages for large production runs.

CHAPTER 19

Hybrids

THE OFFSPRING RESULTING FROM CROSSBREEDING

The word *hybrid* is defined as “*the offspring resulting from crossbreeding.*” Many would agree that this is an apt description for the species of electronic entities also known as *hybrids*, which combine esoteric mixtures of interconnection and packaging technologies. In electronic terms, a hybrid consists of a collection of components mounted on a single insulating base layer called the *substrate*. A typical hybrid may contain a number of packaged or unpackaged integrated circuits and a variety of discrete components such as resistors, capacitors, and inductors, all attached directly to the substrate. Connections between the components are formed on the surface of the substrate; also, some components such as resistors and inductors may be fabricated directly onto the surface of the substrate.

277

HYBRID SUBSTRATES

Hybrid substrates are predominantly formed from alumina (aluminum oxide) or similar ceramic materials. Ceramics have many valuable properties, which have been recognized since the Chinese first created their superb porcelains during the Ming dynasty. In addition to being cheap, light, rugged, and well-understood, ceramics have a variety of characteristics that make them particularly well-suited to electronic applications. They are nonporous and do not absorb moisture, they can be extremely tough,¹ they have very good *lateral thermal conductivity*, and their *coefficient of thermal expansion* is close to that of silicon.

¹As examples of their toughness, ceramics can be used to create artificial bone joints and to line the faces of golf clubs. In fact, during the Cold War, Glock developed a handgun for espionage purposes that was fabricated almost completely out of ceramics (so it wouldn’t trigger metal detectors at airports).

Good lateral thermal conductivity means that heat generated by the components can be conducted horizontally across the substrate, and out through its leads. The coefficient of thermal expansion defines the amount a material expands and contracts due to changes in temperature. If materials with different coefficients are bonded together, changes in temperature will cause shear forces at the interface between them. Because silicon and ceramic have similar coefficients of thermal expansion, they expand and contract at the same rate. This is particularly relevant when unpackaged silicon chips are bonded directly to the hybrid's substrate, because it helps to ensure that a change in temperature will not result in the chips leaping off the substrate.

Hybrid substrates are usually created by placing ceramic powder in a mold and firing it at high temperatures. The resulting substrates have very smooth surfaces and are flat, without any significant curvature. Hybrid substrates typically range in size from 2.5 cm × 2.5 cm to 10 cm × 15 cm and are in the order of 0.8 mm thick.

Some hybrids require numbers of small holes between the top and bottom surfaces of the substrate. These holes will eventually be plated to form conducting paths between the two surfaces, at which time they start to be called *vias*. Additionally, depending on the packaging technology being used, slightly larger holes may be required to accommodate the hybrid's leads. One technique for forming these vias and lead-holes is to introduce tiny pillar structures into the mold. The ceramic powder flows around the pillars and, when the mold (including the pillars) is removed after firing, the substrate contains holes corresponding to the pillars. In the early days, there was no cost-effective method for drilling vias through a ceramic substrate after it had been fired.² However, developments in laser technology made it possible to punch holes through fired substrates using laser beams.

While the majority of hybrid substrates are ceramic, a wide variety of other materials may also be employed. These include glass, small FR4 circuit boards (laminated substrates), and even cardboard. The latter may have appeared among your Christmas presents, embedded in a pair of socks that play an annoying tune when you squeeze them.³

THE THICK-FILM PROCESS

The two most common techniques used to create tracks and components on the surface of hybrid substrates are known as the *thick-film* and *thin-film* processes.

²A process not dissimilar to attempting to bore holes through a dinner plate.

³Thank you, Auntie Barbara, I wear them all the time!

The thick-film process is based on screen-printing, an ancient art whose invention is usually attributed to the Chinese around 3000 BC.

Creating Tracks

An optical mask is created carrying a pattern formed by areas that are either transparent or opaque to ultraviolet frequencies. Next, an extremely fine steel mesh the same size as the hybrid substrate is coated with a layer of photoresistive emulsion (Figure 19.1).

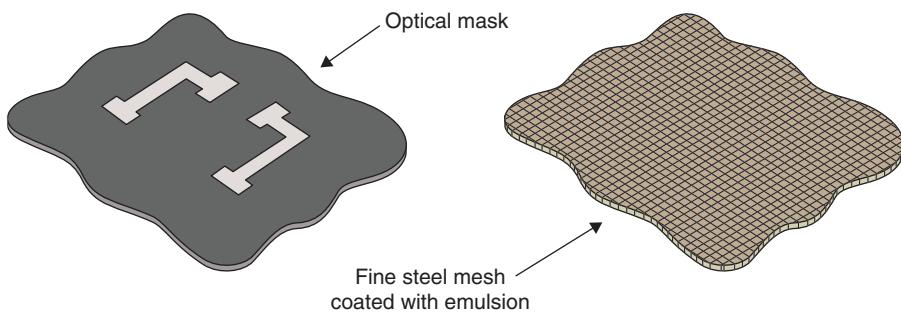
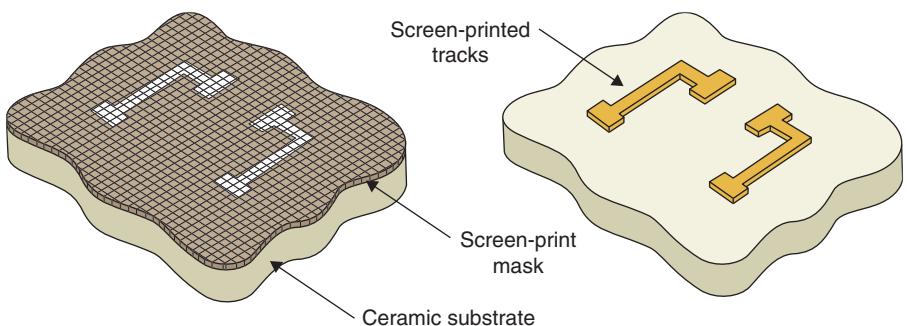


FIGURE 19.1
Thick-film: Optical mask and emulsion-coated fine steel mesh.

Note that the simple patterns shown in these diagrams are used for clarity. In practice, such a mask may contain hundreds or thousands of fine lines and geometric shapes. Also note that these figures show a magnified view of a very small portion of the entire substrate.

The emulsion-coated mesh is first dried, then baked, and then exposed to ultraviolet radiation passed through the optical mask. The ionizing radiation passes through the transparent areas of the optical mask into the emulsion where it breaks down the molecular structure of the resist. The mesh is then bathed in an appropriate solvent to dissolve the degraded resist. Thus, the pattern on the optical mask has been transferred to a corresponding pattern in the resist. The steel mesh with the patterned resist forms a screen-print mask, through which a paste containing metal and glass particles suspended in a solvent is applied to the surface of the substrate (Figure 19.2).

The metal particles suspended in the paste are usually those of a noble metal such as gold, silver, or platinum, or an alloy of such metals as platinium-silver (platinum and silver). When the substrate is dried, the solvent evaporates, leaving the particles of glass and metal forming tracks on the surface of the substrate. Thick-film tracks are in the order of 0.01 mm thick. The widths of

**FIGURE 19.2**

Thick-film: Tracks screen-printed onto the substrate.

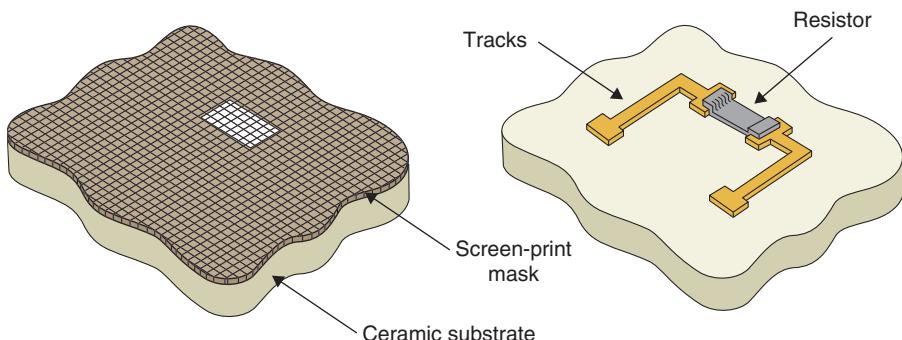
the tracks and the spaces between adjacent tracks are normally in the order of 0.25 mm, but can be as low as 0.1 mm or even finer in a leading-edge process.

Multiple layers of tracks can be printed onto the surface, each requiring the creation of a unique screen-print mask. A pattern of insulating material called a *dielectric layer* must be inserted between each pair of tracking layers to keep them separated. The dielectric patterns are formed from a paste containing only glass particles suspended in a solvent. Each dielectric layer requires its own screen-print mask and is applied using a process identical to that used for the tracking layers. Holes are included in the dielectric patterns where it is required that tracks from adjacent tracking layers be connected to each other. Typical hybrids employ four tracking layers, commercial applications are usually limited to between seven and nine tracking layers, and a practical limit for current process technologies is around fourteen tracking layers.⁴ More layers can be used, but there is a crossover point where decreasing yields make the addition of successive layers cost-prohibitive. When all the tracking and insulating layers have been laid down and dried, the substrate is re-fired to approximately 1000°C.

Creating Resistors

Resistors can be formed from a paste containing carbon compounds suspended in a solvent; the mixture of carbon compounds determines the resistivity of the final component. The resistors require their own screen-print mask and are applied to the substrate using a process identical to that described above (Figure 19.3).

⁴The surface of the substrate becomes increasingly irregular with every layer that is applied. Eventually, the screen-print mask does not make sufficiently good contact across the substrate's surface, and paste "leaks out" under the edges of the patterns.

**FIGURE 19.3**

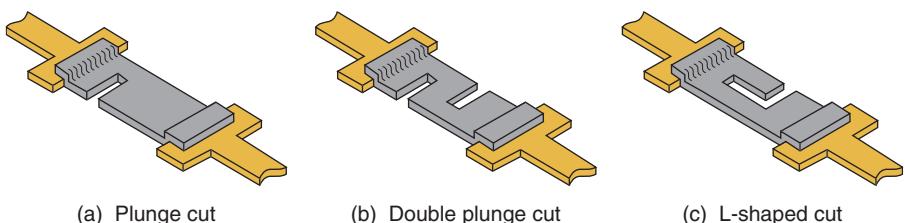
Thick-film: Resistors can be screen-printed onto the substrate.

When the substrate is dried, the solvent evaporates, leaving the carbon compounds to form resistors on the surface of the substrate. Assuming a constant thickness, each resistor has a resistance defined by its length divided by its width, and multiplied by the resistivity of the paste. Thus, multiple resistors with different values can be created in a single screen-print operation by controlling the length and width of each component. However, if a low-resistivity paste is used, resistors with large values will occupy too great an area. Similarly, if a high-resistivity paste is used, resistors with small values will be difficult to achieve within the required tolerances. To overcome these limitations, the process may be repeated with a series of screen-print masks, combined with pastes of different resistivity.

When all of the resistors have been created, the substrate is re-fired to approximately 600°C . An additional screen-print operation is employed to lay a *protective overglaze* over the resistors. This protective layer is formed from a paste containing glass particles in a solvent and is fired at approximately 450°C .

Laser Trimming

Unfortunately, creating resistors as described above is not as exact a process as one could wish for. In order to compensate for process tolerances and to achieve precise values, the resistors have to be trimmed using a laser beam. There are two types of laser trimming: *passive trimming* and *active trimming*. Passive trimming is performed before any of the integrated circuits or discrete components are mounted on the substrate. Probes are placed at each end of a resistor to monitor its value while a laser beam is used to cut parts of the resistive material away. There are a variety of different cuts that may be used to modify the resistor, including *plunge cuts*, *double plunge cuts*, or *L-shaped cuts* (Figure 19.4).

**FIGURE 19.4**

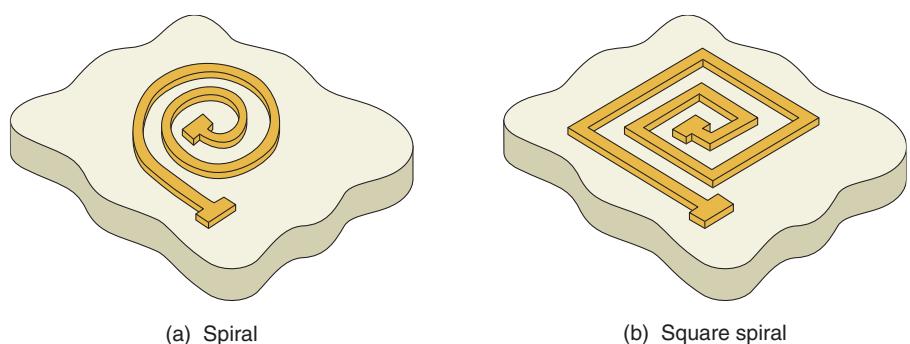
Thick-film: Laser trimming of resistors.

The L-shaped cut combines a plunge cut with a second cut at 90°. In this case, the plunge cut provides a coarse alteration and the second cut supplies finer modifications. After each resistor has been trimmed, the probes are automatically moved to the next resistor and the process is repeated.

By comparison, active trimming is used to fine-tune analog circuits (such as active filters) and requires the integrated circuits and discrete components to be mounted on the substrate. The whole circuit is powered-up and the relevant portion of the circuit is stimulated with suitable signals. A probe is placed at the output of the circuit to monitor characteristics such as amplification and frequency response. A laser is then used to trim the appropriate resistors to achieve the required characteristics while the output of the circuit is being monitored.

Creating Capacitors and Inductors

Capacitors and inductors can also be fabricated directly onto the substrate. However, capacitors created in this way are usually not very accurate, and discrete components are typically used. If inductors are included on the substrate, they are created at the same time and using the same paste as one of the tracking layers. There are two main variations of such inductors: *spiral* and *square spiral* (Figure 19.5).

**FIGURE 19.5**

Thick-film: Inductors can be screen-printed onto the substrate.

The connection to the center-tap of the inductor can be made in several ways. A wire link can be connected to the center-tap, arched over the paths forming the spiral, and connected to a pad on the substrate outside the spiral. A somewhat similar solution is to use a track on another tracking layer to connect the center-tap to a point outside the spiral. As usual, an insulating dielectric layer would be used to separate the layer forming the inductor from the tracking layer. In yet another alternative, the center-tap can be connected to tracks on the bottom side of the substrate by a via placed at the center of the spiral.

Double-sided Thick-Film Hybrids

Thick-film hybrids can support tracking layers on both sides of the substrate, with vias being used to make any necessary connections between the two sides. Components of all types can be mounted on both sides of the substrate as required, but active components such as integrated circuits are usually mounted only on the upper side.

Subtractive Thick-Film Technology

As with any branch of electronics, new developments are always appearing on the scene. For example, a company called Silonex (<http://www1.silonex.com/>) developed a process called *Subtractive Thick Film* (STF), which involves the integration of different technologies. In addition to standard thick film conductors, resistors, capacitors, and inductors, STF features photolithography and chemical etching steps used in conjunction with novel materials. It is claimed that STF produced unprecedented line width density and repeatability, and it expands the capabilities and performance of thick film modules for wireless telecommunication, instrumentation, telemetry, and medical devices such as hearing aids—without sacrificing reliability or cost.

THE THIN-FILM PROCESS

Thin-film processes typically employ either ceramic or glass substrates. The substrate is prepared by sputtering a layer of nichrome (nickel and chromium) alloy across the whole of its upper surface, then electroplating a layer of gold on top of the nichrome. The nichrome sticks to the substrate and the gold sticks to the nichrome. The nichrome and gold layers are each in the order of $5\text{ }\mu\text{m}$ (five-millionths of a meter) thick.

The thin-film process is similar to the opto-lithographic processes used to create integrated circuits. An optical mask is created carrying a pattern formed by areas that are either transparent or opaque to ultraviolet frequencies

(Figure 19.6). As usual, the simple patterns shown in the following diagrams are used for clarity; in practice, such a mask may contain hundreds of thousands of fine lines and geometric shapes.

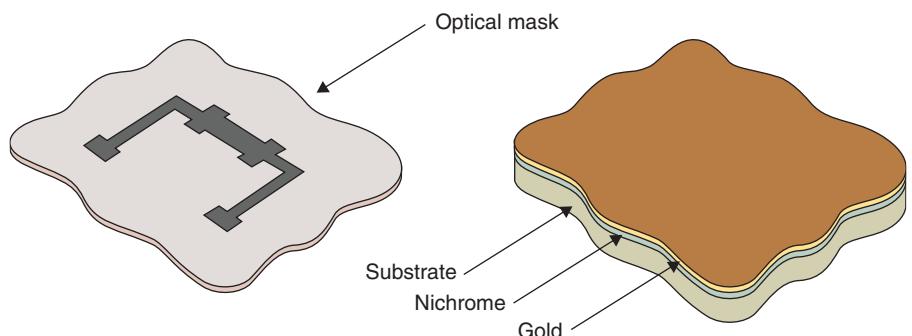


FIGURE 19.6

Thin-film: Optical mask and substrate.

The surface of the gold is coated with a layer of photo-resistive emulsion, which is first dried, then baked, and then exposed to ultraviolet radiation passed through the optical mask. The ionizing radiation passes through the transparent areas of the mask to break down the molecular structure of the emulsion. The substrate is bathed in a solvent that dissolves the degraded resist, then etched with a solvent that dissolves both the gold and nichrome from any areas left unprotected. The nichrome and gold remaining after the first mask-and-etch sequence represent a combination of tracks and resistors (Figure 19.7).

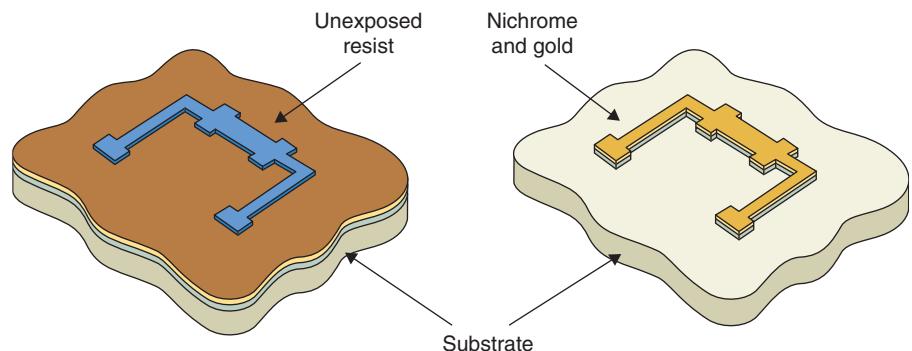


FIGURE 19.7

Thin-film: Combined tracks and resistors.

The thin-film tracks are typically in the order of 0.025 mm in width, but can be as narrow as 0.001 mm in a leading-edge process. The substrate is now recoated with a second layer of emulsion and the process is repeated with a

different mask. The solvent used in this iteration only dissolves any exposed gold, but does not affect the underlying nichrome. The gold is removed from specific sites to expose the nichrome underneath, and it is these exposed areas of nichrome that form the resistors (Figure 19.8).

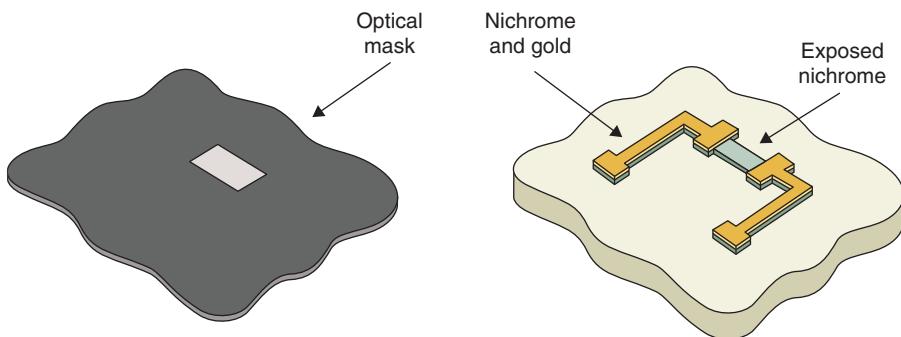


FIGURE 19.8
Thin-film: Separated tracks and resistors.

Laser Trimming

In certain respects, thin-film designers have less freedom in their control of resistance values than do their thick-film counterparts. Although the resistivity of the nichrome layer can be varied to some extent from hybrid to hybrid, the resistivity for a single hybrid is constant across the whole surface. Thus, the only way to select the value of an individual resistor is by controlling its length and width. In addition to simple rectangles, thin-film resistors are often constructed in complex concertina shapes with associated trimming blocks. The resistor values can be subsequently modified by laser trimming (Figure 19.9).

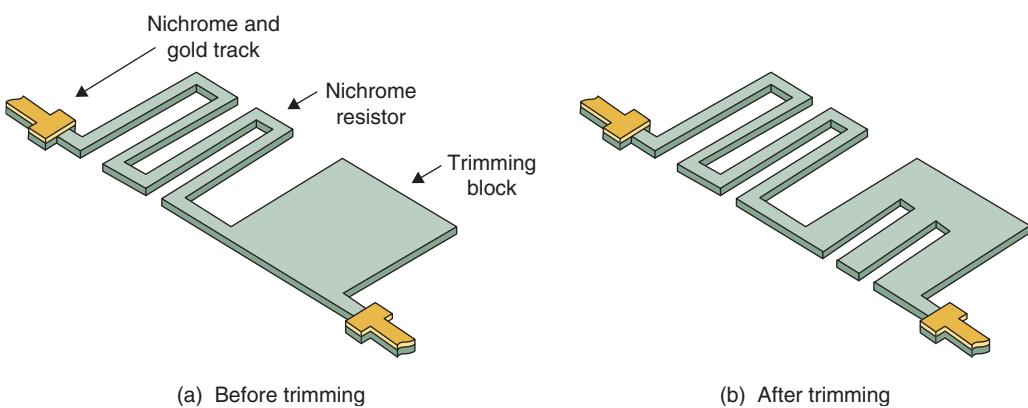


FIGURE 19.9

Thin-film: Resistors may have complex shapes.

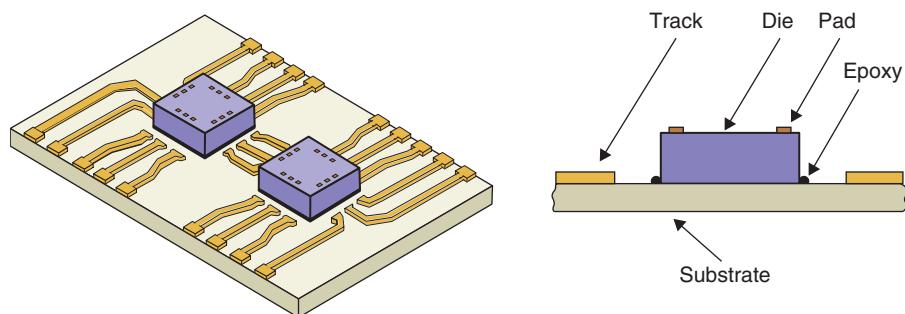
Resistors may also be created in *ladder structures* and their values modified by selectively cutting out rungs from the ladder using the laser. Thin-film hybrids normally employ only one tracking layer on a single side of the substrate; however, multilayer and double-sided variations are available.

THE ASSEMBLY PROCESS

Some hybrids contain only passive components such as resistor-capacitor networks; others contain active devices such as integrated circuits. If integrated circuits are present, they may be individually packaged surface mount devices or unpackaged bare die. Individually packaged integrated circuits are mounted onto the substrate using techniques similar to those used for surface mount technology printed circuit boards.⁵ Unpackaged die are mounted directly onto the substrate using the techniques discussed below.

Attaching the Die

The primary method of removing heat from an unpackaged integrated circuit is by conduction through the back of the die into the hybrid's substrate. One technique for attaching die to the substrate is to use an adhesive such as *silver-loaded epoxy*. The epoxy is screen-printed onto the sites where the die are to be located, the die are pushed down into the epoxy by an automated pick-and-place machine, and the epoxy is then cured in an oven at approximately 180°C (Figure 19.10).



The diagrams shown here have been highly simplified for clarity; in practice, a hybrid may contain many die, each of which could have numerous pads for power and ground connections, and hundreds of pads for input and output signals.

⁵These techniques were introduced in Chapter 18: Circuit Boards.

An alternative method of die attachment is that of a *eutectic bond*,⁶ which requires *gold flash* (a layer of gold with a thickness measured on the molecular level) to be applied to the back of the die and also at the target site on the substrate. The substrate is heated to approximately 300°C, and then an automatic machine presses the die against the substrate and vibrates it at ultrasonic frequencies to create a friction weld. The process of vibrating the die ultrasonically is called *scrubbing*.

Wire Bonds

After the die have been physically attached to the substrate, electrical connections are made between the pads on each die and corresponding pads on the substrate. The most commonly used technique is *wire bonding*, in which an automatic machine makes the connections using gold or aluminum wires (Figure 19.11).

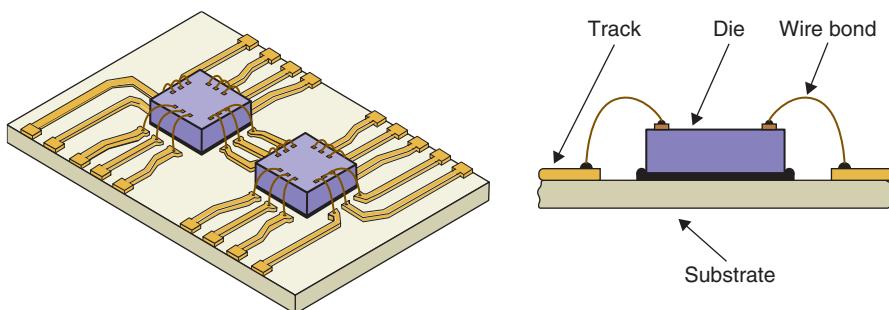


FIGURE 19.11

Wire bonding.

These bonding wires are finer than a human hair,⁷ the gold and aluminum wires typically having diameters in the order of 0.05 mm and 0.025 mm, respectively. In one technique known as a *ball bond*, the end of the wire is heated with a hydrogen flame until it melts and forms a ball, which is then brought into contact with the target. As an alternative to ball bonds, aluminum wires may be attached by pressing each wire against the target and scrubbing it at ultrasonic frequencies to form a friction weld. Similar wire bonds can also be used to form links between tracks on the substrate and to form connections to the hybrid's pins.

⁶Eutectic bonds are primarily used for military and aerospace applications that are intended to withstand high accelerations.

⁷How fine is that? Didn't you read Chapter 14: Integrated Circuits (ICs)? If not, go back and do so immediately!

Tape-Automated Bonding

Another die attachment process is that of *Tape Automated Bonding* (TAB), which is mainly used for high-volume production runs or for devices with a large number of pads. In this process, a transparent flexible tape is coated with a thin layer of metal, into which track-like TAB leads lead frames are patterned using standard opto-lithographic techniques. The pads on the bare die are attached to corresponding pads on the tape, which is then stored in a reel (Figure 19.12).

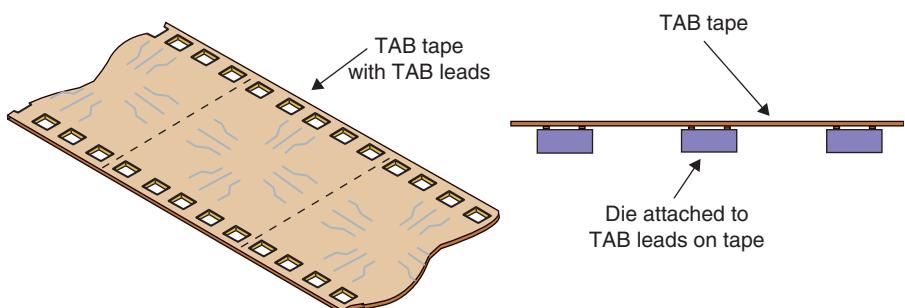


FIGURE 19.12
Bare die attached to TAB tape.

In practice, the TAB leads on the tape can be so fine and so close together that they are difficult to distinguish with the human eye. When the time comes to attach the die to the hybrid's substrate, silver-loaded epoxy is screen-printed onto the substrate at the sites where the devices are to be located. This silver-loaded epoxy is also printed onto the substrate's pads to which the TAB leads are to be attached. The reel of TAB tape with attached die is fed through an automatic machine, which pushes the die and the TAB leads down into the epoxy at their target sites. When the silver-loaded epoxy is subsequently cured, it forms electrical connections between the TAB leads and the pads on the substrate (Figure 19.13).

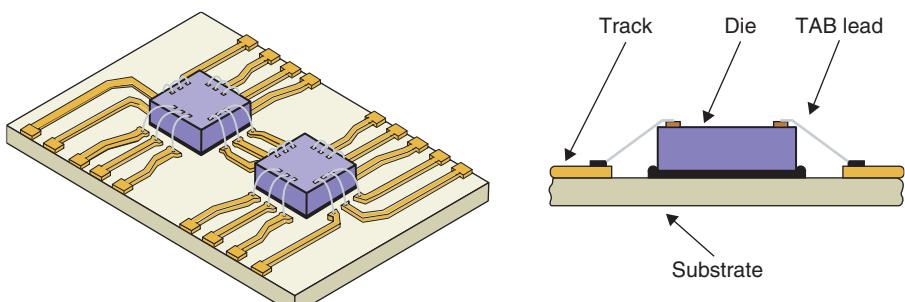


FIGURE 19.13
Tape-automated bonding (TAB).

Flipped-Chip Techniques

As an alternative to the previously described approaches, there are several attachment techniques grouped under the heading of *flipped chip*. In addition to having the lowest inductance for the connections between the die and the substrate, flipped-chip techniques provide the highest packing densities because each die has a small *footprint* (the amount of area occupied by the die). Thus, flipped-chip techniques may be used where the substrate's real estate is at a premium and the die have to be mounted very closely together.

One flipped chip technique called *solder bumping* requires perfect spheres of solder to be formed on the pads of the die. The die are then "flipped over" to bring their solder bumps into contact with corresponding pads on the substrate (Figure 19.14).

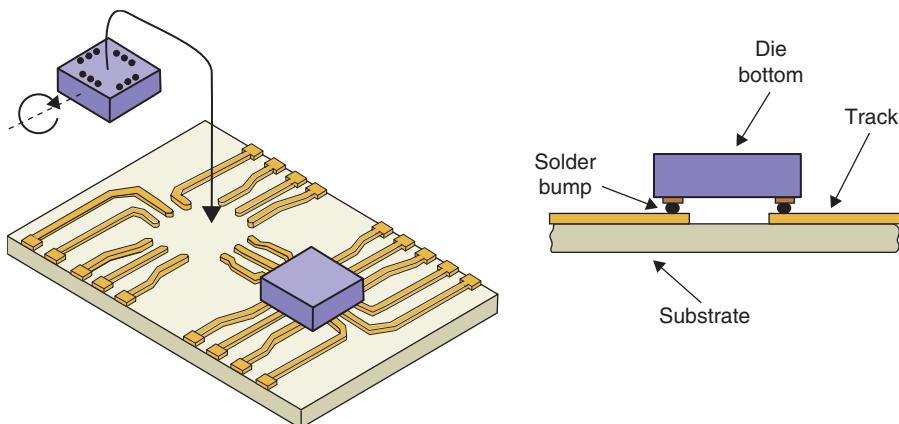


FIGURE 19.14

Flipped-chip: Solder bumping.

One advantage of this solder bumping technique is that the pads are not obliged to be at the periphery of a die; connections may be made to pads located anywhere on a die's surface. When all of the die have been mounted on the substrate, the solder bumps can be melted in a *reflow oven* using infrared radiation or hot air. Alternatively, the solder may be melted using *vapor-phase soldering*, in which the board is lowered into the vapor-cloud above a tank containing boiling hydrocarbons.⁸

⁸As for printed circuit boards, vapor-phase soldering is becoming increasingly less popular due to environmental concerns.

Another flipped chip alternative is *flipped TAB*, which, like standard TAB, may be of use for high-volume production runs or for devices with a large number of pads (Figure 19.15).

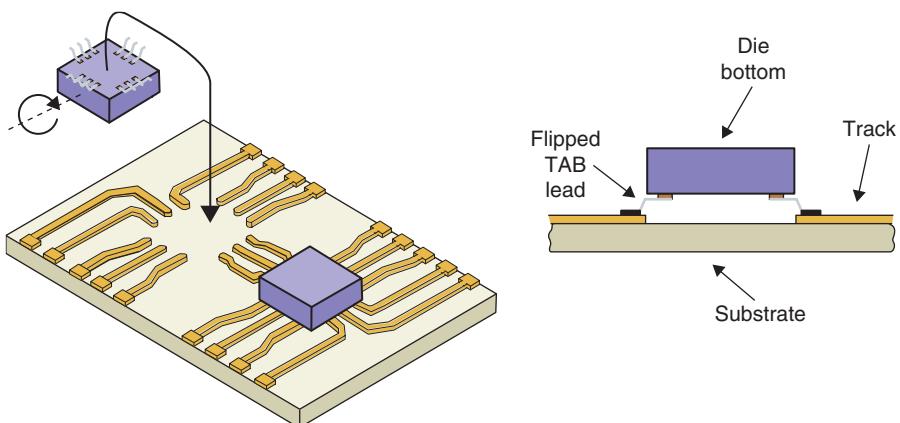


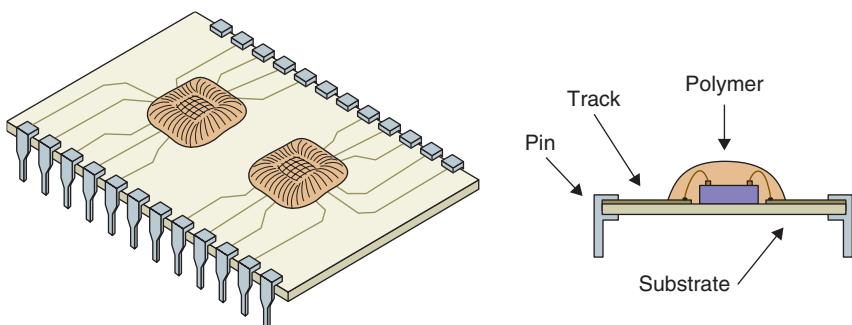
FIGURE 19.15
Flipped-chip: Flipped TAB.

Advantages of Using Bare Die

Hybrids in which bare die are attached directly to the substrate are smaller, lighter, and inherently more reliable than circuits using individually packaged integrated circuits. Every solder joint and other form of connection in an assembly is a potential failure mechanism. Individually packaged integrated circuits already have a connection from each pad on the die to the corresponding package lead and require a second connection from this lead to the substrate. By comparison, in the case of a bare die, only a single connection is required between each pad on the die and its corresponding pad on the substrate. Thus, using bare die results in one less level of interconnect, which can be extremely significant when using die with hundreds or thousands of pads.

THE PACKAGING PROCESS

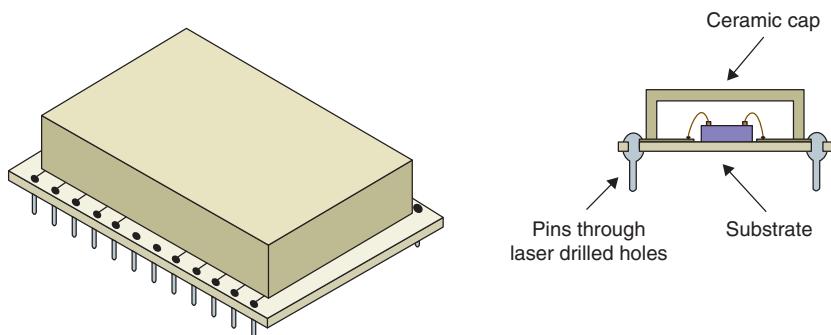
Hybrids come in a tremendous variety of shapes, sizes, and packages. Many hybrids are not packaged at all, but simply have their external leads attached directly to the substrate. In this case, any bare die are protected by covering them with blobs of plastic polymer in a solvent; the solvent is evaporated, leaving the plastic to protect the devices (Figure 19.16).

**FIGURE 19.16**

Bare die encapsulated in plastic.

This is very similar to the *Chip-on-Board* (COB) techniques introduced in Chapter 18: *Circuit Boards*. In fact, if the hybrid's substrate were a small printed circuit, this would fall into the category of chip-on-board.

If the hybrid is targeted for a harsh environment, the entire substrate may be coated with polymer. If necessary, additional physical protection can be provided by enclosing the substrate with a ceramic cap filled with a dry nitrogen atmosphere (Figure 19.17).

**FIGURE 19.17**

Ceramic cap package.

In this case, any bare die may remain uncovered because the dry nitrogen environment protects them. If extreme protection is required, the hybrid may be hermetically sealed in alloy cans filled with dry nitrogen (Figure 19.18).

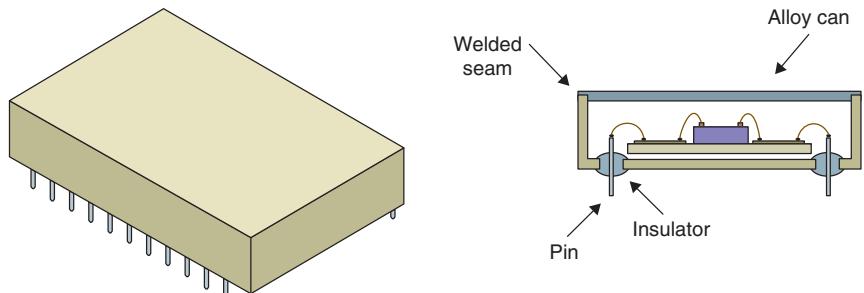


FIGURE 19.18
Hermetically sealed package.

Finally, it should be noted that the majority of the assembly and packaging techniques introduced above are not restricted to hybrids, but may also be applied to individual integrated circuits [as discussed in *Chapter 14: Integrated Circuits (ICs)*] and System-in-Package (SiP) assemblies (as discussed in *Chapter 20: Advanced Packaging Techniques*).

CHAPTER 20

Advanced Packaging Techniques

SLIDING DOWN THE RABBIT HOLE

Traditionally, integrated circuits, hybrids, and printed circuit boards were differentiated on the basis of their substrate materials: semiconductors (mainly silicon), ceramics, and organics, respectively. Unfortunately, even this simple categorization is less than perfect. Some hybrids use laminate substrates, which are, to all intents and purposes, small printed circuit boards. So, at what point does a printed circuit board become a hybrid with a laminate substrate?

The problem is exacerbated in the case of *System-in-Package* (SiP), which is a generic name for a group of advanced interconnection and packaging technologies featuring multiple integrated circuits (which may be presented as bare die and/or individually packaged) mounted directly onto a common substrate (base). We'll consider SiP technologies in more detail in a moment, but first let's remind ourselves of a few things ...

293

WIRE BONDS VERSUS FLIP-CHIP

Let's start by considering a package containing a single silicon chip (die). In one packaging style, the back of the die is attached to the substrate (the base of the package), and an automatic wire-bonding tool connects the pads on the die to corresponding pads on the substrate with wire bonds finer than a human hair.

Alternatively, in the case of a flip-chip technique, the pads on the die are no longer restricted to its periphery, but are instead located over the entire face of the die. A minute ball of solder is attached to each pad, and the die is flipped over and attached to the package substrate. Each pad on the die has a corresponding pad on the package substrate, and the package-die combo is heated so as to melt the solder balls and form good electrical connections between the die and the substrate (Figure 20.1).

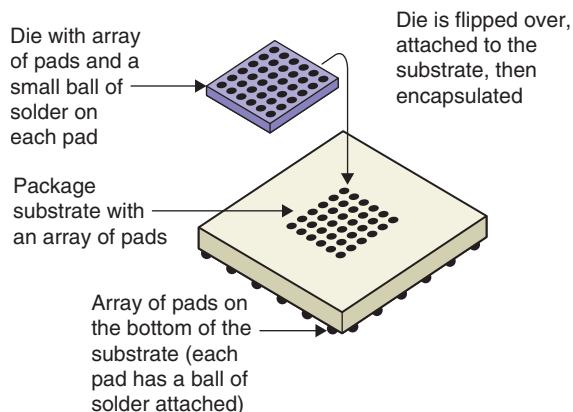


FIGURE 20.1
A flip-chip ball-grid array packaging technique.

Irrespective of whether we use a wire bond or flip-chip approach, the package's substrate itself may be made out of the same material as a printed circuit board, out of ceramic, or out of some even more esoteric material. Whatever its composition, the substrate will contain multiple internal wiring layers that connect the pads on the upper surface with pads (or pins) on the lower surface. The pads (or pins) on the lower surface (the side that actually connects to the circuit board) will be spaced much further apart—relatively speaking—than the pads that connect to the die. At some stage the package will have

to be attached to a circuit board. In one technique known as a *Ball Grid Array* (BGA), the package has an array of pads on its bottom surface, and a small ball of solder is attached to each of these pads. Each pad on the package will have a corresponding pad on the circuit board, and heat is used to melt the solder balls and form good electrical connections between the package and the board (we'll return to look at BGA packages in more detail later in this chapter).

WIRE BONDING AND FLIP-CHIP

Another technique to get the most silicon into a small package is to take a die, flip it over, and mount it face down on the package substrate using a flip-chip technique. Next, we take a second die, mount it back-to-back on top of the first (so that the second die is face-up), and use wire bonds to connect this second die to the package substrate. Then we package the whole thing as a single entity.

CHIP-SCALE PACKAGE (CSP) TECHNOLOGY

Modern packaging technologies are extremely sophisticated. Some ball grid arrays have pins spaced 0.3 mm (one-third of a millimeter) apart! In the case of *Chip-Scale Package* (CSP) technology, the package is barely larger than the die itself.¹

In one technique, the chip is flipped over and mounted on an *interposer*, which is used to redistribute and rearrange the signals. Pads or balls are formed on the bottom of the interposer and these connect to the circuit board (or Hybrid or SiP, as we

¹In order to qualify as “chip scale,” the package must have an area no greater than 1.2 times that of the die.

shall see). This is very similar to the flip-chip BGA technique discussed in the previous topic, except that the interposer is only fractionally larger than the die itself.

Another variation of this technique is referred to as a *Wafer-Level Chip-Scale Package* (WL-CSP). This is so-called because, while the die are still part of the wafer, one or more *Redirection Layers* (RDLs) are added to re-route the internal signals and present them as an array of pads on the surface of the die. Silicon bumps are then added to these pads, the die is coated with a thin protective layer, and the chip is mounted directly to the circuit board (or Hybrid or SiP).

3-D DIE STACKING

Even when using *Chip-Scale-Package* (CSP) or *Chip-on-Board* (COB)² techniques, the intra-chip connections are a source of fairly significant delays. One obvious solution is to mount the chips as closely together as possible, thereby reducing the lengths of the tracks and the delays associated with them. However, each chip can have only eight others mounted in close proximity on a 2-D substrate. Thus, in the early 1990s, some specialist applications began to employ a technique known as *3-D die stacking*, in which several bare die (which are extremely thin) are stacked on top of each other to form a sandwich. The die are connected together and then packaged as a single entity (Figure 20.2).

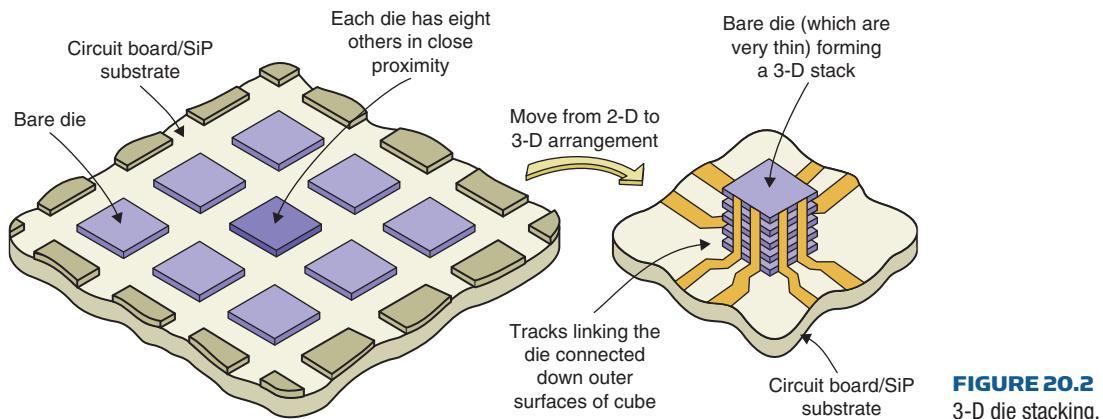


FIGURE 20.2
3-D die stacking.

One problem with this 3-D die stacking technique is the amount of heat that is generated, which drastically affects the inner layers forming the cube (this problem could be eased by constructing the die out of diamond, as discussed

²The concept of *Chip-on-Board* (COB) was introduced in Chapter 18: *Circuit Boards*.

in *Chapter 21: Alternative and Future Technologies*). Another problem with traditional techniques is that any tracks linking the die must be connected down the outer surfaces of the cube. The result is that the 3-D die-stacking technique has typically been restricted to applications utilizing identical die with regular structures. For example, the most common application to date has been large memory devices constructed by stacking SRAM or DRAM die on top of each other.

A relatively new technique which may serve to alleviate the problem of chip-on-chip interconnect is a process for creating vias through silicon substrates. Known as *Through Silicon Via* (TSV), experiments have been performed in which aluminum “blobs” are placed on the surface of a silicon substrate and, by means of a gradient furnace, the aluminum migrates through the silicon, providing vias from one side to the other (Figure 20.3).

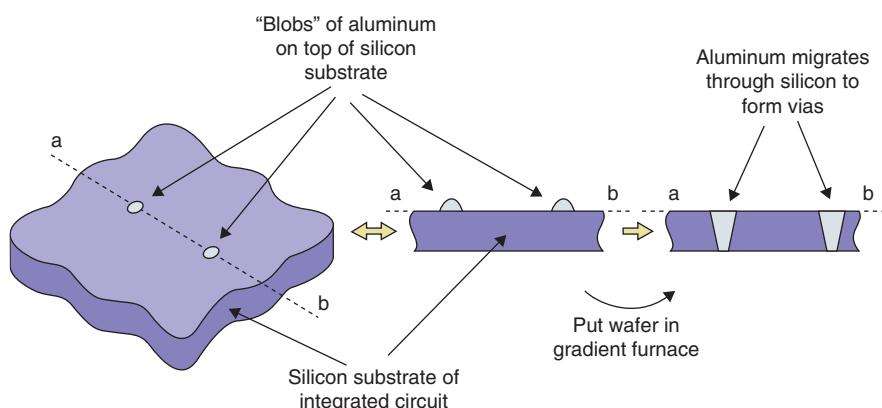


FIGURE 20.3
Creating vias through silicon die.

Another technique more in keeping with the times is to create the vias by punching the aluminum through the silicon using a laser. These developments pave the way for double-sided silicon substrates with chips and interconnect on both sides. Additionally, they offer strong potential for interconnecting the die used in 3-D die stacking structures.³

SYSTEM-IN-PACKAGE (SIP), PIP, AND POP

As we've noted on several occasions, early electronic systems were typically composed of a number of individually packaged integrated circuits, each with

³When I first mentioned these concepts in earlier editions of this tome, it was as an “Alternate and Future Technology” in *Chapter 21: Alternative and Future Technologies*. At the time of this writing in 2008, these processes are starting to come online.

its own particular function (say a microprocessor, some peripheral functions, some memory devices, etc.). For many of today's high-end applications, however, electronics engineers combine all of these functions on a single device, which may be referred to as a *System-on-Chip* (SoC).

The SoC approach is to create a single humongous die, which is subsequently encased in its own package. Although this sounds jolly clever, there are several problems with this tactic, such as the fact that these components are time-consuming (taking anywhere from, say, 9 to 18 months) and expensive (say, \$15 to \$50 million) to develop and deploy. Also, there are issues with integrating analog and digital functions on the same die, not the least of which is that the cutting edge of digital design is currently at the 40-nm technology node, while the bleeding edge of analog design is only at the 90-nm node. There are also yield issues with larger dies. And yet another consideration is the time and expense involved in re-spinning the design in the future to evolve existing functionality or to add new features.

The solution is that of *System-in-Package* (SiP),⁴ in which multiple bare or *Chip-Scale Package* (CSP) die are mounted on a common substrate that is used to connect them all together. The substrate and its components are then placed in (or built into) a single package. This approach has several advantages, including the fact that one can include analog, digital, and *Radio Frequency* (RF) die in the same package, where each die is implemented using that domain's most appropriate technology process. Also, designers can employ a number of off-the-shelf die, coupled, perhaps, with a limited number of relatively small, internally developed ASICs.

Just to increase the fun and frivolity, it's possible to create a number of small SiPs and then mount these in a larger SiP, in which case we have a scenario known as *Package-in-Package* (PiP). Also, in some cases, we have a SiP that is mounted on top of another SiP, which some refer to as a *Package-on-Package* (PoP).

A POSITIVE PLETHORA OF SUBSTRATES

There are a wide variety of substrates that may be used for *System-in-Package* (SiP) assemblies depending on the requirements of the particular application. Some possibilities are as follows:

- Laminates, such as small, fine-line printed circuit boards with copper tracks and copper vias. These are usually made out of FR4 or polyimide, and typically contain 5 to 25 tracking layers.

⁴The forerunners to SiPs were known as *multichip modules*, which first appeared around 1990.

- Ceramic substrates, some of which are similar to those used for hybrids: formed from a single, seamless piece of ceramic and carrying tracks that are created using thick-film or thin-film processes (or a mixture of both). However, a large proportion of ceramic *System-in-Packages* (SiPs) are of the *cofired* variety (see the following topic), in which case they are formed from a material such as aluminum nitride or beryllium oxide, and can contain hundreds of layers.
- Ceramic, glass, or metal substrates, that are covered with a layer of dielectric material, such as polyimide. The dielectric coat is used to modify the substrate's capacitive characteristics and tracks are created on the surface of the dielectric using thin-film processes. Modules of this type typically have around five tracking layers.
- Semiconductor substrates, predominantly silicon, with very fine tracks formed using opto-lithographic processes similar to those used for integrated circuits. Semiconductor substrates are also known as *active substrates*, because components such as transistors and logic gates can be fabricated directly onto their surface. One additional benefit of using silicon as a substrate is that its coefficient of thermal expansion exactly matches that of any silicon chips that are attached to it.

AN EXAMPLE SIP BASED ON COFIRED CERAMICS

Ceramic substrates have numerous benefits (as were discussed in *Chapter 19: Hybrids*). However, once the ceramic has been fired, it is very difficult to machine and cannot be satisfactorily milled, drilled, or cut using currently available technology.⁵ One solution to this problem is *cofired ceramics*, in which a number of slices of ceramic are preformed and then fired together.

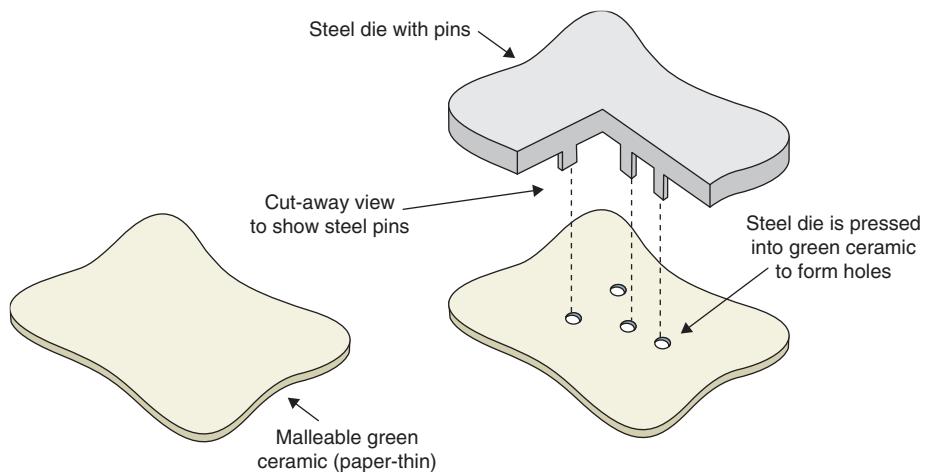
The process begins with unfired, or *green*, ceramic tape, which is rolled out into paper-thin layers. After each layer has been cut to the required size, microscopic holes are punched through it to form vias.⁶ One technique used to punch these holes involves a steel die⁷ studded with round or square pins (Figure 20.4).

Unfortunately, although fast in operation, these dies take a long time to construct and the technique is incredibly expensive—each die may cost in the neighborhood of \$10,000 and can typically be used for only a single layer of a single

⁵Modern laser technology offers some success in drilling and cutting ceramics, but not as much as one might hope.

⁶Cofired substrates typically use a tremendous number of blind and buried vias.

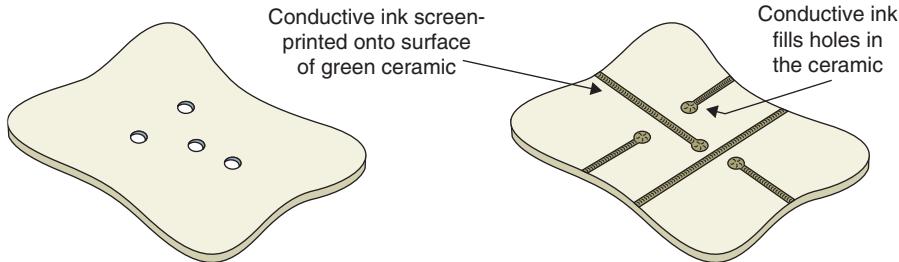
⁷In this context, the term *die* refers to a piece of metal with a design engraved or embossed on it for stamping onto another material, upon which the design appears in relief.

**FIGURE 20.4**

Cofired ceramics:
Punching holes to
form vias.

module. As an alternative, some manufacturers have experimented with programmable variants consisting of a matrix of pins, each of which may be retracted or extended under computer control. Programmable die are very expensive tools to build initially, but can be quickly reprogrammed for multiple layers and multiple modules. Their main drawback with this technique is that all of the pins are on a fixed grid. Another approach that is routinely used is that of a numerically (computer) controlled tool which punches the holes individually. This technique is relatively inexpensive, but requires much more time to complete the task.

After the holes have been punched, a screen-printing operation is used to backfill the holes with a conducting ink (a paste containing metallic powder). The mask used in this step is referred to as a *via-screen*. Next, tracks are screen-printed onto the surface of each layer using the same conducting ink. These processes are similar to the thick-film hybrid technique, but use much finer meshes for the screen print masks, thereby resulting in much finer lines and spaces (Figure 20.5).

**FIGURE 20.5**

Cofired ceramics:
Screen-printing tracks.

After all of the layers have been prepared in this way, they are stacked up, pressed together, and prefired at a relatively low temperature to burn off the binders used in the conductive inks and any moisture in the ceramic. The substrate is then fully fired at around 1600°C, which melts the tracks to form good conductors and vias, and transforms the individual layers into one homogeneous piece of ceramic (Figure 20.6).

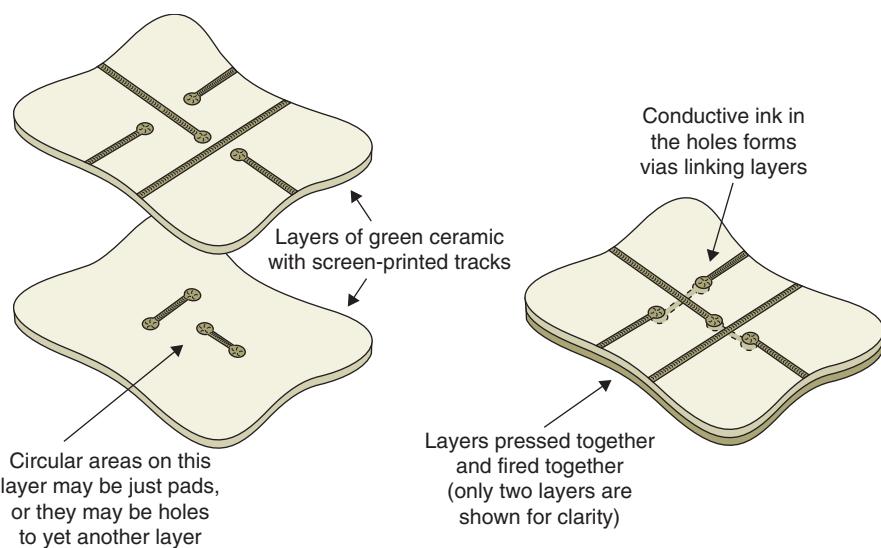


FIGURE 20.6
Cofired ceramics:
Assembling and firing
the layers.

Cofired ceramic substrates constructed in this way can contain an amazing number of layers. Substrates with 100 layers are not uncommon, while leading-edge processes may support 260 layers or more. To provide a sense of scale, an 85-layer substrate may be approximately 4.5 mm thick, while a 120-layer substrate will be around 6.25 mm thick. Finally, after the substrate has been fired, any tracks and pads on the surface layers are plated with a *noble metal*⁸ such as gold.

One not-so-obvious consideration with cofired ceramics is that the act of rolling out the ceramic tape lines up the particles of ceramic in the direction of the roll. This causes the layers to shrink asymmetrically when the ceramic is fired. To counter this effect, the layers are alternated such that adjacent layers are rolled in different directions. The net result is symmetrical shrinkage with respect to the major axes.

⁸Noble metals, such as gold and platinum, are extremely inactive and are unaffected by air, heat, moisture, and most solvents.

Low-Fired Cofired Ceramics

One problem with traditional cofired processes is that the tracks must be created using *refractory metals*.⁹ Although these are reasonable conductors, they are not ideal for high-reliability and high-performance applications. For high reliability, one would ideally prefer a noble metal like gold, while for high performance at a reasonable price one would ideally prefer copper. (Copper is one of the best conductors available for high-frequency applications, especially those operating in the microwave arena.) Unfortunately, metals like gold and copper simply vaporize at refractory temperatures.

The solution is *low-fired cofired processes*, which use modern ceramic materials whose compositions are very different from those of traditional ceramics. To differentiate the two, the term HTCC (*High-Temperature Cofired Ceramics*) is used to refer to ceramics requiring tracks to be formed using refractory metals, while the term LTCC (*Low-Temperature Cofired Ceramics*) refers to their lower temperature counterparts, which can be fired at temperatures as low as 850°C. This allows tracks to be screen-printed using copper-based or noble metal-based conductive inks (the firing must take place in a nitrogen-muffled furnace to prevent the metals from oxidizing).

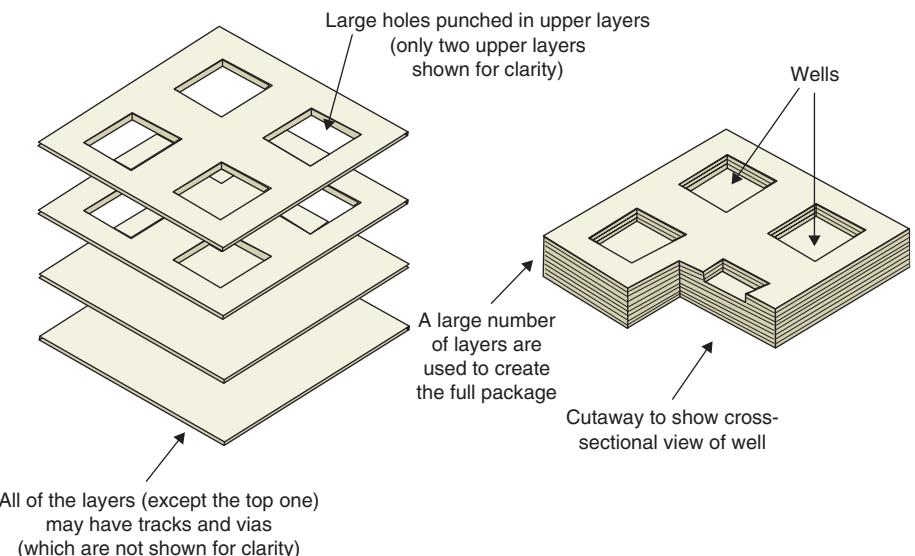
Assembly and Packaging

To a large extent, *System-in-Packages* (SiPs) are created on a design-specific basis. Purely for the sake of these discussions, we will assume a SiP based on a cofired ceramic substrate carrying four bare die (a real-world SiP can contain many more components, including multiple analog, digital, and RF die, and also discrete components as required).

Because cofired substrates are constructed from multiple layers, designers have the ability to create quite sophisticated structures. For example, holes called *cavities*, or *wells*, can be cut into a number of the substrate's upper layers before they are combined together and fired (Figure 20.7).

Remembering that the layers of ceramic are paper-thin, the wells are actually formed from whatever number of layers is necessary to bring them to the required thickness. These wells will eventually accommodate the die, while the remaining ceramic will create a bed for the package's lid. Before the die are mounted on the substrate, however, the external leads must be attached.

⁹Refractory metals are those such as tungsten, titanium, and molybdenum, which are capable of withstanding the extremely high, or refractory, temperatures necessary to fire the ceramic.

**FIGURE 20.7**

Cofired ceramics:
Creating wells.

Pin Grid Arrays

In the case of a packaging style known as a *Pin Grid Array* (PGA), the package's external connections are arranged as an array of conducting leads, or pins, in the base.¹⁰ In many PGAs, the leads are brazed directly onto the bottom layer of the device. Alternatively, in a similar manner to the way in which the wells were created in the upper layers, smaller wells may be cut into a number of the substrate's lower layers before they are combined together and fired. Each of these wells is connected to a track constructed on one of the inner layers. After firing, the wells are lined with a thin layer of metal and the leads are inserted (Figure 20.8).

After the leads have been inserted, the entire assembly is heated to approximately 1000°C, which is sufficient to braze the leads to the layer of metal lining the wells. Finally, the leads are gold-plated to ensure that they will make good electrical connections when the module is eventually mounted on a circuit board.

Pad, Ball, and Column Grid Arrays

As an alternative to pin grid arrays, some modules are presented as *Pad Grid Arrays* (PGAs), in which the external connections are arranged as an array of conducting pads on the base. In yet another alternative, which has become one

¹⁰Any package substrate (ceramic, laminate, etc.) may be presented to the outside world in the form of a *Pin Grid Array* (PGA). Also, individual silicon chips may also be presented in a PGA packages.

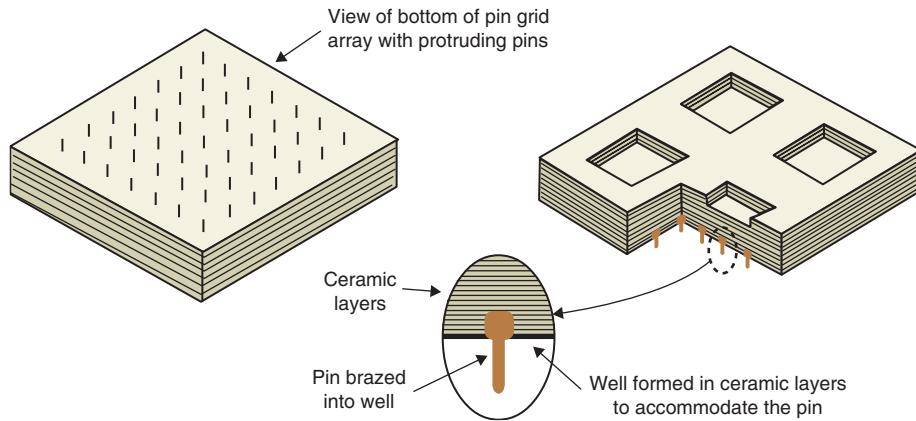


FIGURE 20.8
Pin grid arrays (PGAs).

of the premier packaging technologies currently in use, devices may be presented as *Ball Grid Arrays* (BGAs)^{11,12} or *Column Grid Arrays* (CGAs). These forms of package are similar to a *Pad Grid Array* (PGA), but with the addition of small balls or columns of solder attached to the conducting pads (Figure 20.9).

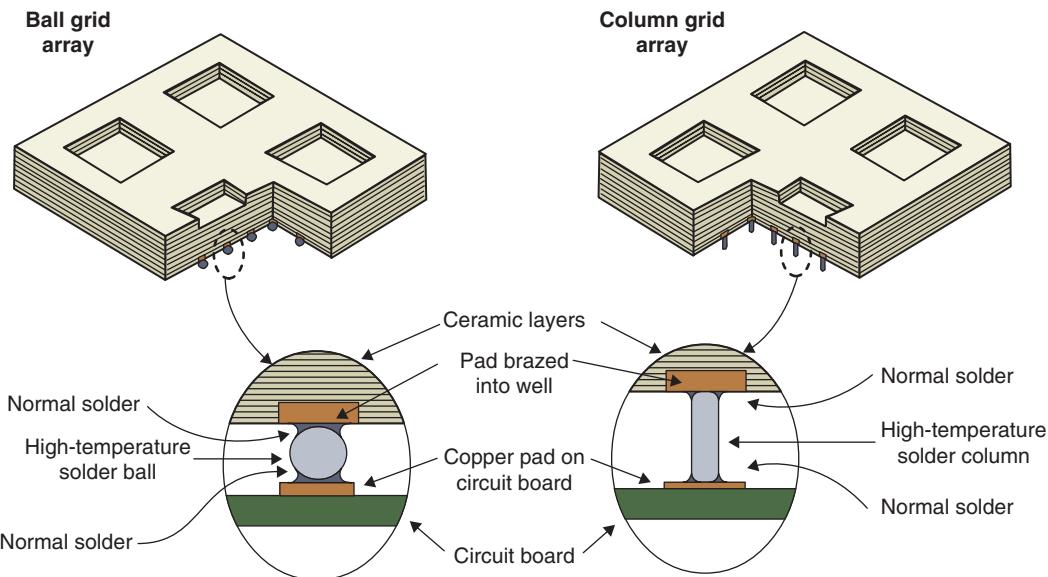


FIGURE 20.9

Ball grid arrays (BGAs) and column grid arrays (CGAs).

¹¹Ball grid arrays may also be referred to as *solder grid arrays*, *bump grid arrays*, or *land grid arrays*.

¹²Any package substrate (ceramic, laminate, etc.) may be presented to the outside world in the form of a *Ball Grid Array* (BGA) or *Column Grid Array* (CGA). Also, individual silicon chips may also be presented in BGA or CGA packages.

The balls (or columns) are made out of a high-temperature solder, which does not melt during the soldering process. Instead, a standard low-temperature solder, referred to as *eutectic solder*, is used to connect the balls (or columns) to the pads. These packages are said to be “self-aligning” because, when they are mounted on a circuit board and heated in a reflow oven, the surface tension of the eutectic solder causes the device to automatically align itself.

Ball grid arrays are physically more robust than column grid arrays. However, column grid arrays are better able to accommodate any thermal strains caused by temperature mismatches between the SiP and the circuit board.

Fuzz-Buttons

In yet another variant, which may be of use for specialist applications, small balls of fibrous gold known as *fuzz-buttons* are inserted between the pads on the base of a *Pad Grid Array* (PGA) package and their corresponding pads on the board. When the package is forced against the board, the fuzz-buttons compress to form good electrical connections. One of the main advantages of the fuzz-button approach is that it allows broken devices to be quickly removed and replaced.

Populating the Die

Once the external leads have been connected, the SiP is ready to be populated with its die (Figure 20.10). In addition to *Chip Scale Packages* (CSPs) and 3-D die stacks, bare die can be physically mounted on the substrate using adhesive, eutectic, or flipped-chip techniques, and electrically connected using wire bonding, tape automated bonding, or solder bumping methodologies. After the die have been mounted, the wells are back-filled with nitrogen, and protective lids are firmly attached.

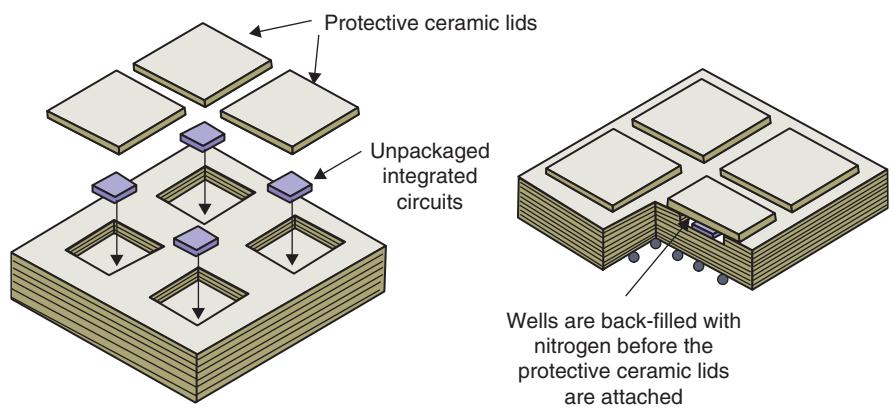


FIGURE 20.10
Populating the SiP.

Before we close, we should remind ourselves that this is just one SiP example. If we had wished, we could have used the same low-fired cofired ceramic substrate, but we could have omitted the wells and mounted the components on top of the substrate. Alternatively, we could have used some other form of substrate such as a laminate (circuit board). Also, we could use ceramic cap or hermetically sealed packages as discussed for hybrids in *Chapter 19: Hybrids*. In fact, generally speaking, there are no limits to the sorts of things people are doing in the SiP arena.

THE MIND BOGGLES

In addition to the substrates introduced earlier in this chapter, some SiPs use more esoteric materials such as quartz, garnet, and sapphire.¹³ These modules are targeted toward specialized applications. For example, the ultra-high frequency capabilities of sapphire are of particular interest in the microwave arena. Additionally, some SiPs use a mixture of substrates: for example, ceramic substrates with areas of embedded or deposited silicon or diamond film. Other devices may use a mixture of tracking techniques: for example, thin-film processes for surface signals, combined with thick-film processes for inner layers, or vice versa.

In addition to complex materials selection and process problems, designers of SiPs are faced with a number of other challenges. Parasitic and thermal management are key issues due to the high interconnection density and high circuit speeds. Thus, it is simply not possible to split a SiP design into an interconnection problem, a thermal management problem, and a packaging problem. Each SiP presents a single mammoth problem in concurrent design and typically requires a unique solution. The result is that working with SiPs can be a very expensive hobby in terms of tooling requirements and long design times.

The SiP designer's task has migrated from one having primarily electronic concerns to one incorporating components of materials science, chemistry, and packaging. In addition to basic considerations—such as selecting the packaging materials and determining the optimal connection strategy (pin versus pad versus ball grid array)—the designer must consider a variety of other packaging problems, including lead junction temperatures, thermal conduction, and convection characteristics.

¹³See also the discussions on diamond substrates in *Chapter 21: Alternative and Future Technologies*.

Furthermore, it is no longer possible for the chip, package, and board design teams to work in isolation. The problem is that there are so many facets to this that it makes your head spin. Consider a SiP containing a number of die, each of which boasts hundreds or thousands of pins. If the I/O placements and bump assignments on the die are performed without considering how they will interface to the package, the result may be signal integrity issues, performance issues, increased package size, and an increase in the number of routing layers inside the package. In turn, excessive package complexity can easily push the cost of the package higher than the cost of the silicon chips it contains, thereby rendering the component uneconomical.

Similarly, if the SiP is designed without considering how it will interface to the circuit board on which it will reside, ineffectively assigned pins on the package can result in problems breaking-out the signals on the board. Once again, this can result in signal integrity issues, performance issues, increased board size, and an increase in the number of routing layers on the board.

The end result is that, in the case of today's bleeding-edge systems, the chips, packages, and boards must be designed in conjunction with each other. The magnitude of the task is daunting, but if it was easy then everybody would be doing it, and you can slap me on the head with a kipper¹⁴ if it doesn't make life exciting!

¹⁴A kipper is a smoked and salted fish (especially tasty when brushed with butter, toasted, and served as a breakfast dish with strong, hot English tea).

CHAPTER 21

Alternative and Future Technologies

A SMORGASBORD OF TECHNOLOGIES

Electronics is one of the most exciting and innovative disciplines around, with evolutionary and revolutionary ideas appearing on almost a daily basis. Some of these ideas skulk around at the edges of the party, but never really look you in the eye or take the trouble to formally introduce themselves; some surface for a short time and then disappear forever into the twilight zone from whence they came; some tenaciously manifest themselves in mutated forms on a seasonal basis; and some leap out as if from nowhere with a fanfare of trumpets, and join the mainstream so quickly that before you know it, they seem like old friends.

307

In this chapter, we introduce a smorgasbord of technologies, many of which have only recently become commercially available or are on the cutting-edge of research and development. Even the most outrageous topics presented below have undergone experimental verification, but nature is a harsh mistress and natural selection will take its toll on all but the fittest. Although some of the following may seem to be a little esoteric at first, it is important to remember that a good engineer can easily believe three impossible things before breakfast. Also remember that the naysayers proclaimed that it was impossible for bumblebees to fly (although they obviously could), that man would never reach the moon, and that I would never finish this book!¹

RECONFIGURABLE COMPUTING

The term *hardware* is generally understood to refer to any of the physical portions constituting an electronic system, including nuts and bolts, connectors, components, circuit boards, power supplies, cabinets, and monitors.

¹Ha! I pluck my chest hairs threateningly in their general direction.

The term *software* refers to programs, or sequences of instructions, that are executed by hardware. Additionally, *firmware* refers to software programs that are hard-coded into nonvolatile memory devices, while *vaporware* refers to either hardware or software that exists only in the minds of the people who are trying to sell them to you.²

In the days of yore, electronic circuits were hard-wired, which was extremely limiting. The advent of computers made things much more interesting, because we now had the ability to modify the software program that was running on the computer hardware, but what about the concept of modifying the hardware itself ...

Unfortunately, the concept of *reconfigurable computing* is akin to the phrase “stretch-resistant socks” in that they both mean different things to different people. To the young and innocent, “stretch-resistant” would tend to imply a pair of socks that will not stretch. But, as those of us who are older, wiser, and a little sadder know, “stretch-resistant” actually refers to socks that will indeed stretch—they just do their best to resist it for a while! Similarly, the concept of *reconfigurable computing* is subject to myriad diverse interpretations depending on the observer’s point of view.

The advent of SRAM-based FPGAs³ presented a new capability to the electronics fraternity—the ability to configure (program) the device to perform different tasks as required. When a system is first turned on, for example, the FPGAs might be configured to perform diagnostic functions, both on themselves and on the rest of the system. Once the diagnostic checks have been completed, the system can reconfigure the FPGAs to fulfill the main function of the design.

The main limitation with the majority of SRAM-based FPGAs is that it is necessary to load the entire device. Also, it is usually necessary to halt the operation of the entire circuit board while these devices are being reconfigured. Additionally, the contents of any registers in the FPGAs are irretrievably lost during the process. These problems are addressed to some extent by hybrid (FLASH-SRAM) FPGAs, in which every programmable element in the device has an associated FLASH bit and an SRAM cell. When the device powers up, the contents of each nonvolatile FLASH bit are copied into its corresponding SRAM cell. Such a device is essentially “instant on,” but it can also be easily

²The term “wetware” may refer either to human brains or programs that are very new and are not yet as robust as one might hope.

³FPGAs were introduced in *Chapter 16: Programmable ICs*.

reprogrammed as required. Furthermore, once the device has been powered up and is running under its initial configuration, the chip can continue running while its FLASH is reprogrammed with a new configuration. This new configuration can subsequently be copied over into the SRAM configuration cells in a fraction of a second.

But, as wonderful as all of this may sound, it's not what many folks consider to be true reconfigurable computing. As far as I (the author) am concerned, reconfigurable computing means having an FPGA-like⁴ component with the ability to reconfigure selected portions (up to and including the entire device) while also providing the following:

- No disruption to the device's inputs and outputs.
- No disruption to the system-level clocking.
- The continued operation of any portions of the device that are not undergoing reconfiguration.
- No disruption to the contents of internal registers during reconfiguration, even in the area being reconfigured.

The latter point is of particular interest, because it allows one instantiation of a function to hand over data to the next function. For example, a group of registers may initially be configured to act as a binary counter. Then, at some time determined by the main system, the same registers may be reconfigured to operate as a *Linear Feedback Shift Register* (LFSR),⁵ whose seed value is determined by the final contents of the counter before it is reconfigured.

All of this would make it possible to "compile" new design variations in real-time, which may be thought of as dynamically creating subroutines in hardware! For example, imagine such a device performing *Digital Signal Processing* (DSP) tasks for a robotic vision system. The hardware could be reconfigured on-the-fly to accommodate changes in the data being processed, such as different lighting conditions or different algorithms for jungle versus urban environments.

Sad to relate, we don't have such components yet. Even if we did, we don't have software tools with the necessary levels of sophistication to configure them. And one can only imagine the problems associated with trying to debug

⁴ And I mean this in the loosest sense; I'm not talking about any FPGA fabric that is currently available on the market.

⁵ *Linear Feedback Shift Registers* (LFSRs) are introduced in detail in Appendix E: *Linear Feedback Shift Registers (LFSRs)*.

a system whose hardware is transmogrifying itself on-the-fly. Having said all this, a new class of device called an *Elemental Computing Array* (ECA) that was announced in 2007 may come close ...

ELEMENTAL COMPUTING ARRAYS (ECAs)

Ah, how I love the smell of freshly minted silicon chips in the morning. So I really wish I'd been present when the folks at Element CXI (<http://www.elementcxi.com/>) unveiled their first *Elemental Computing Array* (ECA) test chip in June 2007.

The folks at Element CXI say that this technology: "*delivers super-computer performance on battery power at consumer price points and is expected to be the new measuring stick for automotive, consumer electronics and communications devices.*" Brave words indeed, but how does this all work ...

Actually, it's difficult to know quite where to start, because there are so many aspects to this technology that it makes your head spin. So, in order to keep what little sanity I have left, I'm going to start at the bottom and work my way up. Conceptually, the lowest-level functional blocks in an ECA are known as *Elements*. There are currently seven types of Elements, which are divided into three main classes: computation, storage, and signaling (Figure 21.1).

Compute-Class Elements:

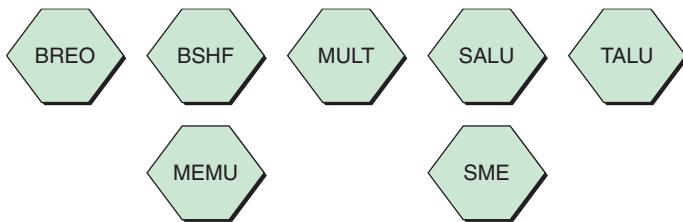


FIGURE 21.1

There are seven fundamental building blocks, called elements.

- **BREO:** The Bit RE-Orderer enables shifting, interleaving, packing, and unpacking operations and can be used for (un)packing, (de)interleaving, (de)puncturing, bit extraction, simple conditionals, etc.
- **BSHF:** The Barrel SHiFter enables shifting operations and can be used for 16-bit barrel shift, left shift, right shift, logical shift, arithmetic shift, concatenation, etc.
- **MULT:** A 16×16 signed and unsigned MULTIplier with an optional 32-bit accumulation stage.

- **SALU:** The Super ALU performs 16-bit and 32-bit arithmetic and logical functions, and can be used for sorts, compares, ANDs, ORs, XORs, ADDs, SUBs, ABS, masking, detecting leading 0's and leading 1's, etc.
- **TALU:** The Triple ALU enables up to three simultaneous logical and arithmetic functions with conditional execution. This little scallywag can be used for sorts, compares, ANDs, ORs, XORs, ADDs, SUBs, ABS, masking, detecting, etc.

Storage Class Elements:

- **MEMU:** The MEMory Unit provides random-access memory and sophisticated DAG (*Data Address Generation*) capabilities used for data storage.

Signaling Class Elements:

- **SME:** The State Machine Element is used to implement sequential code, operate as a coprocessor with other Elements, and operate as a "Virtual Element" for data-flow programs. The SME is a sequential processor, but—unlike traditional processors—it can be augmented by the other Elements in the same Cluster (we'll talk about Clusters in a moment). The SME is also used to implement the real-time operating system, run-time environment, housekeeping functions, test and resilience capabilities, and so forth.

Each Element has four 16-bit inputs and two 16-bit outputs (some Elements have the capability of ganging a pair of inputs or outputs together to perform 32-bit operations). Each input and output of an Element is queued, thereby isolating the Element from interconnect delays, and every Element executes an operation in one clock cycle.

Also, each Element can be reconfigured in a single clock cycle. A single instruction can be used to change the internal structure and function of an Element and to instruct it to operate on its data in a different way (you can reconfigure as many Elements as you wish simultaneously).

Furthermore, each Element can have eight *Contexts*; where each Context defines the way in which that element is configured and the function/task it will perform. This means that each Element is essentially eight Elements. Although only one of the Element's Contexts can execute on any given clock cycle, the other "virtual" Elements remain active by collecting data for operations in the Element input queue. When all required data is available for a "virtual" task or operation, the Element's control circuitry determines the correct configuration for the Element and fires off the task.

The next step up the hierarchy is the concept of a *Zone*, each of which comprises four Elements that are directly connected to each other via a *Crosspoint Switch* (CPS). The Elements in a Zone are tightly bound, communicating with each other in a single clock cycle. In turn, a *Cluster* is comprised of four Zones as illustrated in Figure 21.2.

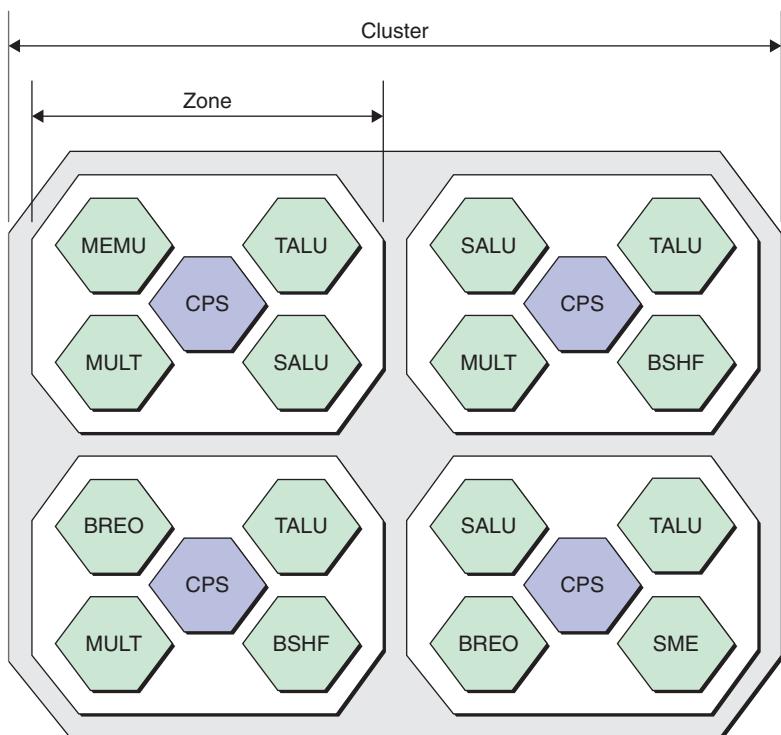


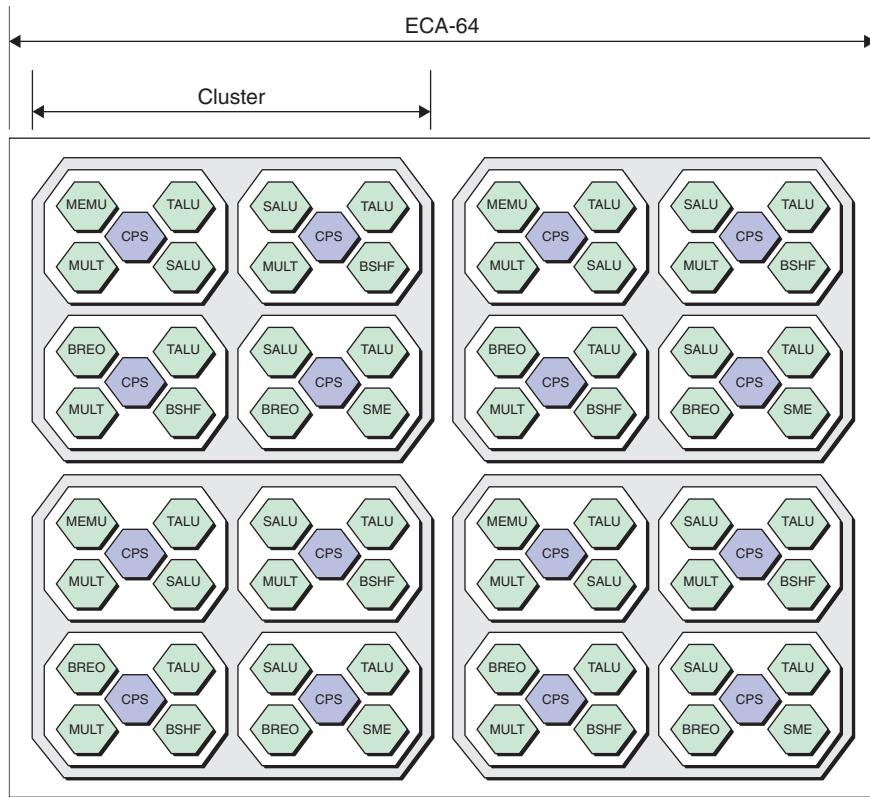
FIGURE 21.2

A Zone comprises four Elements; a Cluster comprises four Zones.

In the first ECA implementations, each Cluster contains only a single *MEMU* (Memory Unit) and a single *State Machine Element* (*SME*). The Cluster is the smallest repeatable structure on an *Elemental Computing Array* (ECA) chip. Thus, the smallest possible ECA would contain only a single Cluster. By comparison, the first production device (the ECA-64) comprises four identical Clusters, as illustrated in Figure 21.3.

All of the Zones in a Cluster communicate with each other using special queues. Up to 16 Clusters can be grouped together to form a *Super-Cluster*. Similarly, up to 16 Super-Clusters can be grouped together to form a *Matrix*.⁶

⁶Are you following all of this? I'll be asking questions later!

**FIGURE 21.3**

The first production device—the ECA-64—comprises four Clusters.

This method of interconnecting levels of hierarchy can be extended indefinitely on a single chip, bounded only by the available levels of integration and device fabrication. Furthermore, ECA devices communicate via PCI Express in the same hierarchical fashion, thereby extending the hierarchy to the board level.

When it comes to running applications on an ECA, the fabric is extremely flexible, allowing the various portions of a task to be distributed across computing Elements for maximum speed and parallelism. Alternatively, a task with lower requirements can be “folded” onto a smaller number of Elements (similar to the hardware design concept of “resource sharing”), thus time-sharing the Element with other portions of the same or other tasks.

Of particular interest is the fact that—from a programming view—a large ECA appears similar to a small ECA, while multiple ECAs on a board appear similar to a single ECA—the only difference is the amount of computing resources that are available.

And, when it comes to programming, applications are broken onto subsets of tasks that express maximum parallelism, and these tasks are then bound to free Elements (either statically or dynamically) and can be scaled across more Elements as they become available (or fewer Elements as additional applications are brought online).

The end result is that ECAs are claimed to offer performance equivalent to or greater than ASICs while consuming less power. This is possible because—in traditional ASIC implementations—each function occupies its own area of silicon real estate. This results in a significant waste of available resources, because only a limited number of functions are typically being exercised at any particular time. By comparison, a single, small ECA can be dynamically reconfigured “on-the-fly” to perform whatever applications are required at that particular time (higher priority tasks can be dynamically allocated more resources as required).

As regards to the actual speeds and feeds, I’m not allowed to talk about this because I’ve been sworn to secrecy. Suffice it to say that the numbers I’ve been shown (extreme performance while consuming minuscule amounts of power) made me squeak with delight!

OPTICAL INTERCONNECT

Electronics systems exhibit ever-increasing requirements to process ever-increasing quantities of data at ever-increasing speeds. Interconnection technologies based on conducting wires are fast becoming the bottleneck that limits the performance of electronic systems.

To relieve this communications bottleneck, a wide variety of optoelectronic interconnection techniques are undergoing evaluation. In addition to the extremely fast propagation of data,⁷ optical interconnects offer greater signal isolation, reduced sensitivity to electromagnetic interference, and a far higher bandwidth than do conducting wires.

Fiber-Optic Interconnect

The fibers used in fiber-optic systems are constructed from two different forms of glass (or other materials) with different refractive indices. These fibers, which are finer than a human hair, can be bent into weird and wonderful shapes without breaking. When light is injected into one end of the fiber, it repeatedly bounces

⁷Light travels at 299,792,458 meters per second in a vacuum. Thus, a beam of light would take approximately 2.6 seconds to make a round trip from the earth to the moon and back again!

off the interface between the two glasses, undergoing almost total internal reflection with minimal loss, until it reemerges at the other end (Figure 21.4).

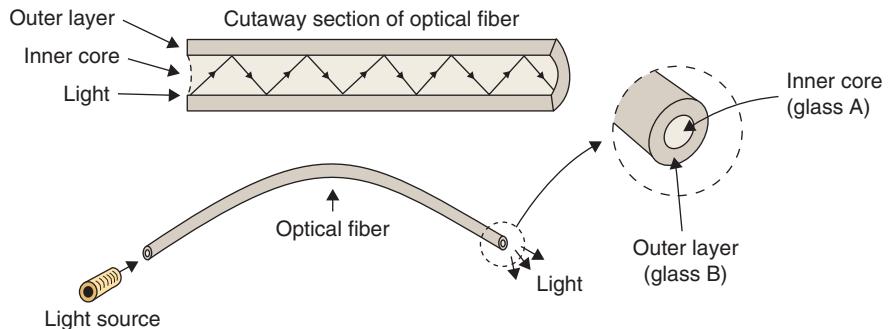


FIGURE 21.4
Light propagating through an optical fiber.

Experimental systems using fiber-optic interconnect have been evaluated at all levels of a system—for example, to link bare die in a *System-in-Package* (SiP)⁸ as illustrated in Figure 21.5.

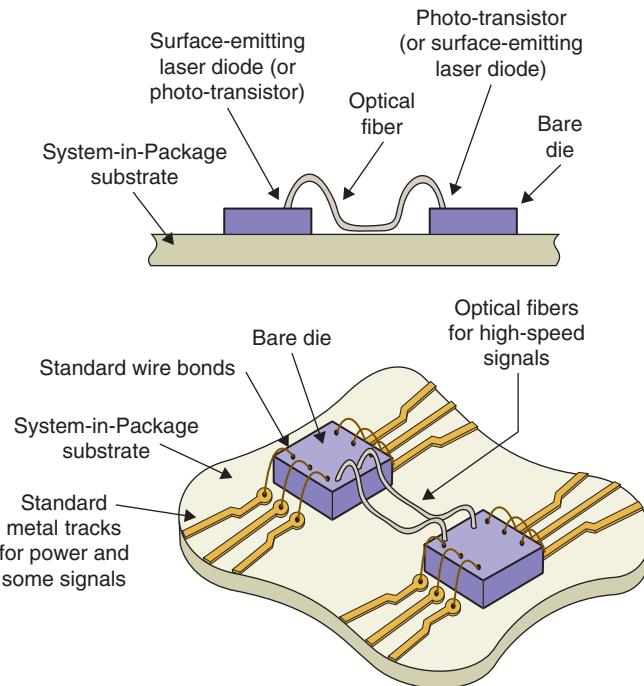


FIGURE 21.5

Using fiber optics to connect bare die in a System-in-Package (SiP).

⁸ *System-in-Package* (SiP) was introduced in Chapter 20: Advanced Packaging Techniques.

The transmitting device employs a surface-emitting laser-diode, which is constructed along with the transistors and other components on the integrated circuit's substrate. The receiving device uses a photo-transistor to convert the incoming light back into an electrical signal.

Each die can support numerous transmitters and receivers, which can be located anywhere on the surface of their substrates. However, there are several problems with this implementation, including the difficulty of attaching multiple optical fibers, the difficulties associated with repair and rework (replacing a defective die), and the physical space occupied by the fibers. Although the individual fibers are extremely thin, SiPs may require many thousands of connections. Additionally, in this form, each optical fiber can be used only to connect an individual transmitter to an individual receiver.

As another alternative, optical fibers may be used to provide intra-board connections, which may be referred to as *optical backplanes*. In this case, the daughter boards may be mounted in a rack without an actual backplane, and groups of optical fibers may be connected into special couplers (Figure 21.6).

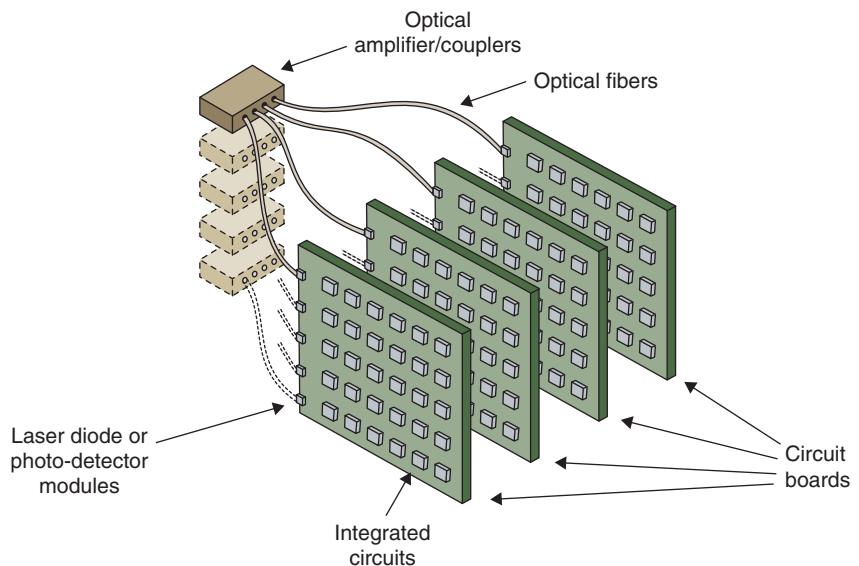


FIGURE 21.6

Using fiber optics to implement an optical backplane.

Each optical fiber from a transmitter is connected into a coupler, which amplifies the optical signal and can retransmit it to multiple receivers. This form of backplane offers great latitude in regard to the proximity of the boards. In fact, boards connected in this way can be separated by as much as tens of meters.

It Pays to Keep Your Eyes Open

In 1834, the Scottish scientist John Scott Russell was observing a barge being pulled along a canal by a pair of horses. When the barge stopped, he noticed that the bow wave continued forward without appearing to deteriorate in any way. Most of us wouldn't have paid this any attention, but Russell jumped on a horse and followed the wave for miles.

What Russell had observed was a special form of wave called a *soliton* that, due to its contour, can retain its shape and speed. One example of a soliton is the Severn bore: a wave caused by unusual tidal conditions (where the Severn is a river that separates England and Wales). Thus a bore (wave) can travel miles up the river without significant diminishment or deformation.

The reason this is of interest is that light waves become distorted and attenuated as they travel through optical fibers—similar to the way in which electrical pulses deteriorate as they propagate through conducting wires. Scientists and engineers are now experimenting with light solitons—pulses that can travel long distances through optical fibers without distortion.

Free-Space Interconnect

In the case of the *free-space* technique, a laser-diode transmitter communicates directly with a photo-transistor receiver without employing an optical fiber. Consider a free-space technique used to link bare die mounted on the substrate of a SiP (Figure 21.7).

In this case, the transmitters are constructed as side-emitting laser diodes along the upper edges of the die; similarly with the photo-transistors on the receivers. Each die may contain multiple transmitters and receivers. The free-space technique removes some of the problems associated with its fiber-optic equivalent: there are no fibers to attach and replacing a defective die is easier.

However, as for optical fibers, each transmitter can still be used only to connect to an individual receiver. Additionally, the free-space technique has its own unique problems: the alignment of the devices is critical, and there are thermal tracking issues. When a laser-diode is turned on, it rapidly cycles from ambient temperature to several hundred degrees Celsius. The heat generated by an individual laser-diode does not greatly affect the die because each diode is so small. However, the cumulative effect of hundreds of such diodes does affect the die, causing it to expand, thereby disturbing the alignment of the transmitter-receiver pairs.

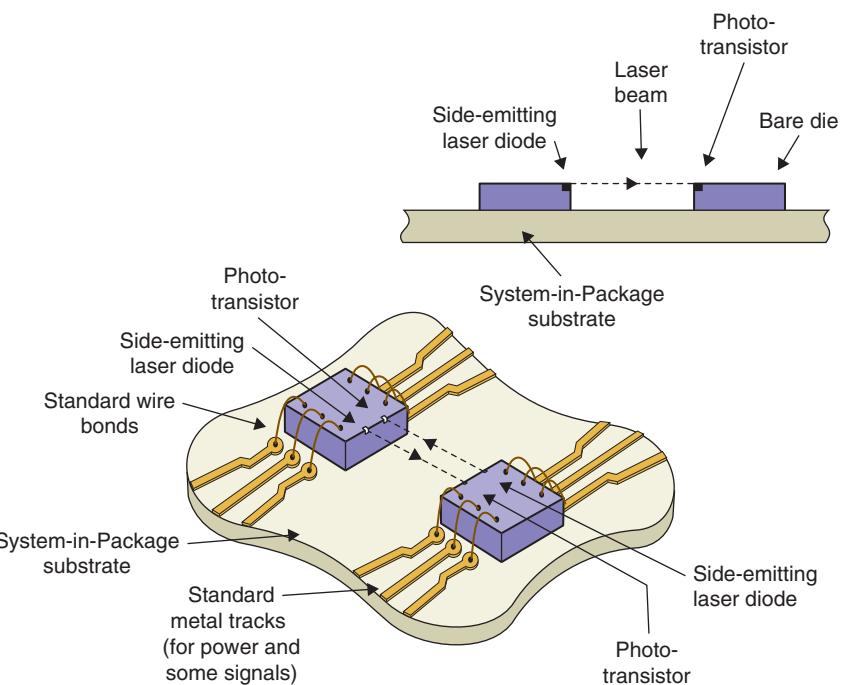


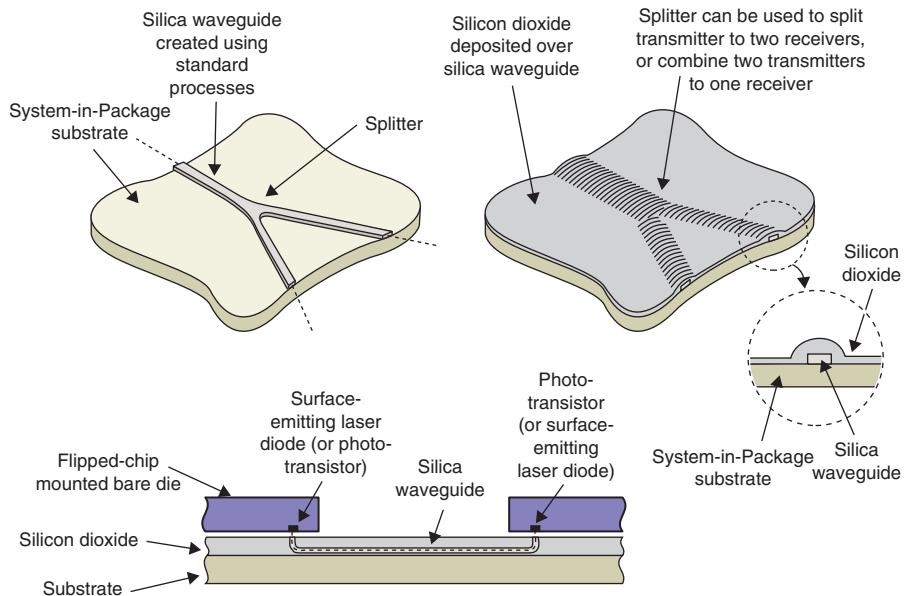
FIGURE 21.7
Free space interconnect.

Guided-Wave Interconnect

Another form of optical interconnect that is receiving significant interest is that of *guided-wave*, whereby optical waveguides are fabricated directly on the substrate of a *System-in-Package* (SiP). These waveguides can be created using variations on standard opto-lithographic thin-film processes. One such process involves the creation of silica waveguides (Figure 21.8).⁹

Using a flipped-chip mounting technique, the surface-emitting laser-diodes and photo-transistors on the component side of the die are pointed down toward the substrate. One major advantage of this technique is that the waveguides can be constructed with splitters, thereby allowing a number of transmitters to drive a number of receivers. On the down side, it is very difficult to route one waveguide over another, because the crossover point tends to act like a splitter and allows light from the waveguides to “leak” into each other.

⁹Thanks for the information on silica waveguides go to Dr. Terry Young of the GEC-Marconi Research Center, Chelmsford, Essex, England, with apologies for all the extremely technical details that were omitted here.

**FIGURE 21.8**

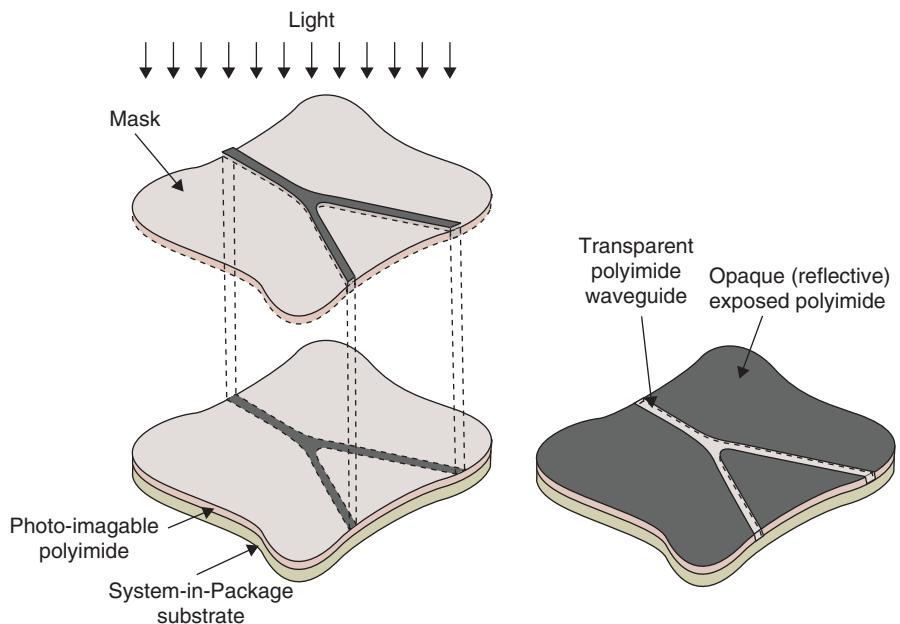
Guided-wave interconnect: Silica waveguides.

An alternative form of waveguide technology, *photo-imagable polyimide interconnect*, which was announced toward the tail-end of 1993, is of particular interest in the case of a *System-in-Package* (SiP) based on a ceramic, glass, or metal substrate that is covered with a layer of dielectric material such as polyimide. The dielectric coat is used to modify the substrate's capacitive characteristics, and tracks are created on the surface of the dielectric using thin-film processes.

When exposed to light passed through an appropriate mask, a layer of *photo-imagable polyimide* can be imprinted with patterns in a similar way to exposing a photograph. After being developed, the polyimide contains low-loss optical waveguides bounded by relatively opaque reflective surfaces (Figure 21.9).

Apart from its inherent simplicity, one of the beauties of this technique is that both the exposed and unexposed areas of polyimide have almost identical dielectric constants. Thus, in addition to leveraging off existing technology, the polyimide waveguides have relatively little impact on any thin-film metallization tracking layers that may be laid over them.

This technique is currently finding its major audience in designers of *System-in-Packages* (SiPs), but it is also being investigated as a technique for multilayer circuit boards. In the future, circuit boards could be fabricated as a mixture of traditional copper interconnect and very high speed optical interconnect.

**FIGURE 21.9**

Guided-wave interconnect: Photo-imaginable polyimide waveguides.

OPTICAL MEMORIES

When I sat down to pen the first edition of this tome circa 1992, it was said that the total sum of human knowledge was doubling approximately every ten years. Now, in 2008, many references say it's doubling every five years.¹⁰ Coupled with this, the amount of information that is being generated, stored, and accessed is increasing at an exponential rate. This is driving the demand for fast, cheap memories that can store gigabits (a billion bits), terabits (a thousand billion bits), or even petabits (a million billion bits) of data.

One medium with the potential to cope with this level of data density is *optical storage*. Among many other techniques, evaluations are being performed on extremely thin layers of glass-based materials,¹¹ which are doped with organic dyes or rare-earth elements. Using a technique known as *Photochemical Hole-Burning* (PHB), a laser in the visible waveband is directed at a microscopic point on the surface of the glass (Figure 21.10).

If the laser is weak, its light will pass through the glass without affecting it and reappear at the other side. If the laser is stronger (but not intense enough to

¹⁰There's no wonder I can't keep up!

¹¹One such material, boric-acid glass, is also widely used in heat-resistant kitchenware!

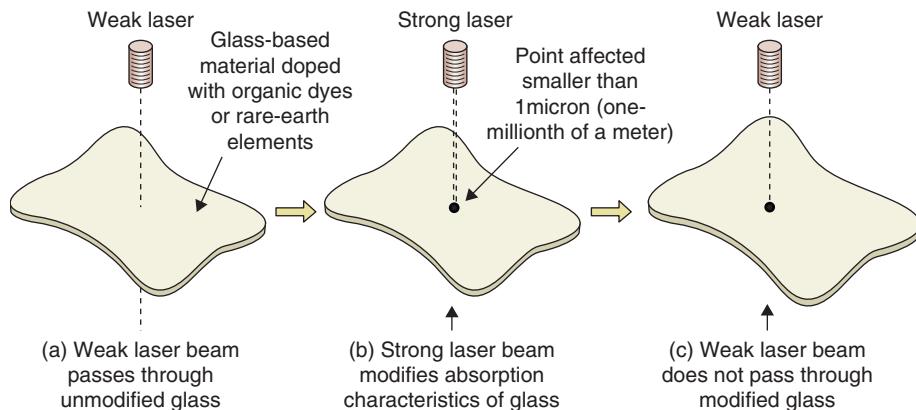


FIGURE 21.10
Optical memories:
Photochemical hole
burning.

physically damage the glass), electrons in the glass will be excited by the light. The electrons can be excited such that they change the absorption characteristics of that area of the glass and leave a band, or hole, in the absorption spectrum. To put this another way, if the weak laser beam is redirected at the same point on the glass surface, its light would now be absorbed and would not reappear at the other side of the glass.

Thus, depending on whether or not the light from the weaker beam passes through the glass, it can be determined whether or not that point has been exposed to the strong laser. This means that each point can be used to represent a binary 0 or 1. Because the point affected by the laser is so small, this process can be replicated millions upon millions of times across the surface of the glass.

If the points occur at one-micron (one-millionth of a meter) intervals, then it is possible to store 100 megabits per square centimeter, but this still does not come close to the terabit storage that will be required. However, it turns out that each point can be “multiplexed” and used to store many bits of information. A small change in the wavelength of the laser can be used to create another hole in a different part of the spectrum. In fact, 100X multiplexing has been achieved, where each point on the glass was used to store 100 bits of data at different wavelengths. Using 100X multiplexing offers a data density of 10 gigabits per square centimeter, and even higher levels of multiplexing may be achieved in the future!

PROTEIN SWITCHES AND MEMORIES

Another area receiving a lot of interest is that of switches and memories based on proteins. We should perhaps commence by pointing out that this concept doesn't imply anything gross like liquidizing hamsters to extract their proteins! Out of

all the elements nature has to play with, only carbon, hydrogen, oxygen, nitrogen, and sulfur are used to any great extent in living tissues, along with the occasional smattering of phosphorous, minuscule portions of a few choice metals, and elements like calcium for bones. The basic building blocks of living tissues are twenty or so relatively simple molecules called amino acids. For example, consider three of the amino acids called threonine, alanine, and serine (Figure 21.11).

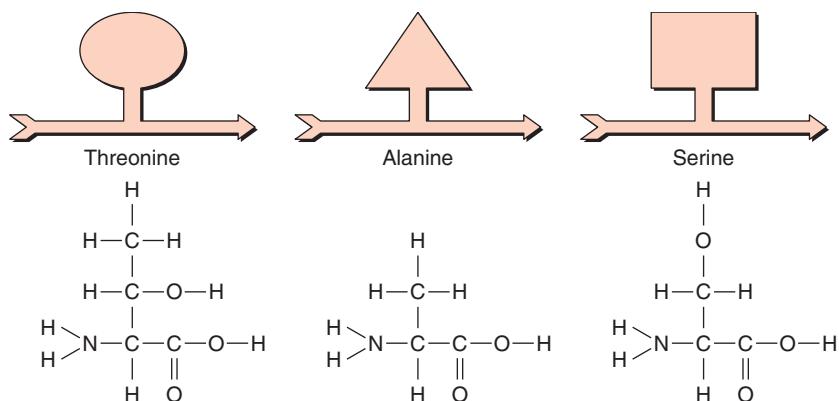


FIGURE 21.11

Threonine, alanine, and serine are three of the twenty or so biological building blocks called amino acids.

These blocks can join together to form chains, where the links between the blocks are referred to as *peptide bonds*, which are formed by discarding a water molecule (H_2O) from adjacent COOH and NH_2 groups (Figure 21.12).

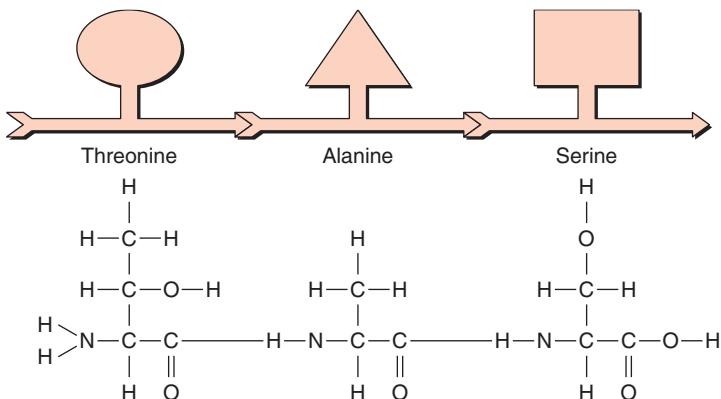


FIGURE 21.12

Amino acids can link together using peptide bonds to form long polypeptide chains.

Proteins consist of hundreds or thousands of such chains of amino acids. Note that the distribution of electrons in each amino acid varies depending on the size of that acid's constituent atoms, leaving areas that are slightly more

positively or negatively charged (similar to a water molecule, as is discussed later in this chapter). The linear chain shown in Figure 21.12 is known as the *primary structure* of the protein, but this chain subsequently coils up into a spring-like helix, whose shape is maintained by the attractions between the positively and negatively charged areas in the chain. This helix is referred to as the protein's *secondary structure*, but there's more, because the entire helix subsequently "folds" up into an extremely complex three-dimensional structure, whose shape is once again determined by the interactions between the positively and negatively charged areas on the helix. Although this may seem to be arbitrarily random, this resulting *tertiary structure* represents the lowest possible energy level for the protein, so proteins of the same type always fold up into identical (and stable) configurations.

Organic molecules have a number of useful properties, not the least of which is that their structures are intrinsically "self healing" and reject contamination. Also, in addition to being extremely small, many organic molecules have excellent electronic properties.

In the case of certain proteins, it's possible to coerce an electron to move to one end of the protein or the other, where it will remain until it's coerced back again (of course, the term "end" is somewhat nebulous in this context). Thus, a protein of this type can essentially be used to store and represent a logic 0 or a logic 1 value based on the location of this electron.¹² Similarly, it's possible for some protein structures to be persuaded to act in the role of switches.

In the case of traditional semiconductor-based transistors, even when one considers structures measured in fractions of a millionth of a meter, each transistor consists of millions of atoms. By comparison, protein-based switches and registers can be constructed using a few thousand atoms, which means that they are thousands of times smaller, thousands of times faster, and consume a fraction of the power of their semiconductor counterparts.

Unlike metallic conductors, some proteins transfer energy by moving electron excitations from place to place rather than relocating entire electrons. This can result in switching speeds that are orders of magnitude faster than their semiconductor equivalents.

¹²In the case of some proteins, rather than physically moving an electron from one "end" to the other, it's possible to simply transfer an excitation from one electron to another. This requires far less power and occurs much faster than moving the electron itself, but it's a little too esoteric a concept to explore in detail here.

Some proteins react to electric fields, while others respond to light. For example, there is a lot of interest in the protein *Rhodopsin*, which is used by certain photosynthetic bacteria to convert light into energy. The bacteria that contain Rhodopsin are the ones that cause ponds to turn red, and their saltwater cousins are responsible for the purple tint that is sometimes seen in San Francisco Bay.

In certain cases, light from a laser can be used to cause such optically responsive proteins to switch from one state to another (which they do by changing color) and back again. Additionally, some varieties of proteins are only responsive to the influence of two discrete frequencies. This feature is extremely attractive, because it offers the possibility of three-dimensional optical protein memories.

Experiments have been performed in which 3-D cubes have been formed as ordered arrays of such bi-frequency proteins suspended in transparent polymers. If the protein were affected by a single laser, then firing a beam into the cube would result in a line of proteins changing state. But in the case of bi-frequency proteins, two lasers mounted at 90 to each other can be used to address individual points in the 3-D space (Figure 21.13).

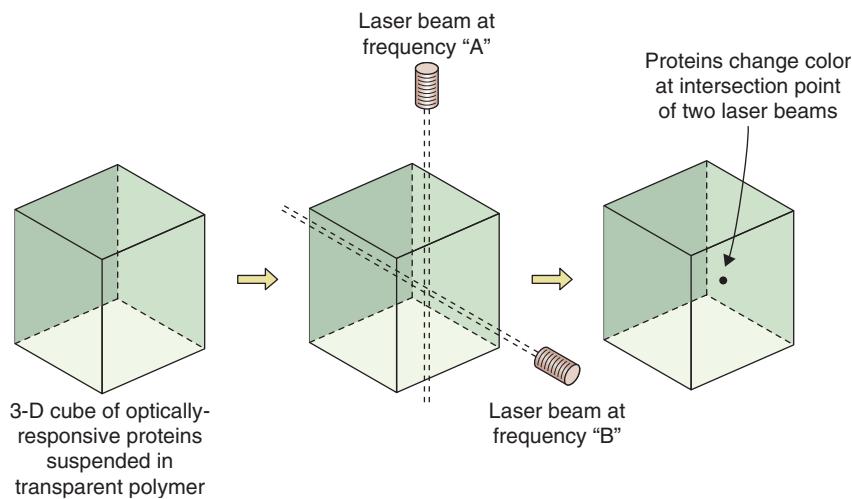


FIGURE 21.13

Protein memories:
Cubic arrays of light-sensitive proteins.

By only slightly enhancing the technology available today, it may be possible to store as much as 30 gigabits in a $1\text{ cm} \times 1\text{ cm} \times 1\text{ cm}$ cube of such material!

ELECTROMAGNETIC TRANSISTOR FABRICATION

For some time it has been known that the application of strong electromagnetic fields to special compound semiconductors can create structures that behave like

transistors. The original technique was to coat the surface of a semiconductor substrate with a layer of dopant material, and then to bring an extremely strong, concentrated electromagnetic field into close proximity.

The theory behind this original technique was that the intense field caused the *electromigration* of the dopant into the substrate. However, much to everyone's surprise, it was later found that this process remained effective even without the presence of the dopant!

Strange as it may seem, nobody actually understands the mechanism that causes this phenomenon. Physicists currently suspect that the strong electromagnetic fields cause microscopic native defects in the crystals to migrate through the crystal lattice and cluster together.

HETEROJUNCTION TRANSISTORS

If there is one truism in electronics, it is that "*faster is better*," and a large proportion of research and development funds are invested in increasing the speed of electronic devices.

Ultimately, there are only two fundamental ways to increase the speed of semiconductor devices. The first is to reduce the size of the structures on the semiconductor, thereby obtaining smaller transistors that are closer together. The second is to use alternative materials that inherently switch faster. However, although there are a variety of semiconductors, such as *gallium arsenide* (GaAs), that offer advantages over silicon for one reason or another, silicon is cheap, readily available, and relatively easy to work with. Additionally, the electronics industry has billions of dollars invested in silicon-based processes.

For these reasons, speed improvements have traditionally been achieved by making transistors smaller. However, some pundits believe that we are reaching the end of this route using conventional technologies. At one time, the limiting factors appeared to be simple process limitations: the quality of the resist, the ability to manufacture accurate masks, and the features that could be achieved with the wavelength of ultraviolet light. Around 1990, when structures with dimensions of 1.0 microns first became available, it was believed that structures of 0.5 microns would be the effective limit that could be achieved with opto-lithographic processes, and that the next stage would be a move to X-ray lithography. However, there have been constant improvements in the techniques associated with mask fabrication, optical systems and lenses, servo motors, and positioning systems, and advances in chemical engineering such

as chemically amplified resists.¹³ The combination of all these factors means that it is currently possible to achieve structures as small as 32 nanometers by continuing to refine existing processes.

However, there are other considerations. The speed of a transistor is strongly related to its size, because it affects the distance electrons have to travel. Thus, to enable transistors to switch faster, technologists have concentrated on reducing their size, a strategy that is commonly referred to as *scaling*. However, while scaling reduces the size of structures, it is necessary to maintain certain levels of dopants to achieve the desired effect. This means that, as the size of the structures is reduced, it is necessary to increase the concentration of dopant atoms. Increasing the concentration beyond a certain level causes leakage, resulting in the transistor being permanently ON, and therefore useless. Thus, technologists are increasingly considering alternative materials and structures.

An interface between two regions of semiconductor having the same basic composition but opposing types of doping is called a *homojunction*. By comparison, the interface between two regions of dissimilar semiconductor materials is called a *heterojunction*. Homojunctions dominate current processes because they are easier to fabricate. However, the interface of a heterojunction has naturally occurring electric fields, which can be used to accelerate electrons, and transistors created using heterojunctions can switch much faster than their homojunction counterparts of the same size.

One form of heterojunction that is attracting a lot of interest is found at the interface between silicon and germanium. Silicon and germanium are in the same family of elements and have similar crystalline structures. In theory this should make it easy to combine them, but it's a little more difficult in practice. One technique is to create a standard silicon wafer with doped regions, and then to grow extremely thin layers of a silicon-germanium alloy where required.

The two most popular methods of depositing these layers are *Chemical Vapor Deposition* (CVD) and *Molecular Beam Epitaxy* (MBE). In the case of chemical vapor deposition, a gas containing the required molecules is converted into a plasma¹⁴ by heating it to extremely high temperatures using microwaves. The plasma carries atoms to the surface of the wafer where they are attracted to

¹³In the case of a chemically amplified resist, the application of a relatively small quantity of ultraviolet light stimulates the formation of chemicals in the resist, which accelerates the degrading process. This reduces the amount of ultraviolet light required to degrade the resist and allows the creation of finer features with improved accuracy.

¹⁴A gaseous state in which the atoms or molecules are dissociated to form ions.

the crystalline structure of the substrate. This underlying structure acts as a template. The new atoms continue to develop the structure to build up a layer on the substrate's surface (Figure 21.14).

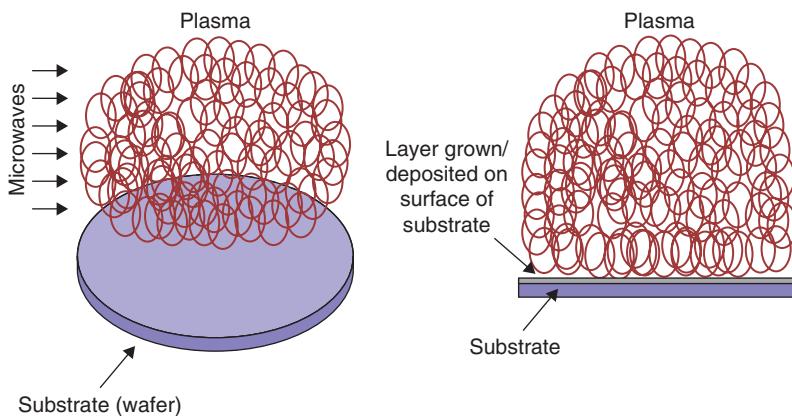


FIGURE 21.14

Heterojunction transistors: Chemical vapor deposition (CVD).

By comparison, in the case of molecular beam epitaxy, the wafer is placed in a high vacuum, and a guided beam of ionized molecules is fired at it, effectively allowing molecular-thin layers to be “painted” onto the substrate where required.¹⁵

Ideally, such a heterojunction would be formed between a pure silicon substrate and a pure layer of germanium. Unfortunately, because germanium atoms are approximately 4% larger than silicon atoms, the resulting crystal lattice cannot tolerate the strains that develop, which results in defects in the structure. In fact, millions of minute inclusions occur in every square millimeter, which prevents the chip from working. The solution is in growing a layer of silicon-germanium alloy, which relieves the stresses in the crystalline structure, thereby preventing the formation of inclusions (Figure 21.15).

Heterojunctions offer the potential to create transistors that switch as fast, or faster, than those formed using gallium arsenide, but use significantly less power. Additionally, they have the advantage of being able to be produced on existing fabrication lines, thereby preserving the investment and leveraging current expertise in silicon-based manufacturing processes.

¹⁵ Molecular Beam Epitaxy (MBE) is similar to Electron Beam Epitaxy (EBE), in which the wafer is first coated with a layer of dopant material before being placed in a high vacuum. A guided beam of electrons is then fired at the wafer causing the dopant to be driven into it.

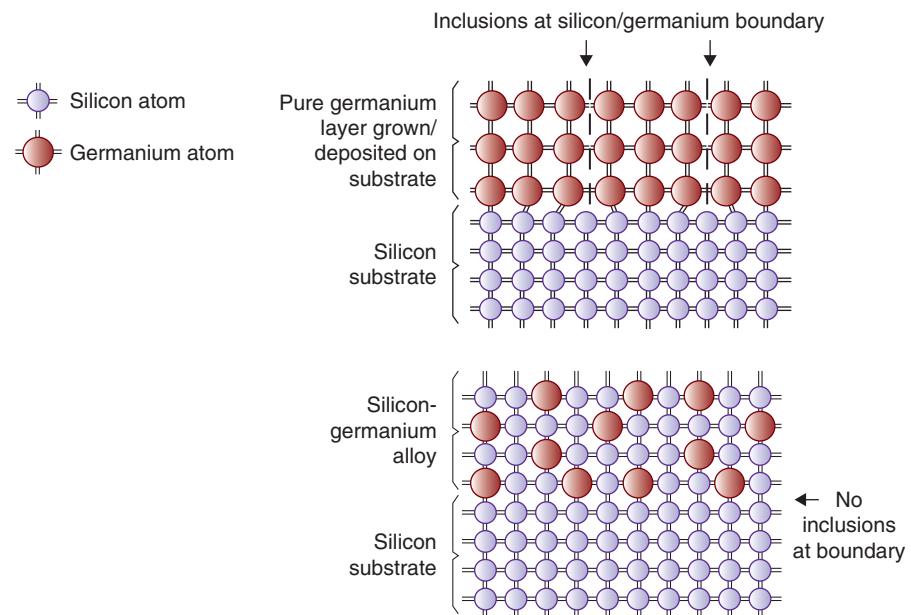


FIGURE 21.15
Heterojunction transistors: Depositing a layer of pure germanium versus a layer of silicon-germanium alloy.

BUCKYBALLS AND NANOTUBES

Prior to the mid-1980s, the only major forms of pure carbon known to us were graphite and diamond. In 1985, however, a third form consisting of spheres formed from 60 carbon atoms (C_{60}) was discovered. Officially known as Buckminsterfullerene¹⁶—named after the American architect R. Buckminster Fuller, who designed geodesic domes with the same fundamental symmetry—these spheres are more commonly known as “buckyballs.” In 2000, scientists with the U.S. Department of Energy’s Lawrence Berkeley National Laboratory (Berkeley Lab) and the University of California at Berkeley reported that they had managed to fashion a transistor from a single buckyball.

Sometime later, scientists discovered another structure called the *nanotube*, which is like taking a thin sheet of carbon and rolling it into a tube. Nanotubes can be formed with walls that are only one atom thick. The resulting tube has a diameter of 1 nanometer (one-thousandth of one-millionth of a meter).

Nanotubes are an almost ideal material. They are stronger than steel, have excellent thermal stability, and they are also tremendous conductors of heat and electricity. For example, a conductor formed from millions of nanotubes arranged to form a 1-cm cross-section could conduct more than 1 billion amps.

¹⁶ *Science* magazine voted Buckminsterfullerene the “Molecule of the Year” in 1991.

In addition to acting as wires, nanotubes can be persuaded to act as transistors; this means that we theoretically have the ability to replace silicon transistors with molecular-sized equivalents at a level where standard semiconductors cease to function. We're only just starting to experiment with nanotube-based transistors, but they can theoretically run at clock speeds of one terahertz or more, which is orders of magnitude faster than today's transistors.

In the longer-term future, it may be that some integrated circuits will use nanotubes to form both the transistors and the wires linking them together. In the shorter-term, we can expect to see interesting mixtures of nanotubes with conventional technologies. For example, in the summer of 2002, IBM announced an experimental transistor called a *Carbon Nanotube FET* (CNFET). This device is based on a field-effect transistor featuring a carbon nanotube acting as the channel (the rest of the transistor is formed using conventional silicon and metal processing technologies). It will be a number of years before these devices become commercially available, but they are anticipated to significantly outperform their silicon-only counterparts.

In the meantime ... nanotubes continue to find many diverse applications. For example, consider color television screens and computer displays. For many years, these were predominantly based on *Cathode Ray Tube* (CRT) technology. More recently, new technologies such as plasma displays and *Liquid Crystal Display* (LCD) screens have become popular, because they are thinner, lighter, and use less power. However plasma displays are expensive and tend to fade over time, while LCDs tend to "wash out" if there's too much ambient light.

Now, returning to nanotubes, of particular interest to us here is the fact that they can be coerced into emitting streams of electrons out of one end. Hmm, tiny little electron guns; what wonders could we perform with these little rascallions? Well, remember that each tiny picture element (pixel) on the screen is formed from three sub-pixels (red, green, and blue). Now, imagine a screen that's thin and flat like a LCD, but is as bright and vibrant as a CRT-based display; that's what you end up with if the screen is formed from a carbon nanotube-based *Surface Emission Display* (SED), as illustrated in Figure 21.16.

As we see, the inside of the screen is covered with red, green, and blue phosphor dots (one of each to form each pixel), and each of these dots has its own carbon nanotube electron gun. This technology has been skulking around in the background for some time, but it appears as though the outstanding issues that had been holding it back have been resolved, and SEDs are poised to leap onto the center stage.

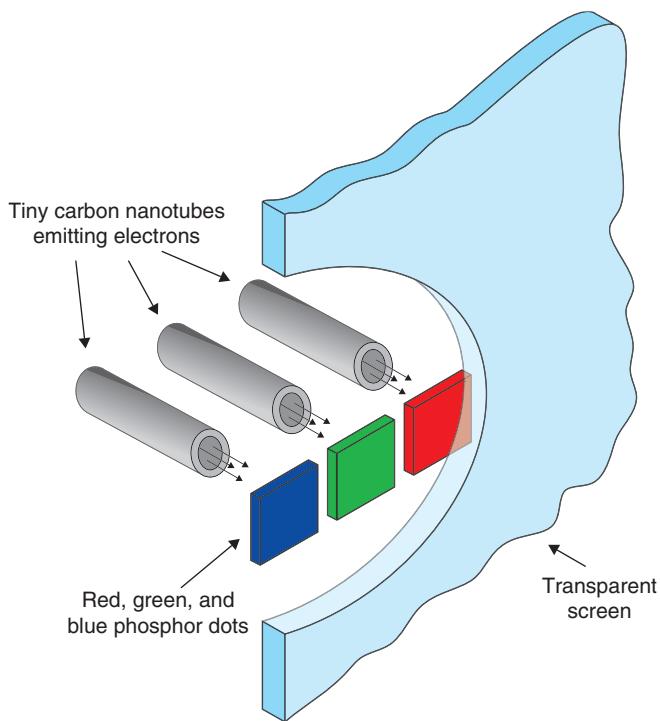


FIGURE 21.16
A single pixel in a Surface Emission Display (SED).

Toshiba hosted the first public demonstration of a large-scale carbon nanotube-based SED at the *Consumer Electronics Show* (CES) in January 2006. Industry expert Dennis P. Barker attended the show, and he later told me (Max):

“High-definition television is incredibly realistic, but SED goes one step beyond. When I saw the Toshiba demonstration, it gave me chills and the hairs on the back of my neck stood to attention. I have seen the future and—to me—the future is SED!”

But wait, there’s more ... because in the early part of 2008, folks started to get really excited about yet another form of carbon called *graphene*, which involves carbon atoms bound in a network of repeating hexagons within a single plane just one atom thick. Not only is a sheet of graphene the thinnest of materials, it’s also immensely strong and conducts electrons faster at room temperature than any other substance known to man. We’re only just starting to explore graphene, but it appears to offer tremendous potential for new electronic devices, such as so-called *ballistic transistors* that could switch much faster than current devices.

DIAMOND SUBSTRATES

As we have noted on several occasions, there is a constant drive towards smaller, more densely packed transistors switching at higher speeds. Unfortunately, packing the little devils closer together and cracking the whip to make them work faster substantially increases the amount of heat that they generate. Similarly, the increasing utilization of optical interconnect relies on the use of laser diodes, but today's most efficient laser diodes only convert 30% to 40% of the incoming electrical power into an optical output, while the rest emerges in the form of heat. Although each laser diode is relatively small (perhaps as small as only 500 atoms in diameter), their heating effect becomes highly significant when tens of thousands of them are performing their version of *Star Wars*.

And so we come to *diamond*, which derives its name from the Greek *adamas*, meaning "invincible." Diamond is famous as the hardest naturally occurring substance known, but it also has a number of other interesting characteristics: it is a better conductor of heat at room temperatures than any other material,¹⁷ in its pure form it is a good electrical insulator, it is one of the most transparent materials available, and it is extremely strong and noncorrosive. For all of these reasons, diamond would form an ideal substrate material for *System-in-Packages* (SiPs).¹⁸

In addition to SiPs, diamond has potential for a variety of other electronics applications. Because diamond is in the same family of elements as silicon and germanium, it can function as a semiconductor and could be used as a substrate for integrated circuits. In fact, in many ways, diamond would be far superior to silicon: it is stronger, it is capable of withstanding high temperatures, and it is relatively immune to the effects of radiation (the bane of components intended for nuclear and space applications). Additionally, due to diamond's high thermal conductivity, each die would act as its own heat sink and would rapidly conduct heat away. It is believed that diamond-based devices could switch up to 50 times faster than silicon, and operate at temperatures over 500°C.

Chemical Vapor Deposition

Unfortunately, today's integrated circuit manufacturing processes are geared around fabricating large numbers of chips on wafers that can be up to 300 mm

¹⁷ Diamond can conduct five times as much heat as copper, which is the second most thermally conductive material known.

¹⁸ As we mentioned in *Chapter 20: Advanced Packaging Techniques*, other exotic substrates are also of interest to electronic engineers, including sapphire, which is of particular use in microwave applications.

in diameter (with 450 mm diameter wafers on the horizon). By comparison, a natural diamond 10 mm in diameter would be considered to be really, REALLY large. It simply wouldn't be cost-effective to take one of these beauties, slice it up, and make diamond-based chips one at a time. Furthermore, if you were to be the proud owner of a large natural diamond, the last thing that would come to mind would be to chop it up into thin slices for electronics applications!

However, there are a number of methods for depositing or growing diamond crystals, one of the most successful being *Chemical Vapor Deposition* (CVD), which was introduced in the earlier discussions on heterojunction transistors. With this CVD process, microwaves are used to heat mixtures of hydrogen and hydrocarbons into a plasma, out of which diamond films nucleate and form on suitable substrates. Although the plasma chemistry underlying this phenomena is not fully understood, polycrystalline diamond films can be nucleated on a wide variety of materials, including metals such as titanium, molybdenum, and tungsten, ceramics, and other hard materials such as quartz, silicon, and sapphire.

Chemical Vapor Infiltration

CVD processes work by growing layers of diamond directly onto a substrate. A similar, more recent technique—known as *Chemical Vapor Infiltration* (CVI)¹⁹—commences by placing diamond powder in a mold. Additionally, thin posts, or columns, can be preformed in the mold, and the diamond powder can be deposited around them. When exposed to the same plasma as used in the CVD technique, the diamond powder coalesces into a polycrystalline mass. After the CVI process has been performed, the posts can be dissolved, leaving holes through the diamond for use in creating vias. CVI processes can produce diamond layers twice the thickness of those obtained using CVD techniques at a fraction of the cost.

Ubiquitous Laser Beams

An alternative, relatively new technique for creating diamond films involves heating carbon in a vacuum using laser beams. Focusing the lasers on a very small area generates extremely high temperatures, which rip atoms away from the carbon and also strip away some of their electrons. The resulting ions fly off and stick to a substrate placed in close proximity. Because the lasers are tightly focused, the high temperatures they generate are localized on the carbon,

¹⁹Thanks go to Crystallume, Menlo Park, CA, USA, for the information on their CVD and CVI processes.

permitting the substrate to remain close to room temperature. Thus, this process can be used to create diamond films on almost any substrate, including semiconductors, metals, and plastics.

The number of electrons stripped from the carbon atoms varies, allowing their ions to reform in *nanophase diamond structures* that have never been seen before. Nanophase materials are a new form of matter that was only discovered relatively recently, in which small clusters of atoms form the building blocks of a larger structure. These structures differ from those of naturally occurring crystals, in which individual atoms arrange themselves into a lattice. In fact, it is believed that it may be possible to create more than thirty previously unknown forms of diamond using these techniques.

The Maverick Inventor

In the late 1980s, a maverick inventor called Ernest Nagy²⁰ invented a simple, cheap, and elegant technique for creating thin diamond films. Nagy's process involves treating a soft pad with diamond powder, spinning the pad at approximately 30,000 revolutions per minute, and maintaining the pad in close contact with a substrate. Although the physics underlying the process is not fully understood, diamond is transferred from the pad to form a smooth and continuous film on the substrate. The diamond appears to undergo some kind of phase transformation, changing from a cubic arrangement into a hexagonal form with an unusual structure. Interestingly enough, Nagy's technique appears to work with almost any material, on almost any substrate!

The Requirement for Single-Crystal Diamond

All of the techniques described above result in films that come respectfully close, if not equal, to the properties of natural diamond in characteristics such as heat conduction. Thus, these films are highly attractive for use as substrates in SiPs. However, the unusual diamond structures that are created fall short of the perfection required for them to be used as a substrate suitable for the fabrication of transistors.

Substrates for integrated circuits require the single, large crystalline structures found only in natural diamond. Unfortunately, there are currently no known materials onto which a single-crystal diamond layer will grow, with the exception of single crystal diamond itself (which sort of defeats the point of doing it in the

²⁰Nagy, whose full name is Ernest Nagy de Nagybaczon, was born in 1942 in Hungary. He left as a refugee in the 1956 uprising and moved to England.

first place). The only answer appears to be to modify the surface of the substrate onto which the diamond layer is grown, and many observers believe that this technology may be developed in the near future. If it does prove possible to create consistent, single-crystal diamond films, then, in addition to being “*a girl’s best friend*,” diamonds would also become “*an electronic engineer’s biggest buddy*.”

CONDUCTIVE ADHESIVES

Many electronics fabrication processes are exhibiting a trend towards mechanical simplicity with underlying sophistication in materials technology. A good example of this trend is illustrated by conductive, or *anisotropic*, adhesives, which contain minute particles of conductive material.

These adhesives find particular application with the flipped-chip techniques used to mount bare die on the substrates of hybrids, SiPs, or circuit boards. The adhesive is screen-printed onto the substrate at the site where the die is to be located, the die is pressed into the adhesive, and the adhesive is cured using a combination of temperature and pressure (Figure 21.17).

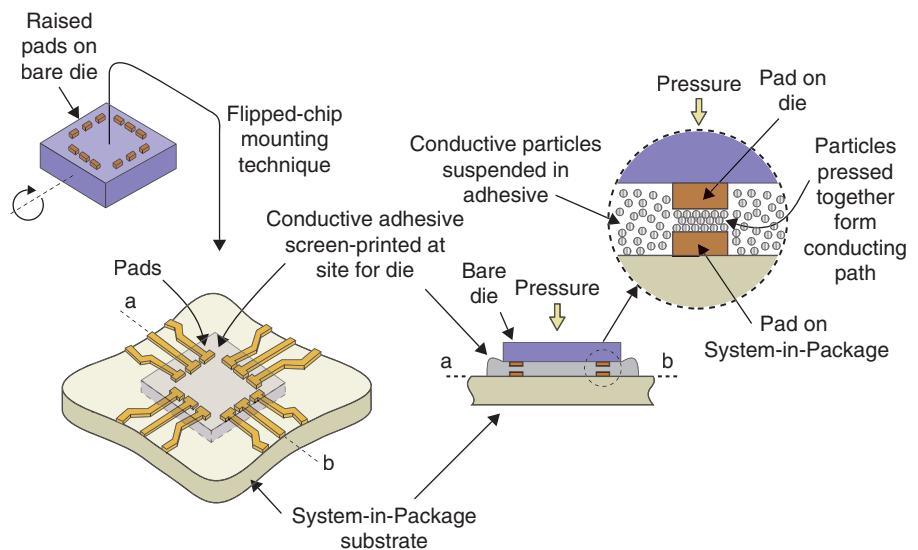


FIGURE 21.17
Conductive adhesives.

The beauty of this scheme is that the masks used to screen print the adhesive do not need to be too complex, and the application of the adhesive does not need to be excessively precise, because it can be spread across all of the component pads. The conducting particles are only brought in contact with each

other at the sites where the raised pads on the die meet their corresponding pads on the substrate, thereby forming good electrical connections.

The original conductive adhesives were based on particles such as silver. But, in addition to being expensive, metals like silver can cause electron migration problems at the points where they meet the silicon substrates. Modern equivalents are based on organic metallic particles, thereby reducing these problems.

In addition to being simpler and requiring fewer process steps than traditional methods, the conductive adhesive technique removes the need for solder, whose lead content [as discussed in *Chapter 18: Printed Circuit Boards (PCBs)*] has raised environmental concerns.

SUPERCONDUCTORS

One of the “Holy Grails” of the electronics industry is to have access to conductors with zero resistance to the flow of electrons, and for such conductors, known as *superconductors*, to operate at room temperatures. As a concept, superconductivity is relatively easy to understand: consider two sloping ramps into which a number of pegs are driven. In the case of the first ramp, the pegs are arranged randomly across the surface, while in the second the pegs are arranged in orderly lines. Now consider what happens when balls are released at the top of each surface (Figure 21.18).

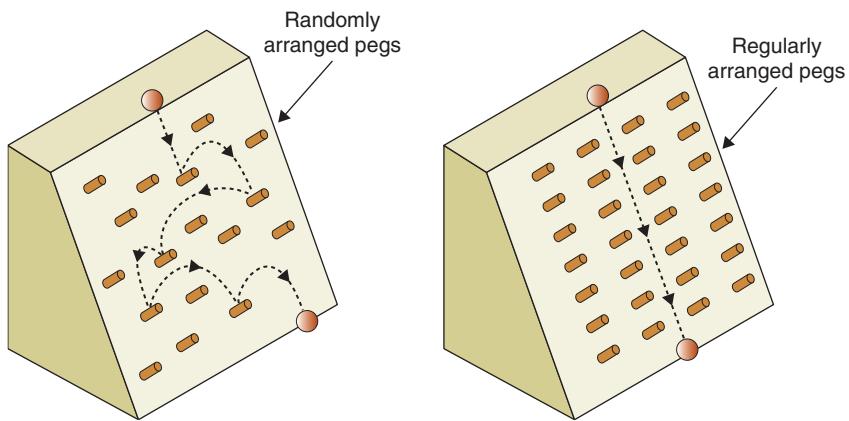


FIGURE 21.18

Graphical representation of superconductivity.

In the case of the randomly arranged pegs, the ball’s progress is repeatedly interrupted, while in the case of the pegs arranged in orderly lines, the ball slips through “like water off a duck’s back.” Although analogies are always suspect (and this one doubly so), the ramps may be considered to represent

conducting materials, the gravity accelerating the balls takes on the role of voltage differentials applied across the ends of the conductors, the balls play the part of electrons, and the pegs portray atoms.

The atoms in materials vibrate due to the thermal energy contained in the material: the higher the temperature, the more the atoms vibrate. An ordinary conductor's electrical resistance is caused by these atomic vibrations, which obstruct the movement of the electrons forming the current. Using the *Kelvin*,²¹ or *absolute*, scale of temperature, 0 K (corresponding to -273°C) is the coldest possible temperature and is known as absolute zero. If an ordinary conductor were cooled to a temperature of absolute zero, atomic vibrations would cease, electrons could flow without obstruction, and electrical resistance would fall to zero. A temperature of absolute zero cannot be achieved in practice, but some materials exhibit superconducting characteristics at higher temperatures.²²

In 1911, the Dutch physicist Heike Kamerlingh Onnes (1853–1926) discovered superconductivity in mercury at a temperature of approximately 4 K (-269°C). Many other superconducting metals and alloys were subsequently discovered but, until 1986, the highest temperature at which superconducting properties were achieved was around 23 K (-250°C) with the niobium-germanium alloy (Nb_3Ge).

In 1986, Georg Bednorz and Alex Müller discovered a metal oxide that exhibited superconductivity at the relatively high temperature of 30 K (-243°C). This led to the discovery of ceramic oxides that superconduct at even higher temperatures. In 1988, an oxide of thallium, calcium, barium, and copper ($\text{Tl}^2\text{Ca}^2\text{Ba}_2\text{Cu}_3\text{O}_{10}$) displayed superconductivity at 125 K (-148°C), and, in 1993, a family based on copper oxide and mercury attained superconductivity at 160 K (-113°C). These “high-temperature” superconductors are all the more noteworthy because ceramics are usually extremely good insulators.

Like ceramics, most organic compounds are strong insulators; however, some organic materials known as *organic synthetic metals* do display both conductivity and superconductivity. In the early 1990s, one such compound was shown to superconduct at approximately 33 K (-240°C). Although this is well below

²¹ Invented by the British mathematician and physicist William Thomson (1824–1907), first Baron of Kelvin.

²² If the author were an expert in superconductivity, this is the point where he might be tempted to start muttering about “*Correlated electron movements in conducting planes separated by insulating layers of mesoscopic thickness, under which conditions the wave properties of electrons assert themselves and electrons behave like waves rather than particles.*” But he’s not, so he won’t.

the temperatures achieved for ceramic oxides, organic superconductors are considered to have great potential for the future.

New superconducting materials are being discovered on a regular basis,²³ and the search is on for room temperature superconductors, which, if discovered, are expected to revolutionize electronics as we know it.

NANOTECHNOLOGY

Nanotechnology is an elusive term that is used by different research and development teams to refer to whatever it is that they're working on at the time. However, regardless of their particular area of interest, nanotechnology always refers to something extremely small; for example, motors and pumps the size of a pinhead, which are created using similar processes to those used to fabricate integrated circuits. In fact, around the beginning of 1994, one such team unveiled a miniature model car which was smaller than a grain of short-grain rice. This model contained a micro-miniature electric motor, battery, and gear train, and was capable of traversing a fair-sized room (though presumably not on a shag-pile carpet).

In 1959, the legendary American physicist Richard Feynman gave a visionary talk, in which he described the possibility by which sub-microscopic computers could perhaps be constructed. Feynman's ideas have subsequently been extended to become one of the more extreme branches of nanotechnology featuring micro-miniature products that assemble themselves! The theory is based on the way in which biological systems operate. Specifically, the way in which enzymes²⁴ act as *biological catalysts*²⁵ to assemble large, complex molecules from smaller molecular building blocks.

Back to the Water Molecule

Before commencing this discussion, it is necessary to return to the humble water molecule.²⁶ As you may recall, water molecules are formed from two hydrogen atoms and one oxygen atom, all of which share electrons between themselves.

²³ A new family of iron-based superconductors was announced in May 2008 as I was penning these words.

²⁴ Enzymes are complex proteins that are produced by living cells and catalyze biochemical reactions at body temperatures.

²⁵ A catalyst is a substance that initiates a chemical reaction under different conditions (such as lower temperatures) than would otherwise be possible. The catalyst itself remains unchanged at the end of the reaction.

²⁶ Water molecules were introduced in *Chapter 2: Atoms, Molecules, and Crystals*.

However, the electrons are not distributed equally, because the oxygen atom is a bigger, more robust fellow which grabs more than its fair share (Figure 21.19).

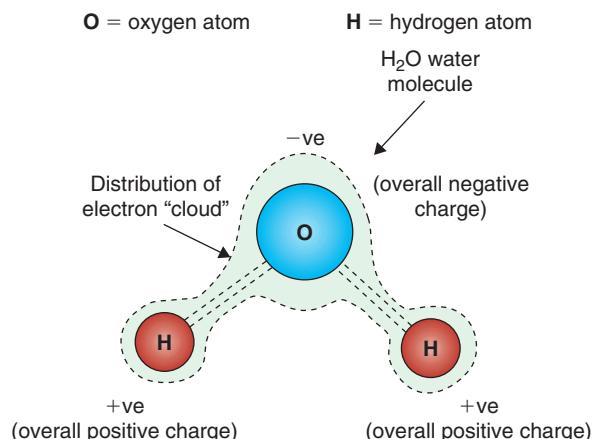


FIGURE 21.19

Distribution of electrons in a water molecule.

The angle formed between the two hydrogen atoms is 105°. This is because, of the six electrons that the oxygen atom owns, two are shared with the hydrogen atoms and four remain the exclusive property of the oxygen. These four huddle together on one side of the oxygen atom and put “pressure” on the bond angle. The bond angle settles on 105° because this is the point where the pressure from the four electrons is balanced by the natural repulsion of the two positively charged hydrogen atoms (similar charges repel each other).

The end result is that the oxygen atom has an overall negative charge, while the two hydrogen atoms are left feeling somewhat on the positive side. This unequal distribution of charge means that the hydrogen atoms are attracted to anything with a negative bias—for example, the oxygen atom of another water molecule. Although the strength of the resulting bond, known as a *hydrogen bond*, is weaker than the bond between the hydrogen atom and its “parent” oxygen atom, it is still quite respectable.

When water is cooled until it freezes, its resulting crystalline structure is based on these hydrogen bonds. Even in its liquid state, the promiscuous, randomly wandering water molecules are constantly forming hydrogen bonds with each other. These bonds persist for a short time until another water molecule clumsily barges into them and knocks them apart. From this perspective, a glass of water actually contains billions of tiny ice crystals that are constantly forming and being broken apart again.

Imagine a Soup

However, we digress. Larger molecules can form similar electrostatic bonds with each other. Imagine a “soup” consisting of large quantities of many different types of molecules, two of which, M_a and M_b , may be combined to form larger molecules of type M_{ab} (Figure 21.20).

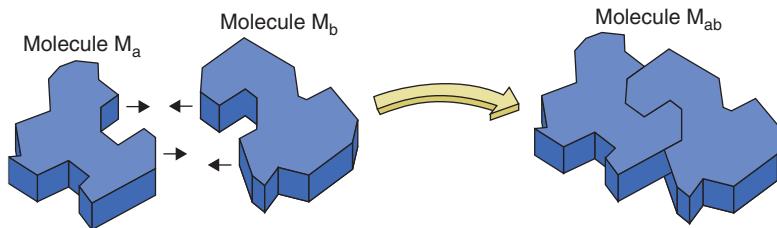


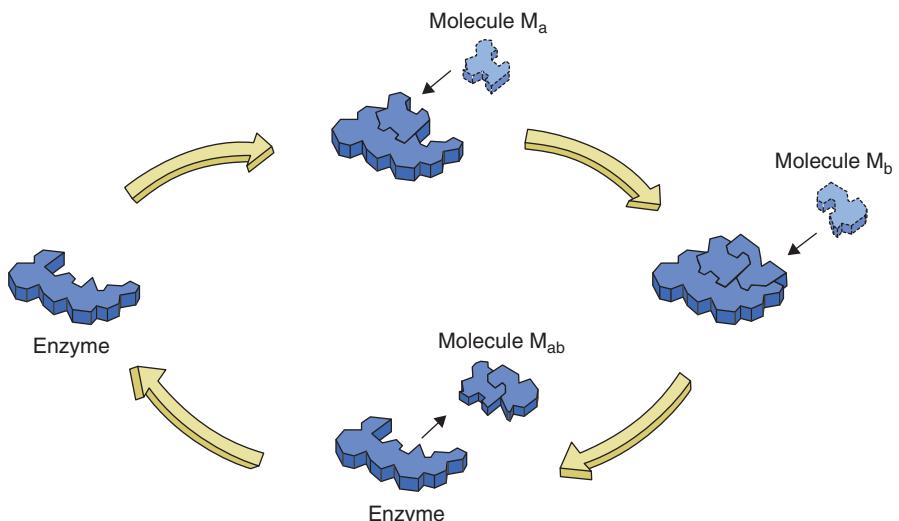
FIGURE 21.20

Combining molecules.

This is similar in concept to two pieces of a jigsaw puzzle, which will only fit together if they are in the correct orientation to each other. Similarly, M_a and M_b will only bond to form M_{ab} if they are formally presented to each other in precisely the right orientation. However, the surfaces of the molecules are extremely complex three-dimensional shapes, and achieving the correct orientation is a tricky affair. Once the molecules have been brought together their resulting bonds are surprisingly strong, but the chances of the two molecules randomly achieving exactly the correct orientation to form the bonds are extremely small.

It is at this point of the story that enzymes reenter the plot. There are numerous enzymes, each dedicated to the task of “matchmaking” for two of their favorite molecules. The surface of an enzyme is also an extremely complex three-dimensional shape, but it is much larger than its target molecules and has a better chance of gathering them up. The enzyme floats around until it bumps into a molecule of type M_a to which it bonds. The enzyme then continues on its trek until it locates a molecule of type M_b . When the enzyme bonds to molecule M_b , it orientates it in exactly the right way to complete the puzzle with molecule M_a . (Figure 21.21).

The bonds between M_a and M_b are far stronger than their bonds to the enzyme. In fact, as soon as these bonds are formed, the enzyme is actually repelled by the two little lovebirds and promptly thrusts M_{ab} away. However, the enzyme immediately forgets its pique, and commences to search for two more molecules (some enzymes can catalyze their reactions at the rate of half a million molecules per minute).

**FIGURE 21.21**

Using an enzyme to form molecule M_{ab}.

The saga continues, because another, larger enzyme may see its task in life as bringing M_{ab} together with yet another molecule M_{cd}. And so it continues, onwards and upwards, until the final result, whatever that may be, is achieved.

As our ability to create “designer molecules” increases, it becomes increasingly probable that we will one day be able to create “designer enzymes.” This would enable us to mass-produce “designer proteins” that could act as alternatives to semiconductors (see also the *Protein Switches and Memories* topic earlier in this chapter). As one of the first steps along this path, a process could be developed to manufacture various proteins that could then be bonded to a substrate or formed into three-dimensional blocks for optical memory applications. At a more sophisticated level, it may be possible for such a process to directly create the requisite combinations of proteins as self-replicating structures across the face of a substrate.

However, the possibilities extend far beyond the mass-production of proteins. It is conceivable that similar techniques could be used to assemble nonorganic structures such as microscopic electromechanical artifacts. All that would be required (he said casually) would be for the individual components to be shaped in such a way that naturally occurring electrostatic fields would cause them to form bonds when they were brought together with their soul mates. In fact, this is one step along the path toward molecular-sized robots known as *nanobots*. Taken to extremes, the discipline of electronics in the future may not involve the extreme temperatures, pressures, and noxious chemicals that are in vogue today. Instead, electronics may simply involve “cookbook” style recipes;

for example, the notes accompanying an electronics course in 2050 AD may well read as follows:

INTERMEDIATE ELECTRONICS (AGES 12 TO 14) SUPERCOMPUTERS 101

*Instructions for creating a micro-miniature massively parallel supercomputer**

- Obtain a large barrel.
- In your barrel, mix two parts water and one part each of chemicals A, B, C, ...
- Add a pinch of nanobot-mix (which you previously created in Nanobots 101).
- Stir briskly for one hour with a large wooden spoon.

Congratulations, you will find your new supercomputers in the sediment at the bottom of the barrel. Please keep one teaspoon of these supercomputers for your next lesson.

*These instructions were reproduced from *Bebop to the Boolean Boogie*, 50th edition, 2050, the most popular electronics book in the history of the universe!

Of course, some of this is a little far-fetched (with the hopeful exception of the references to *Bebop to the Boolean Boogie*). However, for what it's worth, the author would bet his wife's life savings that this type of technology will occur one day, and also that it will be here sooner than you think!

ONCE AGAIN, THE MIND BOGGLERS

In reality, we've only touched on a very few of the myriad ideas that are out there running wild and free. For example, one area that is currently seen as a growth industry is that of *MicroElectroMechanical Systems* (MEMS) and their optical counterparts OMEMS. These refer to devices that contain both electrical, mechanical, and—in the case of OMEMS—optical elements, and are physically very small (sometimes measuring only a few millionths of a meter in size). MEMS can be used to create microscopic sensors to monitor the surrounding environment and actuators that can modify the environment with great precision, while OMEMS find use in communications systems.

As an example, in early 2001, biophysicists at the Hungarian Academy of Sciences devised a way to create microscopic machines that are constructed and operated by light. In one case they created gears that are each less than 1/5000 of an inch ($5\mu\text{m}$) in diameter. The rotors spin when illuminated by a low-power laser as photons hit their flanges. Such devices could pump materials across miniature chemical arrays.

Small as they are, MEMS and OMEMS are still huge on the molecular scale. In early 2001, the University of Illinois Beckman Institute for Advanced Science and Technology reported research with organic molecules like tiny mechanical switches. These molecular switches are only attached to the substrate by a single atom and they can spin as fast as 100 trillion times a second, which provides the potential for switching arrays running at 100 terahertz!

And just to illustrate the range of things that are being investigated, biomedical engineers at the Georgia Institute of Technology and Emory University are working on linking electronics to living neurons to create incredibly sophisticated neural networks.

SUMMARY

The potpourri of technologies introduced here has been offered for your delectation and delight. Some of these concepts may appear to be a little on the wild side, and you certainly should not believe everything that you read or hear. On the other hand, you should also be careful not to close your mind, even to seemingly wild and wacky ideas, in case something sneaks up behind you and bites you on the #####.²⁷ As the *Prize* said in *Where is Earth?* by Robert Sheckley: “*Be admiring but avoid the fulsome, take exception to what you don’t like, but don’t be stubbornly critical; in short, exercise moderation except where a more extreme attitude is clearly called for.*”

²⁷ Arsek no questions!

SECTION 3

Design Tools and Stuff

This page intentionally left blank

CHAPTER 22

General Concepts

STUFF, MORE STUFF, AND YET MORE STUFF

Trying to explain all there is to know about the tools and techniques used to design silicon chips, circuit boards, and electronic systems is a daunting task. Don't panic! It's not that this is particularly complicated (if we're looking at things from a 30,000-foot viewpoint), it's just that there are so many different things to wrap one's brain around.

Another tricky aspect of all of this is that, in many cases, it would be nice to know something about topic "xxx" before we introduce topic "yyy." Unfortunately, oftentimes it would also be preferable to have an understanding of topic "yyy" before we consider topic "xxx." As you can imagine, this poses something of a Zen-like dilemma.

So ... the way we're going to tackle this is that I'm going to waffle on about all sorts of things in whatever order seems to make sense to me (or whatever order they pop into my head). In some cases we will refer to things we haven't talked about yet, but hopefully in a way that will still make sense. If all fails, I suggest reading this chapter twice, because the second time around you'll already know what's coming (if you see what I mean).

345

THE ORIGINS OF EDA

The term *Electronic Design Automation* (EDA) refers to the tools that are used to design and verify *integrated circuits* (ICs), *printed circuit boards* (PCBs), and electronic systems in general.

Prior to the 1970s, electronic circuits were handcrafted. Circuit diagrams (known as *schematics*) showing symbols for the components to be used and the connections between them were drawn using pen, paper, and stencils. Similarly, the copper tracks on a circuit board were drawn using red and blue

pencils or pens to represent the top and bottom of the board. Furthermore, any form of analysis (for example, “*What frequency will this oscillator run at if I use this capacitor and this resistor?*”) was performed with pencil, paper, and a slide rule (or a mechanical calculator, if you were lucky). Not surprisingly, this style of design was time-consuming, expensive, and prone to error.

Computer-Aided Design (CAD)

As electronic designs and devices grew more complex, it became necessary to develop automated techniques to aid in the design process. In the early 1970s, companies like Calma, ComputerVision, and Applicon created special computer programs that helped personnel in the drafting department¹ capture hand-drawn designs in digital form using large-scale digitizing tables.

Over time, these early computer-aided drafting tools evolved into interactive programs that performed integrated circuit layout (that is, they could be used to describe the locations of the transistors forming the integrated circuit and the connections between them). Other companies like Racal-Redac, SCI-Cards, and Telesis created equivalent layout programs for printed circuit boards. These integrated circuit and circuit board layout programs became known as *Computer-Aided Design (CAD)*² tools.

Computer-Aided Engineering (CAE)

Also in the late 1960s and early 1970s, a number of universities and commercial companies started to develop computer programs known as *simulators*. These programs allowed students and engineers to emulate the operation of an electronic circuit without actually having to build it first. Perhaps the most famous of the early simulators was the *Simulation Program with Integrated Circuit Emphasis (SPICE)*.^{3,4} This was developed by the University of California in Berkeley and was made available for widespread use around the beginning of the 1970s. SPICE was designed to simulate the behavior of analog circuits—other programs called logic simulators were developed to simulate the behavior of digital circuits.⁵

¹The drafting department is referred to as the “drawing office” in the UK.

²In conversation, CAD is pronounced as a single word to rhyme with “bad.” The term CAD is also used to refer to computer-aided design tools intended for a variety of other engineering disciplines, such as mechanical and architectural design.

³In conversation, SPICE is pronounced like the seasoning to rhyme with “mice.”

⁴SPICE has proved to be an enduring tool and enhanced versions of the original program remain the mainstay of the analog design domain.

⁵The differences between the analog and digital domains were introduced in *Chapter 1: Analog Versus Digital*.

At the beginning of the 1980s, companies like Daisy, Mentor, and Valid spawned computer programs that allowed engineers to capture gate-level or transistor-level schematic (circuit) diagrams on the computer screen. These tools could then be used to generate textual representations of the circuits called netlists that described the components to be used and the connections between them. In turn, these netlists could be used to drive analog and digital simulators (and eventually layout tools).

The companies promoting front-end tools for schematic capture and simulation classed them as *Computer-Aided Engineering* (CAE).⁶ This was based on the fact that these tools were targeted toward design engineers, and the CAE companies wished to distinguish their products from the CAD tools that were originally used by the drafting department.

Designers Versus Engineers

If you say things the wrong way when talking to someone in the industry, you immediately brand yourself as an outsider (one of “them” instead of one of “us”). For historical reasons that are based on the origins of the terms CAE and CAD, the term *design engineer* or simply *engineer* is typically used to refer to someone who conceives and describes the functionality of an integrated circuit, printed circuit board, or electronic system (what it does and how it does it).⁷ By comparison, the term *layout designer* or simply *designer* is typically used to refer to someone who lays out an integrated circuit or a circuit board (determines the locations of the components and the routes of the tracks connecting them together).

Electronic Design Automation (EDA)

Sometime during the 1980s, all of the CAE and CAD tools used to help design electronic components, circuit boards, and systems came to be referred to by the “umbrella” name of *Electronic Design Automation* (EDA), and everyone was happy (apart from the ones who weren’t, but they don’t count).

AUTOMATION

When you purchase an EDA tool and you run it, you will be presented with a variety of menus, commands, tool bar icons, and such-like. These default

⁶In conversation, CAE is spelled out as “C-A-E.”

⁷Analog design engineers may also be referred to as *circuit designers* (the guys and gals who perform analog layout are still referred to as *layout designers*). Meanwhile, the folks who actually know how to design *Radio Frequency* (RF) and microwave circuits are so clever that they can call themselves whatever they please.

capabilities will be sufficient for many users, but some “power users” or “enterprise-level-users” may wish to add additional capabilities. In order to accommodate this, a high-end EDA tool will come equipped with an *Application Programming Interface* (API) that provides the ability to access, control, and manipulate data inside the application.

In some cases, users (or, more typically, some engineering department in the company for which the users work), will use such an API to add new menus and commands to an individual tool. In other cases, the API’s associated with multiple tools can be used to allow one tool to invoke another tool, run that tool, read/write data from/to that tool, and so forth. Yet another scenario is to launch one or more tools from the command line as “background jobs” [which means that their *Graphical User Interfaces* (GUIs) won’t appear on the user’s computer screen] and have them perform a whole sequence of tasks communicating data back and forth as required.

The term “automation” refers to the ability for end-users to augment, customize, and drive the capabilities of electronic design and verification tools by means of a scripting language⁸ and associated support utilities. (The reasoning behind the “automation terminology” comes from the perspective of the user having the ability to automate a series of actions that are available in the application.)

EMBEDDED SYSTEMS

In a little while we’re going to be talking about different computer languages, including programming languages like C and C++. When most folks think about a computer system, they think of the computer on their desk or perhaps their notepad computer; and when most folks hear the term “program” they think of things like word processors or spreadsheet applications.

That’s not what we’re talking about here. For the purpose of this portion of our discussions, when we talk about *hardware*, *software*, and *firmware*, we’re doing so in the context of *embedded systems*.

But what is an embedded system? Ah, I was hoping you weren’t going to ask me that, because this is not an exactly defined term. Some folks say that an embedded system is a special-purpose computer system designed to perform one or a few dedicated tasks. This definition encompasses things like washing machine controllers, air conditioning controllers, and so forth. Some folks

⁸See also discussions on *Different Languages* later in this chapter.

would also consider cell phones and similar devices to be embedded systems. One industry expert told me that that the simplest definition was: *“a computer without a keyboard.”* Someone else once told me that an embedded system is: *“one that you don’t even know is there until it stops working.”*

So, I hope that’s cleared things up!

PROGRAMMING VERSUS HARDWARE DESIGN LANGUAGES

There are a wide variety of programming languages available, but—excepting specialist application areas—the most commonly used by far are traditional C and its object-oriented offspring, C++. For our purposes here, we will refer to these collectively as C/C++.

By default, statements in languages like C/C++ are executed *sequentially*. For example, assuming that we have already declared three integer variables called a, b, and c, then the following statements ...

```
a = 6; /* Statement in C/C++ program */  
b = 2; /* Statement in C/C++ program */  
c = 9; /* Statement in C/C++ program */
```

...would—perhaps not surprisingly—occur one after the other. However, this has certain implications; for example, if we now assume that the following statements occur sometime later in the program ...

```
a = b; /* Statement in C/C++ program */  
b = a; /* Statement in C/C++ program */
```

... then a (which initially contained “6”) will be loaded with the value currently stored in b (which is “2”); next, b (which initially contained “2”) will be loaded with the value currently stored in a (which is now “2”); so both a and b will end up containing the same value.

The sequential nature of programming languages is the way in which software engineers think. However, hardware design engineers have quite a different view of the world. In the case of hardware, lots of things might be happening at the same time. Hardware engineers would describe this as *“things happening in parallel”* or *“things happening concurrently.”* As we’ve already noted, sequential languages like C/C++ describe things happening sequentially, not in parallel. Hardware design engineers need to be able to express parallel activities. If you have a logical function like an AND or an OR with multiple inputs, for

example, several of those inputs could change simultaneously, so you need some way to be able to describe this type of situation.

In order to address this, hardware design engineers make use of special computer languages called *Hardware Description Languages* (HDLs). Two well-known HDLs are called Verilog and VHDL (we'll talk about these in more detail later). As a simple example, let's assume that a piece of hardware contains two multi-bit registers called *a* and *b* that are driven by a common clock. Let's further assume that these registers have previously been loaded with values of 6 and 2, respectively. Finally, let's assume that at some point in the HDL code we see the following statements:

```
a = b; /* Statement in an HDL */
b = a; /* Statement in an HDL */
```

Note that the above syntax doesn't actually represent VHDL or Verilog, it's just a generic syntax of interest only for the purposes of this example. Generally speaking, hardware engineers would expect both of these statements to be executed concurrently (at the same time). This means that register *a* (which initially contained "6") will be loaded with the value stored in register *b* (which was "2") while—at the same time—register *b* (which initially contained "2") will be loaded with the value stored in *a* (which was "6"). The end result is that the initial contents of *a* and *b* will be exchanged.

As usual, of course, the above is something of a simplification. However, it's fair to say that HDL statements will execute concurrently by default, unless sequential behavior is forced by certain techniques (like blocking assignments). Thus, by default, digital logic simulators will execute the statements shown above in this concurrent manner; similarly, logic synthesis tools will generate hardware that handles these two activities simultaneously. By comparison, unless explicitly directed to do otherwise (using techniques that are beyond the scope of these introductory discussions) C/C++ statements will execute sequentially.

NETLISTS

One concept that is sometimes confusing to the uninitiated is that of a "netlist." The best way to explain this is as follows.

Transistor-Level

A *transistor-level netlist* refers to a circuit representation at the level of individual transistors, resistors, capacitors, and inductors. Each of these items may be equipped with a wide variety of parameters defining its physical and electrical

characteristics. The netlist will also specify the connections (wires) between the various components. This type of netlist might be generated as output from a schematic capture system, for example, and it might be used as input by an analog simulator (we'll talk about all of these tools in *Chapters 23: Design and Verification Tools*).

Gate-Level

A *gate-level netlist* refers to a circuit representation at the level of individual logic gates, registers, and other simple functions. The netlist will also specify the connections (wires) between the various gates and functions.

Component-Level

A *component-level netlist* refers to a circuit representation at the level of individual components. For example, in the case of a *Printed Circuit Board* (PCB), a component-level netlist will detail all of the integrated circuits along with discrete components like resistors, capacitors, and inductors. The netlist will also specify the connections (wires) between the various components.

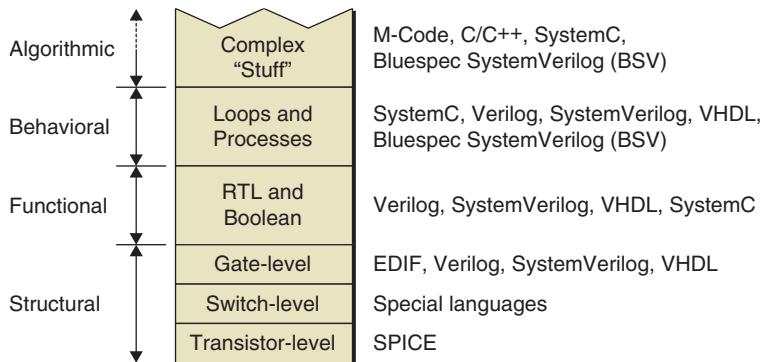
In the early days of electronic design, such netlists would be captured by hand (or automatically generated by a tool) in the form of an ASCII⁹ text file. More recently, the netlist (or the elements required to describe such a netlist) might be stored in a relational database.

DIFFERENT LEVELS OF ABSTRACTION

The functionality of an electronic circuit can be represented at different levels of abstraction; also, different *Hardware Description Languages* (HDLs) support these levels of abstraction to a greater or lesser extent as illustrated in Figure 22.1 (the languages referred to in this figure are discussed in more detail in the following topic).

Note that these levels really aren't "officially" defined, *per se*; different folks will think about these things differently. Also, the list of languages isn't exhaustive (I've omitted some of the analog HDLs because they didn't really fit into

⁹Towards the end of the 1950s, the *American Standards Association* (ASA) began to consider the problem of defining a standard character code mapping that could be used to facilitate the representation, storing, and interchanging of textual data between different computers and peripheral devices. In 1963, the ASA—which changed its name to the *American National Standards Institute* (ANSI) in 1969—announced the first version of the *American Standard Code for Information Interchange* (ASCII).

**FIGURE 22.1**

Different levels of abstraction and associated HDLs.

this picture). And, while we're in the "weasel words," we should probably note that the various languages aren't always as tightly associated with the various levels of abstraction as this figure might indicate. For example, although you can use SystemVerilog and VHDL to represent gate-level netlists, the result is a bit cumbersome, and EDIF is better. Similarly, although you can use SystemC to create RTL representations, you really don't want to if you're given a choice.

Transistor-Level

The lowest level of abstraction (as far as we're concerned for the purposes of these discussions) is a transistor-level netlist, which comprises transistors, resistors, capacitors, inductors, and the connections between them. The most popular language for this is SPICE, which refers to both the netlist language and the corresponding analog simulation capability.

It is common to refer to a SPICE netlist as a "SPICE Deck," which is a hang-over from the days when this data was stored on punched cards (i.e., "a deck of cards"). Also, some versions of the SPICE simulator had a user interface called *Nutmeg*, for no other reason than the fact that nutmeg is a spice. (Oh, how we laughed.)

Switch-Level

The lowest level of abstraction with regard to the digital domain is a switch-level netlist, in which a circuit is represented as a network of *Field-Effect Transistors* (FETs). A corresponding switch-level simulator would model each transistor as being "open," "closed," and "unknown" and each node (wire) as being 0, 1, and X (unknown).

Gate-Level

A slightly higher level of digital abstraction would be the gate level, which refers to describing the circuit as a netlist of primitive logic gates, registers, and other simple functions. Although languages like Verilog and VHDL can be used to represent gate-level netlists, it is more common to do so in a neutral language called EDIF (*Electronic Design Interchange Format*).

Structural

Transistor-, switch-, and gate-level representations may be classed as being *structural* because they capture the structure of the circuit. It should be noted, however, that the term “structural” can have different connotations, because it may also be used to refer to a hierarchical block-level netlist in which each block may have its contents specified using any of the levels of abstraction shown in Figure 22.1.

Functional (Boolean, RTL)

The next level of HDL sophistication is the ability to support functional representations, which covers a range of constructs. In the digital domain, we might start with the ability to describe a function using Boolean equations. For example, assuming that we had already declared a set of signals called *y*, *select*, *dataA*, and *dataB*, we could capture the functionality of a simple 2:1 multiplexer using the following Boolean equation:

```
y = (select & dataA) | (!SELECT & dataB);
```

Note that this is a generic syntax that does not favor any particular HDL and is used only for the purposes of this example. The functional level of abstraction also encompasses representations at the *Register Transfer Level* (RTL). The term “RTL” covers a multitude of manifestations, but the easiest way to wrap one’s brain around the underlying concept is to consider a design formed from a collection of registers linked by combinational logic. These registers are often controlled by a common clock signal, so assuming that we had already declared two signals called *clock* and *control* along with a set of registers called *regA*, *regB*, *regC*, and *regD*, then an RTL-type statement might look something like the following:

```
When clock Rises
If control == 1
    Then regA = regB & regC;
        else regA = regB | regD;
    End If;
End When;
```

In this case, symbols like When, Rises, If, Then, Else, etc. are keywords whose semantics are defined by the owners of the HDL. Once again, this is a generic syntax that does not favor any particular HDL and is used only for the purposes of this example.

Behavioral

The highest level of abstraction sported by traditional HDLs is known as behavioral, which refers to the ability to describe the behavior of a circuit using abstract constructs like loops and processes. This also encompasses using algorithmic elements like adders and multipliers in equations.

Algorithmic

This refers to the ability to represent extremely complex data structures and algorithmic operations. In the case of M-Code,¹⁰ for example, it's possible to represent a complex transformation such as a matrix inversion in a single line of code.

DIFFERENT LANGUAGES

As I ponder how to present this, I realize that it might be a little scary to see all of the following languages for the first time. The point is that very few engineers would be familiar with all of these little rascals; software developers will work with programming languages; hardware design engineers will use one or more of the *Hardware Description Languages* (HDLs); verification engineers will be conversant with one of the formal verification languages; and so forth. The following provides a very brief introduction to some of the more prevalent languages:

Programming Languages

System programming languages such as C, C++, and Java™ are designed to allow programmers to build data structures, algorithms, and—ultimately—applications from the ground up. These languages are said to be “strongly typed,” which means that all constants and variables used in a program must be associated with a specific data type (such as Boolean, integer, real, character, string, etc.) that are predefined as part of the language. Furthermore, certain operations may be allowable only with certain types of data.

In the case of embedded systems, it's not uncommon for software developers to use assembly language for performance-critical functions; for example, the

¹⁰M-Code is the language used by MATLAB® from The Mathworks.

bulk of a *Digital Signal Programming* (DSP) algorithm intended to be run on a special-purpose *Digital Signal Processor* (DSP) may be created in C/C++, but one or more core functions may be implemented in assembly language.

Scripting Languages

Unlike programming languages, scripting languages are not intended to be used to build applications (such as EDA tools) from scratch; instead (in the case of automation applications) they assume that a collection of useful “components” already exist, where these components are typically created using a system programming language. Scripting languages are used to connect—or “glue”—these components together, so they are sometimes referred to as *glue languages* or *system integration languages*.

Scripting languages are typically type-less; for example, a variable can hold an integer one moment, then a real, and then a string, because this makes it easier for them to connect components together. It also speeds the process of script development. Furthermore, code and data are often interchangeable, which means one script can write another script on-the-fly and then execute it.

Scripting languages are often used to perform a series of actions over and over again; hence, their name, which comes from the concept of a written script for a stage play, where the same words and actions are performed identically for each performance.

Possibly the first scripting language was *Perl*, which was presented to the world in 1987. *Perl*, which stands for *Practical Extraction and Report Language*, is an interpreted language that is optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information (it’s also useful for a wide variety of system management tasks).

Today, there are a number of widely used scripting languages, including *JScript* (short for Java Script), *Perl*, *Python*, *Tcl* (pronounced “tickle”), and *VBScript* or *VBS* (short for *Visual Basic Script*).

Hardware Description Languages (Digital)

Life would be so simple if there were only a single HDL to worry about, but no one said that living was going to be easy. In the early days of electronics design (circa the 1970s), anyone who created an HDL-based design tool typically felt moved to create their own language to accompany it. Not surprisingly, the result was a morass of confusion (you had to be there to fully appreciate the dreadfulness of the situation). What was needed was an industry-standard

HDL that could be used by multiple EDA tools and vendors, but where was such a gem to be found?

VHDL: In 1980, the U.S. *Department of Defense* (DoD) launched the *Very High Speed Integrated Circuit* (VHSIC) program, whose primary objective was to advance the state-of-the-art in digital integrated circuit technology. As part of this, a project to develop a new hardware description/documentation language called VHSIC HDL (or VHDL for short) was launched in 1981. As a language, VHDL is very strong at the functional (Boolean Equation and RTL) and behavioral levels of abstraction and it also supports some system-level/algorithmic design constructs. However, VHDL is a little weak when it comes to the structural (switch and gate) level of abstraction, especially with regard to its delay modeling capability.

Verilog: Sometime around the mid-1980s, Phil Moorby (one of the original members of the team who created the famous HILO logic simulator) designed a new HDL called Verilog, which was reasonably strong at the gate-level of abstraction (especially with regard to delay modeling capability); very strong at the functional (Boolean Equation and RTL) level of abstraction; and also supported some behavioral constructs. In 1990, Verilog was placed in the public domain, which prompted a lot of EDA and design companies to start using Verilog as their language of choice. Having a single design representation that could be used by simulation, synthesis, and other tools made everyone's lives a lot easier. It is important to remember, however, that Verilog was originally conceived with simulation in mind; applications like synthesis were something of an afterthought. This means that, when creating a Verilog representation to be used for both simulation and synthesis, one is restricted to using a synthesizable subset of the language (which is loosely defined as whatever collection of language constructs your particular logic synthesis package understands and supports).

UDL/I: As we just noted, Verilog was originally designed with simulation in mind. Similarly, VHDL was created as a design documentation and specification language, without simulation or synthesis really being taken into account. The end result is that one can use both of these languages to describe constructs that can be simulated but not synthesized. In order to address these problems, the *Japan Electronic Industry Development Association* (JEIDA) introduced their own HDL called the *Unified Design Language for Integrated Circuits* (UDL/I) in 1990. The key advantage of UDL/I was that it was designed from the ground up with both simulation and synthesis in mind. The UDL/I environment includes a simulator and a synthesis tool, and is available for free

(including the source code). However, by the time UDL/I arrived on the scene, Verilog and VHDL already held the high ground, and this language never really managed to attract much interest outside of Japan.

Superlog and SystemVerilog: In 1997, things started to get complicated, because that's when a company called Co-Design Automation was formed. Working furiously, the folks at Co-Design developed a "Verilog on Steroids" called Superlog. The two main problems with Superlog were (a) it was essentially a proprietary language, and (b) it was so much more sophisticated than Verilog that getting other EDA vendors to enhance their tools to support it would have been a major feat.

In the summer of 2002, a standards organization called Accellera released the specification for a hybrid language called SystemVerilog 3.0 (don't even ask me about 1.0 and 2.0). The great advantage to this language was that it was an incremental enhancement to the existing Verilog rather than the death-defying leap represented by a full-up Superlog implementation (having said this, SystemVerilog 3.0 featured many of Superlog's language constructs, which were donated by Co-Design).

Bluespec SystemVerilog: Created by a company called Bluespec (surprise), Bluespec SystemVerilog (BSV) was introduced in 2004. Just as the C language provides a higher level of abstraction than assembly language for software design, BSV provides a higher level of abstraction than RTL (as offered by VHDL, Verilog, and SystemVerilog) for hardware design. Hardware design is about concurrency, or parallel behavior. One of the biggest challenges of hardware design is properly expressing and coordinating where parallel behaviors intersect. BSV has extended SystemVerilog to simplify both the management of these intersecting parallel behaviors and the description of hardware structures. From a high-level description of hardware, BSV is compiled into Verilog RTL for integration into the rest of a hardware design flow. BSV is general-purpose, and is used for the modeling, verification, and implementation of hardware.

SystemC: And then we have SystemC, which some design engineers love and others hate with a passion. One big argument for SystemC is that it provides a more natural environment for hardware/software co-design and co-verification. One big argument against it is that the majority of hardware design engineers are very familiar with Verilog and/or VHDL, but they are not familiar with the object-orientated aspects of SystemC. Another consideration is that the majority of today's synthesis offerings represent hundreds of engineer years of development in translating Verilog or VHDL into gate-level netlists.

By comparison, there are far fewer SystemC-based synthesis tools, and those that are available tend to be somewhat less sophisticated than their more traditional counterparts. In reality, SystemC is more applicable to a system-level versus an RTL design environment. It also finds a lot of use in the verification domain.

Hardware Description Languages (Analog)

The language you use depends on where you are and what you're trying to do. For example, to hold a conversation in most of San Francisco, you would typically speak English—but if you want to get a good bargain in the Chinatown area you'll be much better off holding forth in Chinese.

Similarly, analog circuits are very different in character to their digital cousins, so engineers use special analog hardware description languages (AHDLS) to describe them. One of the most widely used AHDLS is SPICE, which is named after the simulator. This allows engineers to represent designs at a low level of abstraction as transistor-capacitor-resistor-inductor netlists. Over time, SPICE has been joined by a number of new AHDLS that can represent circuits at higher levels of abstraction. These include ongoing analog extensions to VHDL and Verilog: VHDL-AMS (analog mixed-signal), Verilog-A (analog), and Verilog AMS (analog mixed-signal).

Verification Languages (General)

Many folks consider SystemC to be a verification language, because it can be used to create and control sophisticated test sequences. It's also of use in creating high-level representations called *Transaction-Level Models* (TLMs).

Perhaps the most sophisticated of the *Hardware Verification Languages* (HVLs) is the aspect-oriented *e*, which was developed by Verisity, which was in turn acquired by Cadence. In case you were wondering, *e* doesn't actually stand for anything now, but originally it was intended to reflect the idea of "English-like" in that it has a natural language feel to it. We can think of *e* as a blend of C and Verilog with a hint of Pascal; it can be used to declare valid ranges and sequences of input values (along with their invalid counterparts) and high-level verification strategies. The *e* description is then used by an appropriate verification environment to guide simulations.

Verification Languages (Formal)

This is where things could start to get really confusing if we're not careful (so let's be careful). We'll start with something called Vera®, which began life with

work done at Sun Microsystems in early 1990s. It was eventually acquired by Synopsys in 1998. Vera is essentially an entire verification environment, similar to the e verification language/environment. Vera encapsulates testbench features and assertion-based capabilities, and Synopsys promoted it as a stand-alone product (with integration into the Synopsys logic simulator). Sometime later, due to popular demand, Synopsys opened things up to for third-party use by making OpenVera™ and *OpenVera Assertions* (OVA) available.

Somewhere around this time, SystemVerilog was equipped with its first-pass at an “assert” statement. Meanwhile, due to the increasing interest in formal verification technology,¹¹ one of the Accellera standards committees started to look around for a formal verification language they could adopt as an industry standard. A number of languages were evaluated (including OVA), but in 2002 the committee eventually opted for the Sugar language from IBM. Just to add to the fun and frivolity, Synopsys then donated OVA to the Accellera committee in charge of SystemVerilog (this was a different committee to the one evaluating formal property languages).

Yet another Accellera committee ended up in charge of something called the *Open Verification Library* (OVL), which refers to a library of assertion/property models available in both VHDL and Verilog.

So now we have the assert statements in VHDL and SystemVerilog, OVL (the library of models), OVA (the assertion language), and the *Property Specification Language* (PSL), which is the Accellera version of IBM’s Sugar language. The advantage of PSL is that it has a life of its own in that it can be used independently to the languages used to represent the functionality of the design itself. The disadvantage is that it doesn’t look like anything the hardware description languages design engineers are familiar with, such as VHDL, Verilog, C/C++, etc. There is some talk of spawning various flavors of PSL, such as a VHDL PSL, a Verilog PSL, a SystemC PSL, and so forth; the syntax would differ between these flavors so as to match the target language, but their semantics would be identical.

ELECTRONIC SYSTEM LEVEL (ESL)

ESL is one of those terms that are very hard to pin down. To some folks, ESL means designing at a high level of abstraction, prior to making any hardware/software portioning decisions. To others, ESL means hardware/software codesign.

¹¹Formal Verification is introduced in *Chapter 23: Design and Verification Tools*.

And others would say that ESL refers to anything that's at a higher level of abstraction than *Register Transfer Level* (RTL) representations.

For example, some folks would say that C/C++ to RTL synthesis tools fall firmly in the ESL category. There are also a number of companies who take C/C++ algorithms and programs, analyze them, and then generate hardware accelerators and/or coprocessors.

And, of course, ESL can apply to both design and verification applications. A really good example of the latter is the concept of sequential equivalence checking, which allows you to compare the sequential behavior of different implementations of your design (such as two RTL representations with different numbers of pipeline stages) so as to ensure that their overall functionality is the same.

And there are always new tools popping up claiming to be ESL. In many cases it's hard to say one way or the other, so my "rule of thumb" is that if such a tool (a) speeds the design of complex systems and/or (b) makes my life easier, then I'm prepared to give it the benefit of the doubt.

CHAPTER 23

Design and Verification Tools

WEASEL WORDS

Before we dive headfirst into the mire, let's start with a few "weasel words."¹ The point is that there are a multifarious multitude of tools used for the design and verification of electronic systems. Introducing only the main contenders in any level of depth would require a book in its own right, so we're going to restrict ourselves to saying a cheery "Hello" to just a few of the usual suspects ...

DESIGN CAPTURE

361

In the days before computer-aided design and verification tools, analog design engineers captured their designs as hand-drawn transistor-level schematics. Similarly, digital design engineers captured their gate-level schematics by hand using a pencil and paper. In those days of yore, functional verification (checking that your circuit would do what you wanted it to do) involved a gathering of you and your colleagues, where you "walked-and-talked" them through your schematics explaining what the various "things" did (or were supposed to do), and—hopefully—everyone saying: "Yes, that looks good to us."

Transistor-Level and Gate-Level Netlists

Now, when you're creating pencil-and-paper schematics, it's easy to make trivial mistakes, like calling two different wires by the same name (this is especially true if several guys and gals are working on different parts of the circuit). Thus, sometime around the 1970s, folks started to create simple computer-aided

¹We should always remember the saying: "*Eagles may soar, but weasels rarely get sucked into jet engines!*"

²The concept of netlists was introduced in *Chapter 22: General Concepts*.

tools that could read a text file containing a netlist² and perform simple checks, such as checking that transistors were connected the right way round (in the case of a transistor level netlist) or that you didn't have two logic gates driving the same wire (in the case of a gate-level netlist; Figure 23.1).

Text File

```

BEGIN CIRCUIT=TEST
    INPUT SET_A, SET_B,
          DATA, CLOCK,
          CLEAR_A, CLEAR_B;
    OUTPUT Q, N_Q;
    WIRE SET, N_DATA, CLEAR;

    GATE G1=NAND (IN1=SET_A,
                  IN2=SET_B,
                  OUT1=SET);
    GATE G2=NOT (IN1=DATA,
                  OUT1=N_DATA);
    GATE G3=OR (IN1=CLEAR_A,
                 IN2=CLEAR_B,
                 OUT1=CLEAR);
    GATE G4=OFF (IN1=SET, IN2=N_DATA,
                  IN3=CLOCK, IN4=CLEAR,
                  OUT1=Q, OUT2=N_Q);

END CIRCUIT=TEST;

```

FIGURE 23.1

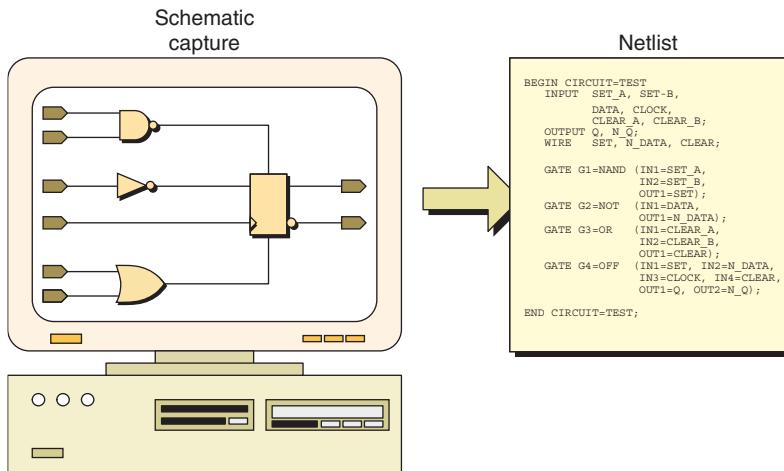
Example of a simple gate-level netlist.

These netlists could also be used by appropriate analog or digital simulators and timing analysis tools as these applications became available (these tools are discussed later in this chapter).

Schematic Capture

So, tools existed that could read text files containing netlists and do useful things with them. At this stage, however, the design engineers were still drawing their schematics with pencil-and-paper. They would then use these schematics as the basis for hand-creating the corresponding netlist using a simple text editor.

As you can imagine, in addition to being time-consuming and boring, hand-transcribing a netlist is prone to error. Not surprisingly, therefore, the next step was to create rudimentary schematic capture packages that allowed users to select items from a library of graphical component symbols, place these symbols on the screen and move them around as required, and connect them together using graphical wires. The schematic capture package could then be used to automatically generate a corresponding netlist (Figure 23.2).

**FIGURE 23.2**

Using schematic capture to generate a netlist.

Higher Levels of Abstraction

On the digital side of the fence, folks started to capture designs at a higher level of abstraction using *Hardware Description Languages* (HDLs) to describe things at the *Register Transfer Level* (RTL).³ The first such HDLs were proprietary; later the industry adopted standard languages such as Verilog and VHDL.

It's interesting to note that the use of higher levels of abstraction like RTL predates logic synthesis technology (as discussed below). Their attraction lay in the fact that engineers could use them to capture the design's functionality quickly and concisely, and also that these representations simulated much faster than their gate-level counterparts.

Graphical Design Entry Lives On

When the first HDL-based flows appeared on the scene, many folks assumed that graphical design entry and visualization tools (such as schematic capture systems) were poised to exit the stage forever. Indeed, for some time, design engineers prided themselves on using text editors like VI (from "Visual Interface") or EMACS as their only design entry mechanism.

But "a picture is worth a thousand words" as they say, and graphical entry techniques remain popular at a variety of levels. For example, it is extremely common to use a block-level schematic editor to capture the design as a collection

³See also the discussions on HDLs and different levels of abstraction like RTL in Chapter 22: *General Concepts*.

of high-level blocks that are connected together. The system might then be used to automatically create a skeleton HDL framework with all of the block names and inputs and outputs declared. Alternatively, the user might create a skeleton framework in HDL, and the system might use this to automatically create a block-level schematic (Figure 23.3).

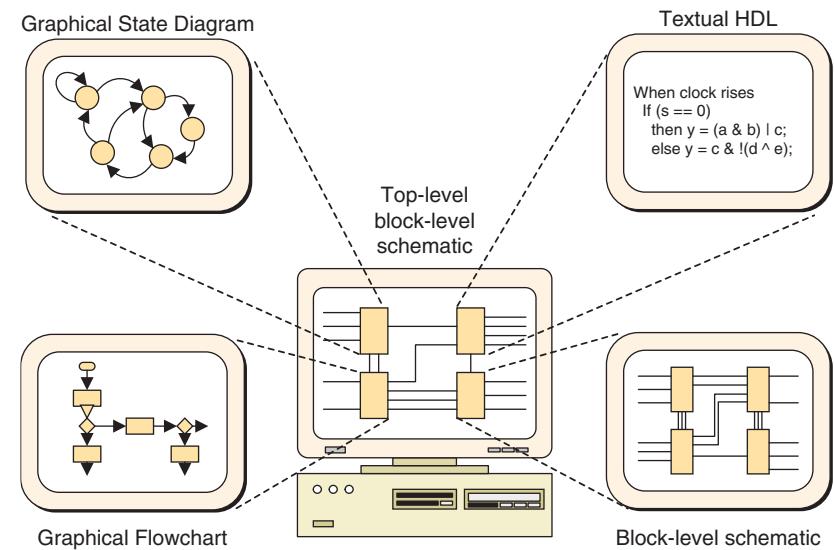


FIGURE 23.3

Mixed-level design capture environment.

From the users' viewpoint, "pushing" down into one of these schematic blocks might automatically open an HDL editor. This could be a pure text-and-command-based editor like VI, or it might be a more sophisticated HDL-specific editor featuring the ability to show language keywords in different colors, automatically complete statements, and so forth.

Furthermore, when "pushing" down into a schematic block, modern design systems often give you a choice between entering and viewing the contents of that block as another, lower-level block-level schematic, raw HDL code, a graphical state diagram (used to represent a finite state machine), a graphical flowchart, and so forth. In the case of the graphical representations like state diagrams and flowcharts, these can subsequently be used to automatically generate their RTL equivalents in the HDL of your choice.

FUNCTIONAL VERIFICATION (SIMULATION)

The idea behind simulation is to create a mathematical model of whatever electronic circuit you are designing. This model, which accurately represents

the function and timing of the various components forming the circuit, is constructed in the computer's memory. You then create a *testbench*, which defines the stimulus to be applied to the circuit's inputs and the responses you expect to see at the circuit's outputs. The simulator then applies this stimulus and models the ensuing effects as they propagate through the circuit.

In the case of an analog circuit, the transistor-level netlist would be read into an analog simulator such as SPICE, and the input stimulus might be presented in the form of constantly varying signals such as sine waves. By comparison, in the case of a digital simulator, the models could be represented at the gate-level and/or as RTL functions, and the stimulus would be defined in digital terms such as binary patterns of 0s and 1s.

FORMAL VERIFICATION

In the not-so-distant past, the term formal verification was considered to be synonymous with *equivalency checking* for the majority of design engineers. In this context, an equivalency checker is a tool that uses formal (rigorous mathematical) techniques to compare two different representations of a design—say an RTL description with a gate-level netlist—to determine whether or not they have the same input to output functionality.

In fact, equivalency checking may be considered to form a subclass of formal verification called *model checking*, which refers to techniques used to explore the state-space of a system to test whether or not certain properties—typically specified in the form of “assertions”—are true.

But just what is formal verification and why is it so cool? Well, in order to provide a starting point for our discussions, let's assume we have a design comprising a number of sub-blocks, and that we are currently working with one of these blocks whose role in life is to perform some specific function. In addition to the *Hardware Description Language* (HDL) representation that defines the functionality of this block, we can also associate one or more assertions/properties with that block (these assertions/properties may be associated with signals at the interface to the block and/or with signals and registers, etc. internal to the block).

A very simple assertion/property might be along the lines of “*Signals A and B should never be active (low) at the same time.*” But these statements can also extend to extremely complex temporal and transaction-level constructs, such as “*When an XYZ write command is received, then a memory write command of type PQR must be issued within 5 to 36 clock cycles.*”

Thus, assertions/properties allow you to describe the behavior of a time-based system in a formal and rigorous manner that provides an unambiguous and universal representation of the design's intent (try saying that quickly). Furthermore, assertions/properties can be used to describe both expected and prohibited behavior.

The fact that assertions/properties are both human and machine-readable makes them ideal for capturing an executable specification, but they go far beyond this. Let's return to considering a very simple assertion/property such as "*Signals A and B should never be active (low) at the same time.*" One term you will hear a lot of is *Assertion-Based Verification* (ABV), which comes in two flavors: *static formal verification* and *dynamic formal verification*.

In the case of static formal verification, an appropriate tool reads in the functional description of the design (typically at the RTL level of abstraction) and then exhaustively analyzes the logic to ensure that this particular condition can never occur. By comparison, in the case of dynamic formal verification, an appropriately augmented logic simulator (or post-simulation analyzer and debugger) will flag a warning if this particular condition ever does occur.

LOGIC SYNTHESIS

Toward the end of the 1980s, *Logic (RTL) Synthesis* tools started to appear for use in digital integrated circuit design. The idea here is that the design engineers capture the desired functionality of the system in RTL, and the synthesis program then automatically converts the RTL representation into a corresponding gate-level netlist. The engineers can direct the synthesis tool to optimize different portions of designs for area (to use the smallest amount of real-estate on the silicon), or for speed.

The original logic synthesis tools didn't know anything about the physical aspects of the integrated circuits for which they were generating netlists. By comparison, today's *physical synthesis* tools actually place the gates close to where they will finally be located and then use these placements to derive more accurate timing estimations.

Furthermore, these modern tools can be provided with (or can automatically generate) a high-level "floorplan," which describes where the main functional blocks will be placed in relation to each other. By means of the floorplan, the synthesis tool can more accurately predict the timing characteristics of the integrated circuit. In turn, this allows the synthesis tool to generate a gate-level netlist that better meets the design goals.

Following synthesis, the engineers (or a separate verification team) may re-simulate the design at the gate level using the same testbench as before. This simulation is performed to ensure that the synthesis tool didn't inadvertently change the functionality of the design. In addition to simulation, the engineers may run a formal verification program called an *equivalency checker* (see previous topic) that compares the RTL and gate-level views of the design to ensure that they are functionally equivalent. (Synthesis tools have been known to make mistakes.)

LAYOUT (PLACE-AND-ROUTE)

In the case of analog designs at the chip or board level, almost all aspects of layout are performed by hand. Having said this, at the time of this writing, we are seeing some interesting developments with regard to automatic analog placement at the chip level.

In the case of digital circuit board designs, the design engineers create a component-level schematic and attach certain constraints to it, such as "*these two wires must be of matched length*" or "*these two wires should be implemented as a differential pair*." This schematic is then passed to the layout designers ...

Perhaps surprisingly, there is little automation in the initial phases of laying out a circuit board. First of all the layout designer has to define the outline of the board (not all boards are rectangular or square; some can have very interesting shapes). The layout tool then reads in the data from the schematic database and presents the collection of components outside the board's outline. The layout designer then places (drags-and-drops) the various components by hand (most layout tools purport to have the ability to perform automatic placement, but the results are always so bad that designers never use these features).

Following placement, the layout tool can be used to perform auto-interactive routing. This part of the process is really very successful; the tool can usually perform 80% of the routing automatically, leaving the layout designer to guide it for the remaining 20%.

The most successful form of automated place-and-route is seen when creating digital integrated circuits. These tools can automatically place tens of millions of logic gates and route tens of millions of wires connecting them.

PARASITIC EXTRACTION

Each wire and via on an integrated circuit or a circuit board has its own physical characteristics (in terms of resistance, capacitance, and inductance) that will affect the way in which signals behave.

Thus, once all of the components (integrated circuits and discrete components on a board; transistors, resistors, capacitors, etc., on an analog chip; logic gates and functions on a digital chip) have been placed and all of the tracks have been routed, engineers run a *parasitic extraction program* to determine characteristics like resistance, capacitance, and inductance associated with each track segment and via. These values will subsequently be used by timing analysis programs as discussed in the next topic.

TIMING ANALYSIS

Timing analysis is run and rerun throughout the design process. Initially, estimated values are used for the various gate and track (wire and via) delays. As more data becomes available, the timing analysis models become more accurate and more sophisticated. Following place-and-route, a parasitic extraction tool is used to determine the resistance, capacitance, and inductance values associated with the tracks and vias. These values are then used to further refine the delay models so as to provide extremely accurate timing analysis capability.

Static Timing Analysis (STA)

In the case of digital designs, the most common form of timing verification in use today is classed as *Static Timing Analysis* (STA). Conceptually this is quite simple, although in practice things are—as usual—more complex than they might first appear.

The timing analyzer essentially sums all of the gate and track delays forming each path through the circuit to give you total input-to-output for each path. (In the case of pipelined designs, the analyzer calculates delays from one bank of registers to the next.)

Prior to place-and-route, the analyzer may make estimations as to track delays. Following place-and-route, the analyzer will employ extracted parasitic values (for resistance, capacitance, and inductance) associated with the physical tracks to provide more accurate results. The analyzer will report any paths that fail to meet their original timing constraints, and it will also warn of potential timing problems (e.g., setup and hold violations) associated with signals being presented to the inputs of registers.

Static timing analysis is particularly well suited to classical synchronous designs and pipelined architectures. The main advantages of static timing analysis are that it is relatively fast, it doesn't require a testbench, and it exhaustively tests every possible path into the ground. On the other hand, static timing analyzers

are little rascals when it comes to detecting “false paths” that will never be exercised during the course of the design’s normal operation. Also, these tools aren’t at their best with designs employing latches, asynchronous circuits, and combinatorial feedback loops.

Statistical Static Timing Analysis (SSTA)

STA is a mainstay of modern digital design flows, but it’s starting to run into problems with the latest process technology nodes. At the time of writing the 45/40-nano nodes are coming online, with the 32-nano node racing towards us.

In the case of modern silicon chips, interconnect delays dominate over logic delays. In turn, interconnect delays are dependent on parasitic capacitance, resistance, and inductance values, which are themselves functions of the topology and cross-sectional shape of the wires.

The problem is that, in the case of the latest technology process nodes, photolithographic processes are no longer capable of producing exact shapes. Thus, as opposed to working with squares and rectangles, we are now working with circles and ellipsoids. Feature sizes like the widths of tracks are now so small, that small variations in the etching process cause deviations that—although slight—are significant with relation to the main feature size. These irregularities are made more significant by the fact that—in the case of high-frequency designs—the so-called “skin-effect” comes into play; this refers to the fact that high-frequency signals travel only through the outer surface, or skin, of the conductor. Furthermore, there are variations in the vertical plane of the track’s cross-section caused by processes like *Chemical Mechanical Polishing* (CMP).

The overall result is that it’s becoming increasingly difficult to accurately calculate track delays. Of course, it is possible to use the traditional engineering fallback of *guard-banding* (using worst-case estimations), but excessively conservative design practices result in device performance that is significantly below the silicon’s full potential, which is an extremely unattractive option in today’s highly competitive marketplace. In fact, the effects of geometry variations are causing the probability distributions of delays to become so wide that worst-case numbers may actually be slower than in an earlier process technology!

One solution is the concept of *Statistical Static Timing Analysis* (SSTA). This is based on generating a probability function for the delay associated with each signal for each segment of a track, then evaluating the total delay probability functions of signals as they propagate through entire paths. Similar statistical based tools are starting to appear for other forms of analysis, such as power analysis.

DESIGN FOR MANUFACTURABILITY (DFT)

Consider a highly simplified view of the silicon chip design flow, as illustrated in Figure 23.4. First, the designer captures the RTL that represents the functionality of the design. The RTL is then synthesized into a gate-level netlist (this will also include instantiations of hard cores from an IP macro library). Following place-and-route, the output from the design phase is a GDSII file, which is subsequently handed over to manufacturing.



FIGURE 23.4
Generic design flow.

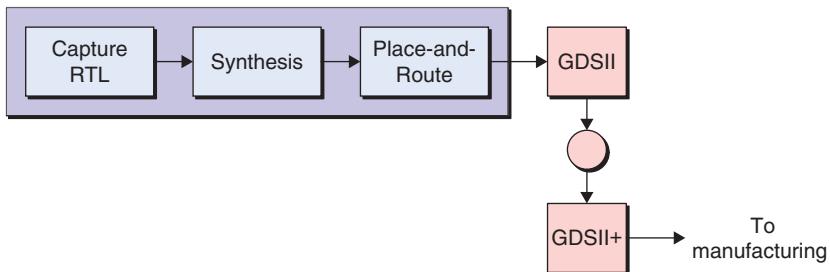
The problem is that the GDSII file generated by the design phase contains “ideal” geometric shapes. However, the wavelength of the light used to image the chip is larger than the structures that are being created in these sub-wavelength processes. This means that the ideal shapes in the original GDSII file will not print as required. Thus, part of the manufacturing process is to manipulate the GDSII file with a variety of *Resolution Enhancement Techniques* (RET), such as *Optical Proximity Correction* (OPC) and *Phase Shift Mask* (PSM).

The problem is that when these tasks are performed downstream in manufacturing, the tools that modify the design to make it manufacturable do so without understanding the design intent, so they may negatively impact things like power consumption and performance.

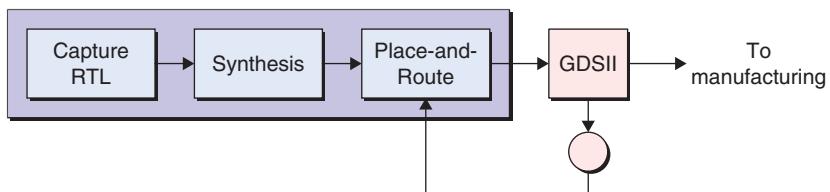
The solution is a variety of tools that fall under the umbrella name of *Design for Manufacturability* (DFM). The “D” for “Design” is important here, because it means bringing manufacturability analysis upstream into the design phase. One DFM scenario is to perform a full-chip analysis of the GDSII and to augment it with “suggestions” that can be used to guide the downstream manufacturing tools, as illustrated in Figure 23.5.

Another scenario is to perform a full-chip analysis of the GDSII looking for “hot-spots,” and to use these results to guide the place-and-route engines to modify the physical implementation, as illustrated in Figure 23.6.

In the future, it may be that the place-and-route engines are augmented to perform DFM tasks “on-the-fly.” For example, every time the placer places

**FIGURE 23.5**

Preprocessing the GDSII before handing it over to manufacturing.

**FIGURE 23.6**

Analyzing the GDSII and using the results to modify the physical implementation.

a component or the router routes a track, they will check to see if there are any manufacturability issues and—if so—they will modify their decisions appropriately.

AND SO MUCH MORE . . .

Oh, there's so much more we could talk about. The following offers a glimpse of a few of the other tools one might run across.

Schematic Synthesis

After an RTL representation of a digital design has been processed by a logic synthesis tool and turned into a gate-level netlist, another program called *schematic synthesis* can be used to take the netlist and automatically generate a gate-level schematic diagram. This graphical representation contains exactly the same information as the netlist, but it can be much easier to understand and to work with.

Analog Synthesis

In certain respects, analog circuits are significantly more complex than their digital counterparts. As a result, there really aren't any special tools that can

take high-level *Analog Hardware Description Language* (AHDL) representations and automatically generate implementation-level resistor-capacitor-inductor-transistor netlists. However, there are some niche tools available for select, well-understood circuit types, like filters. These tools are usually controlled by the engineer selecting items and completing fields on a form on the computer screen; for example, “Select filter type from this list,” “Specify required cut-off frequency,” and so forth. These tools are usually offered as add-ons to analog simulators.

RF/Microwave Design Tools

In the case of electronic designs, it's relatively easy to predict how they will work when signals transition from one value to another at a reasonable speed, and when the frequency of the system clock isn't too high. However, as signal transitions get faster and clock frequencies increase, we move into an area called *high-speed design*, where all sorts of strange effects start to make their presence felt. For example, if a track on a circuit board is not designed correctly, a clock signal might be corrupted by sharp corners in the track and can even reflect (bounce) back off the end of the track and retrace its path, falsely triggering integrated circuits on its way.

As circuit frequencies continue to increase, we move into an area called *Radio Frequency* (RF) and microwave design, where the effects become truly weird and wonderful. For example, a simple via (copper-lined hole) linking two layers on a circuit board can act like a radio antenna, broadcasting “noise” that impacts the functionality of the rest of the circuit. Similar effects occur inside integrated circuits, so design engineers and layout designers rely heavily on special analysis tools when working with these designs.

Hardware Simulation Acceleration and Emulation

As we've already discussed, design engineers have access to software simulators (computer programs). However, these programs can take a long time to simulate a gate-level representation of a high-end integrated circuit containing tens of millions of logic gates. As an alternative, design engineers can use a hardware simulator accelerator, which may be based on arrays of FPGAs (sometimes arrays of CPUs).

A special tool can be used to take the gate-level netlist for a design and use it to reconfigure the FPGAs to perform the same functions. The resulting hardware simulator accelerator can run hundreds or thousands of times faster than its software counterpart.

One point that can confuse the unwary is the difference between hardware simulation acceleration and emulation. This is a bit of a gray area, but a somewhat simplistic overview is as follows.

Hardware simulation acceleration is simply an alternative method of performing a simulation, in which the *Design Under Test* (DUT) runs faster than it would if one were using a software simulator. However, the simulation is essentially that of a single device in isolation, and not as part of a complete system.

By comparison, hardware emulation involves mapping the design under test into another piece of hardware (like an FPGA) that will run fast enough that it can be plugged into the real target system. In this case, we are no longer simulating the DUT in isolation, but are instead verifying its functionality in the context of the system for which it is intended. (It's important to note that this technique also enables engineers to verify the functionality of the whole system prior to having a physical version of the chip in question.)

Mixed-Signal Simulation

Based on our earlier discussions, we know that design engineers can use logic simulators to verify digital circuits and analog simulators to verify analog circuits. In order to simulate circuits with both digital and analog portions, it is necessary to couple the appropriate simulators together or to create a single simulator that can work in both domains. Whichever technique is employed, the result is referred to as *mixed-signal simulation* of a *mixed-signal design*.

Physical Verification (DRC, ERC, LVS)

There are a number of tools that can be used to check that the design conforms to various rules. First there are *Design Rule Checking* (DRC) programs that enforce physical rules, such as checking to see that there is sufficient clearance between tracks. By comparison, *Electrical Rule Checking* (ERC) programs inspect the design to ensure that electrical rules have been followed; for example, to make sure that the engineer hasn't overloaded a logic gate by using it to drive too many load gates. (This is similar to your plugging too many appliances like a hair dryer, food processor, computer, and television into a single power socket in your home.)

Another form of verification is known as *Layout Versus Schematic* (LVS). Once all of the logic gates and tracks forming an integrated circuit have been placed and routed, the end result is a format known as GDSII. This GDSII information describes the shapes forming the masks that will be used to create the various

layers used to build the integrated circuit. An IVS program starts with the GDSII shape information and extracts a netlist, which is compared to the original schematic.

Signal Integrity (SI) Analysis

Electronic signals on circuit boards and inside integrated circuits can be affected by different segments of track and the vias linking tracks on different levels. As we increase the speed of the signals, they can become more and more corrupted until they no longer function as required. Thus, engineers use special *Signal Integrity* (SI) programs to analyze signals and then modify track routes and characteristics so as to ensure the integrity of these signals.

Thermal Analysis

Electronics engineers can run thermal analysis programs that simulate the heat generated by each gate on an integrated circuit and each integrated circuit on a circuit board. These programs can also be used to model the effect of attaching heat sinks to the integrated circuits and passing a cooling airflow across the surface of the circuit board. Based on the results of this analysis, the engineers may decide to relocate some of the integrated circuits so as to improve the flow of air and the removal of heat across the board.

Power Analysis

People tend not to consume much energy when they are lounging around in armchairs. By comparison, they consume a lot more energy if they are jumping up and down waving their arms around, and a room will quickly heat up where there are a lot of people performing some physical activity in it.

Similarly, the logic gates forming an integrated circuit consume power when they are switching from one state to another—the faster they switch, the more power they require. Thus, engineers use power analysis tools to calculate how much power each area of the integrated circuit will consume. This allows them to (a) ensure that each part of the circuit has enough power for its needs, and (b) to make sure the chip won't get too hot.

Electromagnetic Interference and Compliance (EMI and EMC)

If you visit a hospital, you will see signs telling you not to use your cell phone. This is because the radio waves can interfere with delicate medical equipment. Similarly, if you have a pacemaker to help keep your heart beating steadily, you

are advised to keep clear of microwave ovens, because leaking radiation can interfere with the pacemaker's operation.

In fact, the tracks and vias on circuit boards and inside integrated circuits can act like antennas broadcasting electromagnetic radiation. Called "noise," this *Electromagnetic Interference* (EMI) can affect the operation of other portions of the circuit. In fact, EMI can extend beyond the confines of one system and affect other systems nearby. Thus, various governments and standard bodies define rules for the amount of EMI permitted for each type of device. The act of meeting these rules is referred to as *Electromagnetic Compliance* (EMC). Engineers may use special tools to analyze EMI, but these tools require huge amounts of computer power and are not as sophisticated (or accurate) as one might wish.

SCAN, BIST, JTAG, etc.

A common test strategy for electronic systems is called SCAN.⁴ In this case, the board has a special "scan-in" signal that is connected to the "scan-in" pin of the first integrated circuit. A corresponding "scan-out" pin from the first integrated circuit is connected to the "scan-in" pin of the second integrated circuit, and so on across the board. All of the integrated circuits are "daisy-chained" together until finally, the "scan-out" pin from the last integrated circuit is connected to a "scan-out" signal from the board. This is known as a SCAN chain.

The idea is that the operation of the system can be paused and test signals can be loaded into the various integrated circuits using the SCAN chain. Then the system can be presented with a single clock cycle and the SCAN chain can be used to read out the new contents of the integrated circuits. Using this technique, it is possible to resolve problems down to individual tracks or integrated circuits, which can be subsequently fixed or replaced, respectively.

If the SCAN chain is used only to drive and monitor the inputs and outputs of integrated circuits, then this is referred to as *boundary scan* (the primary boundary scan specification is known as JTAG). It's also possible for the scan chain to thread its way throughout each integrated circuit, which is referred to as *full scan*. Last but not least, it's possible to construct special structures inside an integrated circuit that can be used to perform a *Built-In Self-Test* (BIST). The scan chain can be used to initiate the BIST and monitor the results.

⁴SCAN is not an acronym and doesn't really stand for anything, but JTAG stands for the *Joint Test Action Group* within the *Institute of Electrical and Electronics Engineers* (IEEE). In conversation, IEEE is pronounced "eye-triple-ee."

Automatic Test Pattern Generation (ATPG)

A type of program called an *Automatic Test Pattern Generator* (ATPG) can be used to automatically create tests to check individual integrated circuits or entire circuit boards. Note that these programs are only used once a design has been finalized, and that the resulting testbenches are used to check the physical devices and circuit boards. That is, you wouldn't use one of these programs as part of the original design and verification cycle, because the ATPG program doesn't know if your circuit is good or bad—it will simply create a test for whatever circuit it is presented with.

Fault Simulation

A fault simulator is a special form of logic simulator. The fault simulator's first task is to look at the circuit and generate a list of all of the possible faults that might occur; for example, a track might break, or two tracks running side-by-side might inadvertently become connected together. The simulator then applies each of these faults to the simulation model in the virtual world and determines whether or not the testbench will detect each fault.

TURN THAT FROWN UPSIDE DOWN

And so, with our lower lips quivering and little tears rolling down our cheeks, we come to the close of this, the final chapter. But turn that frown upside down into a smile, because there are still oodles of yummy appendices containing enough mouth-watering information to form a book in their own right.

As that great British Prime Minister Winston Spencer Churchill (1874–1965) would have said: "*Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.*"⁵

⁵Speech at the Lord Mayor's Day Luncheon, London, England (November 10, 1942).

APPENDIX A

Assertion-Level Logic

BEWARE—HERE BE DRAGONS!

In ancient times, the phrase “*Beware, here be dragons!*” was penned on maps to indicate unknown territory. Today, the same words of caution should be applied to the terms *active-high* and *active-low*, which are subject to confusion. Some academics (and even textbooks) define an *active-low* signal as one whose asserted (active) state is at a *lower (more negative) voltage level* than its unasserted (inactive) state. On this basis, an *active-high* signal would be one whose asserted state is at a *higher (more positive) voltage level* than its unasserted state. Although these definitions may work most of the time, they cause confusion when combined with *negative logic implementations*.¹

377

Engineers in the trenches generally take the position that an active-high signal is one whose active/asserted state is considered to be TRUE or logic 1, while an active-low signal is one whose active state is considered to be FALSE or logic 0. These definitions allow all forms of logic—including the *assertion-level logic* introduced in this Appendix—to be represented without any confusion, regardless of whether positive or negative logic implementations are employed. These manly-man definitions are the ones used throughout this book. However, when using the terms *active-high* and *active-low* in discussions with other folks, you are strongly advised to make sure that you all understand them to mean the same thing before you find yourself up to your ears in alligators fighting fires.²

¹Negative logic implementations aren’t as common as they once were, but you never know when one will sneak up on you when you aren’t looking (see *Appendix B: Positive Versus Negative Logic* for more details).

²I never metaphor I didn’t like (grin).

STANDARD VERSUS ASSERTION-LEVEL LOGIC

The purpose of a circuit diagram is to convey the maximum amount of information in the most efficient fashion. One aspect of this is assigning meaningful names to wires; for example, naming a wire `system_reset` conveys substantially more information than calling it `big_boy`.³

Another consideration is that the signals carried by wires may be *active-high* or *active-low* (see also the previous topic). Consider a portion of a circuit containing a tri-state buffer with an active-low control input, as illustrated in Figure A.1.

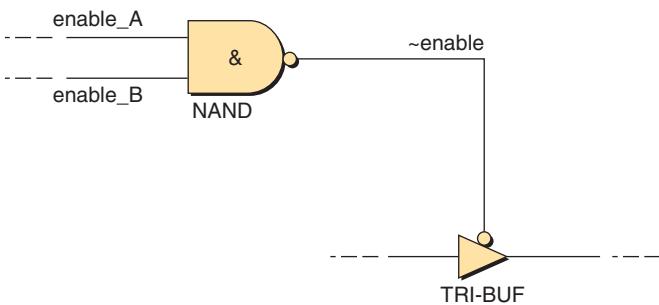


FIGURE A.1

Naming active-low signals.

In order to convey as much information as possible, it is preferable to indicate the nature of an active-low signal in its name. One method of achieving this is to prefix the name with a tilde character “~”; hence, the use of `~enable` in Figure A.1. When both the `enable_A` and `enable_B` signals feeding the NAND gate are placed in their active-high states, the `~enable` signal is driven to its active-low state and the tri-state buffer is enabled.

It is important to note that the active-low nature of the `~enable` signal is not determined by the NAND. The only thing that determines whether a signal is active-high or active-low is the way in which it is used by any target gates, such as the tri-state buffer in this example. Thus, active-low signals can be generated by ANDs and ORs as easily as NANDs and NORs.

³I could tell you some stories! I remember looking at the schematics for some computer and inquiring why a certain element was called “The Banana Register.” I was informed that this was because nothing happened for a long time, and then all of the information “came in bunches.” (I’m not joking.)

The problem with the standard symbols for BUF, NOT, AND, NAND, OR, and NOR is that they are tailored to reflect operations based on active-high signals being presented to their inputs. To address this problem, special assertion-level logic symbols can be used to more precisely indicate the function of gates with active-low inputs. For example, consider how we might represent a NOT gate used to invert an active-low $\sim\text{enable}$ signal into its active-high enable counterpart (Figure A.2).

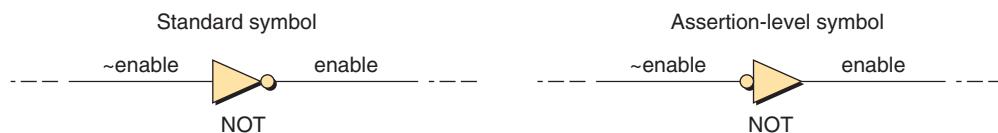


FIGURE A.2

Standard versus assertion-level NOT symbols.

Both of these symbols indicate an inversion, but the assertion-level symbol is more intuitive because it reflects the fact that an active-low input is being transformed into an active-high output. In both cases, the bubbles on the symbols indicate the act of inversion. One way to visualize this is that the symbol for a NOT has been pushed into a symbol for a BUF until only its bubble remains visible (Figure A.3).

In the real world, both standard and assertion-level symbols are implemented using identical logic gates. Assertion-level logic does not affect the final implementation, but simply offers an alternative way of viewing things. Visualizing bubbles as representing inverters is a useful technique for handling more complex functions. Consider a variation on our original circuit as illustrated in Figure A.4: in this case, the $\sim\text{enable}_A$ and $\sim\text{enable}_B$ signals are active-low and the tri-state buffer has an active-high enable input.

Now, whenever we see an AND or NAND symbol, our knee-jerk reaction is to assume that the case of particular interest is when all of its inputs are set to logic 1. The purpose of this circuit, however, is to set the enable signal to its active-high state if either of the $\sim\text{enable}_A$ or $\sim\text{enable}_B$ signals are in their active-low states. Of course, both of the circuit representations

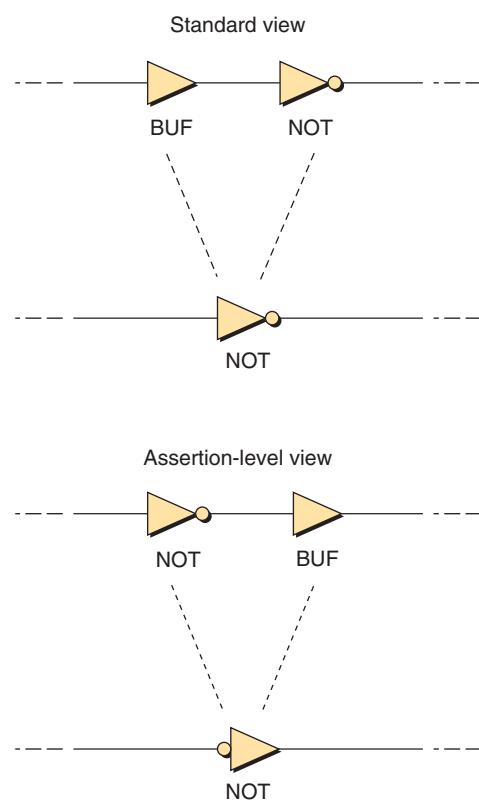
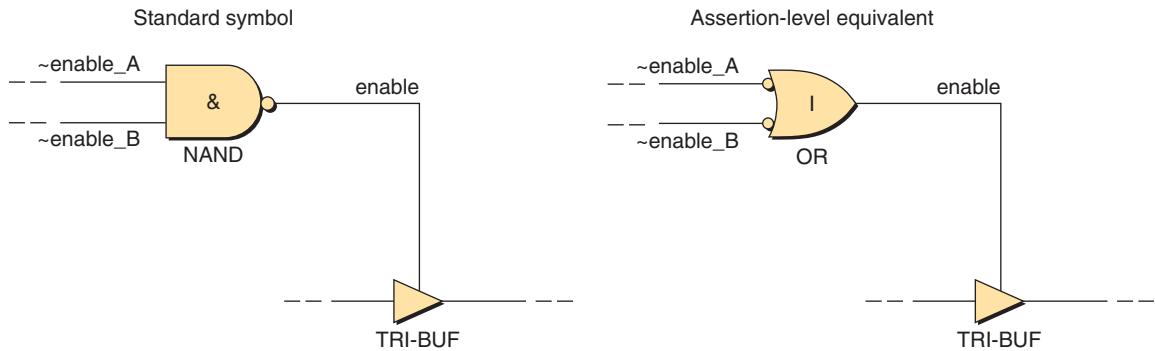


FIGURE A.3

One way to visualize things.

**FIGURE A.4**

Standard NAND versus assertion-level OR symbols.

in Figure A.4 are functionally identical (you can easily prove this by drawing out their truth tables). Once again, however, the assertion-level representation is the more intuitive, especially for someone who is unfamiliar with the function of the circuit. This is because the assertion-level symbol unambiguously indicates that the enable signal will be set to logic 1 if either of the $\sim\text{enable}_A$ or $\sim\text{enable}_B$ signals are presented with logic 0s.

Any standard primitive gate symbol can be transformed into its assertion-level equivalent by inverting all of its inputs and outputs, and then exchanging any $\&$ (AND) operators for $|$ (OR) operators (and vice versa). In fact, a quick review of *Chapter 9: Boolean Algebra* reveals that these steps are identical to those used in a DeMorgan Transformation. Thus, assertion-level symbols may also be referred to as *DeMorgan equivalent symbols*. The most commonly used assertion-level symbols are those for BUF, NOT, AND, NAND, OR, and NOR (Figure A.5).

Note that the assertion-level symbols for XOR and XNOR are identical to their standard counterparts. The reason for this is that if you take an XOR and invert its inputs and outputs, you end up with an XOR again; similarly for an XNOR. A little experimentation with their truth tables will quickly reveal the reason why.

DID SOMEONE JUST SHRIEK?

In the discussions above, we used tilde “ \sim ” characters to indicate active-low signal names; for example, $\sim\text{enable}$. As an alternative, some designers prefer to use an exclamation mark “!” (nicknamed a “shriek”), as in, $!\text{enable}$.

Similarly, both tilde and shriek characters may also be used to indicate complementary outputs; for example, the q and $\sim q$ (or $!q$) outputs from a latch or a flip-flop.

Name	Standard equation and symbol	Assertion-level equivalent
BUF	$y = a$	
NOT	$y = \bar{a}$	
AND	$y = a \& b$	
NAND	$y = \bar{a} \& \bar{b}$	
OR	$y = a \mid b$	
NOR	$y = \bar{a} \mid \bar{b}$	
XOR	$y = a \wedge b$	
XNOR	$y = \bar{a} \wedge \bar{b}$	

FIGURE A.5

Commonly used assertion-level symbols.

Either of these is great ... you just have to be careful when communicating with other people, because different folks may understand these symbols to mean different things.

For example, consider an equation like $y = a \& \text{!}b$. Throughout the course of this book, we've used the $\&$ symbol to represent an AND function, but what about the ! symbol? Some folks might consider this to represent a NOT gate (think in terms of $y = a$ AND NOT b), while others may consider it to be part of an active-low signal name (think in terms of $y = a$ AND $\text{!}b$).

Actually, this is also true of the tilde character. Furthermore, in addition to communicating with other people, we might also be entering these equations as text files that are to be processed by computer-aided tools. In this case, these tools may consider the \sim and ! characters to represent logical functions.

Yet another technique is to draw a horizontal line, or bar, over the name, but this is not recommended for a number of reasons. In addition to the fact that horizontal lines are difficult to replicate in textual form on a computer, such bars are often used to indicate negations when we're writing Boolean equations

(see the equations we used in Figure 9.11 for the DeMorgan Transformation of an AND function, for example).

One way to avoid confusion is to avoid using special characters like “ \sim ” and “ $!$ ” and instead prefix or postfix active-low signal names with a special letter; for example, `Nenable` or `enableN`. Ultimately, you have to either (a) decide what works for you or (b) use whatever techniques are promoted by the company or organization for which you work.

APPENDIX B

Positive Versus Negative Logic

ARE YOU POSITIVE ABOUT THAT?

The terms *positive logic* and *negative logic* refer to two conventions that dictate the relationship between logical values and the physical voltages used to represent them. Unfortunately, although the core concepts are relatively simple, fully comprehending all of the implications associated with these conventions requires an exercise in lateral thinking sufficient to make even a strong man break down and weep!

Before plunging into the fray, it is important to understand that logic 0 and logic 1 are always equivalent to FALSE and TRUE, respectively.¹ The reason these terms are used interchangeably is that digital functions can be considered to represent either *logical* or *arithmetic operations* (Figure B.1).

383

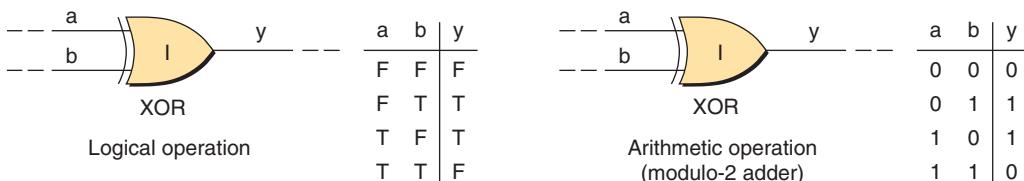


FIGURE B.1

Logical versus arithmetic views of a digital function.

Having said this, it is generally preferable to employ a single consistent format to cover both cases, and it's easier to view logical operations in terms of 0s and 1s than it is to view arithmetic operations in terms of Fs and Ts. The key point to remember as we go forward is that, by definition, logic 0 and logic 1 are logical concepts that have no direct relationship to any physical values.

¹Unless you're really taking a walk on the wild side, in which case all bets are off.

PHYSICAL TO LOGICAL MAPPING (NMOS LOGIC)

Let's gird up our loins and meander our way through the morass one step at a time ... The process of relating logical values to physical voltages begins by defining the frames of reference to be used. One absolute frame of reference is provided by truth tables, which are always associated with specific functions (Figure B.2).

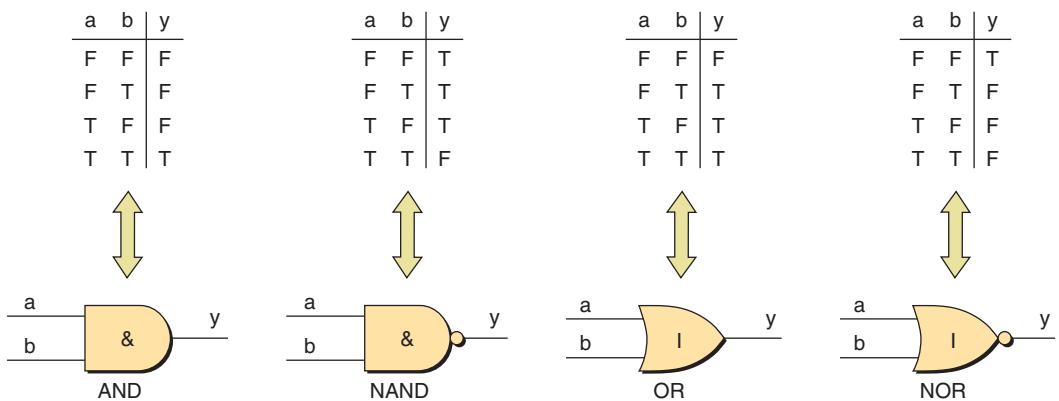


FIGURE B.2

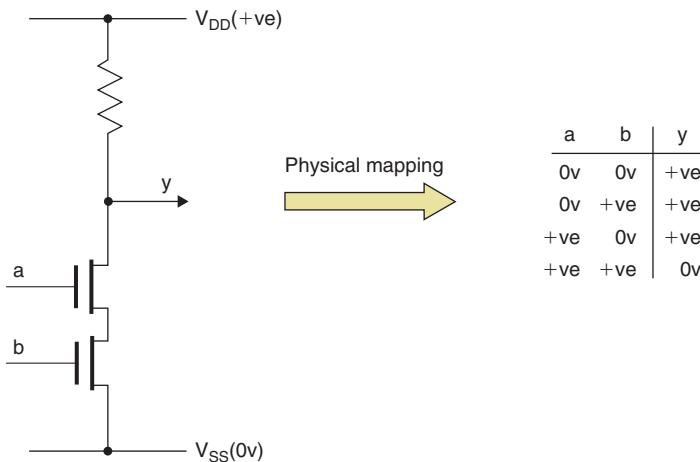
Absolute relationships between logical functions and their truth tables.

Another absolute frame of reference is found in the physical world, where specific voltage levels applied to the inputs of a digital function cause corresponding voltage responses on the outputs. These relationships can also be represented in truth table form. Consider a logic gate constructed using only NMOS transistors (Figure B.3).

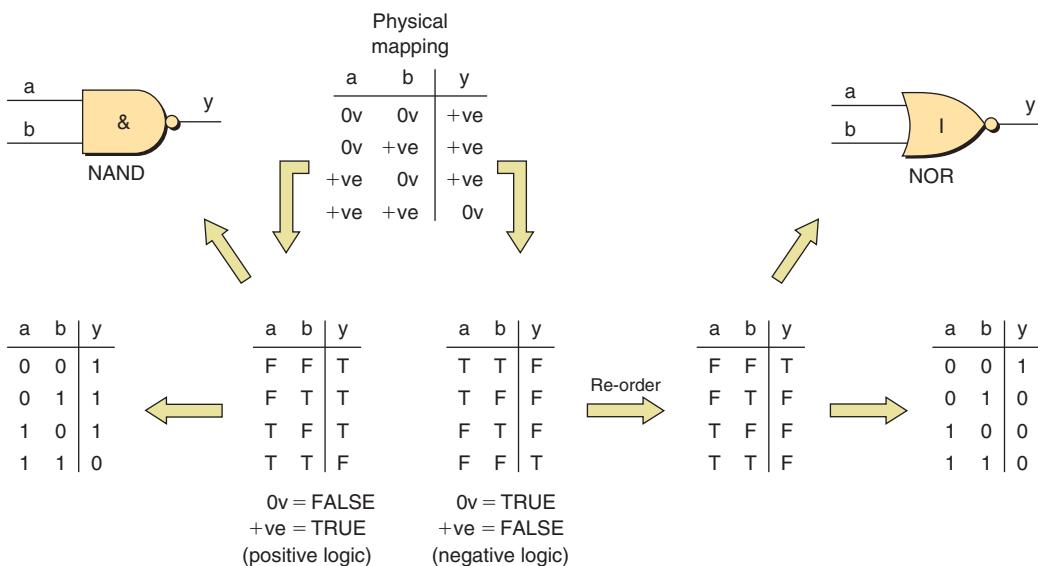
With NMOS transistors connected as shown in Figure B.3, an input connected to the more negative V_{SS} , turns that transistor OFF, and an input connected to the more positive V_{DD} turns that transistor ON. The final step is to define the mapping between the physical and logical worlds; either 0V is mapped to FALSE and +ve (the more positive supply rail) is mapped to TRUE, or vice versa (Figure B.4).

Using the *positive logic convention*, the more positive potential is considered to represent TRUE² and the more negative potential is considered to represent FALSE. By comparison, using the negative logic convention, the more negative

²Hence, positive logic is also known as *positive-true*.

**FIGURE B.3**

The physical mapping of an NMOS logic gate.

**FIGURE B.4**

The physical-to-logical mapping of our NMOS logic gate.

potential is considered to represent TRUE³ and the more positive potential is considered to represent FALSE. Thus, this circuit may be considered to be performing either a NAND function in positive logic or a NOR function in negative logic. (Are we having fun yet?)

³Hence, negative logic is also known as *negative-true*.

PHYSICAL TO LOGICAL MAPPING (PMOS LOGIC)

From the previous example it would appear that positive logic is the more intuitive, as it is easy to relate logic 0 to 0V (no volts) and logic 1 to +ve (presence of volts). On this basis, one may wonder why negative logic ever reared its ugly head. The answer to this, as are so many things, is rooted in history. When the MOSFET technology was originally developed, PMOS transistors were easier to manufacture and were more reliable than their NMOS counterparts (in the very early days, PMOS transistors were the only ones that worked at all; due to a variety of problems, the first NMOS transistors were permanently ON, which wasn't much use to anyone). Thus, the majority of early MOSFET-based logic gates were constructed from combinations of PMOS transistors and resistors. Consider a logic gate constructed using only PMOS transistors (Figure B.5).

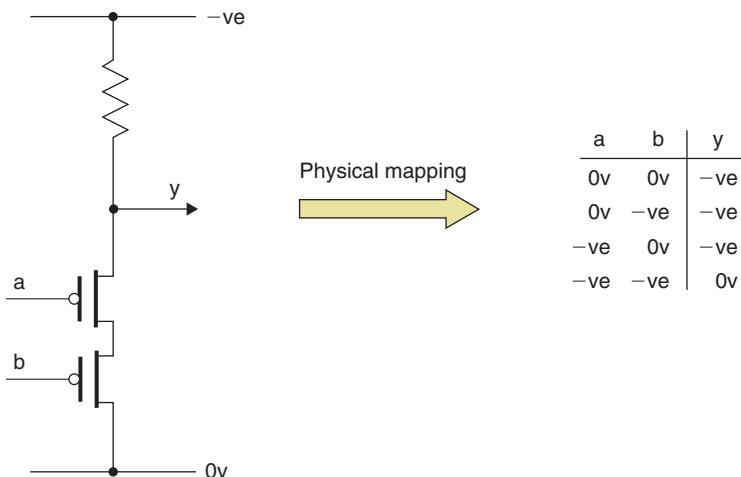
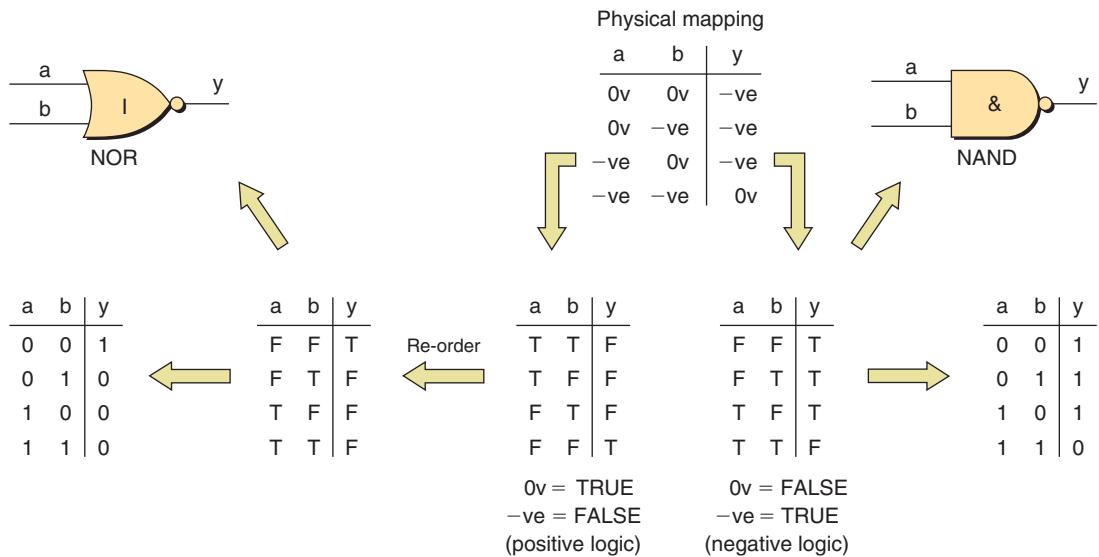


FIGURE B.5

The physical mapping of a PMOS gate.

Circuits constructed using the original PMOS transistors typically used a negative power supply; that is, a power supply with a 0V rail and a negative (*-ve*) rail. With PMOS transistors connected as shown in Figure B.5, an input connected to the more positive 0V rail turns that transistor OFF, while an input connected to the more negative *-ve* rail turns that transistor ON. Once again, the final step is to define the mapping between the physical and logical worlds; either 0V is mapped to FALSE and *-ve* is mapped to TRUE, or vice versa (Figure B.6).

**FIGURE B.6**

The physical-to-logical mapping of our PMOS logic gate.

Thus, this circuit may be considered to be performing either a NOR function in positive logic or a NAND function in negative logic. In this case, negative logic is the more intuitive, as it is easy to relate logic 0 to 0V (no volts) and logic 1 to -ve (presence of volts). Additionally, the physical structure of the PMOS gate is identical to that of the NMOS gate; if the NMOS gate is represented in positive logic and the PMOS gate is represented in negative logic, then both representations equate to NAND functions, which, if nothing else, is aesthetically pleasing.

This page intentionally left blank

APPENDIX C

Reed-Müller Logic

“BUT THAT’S NOT LOGICAL, CAPTAIN!”

Some digital functions can be difficult to optimize if they are represented in the conventional *sum-of-products* or *product-of-sums* forms,¹ which are based on ANDs, ORs, NANDs, NORs, and NOTs. In certain cases it may be more appropriate to implement a function in a form known as Reed-Müller logic, which is based on XORs and XNORs. One indication as to whether a function is suitable for the Reed-Müller form of implementation is if that function’s Karnaugh Map displays a checkerboard pattern of 0s and 1s. Consider a familiar two-input function (Figure C.1).

389

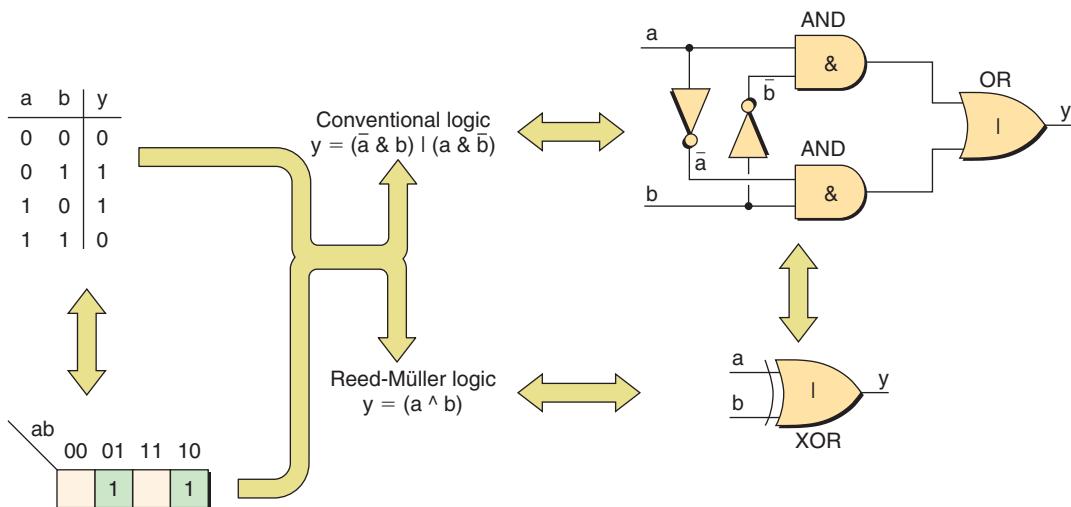


FIGURE C.1

2-input function suitable for Reed-Müller implementation.

¹The concepts of sum-of-products and product-of-sums were introduced in *Chapter 9: Boolean Algebra*.

Since the above truth table is easily recognizable as being that for an XOR function, it comes as no great surprise to find that implementing it as a single XOR gate is preferable to an implementation based on multiple AND, OR and NOT gates.² A similar checkerboard pattern may apply to a three-input function (Figure C.2).

As XORs are both commutative³ and associative,⁴ it doesn't matter which combinations of inputs are applied to the individual gates. The checkerboard pattern for a four-input function continues the theme (Figure C.3).

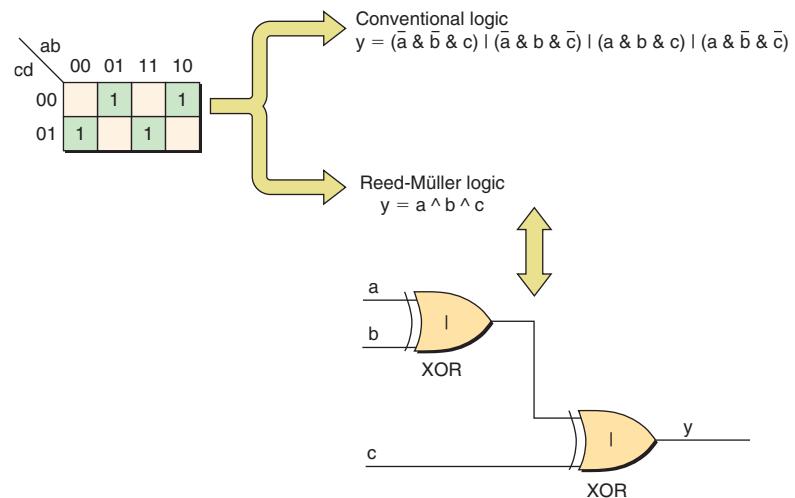


FIGURE C.2

3-input function suitable for Reed-Müller implementation.

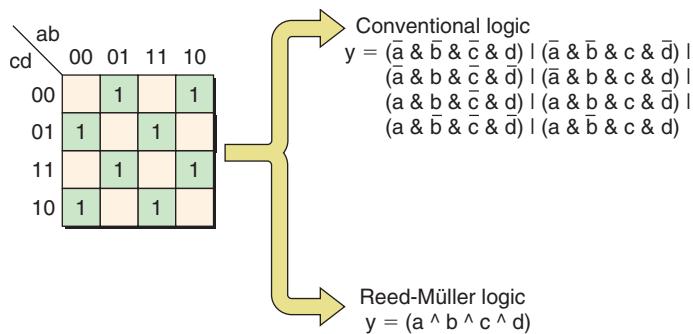


FIGURE C.3

4-input function suitable for Reed-Müller implementation.

²It's common to use “^” characters in Boolean equations to indicate XOR functions.

³For example, $(a ^ b) \equiv (b ^ a)$. (Remember that the “≡” symbol means “... is equivalent to ...”)

⁴For example, $(a ^ b) ^ c \equiv a ^ (b ^ c)$.

Larger checkerboard patterns involving groups of 0s and 1s also indicate functions suitable for a Reed-Müller implementation (Figure C.4).

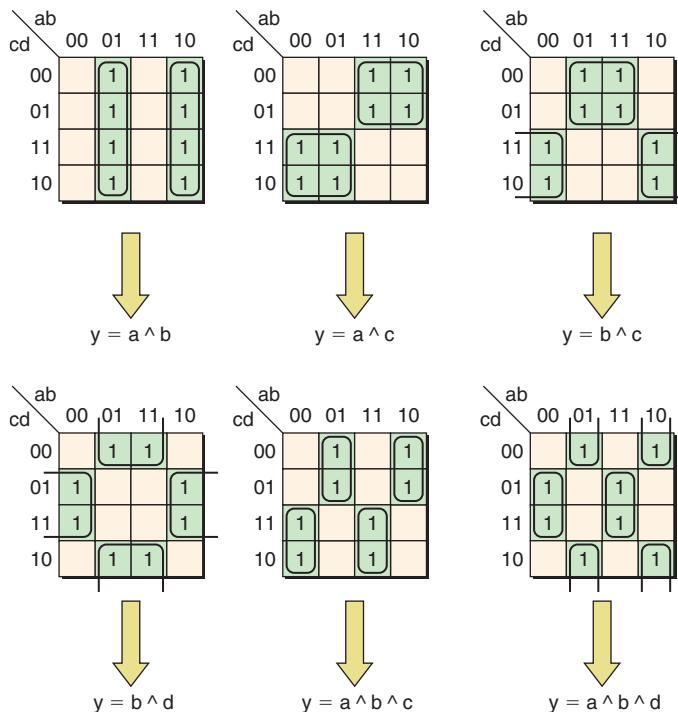


FIGURE C.4

Example functions suitable for Reed-Müller implementations.

Once you have recognized a checkerboard pattern, there is a quick “rule of thumb” for determining the variables to be used in the Reed-Müller implementation. Select any group of 0s or 1s and identify the significant and redundant variables,⁵ and then simply XOR the significant variables together.

Although we don’t bother entering logic 0 values into our Karnaugh maps, we know that any empty squares really contain 0s. As all of the checkerboard patterns we’ve considered this far include a logic 0 in the box in the upper left corner,⁶ the resulting Reed-Müller implementations can be realized using only XORs. Having said this, any pair of XORs may be replaced with XNORs, the only requirement being that there is an *even number* of XNORs.

⁵The significant variables are those whose values are the same for all of the boxes forming the group, while the redundant variables are those whose values vary between boxes.

⁶Corresponding to all of the inputs being logic 0.

Alternatively, if the checkerboard pattern includes a logic 1 in the box in the upper left corner, the Reed-Müller implementation must contain an *odd number* of XNORs. Once again, it does not matter which combinations of inputs are applied to the individual XORs and XNORs (Figure C.5).

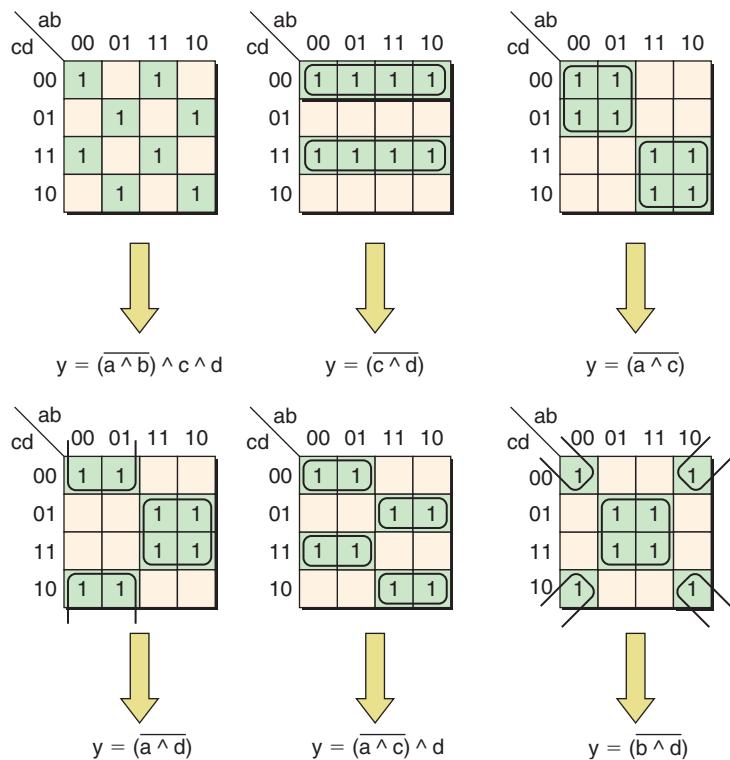


FIGURE C.5

Example Reed-Müller implementations requiring XNOR gates.

Although these examples provide a very limited introduction to the concept of Reed-Müller logic, checkerboard Karnaugh Map patterns are easy to recognize and appear surprisingly often. Reed-Müller implementations are often appropriate for circuits performing arithmetic or encoding functions.⁷

⁷Example encoding and decoding functions suitable for Reed-Müller implementation are presented in Appendix D: *Gray Codes*.

APPENDIX D

Gray Codes

GRAY, BUT NOT GLOOMY

When moving between states in a standard binary sequence, multiple bits may change from 0 to 1 or vice versa; for example, two bits change value when moving from 0010_2 to 0100_2 . In the physical world there is no way to ensure that both bits will transition at exactly the same time, so this system may actually pass through an intermediate state. That is, our intended state change of 0010_2 to 0100_2 might result in the sequence 0010_2 to 0110_2 to 0100_2 , or possibly 0010_2 to 0000_2 to 0100_2 . And if more bits are changing, we might bounce through a series of intermediate values.

One way to avoid this problem is to use a *Gray code*,¹ in which only a single bit changes when moving between states (Figure D.1).

Gray codes are of use for a variety of applications, such as facilitating error correction in digital communications and the ordering of the input variables on Karnaugh Maps.² Another application (the task for which Gray codes were originally designed) is encoding the angle of a mechanical shaft, where a disc attached to the shaft is patterned with areas of conducting material (Figure D.2).

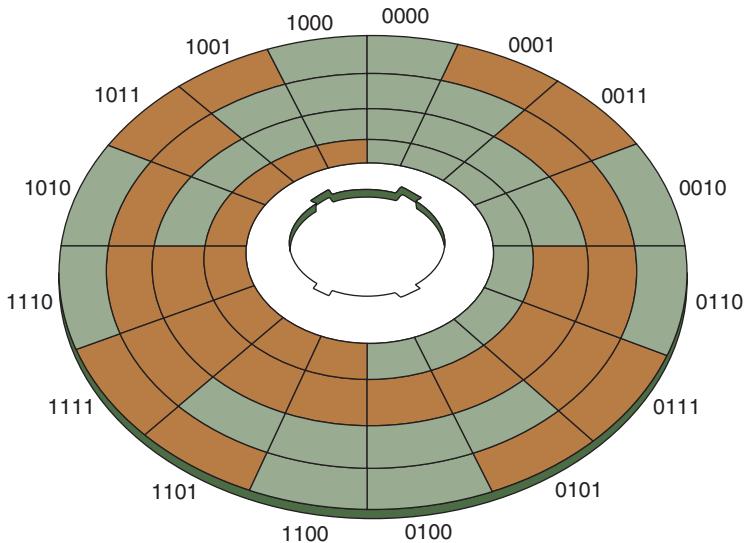
FIGURE D.1
Binary versus Gray codes.

	b[3:0]	g[3:0]
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

393

¹Note that the correct grammar and spelling for Gray codes is “Gray” (not “gray,” “grey,” or “Grey”). This is because Gray codes are named after American physicist and researcher Frank Gray, who patented their use for shaft encoders in 1953.

²Karnaugh maps were introduced in *Chapter 10: Karnaugh Maps*. Also, you can see examples of this ordering in Figure D.4 and Figure D.5.

**FIGURE D.2**

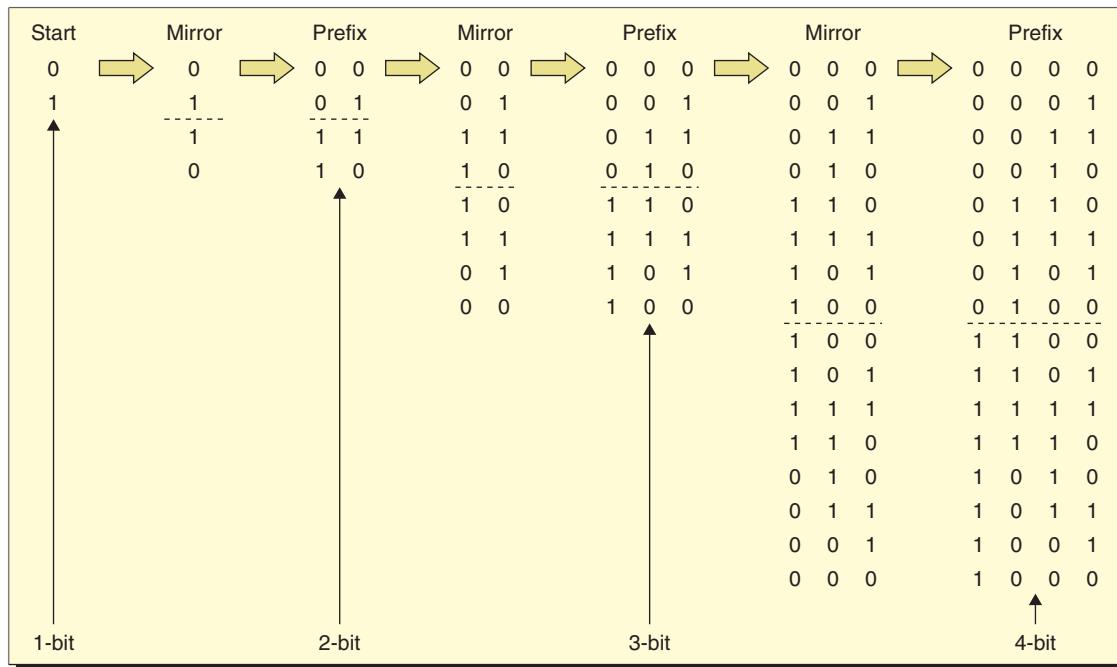
Gray code used for shaft-angle encoding.

The conducting areas are arranged in concentric circles, where each circle represents a binary digit. A set of electrical contacts, one for each of the circles, is used to detect the logic values represented by the presence or absence of the conducting areas. A digital controller can use this information to determine the angle of the shaft. The precision of the measurement depends on the number of bits (circles). A 4-bit value representing 16 unique states allows the shaft's angle to be resolved to 22.5 degrees, while a 10-bit value representing 1024 unique states allows the shaft's angle to be resolved to 0.35 degrees. Using a Gray code sequence to define the conducting and nonconducting areas ensures that no intermediate values are generated as the shaft rotates.

GENERATING A GRAY CODE

Commencing with a state of all zeros, a Gray code can be generated by always changing the least significant bit that results in a new state. An alternative method which may be easier to remember and use is as follows:

- Commence with the simplest Gray code possible; that is, for a single bit.
- Create a mirror image of the existing Gray code below the original values.

**FIGURE D.3**

Using a mirroring process to generate a 4-bit Gray code.

- Prefix the original values with 0s and the mirrored values with 1s.
- Repeat the last two steps until the desired width is achieved.

An example of this mirroring process³ used to generate a 4-bit Gray code is shown in Figure D.3.

BINARY-TO-GRAY AND GRAY-TO-BINARY

It is often required to convert a binary sequence into a Gray code or vice versa. Such converters are easy to create and are of especial interest here due to their affinity to the Reed-Müller implementations that were introduced in Appendix C: *Reed-Müller Logic*. Consider a *Binary-to-Gray* converter (Figure D.4).

The checkerboard patterns of 0s and 1s in the Karnaugh Maps immediately indicate the potential for Reed-Müller implementations. Similar checkerboard patterns are also seen in the case of a *Gray-to-Binary* converter (Figure D.5).

³It might be more correct to call this a “recursive reverse-and-prefix” technique.

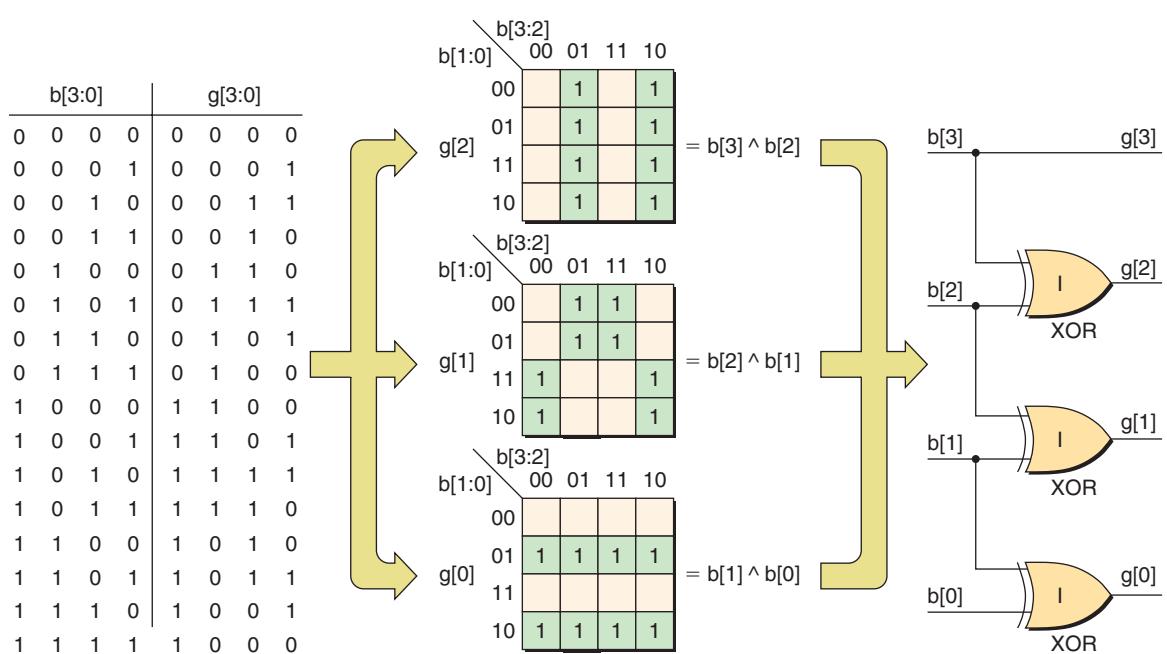


FIGURE D.4

Binary-to-Gray converter.

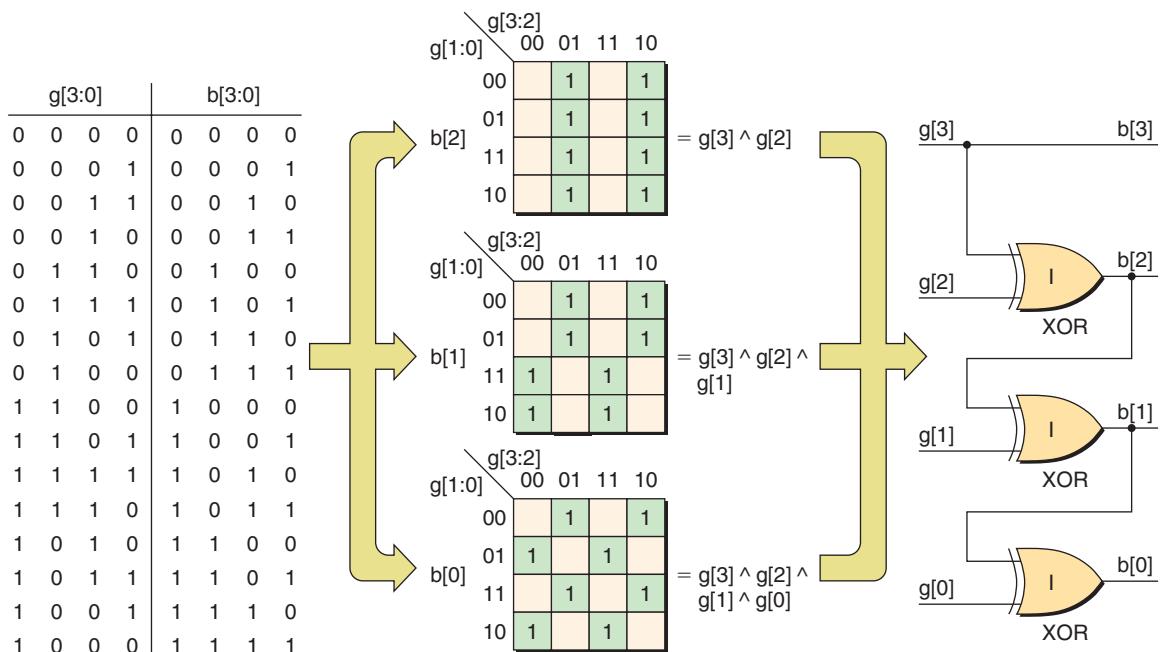


FIGURE D.5

Gray-to-Binary converter.

IT'S TOO NOISY IN HERE!

Gray code counters are of interest for a variety of applications, such as representing the state variables in state machines⁴ or acting as pointers in *First-In First-Out* (FIFO)⁵ memories. This is because only one output bit is ever toggling at a time in a Gray code “counter,” as opposed to possibly multiple bits in a binary counter.

In addition to preventing intermediate states, Gray code counters consume only half the power of an equivalent binary counter, and they generate correspondingly less noise. Actually, while the power and *average noise* difference between a Gray and a binary counter asymptotically approaches two, the *peak noise* difference is equal to the number of bits, since a Gray counter toggles only one bit at a time while a binary counter toggles all of its bits simultaneously two times over the course of a full-count cycle, with fewer bits toggling proportionally more times.

ACTUALLY GENERATING A GRAY CODE

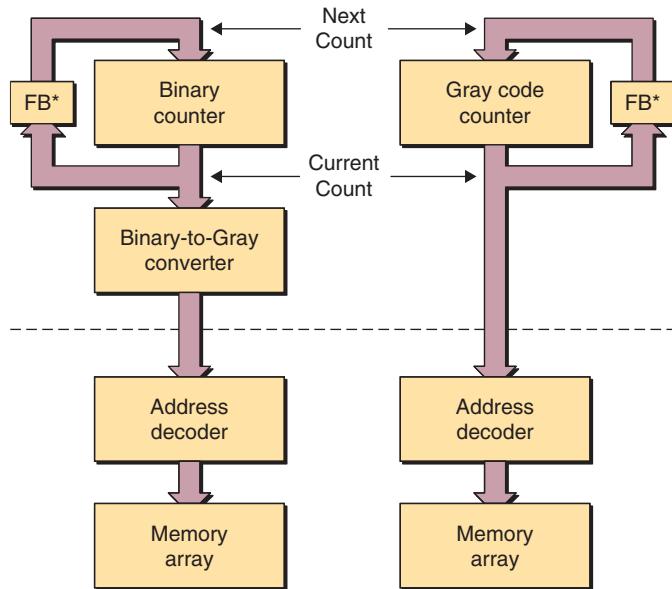
Before we proceed, let’s briefly ponder the process of actually generating a Gray code. Just to give us something to play with, let’s suppose we wish to generate a Gray code that can be used to index into a memory array. For example, suppose we’re implementing something sort-of like a FIFO, and that (for the purposes of this example), we simply wish to keep on cycling through all of the memory locations. The point is that we aren’t particularly concerned as to the order in which we address the locations, just that we sequence our way through them visiting each one a single time before returning to the first location to do it all over again.

Assuming that we’re dealing with a 16-word memory array, one scenario [as illustrated in Figure D.6(a)] starts with a standard 4-bit binary counter, in which the current count value is passed through a chunk of feedback logic to generate the next count value. Also, the current count value is passed through a Binary-to-Gray converter (as introduced in Figure D.4) to generate a corresponding Gray code.

The alternative technique [as illustrated in Figure D.6(b)] is to simply create a Gray code counter from the ground up. Now, if we were to compare the binary

⁴State machines were introduced in *Chapter 12: State Machines*.

⁵FIFO memories are presented in more detail in *Appendix E: Linear Feedback Shift Registers (LFSRs)*.



*FB = Feedback logic used to generate the next count value

a) Using a binary counter with a B-to-G converter

b) Using a Gray code counter directly

FIGURE D.6

Two techniques for generating a Gray code sequence.

counter (including binary-to-Gray converter) with the Gray code counter, there are several things I don't know off the top of my head:

- How many logic gates are required for each implementation?
- How many levels of logic gates are there in the two feedback paths?
- What's the (relative) maximum frequency of each type of counter?
- What's the (relative) switching activity, noise, and power consumption of each type of counter?

I tell you, I'm constantly amazed by the number of things I don't know. Now, we could work this out, but I'm a tad busy at the moment, so we'll leave the pondering of these posers as an exercise for the reader (grin).

GENERATING SUB- 2^n SEQUENCES

Just to make sure we're all tap-dancing to the same drumbeat, let's briefly set the scene. Consider a 4-bit binary counter. As we've already discussed, multiple

bits may change when transitioning from one value to another. For example, four bits change when one of the pointers transitions from 0111_2 to 1000_2 . If multiple bits transitioning will cause us problems, we may decide to use a Gray code counter, in which only one bit changes as we transition from one value to another (Figure D.7).

Observe that when we reach the final (maximum) Gray code value of 1000_2 , the next count will return us to our initial value of 0000_2 , which means that—as we expect—only a single bit changes for this transition also. Also observe that we've shown the hexadecimal values associated with each binary pattern in bold (we'll return to consider in these values in the next topic).

Now, suppose—instead of sequencing through all 16 values—that for some reason we actually wish to cycle through only 10 words. If we use our original Gray code as shown in Figure D.6, the sequence will now be as follows: 0000_2 , 0001_2 , 0011_2 , 0010_2 , 0110_2 , 0111_2 , 0101_2 , 0100_2 , 1100_2 , 1101_2 . The problem is that three bits will change value on the next transition, which will return us to our starting value of 0000_2 from our current value of 1101_2 , which means that this last transition is not a Gray code.

So, is it possible to create a Gray code sequence for any nonpower-of-2 number (so long as it is an even number)? Let us see ...

Throwing Away Adjacent Pairs

Using our original Gray code as the base code, wherever you have a pair of adjacent states where the least-significant bit does not change, then the states before and after such a pair always differ by exactly one bit, as illustrated in Figure D.8.

For example, consider the first four codes in the left-hand table: 0000_2 , 0001_2 , 0011_2 , and 0010_2 . If we remove the two shaded codes (0001_2 and 0011_2), we are left with 0000_2 and 0010_2 , which differ by only one bit. So by removing any such pair from the left-hand table we have a 14-count sequence; removing any

b16[3:0]	g16[3:0]
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F
(a) Binary sequence	(b) Gray code sequence

FIGURE D.7
Binary versus Gray codes.

Pairs where LSB is 1	Pairs where LSB is 0
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 1	0 0 1 1
0 0 1 0	0 0 1 0
0 1 1 0	0 1 1 0
0 1 1 1	0 1 1 1
0 1 0 1	0 1 0 1
0 1 0 0	0 1 0 0
1 1 0 0	1 1 0 1
1 1 0 1	1 1 1 1
1 1 1 1	1 1 1 0
1 1 1 0	1 0 1 0
1 0 1 0	1 0 1 1
1 0 1 1	1 0 0 1
1 0 0 1	1 0 0 0
1 0 0 0	

Remove one or more of the shaded pairs

FIGURE D.8

Throwing away adjacent pairs.

two pairs gives us a 12-count sequence; and removing any three pairs gives us a 10-count sequence. (It would be pointless to remove four pairs to give us an 8-count sequence, because we could achieve the same effect by dropping down to a 3-bit Gray code.)

In fact, we can mix-and-match to some extent, because we could remove one pair of codes whose least-significant bit was 1 and another pair whose least-significant bit was 0, so long as these pairs are not themselves adjacent to each other.

Pruning the Middle

Our previous solution is easy to use by hand, but it's not great as the basis for an algorithmic approach, because it would require us to keep track of which pairs of codes we've removed.

In fact, the solution is rather simple. Remember that we generated our original 4-bit Gray code using what I call the "mirroring method"? Well, it's possible to

16-count	14-count	12-count	10-count
0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
0 0 0 0 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1
0 0 0 1 1	0 0 0 1 1	0 0 0 1 1	0 0 0 1 1
0 0 0 1 0	0 0 0 1 0	0 0 0 1 0	0 0 0 1 0
0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0
0 1 1 1 1	0 1 1 1 1	0 1 1 1 1	0 1 1 1 1
0 1 0 1 1	0 1 0 1 1	0 1 0 1 1	0 1 0 1 1
0 1 0 0 0	0 1 0 0 0	0 1 0 0 0	0 1 0 0 0
1 1 0 0 0	1 1 0 0 0	1 1 0 0 0	1 1 0 0 0
1 1 0 1 1	1 1 0 1 1	1 1 0 1 1	1 1 0 1 1
1 1 1 1 1	1 1 1 1 1	1 1 1 1 1	1 1 1 1 1
1 1 1 1 0	1 1 1 1 0	1 1 1 1 0	1 1 1 1 0
1 0 1 0 0	1 0 1 0 0	1 0 1 0 0	1 0 1 0 0
1 0 1 1 1	1 0 1 1 1	1 0 1 1 1	1 0 1 1 1
1 0 0 1 1	1 0 0 1 1	1 0 0 1 1	1 0 0 1 1
1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0

"Mirror" line

FIGURE D.9

Pruning the middle of the sequence.

simply remove pairs of entries from the center of the table around the “mirror line” (Figure D.9).

If we desire a 14-count sequence, for example, we simply remove two entries from the middle—the one immediately above the “mirror” line and one immediately below. Similarly, if we are looking for a 12-count sequence, we remove two entries above the “mirror” line and two below, and so forth.

Pruning the Ends

The funny thing about digital logic is that there’s almost always multiple ways of doing anything. For example, the logical counterpart to the previous solution is to remove the same numbers of entries from the top and from the bottom of a Gray code sequence (Figure D.10).

In this case, if we desire a 14-count sequence, we simply remove one entry from the top of the table and one from the bottom; if we are looking for a 12-count sequence, we remove two entries from the top and two from the bottom, and so forth.

16-count	14-count	12-count	10-count
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1
0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1
0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0
0 1 1 0	0 1 1 0	0 1 1 0	0 1 1 0
0 1 1 1	0 1 1 1	0 1 1 1	0 1 1 1
0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1
0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0
1 1 0 1	1 1 0 1	1 1 0 1	1 1 0 1
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
1 1 1 0	1 1 1 0	1 1 1 0	1 1 1 0
1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0
1 0 1 1	1 0 1 1	1 0 1 1	1 0 1 1
1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1
1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0

"Mirror" line

FIGURE D.10

Pruning the ends of the sequence.

GENERATING SUB- 2^n SEQUENCES WITH CONSECUTIVE VALUES

With regard to the previous topic in which we generated nonpower-of-2 sequences; the solutions we came up with will be perfectly OK for some applications, but there may be problems if we want to use them for certain tasks, such as pointers into memory arrays.

Let's suppose that—as opposed to a full 2^n count (16 values in the case of the 4-bit examples we've been playing with)—we wish to use a reduced count sequence, such as 14, 12, or 10. Obviously this is easy-peasy when using pure binary addressing schemes, as illustrated in Figure D.11.

As we see, all we have to do is to remove states from the end of the count sequence and to implement corresponding smaller arrays of memory locations.

But let's now consider the case of the various Gray code implementations. As we discussed in the previous topic, there are several approaches we can use in order to generate a reduced-count Gray code sequence. One of these is to simply remove pairs of values from either side of the "mirror line" as illustrated in Figure D.12.

b16[3:0]	b14[3:0]	b12[3:0]	b10[3:0]
0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
1 0 0 0 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1
2 0 0 1 0	0 0 0 1 0	0 0 0 1 0	0 0 0 1 0
3 0 0 1 1	0 0 0 1 1	0 0 0 1 1	0 0 0 1 1
4 0 1 0 0	0 0 1 0 0	0 0 1 0 0	0 1 0 0 0
5 0 1 0 1	0 0 1 0 1	0 0 1 0 1	0 1 0 0 1
6 0 1 1 0	0 0 1 1 0	0 0 1 1 0	0 1 1 0 0
7 0 1 1 1	0 0 1 1 1	0 0 1 1 1	0 1 1 1 0
8 1 0 0 0	0 1 0 0 0	0 1 0 0 0	1 0 0 0 0
9 1 0 0 1	0 1 0 0 1	0 1 0 0 1	1 0 0 0 1
A 1 0 1 0	0 1 0 1 0	0 1 0 1 0	1 0 1 0 0
B 1 0 1 1	0 1 0 1 1	0 1 0 1 1	1 0 1 1 1
C 1 1 0 0	0 1 1 0 0	0 1 1 0 0	1 1 0 0 0
D 1 1 0 1	0 1 1 0 1	0 1 1 0 1	1 1 0 1 0
E 1 1 1 0	0 1 1 1 0	0 1 1 1 0	1 1 1 0 0
F 1 1 1 1	0 1 1 1 1	0 1 1 1 1	1 1 1 1 1
(a) 16-count	(b) 14-count	(c) 12-count	(d) 10-count

FIGURE D.11

Implementing reduced binary sequences.

16-count	14-count	12-count	10-count
0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
0 0 0 0 1	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1
0 0 0 1 1	0 0 0 1 1	0 0 0 1 1	0 0 1 1 1
0 0 1 0 0	0 0 1 0 0	0 0 1 0 0	0 0 1 0 0
0 0 1 0 1	0 0 1 0 1	0 0 1 0 1	0 1 1 0 0
0 0 1 1 0	0 0 1 1 0	0 0 1 1 0	0 1 1 0 1
0 0 1 1 1	0 0 1 1 1	0 0 1 1 1	0 1 1 1 1
0 1 0 0 1	0 1 0 0 1	0 1 0 0 1	0 1 0 0 1
0 1 0 0 0	0 1 0 0 0	0 1 0 0 0	0 1 0 0 0
1 1 0 0 0	1 1 0 0 0	1 1 0 0 0	1 1 0 0 0
1 1 0 0 1	1 1 0 0 1	1 1 0 0 1	1 1 0 0 1
1 1 0 1 1	1 1 0 1 1	1 1 0 1 1	1 1 1 1 1
1 1 1 0 0	1 1 1 0 0	1 1 1 0 0	1 1 1 1 0
1 0 1 0 0	1 0 1 0 0	1 0 1 0 0	1 0 1 0 0
1 0 1 0 1	1 0 1 0 1	1 0 1 0 1	1 0 1 1 1
1 0 0 1 1	1 0 0 1 1	1 0 0 1 1	1 0 0 1 1
1 0 0 0 1	1 0 0 0 1	1 0 0 0 1	1 0 0 0 1
1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0

"Mirror" line

FIGURE D.12

One way to implement reduced Gray code sequences.

OK so far, but now let's consider what happens when we apply one of these reduced sequences to our memory addressing logic; for example, let's take our 10-count example. Basically we are going to end up with "holes" in our memory array, as illustrated in Figure D.13.

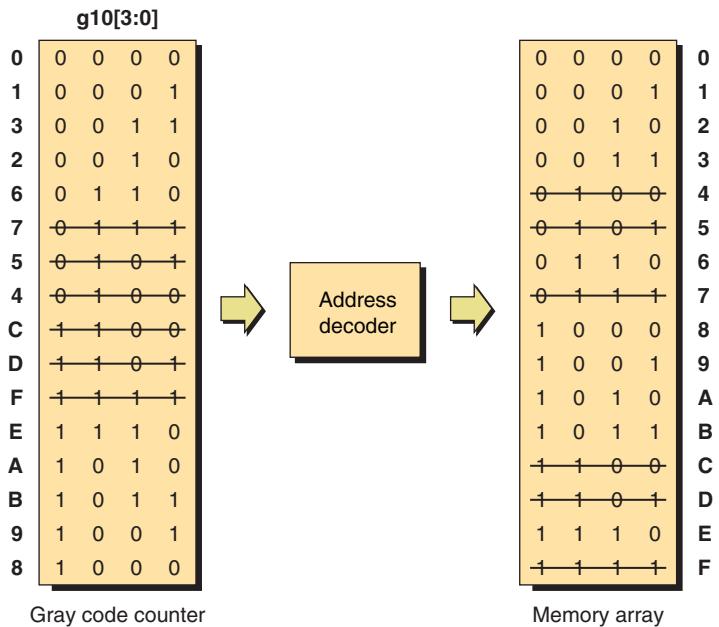


FIGURE D.13

This approach leaves "holes" in our memory array.

As we see, the sequence followed by this Gray code counter is: 0, 1, 3, 2, 6, E, A, B, 9, 8, and it then cycles back to 0 again. Each step is a Gray code (single-bit) transition, including the wrap-around from 8 to 0. (The right-hand side of Figure D.13 simply reflects which locations are hit in the memory array—not the order in which they are hit.)

The end result is that it is no longer a trivial task to create a cut-down memory array, because these "holes" are going to mess everything up. So, the real question (the "Crux of the Biscuit" as it were),⁶ is: *"Is it possible to create a Gray code sequence to address a 10-word FIFO using sequential addresses of 0-to-9?"*

Well, here's a technique that Mike Jarvis, a Design Engineer at Cray, taught me. First of all we create an empty table with the number of entries we desire: ten

⁶The full quote by American composer, guitarist, singer, film director, and satirist Frank Vincent Zappa is: *"The crux of the biscuit is the apostrophe."* (I actually know what he meant by this, but that's a story for another day.)

in the case of this particular example. We set the least-significant bit to 0 for all of the entries in the upper half of the table, and to 1 for all of the entries in the lower half of the table [Figure D.14(a)].

g10[3:0]	g10[3:0]	g10[3:0]
- - - 0	- - - 0	4 0 1 0 0
- - - 0	- - - 0	6 0 1 1 0
- - - 0	- - - 0	2 0 0 1 0
- - - 0	- - - 0	0 0 0 0 0
- - - 0	8 1 0 0 0	8 1 0 0 0
- - - 1	9 1 0 0 1	9 1 0 0 1
- - - 1	- - - 1	1 0 0 0 1
- - - 1	- - - 1	3 0 0 1 1
- - - 1	- - - 1	7 0 1 1 1
- - - 1	- - - 1	5 0 1 0 1
(a) Starting point	(b) Add two highest values	(c) Populate rest of table

FIGURE D.14

The process of sub- 2^n consecutive Gray code generation.

Next, we start with our highest two addresses (8 and 9, which equate to 1000_2 and 1001_2 in the case of this example) straddling the “mirror line” [Figure D.14(b)]. And then we populate the rest of the table using smaller values, working out from the center [Figure D.14(c)]. The result is a 10-sequence Gray code using consecutive addresses of 0000_2 to 1001_2 (0 to 9 in decimal) as illustrated in Figure D.15.

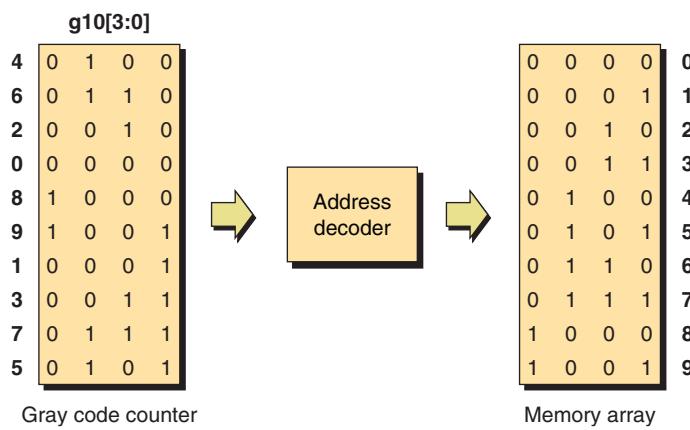


FIGURE D.15

Hurray! No “holes” in the memory array.

I tell you, it amazes me that you have a concept as simple as Gray codes, but every time you think you know it all, something new comes up that blows your socks off. Of course, the test case shown here is just a proof-of-concept. We populated the table by hand, but we don't yet have an algorithm for this. Can we extrapolate this to a generic algorithm that allows us to do the same thing for any non- 2^n Gray code sequence? In the case of a 1128-word memory array, for example, can we algorithmically generate a Gray code sequence that ends up using sequential addresses of 0 to 1127 in the memory array?

Once again, I think we'll leave this as an exercise for the reader (grin).

APPENDIX E

Linear Feedback Shift Registers (LFSRs)

THE OUROBOROS OF THE DIGITAL CONSCIOUSNESS

The *Ouroboros*—a symbol of a serpent or dragon devouring its own tail and thereby forming a circle—has been employed by a variety of ancient cultures around the world to depict eternity or renewal.¹ The equivalent to the Ouroboros in the world of electronics would be the *Linear Feedback Shift Register* (LFSR),² in which the output from a standard shift register is cunningly manipulated and fed back into its input in such a way as to cause the function to endlessly cycle through a sequence of patterns.

407

MANY-TO-ONE IMPLEMENTATIONS

LFSRs are simple to construct and are useful for a wide variety of applications. One of the more common forms of LFSR is formed from a simple shift register with feedback from two or more points, or *taps*, in the register chain (Figure E.1).

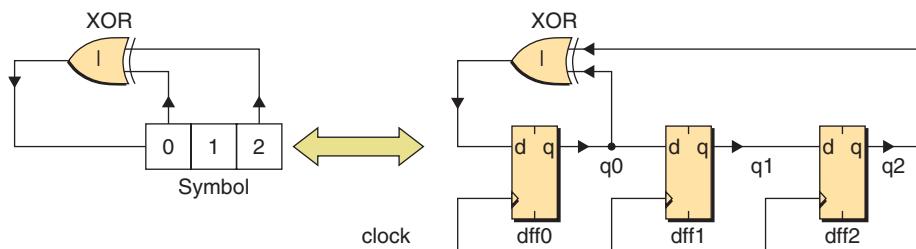


FIGURE E.1

LFSR with XOR feedback path.

¹Not to be confused with the *Amphisbaena*, a serpent in classical mythology having a head at each end and capable of moving in either direction.

²In conversation, LFSR is spelled out as "L-F-S-R."

The taps in this example are at bit 0 and bit 2, and can be referenced as [0,2]. All of the register elements share a common clock input, which is omitted from the symbol for reasons of clarity. The data input to the LFSR is generated by XOR-ing or XNOR-ing the tap bits; the remaining bits function as a standard shift register. The sequence of values generated by an LFSR is determined by its feedback function (XOR versus XNOR) and tap selection (Figure E.2).

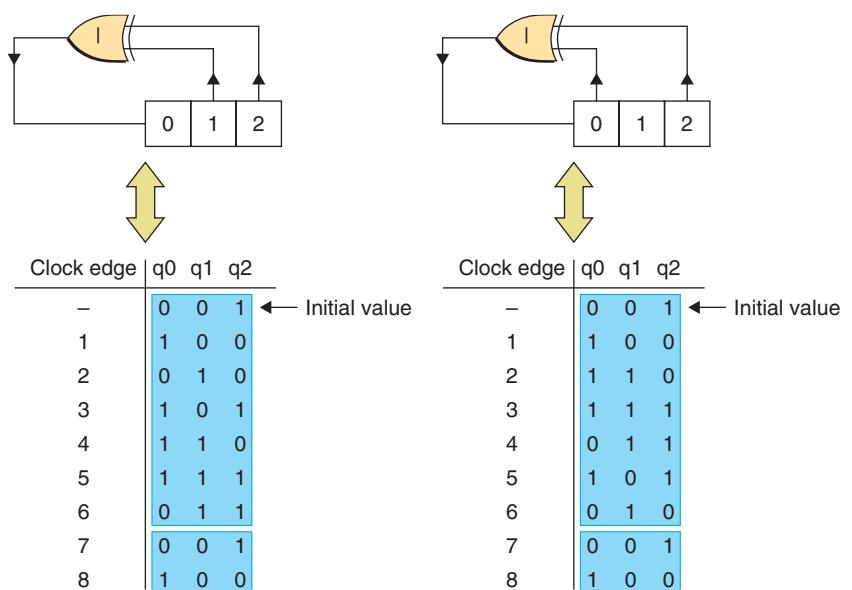
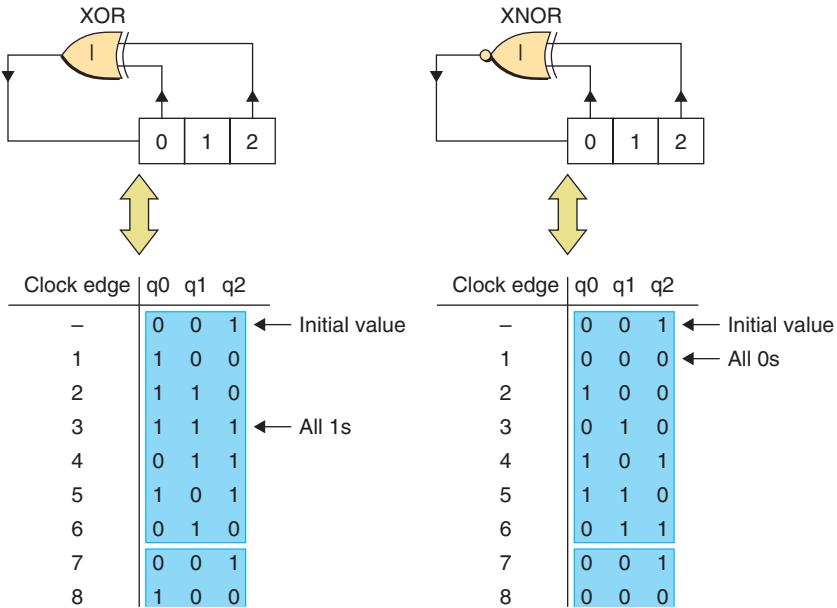


FIGURE E.2

Comparison of alternative tap selections.

Both LFSRs start with the same initial value but, due to the different tap selections, their sequences rapidly diverge as clock pulses are applied. In some cases, an LFSR will end up cycling around a loop comprising a limited number of values. However, both of the LFSRs shown in Figure E.2 are said to be of *maximal length* because they sequence through every possible value (excluding all of the bits being 0) before returning to their initial values.

A binary field with n bits can assume 2^n unique values, but a maximal-length LFSR with n register bits will only sequence through $(2^n - 1)$ values. This is because LFSRs with XOR feedback paths will not sequence through the value where all the bits are logic 0, while their XNOR equivalents will not sequence through the value where all the bits are logic 1 (Figure E.3).

**FIGURE E.3**

Comparison of XOR versus XNOR feedback paths.

MORE TAPS THAN YOU KNOW WHAT TO DO WITH

Each LFSR supports a number of tap combinations that will generate maximal-length sequences. The problem is weeding out the ones that do from the ones that don't, because badly chosen taps can result in the register entering a loop comprising only a limited number of states.

The author created a simple C program to determine the taps for maximal-length LFSRs with 2 to 32 bits. These values are presented for your delectation and delight in Figure E.4 (the "*" annotations indicate sequences whose length is a prime number).

The taps are identical for both XOR-based and XNOR-based LFSRs, although the resulting sequence will, of course, differ. As was previously noted, alternative tap combinations may also yield maximum-length LFSRs, although the resulting sequences will vary. For example, in the case of a 10-bit LFSR, there are two 2-tap combinations that result in a maximal-length sequence: [2,9] and [6,9]. There are also twenty 4-tap combinations, twenty-eight 6-tap combinations, and ten 8-tap combinations that satisfy the maximal-length criteria.

# of bits	Length of loop	Taps
2	3 *	[0,1]
3	7 *	[0,2]
4	15	[0,3]
5	31 *	[1,4]
6	63	[0,5]
7	127 *	[0,6]
8	255	[1,2,3,7]
9	511	[3,8]
10	1,023	[2,9]
11	2,047	[1,10]
12	4,095	[0,3,5,11]
13	8,191 *	[0,2,3,12]
14	16,383	[0,2,4,13]
15	32,767	[0,14]
16	65,535	[1,2,4,15]
17	131,071 *	[2,16]
18	262,143	[6,17]
19	524,287 *	[0,1,4,18]
20	1,048,575	[2,19]
21	2,097,151	[1,20]
22	4,194,303	[0,21]
23	8,388,607	[4,22]
24	16,777,215	[0,2,3,23]
25	33,554,431	[2,24]
26	67,108,863	[0,1,5,25]
27	134,217,727	[0,1,4,26]
28	268,435,455	[2,27]
29	536,870,911	[1,28]
30	1,073,741,823	[0,3,5,29]
31	2,147,483,647 *	[2,30]
32	4,294,967,295	[1,5,6,31]

FIGURE E.4

Example taps for maximal length LFSRs with 2 to 32 bits.

of values will differ between them. But the main point is that using the one-to-many style means that there is never more than one level of combinational logic in the feedback path, irrespective of the number of taps being employed.

ONE-TO-MANY IMPLEMENTATIONS

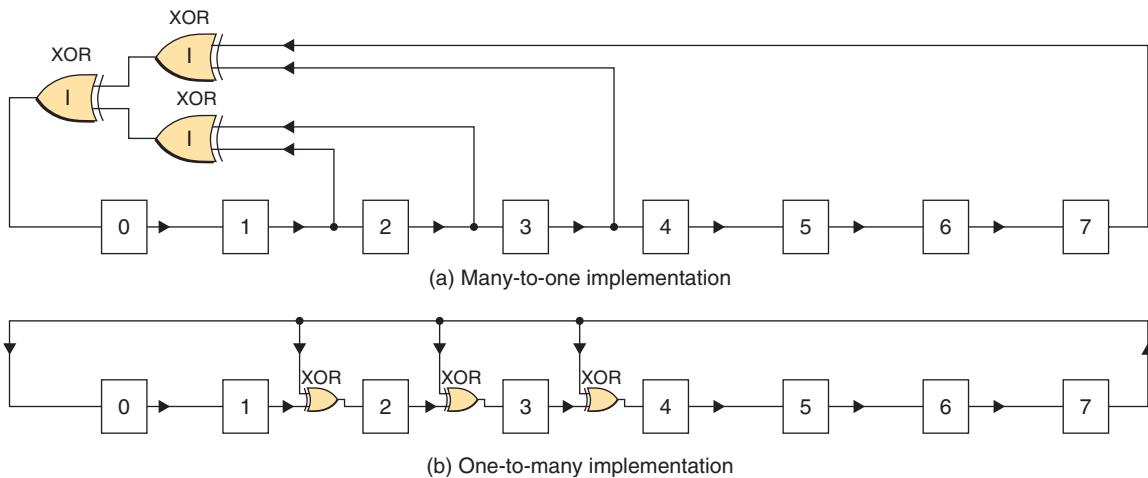
Consider the case of an 8-bit LFSR, for which the minimum number of taps that will generate a maximal-length sequence is four. In the real world, XOR gates only have two inputs, so a four-input XOR function has to be created using three XOR gates arranged as two levels of logic. Even in those cases where an LFSR does support a minimum of two taps, for one reason or another you may actually wish to use a greater number of taps, such as eight (which would result in three levels of XOR logic).

The problem is that increasing the levels of logic in the combinational feedback path can negatively impact the maximum clocking frequency of the function. One solution is to transpose the *many-to-one implementations* discussed above into their *one-to-many counterparts* (Figure E.5).

The traditional many-to-one implementation for the eight-bit LFSR has taps at [7,3,2,1]. To convert this into its one-to-many counterpart, the most-significant tap (which is always the most significant bit) is fed back directly into the least significant bit, and is also individually XOR-ed with the other original taps (bits [3,2,1] in this example). Note that although both styles result in maximal-length LFSRs, the actual sequences

SEEDING AN LFSR

One quirk with an XOR-based LFSR is that, if it happens to find itself in the all 0s value, it will happily continue to shift all 0s indefinitely (similarly for an XNOR-based LFSR and the all 1s value). This is of particular concern when power is first applied to the circuit. Each register bit can randomly initialize, containing either a logic 0 or a logic 1, and the LFSR can therefore “wake up” containing its “forbidden” value. For this reason, it is necessary to initialize an LFSR with a *seed value*.

**FIGURE E.5**

Many-to-one versus one-to-many LFSR implementations.

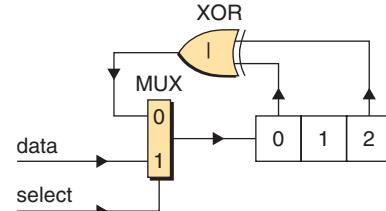
One method for loading a seed value is to use registers with reset and/or set inputs. A single control signal can be connected to the reset inputs on some of the registers and the set inputs on others. When this control signal is placed in its active state, the LFSR will load with a hard-wired seed value. In certain applications, however, it is desirable to be able to vary the seed value. One technique for achieving this is to include a multiplexer at the input to the LFSR (Figure E.6).

When the multiplexer's data input is selected, the device functions as a standard shift register, and any desired seed value may be loaded. After loading the seed value, the feedback path is selected and the device returns to its LFSR mode of operation.

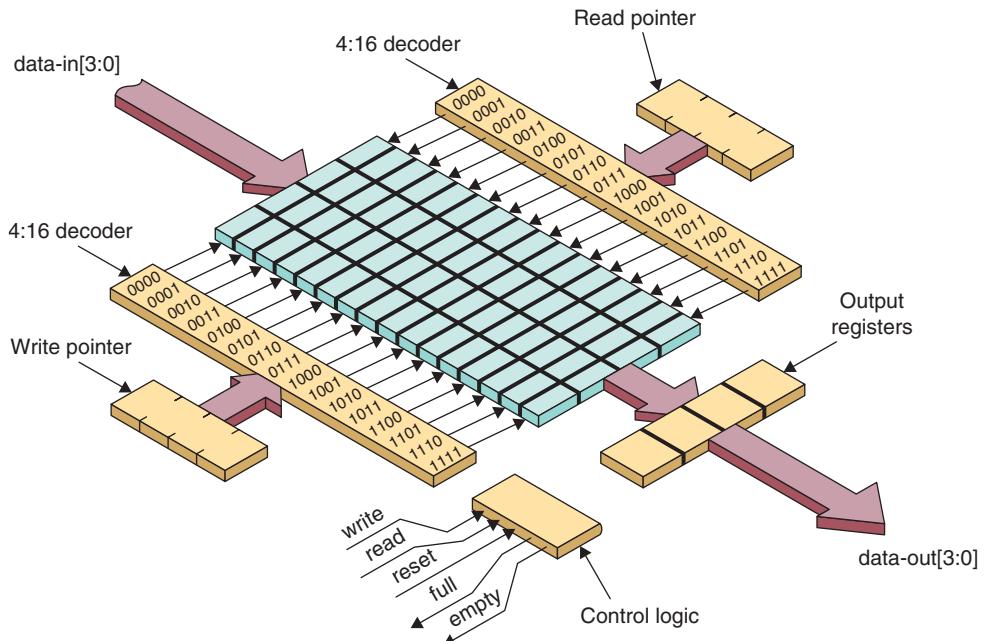
FIFO APPLICATIONS

The fact that an LFSR generates an unusual sequence of values is irrelevant in many applications. Consider a 4-bit \times 16-word *First-In First-Out* (FIFO) memory as illustrated in Figure E.7.

A brief summary of the FIFO's operation is as follows. The write and read pointers are essentially 4-bit registers whose outputs are processed by 4:16 decoders to select one of the sixteen words in the memory array. The reset input is used to initialize the device, primarily by clearing the write and read pointers such that they both point to the same memory word. The initialization also causes

**FIGURE E.6**

Circuit for loading different seed values.

**FIGURE E.7**

First-In First-Out (FIFO) memory.

the empty output to be placed in its active state and the full output to be placed in its inactive state.³

The write and read pointers chase each other around the memory array in an endless loop. An active edge on the write input causes any data on the data-in[3:0] bus to be written into the word pointed to by the write pointer. The empty output is placed in its inactive state (because the device is no longer empty) and the write pointer is incremented to point to the next empty word.

Data can be written into the FIFO until all the words in the array contain values. When the write pointer catches up to the read pointer, the full output is placed in its active state (indicating that the device is full) and no more data can be written into the device.

An active edge on the read input causes the data in the word pointed to by the read pointer to be copied into the output register, from whence it appears on

³More-sophisticated versions may have additional signals, such as nearly-full and nearly-empty.

the `data-out[3:0]` signals. The full output is placed in its inactive state and the read pointer is incremented to point to the next word containing data.⁴

Data can be read out of the FIFO until the array is empty. When the read pointer catches up to the write pointer, the `empty` output is placed in its active state, and no more data can be read out of the device.

The write and read pointers for a 16-word FIFO are often implemented using 4-bit binary counters.⁵ However, a moment's reflection reveals that there is no intrinsic advantage to a binary sequence for this particular application, and the sequence generated by a 4-bit LFSR would serve equally well. In fact, the two functions operate in a very similar manner, as is illustrated by their block diagrams (Figure E.8).

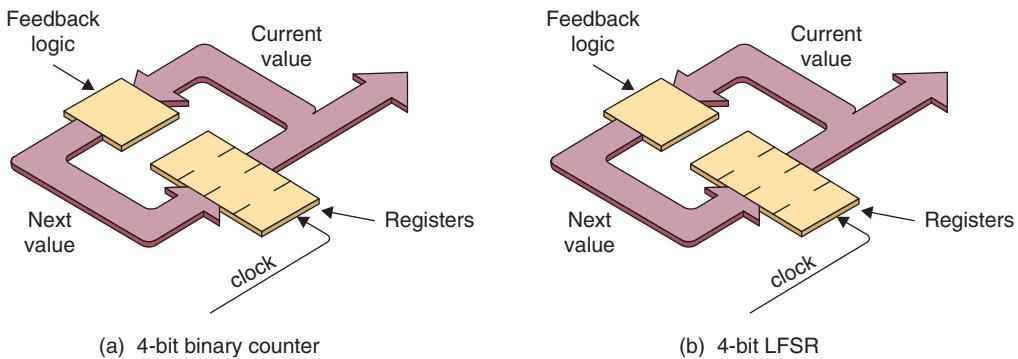


FIGURE E.8

Block diagrams of a binary counter and an LFSR.

However, the combinational logic for the 4-bit LFSR consists of a single, two-input XOR gate, while the combinational logic for the 4-bit binary counter requires a number of AND and OR gates. This means that the LFSR requires fewer tracks (wires) and is more efficient in terms of silicon real estate. Additionally, the LFSR's feedback only passes through a single level of logic, while the binary counter's feedback passes through multiple levels of logic. This means that the new data value is available sooner for the LFSR, which can therefore be clocked at a higher frequency. These differentiations become even

⁴These discussions assume the use of write-and-increment and read-and-increment techniques, but we should note that some FIFOs employ an increment-and-write and increment-and-read approach.

⁵The implementation of a 4-bit binary counter was discussed in *Chapter 11: Slightly More Complex Functions*.

more pronounced for FIFOs with more words requiring pointers with more bits. Thus, LFSR's may be a very attractive choice for the discerning designer of FIFOs.

MODIFYING LFSRS TO SEQUENCE 2^n VALUES

One downside to the 4-bit LFSRs in the FIFO scenario above is that they will sequence through only 15 values ($2^4 - 1$), as compared to the binary counter's sequence of 16 values (2^4). Designers may not regard this to be a problem, especially in the case of larger FIFOs. However, if it is required that an LFSR sequence through every possible value, there is a simple solution (Figure E.9).

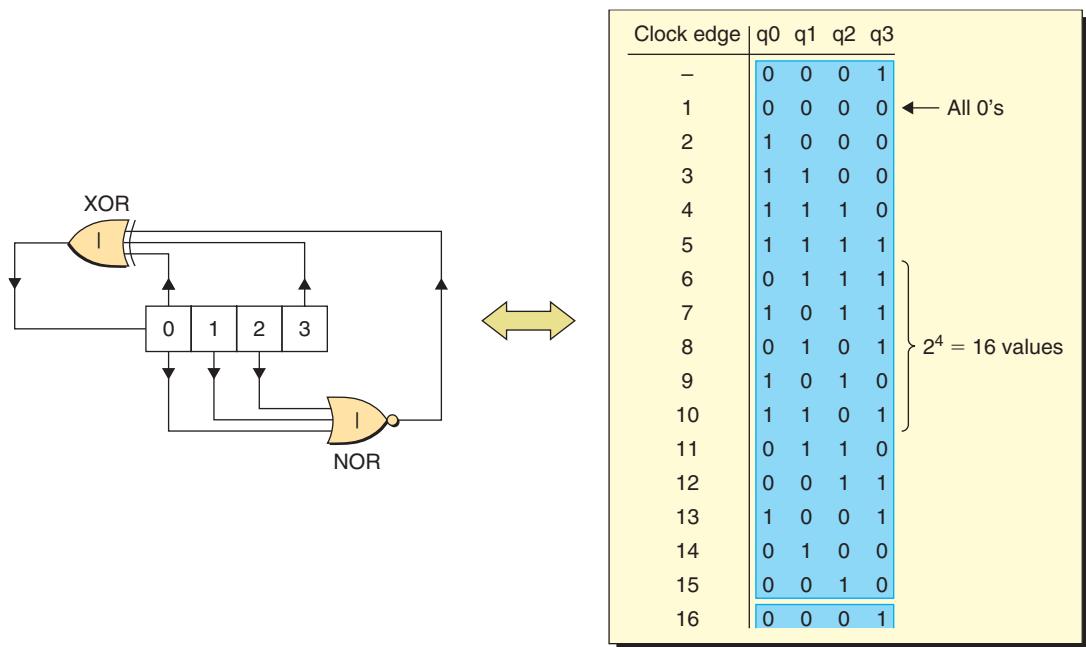


FIGURE E.9

LFSR modified to sequence through 2^n values.

For the value for which all of the bits are logic 0 to appear, the preceding value must have comprised a logic 1 in the *Most-Significant Bit* (MSB)⁶ and logic 0s in the remaining bit positions. In an *unmodified LFSR*, the next clock would result in a logic 1 in the *Least-Significant Bit* (LSB) and logic 0s in the remaining bit

⁶As is often the case with any form of shift register, the MSB in these examples is taken to be on the right-hand side of the register, and the LSB is taken to be on the left-hand side (this is opposite to the way we usually do things).

positions. In the *modified LFSR*, however, the output from the NOR is a logic 0 for every case but two: the value preceding the one where all the bits are 0, and the value where all the bits are 0. These two values force the NOR's output to a logic 1, which inverts the usual output from the XOR. This in turn causes the sequence to first enter the all-0s value and then resume its normal course. (In the case of LFSRs with XNOR feedback paths, the NOR can be replaced with an AND, which causes the sequence to cycle through the value where all of the bits are logic 1.)

ACCESSING AN LFSR'S PREVIOUS VALUE

In some applications, it is required to make use of a register's previous value. In certain FIFO implementations, for example, the "full" condition is detected when the write pointer is pointing to the location preceding the location pointed to by the read pointer.⁷ This implies that a comparator must be used to compare the *current value* in the write pointer with the *previous value* in the read pointer. Similarly, the "empty" condition may be detected when the read pointer is pointing to the location preceding the location pointed to by the write pointer. This implies that a second comparator must be used to compare the *current value* in the read pointer with the *previous value* in the write pointer.

In the case of binary counters, there are two techniques by which the previous value in the sequence may be accessed. The first requires the provision of an additional set of so-called *shadow registers*. Every time the counter is incremented, its current contents are first copied into the shadow registers. Alternatively, a block of combinational logic can be used to decode the previous value from the current value. Unfortunately, both of these techniques involve a substantial overhead in terms of additional logic. By comparison, LFSRs inherently remember their previous value. All that is required is the addition of a single register bit appended to the MSB (Figure E.10).

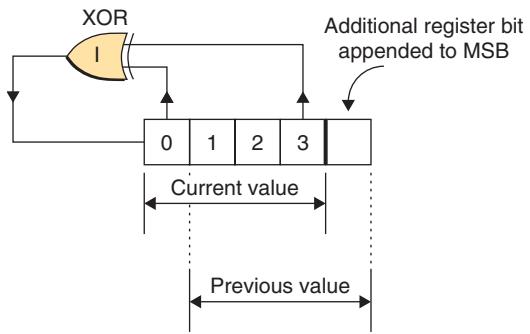
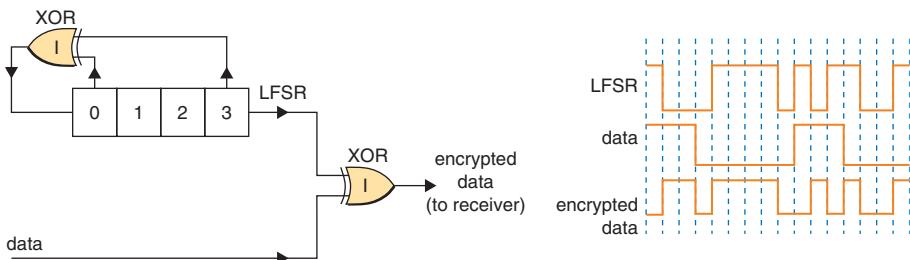


FIGURE E.10
Accessing an LFSR's previous value

ENCRYPTION AND DECRYPTION APPLICATIONS

The unusual sequence of values generated by an LFSR can be gainfully employed in the encryption (scrambling) and decryption (unscrambling) of data. For example, a stream of data bits can be encrypted by XOR-ing them with the output from an LFSR (Figure E.11).

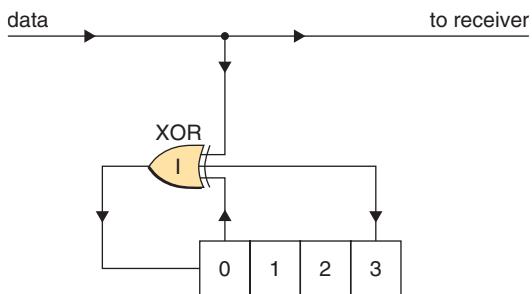
⁷Try saying that quickly!

**FIGURE E.11**

Data encryption using an LFSR.

The stream of encrypted data bits seen by a receiver can be decrypted by XOR-ing them with the output of an identical LFSR.⁸

CYCLIC REDUNDANCY CHECK (CRC) APPLICATIONS

**FIGURE E.12**

Cyclic Redundancy Check (CRC) calculation.

A traditional application for LFSRs is in *Cyclic Redundancy Check* (CRC) calculations, which can be used to detect errors in data communications. In this case, the stream of data bits being transmitted is used to modify the values fed back into an LFSR (Figure E.12).

The final CRC value stored in the LFSR is known as a *checksum*, and is dependent on every bit in the data stream. After all of the data bits have

been transmitted, the transmitter sends its checksum value to the receiver. The receiver contains an identical CRC calculator and generates its own checksum value from the incoming data. Once all of the data bits have arrived, the receiver compares its internally generated checksum value with the checksum sent by the transmitter in order to determine whether any corruption occurred during the course of the transmission. This form of error detection is very efficient in terms of the small number of bits that have to be transmitted in addition to the data.

In the real world, a 4-bit CRC calculator would not be considered to provide sufficient confidence in the integrity of the transmitted data. This is due to the

⁸The rudimentary example presented here is obviously a very trivial form of encryption that's not very secure, but it's "cheap-and-cheerful" and may be useful in certain applications.

fact that a 4-bit LFSR can only represent 16 unique values, which means that there is a significant probability that multiple errors in the data stream could result in the two checksum values being identical. As the number of bits in a CRC calculator increases, however, the probability that multiple errors will cause identical checksum values approaches zero. For this reason, CRC calculators typically use a minimum of 16-bits providing 65,536 unique values.

There is a variety of standard communications protocols, each of which specifies the number of bits employed in their CRC calculations and the taps to be used. The taps are selected such that an error in a single data bit will cause the maximum possible disruption to the resulting checksum value. Thus, in addition to being referred to as *maximal-length*, these LFSRs may also be qualified as *maximal-displacement*.

DATA COMPRESSION APPLICATIONS

The CRC calculators discussed in the previous topic can also be used in a data compression role. One such application is found in the circuit board test strategy known as *functional test*. In this case, the board is plugged into a functional tester by means of its edge connector. The tester applies a pattern of signals to the board's inputs, allows sufficient time for any effects to propagate around the board, and then compares the actual values seen on the outputs with a set of expected values stored in the system. This process is repeated for a series of input patterns, which may number in the tens or hundreds of thousands (Figure E.13).

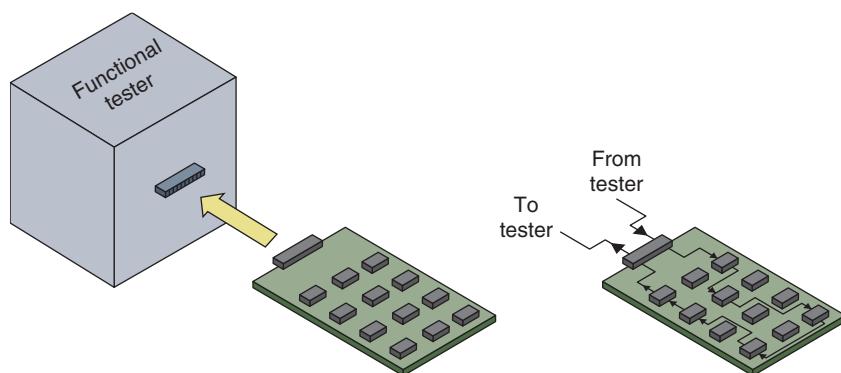
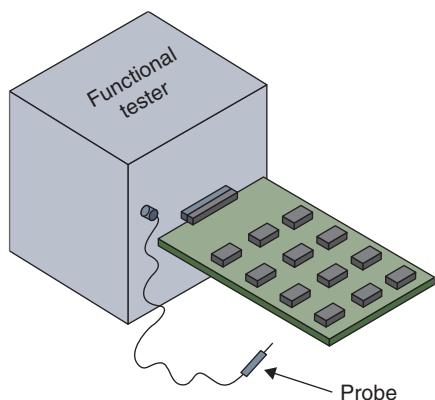


FIGURE E.13

Simplified representation of a functional board test scenario.

**FIGURE E.14**

Guided probe analysis.

The illustration above is simplified for reasons of clarity. In practice, the edge connector may contain hundreds of pins while the board may contain thousands of components and tracks (wires). If the board fails the preliminary tests, a more sophisticated form of analysis known as *guided probe* may be employed to identify the cause of the failure (Figure E.14).

The idea here is that the functional tester instructs the operator to place the probe at a particular location on the board, and then the entire sequence of test patterns is rerun. The tester compares the actual sequence of values

seen by the probe with a sequence of expected values that are stored in the system. This process (placing the probe and running the tests) is repeated until the tester has isolated the faulty component or track.

A major consideration when supporting a guided probe strategy is the amount of expected data that must be stored. Consider a test sequence comprising 10,000 patterns, driving a board containing 10,000 tracks. If the data were not compressed, the system would have to store 10,000 bits of expected data per track, which amounts to 100,000,000 bits of data for the board. Additionally, for each application of the guided probe, the tester would have to compare the 10,000 data bits observed by the probe with the 10,000 bits of expected data stored in the system. Thus, using data in an uncompressed form is an expensive option in terms of storage and processing requirements.

One solution to these problems is to employ LFSR-based CRC calculators. The sequence of expected values for each track can be passed through a 16-bit CRC calculator implemented in software. Similarly, the sequence of actual values seen by the guided probe can be passed through an identical CRC calculator implemented in hardware. In this case, the calculated checksum values are also known as *signatures*, and a guided probe process based on this technique is known as *signature analysis*. Irrespective of the number of test patterns used, the system has to store only two bytes of data for each track. Additionally, for each application of the guided probe, the tester has to compare only the two bytes of data gathered by the probe with two bytes of expected data stored in the system. The end result is that compressing the data results in storage requirements that are orders of magnitude smaller—and comparison times that are orders of magnitude faster—than the uncompressed data approach.

BUILT-IN SELF-TEST (BIST) APPLICATIONS

One test strategy that may be employed in complex integrated circuits is that of *Built-in Self-Test* (BIST).⁹ Devices using BIST contain special test generation and result gathering circuits (Figure E.15).

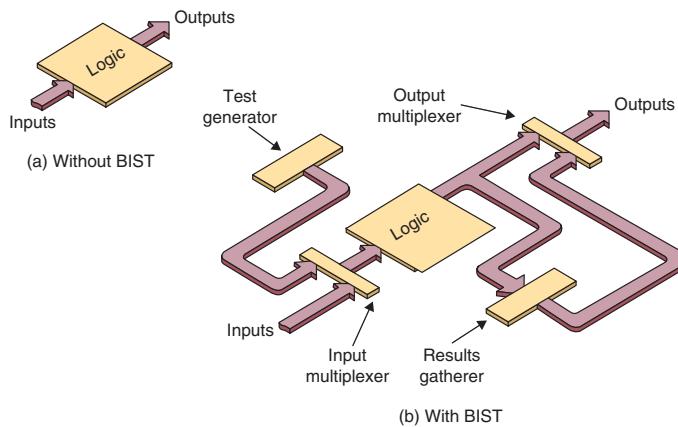


FIGURE E.15

Built-In Self-Test (BIST).

A multiplexer is used to select between the standard inputs and those from a *test generator*. A second multiplexer selects between the standard outputs and those from a *results gatherer*. The point is that both the test generator and results gatherer can be implemented using LFSRs (Figure E.16).¹⁰

The LFSR forming the test generator is used to create a sequence of test patterns, while the LFSR forming the results gatherer is used to capture the results. The results-gathering LFSR features modifications that allow it to accept parallel data. (Note that the two LFSRs are not obliged to contain the same number of bits, because the number of inputs to the logic being tested may be different from the number of outputs from the logic.)

Once the self-test has been run, the contents of the results gathering LFSR can be compared to a known-good value to determine if the core logic is functioning as expected.

⁹There are several specialized versions of BIST; for example, *Logic Built-in Self-Test* (LBIST) and *Memory Built-in Self-Test* (MBIST). We're focusing on LBIST in these discussions.

¹⁰The standard inputs and outputs (along with their multiplexers) have been omitted from Figure E.16 in order to keep things simple.

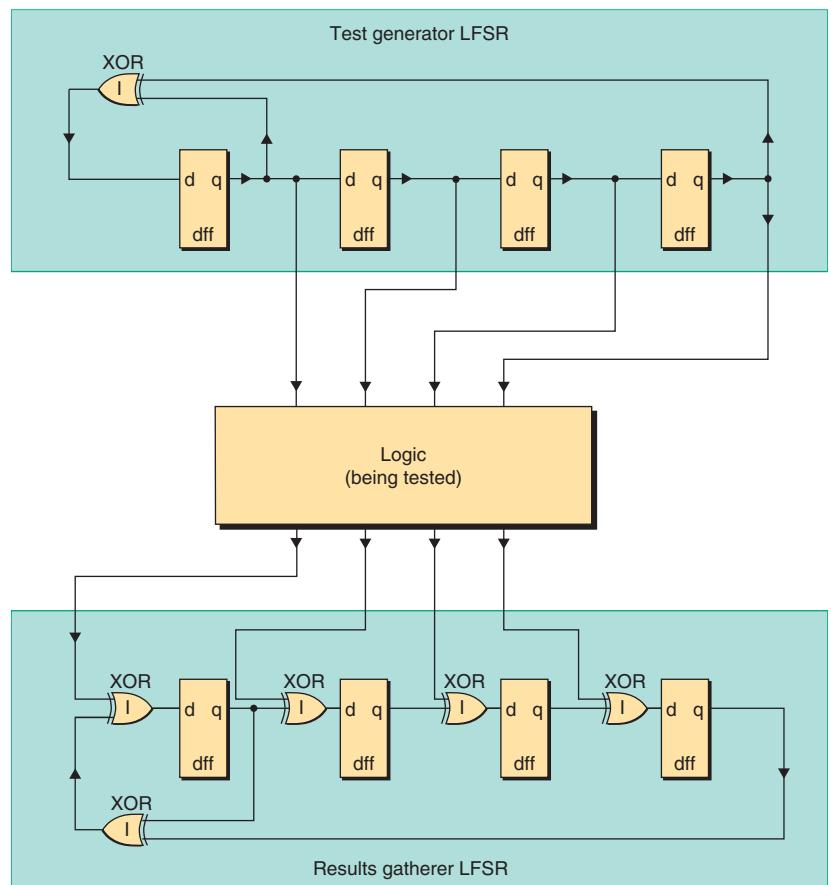


FIGURE E.16
BIST implemented using LFSRs.

PSEUDO-RANDOM NUMBER APPLICATIONS

Many computer programs rely on an element of randomness. Computer games such as “Space Invaders” employ random events to increase the player’s enjoyment. Graphics programs may exploit random numbers to generate intricate patterns. All forms of computer simulation may utilize random numbers to more accurately represent the real world. For example, digital simulations¹¹

¹¹Digital simulation is based on a program called a *logic simulator*, which is used to build a virtual representation of an electronic design in the computer’s memory. The simulator then applies stimulus to the design’s virtual inputs, simulates the effect of these signals as they propagate through the design, and checks the responses at the design’s virtual outputs.

may benefit from the portrayal of random stimulus such as external interrupts. Random stimulus can result in more realistic design verification, which can uncover problems that may not be revealed by more structured tests.

Random number generators can be constructed in both hardware and software. The majority of these generators are not truly random, but they give the appearance of being random and are therefore said to be *pseudo-random*. In fact, pseudo-random numbers have an advantage over truly random numbers, because the majority of computer applications typically require repeatability. For example, a designer repeating a digital simulation would expect to receive answers identical to those from the previous run. However, designers also need the ability to modify the seed value of the pseudo-random number generator so as to spawn different sequences of values as required.

There is a variety of methods available for generating pseudo-random numbers. A popular cheap-and-cheerful technique uses the remainder from a division operation as the next value in a pseudo-random sequence. For example, a C function that returns a pseudo-random number in the range 0 to 32767 could be written as follows:¹²

```
int rand ()
{
    next = next * 1103515245 + 12345;
    return (unsigned int) (next/65535) % 32768;
}
```

The variable `next` would be declared as global and initialized with some seed value. Also, an additional function would be provided to allow the programmer to load `next` with a new seed value if required. Every time our `rand` function is accessed it returns a new pseudo-random value. Unfortunately, pseudo-random implementations based on division do not always produce a “white spectrum.” The series of generated values may be composed of a collection of subseries, in which case the results will not be of maximal length. By comparison, the sequence of values generated by a software implementation of a maximal-length LFSR provides a reasonably good pseudo-random source, but is somewhat more expensive in terms of processing requirements.

¹²This example came from *The C Programming Language* (2nd ed.), by Brian W. Kernighan and Dennis M. Ritchie.

LAST BUT NOT LEAST

LFSRs are simple to construct and are useful for a wide variety of applications, but be warned that choosing the optimal polynomial (which ultimately boils down to selecting the optimal tap points) for a particular application is a task that is usually reserved for a master of the mystic arts. The math behind this can be hairy enough to make a grown man break down and cry (and don't even get me started on the subject of *cyclotomic polynomials*,¹³ which are key to the tap-selection process).

¹³Mainly because I don't have the faintest clue as to what a cyclotomic polynomial is!

APPENDIX F

Pass-Transistor Logic

423

“WOULD YOU PASS THE LOGIC, PLEASE?”

The term *pass-transistor logic* refers to techniques for connecting MOSFET transistors such that “data” signals pass between their source and drain terminals. These techniques minimize the number of transistors required to implement a function, but they are not recommended for the novice or the unwary because strange and unexpected effects can ensue. Pass-transistor logic is typically employed by designers of full-custom integrated circuits or ASIC cell libraries. The following examples introduce the concepts of pass-transistor logic, but they are not intended to indicate recommended design practices.

In the case of the AND (Figure F.1), the resistance of R_1 is assumed to be sufficiently high that its effect on output y is that of a very weak logic 0. When input b is presented with a logic 0, the NMOS transistor Tr_1 is turned OFF and y is “pulled down” to logic 0 by resistor R_1 . When input b is set to logic 1, Tr_1 is turned ON and output y is connected to input a . In this case, a logic 0 on input a leaves y at logic 0, but a logic 1 on input a will overdrive the effect of resistor R_1 and force y to a logic 1.

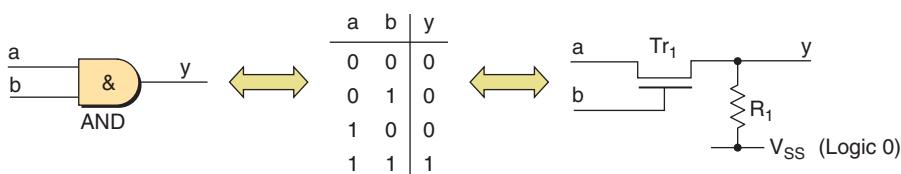


FIGURE F.1

Pass-transistor implementation of an AND gate.

In the case of the OR (Figure F.2), the resistance of R_1 is assumed to be sufficiently high that its effect on output y is that of a very weak logic 1. A logic 1 applied to input b turns the PMOS transistor Tr_1 OFF and output y is “pulled up” to logic 1

by resistor R_1 . When input b is presented to logic 0, Tr_1 is turned ON and output y is connected to input a. In this case, a logic 1 on input a leaves y at logic 1, but a logic 0 on input a will overdrive the effect of resistor R_1 and force y to a logic 0.

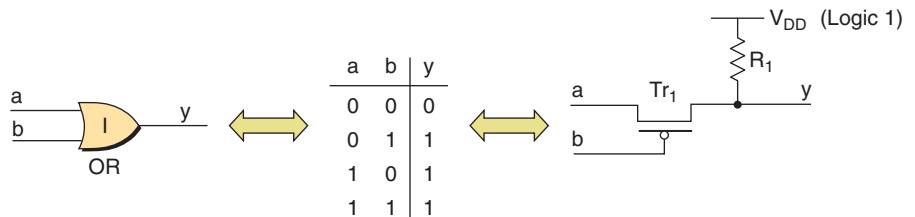


FIGURE F.2

Pass-transistor implementation of an OR gate.

In the case of the XOR (Figure F.3), logic 1s applied to inputs a and b turn Tr_1 and Tr_2 OFF, respectively, leaving output y to be “pulled down” to logic 0 by resistor R_1 . When input b is presented with a logic 0, Tr_1 is turned ON and output y is connected to input a. In this case, a logic 0 on input a leaves y at logic 0, but a logic 1 on input a will overdrive the effect of resistor R_1 and force y to a logic 1. Similarly, when input a is presented with a logic 0, Tr_2 is turned ON and output y is connected to input b. In this case, a logic 0 on input b leaves y at logic 0, but a logic 1 on b will overdrive the effect of resistor R_1 and force y to a logic 1 (phew!).

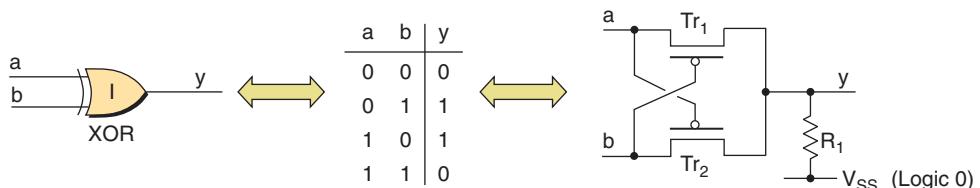
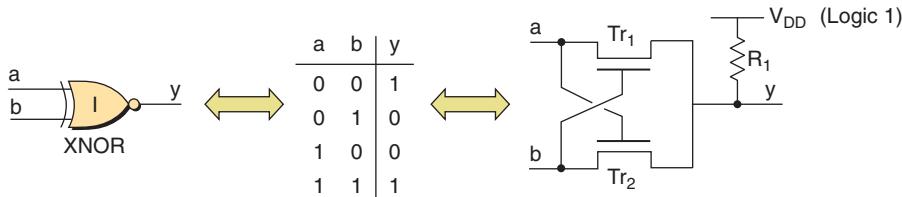


FIGURE F.3

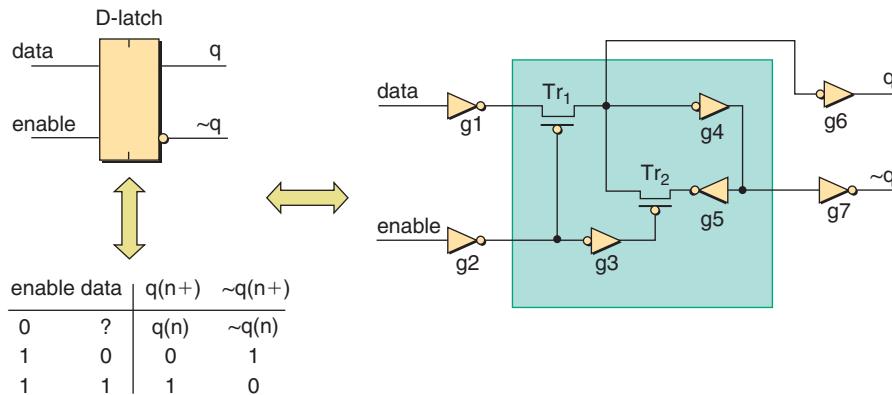
Pass-transistor implementation of an XOR gate.

The case of the XNOR (Figure F.4) is simply the inverse of that for the XOR. Logic 0s applied to inputs a and b turn transistors Tr_1 and Tr_2 OFF, leaving output y to be “pulled up” to logic 1 by resistor R_1 . When input b is set to logic 1, Tr_1 is turned ON and output y is connected to input a. In this case, a logic 1 on input a leaves y at logic 1, but a logic 0 on input a will overdrive the effect of resistor R_1 and force y to a logic 0. Similarly, when input a is set to logic 1, Tr_2 is turned ON and output y is connected to input b. In this case, a logic 1 on input b leaves y at logic 1, but a logic 0 on b will overdrive the effect of resistor R_1 and force y to a logic 0.

**FIGURE F.4**

Pass-transistor implementation of an XNOR gate.

Before examining the pass-transistor implementation of a D-type latch with an active-high enable shown in Figure F.5, we should briefly review our definition of these functions. First, when the enable input is in its active state (a logic 1), the value on the data input is passed through the device to appear on the q and $\sim q$ outputs.¹ Second, if the data input changes while the enable input remains active, the outputs will respond to reflect the new value. Third, when the enable input is driven to its inactive state, the outputs remember their previous values and no longer respond to any changes on the data input.

**FIGURE F.5**

Pass-transistor implementation of a D-type latch.

Now onto the pass-transistor implementation itself.² When the enable input is in its active state, the output from gate g2 turns transistor Tr₁ ON, and the output from g3 turns transistor Tr₂ OFF. Thus, the value on the data input passes through g1, Tr₁, and g6 to appear on the q output, and through g1, Tr₁, g4, and g7 to appear at the $\sim q$ output.³ Any changes on the data input will be reflected

¹As usual, the data appears in inverted form on the $\sim q$ (complementary) output.

²Observe the use of assertion-level logic symbols for gates g3, g4, and g6 in Figure F.5. These symbols were introduced in Appendix A: *Assertion-Level Logic*.

³Observe the use of the tilde “~” character to indicate the complementary output $\sim q$. The use of tildes was discussed in Appendix A: *Assertion-Level Logic*.

on the q and $\sim q$ outputs after the delays associated with the gates and transistors have been satisfied.

Now, when the enable input is driven to its inactive state, the output from g_2 turns T_{r_1} OFF and the output from g_3 turns T_{r_2} ON. Disabling T_{r_1} blocks the path from the data input, while enabling T_{r_2} completes the self-sustaining loop formed by g_4 and g_5 . It is this feedback loop that acts as the “memory” for the device.

Compare this pass-transistor implementation with its standard counterpart introduced in *Chapter 11: Slightly More Complex Functions*. The standard implementation required one NOT, two ANDs, and two NORs, totaling twenty-two transistors. By comparison, the pass-transistor implementation requires two discrete transistors and seven NOTs, totaling only sixteen transistors. In fact, the heart of the pass transistor implementation comprising T_{r_1} , T_{r_2} , g_3 , g_4 , and g_5 requires only eight transistors. The remaining gates (g_1 , g_2 , g_6 , and g_7) are used only to buffer the latch’s input and outputs from the outside world.⁴

⁴NOT gates are used in preference to BUFs for this role because NOTs require fewer transistors and are faster (these considerations were introduced in *Chapter 6: Using Transistors to Build Logic Gates*).

APPENDIX G

More on Semiconductors

P-N JUNCTIONS, DEPLETION ZONES, AND DIODES

Do you recall way back in the mists of time [known as *Chapter 4: Semiconductors (Diodes and Transistors)*] when we first introduced the concept of a semiconductor diode? This was formed by doping a piece of silicon such that one part was P-type and the other was N-type (Figure G.1). As we discussed, the resulting *p-n junction* conducts electricity in only one direction: in the other direction it behaves like an OPEN (OFF) switch.

427

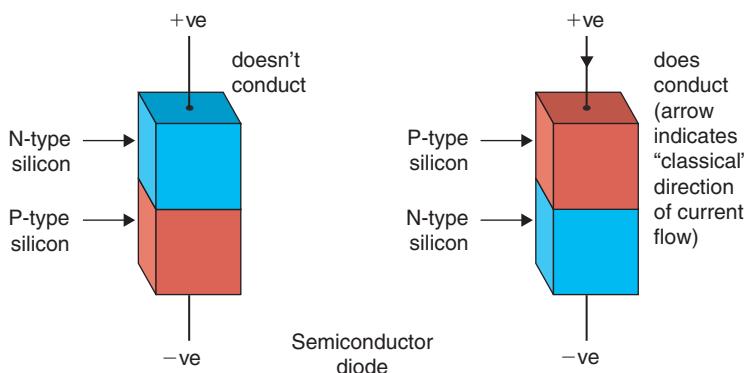


FIGURE G-1

Creating a diode from P-type and N-type silicon.

However, we sort of “glossed over” the way in which this actually works, so let’s consider this in a little more detail. Let’s start off with a piece of pure silicon [Figure G.2(a)]. First, we’ll dope one-half of the silicon with phosphorus to create *N-type silicon* [Figure G.2(b)]. Since phosphorous atoms have five electrons in their outermost electron shells, the site (location) of a phosphorous atom in the silicon crystal matrix will donate an electron with relative ease, and we can visualize this site as being “sort-of” negative.

Next, we'll dope the other half of the silicon with boron to create *P-type* silicon [Figure G.2(c)]. Since boron atoms have three electrons in their outermost electron shell, they can only make bonds with three of the silicon atoms surrounding them. This leaves the fourth silicon atom unsaturated and eager to fill its outermost electron shell. Thus, the site occupied by a boron atom in the silicon crystal will accept a free electron with relative ease, and we can visualize this site (which is called a *hole*) as being “sort-of” positive.

So, the N-type silicon has an excess of free electrons and the P-type silicon has an excess of holes. Electrons and holes are both charge carriers, which means that both the N-type and P-type silicon can conduct. However, something interesting occurs at the p-n junction, because the electrons and holes cancel each other out, thereby depleting this area of any free charge carriers, leaving none to carry a current. The resulting insulating area is known as the *depletion region* or the *depletion zone* [Figure G.2(d)].¹

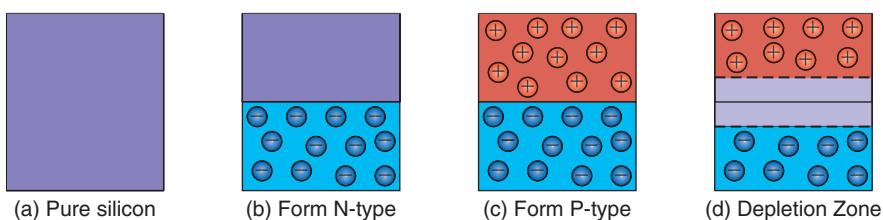
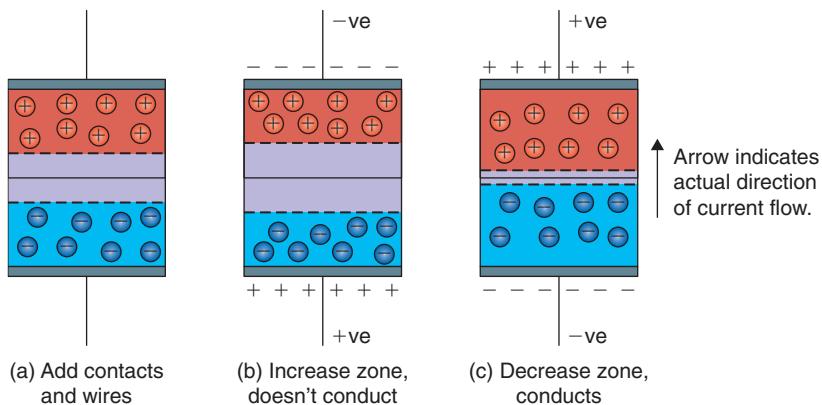


FIGURE G.2
Creating a p-n junction.

Now, let's assume that we add metal contacts and wires to the silicon on either end of our diode [Figure G.3(a)]. Suppose we apply a positive (+ve) potential to the wire connected to the N-type silicon and a negative (-ve) potential to the wire connected to the P-type silicon [Figure G.3(b)]. In this case, the positive potential connected to the N-type silicon attracts the electrons in the N-type silicon and repels the holes in the P-type silicon. Similarly, the negative potential connected to the P-type silicon attracts the holes in the P-type silicon and repels the electrons in the N-type silicon. The result is to increase the size of the insulating depletion zone, thereby preventing the diode from conducting.

By comparison, consider what happens if we apply a negative (-ve) potential to the wire connected to the N-type silicon and a positive (+ve) potential to the wire connected to the P-type silicon [Figure G.3(c)]. In this case, the negative potential connected to the N-type silicon repels the electrons in the N-type silicon and attracts the holes in the P-type silicon. Similarly, the positive potential connected to the P-type silicon repels the holes in the P-type silicon and

¹This area may also be referred to as the *junction region* or the *space charge* region.

**FIGURE G.3**

Applying electric potentials to our p-n junction.

attracts the electrons in the N-type silicon. The result is to reduce the size of the depletion zone until it breaks down (in a nondestructive way), thereby allowing the diode to conduct.

We can also use p-n junctions to create more complex components called *transistors*. As we discussed in *Chapter 4: Semiconductors (Diodes and Transistors)*, the first *Bipolar Junction Transistor (BJT)* was constructed in 1947. For the purposes of our discussions here, however, we are going to focus on *Field Effect Transistors (FETs)*.

JUNCTION FETS (JFETS) AND MESFETS

Junction Field-Effect Transistors (JFETs or JUGFETs) were first analyzed by William Shockley at Bell Labs in 1952. The following year, the first working device was realized by two members of Shockley's team: G. C. Dacey and Ian Munro Ross.

Junction FETs are so-named because they are formed by the junction of P-type and N-type silicon, as illustrated in Figure G.4. We can think of the *drain* and *source* as forming "data" terminals and the *gate* acting as the "control" terminal. Observe that, when no signal is being applied to the gate terminal, the default depletion region [Figure G.4(a)] still leaves a channel of N-type silicon between the source and the drain, thereby allowing current to flow.² This means that these devices are ON by default and we have to apply a signal to the gate terminal in order to turn them OFF.

²Actually, since current is defined as "the flow of electrons," purists would say that phrases like "... allowing current to flow ..." is redundant and/or incorrect, because they would read this as "... allowing the flow of electrons to flow ..." Personally, I really don't care, and it's my book, so there!

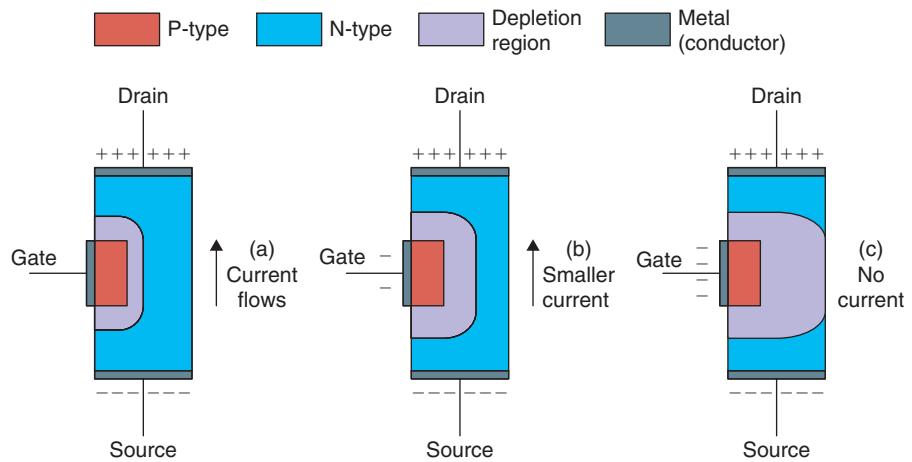


FIGURE G.4
N-channel junction FET
(JFET).

If we apply a small negative potential to the gate terminal [Figure G.4(b)], the electrons on the gate attract the positive holes in the P-type silicon and repel the negative electrons in the N-type silicon. Much like pinching a garden hosepipe to reduce the flow of water, this reduces the size of the conducting channel between the source and the drain terminals. In turn, this increases the resistance of the channel and reduces the flow of current between the source and the drain terminals.

If we keep on increasing the negative potential on the gate terminal, at some stage the depletion zone will completely block the channel as illustrated in Figure G.4(c).

Observe that the transistor shown in Figure G.4 is an n-channel device. We can also create an equivalent p-channel component by swapping the P-type and N-type diffusion regions.

JFETs have good linearity and low noise, and they are used almost entirely for processing analog signals. Typical applications include low-level audio amplification and Radio Frequency (RF) circuits such as RF mixers. JFETs also have a very high input impedance, which makes them suitable for applications like test equipment because they have minimal disturbance on the signals being measured.

We should also note that the MESFETs (*Metal-Epitaxial Semiconductor Field-Effect Transistors*) that were briefly mentioned in *Chapter 4: Semiconductors (Diodes and Transistors)* are similar to JFETs in construction and terminology. The difference is that, instead of using a p-n junction for a gate, a Schottky (metal-semiconductor) junction is employed. With the ability to switch at rates measured in tens of gigahertz (GHz), MESFETs are significantly faster than silicon-based JFETs and so

they are commonly used for applications like microwave frequency communications and radar.

DEPLETION-MODE MOSFETS

As we discussed in *Chapter 4: Semiconductors (Diodes and Transistors)*, as far back as 1928, the Austro-Hungarian scientist Dr. Julius Edgar Lilienfield (1882–1963) applied for US Patent 1,900,018 in which he described what we would now recognize as a depletion-mode MOSFET (*Metal-Oxide Semiconductor Field-Effect Transistor*).

As we see in Figure G.5, the depletion-mode MOSFET is conceptually the easiest to understand of all of the members of the field-effect transistor family.

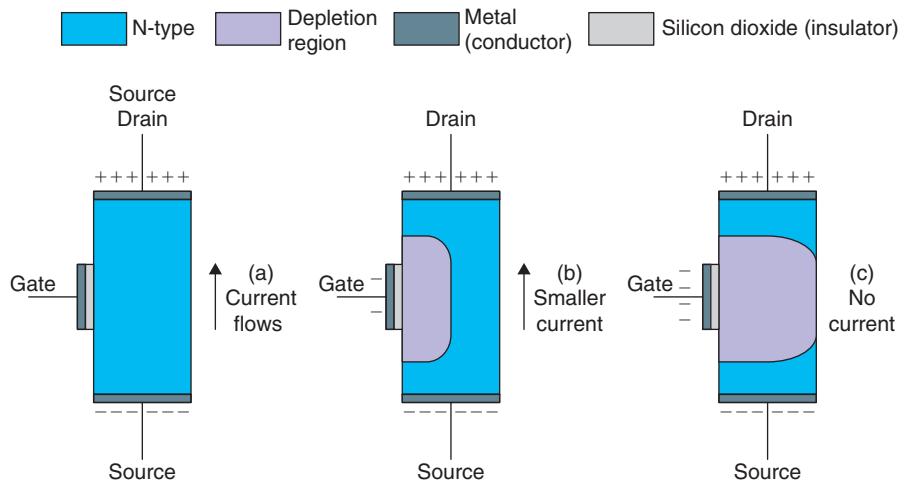


FIGURE G.5
Depletion-mode
n-channel MOSFET.

The control terminal is connected to a conducting plate, which is insulated from the silicon by a layer of nonconducting oxide. In the original devices the conducting plate was metal—hence, the term “metal-oxide”—but this is now something of a misnomer because modern versions tend to use a layer of *polycrystalline silicon (polysilicon)*. When a signal is applied to the gate terminal, the plate, insulated by the oxide, creates an electromagnetic field, which turns the transistor ON or OFF—hence, the term “field-effect.”

When no signal is being applied to the gate, the channel of N-type silicon between the source and the drain allows current to flow. This means that these devices are ON by default and we have to apply a signal to the gate terminal in order to turn them OFF.

If we apply a small negative potential to the gate terminal [Figure G.5(b)], the electrons on the gate repel the negative electrons in the N-type silicon. This leaves an area depleted of electron charge carriers, so we still refer to this as being a depletion region, even though it's not being formed by a p-n junction.

Once again, we can think of this as pinching a garden hosepipe to reduce the flow of water. Applying a negative potential to the gate terminal reduces the size of the conducting channel between the source and the drain. In turn, this increases the resistance of the channel and reduces the flow of current between the source and the drain terminals. If we keep on increasing the negative potential on the gate terminal, at some stage the depletion zone will completely block the channel as illustrated in Figure G.5(c).

Observe that the transistor shown in Figure G.5 is an n-channel device. We can also create an equivalent p-channel component by swapping the P-type and N-type diffusion regions.

Depletion-mode FETs can be used for a variety of purposes—typically in the processing of analog signals—acting as voltage-controlled resistors, for example.

ENHANCEMENT-MODE MOSFETS

As you may surmise from peeking at Figure G.6, the enhancement-mode MOSFET is conceptually the hardest to understand of all of the members of the field-effect transistor family. Enhancement-mode MOSFETs are formed from two p-n-junctions, which act like two back-to-back diodes. This means that, by

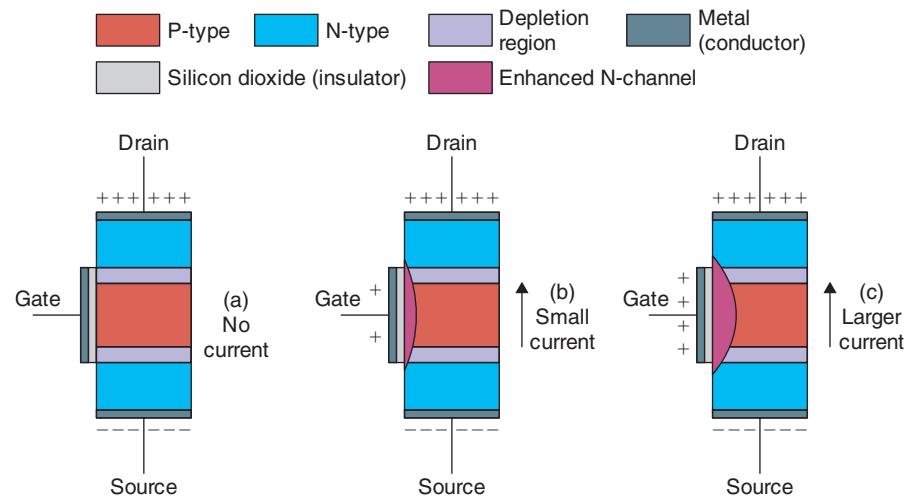


FIGURE G.6
Enhancement-mode
N-channel MOSFET.

default, this device is OFF and we have to apply an electrical potential to its gate terminal to turn it ON.

The MOSFET illustrated in Figure G.6 is an n-channel device (we could, of course, create its p-channel counterpart by swapping the P-type and N-type diffusion regions). This may be a little confusing at a first glance, because the term “channel” refers to the piece of silicon under the gate terminal that links the source and drain regions. As we see, this is formed from P-type material in our n-channel device.

Happily, there is reason behind the madness. In order to turn our n-channel device ON, a positive voltage is applied to the gate. This positive voltage repels positive holes and attracts negative electrons in the P-type material. The electrons accumulate beneath the oxide layer, where they form a negative channel—hence, the term “n-channel.”

Enhancement-mode MOSFETs are the most widely used member of the FET family, finding application in both analog and digital circuits, especially in today’s digital integrated circuits, the largest of which may literally contain billions of these transistors.

This page intentionally left blank

APPENDIX H

Rounding Algorithms 101

INTRODUCTION

We all remember being taught the concept of rounding in our younger years at school. Common problems involved monetary values, such as rounding some amount like \$26.19 to the nearest dollar (which would be 26 dollars, in the case of this example). However, although this may seem simple at a first glance, there's a lot more to rounding than might at first meet the eye ...

One key aspect of rounding that is easy to overlook (because it's so obvious) is that it involves transforming some quantity from a greater precision to a lesser precision. As we've just seen, for example, rounding a more precise value like \$26.19 to the nearest dollar results in 26 dollars, which is less precise.

435

This means that if we have to perform rounding, we would prefer to use an algorithm¹ that minimizes the effects of this loss of precision. We especially wish to minimize the loss of precision if we are performing some procedure that involves cycling around a loop performing a series of operations (including rounding) on the same data, over and over again. If we fail, the result will be so-called *creeping errors*, which refers to errors that increase over time as we iterate around the loop.

So how hard can this be? Well, in fact, things can become quite interesting, because there is a plethora of different rounding algorithms that we might use. These include *round-toward-nearest* (which itself encompasses *round-half-up* and *round-half-down*), *round-up*, *round-down*, *round-toward-zero*, *round-away-from-zero*, *round-ceiling*, *round-floor*, *truncation (chopping)* ... and the list goes on.

¹The term *algorithm*, which is named after the legendary Persian astrologer, astronomer, mathematician, scientist, and author Al-Khawarizmi [circa 800–840], refers to a detailed sequence of actions that are used to accomplish or perform a specific task.

Just to increase the fun and frivolity, some of these terms can sometimes refer to the same thing, while at other times they may differ (this can depend on whom you are talking to, the particular computer program or hardware implementation you are using, and so forth). Furthermore, the effect of a particular algorithm may vary depending on the form of number representation to which it is being applied, such as *unsigned values*, *sign-magnitude values*, and *signed (complement) values*.

In order to introduce the fundamental concepts, we'll initially assume that we are working with standard (sign-magnitude) decimal values, such as +3.142 and -3.142. (For the purposes of these discussions, any number without an explicit sign is assumed to be positive.) Also, we'll assume that we wish to round to integer values, which means that +3.142 will round to +3 (at least, it will if we are using the *round-toward-nearest* algorithm as discussed below). Once we have the basics out of the way, we'll consider some of the implications associated with applying rounding to different numerical representations, such as *signed binary numbers*.²

Round-Toward-Nearest

As its name suggests, this algorithm rounds towards the nearest significant value (this would be the nearest whole number, or integer, in the case of these particular examples).

In many ways, round-toward-nearest is the most intuitive of the various rounding algorithms. In this case, values such as 5.1, 5.2, 5.3, and 5.4 would round down to 5, while values of 5.6, 5.7, 5.8, and 5.9 would round up to 6 (Figure H.1).

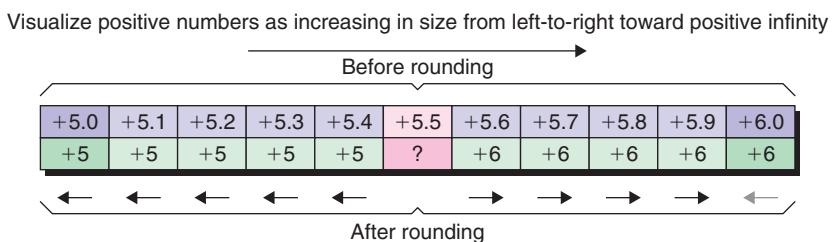


FIGURE H.1
Round-toward-nearest.

But what should we do in the case of a “half-way” value such as 5.5? Well, there are two obvious options: we could round it up to 6 or we could round it down to 5; these schemes, which are introduced below, are known as *round-half-up* and *round-half-down*, respectively.

²Unsigned and signed binary numbers were introduced in *Chapter 8: Binary Arithmetic*.

ROUND-HALF-UP (ARITHMETIC ROUNDING)

The *round-half-up* incarnation of the *round-toward-nearest* algorithm, which may also be referred to as *arithmetic rounding*, is the one that we typically associate with the concept of rounding that we learned at school. In this case, a “half-way” value such as 5.5 will round up to 6 (Figure H.2).

+5.0	+5.1	+5.2	+5.3	+5.4	+5.5	+5.6	+5.7	+5.8	+5.9	+6.0
+5	+5	+5	+5	+5	+6	+6	+6	+6	+6	+6

FIGURE H.2

Round-half-up (positive values).

One way to view this is that (at this level of precision and for this particular example) we can consider there to be ten values that commence with a 5 in the most-significant place (5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9). On this basis, it intuitively makes sense for five of the values to round down and for the other five to round up; that is, for the five values 5.0 through 5.4 to round down to 5, and for the remaining five values 5.5 through 5.9 to round up to 6.

Before we move on, it’s worth taking a few moments to note that “half-way” values will always round up when using this algorithm, irrespective of whether the final result is odd or even (Figure H.3).

+5.0	+5.1	+5.4	+5.5	+5.6	+5.9	+6.0	+6.1	+6.4	+6.5	+6.6	+6.9	+7.0
+5	+5	+5	+6	+6	+6	+6	+6	+6	+7	+7	+7	+7

FIGURE H.3

Round-half-up (odd and even positive values).

As we see from the above example, the 5.5 value rounds up to 6, which is an even number; and the 6.5 value rounds up to 7, which is an odd number. The reason we are emphasizing this point is that the *round-half-even* and *round-half-odd* algorithms—which are introduced a little later—would treat these two values differently. (The only reason we’ve omitted the values 5.2, 5.3, 5.7, 5.8, 6.2, 6.3, 6.7, and 6.8 from the above illustration is to save space so that we can display both odd and even values.)

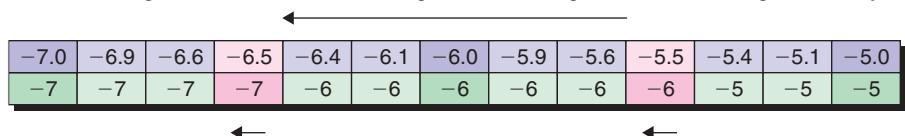
The tricky point with the *round-half-up* algorithm arrives when we come to consider negative numbers. There is no problem in the case of the values like -5.1 , -5.2 , -5.3 , and -5.4 , because these will all round to the nearest integer, which is -5 . Similarly, there is no problem in the case of values like -5.6 , -5.7 , -5.8 , and -5.9 , because these will all round to -6 . The problem arises

in the case of “half-way” values like -5.5 and -6.5 and our definition as to what “up” means in the context of “round-half-up.” Based on the fact that positive values like $+5.5$ and $+6.5$ round up to $+6$ and $+7$, respectively, most of us would intuitively expect their negative equivalents of -5.5 and -6.5 to round to -6 and -7 , respectively. In this case, we would say that our algorithm was symmetric (with respect to zero) for positive and negative values (Figure H.4).

FIGURE H.4

Round-half-up (odd and even negative values—symmetric implementation).

Visualize negative numbers as “increasing” in size from right-to-left toward negative infinity

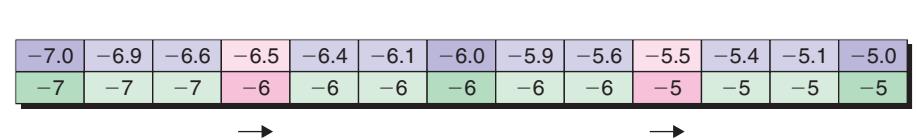


Observe that the above illustration refers to negative numbers as “increasing” in size toward negative infinity. In this case, we are talking about the absolute size of the values. [By “absolute,” we mean the size of the number if we disregard its sign and consider it to be a positive quantity; for example, $+1$ is bigger than -2 in real-world terms, but the absolute value of -2 (which is written as $|-2|$) is $+2$, which is bigger than $+1$].

But we digress. The point is that some applications (and some mathematicians) would regard “up” as referring to positive infinity. Based on this, -5.5 and -6.5 would actually round to -5 and -6 , respectively, in which case we would class this as being an asymmetric (with respect to zero) implementation of the round-half-up algorithm (Figure H.5).

FIGURE H.5

Round-half-up (odd and even negative values—asymmetric implementation).



The problem is that different applications treat things differently. For example, the *round method* of the Java Math Library provides an asymmetric implementation of the *round-half-up* algorithm, while the *round function* in the mathematical modeling, simulation, and visualization tool MATLAB® from The MathWorks (<http://www.mathworks.com/>) provides a symmetric implementation. And, just for giggles and grins, the *round function* in Visual Basic for Applications 6.0 actually implements the *round-half-even* (Banker’s rounding) algorithm, which is presented later in this appendix.

As a point of interest, the symmetric versions of rounding algorithms are sometimes referred to as *Gaussian implementations*. This is because the theoretical frequency distribution known as a “Gaussian Distribution”—which is named for the German mathematician and astronomer Karl Friedrich Gauss (1777–1855)—is symmetrical about its mean value.

ROUND-HALF-DOWN

Perhaps not surprisingly, this incarnation of the *round-toward-nearest* algorithm acts in the opposite manner to its *round-half-up* counterpart, as discussed in the previous topic. In this case, “half-way” values such as 5.5 and 6.5 will round *down* to 5 and 6, respectively (Figure H.6).

+5.0	+5.1	+5.4	+5.5	+5.6	+5.9	+6.0	+6.1	+6.4	+6.5	+6.6	+6.9	+7.0
+5	+5	+5	+5	+6	+6	+6	+6	+6	+6	+7	+7	+7

FIGURE H.6
Round-half-down
(odd and even positive
values).

Once again, we run into a problem when we come to consider negative numbers, because what we do with “half-way” values depends on what we understand the term “down” to mean. On the basis that positive values of +5.5 and +6.5 round to +5 and +6, respectively, a symmetric implementation of the round-half-down algorithm will round values of -5.5 and -6.5 to -5 and -6 , respectively (Figure H.7).

-7.0	-6.9	-6.6	-6.5	-6.4	-6.1	-6.0	-5.9	-5.6	-5.5	-5.4	-5.1	-5.0
-7	-7	-7	-6	-6	-6	-6	-6	-6	-5	-5	-5	-5

FIGURE H.7
Round-half-down (odd
and even negative
values—symmetric
implementation).

By comparison, in the case of an asymmetric implementation of the algorithm, in which “down” is understood to refer to negative infinity, values of -5.5 and -6.5 will actually be rounded to -6 and -7 , respectively (Figure H.8).

-7.0	-6.9	-6.6	-6.5	-6.4	-6.1	-6.0	-5.9	-5.6	-5.5	-5.4	-5.1	-5.0
-7	-7	-7	-7	-6	-6	-6	-6	-6	-6	-5	-5	-5

FIGURE H.8
Round-half-down (odd
and even negative
values—asymmetric
implementation).

ROUND-HALF-EVEN (BANKER'S ROUNDING)

If “half-way” values are always rounded in the same direction (for example, if +5.5 rounds up to +6 and +6.5 rounds up to +7, as is the case with the

round-half-up algorithm presented earlier), the result can be a bias that grows as more and more rounding operations are performed. One solution toward minimizing this bias is to sometimes round up and sometimes round down.

In the case of the round-half-even algorithm (which is often referred to as “Bankers Rounding” because it is commonly used in financial calculations), half-way values are rounded toward the nearest *even number*. Thus, +5.5 will round *up* to +6 and +6.5 will round *down* to +6 (Figure H.9).

FIGURE H.9

Round-half-even (odd and even positive values).

+5.0	+5.1	+5.4	+5.5	+5.6	+5.9	+6.0	+6.1	+6.4	+6.5	+6.6	+6.9	+7.0
+5	+5	+5	+6	+6	+6	+6	+6	+6	+6	+7	+7	+7

→ ←

The *round-half-even* algorithm is, by definition, symmetric for positive and negative values, so both -5.5 and -6.5 will round to the nearest even value, which is -6 (Figure H.10).

FIGURE H.10

Round-half-even (odd and even negative values).

-7.0	-6.9	-6.6	-6.5	-6.4	-6.1	-6.0	-5.9	-5.6	-5.5	-5.4	-5.1	-5.0
-7	-7	-7	-6	-6	-6	-6	-6	-6	-6	-5	-5	-5

→ ←

ROUND-HALF-ODD

This is the theoretical counterpart to the *round-half-even* algorithm; but in this case, “half-way” values are rounded toward the nearest *odd number*. For example, $+5.5$ and $+6.5$ will round to $+5$ and $+7$, respectively (Figure H.11).

FIGURE H.11

Round-half-odd (odd and even positive values).

+5.0	+5.1	+5.4	+5.5	+5.6	+5.9	+6.0	+6.1	+6.4	+6.5	+6.6	+6.9	+7.0
+5	+5	+5	+5	+6	+6	+6	+6	+6	+7	+7	+7	+7

← →

As for its “even” counterpart, the *round-half-odd* algorithm is, by definition, symmetric for positive and negative values. Thus, -5.5 will round to -5 and -6.5 will round to -7 (Figure H.12).

FIGURE H.12

Round-half-odd (odd and even negative values).

-7.0	-6.9	-6.6	-6.5	-6.4	-6.1	-6.0	-5.9	-5.6	-5.5	-5.4	-5.1	-5.0
-7	-7	-7	-7	-6	-6	-6	-6	-6	-5	-5	-5	-5

← →

The reason we used the “theoretical” qualifier at the beginning of this topic is that, in practice, the *round-half-odd* algorithm is rarely (if ever) used because it will never round to zero, and rounding to zero is often a desirable attribute for rounding algorithms.

ROUND-CEILING (TOWARD POSITIVE INFINITY)

This refers to always rounding towards positive infinity. In the case of a positive number, the result will remain unchanged if the digits to be discarded are all zero; otherwise it will be rounded up. For example, +5.0 will be rounded to +5, but +5.1, +5.2, +5.3, +5.4, +5.5, +5.6, +5.7, +5.8, and +5.9 will all be rounded up to +6. (Similarly, +6.0 will be rounded to +6, while +6.1 through +6.9 will all be rounded up to +7; Figure H.13.)

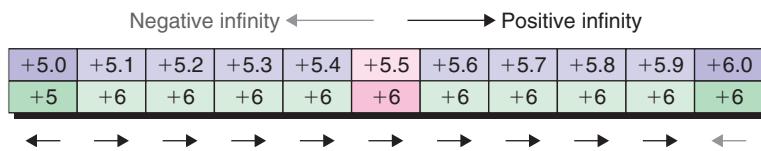


FIGURE H.13

Round-ceiling (positive values).

By comparison, in the case of a negative number, the unwanted digits are simply discarded. For example, -5.0, -5.1, -5.2, -5.3, -5.4, -5.5, -5.6, -5.7, -5.8, and -5.9 will all be rounded to -5 (Figure H.14).

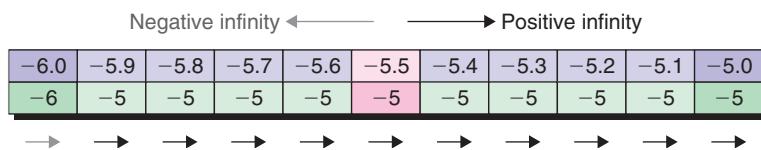


FIGURE H.14

Round-ceiling (negative values).

From the above illustrations, it's easy to see that the *round-ceiling* algorithm results in a cumulative positive bias. Thus, in the case of software implementations, or during analysis of a system using software simulation, this form of rounding is sometimes employed to determine the upper limit of the algorithm (that is, the upper limit of the results generated by the algorithm for a given data set) for use in diagnostic functions.

In the case of hardware (a physical realization in logic gates), this algorithm requires a significant amount of additional logic, and it is therefore rarely used in hardware implementations.

ROUND-FLOOR (TOWARD NEGATIVE INFINITY)

This is the counterpart to the *round-ceiling* algorithm; the difference being that in this case we round toward negative infinity. This means that, in the case of a negative value, the result will remain unchanged if the digits to be discarded are all zero; otherwise it will be rounded toward negative infinity. For example, -5.0 will be rounded to -5 , but -5.1 , -5.2 , -5.3 , -5.4 , -5.5 , -5.6 , -5.7 , -5.8 , and -5.9 will all be rounded to -6 (similarly, -6.0 will be rounded to -6 , while -6.1 through -6.9 will all be rounded to -7 ; Figure H.15).

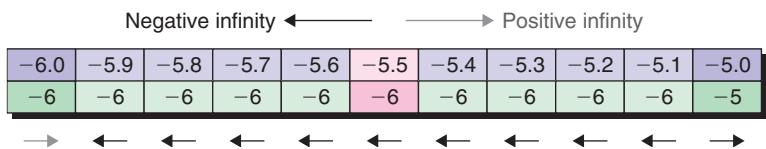


FIGURE H.15
Round-floor (negative values).

By comparison, in the case of a positive value, the unwanted digits are simply discarded. For example, $+5.0$, $+5.1$, $+5.2$, $+5.3$, $+5.4$, $+5.5$, $+5.6$, $+5.7$, $+5.8$, and $+5.9$ will all be rounded to $+5$ (Figure H.16).

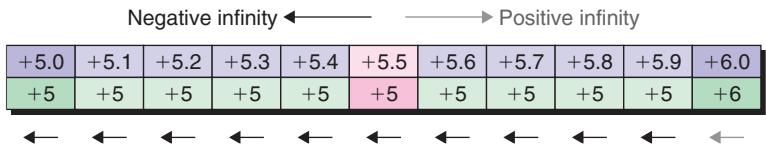


FIGURE H.16
Round-floor (positive values).

From the above illustrations, it's easy to see that the round-floor algorithm results in a cumulative negative bias. Thus, as for its *round-ceiling* counterpart, in the case of software implementations, or during analysis of a system using software simulation, the *round-floor* technique is sometimes employed to determine the lower limit of the algorithm (that is, the lower limit of the results generated by the algorithm for a given data set) for use in diagnostic functions.

Furthermore, this approach is "cheap" in terms of hardware (a physical realization in logic gates) when working with signed binary numbers since it involves only a simple truncation (the reason why this should be so is discussed later in this Appendix); this technique is therefore very often used with regard to hardware implementations.

ROUND-TOWARD-ZERO

As its name suggests, this refers to rounding in such a way that the result heads toward zero. For example, +5.0, +5.1, +5.2, +5.3, +5.4, +5.5, +5.6, +5.7, +5.8, and +5.9 will all be rounded to +5 (this works the same for odd and even values, so +6.0 through +6.9 will all be rounded to +6; Figure H.17).

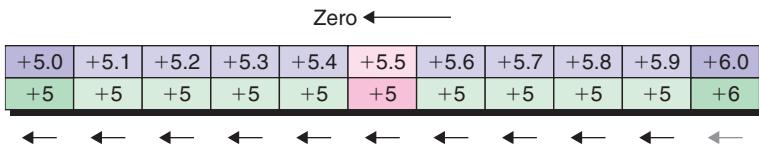


FIGURE H.17

Round-toward-zero (positive values).

Similarly, -5.0, -5.1, -5.2, -5.3, -5.4, -5.5, -5.6, -5.7, -5.8, and -5.9 will all be rounded to -5 (again, this works the same for odd and even values, so -6.0 through -6.9 will all be rounded to -6; Figure H.18).

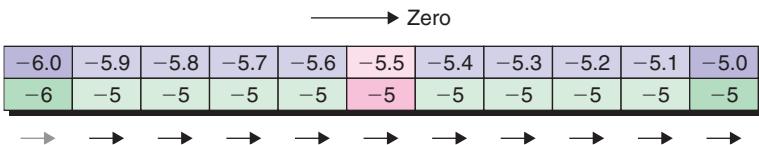


FIGURE H.18

Round-toward-zero (negative values).

Another way to think about this form of rounding is that it acts in the same way as the *round-floor* algorithm for positive numbers and as the *round-ceiling* algorithm for negative numbers.

ROUND-AWAY-FROM-ZERO

This is the counterpart to the *round-toward-zero* algorithm; in this case, of course, we round away from zero. For example, +5.1, +5.2, +5.3, +5.4, +5.5, +5.6, +5.7, +5.8, and +5.9 will all be rounded to +6 (this works the same for odd and even values, so +6.1 through +6.9 will all be rounded to +7; Figure H.19).

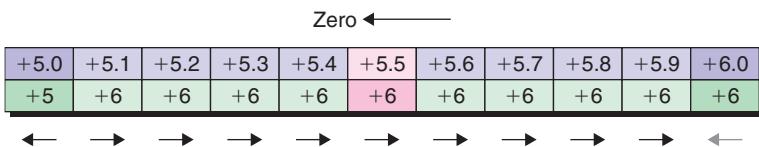


FIGURE H.19

Round-away-from-zero (positive values).

Similarly, -5.0 , -5.1 , -5.2 , -5.3 , -5.4 , -5.5 , -5.6 , -5.7 , -5.8 , and -5.9 will all be rounded to -6 (again, this works the same way for odd and even values, so -6.1 through -6.9 will all be rounded to -7 ; Figure H.20).

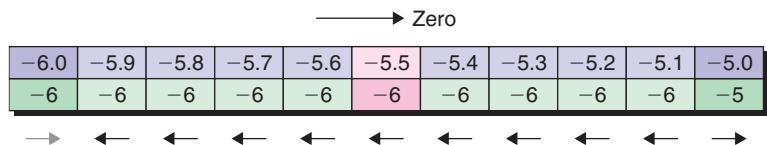


FIGURE H.20

Round-away-from-zero (negative values).

Another way to think about this form of rounding is that it acts in the same way as the *round-ceiling* algorithm for positive numbers and as the *round-floor* algorithm for negative numbers.

ROUND-UP

The actions of this rounding mode depend on what one means by “up.” Some applications understand “up” to refer to heading towards positive infinity; in this case, *round-up* is synonymous for *round-ceiling*. Alternatively, some applications regard “up” as referring to an absolute value heading away from zero; in this case, *round-up* acts in the same manner as the *round-away-from-zero* algorithm.

ROUND-DOWN

This is the counterpart to the *round-up* algorithm. The actions of this mode depend on what one means by “down.” Some applications understand “down” to refer to heading towards negative infinity; in this case, *round-down* is synonymous for *round-floor*. Alternatively, some applications regard “down” as referring to an absolute value heading toward zero; in this case, *round-down* acts in the same manner as the *round-toward-zero* algorithm.

TRUNCATION (CHOPPING)

Also known as *chopping*, *truncation* simply means discarding any unwanted digits. This means that in the case of the standard (sign-magnitude) decimal values we’ve been considering thus far—and also when working with *unsigned binary values*—the actions of truncation are identical to those of the *round-toward-zero* mode.

But things are never simple: when working with *signed binary values*, the actions of truncation reflect those of the *round-floor mode* (see also the discussions on *Rounding Sign-Magnitude Binary Numbers* and *Rounding Signed Binary Numbers* later in this appendix).

ROUND-ALTERNATE

Also known as *alternate rounding*, this is similar in concept to the *round-half-even* and *round-half-odd* schemes discussed earlier, in that the purpose of this algorithm is to minimize the bias that can be caused by always rounding “half-way” values in the same direction.

The problem with the *round-half-even* scheme, for example, is that it is possible for a bias to occur if the data being processed contained a disproportionate number of odd and even “half-way” values. One solution is to use the *round-alternate* approach, in which the first “half-way” value is rounded up (for example), the next is rounded down, the next up, the next down, and so on.

ROUND-RANDOM (STOCHASTIC ROUNDING)

This may also be referred to as *random rounding* or *stochastic rounding*, where the term “stochastic” comes from the Greek *stokhazesthai*, meaning “to guess at.” In this case, when the algorithm is presented with a “half-way” value, it effectively tosses a metaphorical coin in the air and randomly (or pseudo-randomly) rounds the value up or down.

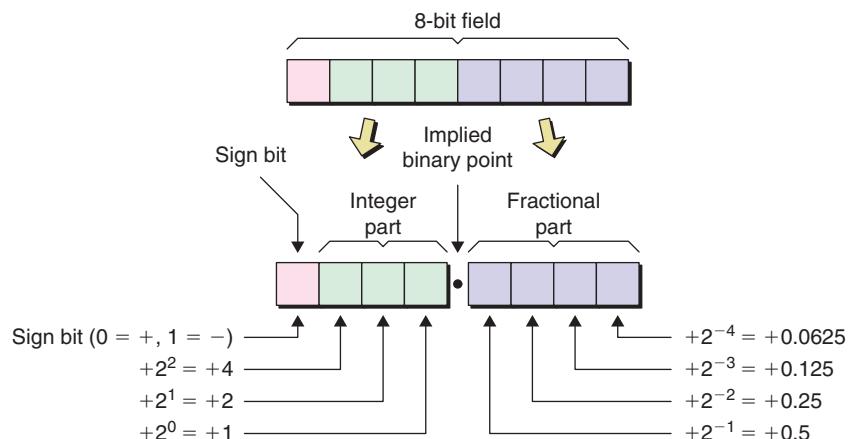
Although this technique typically gives the best overall results over large numbers of calculations, it is only employed in very specialized applications, because the nature of this algorithm makes it difficult to implement and tricky to verify the results.

ROUNDING SIGN-MAGNITUDE BINARY VALUES

Until this point, all of the discussions in this appendix have been based on the concept of standard sign-magnitude decimal values. Now let's consider what happens to our rounding algorithms in the case of binary representations. For the purposes of these discussions, we will focus on the *truncation* and *round-half-up* algorithms, because these are the techniques commonly used in hardware implementations constructed out of physical logic gates (in turn, this is because these algorithms entail relatively little overhead in terms of additional logic).

We'll start by considering an 8-bit binary sign-magnitude fixed-point representation comprising a sign bit, three integer bits, and four fractional bits (Figure H.21).

Remember that the differences between unsigned, sign-magnitude, and signed binary numbers were introduced in *Chapter 8: Binary Arithmetic*. For our purposes here, we need only note that—in the case of a sign-magnitude representation—the sign bit is used only to represent the sign of the value

**FIGURE H.21**

8-bit sign-magnitude binary 1.3.4 fixed-point representation.

FIGURE H.22

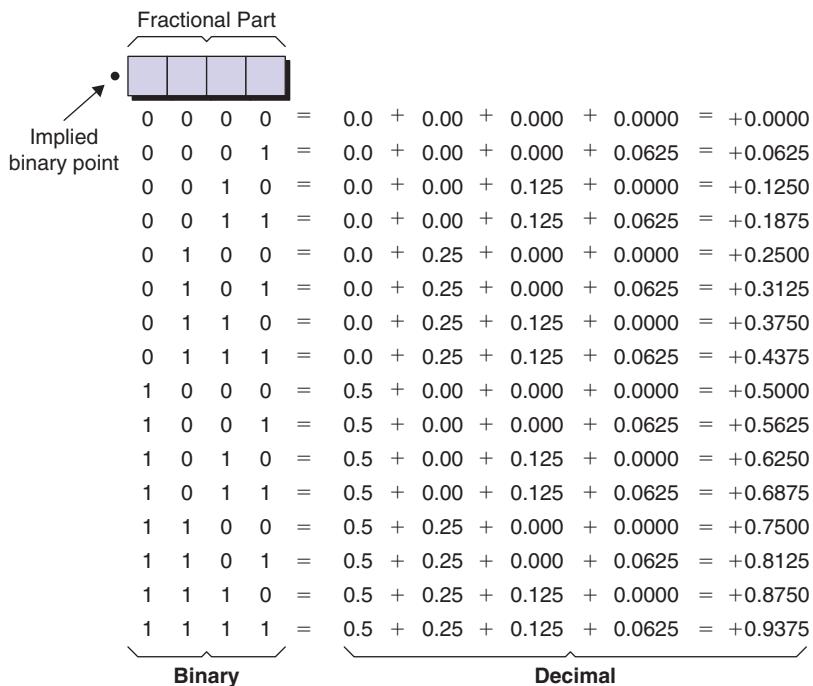
The sign bit and the three integer bits of our 1.3.4 sign-magnitude representation.

(0 = positive, 1 = negative). In the case of this particular example, we have three integer bits that can be used to represent an integer in the range 0 to 7. Thus, the combination of the sign bit and the three integer bits allows us to represent positive and negative integers in the range -7 through $+7$ (Figure H.22).

	Sign bit	Integer bits	•	Implied binary point
+0	0	0 0 0		
+1	0	0 0 1		
+2	0	0 1 0		
+3	0	0 1 1		
+4	0	1 0 0		
+5	0	1 0 1		
+6	0	1 1 0		
+7	0	1 1 1		
-0	1	0 0 0		
-1	1	0 0 1		
-2	1	0 1 0		
-3	1	0 1 1		
-4	1	1 0 0		
-5	1	1 0 1		
-6	1	1 1 0		
-7	1	1 1 1		
Decimal		Binary		

As we know [at least, we know if we've read *Chapter 8* (hint, hint)], one of the problems with this format is that we end up with both positive and negative versions of zero, but that need not concern us here.

When we come to the fractional portion of our 8-bit field, the first fractional bit is used to represent $1/2 = 0.5$; the second fractional bit is used to represent $0.5/2 = 0.25$; the third fractional bit is used to represent $0.25/2 = 0.125$, and the fourth fractional bit is used to represent $0.125/2 = 0.0625$. Thus, our 4-bit field allows us to represent fractional values in the range 0.0 through 0.9375 as shown in Figure H.23 (of course, more fractional bits would allow us to represent more precise fractional values, but four bits will serve the purposes of our discussions here).

**FIGURE H.23**

The four fractional bits of our 1.3.4 sign-magnitude representation.

Truncation

So, now let's suppose that we have a binary value of 0101.0100, which equates to +5.25 in decimal. If we simply truncate this by removing the fractional field, we end up with an integer value of 0101 in binary, or +5 in decimal, which is what we would expect. Similarly, if we started with a value of 1101.0100, which equates to -5.25 in decimal, then truncating the fractional part leaves us with an integer value of 1101 in binary, or -5 in decimal, which—again—is what we expect. Let's try this with some other values, as illustrated in Figure H.24.

The end result is that, as we previously noted, in the case of sign-magnitude binary numbers, using truncation (simply discarding the fractional bits) is the same as performing the *round-toward-zero* algorithm as discussed earlier in this appendix. Also, as we see, this works exactly the same way for both positive and negative (and odd and even) values. (Compare these results to those obtained when performing this operation on signed binary values as discussed in the next topic.)

	Initial binary value	Truncate
Positive values	0 1 0 1 . 0 0 0 0 = +5.00	0 1 0 1 = +5
	0 1 0 1 . 0 1 0 0 = +5.25	0 1 0 1 = +5
	0 1 0 1 . 1 0 0 0 = +5.50	0 1 0 1 = +5
	0 1 0 1 . 1 1 0 0 = +5.75	0 1 0 1 = +5
	0 1 1 0 . 0 0 0 0 = +6.00	0 1 1 0 = +6
	0 1 1 0 . 0 1 0 0 = +6.25	0 1 1 0 = +6
	0 1 1 0 . 1 0 0 0 = +6.50	0 1 1 0 = +6
	0 1 1 0 . 1 1 0 0 = +6.75	0 1 1 0 = +6
Negative values	1 1 0 1 . 0 0 0 0 = -5.00	1 1 0 1 = -5
	1 1 0 1 . 0 1 0 0 = -5.25	1 1 0 1 = -5
	1 1 0 1 . 1 0 0 0 = -5.50	1 1 0 1 = -5
	1 1 0 1 . 1 1 0 0 = -5.75	1 1 0 1 = -5
	1 1 1 0 . 0 0 0 0 = -6.00	1 1 1 0 = -6
	1 1 1 0 . 0 1 0 0 = -6.25	1 1 1 0 = -6
	1 1 1 0 . 1 0 0 0 = -6.50	1 1 1 0 = -6
	1 1 1 0 . 1 1 0 0 = -6.75	1 1 1 0 = -6

FIGURE H.24

Applying truncation to sign-magnitude binary numbers.

Round-Half-Up

As opposed to *truncation*, the other common rounding algorithm used in hardware implementations is that of *round-half-up*. The reason this is so popular (as compared to a round-half-even approach, for example) is that it doesn't require us to perform any form of comparison; all we have to do is to add 0.5 to our original value and then truncate the result. For example, let's suppose that we have a binary value of 0101.1100, which equates to +5.75 in decimal. If we now add 0000.1000 (which equates to 0.5 in decimal) we end up with 0110.0100, which equates to +6.25 in decimal. And if we then truncate this value, we end up with 0110, or +6 in decimal, which is exactly what we would expect from a *round-half-up* algorithm. Let's try this with some other values as illustrated in Figure H.25.

Thus, the end result is that, in the case of sign-magnitude binary numbers, adding a value of 0.5 and truncating the product gives exactly the same results as performing a symmetrical version of the *round-half-up* algorithm, as discussed earlier in this appendix. And, as we would expect from the symmetrical version of this algorithm, this works exactly the same way for both positive and negative (and odd and even) values. (Once again, compare these results to those obtained when performing this operation on signed binary values as discussed in the next topic.)

Initial binary value	Add 0000.1000 (0.5 in decimal)	Truncate
0 1 0 1 . 0 0 0 0 = +5.00	0 1 0 1 . 1 0 0 0 = +5.50	0 1 0 1 = +5
0 1 0 1 . 0 1 0 0 = +5.25	0 1 0 1 . 1 1 0 0 = +5.75	0 1 0 1 = +5
0 1 0 1 . 1 0 0 0 = +5.50	0 1 1 0 . 0 0 0 0 = +6.00	0 1 1 0 = +6
0 1 0 1 . 1 1 0 0 = +5.75	0 1 1 0 . 0 1 0 0 = +6.25	0 1 1 0 = +6
0 1 1 0 . 0 0 0 0 = +6.00	0 1 1 0 . 1 0 0 0 = +6.50	0 1 1 0 = +6
0 1 1 0 . 0 1 0 0 = +6.25	0 1 1 0 . 1 1 0 0 = +6.75	0 1 1 0 = +6
0 1 1 0 . 1 0 0 0 = +6.50	0 1 1 1 . 0 0 0 0 = +7.00	0 1 1 1 = +7
0 1 1 0 . 1 1 0 0 = +6.75	0 1 1 1 . 0 1 0 0 = +7.25	0 1 1 1 = +7
1 1 0 1 . 0 0 0 0 = -5.00	1 1 0 1 . 1 0 0 0 = -5.50	1 1 0 1 = -5
1 1 0 1 . 0 1 0 0 = -5.25	1 1 0 1 . 1 1 0 0 = -5.75	1 1 0 1 = -5
1 1 0 1 . 1 0 0 0 = -5.50	1 1 1 0 . 0 0 0 0 = -6.00	1 1 0 1 = -6
1 1 0 1 . 1 1 0 0 = -5.75	1 1 1 0 . 0 1 0 0 = -6.25	1 1 0 1 = -6
1 1 1 0 . 0 0 0 0 = -6.00	1 1 1 0 . 1 0 0 0 = -6.50	1 1 1 0 = -6
1 1 1 0 . 0 1 0 0 = -6.25	1 1 1 0 . 1 1 0 0 = -6.75	1 1 1 0 = -6
1 1 1 0 . 1 0 0 0 = -6.50	1 1 1 1 . 0 0 0 0 = -7.00	1 1 1 0 = -7
1 1 1 0 . 1 1 0 0 = -6.75	1 1 1 1 . 0 1 0 0 = -7.25	1 1 1 0 = -7

FIGURE H.25
Applying round-half-up to sign-magnitude binary numbers.

ROUNDING SIGNED BINARY VALUES

For this portion of our discussions, we will base our examples on an 8-bit signed binary fixed-point representation comprising four integer bits (the most-significant of which also acts as a sign bit) and four fractional bits (Figure H.26).

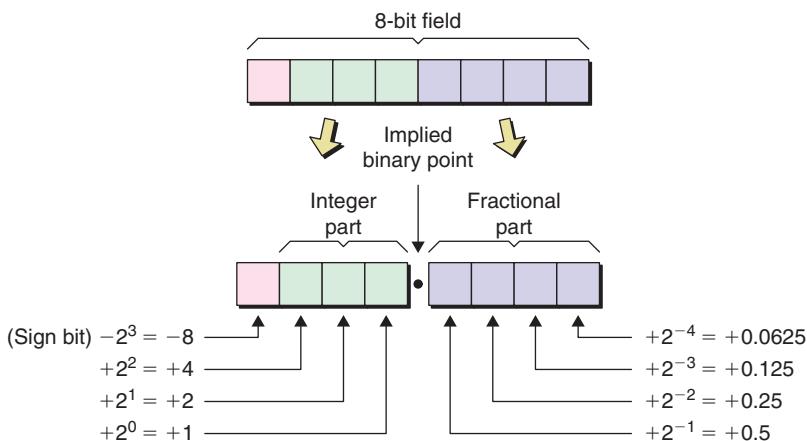


FIGURE H.26

8-bit signed binary 4.4 fixed-point representation.

	Integer bits	
Sign bit →		•
+0	0 0 0 0	
+1	0 0 0 1	
+2	0 0 1 0	
+3	0 0 1 1	
+4	0 1 0 0	
+5	0 1 0 1	
+6	0 1 1 0	
+7	0 1 1 1	
$-8 + 0 = -8$	1 0 0 0	
$-8 + 1 = -7$	1 0 0 1	
$-8 + 2 = -6$	1 0 1 0	
$-8 + 3 = -5$	1 0 1 1	
$-8 + 4 = -4$	1 1 0 0	
$-8 + 5 = -3$	1 1 0 1	
$-8 + 6 = -2$	1 1 1 0	
$-8 + 7 = -1$	1 1 1 1	
<hr/> Decimal		<hr/> Binary

FIGURE H.27

The sign bit and the three integer bits of our 4.4 signed binary representation.

Once again, the differences between unsigned, sign-magnitude, and signed binary numbers were introduced in *Chapter 8: Binary Arithmetic*. For our purposes here, we need only note that—in the case of a signed binary representation—the sign bit is used to signify a negative quantity (not just the sign), while the remaining bits continue to represent positive values. Thus, in the case of our 4-bit integer field, a 1 in the sign bit reflects a value of $-2^3 = -8$, which therefore allows us to use this 4-bit field to represent integers in the range -8 through $+7$ (Figure H.27).

The fractional portion of our 8-bit field behaves in exactly the same manner as for the sign-magnitude representations we discussed in the previous section; the first fractional bit is used to represent $1/2 = 0.5$;

the second fractional bit is used to represent $0.5/2 = 0.25$; the third fractional bit is used to represent $0.25/2 = 0.125$; and the fourth fractional bit is used to represent $0.125/2 = 0.0625$. Thus, our four-bit field allows us to represent fractional values in the range 0.0 through 0.9375 as shown below (once again, more fractional bits would allow us to represent more precise fractional values, but four bits will serve the purposes of our discussions) (Figure H.28).

Truncation

So, now let's suppose that we have a signed binary value of 0101.1000, which equates to $+5.5$ in decimal. If we simply truncate this by removing the fractional field, we end up with an integer value of 0101 in binary, or $+5$ in decimal, which is what we would expect.

However, now consider what happens if we wish to perform the same operation on the equivalent negative value of -5.5 . In this case, our binary value will be 1010.1000, which equates to $-8 + 2 + 0.5 = -5.5$ in decimal. Thus, truncating the fractional part leaves us with an integer value of 1010 in binary, or -6 (as opposed to the -5 we received in the case of the sign-magnitude representations in the previous section). Let's try this with some other values, as illustrated in Figure H.29.

Fractional part		
Binary		Decimal
0	0	= 0.0 + 0.00 + 0.000 + 0.0000 = +0.0000
0	0	= 0.0 + 0.00 + 0.000 + 0.0625 = +0.0625
0	0	= 0.0 + 0.00 + 0.125 + 0.0000 = +0.1250
0	0	= 0.0 + 0.00 + 0.125 + 0.0625 = +0.1875
0	1	= 0.0 + 0.25 + 0.000 + 0.0000 = +0.2500
0	1	= 0.0 + 0.25 + 0.000 + 0.0625 = +0.3125
0	1	= 0.0 + 0.25 + 0.125 + 0.0000 = +0.3750
0	1	= 0.0 + 0.25 + 0.125 + 0.0625 = +0.4375
1	0	= 0.5 + 0.00 + 0.000 + 0.0000 = +0.5000
1	0	= 0.5 + 0.00 + 0.000 + 0.0625 = +0.5625
1	0	= 0.5 + 0.00 + 0.125 + 0.0000 = +0.6250
1	0	= 0.5 + 0.00 + 0.125 + 0.0625 = +0.6875
1	1	= 0.5 + 0.25 + 0.000 + 0.0000 = +0.7500
1	1	= 0.5 + 0.25 + 0.000 + 0.0625 = +0.8125
1	1	= 0.5 + 0.25 + 0.125 + 0.0000 = +0.8750
1	1	= 0.5 + 0.25 + 0.125 + 0.0625 = +0.9375

FIGURE H.28

The four fractional bits of our 4.4 signed binary representation.

Initial binary value		Truncate	
Positive values	0 1 0 1 . 0 0 0 0 =	+5.00	0 1 0 1 = +5
	0 1 0 1 . 0 1 0 0 =	+5.25	0 1 0 1 = +5
	0 1 0 1 . 1 0 0 0 =	+5.50	0 1 0 1 = +5
	0 1 0 1 . 1 1 0 0 =	+5.75	0 1 0 1 = +5
	0 1 1 0 . 0 0 0 0 =	+6.00	0 1 1 0 = +6
	0 1 1 0 . 0 1 0 0 =	+6.25	0 1 1 0 = +6
	0 1 1 0 . 1 0 0 0 =	+6.50	0 1 1 0 = +6
	0 1 1 0 . 1 1 0 0 =	+6.75	0 1 1 0 = +6
Negative values	1 0 1 1 . 0 0 0 0 = -8 + 3 + 0.00 = -5.00	1 0 1 1 = -5	
	1 0 1 0 . 1 1 0 0 = -8 + 2 + 0.75 = -5.25	1 0 1 0 = -6	
	1 0 1 0 . 1 0 0 0 = -8 + 2 + 0.50 = -5.50	1 0 1 0 = -6	
	1 0 1 0 . 0 1 0 0 = -8 + 2 + 0.25 = -5.75	1 0 1 0 = -6	
	1 0 1 0 . 0 0 0 0 = -8 + 2 + 0.00 = -6.00	1 0 1 0 = -6	
	1 0 0 1 . 1 1 0 0 = -8 + 1 + 0.75 = -6.25	1 0 0 1 = -7	
	1 0 0 1 . 1 0 0 0 = -8 + 1 + 0.50 = -6.50	1 0 0 1 = -7	
	1 0 0 1 . 0 1 0 0 = -8 + 1 + 0.25 = -6.75	1 0 0 1 = -7	

FIGURE H.29

Applying truncation to signed binary numbers.

Observe the values shown in red/bold and compare these values to those obtained when performing this operation on sign-magnitude binary values as discussed in the previous topic. The end result is that, as we previously noted, in the case of signed binary numbers, using truncation (simply discarding the fractional bits) is the same as performing the *round-floor* algorithm as discussed earlier in this appendix.

Round-Half-Up

As we previously noted, the other common rounding algorithm used in hardware implementations is that of *round-half-up*. Once again, the reason this algorithm is so popular (as compared to a *round-half-even* approach, for example) is that it doesn't require us to perform any form of comparison; all we have to do is to add 0.5 to our original value and then truncate the result.

As you will recall, in the case of positive values, we expect this algorithm to round to the next integer for fractional values of 0.5 and higher. For example, +5.5 should round to +6. Let's see if this works. If we have an initial value of 0101.1000 (which equates to +5.5 in decimal) and we add 0000.1000 (which equates to 0.5 in decimal) we end up with 0110.0000, which equates to +6.0 in decimal. And if we now truncate this value, we end up with 0110 or +6 in decimal, which is what we would expect.

But what about negative values? If we have an initial value of 1010.1000 (which equates to $-8 + 2 + 0.5 = -5.5$ in decimal) to which we add 0000.1000 (which equates to 0.5 in decimal) we end up with 1011.0000, which equates to $-8 + 3 = -5$ in decimal. If we then truncate this value, we end up with 1011 or -5 in decimal, as opposed to the value of -6 that we might have hoped for. Just to make the point a little more strongly, let's try this with some other values, as illustrated in Figure H.30.

As we see, in the case of signed binary numbers, adding a value of 0.5 and truncating the product gives exactly the same results as performing an asymmetrical version of the round-half-up algorithm as discussed earlier in this appendix. As we would expect from the asymmetrical version of this algorithm, this works differently for positive and negative values that fall exactly on the "half-way" (0.5) boundary. (Compare these results to those obtained when performing this operation on signed binary values, as discussed in the previous topic.)

SUMMARY

Figure H.31 reflects a summary of the main rounding modes discussed above, as applied to standard (sign-magnitude) decimal values .

Initial binary value	Add 0000.1000 (0.5 in decimal)	Truncate
0 1 0 1 . 0 0 0 0 = +5.00	0 1 0 1 . 1 0 0 0 = +5.50	0 1 0 1 = +5
0 1 0 1 . 0 1 0 0 = +5.25	0 1 0 1 . 1 1 0 0 = +5.75	0 1 0 1 = +5
0 1 0 1 . 1 0 0 0 = +5.50	0 1 1 0 . 0 0 0 0 = +6.00	0 1 1 0 = +6
0 1 0 1 . 1 1 0 0 = +5.75	0 1 1 0 . 0 1 0 0 = +6.25	0 1 1 0 = +6
0 1 1 0 . 0 0 0 0 = +6.00	0 1 1 0 . 1 0 0 0 = +6.50	0 1 1 0 = +6
0 1 1 0 . 0 1 0 0 = +6.25	0 1 1 0 . 1 1 0 0 = +6.75	0 1 1 0 = +6
0 1 1 0 . 1 0 0 0 = +6.50	0 1 1 1 . 0 0 0 0 = +7.00	0 1 1 1 = +7
0 1 1 0 . 1 1 0 0 = +6.75	0 1 1 1 . 0 1 0 0 = +7.25	0 1 1 1 = +7
1 0 1 1 . 0 0 0 0 = -5.00	1 0 1 1 . 1 0 0 0 = -4.50	1 0 1 1 = -5
1 0 1 0 . 1 1 0 0 = -5.25	1 0 1 1 . 0 1 0 0 = -4.75	1 0 1 1 = -5
1 0 1 0 . 1 0 0 0 = -5.50	1 0 1 1 . 0 0 0 0 = -5.00	1 0 1 1 = -5
1 0 1 0 . 0 1 0 0 = -5.75	1 0 1 0 . 1 1 0 0 = -5.25	1 0 1 0 = -6
1 0 1 0 . 0 0 0 0 = -6.00	1 0 1 0 . 1 0 0 0 = -5.50	1 0 1 0 = -6
1 0 0 1 . 1 1 0 0 = -6.25	1 0 1 0 . 0 1 0 0 = -5.75	1 0 1 0 = -6
1 0 0 1 . 1 0 0 0 = -6.50	1 0 1 0 . 0 0 0 0 = -6.00	1 0 1 0 = -6
1 0 0 1 . 0 1 0 0 = -6.75	1 0 0 1 . 1 1 0 0 = -6.25	1 0 0 1 = -7

FIGURE H.30
Applying round-half-up to signed binary numbers.

Mode	← Negative infinity					Zero →					← Zero					Positive infinity →				
R-H-U (s)	-2	-2	-2	-1	-1	-1	-1	-0	0	0	1	1	1	1	2	2	2	2	2	2
R-H-U (a)	-2	-2	-1	-1	-1	-1	-0	-0	0	0	1	1	1	1	2	2	2	2	2	2
R-H-D (s)	-2	-2	-1	-1	-1	-1	-0	-0	0	0	0	1	1	1	1	2	2	2	2	2
R-H-D (a)	-2	-2	-2	-1	-1	-1	-1	-0	0	0	0	1	1	1	1	2	2	2	2	2
R-H-E	-2	-2	-2	-1	-1	-1	-0	-0	0	0	0	1	1	1	1	2	2	2	2	2
R-H-O	-2	-2	-1	-1	-1	-1	-1	-0	0	0	1	1	1	1	1	2	2	2	2	2
R-C	-2	-1	-1	-1	-1	-0	-0	-0	0	1	1	1	1	2	2	2	2	2	2	2
R-F	-2	-2	-2	-2	-1	-1	-1	-1	0	0	0	0	1	1	1	1	1	1	1	2
R-T-Z	-2	-1	-1	-1	-1	-0	-0	-0	0	0	0	0	1	1	1	1	1	1	1	2
R-A-F-Z	-2	-2	-2	-2	-1	-1	-1	-1	0	1	1	1	1	2	2	2	2	2	2	2

R-H-U = Round-Half-Up

R-H-E = Round-Half-Even

R-C = Round-Ceiling

R-T-Z = Round-Toward-Zero

R-H-D = Round-Half-Down

R-H-O = Round-Half-Odd

R-F = Round-Floor

R-A-F-Z = Round-Away-From-Zero

(s) = Symmetric

(a) = Asymmetric

FIGURE H.31

Summary of rounding algorithms.

With regard to the above illustration, it's important to remember the following points (all of which are covered in more detail in our earlier discussions):

- The generic concept of *round-toward-nearest* encompasses both the *round-half-up* and *round-half-down* modes.
- The term *arithmetic rounding* is another name for the *round-half-up* algorithm (of which there can be both symmetric and asymmetric versions).
- Depending on the application/implementation, the actions of the *round-up* algorithm may either equate to those of the *round-ceiling* mode or to those of the *round-away-from-zero* modes.
- Depending on the application/implementation, the actions of the *round-down* algorithm may either equate to those of the *round-floor* mode or to those of the *round-toward-zero* mode.
- In the case of sign-magnitude or unsigned binary numbers, the actions of truncation are identical to those of the *round-toward-zero* mode. By comparison, in the case of signed binary numbers, the actions of truncation are identical to those of the *round-floor* mode.
- Finally, this entire appendix was spawned from just a single topic in a book I coauthored called *How Computers Do Math* (ISBN: 0471732788). If you've found these discussions to be useful and interesting, just imagine how much fun you'd have reading the book!

APPENDIX I

An Interesting Conundrum

INVERTING THREE SIGNALS USING ONLY TWO NOT GATES

As we discussed in *Chapter 9: Boolean Algebra*, if you are new to Boolean algebra, you may well feel that it's horribly complicated. On this basis, you may be surprised to hear that engineers actually enjoy posing (and solving) logical problems. For example, here's an interesting conundrum that should get the old brain cells "firing on all six cylinders." Consider a "black box" with three inputs (A, B, C) and three outputs ($\text{not}A, \text{not}B, \text{not}C$) as illustrated in Figure I.1.

455

The $\text{not}A$ output is the logical negation of the A input (if A is 0, then $\text{not}A$ will be 1; if A is 1, then $\text{not}A$ will be 0). Similarly, the $\text{not}B$ and $\text{not}C$ outputs are the logical negations of the B and C inputs, respectively.

So, our task is to implement the contents of the black box. Now, this task would be trivial if we could use any primitive logic gates we so-desired. For example, we could simply use three NOT gates as illustrated in Figure I.2.

FIGURE I.1

A block box with three inputs and three outputs.

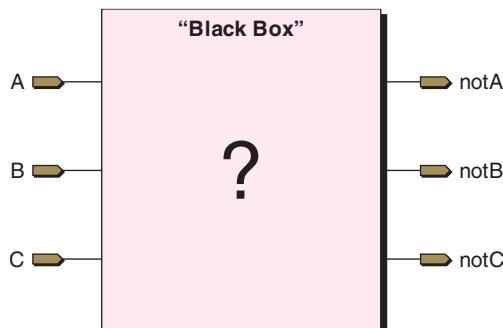
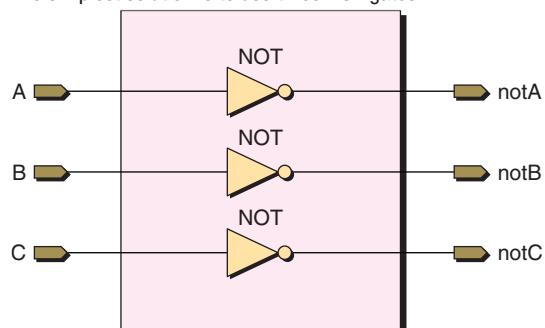


FIGURE I.2

The simplest solution is to use three NOT gates.



Alternatively, and for no other reason than to show that there are usually lots of different ways of realizing things in logic, we could implement the contents of our black box using three XOR gates, as illustrated in Figure I.3.

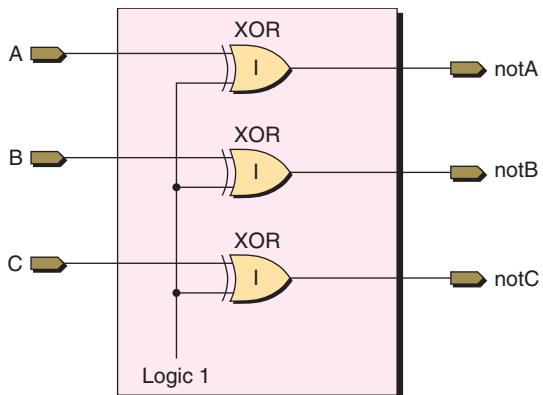


FIGURE I.3

An alternative solution using three XOR gates.

But here's the problem; for the purposes of this conundrum, although we are allowed to use as many AND and OR gates as we wish, we can use *only two* NOT gates, and we *can't use any* NAND, NOR, XOR, or XNOR gates.

As fate would have it, I once posed this problem in a blog. One of my friends soon sent me an e-mail saying: "*I don't think it can be done.*" This was followed a little later by a message saying: "*I stand corrected, I have the glimpse of an idea as to how it might be done.*" And, the following morning (after a sleepless night on his part), a triumphant e-mail arrived saying: "*It can be done, and this is how you do it!*" My friend's solution was as follows:

Inputs:

A, B, C;

Outputs:

notA, notB, notC;

Internal Nodes:

```

2Or3Ones = ((A & B) | (A & C) | (B & C));
0Or1Ones = !(2Or3Ones);
1One = 0Or1Ones & (A|B|C);
1Or3Ones = 1One | (A & B & C);
0Or2Ones = !(1Or3Ones);
0Ones = 0Or2Ones & 0Or1Ones;
2Ones = 0Or2Ones & 2Or3Ones;
  
```

Equations for Outputs:

```
notA = 0Ones | (1One & (B|C)) | (2Ones & (B & C));  
notB = 0Ones | (1One & (A|C)) | (2Ones & (A & C));  
notC = 0Ones | (1One & (A|B)) | (2Ones & (A & B));
```

Remember that we're using "!" to represent a NOT gate, "&" to represent an AND gate, and "|" to represent an OR gate.

You have to admit that this is really jolly clever. What my friend did was to create a suite of internal signals that tell him whether he has zero 1s, one 1, two 1s, three 1s, and so forth (as we see, this solution uses only AND, OR, and two NOT gates). We can then combine these signals with the input values to generate the output equations. I LOVE this solution.

This page intentionally left blank

APPENDIX J

A No-Holds Barred Seafood Gumbo

DOWNHILL MUD-WRESTLING

Well, my little avocado pears, festooned with slivers of spring onions and deluged with lashings of spicy Italian dressing ... here we are at the end of the book (sob sob).

Having absorbed the myriad juicy snippets of information that are strewn so lavishly throughout *Bebop to the Boolean Boogie*, you are now uniquely qualified to regale an assembled throng with a discourse on almost any subject under the sun. So, this would be the perfect time to rest our weary brains and turn our attention to other facets of life's rich tapestry. For example, as soon as I've finished penning these last few words, I must return to practicing my downhill mud-wrestling skills as, I have no doubt, so do you.

459

Beware! If you are under 21, male, or a politician, don't attempt to do anything on the culinary front without your mother's permission and supervision, because kitchens contain sharp things, hot things, and a wide variety of other potentially dangerous things.

But first we need to recline, relax, lay back, unwind, and take the time to recharge our batteries—and what could be more appropriate than a steaming bowl of no-holds-barred seafood gumbo? So, here's the recipe for a fulsomely flavored Epicurean taste-fest sensation sufficient to cause the most sophisticated of gastronomes to start salivating surreptitiously and to make a grown man break down and cry.

This pert little beauty will pulsate promiscuously across your pallet and pummel it with a passion; titillate your taste buds and have them tap-dancing the tango on your tongue; reverberate and resonate resoundingly throughout your nervous

system, and warm the cockles of your heart. In short, this frisky little number will grab you by the short-and-curly, swing you sybaritically around the room in a syncopated symphony of delight, and leave you groveling on your knees, gnashing your teeth, and gasping for more.

INGREDIENTS

The following ingredients are for the main body of the gumbo—you'll have to sort out any rice, bread, and side dishes for yourself (this isn't a cheap dish, by the way, but it's well worth the expense for the acclaim¹ you will win from your family and friends):

- 2 cups of diced onions
- 1½ cups of diced green bell peppers
- 1 cup of diced celery
- 2 cups of halved button mushrooms
- 3 large cloves of finely diced garlic
- 1 finely diced scotch bonnet or habañero pepper
- 10 thick-cut slices of bacon
- 1 pound of Cajun-style sausage
- 1 pound of uncooked, peeled, medium-sized shrimp
- ½ pound of scallops
- ¾ pound white fish cut into slices
- 1 small tin of anchovies
- 2 bay leaves
- ½ teaspoon dried thyme leaves
- ¼ teaspoon dried oregano
- ½ teaspoons of salt
- 1½ teaspoons of white pepper
- ½ teaspoon of black pepper
- ½ teaspoon of cayenne pepper
- 2 teaspoons of Gumbo Filé
- 5½ cups of chicken stock
- Lots of butter (more below)
- ¾ cup flour

If you can't get Cajun-style sausage, then Polish sausage will do nicely. Note that the teaspoon quantities in the list of ingredients do not refer to level measures,

¹I once won a chili cook-off competition with this recipe; it was so good they changed the rules to let me win; I kid you not!

nor should you attempt to set a new world record for the amount you can balance on one spoon—just try to aim for roughly the same sensuously rounded profile you get when you’re casually spooning sugar into a cup of coffee. So, without further ado, let’s gird up our loins and proceed to the fray ...

STEP-BY-STEP INSTRUCTIONS

1. First of all there’s an art to cooking, and it starts by doing the washing up you’ve been putting off all day, and putting all of the pots away.
2. Grill (broil) the bacon until it’s crispy and crunchy; then put it on a plate to cool and set it to one side.
3. Prepare all of the vegetables, mushrooms, garlic, and scotch bonnet or habañero pepper. (Be careful with the latter—it’s best to wear gloves here—because these are ferociously hot and if you get the juice on your finger and then touch anywhere near your eyes you’re going to be a very unhappy camper.) Put them all in separate bowls, except for the scotch bonnet and garlic, which can go together.
4. Chop the Cajun-style sausage into $\frac{1}{4}$ -inch pieces (cut at about a 45-degree angle because this looks nicer); put them in a bowl and set it to one side.
5. Mix the salt, thyme, oregano, Gumbo Filé, and the white, black, and cayenne peppers together in a cup and set it to one side (you’ll need your hands free later).
6. Wash up all the knives, chopping boards, and everything else you’ve used and put them all away, and then wipe down all of your working surfaces. Trust me—you’ll feel better when everything is clean and tidy. (Have I ever lied to you before?) Take a five-minute break and (assuming you are of legal age) quaff² a glass of wine—after all, who deserves it more than you?
7. Put the chicken stock into a large saucepan and bring it to a boil. Then reduce the heat to a low, slow simmer and leave it on the back burner.
8. Using a medium- to medium-high heat, melt $\frac{3}{4}$ of a cup of butter in a large, heavy skillet until it starts to bubble. Gradually add the flour using a whisk and stir constantly until the resulting roux is a darkish, reddish brown (the darker the better—try to be brave here and leave it longer than you expect). Remove the skillet from the heat, but keep on stirring until it’s cooled down enough so that the mixture won’t stick and burn.

²Quaffing is just like regular drinking, except that you do it with gusto and abandon and you tend to spill more down your chest.

9. Maintain the stock at a low simmer and add the mixture that you've just made, stirring it in one spoonful at a time and waiting for each spoonful to dissolve before adding the next.
10. Clean the skillet, put it on a medium-high to high heat, and melt a chunk of butter. Stirring all the time, sauté the celery for one minute, add the bell peppers and sauté for 1½ minutes, add the onions and sauté for 1½ minutes, then add the scotch bonnets and garlic along with the mixture of herbs, salt, and pepper, and sauté for one more minute. Finally, chuck the whole lot into the saucepan with the stock.
11. Break the bacon into ½-inch pieces and toss them into the stock. Flake the anchovies with a fork and cast them into the stock. Hurl in the Cajun-style sausage and the bay leaves. Also, if you happen to have any lying around, add a couple of teaspoons of English Worcestershire sauce. Cover the saucepan and leave on a low simmer.
12. After about half an hour, return the skillet to a medium-high to high heat and melt another chunk of butter. Sauté the mushrooms until they're golden brown and squealing for more, then use them to swell the contents of the saucepan.
13. Simmer the whole mixture (stirring often) for at least another half hour, which, by some strange quirk of fate, will give you all the time you need to wash the skillet and the dishes you used and put them away again. If you're ravenous, you can proceed immediately to the next step; if you're wise, however, you'll remove the heat, let the contents cool, put the saucepan in the fridge, and leave your cunningly captivating creation to stand overnight (chilies, stews, curries, and gumbos always taste better if the ingredients have the time to formally introduce themselves). The next day, when you're ready to chow down, return the saucepan to the stove, bring the contents to a low simmer, and proceed to the next step.
14. Chop the scallops and fish into bite-sized chunks. Shortly before you're ready to eat, add the shrimp, scallops, and fish to the stock, bring everything to a boil, and then return it to a simmer. Maintain the simmer until the seafood is cooked (I personally opt for around 10 minutes) and you're ready to rock and roll.

SERVING YOUR GORGEOUS GOURMET GUMBO

This little beauty will put hairs on your chest, make them curl, and then take them off again. Seriously, this gumbo really is seductively, scintillatingly, and

(with the addition of the habañero pepper) scorchingly tasty—your guests will be singing your praises and tap-dancing in the streets.

You can serve your gorgeous gourmet gumbo over steamed or boiled rice, with crusty French bread, or with whatever else your heart desires. The quantities given above will serve eight to ten manly-man sized portions with a little something left over for the following day.

Of course, no meal would be complete without some wine—and the perfect complement to your scrumptious repast is ... to be found in a very large bottle. Enjoy!

This page intentionally left blank

Glossary

µC

See Microcontroller.

µP

See Microprocessor.

Absolute Scale of Temperature

A scale of temperature that was invented by the British mathematician and physicist William Thomas, first Baron of Kelvin. Under the *absolute*, or *Kelvin*, scale of temperature, 0K (corresponding to -273°C) is the coldest possible temperature and is known as *Absolute Zero*.

465

Absolute Zero

See Absolute Scale of Temperature.

Active-High

Beware, here be dragons. In this book an *active-high* signal is defined as a signal whose active state is considered to be a logic 1. This definition, which is used by the engineers in the trenches, allows all forms of logic, including *assertion-level* logic, to be represented without any confusion, regardless of whether *positive-* or *negative-logic* implementations are employed.

Unfortunately, some academics (and even textbooks) define an active-high signal as being one whose asserted (TRUE or logic 1) state is at a higher voltage level than its unasserted (FALSE or logic 0) state. However, this definition serves only to cause confusion when combined with negative-logic implementations. Thus, when using the term active-high in discussions with other people, you are strongly advised to make sure that you're all talking about the same thing before you start.

Active-Low

In this book an *active-low* signal is defined as a signal whose active state is considered to be a logic 0. This definition, which is used by the engineers in the trenches, allows all forms of logic, including *assertion-level logic*, to be represented without any confusion, regardless of whether *positive-* or *negative-logic* implementations are employed.

Unfortunately, some academics (and even textbooks) define an active-low signal as being one whose asserted (TRUE or logic 1) state is at a lower voltage level than its unasserted (FALSE or logic 0) state. However, this definition serves only to cause confusion when combined with negative-logic implementations. Thus, when using the term active-low in discussions with other people, you are strongly advised to make sure that you're all talking about the same thing before you start.

Active Substrate

A hybrid or *System-in-Package* (SiP) substrate formed from a semiconductor. Termed *active* because components such as transistors can be fabricated directly into the substrate.

Active Trimming

The process of trimming components such as resistors while the circuit is under power. Such components are fabricated directly onto the substrate of a hybrid or *System-in-Package* (SiP), and the trimming is usually performed using a laser beam.

Actuator

A transducer that converts an electronic signal into a physical equivalent. For example, a loudspeaker is an actuator that converts electronic signals into corresponding sounds.

A/D (Analog-to-Digital)

The process of converting an analog value into its digital equivalent.

Additive Process

A process in which conducting material is added to specific areas of a substrate. Groups of tracks, individual tracks, or portions of tracks can be built up to precise thicknesses by iterating the process multiple times with selective masking.

Address Bus

A unidirectional set of signals used by a computer to point to memory locations in which it is interested.

Analog

A continuous value that most closely resembles the real world and can be as precise as the measuring technique allows.

Analog Circuit

A collection of components used to process or generate analog signals.

Analog-to-Digital

See A/D.

Analogue

The way they spell “analog” in England.

Anisotropic Adhesives

Special adhesives that contain minute particles of conductive material. These adhesives find particular application with the flipped-chip techniques used to mount bare die on the substrates of hybrids, *System-in-Package* (SiP) assemblies, or circuit boards. The conducting particles are only brought in contact with each other at the sites where the raised pads on the die are pressed down over their corresponding pads on the substrate, thereby forming good electrical connections between the pads.

Anti-Fuse Technology

A programmable logic device technology in which conducting paths (*anti-fuses*) are “grown” by applying signals of relatively high voltage and current to the device’s inputs.

Anti-Pad

The area of copper etched away around a via or a plated through-hole on a power or ground plane, thereby preventing an electrical connection being made to that plane.

Application-Specific Integrated Circuit

See ASIC.

Application-Specific Standard Part

See ASSP.

ASIC (Application-Specific Integrated Circuit)

A device whose function is determined by a designer for a particular application or group of applications.

ASIC Cell

A logic function in the cell library defined by the manufacturer of an application-specific integrated circuit.

Assertion-Level Logic

A technique used to draw symbols that more precisely represent the function of logic gates with active-low inputs.

Associative Rules

Algebraic rules that state that the order in which pairs of variables are associated together will not affect the result of an operation; for example,¹ $(a \& b) \& c \equiv a \& (b \& c)$.

ASSP (Application-Specific Standard Part)

Refers to complex integrated circuits created by a device manufacturer using ASIC technologies, where these components are to be sold as standard parts to anybody who wants to buy them.

Asynchronous

A signal whose data is acknowledged or acted upon immediately, regardless of any clock signal.

Atto

Unit qualifier (symbol = a) representing one-millionth of one-millionth of one-millionth, or 10^{-18} . For example, "3 as" stands for 3×10^{-18} seconds.

Backplane

The medium used to interconnect a number of circuit boards. Typically refers to a special, heavy-duty printed circuit board.

Ball Grid Array

See BGA.

Bare Die

An unpackaged integrated circuit.

Barrier Layer

See Overglassing.

Base

Refers to the number of digits in a numbering system. For example, the decimal numbering system is said to be base-10. (May also be referred to as the "radix.")

¹Note that the symbol \equiv indicates "is equivalent to" or "is the same as."

Basic Cell

A predefined group of unconnected components that is replicated across the surface of a gate array form of ASIC.

BDD (Binary Decision Diagram)

A method of representing Boolean equations as decision trees.

Bebop

A form of music characterized by fast tempos and agitated rhythms that became highly popular in the decade following World War II.

BEDO (Burst EDO)

An asynchronous form of DRAM-based computer memory that was popular for a while in the latter half of the 1990s. Along with other asynchronous memory techniques, BEDO was eventually superseded by SDRAM technologies. *See also FPM, EDO, and SDRAM.*

BGA (Ball Grid Array)

A packaging technology similar to a *Pad Grid Array (PGA)*, in which a device's external connections are arranged as an array of conducting pads on the base of the package. However, in the case of a ball grid array, small balls of solder are attached to the conducting pads.

BiCMOS (Bipolar-CMOS)

A technology in which the function of each logic gate is implemented using low-power CMOS, while the output stage is implemented using high-drive bipolar transistors.

Binary

Base-2 numbering system (the type of numbers that computers use internally).

Binary Decision Diagram

See BDD.

Binary Digit

A numeral in the binary scale of notation. A binary digit (typically abbreviated to "bit") can adopt one of two values: 0 or 1.

Binary Encoding

A form of state assignment for state machines that requires the minimum number of state variables.

Binary Logic

Digital logic gates based on two distinct voltage levels. The two voltages are used to represent the binary values 0 and 1, and their logical equivalents FALSE and TRUE.

BiNMOS (Bipolar NMOS)

A relatively new low-voltage integrated circuit technology in which complex combinations of bipolar and NMOS transistors are used for sophisticated output stages providing both high speed and low static power dissipation.

Bipolar Junction Transistor

See BJT.

Bi-quinary

A system that utilizes two bases, base-2 and base-5, to represent decimal numbers. Each decimal digit is represented by the sum of two parts, one of which has the value of decimal zero or five, and the other the values of zero through four. The abacus is one practical example of the use of a bi-quinary system.

BIST (Built-in Self-test)

A test strategy in which additional logic is built into a component, thereby allowing it to test itself.

Bit

Abbreviation of *binary digit*. A binary digit can adopt one of two values: 0 or 1.

BJTs (Bipolar Junction Transistors)

A family of transistors.

Blind Via

A via that is only visible from one side of the substrate.

Bobble

A small circle used on the inputs to a logic gate symbol to indicate an active low input or control, or on the outputs to indicate a negation (inversion) or complementary signal. Some engineers prefer to use the term “bubble.”

Boolean Algebra

A mathematical way of representing logical expressions.

Bootstrapping

A sequence of initialization operations performed by a computer when it is first powered up.

Braze

To unite or fuse two pieces of metal by heating them in conjunction with a hard solder with a high melting point.

Buckyballs

Prior to the mid-1980s, the only major forms of pure carbon known to us were graphite and diamond. In 1985, however, a third form consisting of spheres formed from 60 carbon atoms (C_{60}) was discovered. Officially known as Buckminsterfullerene²—named after the American architect R. Buckminster Fuller who designed geodesic domes with the same fundamental symmetry—these spheres are more commonly known as “buckyballs.” In 2000, scientists with the U.S. Department of Energy’s Lawrence Berkeley National Laboratory (Berkeley Lab) and the University of California at Berkeley reported that they had managed to fashion a transistor from a single buckyball.

Built-in Self-Test

See BIST.

Bulk Storage

Refers to some form of media, typically magnetic, such as tape or a disk drive, which can be used to store large quantities of information relatively inexpensively.

Buried Via

A via used to link conducting layers internal to a substrate. Such a via is not visible from either side of the substrate.

Burst EDO

See BEDO.

Bus

A set of signals performing a common function and carrying similar data. Typically represented using vector notation; for example, $\text{data}[7:0]$.

Bundle

A set of signals related in some way that makes it appropriate to group them together for ease of representation or manipulation. May contain both scalar and vector elements; for example, $\{a,b,c,d[5:0]\}$.

Byte

A group of eight binary digits, or bits.

²Science magazine voted Buckminsterfullerene the “Molecule of the Year” in 1991.

Cache Memory

A small, high-speed, memory (usually SRAM) used to buffer the central processing unit from any slower, lower cost memory devices such as DRAM. The high-speed cache memory is used to store active instructions and data,³ while the bulk of the instructions and data reside in the slower memory.

Canonical Form

In a mathematical context this term is taken to mean a generic or basic representation. Canonical forms provide the means to compare two expressions without falling into the trap of trying to compare “apples” with “oranges.”

Capacitance

A measure of the ability of two adjacent conductors separated by an insulator to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads.

Carbon Nanotube FET

See CNFET.

Catalyst

A substance that initiates a chemical reaction under different conditions (such as lower temperatures) than would otherwise be possible. The catalyst itself remains unchanged at the end of the reaction.

Cell

See ASIC Cell, Basic Cell, Cell Library, and Memory Cell.

Cell Library

The collective name for the set of logic functions defined by the manufacturer of an application-specific integrated circuit. The designer decides which types of cells should be realized and connected together to make the device perform its desired function.

Central Processing Unit

See CPU.

Ceramic

An inorganic, nonmetallic material, such as alumina, beryllia, steatite, or forsterite, which is fired at a high temperature and is often used in electronics as a substrate or to create component packages.

³In this context, “active” refers to data or instructions that a program is currently using, or which the operating system believes that the program will want to use in the immediate future.

CGA (Column Grid Array)

A packaging technology similar to a *Pad Grid Array (PGA)*, in which a device's external connections are arranged as an array of conducting pads on the base of the package. However, in the case of a column grid array, small columns of solder are attached to the conducting pads.

CGH (Computer-Generated Hologram)

In the context of this book, this refers to a slice of quartz or similar material into which three-dimensional patterns are cut using a laser. The angles of the patterns cut into the quartz are precisely calculated for use in the optical communication strategy known as holographic interconnect. All of these calculations are performed by a computer, and the laser used to cut the three-dimensional patterns into the quartz is also controlled by a computer. Thus, the slice of quartz is referred to as a *computer-generated hologram*.

Channel

(1) The area between two arrays of basic cells in a channeled gate array. (2) The gap between the source and drain regions in a MOS transistor.

Channeled Gate Array

An *Application-Specific Integrated Circuit (ASIC)* organized as arrays of basic cells. The areas between the arrays are known as channels.

Channel-Less Gate Array

An *Application-Specific Integrated Circuit (ASIC)* organized as a single large array of basic cells. May also be referred to as a "sea of cells" or a "sea of gates" device.

Checksum

The final cyclic-redundancy check value stored in a linear feedback shift register (or software equivalent). Also known as a "signature" in the guided-probe variant of a functional test.

Chemically Amplified Resist

In the case of a chemically amplified resist, the application of a relatively small quantity of ultraviolet light stimulates the formation of chemicals in the resist, where these chemicals accelerate the degrading process. This reduces the amount of ultraviolet light required to degrade the resist and allows the creation of finer features with improved accuracy.

Chemical Mechanical Polishing

See CMP.

Chemical Vapor Deposition

See CVD.

Chemical Vapor Infiltration

See CVI.

Chip

Popular name for an integrated circuit.

Chip-on-Board

See COB.

Chip-on-Chip

See COC.

Chip-on-Flex

See COF.

Chip Scale Package

See CSP.

Circuit Board

The generic name for a wide variety of interconnection techniques, which include rigid, flexible, and rigid-flex boards in single-sided, double-sided, and multilayer configurations.

CMOS (Complementary Metal Oxide Semiconductor)

Logic gates constructed using both NMOS and PMOS transistors connected in a complementary manner.

CMP (Chemical Mechanical Polishing)

A process used to re-planarize a wafer—smoothing the surface by polishing out the “bumps” caused by adding a metallization (tracking) layer.

CNFET (Carbon Nanotube FET)

A type of transistor announced by IBM in the summer of 2002. This device is based on a field-effect transistor featuring a carbon nanotube measuring 1.4 nanometers in diameter as the channel (the rest of the transistor is formed using conventional silicon and metal processing technologies). It will be a number of years before these devices become commercially available, but they are anticipated to significantly outperform their silicon-only counterparts.

Coaxial Cable

A conductor in the form of a central wire surrounded first by a dielectric (insulating) layer, and then by a conducting tube that serves to shield the central wire from external interference.

COB (Chip-on-Board)

A process in which unpackaged integrated circuits are physically and electrically attached to a circuit board, and are then encapsulated with a “glob” of protective material such as epoxy.

COC (Chip-on-Chip)

A process in which unpackaged integrated circuits are mounted on top of each other. Each die is very thin and it is possible to have over a hundred die forming a 3-D cube.

Coefficient of Thermal Expansion

Defines the amount that a material expands and contracts due to changes in temperature. If materials with different coefficients of thermal expansion are bonded together, changes in temperature will cause shear forces at the interface between them.

COF (Chip-on-Flex)

Similar to *Chip-on-Board*, except that the unpackaged integrated circuits are attached to a flexible printed circuit.

Cofired Ceramic

A substrate formed from multiple layers of “green” ceramic that are bonded together and fired at the same time.

Column Grid Array

See CCA.

Combinational

A digital function whose output value is directly related to the current combination of values on its inputs.

Combinatorial

See Combinational.

Commutative Rules

Algebraic rules that state that the order in which variables are specified will not affect the result of an operation: for example,⁴ $a \& b \equiv b \& a$.

⁴Note that the symbol \equiv indicates “is equivalent to” or “is the same as.”

Comparator (digital)

A logic function that compares two binary values and outputs the results in terms of binary signals representing *less-than* and/or *equal-to* and/or *greater than*.

Compiled Cell Technology

A technique used to create portions of a standard cell *Application-Specific Integrated Circuit (ASIC)*. The masks used to create components and interconnections are directly generated from Boolean representations using a program called a *silicon compiler*. May also be used to create data-path and memory functions.

Complementary Output

Refers to a function with two outputs carrying complementary logical values. One output is referred to as the *true output* and the other as the *complementary output*.

Complementary Rules

Rules in Boolean Algebra derived from the combination of a single variable with the inverse of itself.

Complex Programmable Logic Device

See CPLD.

Computer-Generated Hologram

See CGH.

Computer Virus

There are many different types of computer viruses but, for the purposes of this book, a computer virus may be defined as a self-replicating program released into a computer system for a number of purposes. These purposes range from the simply mischievous, such as displaying humorous or annoying messages, to the downright nefarious, such as corrupting data or destroying (or subverting) the operating system.

Conditioning

See Signal Conditioning.

Conductive Ink Technology

A technique in which tracks are screen-printed directly onto the surface of a circuit board using conductive ink.

Conjunction

Propositions combined with an AND operator: for example, "You have a parrot on your head AND you have a fish in your ear." The result of a conjunction is true if all the propositions comprising that conjunction are true.

CPLD (Complex PLD)

A device that contains a number of PLA or PAL functions sharing a common programmable interconnection matrix.

CPU (Central Processing Unit)

The “brain” of a computer, where all of the decision making and number crunching is performed.

CRC (Cyclic Redundancy Check)

A calculation used to detect errors in data communications, typically performed using a linear feedback shift register. Similar calculations may be used for a variety of other purposes such as data compression.

CSIC (Customer-Specific Integrated Circuit)

An alternative and possibly more accurate name for an ASIC, but this term is rarely used in the industry and shows little indication of finding favor with the masses.

CSP (Chip Scale Package)

An integrated circuit packaging technique in which the package is only fractionally larger than the silicon die.

CSSP (Customer-Specific Standard Product)

A silicon chip that combines traditional FPGA programmable fabric with collections of hard macros; these macros include functions that can interface to different types of external memory devices, provide various communications functions, the ability to drive display devices, and so forth.

Cure

To harden a material using heat, ultraviolet light, or some other process.

Cured

Refers to a material that has been hardened using heat, ultraviolet light or some other process.

Customer-Specific Integrated Circuit

See CSIC.

Customer-Specific Standard Product

See CSSP.

CVD (Chemical Vapor Deposition)

A process for growing thin films on a substrate, in which a gas containing the required molecules is converted into *plasma* by heating it to extremely high temperatures using microwaves. The plasma carries atoms to the surface of

the substrate where they are attracted to the crystalline structure of the substrate. This underlying structure acts as a template. The new atoms continue to develop the structure to build up a layer on the substrate's surface.

CVI (Chemical Vapor Infiltration)

A process similar to *Chemical Vapor Deposition (CVD)* but, in this case, the process commences by placing a crystalline powder of the required substance in a mold. Additionally, thin posts, or columns, can be preformed in the mold, and the powder can be deposited around them. When exposed to the same plasma as used in the CVD technique, the powder coalesces into a polycrystalline mass. After the CVI process has been performed, the posts can be dissolved, leaving holes through the crystal for use in creating vias. CVI processes can produce layers twice the thickness of those obtained using CVD techniques at a fraction of the cost.

Cyclic Redundancy Check

See CRC.

D/A (Digital-to-Analog)

The process of converting a digital value into its analog equivalent.

Data Bus

A bidirectional set of signals used by a computer to convey information from a memory location to the central processing unit and vice versa. More generally, a set of signals used to convey data between digital functions.

Data-Path Function

A well-defined function such as an adder, counter, or multiplier used to process digital data.

Daughter Board

If a backplane contains active components such as integrated circuits, then it is usually referred to as a motherboard. In this case, the boards plugged into it are referred to as daughter boards.

DDR (Double Data Rate)

A modern form of SDRAM-based memory. The original SDRAM specification was based on using one of the clock edges only (say the rising edge) to read/write data out-of/into the memory. DDR refers to memory designed in such a way that data can be read/written on both edges of the clock. This effectively doubles the amount of data that can be pushed through the system without increasing the clock frequency (like many things this sounds simple if you say it fast, but making this work is trickier than it may at first appear). *See also QDR and SDRAM.*

Decimal

Base-10 numbering system.

Decoder (digital)

A logic function that uses a binary value, or address, to select between a number of outputs and to assert the selected output by placing it in its active state.

Deep Sub-Micron

See DSM.

Delamination

Occurs when a composite material formed from a number of layers is stressed, thermally or otherwise, such that the layers begin to separate.

DeMorgan Transformation

The transformation of a Boolean expression into an alternate, and often more convenient, form.

Die

(1) An unpackaged integrated circuit. In this case, the plural of die is also die (in much the same way that “a shoal of herring” is the plural of “herring”). (2) A piece of metal with a design engraved or embossed on it for stamping onto another material, upon which the design appears in relief.

Dielectric Layer

(1) An insulating layer used to separate two signal layers. (2) An insulating layer used to modify the electrical characteristics of a substrate.

Die Separation

The process of separating individual die from the wafer by marking the wafer with a diamond scribe and fracturing it along the scribed lines.

Die Stacking

A technique used in specialist applications in which several bare die are stacked on top of each other to form a sandwich. The die are connected together and then packaged as a single entity.

Diffusion Layer

The surface layer of a piece of semiconductor into which impurities are diffused to form P-type and N-type material. In addition to forming components, the diffusion layer may also be used to create embedded traces.

Digital

A value represented as being in one of a finite number of discrete states called *quanta*. The accuracy of a digital value is dependent on the number of quanta used to represent it.

Digital Circuit

A collection of logic gates used to process or generate digital signals.

Digital Signal Processing/Processor

See DSP.

Digital-to-Analog

See D/A.

DIMM (Dual Inline Memory Module)

A single memory integrated circuit can only contain a limited amount of data, so a number are gathered together onto a small circuit board called a *memory module*. Each memory module has a line of gold-plated pads on both sides of one edge of the board. These pads plug into a corresponding connector on the main computer board. In the case of a *Dual Inline Memory Module (DIMM)*, the pads on opposite sides of the board are electrically isolated from each other and form two separate contacts. *See also SIMM and RIMM.*

Diode

A two-terminal device that conducts electricity in only one direction; in the other direction it behaves like an open switch. These days the term diode is almost invariably taken to refer to a semiconductor device, although alternative implementations such as vacuum tubes are available.

Diode-Transistor Logic

See DTL.

Discrete Device

Typically taken to refer to an electronic component such as a resistor, capacitor, diode, or transistor that is presented in an individual package. More rarely, the term may be used in connection with a simple integrated circuit containing a small number of primitive gates.

Disjunction

Propositions combined with an OR operator; for example, "You have a parrot on your head OR you have a fish in your ear." The result of a disjunction is true if at least one of the propositions comprising that disjunction is true.

Distributive Rules

Two very important rules in Boolean Algebra. The first states that the AND operator distributes over the OR operator: for example,⁵ $a \& (b | c) \equiv (a \& b) | (a \& c)$. The second states that the OR operator distributes over the AND operator: for example, $a | (b \& c) \equiv (a | b) \& (a | c)$.

Doping

The process of inserting selected impurities into a semiconductor to create P-type or N-type material.

Double Data Rate

See DDR.

Double-Sided

A printed circuit board with tracks on both sides.

DRAM (Dynamic RAM)

A memory device in which each cell is formed from a transistor-capacitor pair. Called *dynamic* because the capacitor loses its charge over time, and each cell must be periodically recharged if it is to retain its data.

DSM (Deep Submicron)

Typically taken to refer to integrated circuits containing structures that are smaller than 0.5 microns (one-half of one-millionth of a meter).

DSP (Digital Signal Processing)

Perhaps not surprisingly, the term *Digital Signal Processing* (DSP) refers to processing data (signals) in the digital domain. This processing can be performed using a general-purpose microprocessor, a special-purpose *Digital Signal Processor* (DSP), or in a *Field Programmable Gate Array* (FPGA). In this latter case, the FPGA can be configured (programmed) to perform multiple operations in parallel, which allows it to perform the digital signal processing task extremely quickly.

DSP (Digital Signal Processor)

A special form of microprocessor that has been designed to perform a specific processing task on a specific type of digital data much faster and more efficiently than can be achieved using a general-purpose microprocessor.

DTL (Diode-Transistor Logic)

Logic gates implemented using particular configurations of diodes and bipolar junction transistors. For the majority of today's designers, diode-transistor logic is of historical interest only.

⁵Note that the symbol \equiv indicates "is equivalent to" or "is the same as."

Dual Inline Memory Module

See DIMM.

Duo-decimal

Base-12 numbering system.

Dynamic Flex

A type of *flexible printed circuit* which is used in applications that are required to undergo constant flexing such as ribbon cables in printers.

Dynamically Reconfigurable Hardware

A product whose function may be customized on-the-fly while remaining resident in the system.

Dynamic RAM

See DRAM.

EBE (Electron Beam Epitaxy)

A technique for creating thin films on substrates in precise patterns. The substrate is first coated with a layer of dopant material before being placed in a high vacuum. A guided beam of electrons is fired at the substrate, causing the dopant to be driven into it, effectively allowing molecular-thin layers to be “painted” onto the substrate where required.

ECC (Error-Correcting Code)

Computer systems are very complicated and there's always the chance that an error will occur when reading or writing to the memory (a stray pulse of “noise” may flip a logic 0 to a logic 1 while your back is turned). Thus, serious computers use ECC memory, which includes extra bits and special circuitry that tests the accuracy of data as it passes in and out of memory and corrects any (simple) errors.

ECL (Emitter-Coupled Logic)

Logic gates implemented using particular configurations of bipolar junction transistors.

Edge-Sensitive

An input that only affects a function when it transitions from one logic value to another.

EDO (Extended Data Out)

An asynchronous form of DRAM-based computer memory that was popular for a while in the latter half of the 1990s. Along with other asynchronous memory

techniques, EDO was eventually superseded by SDRAM technologies. *See also FPM, BEDO, and SDRAM.*

EEPROM or E²PROM (Electrically-Erasable Programmable Read-Only Memory)

A memory device whose contents can be electrically programmed by the designer. Additionally, the contents can be electrically erased, allowing the device to be reprogrammed.

Electrically-Erasable Programmable Read-Only Memory

See EEPROM.

Electromigration

(1) A process in which structures on an integrated circuit's *substrate* (particularly structures in deep sub-micron technologies) are eroded by the flow of electrons in much the same way as a river erodes land. (2) The process of forming transistor-like regions in a semiconductor using an intense magnetic field.

Electron Beam Epitaxy

See EBE.

Electron Beam Lithography

An integrated circuit fabrication process in which fine beams of electrons are used to draw extremely high-resolution patterns directly into the resist without the use of a mask.

Electrostatic Discharge

See ESD.

Emitter-Coupled Logic

See ECL.

Enzyme

One of numerous complex proteins that are produced by living cells and catalyze biochemical reactions at body temperatures.

EPROM (Erasable Programmable Read-Only Memory)

A memory device whose contents can be electrically programmed by the designer. Additionally, the contents can be erased by exposing the die to ultraviolet light through a quartz window mounted in the top of the component's package.

Equivalent Gate

A concept in which each type of logic function is assigned an equivalent gate value for the purposes of comparing functions and devices. However, the definition of an equivalent gate varies depending on whom you're talking to.

Erasable Programmable Read-Only Memory

See EPROM.

Error-Correcting Code

See ECC.

ESD (Electro-Static Discharge)

This refers to a charged person, or object, discharging static electricity, which can be generated in the process of moving around. Although the current associated with such a static charge is extremely low, the electric potential can be in the millions of volts and can severely damage electronic components. CMOS devices are particularly prone to damage from static electricity.

Etching

The process of selectively removing any material not protected by a resist using an appropriate solvent or acid. In some cases the unwanted material is removed using an electrolytic process.

Eutectic Bond

A bond formed when two pieces of metal, or metal-coated materials, are pressed together and vibrated at ultrasonic frequencies.

Extended Data Out

See EDO.

Falling-Edge

See Negative-Edge.

Fan-In Via

See Fan-Out Via.

Fan-Out Via

In the case of surface mount devices, each component pad is usually connected by a short length of track to a via which forms a link to other conducting layers, and this via is known as a fan-out via. Some engineers attempt to differentiate vias that fall inside the device's footprint (under the body of the device) from vias that fall outside the device's footprint by referring to the former as *fan-in vias*, but this is not a widely used term.

Fast Page Mode

See FPM.

Femto

Unit qualifier (symbol = f) representing one-thousandth of one-millionth of one-millionth, or 10^{-15} . For example, "3 fs" stands for 3×10^{-15} seconds.

FET

A transistor whose control, or gate, signal creates an electromagnetic field, which turns the transistor ON or OFF.

Field-Effect Transistor

See FET.

Field-Programmable Device

See FPD.

Field-Programmable Gate Array

See FPGA.

FIFO (First-In First-Out)

A memory device in which data is read out in the same order that it was written in.

Finite State Machine

See FSM.

Firmware

Refers to programs, or sequences of instructions, that are hard-coded into non-volatile memory devices.

First-In First-Out

See FIFO.

Flash

See Gold Flash.

FLASH Memory

An evolutionary technology that combines the best features of the EPROM and E²PROM technologies. The name FLASH is derived from the technology's fast reprogramming time compared to EPROM.

Flex

See FPC.

Flexible Printed Circuit

See FPC.

Flipped-Chip

A generic name for processes in which unpackaged integrated circuits are “flipped over” and mounted directly onto a *substrate* with their component sides facing the substrate.

Flipped-TAB

A combination of flipped-chip and *Tape-Automated Bonding* (TAB).

Footprint

The area occupied by a device mounted on a *substrate*.

FPC (Flexible Printed Circuit)

A specialist circuit board technology, often abbreviated to “flex,” in which tracks are printed onto flexible materials. There are a number of flavors of flex, including *static flex*, *dynamic flex.*, and *rigid flex*.

FPD (Field Programmable Device)

A generic name that encompasses SPLDs, CPLDs, and FPGAs.

FPGA (Field-Programmable Gate Array)

A programmable logic device that is more versatile than traditional programmable devices such as PALs and PLAs, but less versatile than an *Application-Specific Integrated Circuit* (ASIC). Some field-programmable gate arrays use anti-fuses such as those found in programmable logic devices, while others are based on SRAM equivalents.

FPM (Fast Page Mode)

An asynchronous form of DRAM-based computer memory, which was popular for a while in the latter half of the 1990s. Along with other asynchronous memory techniques, FPM was eventually superseded by SDRAM technologies. *See also EDO, BEDO, and SDRAM.*

FR4

The most commonly used insulating base material for circuit boards. FR4 is made from woven glass fibers that are bonded together with an epoxy. The board is cured using a combination of temperature and pressure, which causes the glass fibers to melt and bond together, thereby giving the board strength and rigidity. The first two characters stand for “Flame Retardant” and you can

count the number of people who know what the “4” stands for on the fingers of one hand. FR4 is technically a form of fiberglass, and some people do refer to these composites as *fiberglass boards* or *fiberglass substrates*, but not often.

Free-Space Optical Interconnect

A form of optical interconnect in which laser diode transmitters communicate directly with photo transistor receivers without employing optical fibers or optical waveguides.

FSM (Finite State Machine)

The actual implementation (in hardware or software) of a function that can be considered to consist of a finite set of states through which it sequences.

Full Custom

An *Application-Specific Integrated Circuit* in which the design engineers have complete control over every mask layer used to fabricate the device. The ASIC vendor does not provide a cell library or prefabricate any components on the substrate.

Functional Test

A test strategy in which signals are applied to a circuit’s inputs, and the resulting signals—which are observed on the circuit’s outputs—are compared to known good values.

Fuse

See Fusible-Link Technology.

Fusible Link Technology

A programmable logic device technology that employs links called *fuses*. Individual fuses can be removed by applying pulses of relatively high voltage and current to the device’s inputs.

Fuzz-Button

A small ball of fibrous gold used in one technique for attaching components such as *System-in-Packages* (SiPs) to circuit boards. Fuzz-buttons are inserted between the pads on the base of the package and their corresponding pads on the board. When the package is forced against the board, the fuzz-buttons compress to form good electrical connections. One of the main advantages of the fuzz-button approach is that it allows broken devices to be quickly removed and replaced. Even though fuzz-button technology would appear to be inherently unreliable, it is used in such devices as missiles, so one can only assume that it is fairly robust.

GaAs (Gallium Arsenide)

A 3:5 valence high-speed semiconductor formed from a mixture of gallium and arsenic. GaAs transistors can switch approximately eight times faster than their silicon equivalents. However, GaAs is hard to work with, which results in GaAs transistors being more expensive than their silicon cousins.

GAL (Generic Array Logic)

A variation on a PAL device from a company called Lattice Semiconductor Corporation.⁶

Gallium Arsenide

See GaAs.

Garbage-In Garbage-Out

See GIGO.

Gate Array

An *Application-Specific Integrated Circuit* in which the manufacturer prefabricates devices containing arrays of unconnected components (transistors and resistors) organized in groups called *basic cells*. The designer specifies the function of the device in terms of cells from the cell library and the connections between them, and the manufacturer then generates the masks used to create the metallization layers.

Generic Array Logic

See GAL.

Geometry

Refers to the size of structures created on an integrated circuit. The structures typically referenced are the width of the tracks and the length of the transistor's channels; the dimensions of other features are derived as ratios of these structures.

Giga

Unit qualifier (symbol = G) representing one thousand million, or 10^9 . For example, 3 GHz stands for 3×10^9 hertz.

GIGO (Garbage-In Garbage-Out)

An electronic engineer's joke, also familiar to the writers of computer programs.

⁶GAL is a registered trademark of Lattice Semiconductor Corporation.

Glue Logic

Simple logic gates used to interface more complex functions.

Gold Flash

An extremely thin layer of gold with a thickness measured on the molecular level, which is either electroplated or chemically plated⁷ onto a surface.

Gray Code

A sequence of binary values in which each pair of adjacent values differs by only a single bit: for example: 00, 01, 11, 10.

Green Ceramic

Unfired, malleable ceramic.

Ground Plane

A conducting layer in, or on, a substrate providing a grounding, or reference, point for components. There may be several ground planes separated by insulating layers.

Guard Condition

A Boolean expression associated with a state transition in a state diagram or state table. The expression must be satisfied for that state transition to be executed.

Guided Probe

A form of functional test in which the operator is guided in the probing of a circuit to isolate a faulty component or track.

Guided Wave

A form of optical interconnect, in which optical waveguides are fabricated directly on the substrate of a *System-in-Package* (SiP). These waveguides can be created using variations on standard opto-lithographic thin-film processes.

Hard Macro (Macro Cell)

A logic function defined by the manufacturer of an *Application-Specific Integrated Circuit* (ASIC). The function is described in terms of the simple functions provided in the cell library and the connections between them. The manufacturer also defines how the cells forming the macro will be assigned to basic cells and the routing of tracks between the basic cells.

⁷That is, using an electroless plating process.

Hardware

Generally understood to refer to any of the physical portions constituting an electronic system, including components, circuit boards, power supplies, cabinets, and monitors.

Hardware Description Language

See HDL.

HDL (Hardware Description Language)

Today's digital integrated circuits can end up containing millions of logic gates, and it simply isn't possible to capture and manage designs of this complexity at the schematic (circuit diagram) level. Thus, as opposed to using schematics, the functionality of a high-end integrated circuit is now captured in textual form using a (HDL). The two most popular HDLs are Verilog and VHDL.

Hertz

See HZ.

Heterojunction

The interface between two regions of dissimilar semiconductor materials. The interface of a heterojunction has naturally occurring electric fields that can be used to accelerate electrons, and transistors created using heterojunctions can switch much faster than their homojunction counterparts of the same size.

Hexadecimal

Base-16 numbering system. Each hexadecimal digit can be directly mapped onto four binary digits, or bits.

High Impedance State

The state of a signal that is not currently being driven by anything. A high-impedance state is indicated by the "Z" character.

Hologram

A three-dimensional image (from the Greek *holos*, meaning "whole" and *gram*, meaning "message").

Holography

The art of creating three-dimensional images known as holograms.

Homojunction

An interface between two regions of semiconductor having the same basic composition but opposing types of doping. Homojunctions dominate current processes because they are easier to fabricate than their *heterojunction* cousins.

Hybrid

An electronic subsystem in which a number of integrated circuits (packaged and/or unpackaged) and discrete components are attached directly to a common substrate. Connections between the components are formed on the surface of the substrate, and some components such as resistors and inductors may also be fabricated directly onto the substrate.

Hydrogen Bond

The electrons in a water molecule are not distributed equally, because the oxygen atom is a bigger, more robust fellow that grabs more than its fair share. The end result is that the oxygen atom has an overall negative charge, while the two hydrogen atoms are left feeling somewhat on the positive side. This unequal distribution of charge means that the hydrogen atoms are attracted to anything with a negative bias: for example, the oxygen atom of another water molecule. The resulting bond is known as a hydrogen bond.

Hz (hertz)

Unit of frequency. One hertz equals one cycle—or one oscillation—per second.

IC (Integrated Circuit)

A device in which components such as resistors, diodes, and transistors are formed on the surface of a single piece of semiconductor.

ICR (In-Circuit Reconfigurable)

An SRAM-based or similar component that can be dynamically reprogrammed on-the-fly while remaining resident in the system.

Idempotent Rules

Rules derived from the combination of a single Boolean variable with itself.

IDM (Integrated Device Manufacturer)

A company that focuses on designing, manufacturing, and selling integrated circuits as opposed to complete electronic systems.

Impedance

The resistance to the flow of current caused by resistive, capacitive, or inductive devices (or undesired parasitic elements) in a circuit.

In-Circuit Reconfigurable

See ICR.

Inclusion

A defect in a crystalline structure.

Inductance

A property of a conductor that allows it to store energy in a magnetic field that is induced by a current flowing through it. The base unit of inductance is the *henry*.

Inert Gas

See Noble Gas.

In-System Programmable

See ISP.

Integrated Circuit

See IC.

Integrated Device Manufacturer

See IDM.

Intellectual Property

See IP.

Invar

An alloy similar to bronze.

Involution Rule

A rule that states that an even number of Boolean inversions cancel each other out.

Ion

A particle formed when an electron is added to, or subtracted from, a neutral atom or group of atoms.

Ion Implantation

A process in which beams of ions are directed at a semiconductor to alter its type and conductivity in certain regions.

IP (Intellectual Property)

When a team of electronics engineers is tasked with designing a complex integrated circuit, rather than “reinvent the wheel,” they may decide to purchase the plans for one or more functional blocks that have already been created by someone else. The plans for these functional blocks are known as *intellectual property (IP)*. IP blocks can range all the way up to sophisticated communications functions and microprocessors. The more complex functions—like microprocessors—may be referred to as “cores.”

ISP (In-System Programmable)

An E²-based, FLASH-based, or similar component that can be reprogrammed while remaining resident on the circuit board.

JEDEC (Joint Electronic Device Engineering Council)

A council that creates, approves, arbitrates, and oversees industry standards for electronic devices. In programmable logic, the term JEDEC refers to a textual file containing information used to program a device. The file format is a JEDEC-approved standard and is commonly referred to as *a JEDEC file*.

Jelly Bean Device

An integrated circuit containing a small number of simple logic functions.

Joint Electronic Device Engineering Council

See JEDEC.

Jumper

A small piece of wire used to link two tracks on a circuit board.

Karnaugh Map

A graphical technique for representing a logical function. Karnaugh maps are often useful for the purposes of minimization.

Kelvin Scale of Temperature

A scale of temperature that was invented by the British mathematician and physicist William Thomas, first Baron of Kelvin. Under the *Kelvin*, or *absolute*, scale of temperature, 0 K (corresponding to -273°C) is the coldest possible temperature and is known as *absolute zero*.

Kilo

Unit qualifier (symbol = k) representing one thousand, or 10^3 . For example, 3 kHz stands for 3×10^3 hertz.

Kipper

A fish cured by smoking and salting.⁸

Laminate

A material constructed from thin layers or sheets. Often used in the context of circuit boards.

Large-Scale Integration

See LSI.

Laser Diode

A special semiconductor diode that emits a beam of coherent light.

Last-In First-Out

See LIFO.

⁸Particularly tasty as a breakfast dish.

Latch-Up Condition

A condition in which a circuit draws uncontrolled amounts of current, and certain voltages are forced, or “latched-up,” to some level. Particularly relevant in the case of CMOS devices, which can latch-up if their operating conditions are violated.

Lateral Thermal Conductivity

Good lateral thermal conductivity means that the heat generated by components mounted on a substrate can be conducted horizontally across the substrate and out through its leads.

Lead

(1) A metallic element (chemical symbol Pb). (2) A metal conductor used to provide a connection from the inside of a device package to the outside world for soldering or other mounting techniques. Leads are also commonly called *pins*.

Lead Frame

A metallic frame containing leads (pins) and a base to which an unpackaged integrated circuit is attached. Following encapsulation, the outer part of the frame is cut away and the leads are bent into the required shapes.

Lead Through-Hole

See LTH.

Least-Significant Bit

See LSB.

Least-Significant Byte

See LSB.

LED (Light-Emitting Diode)

A semiconductor diode that behaves in a similar manner to a normal diode except that, when turned on, it emits light in the visible or *Infrared (IR)* regions of the electromagnetic spectrum.

Level-Sensitive

An input whose effect on a function depends only on its current logic value or level, and is not directly related to it transitioning from one logic value to another.

LFSR (Linear Feedback Shift Register)

A shift register whose data input is generated as an XOR or XNOR of two or more elements in the register chain.

LIFO (Last-In First-Out)

A memory device in which data is read out in the reverse order to which it was written in.

Light-Emitting Diode

See LED.

Line

Used to refer to the width of a track: for example, “*This circuit board track has a line-width of 5 mils (five-thousandths of an inch).*”

Linear Feedback Shift Register

See LFSR.

Literal

A variable (either true or inverted) in a Boolean equation.

Logic Function

A mathematical function that performs a digital operation on digital data and returns a digital value.

Logic Gate

The physical implementation of a logic function.

Logic Synthesis

A process in which a program is used to automatically convert a high-level textual representation of a design [specified using a *Hardware Description Language (HDL)* at the *Register Transfer Level (RTL)* of abstraction] into equivalent Boolean Equations (like the ones introduced in *Chapter 9: Boolean Algebra*). The synthesis tool automatically performs simplifications and minimizations, and eventually outputs a gate-level netlist.

Low-Fired Cofired

Similar in principle to standard cofired ceramic substrate techniques. However, low-fired cofired uses modern ceramic materials with compositions that allow them to be fired at temperatures as low as 850°C. Firing at these temperatures in an inert atmosphere such as nitrogen allows nonrefractory metals such as copper to be used to create tracks.

LSB

(1) (Least Significant Bit) The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). **(2) (Least Significant Byte)** The byte in a multi-byte word that represents the least significant values (typically the right-hand byte).

LSI (Large Scale Integration)

Refers to the number of logic gates in a device. By one convention, large-scale integration represents a device containing 100 to 999 gates.

LTH (Lead Through-Hole)

A technique for populating circuit boards in which component leads are inserted into plated through-holes. Often abbreviated to "through-hole" or "thru-hole." When all of the components have been inserted, they are soldered to the board, usually using a wave soldering technique.

Macro Cell

See Hard Macro.

Macro Function

See Soft Macro.

Magnetic Random Access Memory

See MRAM.

Magnetic Tunnel Junction

See MTJ.

Mask

See Optical Mask.

Mask Programmable

A device such as a read-only memory, which is programmed during its construction using a unique set of masks.

Maximal Displacement

A linear feedback shift register whose taps are selected such that changing a single bit in the input data stream will cause the maximum possible disruption to the register's contents.

Maximal Length

A linear feedback shift register that sequences through $(2^n - 1)$ states before returning to its original value.

Maxterm

The logical OR of the inverted variables associated with an input combination to a logical function.

MBE (Molecular Beam Epitaxy)

A technique for creating thin films on substrates in precise patterns, in which the substrate is placed in a high vacuum, and a guided beam of ionized molecules is

fired at it, effectively allowing molecular-thin layers to be “painted” onto the substrate where required.

MCM (Multichip Module)

A generic name commonly used in the late 1990s for a group of advanced interconnection and packaging technologies featuring unpackaged integrated circuits mounted directly onto a common substrate. Today, MCMs would be regarded as being forerunners of *System-in-Package* (SiP) technology.

Medium-Scale Integration

See MSI.

Mega

Unit qualifier (symbol = M) representing one million, or 10^6 . For example, 3 MHz stands for 3×10^6 hertz.

Memory Cell

A unit of memory used to store a single binary digit, or bit, of data.

Memory Word

A number of memory cells logically and physically grouped together. All the cells in a word are typically written to, or read from, at the same time.

Metallization Layer

A layer of conducting material on an integrated circuit that is selectively deposited or etched to form connections between logic gates. There may be several metallization layers separated by dielectric (insulating) layers.

Metal-Oxide Semiconductor

See MOS.

Meta-Stable

A condition where the outputs of a logic function are oscillating uncontrollably between undefined values.

Micro

Unit qualifier (symbol = μ) representing one millionth, or 10^{-6} . For example, 3 μ s stands for 3×10^{-6} seconds.

Microcontroller (μ C)

A microprocessor augmented with special-purpose inputs, outputs, and control logic like counter-timers.

Microprocessor (μ P)

A general-purpose computer implemented on a single integrated circuit (or sometimes on a group of related chips called a *chipset*).

Milli

Unit qualifier (symbol = m) representing one thousandth, or 10^{-3} . For example, 3 ms stands for 3×10^{-3} seconds.

Minimization

The process of reducing the complexity of a Boolean expression.

Minterm

The logical AND of the variables associated with an input combination to a logical function.

Mixed-signal

Typically refers to an integrated circuit that contains both analog and digital elements.

Mod

See Modulus.

Modulo

See Modulus.

Modulus

Refers to the number of states that a function such as a counter will pass through before returning to its original value. For example, a function that counts from 0000_2 to 1111_2 (0 to 15 in decimal) has a modulus of 16 and would be called a modulo-16 or mod-16 counter.

Molecular Beam Epitaxy

See MBE.

Moore's Law

In 1965, Gordon Moore (who was to cofound Intel Corporation in 1968) noted that new generations of memory devices were released approximately every 18 months, and that each new generation of devices contained roughly twice the capacity of its predecessor. This observation subsequently became known as *Moore's Law*, and it has been applied to a wide variety of electronics trends.

MOS (Metal-Oxide Semiconductor)

A family of transistors.

Most-Significant Bit

See MSB.

Most-Significant Byte

See MSB.

Motherboard

A backplane containing active components such as integrated circuits.

MSB

(1) **(Most Significant Bit)** The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). (2) **(Most Significant Byte)** The byte in a multi-byte word that represents the most significant values (typically the left-hand byte).

MSI (Medium Scale Integration)

Refers to the number of logic gates in a device. By one convention, medium-scale integration represents a device containing 13 to 99 gates.

MTJ (Magnetic Tunnel Junction)

A sandwich of two ferromagnetic layers separated by a thin insulating layer. An MRAM memory cell is created by the intersection of two wires (say a "row" line and a "column" line) with an MTJ sandwiched between them.

Multichip Module

See MCM.

Multilayer

A printed circuit board constructed from a number of very thin single-sided and/or double-sided boards, which are bonded together using a combination of temperature and pressure.

Multiplexer (Digital)

A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.

Nano

Unit qualifier (symbol = n) representing one-thousandth of one-millionth, or 10^{-9} . For example, 3 ns stands for 3×10^{-9} seconds.

Nanobot

A molecular-sized robot. *See also Nanotechnology.*

Nanophase Materials

A form of matter that was only (relatively) recently discovered, in which small clusters of atoms form the building blocks of a larger structure. These structures differ from those of naturally occurring crystals, in which individual atoms arrange themselves into a lattice.

Nanotechnology

This is an elusive term that is used by different research and development teams to refer to whatever it is that they're working on at the time. However, regardless of their area of interest, nanotechnology always refers to something extremely small. Perhaps the "purest" form of nanotechnology is that of molecular-sized units that assemble themselves into larger products.

Nanotubes

A structure that may be visualized as taking a thin sheet of carbon and rolling it into a tube. Nanotubes can be formed with walls that are only one atom thick. The resulting tube has a diameter of 1 nano (one thousandth of one millionth of a meter). Nanotubes are an almost ideal material: they are stronger than steel, have excellent thermal stability, and they are also tremendous conductors of heat and electricity. In addition to acting as wires, nanotubes can be persuaded to act as transistors—this means that we now have the potential to replace silicon transistors with molecular-sized equivalents at a level where standard semiconductors cease to function.

Negative-Edge

Beware, here be dragons! In this book a *negative-edge* is defined as a transition from a logic 1 to a logic 0. This definition is therefore consistent with the other definitions used throughout the book, namely those for *active-high*, *active-low*, *assertion-level logic*, *positive-logic*, and *negative-logic*. It should be noted, however, that some would define a negative-edge as "*a transition from a higher to a lower voltage level, passing through a threshold voltage level.*" However, this definition serves only to cause confusion when combined with negative-logic implementations.

Negative Ion

An atom or group of atoms with an extra electron.

Negative Logic

A convention dictating the relationship between logical values and the physical voltages used to represent them. Under this convention, the more negative potential is considered to represent TRUE and the more positive potential is considered to represent FALSE.

Negative Resist

A process in which ultraviolet radiation passing through the transparent areas of a mask causes the resist to be cured. The uncured areas are then removed using an appropriate solvent.

Negative-True

See Negative Logic.

Nibble

See *Nybble*.

NMOS (N-channel MOS)

Refers to the order in which the semiconductor is doped in a MOS device. That is, which structures are constructed as N-type versus P-type material.

Noble Gas

Gases whose outermost electron shells are completely filled with electrons. Such gases are extremely stable and it is difficult to coerce them to form compounds with other elements. There are six noble gases: helium,⁹ neon, argon, krypton, xenon, and radon. This group of elements was originally known as the *inert gases*, but in the early 1960s it was found to be possible to combine krypton, xenon, and radon with fluorine to create compounds. Although helium, neon, and argon continue to resist, there is an increasing trend to refer to this group of gases as *noble* rather than *inert*.

Noble Metal

Metals such as gold, silver, and platinum that are extremely inactive and are unaffected by air,¹⁰ heat, moisture, and most solvents.

Noise

The miscellaneous rubbish that gets added to a signal on its journey through a circuit. Noise can be caused by capacitive or inductive coupling, or from externally generated electromagnetic interference.

Nonrecurring Engineering

See *NRE*.

Nonvolatile

A memory device that does not lose its data when power is removed from the system.

Nonvolatile RAM (nV RAM)

A device that is generally formed from an SRAM die mounted in a package with a very small battery, or as a mixture of SRAM and E²PROM cells fabricated on the same die.

⁹The second most common element in the universe (after hydrogen).

¹⁰Before you start penning letters of protest, silver can be attacked by sulfur and sulfides, which occur naturally in the atmosphere. Thus, the tarnishing seen on your mother's silver candlesticks is actually silver sulfide and not silver oxide.

NPN (N-type P-type N-type)

Refers to the order in which the semiconductor is doped in a bipolar junction transistor.

NRE (Nonrecurring Engineering)

This typically refers to the costs associated with developing an ASIC or ASSP. The NRE depends on a number of factors, including the complexity of the design, the style of packaging, and who does what in the design flow (that is, how the various tasks are divided between the system design house and the ASIC vendor).

N-type

A piece of semiconductor doped with impurities that make it amenable to donating electrons.

Nibble

A group of four binary digits, or bits.

Octal

Base-8 numbering system. Each octal digit can be directly mapped onto three binary digits, or bits.

Ohm

Unit of resistance. The Greek letter omega, Ω , is often used to represent ohms; for example, $1\text{ M}\Omega$ indicates one million ohms.

One-Hot Encoding

A form of state assignment for state machines in which each state is represented by an individual state variable.

One-Time Programmable

A device such as a PAL, PLA, or PROM that can be programmed a single time and whose contents cannot be subsequently erased.

Operating System

The collective name for the set of master programs that control the core operation and the base-level user-interface of a computer.

Optical Interconnect

The generic name for interconnection strategies based on optoelectronic systems, including *fiber-optics*, *free-space*, *guided-wave*, and *holographic* techniques.

Optical Lithography

A process in which radiation at optical wavelengths (usually in the *ultraviolet (UV)* and *extreme ultraviolet (EUV)* ranges) is passed through a mask, and the resulting patterns are projected onto a layer of resist coating the substrate material.

Optical Mask

A sheet of material carrying patterns that are either transparent or opaque to the wavelengths used in an optical-lithographic process. Such a mask can carry millions of fine lines and geometric shapes.

Optoelectronic

Refers to a system that combines optical and electronic components.

Organic Resist

A material that is used to coat a substrate and is then selectively cured to form an impervious layer. These materials are called *organic* because they are based on carbon compounds as are living creatures.

Organic Solvent

A solvent for organic materials such as those used to form organic resists.

Organic Substrate

Substrate materials such as FR4, in which woven glass fibers are bonded together with an epoxy. These materials are called *organic* because epoxies are based on carbon compounds as are living creatures.

Overgassing

One of the final stages in the integrated circuit fabrication process in which the entire surface of the wafer is coated with a layer of silicon dioxide or silicon nitride. This layer may also be referred to as the *barrier layer* or the *passivation layer*. An additional lithographic step is required to pattern holes in this layer to allow connections to be made to the pads.

Pad

An area of metallization on a substrate used for probing or to connect to a via, plated through-hole, or an external interconnect.

Padcap

A special flavor of circuit board used for high-reliability military applications, also known as *pads-only-outer-layers*. Distinguished by the fact that the outer surfaces of the board have pads but no tracks. Signal layers are only created on the inner planes, and tracks are connected to the surface pads by vias.

Pad Grid Array

See PGA.

Pad Stack

Refers to any *pads*, *anti-pads*, and *thermal relief pads* associated with a via or a plated through-hole as it passes through the layers forming the substrate.

PAL (Programmable Array Logic)¹¹

A programmable logic device in which the AND array is programmable but the OR array is predefined. *See also PLA, PLD, and PROM.*

Parallel-In Serial-Out

See PISO.

Parasitic Effects

The effects caused by undesired resistive, capacitive, or inductive elements inherent in the material or topology of a track or component.

Passivation Layer

See Overgassing.

Passive Trimming

A process in which a laser beam is used to trim components such as thick-film and thin-film resistors on an otherwise unpopulated and unpowered hybrid or *System-in-Package* (SiP) substrate. Probes are placed at each end of a component to monitor its value while the laser cuts away (evaporates) some of the material forming the component.

Pass-Transistor Logic

A technique for connecting MOS transistors such that data signals pass between their source and drain terminals. Pass-transistor logic minimizes the number of transistors required to implement a function, and is typically employed by designers of cell libraries or full-custom integrated circuits.

PC100

A popular form of SDRAM-based computer memory that runs at 100 MHz (the data bus is 64-bits wide, although 128-bit-wide versions (using dual 64-bit cards in parallel) have been used in high-end machines).

PC133

A popular form of SDRAM-based computer memory that runs at 133 MHz (the data bus is 64-bits wide, although 128-bit-wide versions (using dual 64-bit cards in parallel) have been used in high-end machines).

PCB (Printed Circuit Board)

A type of circuit board that has conducting tracks superimposed, or "printed," on one or both sides, and may also contain internal signal layers and power and ground planes. An alternative name, *Printed Wire Board* (PWB), is commonly used in America.

¹¹PAL is a registered trademark of Monolithic Memories, Inc.

Peta

Unit qualifier (symbol = P) representing one thousand million million, or 10^{15} . For example, 3 PHz stands for 3×10^{15} hertz.

PGA

- (1) (Pad Grid Array)** A packaging technology in which a device's external connections are arranged as an array of conducting pads on the base of the package.
(2) (Pin Grid Array) A packaging technology in which a device's external connections are arranged as an array of conducting leads, or pins, on the base of the package.

PHB (Photochemical Hole Burning)

An optical memory technique, in which a laser in the visible waveband is directed at a microscopic point on the surface of a slice of glass that has been doped with organic dyes or rare-earth elements. The laser excites electrons in the glass such that they change the absorption characteristics of that area of the glass and leave a *band*, or *hole*, in the absorption spectrum.

Photochemical Hole Burning

See PHB.

Phototransistor

A special transistor that converts an optical input in the form of light into an equivalent electronic signal in the form of a voltage or current.

Pico

Unit qualifier (symbol = p) representing one-millionth of one-millionth, or 10^{-12} . For example, 3 ps stands for 3×10^{-12} seconds.

Pin

See Lead.

Pin Grid Array

See PGA.

PISO (Parallel-In Serial Out)

Refers to a shift register in which the data is loaded in parallel and read out serially.

PLA (Programmable Logic Array)

The most user-configurable of the traditional programmable logic devices, because both the AND and OR arrays are programmable. *See also PAL, PLD, and PROM.*

Place Value

Refers to a numbering system in which the value of a particular digit depends both on the digit itself and its position in the number.

Plasma

A gaseous state in which the atoms or molecules are dissociated to form ions.

Plated Through-Hole

See PTH.

PLD (Programmable Logic Device)

The generic name for a device constructed in such a way that the designer can configure, or "program" it to perform a specific function. *See also PAL, PLA, and PROM.*

PMOS (P-channel MOS)

Refers to the order in which the semiconductor is doped in a MOS device. That is, which structures are constructed as P-type versus N-type material.

PNP (P-type N-type P-type)

Refers to the order in which the semiconductor is doped in a bipolar junction transistor.

Polysilicon Layer

An internal layer in an integrated circuit used to create the gate electrodes of MOS transistors. In addition to forming gate electrodes, the polysilicon layer can also be used to interconnect components. There may be several polysilicon layers separated by dielectric (insulating) layers.

Populating

The act of attaching components to a substrate.

Positive-Edge

Beware, here be dragons! In this book a *positive-edge* is defined as a transition from a logic 0 to a logic 1. This definition is therefore consistent with the other definitions used throughout the book, namely those for *active-high*, *active-low*, *assertion-level logic*, *positive-logic*, and *negative-logic*. It should be noted, however, that some would define a positive-edge as "*a transition from a lower to a higher voltage level, passing through a threshold voltage level.*" However, this definition serves only to cause confusion when combined with negative-logic implementations.

Positive Ion

An atom or group of atoms lacking an electron.

Positive Logic

A convention that dictates the relationship between logical values and the physical voltages used to represent them. Under this convention, the more positive potential is considered to represent TRUE and the more negative potential is considered to represent FALSE.

Positive Resist

A process in which radiation passing through the transparent areas of a mask causes previously cured resist to be degraded. The degraded areas are then removed using an appropriate solvent.

Positive-True

See Positive Logic.

Power Plane

A conducting layer in or on the substrate providing power to the components. There may be several power planes separated by insulating layers.

Precedence of Operators

Determines the order in which operations are performed. For example, in standard arithmetic the multiplication operator has a higher precedence than the addition operator. Thus, in the equation $6 + 2 \times 3$, the multiplication is performed before the addition and the result is 12. Similarly, in Boolean Algebra, the AND operator has a higher precedence than the OR operator.

Prepreg

Nonconducting semi-cured layers of FR4 used to separate conducting layers in a multilayer circuit board.

Primitives

Simple logic functions such as BUF, NOT, AND, NAND, OR, NOR, XOR, and XNOR. These may also be referred to as *primitive logic gates*.

Printed Circuit Board

See PCB.

Printed Wire Board

See PWB.

Product-of-Sums

A Boolean equation in which all of the *maxterms* corresponding to the lines in the truth table for which the output is a logic 0 are combined using AND operators.

Product Term

A set of literals linked by an AND operator.

Programmable Array Logic

See PAL.

Programmable Logic Array

See PLA.

Programmable Logic Device

See PLD.

Programmable Read-Only Memory

See PROM.

PROM (Programmable Read-Only Memory)

A programmable logic device in which the OR array is programmable but the AND array is predefined. Usually considered to be a memory device whose contents can be electrically programmed (once) by the designer. *See also PAL, PLA, and PLD.*

Proposition

A statement that is either true or false with no ambiguity. For example, the proposition "*I just tipped a bucket of burning oil into your lap*" is either true or false, but there's no ambiguity about it.

Protein

A complex organic molecule formed from chains of amino acids, which are themselves formed from combinations of certain atoms, namely: carbon, hydrogen, nitrogen, oxygen, usually sulfur, and occasionally phosphorous or iron. Additionally, the chain of amino acids "*folds in on itself*," thereby forming an extremely complex three-dimensional shape. Organic molecules have a number of useful properties, not the least of which is that their structures are intrinsically "*self healing*" and reject contamination. Also, in addition to being extremely small, many organic molecules have excellent electronic properties. Unlike metallic conductors, they transfer energy by moving electron excitations from place to place rather than relocating entire electrons. This can result in switching speeds that are orders of magnitude faster than their semiconductor equivalents.

Protein Memory

A form of memory based on organic proteins. *See also Protein.*

Protein Switch

A form of switch based on organic proteins. *See also Protein.*

Pseudo-Random

An artificial sequence of values that give the appearance of being random, but which are also repeatable.

PTH (Plated Through-Hole)

(1) A hole in a double-sided or multilayer board that is used to accommodate a through-hole component lead and is plated with copper. (2) An alternative name for the *Lead Through-Hole* (LTH) technique for populating circuit boards in which component leads are inserted into plated through-holes.

P-type

A piece of semiconductor doped with impurities that make it amenable to accepting electrons.

PWB (Printed Wire Board)

A type of circuit board that has conducting tracks superimposed, or "printed," on one or both sides, and may also contain internal signal layers and power and ground planes. An alternative name, *Printed Circuit Board* (PCB), is predominantly used in Europe and Asia.

QDR (Quad Data Rate)

A modern form of SDRAM-based memory. The original SDRAM specification was based on using one of the clock edges only (say the rising edge) to read/write data out-of/into the memory. QDR refers to memory that has separate data in and data out busses both designed in such a way that data can be read/written on both edges of the clock. This effectively quadruples the amount of data that can be pushed through the system without increasing the clock frequency (like many things this sounds simple if you say it fast, but making this work is trickier than it may at first appear). *See also DDR and SDRAM.*

QFP (Quad Flat Pack)

The most commonly used package in surface mount technology to achieve a high lead count in a small area. Leads are presented on all four sides of a thin square package.

Quad Date Rate

See QDR.

Quad Flat Pack

See QFP.

Quantization

Part of the process by which an analog signal is converted into a series of digital values. First of all the analog signal is sampled at specific times. For each sample, the complete range of values that the analog signal can assume is divided into a set of discrete bands or quanta. Quantization refers to the process of determining which band the current sample falls into. *See also Sampling.*

Quinary

Base-5 numbering system.

Radix

Refers to the number of digits in a numbering system. For example, the decimal numbering system is said to be radix-10. May also be referred to as the "base."

RAM (Random Access Memory)

A data storage device from which data can be read out and new data can be written in. Unless otherwise indicated, the term RAM is typically taken to refer to a semiconductor device in the form of an integrated circuit.

RDRAM (Rambus DRAM)

A form of DRAM computer memory based on Rambus technology. *See also Rambus.*

Read-Only Memory

See ROM.

Read-Write Memory

See RWM.

Real Estate

Refers to the amount of area available on a *substrate*.

Reed-Müller Logic

Logic functions implemented using only XOR and XNOR gates.

Reflow Oven

An oven employing *Infrared (IR)* radiation or hot air.

Reflow Soldering

A *surface mount technology* process in which the substrate and attached components are passed through a reflow oven to melt the solder paste.

Refractory Metal

Metals such as tungsten, titanium, and molybdenum (try saying this last one quickly), which are capable of withstanding extremely high, or refractory, temperatures.

Register Transfer Level

See RTL.

Remotely Reconfigurable Hardware

A product whose function may be customized remotely, by telephone or radio, while remaining resident in the system.

Resist

A material that is used to coat a substrate and is then selectively cured to form an impervious layer.

Resistor-Transistor Logic

See RTL.

Rigid Flex

Hybrid constructions that combine standard rigid circuit boards with flexible printed circuits (flex), thereby reducing the component count, weight, and susceptibility to vibration of the circuit, and greatly increasing its reliability.

RIMM

This really doesn't stand for anything, *per se*, but it is the trademarked name for a *Rambus* memory module. RIMMs are similar in concept to DIMMs, but have a different pin count and configuration. *See also* DIMM and SIMM.

Rising-Edge

See Positive-Edge.

ROM (Read-Only Memory)

A data storage device from which data can be read out, but new data cannot be written in. Unless otherwise indicated, the term ROM is typically taken to refer to a semiconductor device in the form of an integrated circuit.

RTL

(1) (**Register Transfer Level**) A relatively high level of abstraction at which design engineers capture the functionality of digital integrated circuit designs described using a *Hardware Description Language (HDL)*. (2) (**Resistor-Transistor Logic**) Logic gates implemented using particular configurations of resistors and bipolar junction transistors. For the majority of today's designers, resistor-transistor logic is of historical interest only.

RWM (Read-Write Memory)

Alternative (and possibly more appropriate) name for a *Random Access Memory (RAM)*.

Sampling

Part of the process by which an analog signal is converted into a series of digital values. Sampling refers to observing the value of the analog signal at specific times. *See also Quantization.*

Scalar Notation

A notation in which each signal is assigned a unique name; for example, a_3 , a_2 , a_1 , and a_0 . *See also Vector Notation.*

Scaling

A technique for making transistors switch faster by reducing their size. This strategy is known as *scaling*, because all of the transistor's features are typically reduced by the same proportion.

Schematic

Common name for a circuit diagram.

Schematic Capture

A software program used to capture schematic diagrams.

Scrubbing

The process of vibrating two pieces of metal (or metal-coated materials) at ultrasonic frequencies to create a friction weld.

SDRAM (Synchronous DRAM)

Until the latter half of the 1990s, DRAM-based computer memories were asynchronous, which means they weren't synchronized to the system clock. Over time, the industry migrated to *Synchronous DRAM (SDRAM)*, which is synchronized to the system clock and makes everyone's lives much easier. Note that SDRAM is based on core DRAM concepts it's all in the way you tweak the chips and connect them together.

Sea-of-Cells

Popular name for a channel-less gate array.

Sea-of-Gates

Popular name for a channel-less gate array.

Seed Value

An initial value loaded into a *Linear Feedback Shift Register* or random number generator.

Semiconductor

A special class of material that can exhibit both conducting and insulating properties.

Sensor

A transducer that detects a physical quantity and converts it into a form suitable for processing. For example, a microphone is a sensor that detects sound and converts it into a corresponding voltage or current.

Sequential

A function whose output value depends not only on its current input values, but also on previous input values. That is, the output value depends on a sequence of input values.

Serial-In Parallel-Out

See SIPO.

Serial-In Serial-Out

See SISO.

Sexagesimal

Base-60 numbering system.

Side-Emitting Laser Diode

A laser diode constricted at the edge of an integrated circuit's substrate such that, when power is applied, the resulting laser beam is emitted horizontally; that is, parallel to the surface of the substrate.

Signal Conditioning

Amplifying, filtering, or otherwise processing a (typically analog) signal.

Signal Layer

A layer carrying tracks in a circuit board, hybrid, or *System-in-Package* (SiP).

Signature

Refers to the *checksum* value from a *Cyclic-Redundancy Check* (CRC) when used in the guided-probe form of functional test.

Signature Analysis

A guided-probe functional-test technique based on *signatures*.

Sign Bit

The most significant binary digit, or bit, of a signed binary number (if set to a logic 1, this bit represents a negative quantity).

Signed Binary Number

A binary number in which the most-significant bit is used to represent a negative quantity. Thus, a signed binary number can be used to represent both positive and negative values.

Sign-Magnitude

Negative numbers in standard arithmetic are typically represented in sign-magnitude form by prefixing the value with a minus sign: for example, -27. For reasons of efficiency, computers rarely employ the sign-magnitude form. Instead, they use *signed binary numbers* to represent negative values.

Silicon Bumping

The process of depositing additional metallization on a die's pads to raise them fractionally above the level of the *barrier layer*.

Silicon Chip

Although a variety of semiconductor materials are available, the most commonly used is silicon, and integrated circuits are popularly known as *silicon chips*, or simply *chips*.

Silicon Compiler

The program used in compiled cell technology to generate the masks used to create components and interconnections. May also be used to create data-path functions and memory functions.

SIMM (Single Inline Memory Module)

A single memory integrated circuit can only contain a limited amount of data, so a number are gathered together onto a small circuit board called a *memory module*. Each memory module has a line of gold-plated pads on both sides of one edge of the board. These pads plug into a corresponding connector on the main computer board. A *single inline memory module* (SIMM) has the same electrical signal on corresponding pads on the front and back of the board (that is, the pads on opposite sides of the board are "tied together"). *See also DIMM and RIMM.*

Simple PLD

See SPLD.

Single Inline Memory Module

See SIMM.

Single-Sided

A printed circuit board with tracks on one side only.

Sintering

A process in which ultra-fine metal powders weld together at temperatures much lower than those required for larger pieces of the same materials.

SiP (System-in-Package)

A sophisticated package in which multiple silicon chips and other components are presented in a single package. The chips may be bare die or presented as *Chip Scale Packages (CSP)*; also, the chips may provide a mixture of analog and digital functions.

SIPO (Serial-In Parallel-Out)

Refers to a shift register in which the data is loaded in serially and read out in parallel.

SISO (Serial In Serial-Out)

Refers to a shift register in which the data is both loaded in and read out serially.

Skin Effect

In the case of high frequency signals, electrons are only conducted on the outer surface, or skin, of a conductor. This phenomenon is known as the *skin effect*.

Small-Scale Integration

See SSI.

SMD (Surface Mount Device)

A component whose packaging is designed for use with *Surface Mount Technology*.

SMOBC (Solder Mask Over Bare Copper)

A technique in which the *solder mask* is applied in advance of the *tin-lead plating*. This prevents solder from leaking under the mask when the tin-lead alloy melts during the process of attaching components to the board.

SMT (Surface Mount Technology)

A technique for populating hybrids, multi-chip modules, and circuit boards, in which packaged components are mounted directly onto the surface of the substrate. A layer of solder paste is screen-printed onto the pads, and the components are attached by pushing their leads into the paste. When all of the components have been attached, the solder paste is melted using either *reflow soldering* or *vapor-phase soldering*.

SoC (System-on-Chip)

In the past, an electronic system was typically composed of a number of integrated circuits, each with its own particular function (say a microprocessor, a communications function, some memory devices, etc.). For many of today's high-end applications, however, electronics engineers are combining all of these functions on a single device, which may be referred to as a *system-on-chip* (SoC).

Soft Macro (Macro Function)

A logic function defined by the manufacturer of an *Application-Specific Integrated Circuit (ASIC)* or by a third-party IP provider. The function is described in terms of the simple functions provided in the cell library and connections between them. The assignment of cells to basic cells and the routing of the tracks is

determined at the same time, and using the same tools, as for the other cells specified by the designer.

Software

Refers to intangible programs, or sequences of instructions, that are executed by *hardware*.

Solder

An alloy with a comparatively low melting point used to join less fusible metals. Note that the solder used in a brazing process is of a different type, being a hard solder with a comparatively high melting point composed of an alloy of copper and zinc (brass).

Solder Bump Bonding

A flip-chip technique (also known as *Solder Bumping*) for attaching a die to a package substrate. A minute ball of solder is attached to each pad on the die, and the die is flipped over and attached to the package substrate. Each pad on the die has a corresponding pad on the package substrate, and the package-die combo is heated so as to melt the solder balls and form good electrical connections between the die and the substrate. *See also Solder Bumping.*

Solder Bumping

A flipped chip technique (also known as *Solder Bump Bonding*) in which spheres of solder are formed on the die's pads. The die is flipped and the solder bumps are brought into contact with corresponding pads on the substrate. When all the chips have been mounted on the substrate, the solder bumps are melted using reflow soldering or vapor-phase soldering. *See also Solder Bump Bonding.*

Solder Mask

A layer applied to the surface of a printed circuit board that prevents solder from sticking to any metallization except where holes are patterned into the mask.

Solder Mask Over Bare Copper

See SMOBC.

Space

Used to refer to the width of the gap between adjacent tracks.

SPLD (Simple PLD)

Originally all PLDs contained a modest number of equivalent logic gates and were fairly simple. As more *Complex PLDs (CPLDs)* arrived on the scene, however, it became common to refer to their simpler cousins as *Simple PLDs (SPLDs)*.

SRAM (Static RAM)

A memory device in which each cell is formed from four or six transistors configured as a latch or a flip-flop. The term *static* is used because, once a value has been loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.

SSI (Small Scale Integration)

Refers to the number of logic gates in a device. By one convention, small-scale integration represents a device containing 1 to 12 gates.

Standard Cell

A form of *Application-Specific Integrated Circuit (ASIC)*, which, unlike a gate array, does not use the concept of a *basic cell* and does not have any prefabricated components. The ASIC vendor creates custom masks for every stage of the device's fabrication, allowing each logic function to be created using the minimum number of transistors.

State Assignment

The process by which the states in a state machine are assigned to the binary patterns that are to be stored in the state variables.

State Diagram

A graphical representation of the operation of a state machine.

State Machine

See FSM.

Statement

A sentence that asserts or denies an attribute about an object or group of objects. For example, "Your face resembles a cabbage."

State Table

A tabular representation of the operation of a state machine. Similar to a truth table, but also includes the *current state* as an input and the *next state* as an output.

State Transition

An arc connecting two states in a state diagram.

State Variable

One of a set of registers whose values represent the current state occupied by a state machine.

Static Flex

A type of *flexible printed circuit* that can be manipulated into permanent three-dimensional shapes for applications such as calculators and high-tech cameras, which require efficient use of volume and not just area.

Static RAM

See SRAM.

Steady State

A condition in which nothing is changing or happening.

Subatomic Erosion

See Electromigration.

Substrate

Generic name for the base layer of an integrated circuit, hybrid, *System-in-Package* (SiP), or circuit board. Substrates may be formed from a wide variety of materials, including semiconductors, ceramics, FR4 (fiberglass), glass, sapphire, or diamond depending on the application. Note that the term "substrate" has traditionally not been widely used in the circuit board world, at least not by the people who manufacture the boards. However, there is an increasing tendency to refer to a circuit board as a substrate by the people who populate the boards. The main reason for this is that circuit boards are often used as substrates in hybrids and *System-in-Packages* (SiPs), and there is a trend toward a standard terminology across all forms of interconnection technology.

Subtractive Process

A process in which a substrate is first covered with conducting material, and then any unwanted material is subsequently removed, or subtracted.

Sum-of-Products

A Boolean equation in which all of the *minterms* corresponding to the lines in the truth table for which the output is a logic 1 are combined using OR operators.

Superconductor

A material with zero resistance to the flow of electric current.

Surface-Emitting Laser Diode

A laser diode constricted on an integrated circuit's substrate such that, when power is applied, the resulting laser beam is emitted directly away from the surface of the substrate.

Surface Mount Device

See SMD.

Surface Mount Technology

See SMT.

Symbolic Logic

A mathematical form in which propositions and their relationships may be represented symbolically using Boolean equations, truth tables, Karnaugh Maps, or similar techniques.

Synchronous

(1) A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal. (2) A system whose operation is synchronized by a clock signal.

Synchronous DRAM

See SDRAM.

System-in-Package

See SiP.

System-on-Chip

See SoC.

TAB (Tape-Automated Bonding)

A process in which transparent flexible tape has tracks created on its surface. The pads on unpackaged integrated circuits are attached to corresponding pads on the tape, which is then stored in a reel. Silver-loaded epoxy is screen printed on the substrate at the site where the device is to be located and onto the pads to which the device's leads are to be connected. The reel of TAB tape is fed through an automatic machine, which pushes the device and the TAB leads into the epoxy. When the silver-loaded epoxy is cured using *reflow soldering* or *vapor-phase soldering*, it forms electrical connections between the TAB leads and the pads on the substrate.

Tap

A register output that is used to generate the next data input to a linear feedback shift register.

Tape Automated Bonding

See TAB.

Tera

Unit qualifier (symbol = T) representing one million million, or 10^{12} . For example, 3 THz stands for 3×10^{12} hertz.

Tertiary

Base-3 numbering system.

Tertiary Digit

A numeral in the tertiary scale of notation. A tertiary digit can adopt one of three states: 0, 1, or 2. Often abbreviated to "trit."

Tertiary Logic

An experimental technology in which logic gates are based on three distinct voltage levels. The three voltages are used to represent the tertiary digits 0, 1, and 2, and their logical equivalents FALSE, TRUE, and MAYBE.

Thermal Relief Pad

A special pattern etched around a via or a plated through-hole to connect it into a power or ground plane. A thermal relief pad is necessary to prevent too much heat being absorbed into the power or ground plane when the board is being soldered.

Thermal Tracking

Typically used to refer to the problems associated with optical interconnection systems whose alignment may be disturbed by changes in temperature.

Thick-Film Process

A process used in the manufacture of hybrids and *System-in-Packages* (SiPs) in which signal and dielectric (insulating) layers are screen-printed onto the substrate.

Thin-Film Process

A process used in the manufacture of hybrids and *System-in-Packages* (SiPs) in which signal layers and dielectric (insulating) layers are created using opto-lithographic techniques.

Through-Hole

See Lead Through-Hole, Plated Through-Hole, and Through-Hole Via.

Through-Hole Via

A via that passes all the way through the substrate.

Thru-Hole

A commonly used abbreviation of "through-hole."

Time-of-Flight

The time taken for a signal to propagate from one logic gate, integrated circuit output, or optoelectronic component to another.

Tinning

An electroless plating process in which exposed areas of copper on a circuit board are coated with a layer of protective alloy. The alloy is used to prevent the copper from oxidizing and provides protection against contamination.

Toggle

Refers to the contents or outputs of a logic function switching to the inverse of their previous logic values.

Trace

See Track.

Track

A conducting connection between electronic components. May also be called a *trace* or a *signal*. In the case of integrated circuits, such interconnections are often referred to collectively as *metallization*.

Transducer

A device that converts input energy of one form into output energy of another.

Transistor

A three-terminal semiconductor device that, in the digital world, can be considered to operate like a switch.

Transistor-Transistor Logic

See TTL.

Tri-State Function

A function whose output can adopt three states: 0, 1, and Z (high-impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit.

Trit

Abbreviation of *tertiary digit*. A tertiary digit can adopt one of three values: 0, 1, or 2.

Truth Table

A convenient way to represent the operation of a digital circuit as columns of input values and their corresponding output responses.

TTL (Transistor-Transistor Logic)

Logic gates implemented using particular configurations of bipolar junction transistors.

ULA (Uncommitted Logic Array)

One of the original names used to refer to gate array devices. This term has largely fallen into disuse.

ULSI (Ultra-Large-Scale Integration)

Refers to the number of logic gates in a device. By one convention, ultra-large-scale integration represents a device containing a million or more gates.

Ultra-Large-Scale Integration

See ULSI.

Uncommitted Logic Array

See ULA.

Unsigned Binary Number

A binary number in which all the bits are used to represent positive quantities. Thus, an unsigned binary number can only be used to represent positive values.

Vapor-Phase Soldering

A surface mount process in which a substrate carrying components attached by solder paste is lowered into the vapor-cloud of a tank containing boiling hydrocarbons. This melts the solder paste thereby forming good electrical connections. However, vapor-phase soldering is becoming increasingly less popular due to environmental concerns.

Vaporware

Refers to either hardware or software that exists only in the minds of the people who are trying to sell it to you.

Vector Notation

A notation in which a single name is used to reference a group of signals, and individual signals within the group are referenced by means of an index: for example, $a[3]$, $a[2]$, $a[1]$, and $a[0]$. This concept of a vector is commonly used in the context of electronics tools such as schematic capture packages, logic simulators, and graphical waveform displays. Some people prefer to use the phrase *binary group notation*, but the *phrase vector notation* is commonly used by practicing electronic engineers. (Note that this type of vector notation is in no way related to the algebraic concept of vector notation for Cartesian 2-space or 3-space.)

Very-Large-Scale Integration

See VLSI.

VHDL

A *Hardware Description Language (HDL)*, which came out of the U.S. Department of Defense (DoD), and which has evolved into an open standard. VHDL is an acronym for *VHSIC HDL* (where VHSIC is itself an acronym for *Very High Speed Integrated Circuit*).

Via

A hole filled or lined with a conducting material, which is used to link two or more conducting layers in a substrate.

Virtual Memory

A trick used by a computer's operating system to pretend that it has access to more memory than is actually available. For example, a program running on the computer may require 500 megabytes to store its data, but the computer may have only 128 megabytes of memory available. To get around this problem, whenever the program attempts to access a memory location that does not physically exist, the operating system performs a slight-of-hand and exchanges some of the contents in the memory with data on the hard disk.

Virus

See Computer Virus.

VLSI (Very Large Scale Integration)

Refers to the number of logic gates in a device. By one convention, very-large-scale integration represents a device containing 1000 to 999,999 gates.

Volatile

Refers to a memory device that loses any data it contains when power is removed from the system: for example, random-access memory in the form of SRAM or DRAM.

Wafer

A paper-thin slice cut from a cylindrical crystal of pure semiconductor.

Wafer Probing

The process of testing individual integrated circuits while they still form part of a wafer. An automated tester places probes on the device's pads, applies power to the power pads, injects a series of signals into the input pads, and monitors the corresponding signals returned from the output pads.

Waveguide

A transparent path bounded by nontransparent, reflective areas, which is fabricated directly onto the surface of a substrate. Used in the optical interconnection strategy known as *guided-wave*.

Wave Soldering

A process used to solder circuit boards populated with through-hole components. A machine creates a wave¹² of hot, liquid solder that travels across the surface of the tank. The populated circuit boards are passed over the wave-soldering machine on a conveyer belt. The velocity of the conveyer belt is carefully controlled and synchronized such that the solder wave brushes across the bottom of the board only once.

Wire Bonding

The process of connecting the pads on an unpackaged integrated circuit to corresponding pads on a substrate using wires that are finer than a human hair. Wire bonding may also be used to connect the pads on an unpackaged integrated circuit, hybrid, or *System-in-Package* (SiP) to the leads of the component package.

Word

A group of signals or logic functions performing a common task and carrying or storing similar data: for example, a value on a computer's *data* bus could be referred to as a "data word" or "a word of data."

X Architecture

An initiative proposed by a group of companies in 2001 to use diagonal tracks to connect functions on silicon chips (as opposed to traditional North-South and East-West tracking layers). Initial evaluations apparently show that this diagonal interconnect strategy can increase chip performance by 10% and reduce power consumption by 20%. However, it may take some time for design tools and processes (and popular acceptance) to catch up and start using this technique.

X-Ray Lithography

Similar in principle to optical lithography, but capable of constructing much finer features due to the shorter wavelengths involved. However, X-ray lithography requires an intense source of X-rays, is more difficult to use, and is considerably more expensive than optical lithography.

Yield

The number of devices that work as planned, specified as a percentage of the total number actually fabricated.

¹²A large ripple, actually.

Index

f: footnote; s: sidebar; p: pronunciation

Ω , 20^s
 μ , 188^f
“?” character, 123, 127
“!” (shriek character), 220^f, 380–2
“&” character, 100, 105, 220^f, 221
“^” character, 220^f
“|” character, 220^f
“~” character, 130, 138, 378, 425^f
~chip_select signal, 199
~clear inputs, 141
~enable signal, 378, 379, 380
~output_enable, 198
~preset input, 141
~rd, 198
~reset inputs, 141
1T versus 6T SRAM, 241–2
2:1 multiplexers, 127, 128, 353
2:4 decoders, 129, 132, 202
III-V valence semiconductor, 45
3:8 decoders, 130
3-D die stacking, 295–6, 304
4:1 multiplexers, 129
4:16 decoders, 411, 412
4-bit binary counter, 147, 413
4-bit LFSR, 413, 417
6T versus 1T SRAM, 241–2
8:1 multiplexers, 129
8-bit signed binary number, 94
 fixed-point representation, 449
54xx series, 174
74xx series, 174
100X multiplexing, 321
907 device, 174
1103 device, 175
4004 microprocessor, 175
4100 device, 175
7400 device, 174
7402 device, 174

A

abacus, 69
absolute scale of temperature *see*
 absolute zero
absolute zero, 336

abstraction, levels of
 algorithmic, 354
 behavioral, 354
 functional (Boolean, RTL),
 353–4
 gate-level netlist, 353
 higher, 363
 structural, 353
switch-level netlist, 352
transistor-level netlist, 352
Acanthostega, 82
Accellera committee, 357, 359
acceptor, 37
active-high
 control, 58
 outputs, 129^f
 signal, 377, 378
active-low
 control, 58
 outputs, 129, 130
 signal, 377, 378
active state, 129
active substrates *see* semiconductor
 substrates
active trimming, 281, 282
active versus passive devices, 47
actuator, 164
A/D (Analog-to-Digital) converter,
 162–4
addition
 signed binary numbers, 95–6
 unsigned binary numbers, 88–9
additive process, 255–7
address
 for decoders, 129
 for multiplexers, 127
address bus, 193, 196, 198, 201
admittance, 29–30
advanced packaging techniques
 3-D die stacking, 295–6
 chip-scale package (CSP)
 technology, 294–5
 mind boggles, 305–6
 Package-In-Package (PiP), 297

Package-On-Package (PoP), 297
rabbit hole, 293
System-In-Package (SiP), 296–7
 example, based on cofired
 ceramics, 298–305
 substrates, positive plethora of,
 297–8
 wire bonds versus flip-chip,
 293–4
AHDL (Analog Hardware
 Description Language), 358,
 372
Albert Hanson, 251
Alex Müller, 336
algorithm, 435^f
algorithmic, in level of abstraction,
 354
Al-Khawarizmi, 435^f
Allan Marquand, 117
Alon Kfir, 157^f
Altera Corporation, 192, 225, 226
alternate rounding *see* round-
 alternate algorithm
alternating current (AC), 28
alternative and future technologies
 buckyballs and nanotubes,
 328–30
 conductive adhesives, 334–5
 diamond substrates
 chemical vapor deposition,
 331–2
 chemical vapor infiltration, 332
 maverick inventor, 333
 single-crystal diamond,
 requirements for, 333–4
 ubiquitous laser beams, 332–3
electromagnetic transistor
 fabrication, 324–5
elemental computing arrays
 (ECAs), 310–14
heterojunction transistors, 325–7,
 328
mind boggles, 341–2
nanotechnology, 337–41

- alternative and future technologies (*continued*)
 optical interconnect, 314
 fiber-optic interconnect, 314–16
 free-space interconnect, 317,
 318
 guided-wave interconnect,
 318–19, 320
 optical memories, 320–1
 protein switches and memories,
 321–4
 reconfigurable computing, 307–10
 smorgasbord, 307
 superconductors, 335–7
 alumina, 227
 aluminum gallium arsenide (AlGaAs), 46
 aluminum indium gallium phosphite, 46
 American National Standards Institute (ANSI), 351^f
 American Standard Code for Information Interchange (ASCII), 351, 351^f
 American Standards Association (ASA), 351^f
 amino acids, 322
 amorphous crystalline structure, 15, 17
 amorphous silicon, 215–16
 Amphisbaena, 407^f
 Amplitude Modulation (AM) radio, 166
 amps, 19, 19^f
 Ampère, André-Marie, 19^f
 analog
 circuit, 346, 358, 365, 371
 design engineers, 347^f, 361
 versus digital system, 3
 bricks, experiments with, 6–9
 multi-value digital systems, 5–6
 synthesis, 371–2
 waveform, 7, 8
 Analog Hardware Description Language (AHDL), 358, 372
 Analog Signal Processing (ASP), 165–6
 analog-to-digital (A/D) converter, 162–4
 analogue, 4^f
 AND gate/function, 52–3, 60, 100–1, 126, 132, 389
 pass-transistor implementation, 423
 switch representation, 49
 transistor implementation, 61
 from two NANDs, 56
 AND-OR architecture, 223, 223^f
 André-Marie Ampère, 19^f
 anisotropic adhesives *see* conductive adhesives
 annoying tune, 278
 antifuse technologies, 215–17, 234
 anti-pads, 268, 269
 API (Application Programming Interface), 348
 Application Programming Interface (API), 348
 Application-Specific Integrated Circuits (ASICs), 174, 174^f, 235
 1T versus 6T SRAM, 241–2
 versus Application-Specific Standard Product (ASSP), 246
 basic cell, 236
 channeled gate arrays, 237
 channel-less devices, 237
 design flow, 238–40
 full custom devices, 236
 gate arrays, 236–8
 hard macros, 239
 input/output cells and pads, 245–6
 Micromosaic, 235
 players, 246–7
 sea-of-gates, 238
 soft macros, 239
 standard cell devices, 240–1
 structured ASICs, 242–5
 Applicon, 346
 Arabs, 70
 Aristotle, 117
 arithmetic rounding *see* round-half-up algorithm
 arrays, 195–6
 arsenic (As), 45
 Arthur Berry, 251
 artificial bones, 277^f
 ASCII (American Standard Code for Information Interchange), 351, 351^f
 ASICs *see* Application-Specific Integrated Circuits
 ASP (Analog Signal Processing), 165–6
 Assertion-Based Verification (ABV), 366
 assertion-level logic, 377
 shriek character ("!"), 380–2
 standard versus assertion-level logic, 378
 associative rules, 104
 ASSPs versus ASICs, 246
 Assyrians, 71
 Atalla, Martin M., 41
 Atmel Corporation, 472^f
 atoms, 11
 boron atoms (B), 36, 37, 428
 carbon atom, 15^f
 germanium atom, 327
 helium atom, 11, 12, 14, 14^f
 hydrogen atom, 12
 isotopes, 12
 negative ion, 12
 oxygen atom, 14
 phosphorus atoms, 36, 37
 positive ion, 12
 silicon atom, 15
 ATPG (Automatic Test Pattern Generator), 376
 atto, 31
 Augustus DeMorgan, 106
 Aurignacian period, 67
 Automatic Test Pattern Generator (ATPG), 376
 "automation", 347–8
 Aztecs, 76
- B**
- Babylonians, 71
 backplanes, 271–2
 optical backplanes, 316
 Bakelite, 252
 ball bond, 287
 ball grid array (BGA), 184, 260^f, 294, 303, 304, 305
 ballistic transistors, 330
 bankers rounding *see* round-half-even algorithm
 Bardeen, John, 39
 bare die, 286, 288
 encapsulation, in plastic, 290, 291
 barrier layer, 181
 base, 39
 base-2 (binary), 78–9
 base-5 (quinary), 77, 78
 base-8 (octal), 80–1
 base-10 (decimal), 70–1
 base-12 (duo-decimal), 71–3
 base-16 (hexadecimal), 80–1

- base-20 (vigesimal), 76
 base-60 (sexagesimal), 73–4
 basic cell, 236, 237, 238
 beauty (quarks), 11^f
 Bednorz, Georg, 336
 BEDO (Burst EDO), 208
 behavioral, in level of abstraction, 354
 Bell Laboratories, 36, 39, 41
 Berry, Arthur, 251
 BGA (Ball Grid Array), 184, 294
 BiCMOS (Bipolar CMOS), 187
 bidirectional data, 194, 200
 bit, 79
 billion, 31^f
 binary arithmetic
 base-2 (binary), 78–9, 87
 division, 98
 multiplication, 97–8
 nines' and ten's complements, 89–90
 signed binary numbers, 94–5
 addition, 95–6
 subtraction, 96–7
 sign-magnitude binary numbers, 93
 unsigned binary numbers, 87–8
 addition, 88–9
 subtraction, 91–3
 binary encoding, 155
 binary-to-gray converter, 395–6
 binary versus gray codes, 399
 binit, 79
 biological catalysts, 337, 337^f
 Bipolar CMOS (BiCMOS), 187
 bipolar junction transistors (BJTs), 39–41, 187, 429
 bipolar transistor, 39, 40, 45
 bi-quinary system, 69
 BIST (Built-In Self-Test), 375, 419–20, 419^f
 bits, 79
 bit 0, 87
 bit 7, 87
 and bytes, 197
 BJTs (bipolar junction transistors), 39–41, 187, 429
 black box, 112^f, 113, 114
 blind vias, 266, 267
 "blowing the fuses"/"burning the device", 215
 Bluespec SystemVerilog (BSV), 357
 bobble, 51^f, 58
 bones with notches, 67–8
 BOOL logic synthesis, 157^f
 Boole, George, 99, 100
 Boolean algebra, 85, 99
 associative rules, 104
 canonical forms, 114
 commutative rules, 104
 complementary rules, 102, 103
 conundrum, 114–15
 DeMorgan transformations, 106–7, 109–12
 first distributive rule, 105, 106
 idempotent rules, 102, 103
 involution rule, 103
 logic 0/logic 1, combining single variable with, 102
 maxterms, 112
 minterms, 112
 precedence of operators, 105
 primitive logic functions, 100–1
 product-of-sums form, 112–13,
 114
 second distributive rule, 105–6,
 107
 simplification rules, 106, 108
 sum-of-products form, 112–13,
 114
 boot-strapping, 195
 boric-acid glass, 320^f
 boron atoms (B), 36, 37, 428
 boron-doped silicon, 36
 borrow operations, 89, 90, 90^f
 bottom (quarks), 11^f
 boundary scan, 375
 Brattain, Walter, 39
 braze, 302
 BREO (Bit RE-Orderer), 310
 brick
 by brick, 149
 experiments with, 6–9
 BSHF (Barrel SHiPter), 310
 BSV (Bluespec SystemVerilog), 357
 bubble, 51^f
 Buckminsterfullerine, 328, 328^f
 buckyballs, 328
 BUF gate/function, 51–2, 58
 transistor implementation, 59–60
 buildup technology, 270
 Built-In Self-Test (BIST), 375, 419–20, 419^f
 bulk storage, 194
 Bulwer-Lytton, Edward George, 3
 Bulwer-Lytton fiction contest (2008), 9^s
 bump grid array *see* ball grid array
 bumping, silicon, 181
 buried vias, 266, 267
 Burst EDO (BEDO), 208
 byte, 79, 197
- C**
- C++, 354
 C.A. Swanson company, 39^f
 CAD (Computer-Aided Design), 235, 246, 346, 346^f, 346^{fp}
 CAE (Computer-Aided Engineering), 246, 346–7, 347^{fp}
 Calma, 346
 canonical forms, 114
 capacitance, 21–3
 capacitors, 21–3, 28
 creation, in thick-film hybrids, 282
 carbon atom, 15^f
 Carbon Nanotube FET (CNFET), 329
 carborundum, 45
 Carroll, Lewis, 117–18
 catalysts, biological, 337^f
 cathode ray tube (CRT), 329
 cavities, in multichip modules, 301, 302
 CBRAMs (Conductive-Bridging RAMs), 211–12
 cell library, 238
 cells, 195
 basic cells, 236, 237, 238
 input/output cells, 245–6
 Celts, 76
 Central Processing Unit (CPU), 193, 194
 ceramics
 cap hybrid package, 291
 cofired ceramics, 298–305
 handgun, 277^f
 hybrid substrates, 277, 277^f, 298
 low-fired cofired, 301
 channel, 42, 179, 237, 433
 channeled gate arrays, 237
 channel-less gate array, 237, 238
 Charles Ducas, 251
 Charles Lutwidge Dodgson, 117–18
 charm (quarks), 11^f
 chawmp, 79^f
 cheap-and-cheerful process, 182
 checksum, 416
 chemically amplified resist, 326^f
 Chemical Mechanical Polishing (CMP), 180, 369

- Chemical Vapor Deposition (CVD), 326–7, 331–2
 Chemical Vapor Infiltration (CVI), 332
 chevrons, 139
 Chip-On-Board (COB), 273–4, 295
 Chip-On-Flex (COF), 274
 Chip-Scale Package (CSP)
 technology, 185, 294–5
 chopping *see* truncation algorithm
 Chua, Leon O., 28
 Churchill, Winston, 342
 circuit diagrams, 345
 Claude Elwood Shannon, 100
 clepsydra, 72, 72^f
 clock input, 142, 142^f
 cloud of electrons, 11
 Cluster, 312
 CMOS (Complementary Metal-Oxide Semiconductor), 57, 57^f, 187
 CMP (Chemical Mechanical Polishing), 180, 369
 CNFET (Carbon Nanotube FET), 329
 COB (Chip-On-Board), 273–4, 295
 codes
 gray codes, 118, 147, 158, 393
 Co-Design Automation, 357
 coefficient of thermal expansion, 277, 278
 cofired ceramics, 298
 assembly and packaging, 301
 Ball Grid Arrays (BGAs), 302–3, 304
 Column Grid Arrays (CGAs), 303, 304
 die, populating, 304–5
 fuzz-buttons, 304
 low-fired cofired processes, 301
 pin grid arrays, 302, 303
 coin-operated machine, 151, 158, 159
 collector, 39
 Column Grid Arrays (CGAs), 303, 304
 combinational
 logic, 147, 219
 versus sequential functions, 132
 combinatorial *see* combinational
 commutative rules, 104
 Complementary Metal-Oxide Semiconductor (CMOS), 57, 57^f, 187
 complementary outputs, 132
 complementary rules, 102, 103
 complement
 nines' complement, 89–90, 91
 ones' complement, 91
 ten's complement, 89–90
 two's complement, 91–2
 complex functions
 brick by brick, 149
 combinational versus sequential functions, 132
 counters, 146–7
 decoders, 129–30
 D-type flip-flop, 139–41
 positive-edge triggered implementation, 142
 D-type latches, 138–9
 equality comparators, 126–7
 JK and T flip-flops, 143
 logic gates, 125
 multiplexers, 127–9
 Reset-Set latch (RS latch)
 NAND-based implementation, 137–8
 NOR implementation, 132–7
 scalar versus vector notation, 125–6
 setup and hold times, 148–9
 shift registers, 144–6
 tri-state functions, 130–2
 Complex PLDs (CPLDs), 224–7
 component-level netlist, 351
 compute-class elements, 310–11
 Computer-Aided Design (CAD), 235, 246, 346, 346^f, 346^p
 Computer-Aided Engineering (CAE), 246, 346–7, 347^{fp}
 computer-on-a-chip, 175
 ComputerVision, 346
 conditioning, 163
 conductance, 29–30
 conductive adhesives, 334–5
 Conductive-Bridging RAMs (CBRAMs), 211–12
 conductive ink technology, 272–3
 conductors, 17
 conjunction, 100
 “connect the NOTs”, 52
 Consumer Electronics Show (CES), 330
 Contexts
 Elements, 311
 continuous current *see* direct current
 copper, 17, 301
 core supply voltages, 187
 Count Alessandro Volta, 19^f
 counters, 146–7
 setup and hold times, 148–9
 Count Ferdinand von Zeppelin, 41^f
 counting
 in binary, 79
 in decimal, 71
 in duo-decimal, 71–3
 in hexadecimal, 81
 in octal, 81
 in quinary, 77, 78
 CPLDs (Complex PLDs), 224–7
 C programming language, 354
 CPU (Central Processing Unit), 193, 194
 CRC (Cyclic Redundancy Check)
 applications, 416–17
 creeping errors, 435
 Cro-Magnon, 67^f
 Crosspoint Switch (CPS), 312
 crumb, 79^f
 crystals, 11, 15
 CSP (Chip-Scale Package)
 technology, 185, 294–5
 CSSPs (Customer-Specific Standard Products), 233
 current, 18–19
 Customer-Specific Standard Products (CSSPs), 233
 CVI (Chemical Vapor Infiltration), 332
 Cyclic Redundancy Check (CRC)
 applications, 416–17

D

- d0 data input 127, 128
 d1 data input, 127
 D/A (Digital-to-Analog) converter, 162, 164–5
 data, 164^f
 data bus, 194
 daughter cards, 271
 Dawon Kahng, 41
 DDR, 210
 DDR2, 210
 DDR3, 210
 decimal (base-10) system, 70–1
 decoders, 129–30
 decryption (LFSR application), 415–16
 Deep Submicron (DSM), 148, 189
 de Forest, Lee, 35

- delamination, 269
 DeMorgan, Augustus, 106
 DeMorgan transformations, 106–7, 109–12
 depletion-mode MOSFET, 41, 41^f, 431–2
 depletion zones, 428, 429
 design and verification tools
 design capture, 361
 abstraction, higher levels of, 363
 gate-level netlist, 361–2
 graphical design entry lives on, 363–4
 schematic capture, 362–3
 transistor-level netlist, 361–2
 Design for Manufacturability (DFM), 370
 analog synthesis, 371–2
 Automatic Test Pattern Generator (ATPG), 376
 Built-In Self-Test (BIST), 375
 Electromagnetic Compliance (EMC), 374–5
 Electromagnetic Interference (EMI), 374–5
 fault simulation, 376
 hardware simulation
 acceleration and emulation, 372–3
 JTAG, 375
 mixed-signal simulation, 373
 physical verification, 373–4
 power analysis, 374
 RF/microwave design tools, 372
 SCAN, 375
 schematic synthesis, 371
 Signal Integrity (SI) analysis, 374
 thermal analysis, 374
 Design Rule Checking (DRC), 373
 Design Under Test (DUT), 373
 deuterium, 12
 device geometries, 188–90
 DFM *see* Design for Manufacturability
 diamond, 15^f, 331, 331^f
 substrates
 Chemical Vapor Deposition (CVD), 331–2
 Chemical Vapor Infiltration (CVI), 332
 maverick inventor, 333
 single-crystal diamond, requirements for, 333–4
 ubiquitous laser beams, 332–3
 die, 182, 182^f, 298–9, 298^f
 attachment, hybrids, 286–7
 populating, 304–5
 stacking, 185
 dielectric layer, 272, 280
 diffusion, 36
 layer, 180
 digital quantity, 4
 Digital Signal Processing, 162, 165, 166, 167–9
 Digital Signal Processor, 165, 168, 355
 Digital Signal Programming, 355
 digital simulation, 420^f
 Digital-to-Analog (D/A) converter, 162, 164–5
 digital versus analog system *see* analog versus digital system
 diminished radix complement, 89–90, 91
 DIMM (Dual In-Line Memory Module), 210–11
 diode, 35, 428–9
 see also semiconductor diodes
 DIP, 183, 183^f
 direct current (DC), 28
 Direct Rambus DRAM (DRDRAM), 210
 discrete components, 38
 versus Integrated Circuits (IC), 185–6
 disjunction, 100
 division in binary, 98
 Dodgson, Charles Lutwidge, 117–18
 don't care states, 123, 127, 128, 158
 dopants, 36
 doping, 36
 Double Data Rate (DDR), 210
 double plunge cuts, 281, 282
 double-sided boards, 262–4
 double-sided thick-film hybrids, 283
 down (quarks), 11^f
 dozen, 76
 drafting department, 346, 346^f
 DRAMs (Dynamic RAMs), 208, 242
 DRC (Design Rule Checking), 373
 DRDRAM (Direct Rambus DRAM), 210
 DSM (Deep Submicron), 148, 189
 D-type flip-flop, 139, 238
 positive-edge triggered implementation, 142
 D-type latches, 138–9
 Dual In-Line (DIL) package, 183, 183^f
 Dual In-Line Memory Module (DIMM), 210–11
 dual-port RAM, 200^f
 Ducas, Charles, 251
 Dummer, Geoffrey William Arnold, 173
 duo-decimal (base-12) systems, 71–3
 DUT (Design Under Test), 373
 DVDs, 161^f
 dynamic flex, 274
 Dynamic RAMs (DRAMs), 208
 dynner, 79^f

E

e (verification languages), 358
e-beam lithography *see* electron beam lithography
ECA-64, 312, 313
ECC memory (Error-Correcting Code memory), 211
echo, 165
ECL (Emitter-Coupled Logic), 187
EDA (Electronic Design Automation), 246
edge-sensitive, 139
EDIF (Electronic Design Interchange Format), 353
Edison, Thomas Alva, 35
Edison Effect, 35
EDO (Extended Data Out), 208
Edward George Bulwer-Lytton, 3^s
EEPROM/E²PROM (Electrically-Erasable Read-Only Memory), 207, 217
electrical impedance *see* impedance
Electrically-Erasable Read-Only Memory (EEPROM/E²PROM), 207, 217
Electrical Rule Checking (ERC), 373
electricity, 33
electric vs. electronic circuit, 47
electroless plating, 256, 257
Electromagnetic Compliance (EMC), 374–5
Electromagnetic Interference (EMI), 375
electromagnetic transistor
 fabrication, 324–5
electromechanical relay, 33–4
electromigration, 190, 325
electron(s), 11, 12
 shells, 13
Electron Beam Epitaxy (EBE), 327^f
electron beam lithography, 191
Electronic Design Automation (EDA), 246, 347
 origins of, 345–6
 Computer-Aided design (CAD), 346, 346^f, 346^{fp}
 Computer-Aided Engineering (CAE), 346–7, 347^{fp}
 designers versus engineers, 347
Electronic Numerical Integrator And Calculator (ENIAC), 35
electronics, 33
Electronic System Level (ESL), 359–60

electronic vs. electric circuit, 47
Elemental Computing Arrays (ECAs), 310–14
Elements, 310
EMACS, 363
embedded systems, 348–9
EMC (Electromagnetic Compliance), 375
EMI (Electromagnetic Interference), 374–5
emitter, 39
Emitter-Coupled Logic (ECL), 187
Emory University, 342
emulsion-coated fine steel mesh, 279
enable, 130
 input, 138
encapsulation, 273
encryption (LFSR application), 415–16
end-around-carry operation, 90
engineers versus designers, 347
enhancement-mode MOSFETs, 432–3
ENIAC (Electronic Numerical Integrator And Calculator), 35
enzymes, 337^f
EPROM (Erasable Programmable Read-Only Memory), 205–7, 217
equality comparators, 126–7
equivalency
 checker, 365, 367
 checking, 365
equivalent gate, 188
Erasable Programmable Read-Only Memory (EPROM), 205–7, 217
ERC (Electrical Rule Checking), 373
Ernest Nagy, 333
Ernest Nagy de Nagybaczon, 333
Ernst Werner von Siemens, 30
Error-Correcting Code memory (ECC memory), 211
ESL (Electronic System Level), 359–60
etching, 178, 254
Euler, Leonhard, 117
eutectic bond, 287, 287^f
eutectic solder, 304
EUV (Extreme Ultraviolet), 191
exa, 31
exclusive-OR, 53

exponent, 83
Extended Data Out (EDO), 208
Extreme Data Rate (XDR) DRAM, 210
ExtremeUltraviolet (EUV), 191

F

fab-less semiconductor company, 247
fabrication process, 175–80, 181, 334
“fabric”, 229, 229^f
fabs, 247
Faggin, Federico, 175
Fairchild and Texas Instruments, 174
Fairchild Semiconductor, 174, 175, 235, 236
falling-edge, 139
FALSE and TRUE functions
 versus OPEN and CLOSED functions, 50–1
families, 187
fan-in via, 265^f
fan-out via, 265
Faraday, Michael, 22^f
farads, 22, 22^f
Fast Page Mode (FPM), 208
fault simulation, 376
Federico Faggin, 175
feedback, 133
femto, 30, 31
Ferroelectric RAM (FRAM), 211–12
FETs (Field-Effect Transistors), 352, 432, 433
Feynman, Richard, 337
fiber-optic interconnect, 314–16
field-effect, 42, 431
Field-Effect Transistors (FETs), 352, 432, 433
Field Programmable Gate Arrays (FPGAs), 168, 169, 227–9, 308, 372
 architectures, 229–32
 configuration technologies, 232–3
FIFO (First-In First-Out)
 applications, 397, 411–14
fingers, 67
Finite State Machine (FSM) *see* state machines
firmware, 308
first distributive rule, 105, 106
First-In First-Out (FIFO)
 applications, 397, 411–14

five-quanta digital system, 9
 FLASH, 195^f, 207–8, 217, 232
 Fleming, Sir John Ambrose, 35
 flex, 274
 flexible printed circuits (FPCs), 274–6
 dynamic flex, 274
 rigid flex, 274, 275
 static flex, 274, 275
 flipped-chip technique, 184, 273, 289–90, 318, 334
 versus wire bonds, 293–4
 flipped TAB, 290
 floorplan, 366
 footprint, 259, 289
 formal verification, 359, 365–6
 foundry, 244, 247
 four-valence semiconductor *see* silicon
 FPCs *see* flexible printed circuits
 FPGAs (Field Programmable Gate Arrays), 168, 169, 227–9, 308, 372
 architectures, 229–32
 configuration technologies, 232–3
 FPM (Fast Page Mode), 208
 FR4, 252
 FRAM (Ferroelectric RAM), 211–12
 free-space interconnect, 317, 318
 FSM (Finite State Machine) *see* state machines
 full custom ASICs, 236, 249
 functional, in level of abstraction, 353–4, 356
 functional tester, 417, 418
 functional verification (stimulation), 364–5
 fusible-link technologies, 205, 214–15
 fuzz-buttons, 304

G

GaAs (gallium arsenide), 44–5, 46, 187, 325, 327
 GAL (Generic Array Logic) devices, 224
 gallium (Ga), 45
 gallium arsenide (GaAs), 44–5, 46, 187, 325, 327
 gallium phosphide, 46
 gate arrays, 236–8, 248
 design flow, 238–40
 gate-level netlist, 238, 351, 353, 361–2, 371, 372

gates versus functions, 56
 Gaussian implementations, 439
 gawble, 79^f
 GDSII, 370, 371, 373–4
 Generic Array Logic (GAL) devices, 224
 Geoffrey William Arnold Dummer, 173
 Georg Bednorz, 336
 George Boole, 99, 100
 George Thomson, 21
 Georgia Institute of Technology, 342
 Georg Simon Ohm, 19^f
 germanium (Ge), 15, 36, 326, 327
 giga, 30, 31, 196–7
 “gizmo”, 151, 152, 159
 glass substrate, 283, 298
 glue languages, 355
 glue logic, 248
 GPU (Graphics Processing Unit), 167
 graphical design entry, 363–4
 Graphics Processing Unit (GPU), 167
 gray code counters, 397, 398, 399, 404
 gray codes, 118, 393, 393^f
 versus binary codes, 399
 binary-to-gray, 395–6
 generation of, 394–5, 397–8
 gray code counters, 397
 gray-to-binary, 395–6
 sub-2ⁿ sequences generation, 398
 adjacent pairs, throwing, 399–400
 with consecutive values, 402–6
 mirroring process, 400–1
 pruning the ends, 401–2
 gray-to-binary converter, 395–6
 gross, 76
 ground planes, 267–70
 guard conditions, 153
 guided probe analysis, 418
 guided-wave interconnect, 318–19, 320
 gumbo, seafood, 459–63

H

Hanson, Albert, 251
 hard macros, 233, 239, 240
 hardware, 307
 Hardware Description Languages (HDLs), 350, 351, 355–8, 363, 365

hardware design versus programming languages, 349–50
 hardware simulation Acceleration and emulation, 372–3
 Hardware Verification Languages (HVLs), 358
 Harvard Mark I, 34
 HDI (high density interconnect), 270–1
 HDLs (Hardware Description Languages), 350, 351, 355–8, 363, 365
 Heike Kamerlingh Onnes, 336
 Heiman, Fredric, 41
 helium atom, 11, 12, 14, 14^f
 Henry, Joseph, 23^f
 herding wild electrons, 33
 hermetically sealed hybrid package, 291, 292
 heterojunction transistors, 325–7, 328
 Hewlett Packard (HP), 28
 hexadecimal (base-16) system, 80–1
 high density interconnect (HDI), 270–1
 high energy injection *see* hot electron injection
 high-impedance, 131
 high-speed design, 372
 High-Temperature Cofired Ceramics (HTCC), 301
 Hindus, 70
 H.J. Round, 45
 Hoerni, Jean, 41, 174
 Hoff, Ted, 175
 Hofstein, Steven, 41
 holes, 37
 versus vias, 264–5
 homojunction, 326
 hot electron injection, 205
 HTCC (High-Temperature Cofired Ceramics), 301
 HVLs (Hardware Verification Languages), 358
 hybrids, 293
 FPGAs, 308–9
 packaging process, 290–2
 substrates, 277–8
 thick-film process, 278
 capacitors creation, 282
 double-sided thick-film hybrids, 283
 inductors creation, 282–3

hybrids (*continued*)
 laser trimming, 281–2
 resistors creation, 280–1
 Subtractive Thick Film (STF) technology, 283
 tracks creation, 279–80
 thin-film hybrids, 283
 advantages, using bare die, 290
 assembly process, 286
 die attachment, 286–7
 flipped-chip techniques, 289–90
 laser trimming, 285–6
 Tape Automated Bonding (TAB), 288
 wire bonding, 287
 hydrogen atom, 12
 hydrogen bond, 338
 hydrogen molecule, 14

I
 ICs *see* integrated circuits
 idempotent rules, 102, 103
 IDMs (Integrated Device Manufacturers), 247
 IGFETs (Insulated Gate Field-Effect Transistors), 41^f
 impedance (*Z*), 28–9, 30
 inclusive-OR, 53
 inductance, 23–7
 inductors, 23–7, 28
 creation, 282–3
 inert gas *see* noble gas
 input/output cells and pads, 245–6
 Insulated Gate Field-Effect Transistors (IGFETs), 41^f
 insulators, 18, 36, 37, 336
 In-System Programmable (ISP) devices, 207, 232
 integrated circuits (ICs), 173^{fp}
 ASICs, 235
 Complementary Metal-Oxide Semiconductor (CMOS), 187
 core supply voltages, 187
 device geometries, 188–90
 versus discrete components, 185–6
 Emitter-Coupled Logic (ECL), 187
 equivalent gates, 188
 fabrication process, 175–80
 Moore's law, 192
 optical lithography, 190–1
 packaging process, 181–5

transistors, 192
 Transistor-Transistor Logic (TTL), 187
 types of, 186
 Integrated Device Manufacturers (IDMs), 247
 Intel, 175, 192, 205
 intellectual property (IP), 240–1
 intense electromagnetic field, 25
 International System of Units, 21, 196^f
 interposer, 294–5
 INV function, 51^f
 involution rule, 103
 I/O cells and pads, 245–6
 IP (intellectual property), 240–1
 isotopes, 12
 ISP (In-System Programmable) devices, 207, 232

J
 Jack St. Clair Kilby, 173, 174
 James Watt, 19^f
 Japan Electronic Industry Development Association (JEIDA), 356
 Java Math Library, 438
 Java™, 354
 Jean Hoerni, 41, 174
 JEIDA (Japan Electronic Industry Development Association), 356
 jelly bean devices, 174, 248
 JFETs (Junction Field-Effect Transistors), 41^f, 429–31
 JK and T flip-flops, 143
 John Bardeen, 39
 John Scott Russell, 317
 John Venn, 117
 John von Neumann, 168^f
 John Wilder Tukey, 78
 Joint Test Action Group (JTAG), 375, 375^f
 Joseph Henry, 23^f
 JTAG (Joint Test Action Group), 375, 375^f
 JUGFETs *see* Junction Field-Effect Transistors
 Julius Edgar Lilienfield, 41, 431
 jumpers, 262, 263
 Junction Field-Effect Transistors (JFETs), 41^f, 429–31
 junction region, 428^f

K
 Kahng, Dawon, 41
 Karnaugh, Maurice, 118, 119
 Karnaugh Map, 389, 393, 393^f
 Carroll, Lewis, 117–18
 incompletely specified functions, 122–3
 Karnaugh, Maurice, 118, 119
 Marquand, Allan, 117
 minimization, 119–20, 128, 128^f
 minterms, grouping, 120–2
 Tree of Porphyry, 117
 using 0s versus 1s, 123–4
 Venn, John, 117
 Kelvin, 336, 336^f
 Kfir, Alon, 157^f
 Kilby, Jack St. Clair, 173, 174
 kilo, 30, 31, 196–7
 kipper, 306^f

L
 laminates, 252, 293, 297
 land grid array *see* ball grid array
 Large Scale Integration (LSI), 188
 laser trimming
 thick-film hybrids, 281–2
 thin-film hybrids, 285–6
 latch-up conditions, 159
 lateral thermal conductivity, 277, 278
 Lattice Semiconductor Corporation, 224
 layout (place-and-route), 367
 designer, 246, 347
 Layout Versus Schematic (LVS), 373–4
 LCD (Liquid Crystal Display), 47, 329
 LDRs (light-dependent resistors), 20^f
 Lead Through Hole (LTH), 184, 259
 Least-Significant Bit (LSB), 87, 414
 LEDs (light-emitting diodes), 45–6, 45^{fp}
 Lee de Forest, 35
 Leonhard Euler, 117
 Leon O. Chua, 28
 level-sensitive, 139
 Lewis Carroll, 117–18
 LFSRs *see* Linear Feedback Shift Registers
 light-dependent resistors (LDRs), 20^f
 light-emitting diodes (LEDs), 45–6, 45^{fp}

- Lilienfield, Julius Edgar, 41, 431
L
 Linear Feedback Shift Registers (LFSRs), 147, 309, 407^f, 422
 accessing previous value, 415
 Built-In Self-Test (BIST)
 applications, 419–20, 419^f
 Cyclic Redundancy Check (CRC)
 applications, 416–17
 data compression applications, 417–18
 decryption application, 415–16
 encryption application, 415–16
 First-In First-Out (FIFO)
 applications, 411–14
 many-to-one implementations, 407–9
 one-to-many implementations, 410
 pseudo-random number
 applications, 420–1
 seeding, 410–11
 sequence through 2^n values,
 modified to, 414–15
 tapes for, 409
 Liquid Crystal Display (LCD), 47, 329
 literal, 156, 221
 logic 0/logic 1, combining single variable with, 102
 logical addition, 221
 logical/arithmetic operations, of digital function, 383
 logical multiplication, 221
 logic functions, 56, 100–1
 logic gates, 56, 125
 logic simulator, 420^f, 346
 logic synthesis, 366–7
 Look-Up Tables (LUTs), 226, 230, 231, 234
 Losov, O.V., 45
 loudspeaker, 164
 low-fired cofired processes, 301
 Low-Temperature Cofired Ceramics (LTCC), 301
 LSB (Least-Significant Bit), 87, 414
 L-shaped cuts, 281, 282
 LSI (Large Scale Integration), 188
 LTCC (Low-Temperature Cofired Ceramics), 301
 LTH (Lead Through Hole), 184
 lucky numbers, 84
 LUTs (Look-Up Tables), 226, 230, 231, 234
 LVS (Layout Versus Schematic), 373–4
M
 MAC (Multiply-And-Accumulate) units, 168
 macro-cells *see* hard macros
 macro-functions *see* soft macros
 macros
 hard macros, 239
 soft macros, 239
 in software terms, 239^f
 Magnetic Tunnel Junction (MTJ), 211
 Magnetoresistive Random Access Memory (MRAM), 211
 mapping
 hexadecimal to binary, 81
 octal to binary, 81
 physical to logical mapping
 NMOS logic, 384–5
 PMOS logic, 386–7
 Marquand, Allan, 117
 Martin M. Atalla, 41
 mask-programmed ROMs, 202–3
 master-slave relationship, 142
 MATLAB®, 438
 Matrix, 312
 Maurice Karnaugh, 118, 119
 maverick inventor, 333
 maxterms and minterms, 112
 Mayans, 76
 Mazor, Stan, 175
 MBE (Molecular Beam Epitaxy), 326, 327^f
 McKinley microprocessor, 192
 M-Code, 354, 354^f
 Mealy machine, 154, 155
 Medium Scale Integration (MSI), 188
 mega, 30, 31, 196–7
 MegaPAL, 225
 memory cells, 195–6
 memory elements, 132
 memory ICs, 193
 arrays, 195–6
 bits and bytes, 197
 CBRAMs, 211–12
 cells, 195–6
 DDR, 210
 DDR2, 210
 DDR3, 210
 DIMMs, 210–11
 DRAMs, 208
 ECC memory, 211
 EPROMs, 205–7
 EEPROMs/E²PROMs, 207
 FLASH, 207–8
 FRAMs, 211–12
 giga, 196–7
 kilo, 196–7
 mask-programmed ROMs, 202–3
 mega, 196–7
 MRAMs, 211
 nvRAMs, 211–12
 PRAMs, 211–12
 PROM, 203–4
 QDR, 210
 RAMBUS, 210
 Random-Access Memories (RAMs), 193–5
 with data in and data out
 busses, 199–200
 with single bidirectional bus, 200–1
 Read-Only Memory (ROM), 193–5
 control decoding, 197–8
 RIMMs, 210–11
 RRAMS, 211–12
 SDRAMs, 208–10
 SIMMs, 210–11
 SONOS, 211–12
 SRAMs, 208
 tera, 196–7
 width and depth, increasing, 201–2
 word addressing in, 196
 words, 195–6
 memory module, 210
 memory resistor *see* memristor
 MEMory Unit (MEMU), 311, 312
 memristance, 28
 memristor, 28
 MEMS (microelectromechanical systems), 341, 342
 MEMU (MEMory Unit), 311, 312
 mesa process, 40
 MESFETs (Metal-Epitaxial Semiconductor Field-Effect Transistors), 41, 41^f, 429, 430–1
 Metal-Epitaxial Semiconductor Field-Effect Transistors (MESFETs), 41, 41^f, 429, 430–1
 metallization layers, 180
 metal-oxide, 42
 metal-oxide semiconductor field-effect transistors (MOSFETs), 41–3, 41^f, 57, 187

metal substrates, 298
 metastable condition, 137
 Michael Faraday, 22^f
 micro, 31
 microelectromechanical systems (MEMS), 341, 342
 Micromosaic, 174, 235
 microprocessor (μ P), 175
 microvia technologies, 270–1
 milli, 20^s, 31
 minimization, 106
 using Karnaugh Maps, 119–20
 minterms
 grouping, 120–2
 and maxterms, 112
 mirroring process, 395
 mirror line, 402, 403
 mixed-signal FPGAs, 233
 mixed-signal simulation, 373
 model checking, 365
 modulus, of the counter, 146
 mohs, 30
 Molecular Beam Epitaxy (MBE), 326, 327^f
 molecules, 11, 13–15
 combining, 339
 organic, 323
 monolithic integrated circuit, 173
 Moorby, Phil, 356
 Moore machine, 154, 155
 Moore's law, 192
 MOSFETs (metal-oxide semiconductor field-effect transistors), 41–3, 41^{fp}, 57, 187
 Most Significant Bit (MSB), 414, 414^f, 87, 202
 motherboards, 271–2
 MRAM (Magnetoresistive Random Access Memory), 211
 MSB (Most Significant Bit), 414, 414^f, 87, 202
 MSI (Medium Scale Integration), 188
 MTJ (Magnetic Tunnel Junction), 211
 MULt (MULTiplier), 310
 multilayer boards, 265–6
 multiplexer (MUX)
 Multiply-And-Accumulate (MAC) units, 168
 multi-value digital systems, 5–6
 MUX (multiplexer), 127–9, 230, 234, 238
 Müller, Alex, 336

N

Nagy, Ernest, 333
 Nakamura, Shuji, 46
 NAND gate/function, 53, 54–5, 60, 101, 132, 389
 transistor implementation, 61
 nano, 31, 190
 nanobots, 340
 nanophasé diamond structures, 333
 nanophasé materials, 333
 nanotechnology, 337–41
 nanotubes, 328–30
 NC drilling machine, 257^f
 n-channel, 48, 49
 negative-edge, 139
 negative hold times, 148
 negative ion, 12
 negative logic, 57, 383
 negative logic convention, 384–5, 385^f
 negative logic implementations, 377, 377^f
 negative numbers, origin of, 74–5
 negative quantity, 94
 negative-resist process, 178, 254^f
 negative setup, 148
 negative signed binary numbers, 94–5
 negative-true *see* negative logic
 netlists, 347
 component-level netlist, 351
 gate-level netlist, 351
 transistor-level netlist, 350–1
 neutrons, 11, 12
 nibble, 79
 nichrome, 283
 nines' complement *see* diminished radix complement
 NMOS transistor, 42, 43^{fp}, 57, 384–5
 noble gas, 14^f
 noble metals, 300^f
 nonvolatile RAMs (nvRAMs), 211–12
 NOR gate/function, 53, 54^{fp}, 54–5, 61–2, 131, 132, 133, 389
 notches on bones, 67–8
 NOT gate/function, 55, 58, 101, 389
 and BUF function, 51–2
 in signal inverting, 455
 transistor implementation, 59
 Noyce, Robert, 41, 174
 NPN, 39, 39^f
 N-type diffusions region, 179, 180
 N-type silicon, 36, 37, 38, 427, 428, 429, 430, 431, 432

nucleus, 11

number systems, 67
 abacus, 69
 Acanthostega, 82
 binary (base-2), 78–9
 bones with notches, 67–8
 decimal (base-10), 70–1
 duo-decimal (base-12), 71–3
 fingers, toes, and pebbles, 67
 hexadecimal (base-16), 80–1
 lucky and unlucky numbers, 84
 octal (base-8), 80–1
 powers, use of, 82–4
 quinary (base-5), 77, 78
 Roman Numerals, 69–70
 sexagesimal (base-60), 73–4
 tally sticks, 68
 tertiary logic, 85
 tetrapods, 82
 time-travelers, jobs abound for, 76–7
 vigesimal (base-20), 76
 zero and negative numbers, 74–5

Nutmeg, 352

nvRAMs (Nonvolatile RAMs), 211–12

nybble, 79

O

octal (base-8) system, 80–1
 Ohm, Georg Simon, 19^f
 Ohms, 19, 19^f
 Ohm's law, 20
 Ohm's triangle, 20
 OLEDs (organic LEDs), 46–7
 omega (Ω), 20^s, 30, 30^f
 OMEMS (optical microelectromechanical systems), 341, 342
 one-hot encoding, 158
 ones' complement, 91
 One-Time Programmable (OTP) devices, 204, 215, 217
 Onnes, Heike Kamerlingh, 336
 OPC (Optical Proximity Correction), 191, 370
 OPEN and CLOSED functions
 FALSE and TRUE functions, 50–1
 OpenVera Assertions (OVA), 359
 OpenVera™, 359
 Open Verification Library (OVL), 359
 Operating System (OS), 195

- optical backplanes, 316
 optical interconnect
 fiber-optic interconnect, 314–16
 free-space interconnect, 317, 318
 guided-wave interconnect, 318–19, 320
 optical lithography, 175, 190–1
 optical memories, 320–1
 optical microelectromechanical systems (OMEMS), 341, 342
 Optical Proximity Correction (OPC), 191, 370
 organic LEDs (OLEDs), 46–7
 organic substrates, 252
 organic synthetic metals, 336
 OR gate/function
 pass-transistor implementation, 424
 OR gates/function, 49, 50, 52–3, 56, 61–2, 100–1, 110, 125, 131, 132, 389, 390, 413, 424
 OS (Operating System), 195
 OTP (One-Time Programmable) devices, 204, 215, 217
 Ouroboros, 407^f
 OVA (OpenVera Assertions), 359
 overglossing, 181
 OVL (Open Verification Library), 359
 O.V. Losov, 45
 oxygen atom, 14
- P**
- Package-in-Package (PiP), 297
 Package-on-Package (PoP), 297
 packaging process, 181–5
 hybrids, 290–2
 padcap, 270
 Pad Grid Arrays (PGAs), 260^f, 302
 pads, 181, 257
 pad-stack, 269^f
 PAL (Programmable Array Logic), 223, 223^f, 224
 Parallel-In-Serial-Out (PISO), 146
 parasitic extraction, 367–8
 partial product, 98
 passivation layer, 181
 passive¹² electronic components, 28
 passive trimming, 281
 passive vs. active devices, 47
 pass-transistor logic
 implementation, 65, 423–6
 AND gate, 423
- D-type latch, 425
 OR gate, 424
 XNOR gate, 425
 XOR gate, 424
 Paul Clifford, 3^s
 p-channel, 48, 49
 pebbles, 67
 Perl, scripting language, 355
 peta, 31
 PGAs (Pad Grid Arrays), 260^f, 302
 Phase-Change Memory (PCM/
 PRAM/PCRAM), 211–12
 Phase Shift Mask (PSM), 191, 370
 PHB (photochemical hole-burning), 320–1
 Phil Moorby, 356
 phosphorus atoms (P), 36, 37
 phosphorus-doped silicon, 36
 photochemical hole-burning (PHB), 320–1
 photo-imagable polyimide
 interconnect, 319, 320
 photo-mask, 175
 pick-and-place machine, 261
 pico, 31
 pin grid arrays, 302, 303
 PiP (Package-in-Package), 297
 PISO (Parallel-In-Serial-Out), 146
 PISO (Parallel-In-Serial-Out), 146
 place-and-route tools, 239
 place engine/placer, 238
 place-value number systems, 69, 70
 planar process, 41, 174
 PLAs (Programmable Logic Arrays), 221–3
 plasma, 326^f
 plated through-hole (PTH), 259, 265, 269, 272
 plating (in America), 257
 playte, 79^f
 PLDs (Programmable Logic
 Devices), 217
 plunge cuts, 281, 282
 PMOS transistor, 42, 43^{fp}, 57, 386–7
 P-N junctions, 38, 46, 427–9
 PNP, 39, 39^f
 polycrystalline silicon, 42, 179, 431
 polysilicon, 216
 layers, 180
 PoP (Package-on-Package), 297
 populating circuit boards, 259
 positive-edge, 139, 142, 144, 147
 positive ion, 12
 positive logic convention, 57, 383, 384, 384^f
 positive-resist process, 178, 254
 positive signed binary numbers, 94–5
 positive versus negative logic, 383
 NMOS logic, 384–5
 PMOS logic, 386–7
 potential links, 213
 potentiometers, 20^f
 power, 19
 and ground planes, 267–70
 power analysis, 374
 power planes, 267–70
 powers, representing numbers using, 82–4
 Practical Extraction and Report Language, 355
 precedence of operators, 105
 prepreg, 266
 primitive gates, 56
 Primitive Logic Functions, 49, 55–6,
 100–1
 AND, OR, and XOR functions, 52–3
 AND and OR functions
 switch representations, 49–50
 BUF and NOT functions, 51–2
 “connect the NOTs”, 52
 FALSE and TRUE versus OPEN
 and CLOSED, 50–1
 NAND, NOR, and XNOR
 functions, 53–5
 simple functions versus gates, 56
 primitive logic gate, 238
 primitives, 56
 printed circuit boards (PCBs), 251, 351
 additive process, 255–7
 backplanes, 271–2
 blind vias, 266–7
 buried vias, 266–7
 chip-on-board (COB), 273–4
 conductive ink technology, 272–3
 double-sided boards, 262–4
 flexible printed circuits (FPCs), 274–6
 ground planes, 267–70
 high density interconnect (HDI), 270–1
 holes versus vias, 264–5
 lead through-hole (LTH), 259
 microvia technologies, 270–1
 motherboards, 271–2

- printed circuit boards (PCBs)
(continued)
 multilayer boards, 265–6
 power planes, 267–70
 and PWBS, 252
 RoHS and lead-free solder, 252–3
 single-sided boards, 257–9
 subtractive process, 253–5
 surface mount technology (SMT),
 260–2
 through-hole vias, 266–7
 wave soldering, 259–60
 printed wire boards (PWBS), 252
 process node, 189
 product-of-sums forms, 389
 and sum-of-products form,
 112–13, 114
see also canonical forms
 product term, 221
 sharing, 223
 Programmable Array Logic (PAL),
 223, 223^f, 224
 programmable ICs
 additional programmable options,
 224
 antifuse technologies, 215–17
 Complex PLDs (CPLDs), 224–7
 Customer-Specific Standard
 Products (CSSPs), 233
 E²PROM technology, 217
 EPROM technology, 217
 Field Programmable Gate Arrays
 (FPGAs), 227–33
 FLASH technology, 217
 fusible-link technologies, 214–15
 Generic Array Logic (GAL) devices,
 224
 mixed-signal FPGAs, 233
 Programmable Array Logic (PAL),
 223, 223^f, 224
 Programmable Logic Arrays
 (PLAs), 221–3
 Programmable Logic Devices
 (PLDs), 217
 Programmable Read-Only
 Memories (PROMs), 218–21
 simple programmable function,
 213
 SRAM technology, 217
 programmable interconnect matrix,
 225, 226
 Programmable Logic Arrays (PLAs),
 221–3
 Programmable Logic Devices
 (PLDs), 217
- programmable multiplexer, 226, 227
 Programmable Read-Only Memory
 (PROM), 203–4, 203^f,
 218–21
 programming languages, 349,
 354–5
 “programming the device”, 215
 programming versus hardware
 design languages, 349–50
 PROM (Programmable Read-Only
 Memory), 203–4, 203^f,
 218–19
 Property Specification Language
 (PSL), 359
 proposition, 99, 100
 protein switches and memories,
 321–4
 protons, 11, 12
 pseudo-random number
 applications, 420–1
 PSL (Property Specification
 Language), 359
 PSM (Phase Shift Mask), 191, 370
 PTH (plated through-hole), 259,
 265, 269, 272
 P-type diffusions region, 180
 P-type silicon, 36, 37, 427, 428
 mixing, 38
 PWBS (printed wire boards), 252
- Q**
 QDR (Quad Data Rate), 210
 QFP (quad flat pack), 260
 Quad Data Rate (QDR), 210
 quad flat pack (QFP), 260
 quanta, 5
 quantization, of analog signal, 163,
 164
 quantization error, 164
 quantization noise, 164
 quantum levels, 13
 quarks, 11^f
 Queen Victoria, 33
 quinary (base-5) system, 77, 78
- R**
 Radio Frequency (RF), 372
 radix-10, 70
 radix complement, 89–90
 RAMBUS, 210
 RAMs *see* Random-Access Memories
- Random-Access Memories (RAMs),
 193–5
 with separate data in and data out
 busses, 199–200
 with single bidirectional bus,
 200–1
 random rounding *see* round-random
 algorithm
 RCA research laboratory, 41
 RC time constant, 23
 reactance, 28–9, 30
 Read-Only Memory (ROM), 193–5
 control decoding, 197–8
 Read-Write Memory (RWM), 193,
 193^f
 read-write, 200
 real estate, 262
 reconfigurable computing, 307–10
 redundant variables, 391, 391^f
 Reed-Müller implementation
 2-input function for, 389
 3-input function for, 390
 4-input function for, 390
 Reed-Müller logic, 389–92
 refractory metals, 301^f
 refreshing, 208
 register, 139
 Register Transfer Level (RTL), 353,
 360, 363, 363^f, 366, 370
 reset condition, 134
 reset inputs, 132, 133, 134, 135, 136
 Reset-Set latch (RS latch)
 NAND-based implementation,
 137–8
 NOR implementation, 132–7
 resistance (R), 18–19, 29
 and resistors, 19–21
 Resistive Random Access Memory
 (RRAM), 211–12
 resistor-capacitor-switch circuit, 23
 voltage and current characteristics,
 24
 resistor-inductor-switch circuit, 26
 voltage and current characteristics,
 26
 resistor-NMOS transistor circuit, 44
 resistors, 28
 and resistance, 19–21
 resistors creation, 280–1
 thick-film hybrids, 280–1
 resistor-switch circuit, 43
 Resolution Enhancement Techniques
 (RET), 191, 370
 Restriction of Hazardous Substances
 Directive (RoHS)

results gatherer, 419
 RET (Resolution Enhancement Techniques), 370
 reverb, 165
 RF (Radio Frequency), 372
 RF/microwave design tools, 372
 Rhodopsin, 324
 Richard Feynman, 337
 rigid flex, 274, 275
 RIMMs, 210–11
 rising-edge, 139
 Robert Noyce, 41, 174
 Robert Sheckley, 342
 RoHS (Restriction of Hazardous Substances Directive), 253
 ROM *see* Read-Only Memory
 Roman numerals, 69–70
 Round, H.J., 45
 round-alternate algorithm, 445
 round-away-from-zero algorithm, 443–4
 round-ceiling algorithm, 436, 441
 round-down algorithm, 444
 round-floor algorithm, 436, 442
 round-half-down algorithm, 436, 439
 round-half-even algorithm, 437, 439–40
 round-half-odd algorithm, 437, 440–1
 round-half-up algorithm, 436, 437–9, 448–9, 452
 rounding algorithms, 164
 rounding algorithms, 101, 435
 round-alternate, 445
 round-away-from-zero, 443–4
 round-ceiling, 441
 round-down, 444
 round-floor, 442
 round-half-down, 439
 round-half-even, 439–40
 round-half-odd, 440–1
 round-half-up, 437–9
 rounding sign-magnitude binary values, 445
 round-half-up, 448–9
 round-half-up, 452
 rounding signed binary values, 449
 truncation, 447–8, 450–2
 round-random, 445
 round-toward-nearest, 436
 round-toward-zero, 443
 round-up, 444
 truncation, 444

round method, 438
 round-random algorithm, 445
 round-toward-nearest algorithm, 436, 437
 round-toward-zero algorithm, 443
 round-up algorithm, 444
 route engine/router, 239
 RRAM (Resistive Random Access Memory), 211–12
 RS latch
 NAND-based implementation, 137–8
 NOR-based implementation, 132–7
 R. Stanley Williams, 28
 RTL (Register Transfer Level), 353, 360, 366, 370
 rubber, 18
 Russell, John Scott, 317
 RWM (Read-Write Memory), 193, 193^f

S

SALU (Super ALU), 311
 sampling, of analog signal, 163
 scalar entity, 125
 scalar versus vector notation, 125–6
 scaling, 326
 SCAN, 375
 scan-in pin, 375
 scan-out pin, 375
 schematic capture, 362–3
 schematic synthesis, 371
 score, 76
 scripting languages, 355
 scrubbing, 287
 SDR (Single Data Rate), 210
 SDRAM (Synchronous DRAM), 208–10, 242
 seafood gumbo, 459–63
 sea-of-gates/sea-of-cells, 238
 second distributive rule, 105–6, 107
 seed values
 LFSRs, 410–11
 “self-aligning”, 304
 semiconductor diodes, 37–9
 semiconductors, 33, 36–7
 active vs. passive devices, 47
 BJTs, 39–41
 depletion-mode MOSFET, 431–2
 depletion zones, 428, 429
 diodes, 428–9
 electric vs. electronic circuit, 47
 electromechanical relay, 33–4

enhancement-mode MOSFETs, 432–3
 gallium arsenide semiconductors, 44–5
 herding wild electrons, 33
 Junction Field-Effect Transistors (JFETs), 429–31
 LEDs, 45–6
 Metal-Epitaxial Semiconductor Field-Effect Transistors (MESFETs), 429, 430–1
 MOSFETs, 41–3
 organic LEDs, 46–7
 p-n junctions, 427–9
 semiconductor diodes, 37–9
 transistor, as switch, 43–4
 vacuum tubes, 35
 semiconductor substrates, 298
 semi-custom devices, 240
 sensor, 162
 sequential function, 132
 Serial-In-Parallel-Out (SIPO), 144
 Serial-In-Serial-Out (SISO), 146
 set condition, 133, 134
 set inputs, 132, 133, 135, 136, 141
 sexagesimal (base-60) numbering system, 73–4
 shadow registers, 415
 Shannon, Claude Elwood, 100
 Scheckley, Robert, 342
 shift-and-add technique, 97–8
 shift registers, 144–6
 Shockley, William, 39, 429
 shriek character (“!”), 380–2
 Shuji Nakamura, 46
 siemens (S), 30
 signaling class elements, 311
 Signal Integrity (SI) analysis, 374
 signal layers, 267
 signature analysis, 418
 signatures, 418
 sign bit, 94
 signed binary numbers, 94–5, 436
 addition, 95–6
 subtraction, 96–7
 signed binary values, rounding, 449
 round-half-up, 452
 truncation, 450–2
 significant variables, 391, 391^f
 sign-magnitude binary numbers, 93
 sign-magnitude binary values, 445
 rounding, 445
 round-half-up, 448–9
 truncation, 447–8
 silica waveguides, 318, 319

- silicon (Si), 36, 44, 46, 325, 326
 silicon atom, 15
 silicon bumping, 181
 silicon chips, 173, 236
 Silicon-Oxide-Nitride-Oxide-Silicon (SONOS), 211–12
 Silonex, 283
 silver-loaded epoxy, 286
 SIMM (Single In-Line Memory Module), 210–11
 simple functions versus gates, 56
 Simple PLDs (SPLDs), 224, 225, 226
 simple programmable function, 213
 simplification rules, 106, 108
 Simulation Program with Integrated Circuit Emphasis (SPICE), 346, 346^f, 346^p
 simulators, 346
 single-crystal diamond, requirements for, 333–4
 Single Data Rate (SDR), 210
 Single In-Line Memory Module (SIMM), 210–11
 single-sided boards, 257–9
 sintering, 273
 SIPO (Serial-In-Parallel-Out), 144
 Sir John Ambrose Fleming, 35
 Sir Joseph Wilson Swan, 35^f
 SISO (Serial-In-Serial-Out), 146
 skin effect, 369
 Small Outline Package (SOP), 183
 Small Scale Integration (SSI), 188
 SME (state machine element), 311
 SMT (Surface Mount Technology), 184
 snoopy, 3
 SoC (System-On-Chip), 186, 233, 297
 soft macros, 239
 software, 308
 solder bumping *see* flipped-chip techniques
 solder grid array *see* ball grid array
 soldering, 253
 solder mask, 257
 solder mask over bare copper (SMOBC), 259
 solders, 252, 252^f
 solid-state electronics, 39
 soliton, 317
 SONOS (Silicon-Oxide-Nitride-Oxide-Silicon), 211–12
 SOP (Small Outline Package), 183
 SoPC (System-On-A-Programmable-Chip), 233
- space charge region, 428^f
 SPICE (Simulation Program with Integrated Circuit Emphasis), 346, 346^f, 346^p, 358
 SPICE Deck, 352
 spicules, 257
 spiral inductor, 282
 SPLDs (Simple PLDs), 224, 225, 226
 square spiral inductor, 282
 SRAF (Sub-Resolution Assist Features), 191
 SRAM-based FPGAs, 308
 SRAMs (static RAMs), 208, 232, 242
 SRAM technology, 217
 SSI (Small Scale Integration), 188
 SSTA (Statistical Static Timing Analysis), 369
 STA (Static Timing Analysis), 368–9
 standard cell devices, 240–1
 standard MOSFET versus EPROM transistors, 205
 Standard Ultraviolet (UV), 191
 standard versus assertion-level logic, 378
 Stan Mazor, 175
 state assignment, 155–8
 state diagram, 152–3
 state machine element (SME), 312
 state machines, 151, 154–5
 statement, 99
 state table, 153–4
 state transitions, 153
 state variables, 154
 static flex, 274, 275
 static formal verification, 366
 static RAMs (SRAMs), 208
 Static Timing Analysis (STA), 368–9
 Statistical Static Timing Analysis (SSTA), 369
 step-and-repeat process, 176
 step-down transformer, 27
 step-up transformer, 27
 Steven Hofstein, 41
 STF (Subtractive Thick Film) technology, 283
 stochastic rounding *see* round-random algorithm
 storage class elements, 311
 strange (quarks), 11^f
 stretch-resistant socks, 308
 structural, in levels of abstraction, 353
 “Structured ASICs”, 242–5
 advantages, 244
 designing, 244
- sub-2ⁿ sequences generation, 398
 adjacent pairs, throwing, 399–400
 mirroring method, 400–1
 pruning the ends, 401–2
 with consecutive values, 402–6
 subatomic erosion, 190
 sub-board, 266, 266^f
 Sub-Resolution Assist Features (SRAF), 191
 substrate(s), 177, 252
 hybrid, 277–8
 for System-in-Package, 297–8
 subtraction
 in signed binary numbers, 96–7
 in unsigned binary numbers, 91–3
 subtractive process, 253–5
 Subtractive Thick Film (STF) technology, 283
 sum, 88
 Sumerians, 71
 sum-of-products forms, 389
 and product-of-sums form, 112–13, 114
see also canonical forms
 Super-Cluster, 312
 superconductors, 335–7
 Superlog, 357
 surface emission displays (SEDs), 329–30
 surface mount devices (SMDs), 260
 surface mount technology (SMT), 184, 260–2
 susceptance (B), 29–30
 Swan, Sir Joseph Wilson, 35^f
 switch-level netlist, 352
 symbolic logic, 100
 Synchronous DRAM (SDRAM), 208–10, 242
 Synopsys, 359
 SystemC, 357–8
 System House, 247
 system-in-package (SiP), 293, 297, 305–6, 315, 318, 319, 331
 example, based on cofired ceramics, 298–305
 populating, 304–5
 substrates, positive plethora of, 297–8
 system integration languages, 355
 System-On-A-Programmable-Chip (SoPC), 233
 System-On-Chip (SoC), 186, 233, 297
 SystemVerilog 3.0, 357

T

TAB (Tape Automated Bonding), 288
 tally, 67
 tally sticks, 68
 TALU (Triple ALU), 311
 Tape Automated Bonding (TAB), 288
 tayble, 79^f
 tayne, 79^f
 technologies, 187
 technology node, 189
 Ted Hoff, 175
 ten's complement *see* radix complement
 tera, 30, 31, 196–7
 tertiary (base-3) values, 85
 tertiary logic, 85
 test generator, 419
 tetrapod, 82
 thermal analysis, 374
 thermal relief pads, 269, 269^f
 thermistors, 20^f
 thick-film hybrids, 278
 capacitors creation, 282
 double-sided thick-film hybrids, 283
 inductors creation, 282–3
 laser trimming, 281–2
 resistors creation, 280–1
 Subtractive Thick Film (STF) technology, 283
 tracks creation, 279–80
 thin-film hybrids, 283
 advantages, using bare die, 290
 assembly process, 286
 die attachment, 286–7
 flipped-chip techniques, 289–90
 laser trimming, 285–6
 Tape Automated Bonding (TAB), 288
 wire bonding, 287
 Thomas Alva Edison, 35, 251
 Thomson, George, 21
 three-dimensional silicon crystal, 15
 through-hole technique, 257, 259
 through-hole via, 266, 267
 TI, 236
 time-travelers, jobs abound for, 76–7
 timing analysis, 368
 Static Timing Analysis (STA), 368–9
 Statistical Static Timing Analysis (SSTA), 369
 tinning (in England), 257

TLMs (Transaction-Level Models), 358

toes, 67

top (quarks), 11^f

Toshiba, 330

tracking layers, 180

tracks, 179

 creation

 thick-film hybrids, 279–80

Transaction-Level Models (TLMs), 358

transducer, 162

transformers, 27

transistor-and-fusible-link-based PROM cell, 204

transistor-level netlist, 350–1, 352, 361–2

transistors, 33, 39, 192

 0s and 1s, use of, 57–8

 AND gate, 60, 61

 BUF gate, 58, 59–60

 CMOS, 57

 electromagnetic, 324–5

 heterojunction, 325–8

 for logic gates construction, 57

 NAND gate, 60, 61

 NMOS, 57

 NOR gates, 61–2

 NOT gate, 58–9

 OR gates, 61–2

 pass-transistor logic, 65

 PMOS, 57

 as switch, 43–4

 XNOR gates, 62–3

 XOR gates, 62–3

 pass-transistor

 implementations, 63–5

Transistor-Transistor Logic (TTL), 174, 187

Tree of Porphyry, 117

triode, 35

tri-state buffer, 131

tri-state functions, 130–2

tri-state gates, 198, 198^f, 201

trit, 85^f

tritium, 12

TRUE and FALSE functions

 versus OPEN and CLOSED

 functions, 50–1

true outputs, 132

truncation algorithm, 436, 444,

 447–8, 450–2

truth (quarks), 11^f

TTL (Transistor-Transistor Logic), 187

two-quanta digital system, 8

two's complement, 91–2

U

ubiquitous laser beams, 332–3

UDL/I (Unified Design Language for Integrated Circuits), 356

UDSM (Ultra-Deep-Submicron), 189

ULAs (Uncommitted Logic Arrays), 236^f

Ultra-Deep-Submicron (UDSM), 189

Ultra-Large-Scale Integration (ULSI), 188

Uncommitted Logic Arrays (ULAs), 236^f

unidirectional, 193

Unified Design Language for Integrated Circuits (UDL/I), 356

unit qualifiers, 30–1

University of Aberdeen, 21

University of California, Berkeley, 28, 328, 346

University of Illinois, 342

University of Pennsylvania, 35

unlucky numbers, 84

unsigned binary numbers, 87–8

 addition, 88–9

 subtraction, 91–3

unused states, 158–9

up (quarks), 11^f

U.S. Department of Defense (DoD), 356

V

vacuum tubes, 35, 35^f

valence bonds, 14

valence electrons, 14

valves, 35^f

vapor-phase soldering, 261, 289,

 289^f

vaporware, 308

variable resistors, 20^f

varistors, 20^f

VDRs, 20^f

vector, 125

Venn, John, 117

Venn diagrams, 117

Vera®, 358–9

verification languages, 358–9
 Verilog, 350, 356
 Verisity, 358
 Very High Speed Integrated Circuit (VHSIC) program, 356
 Very-Large Scale Integration (VLSI), 188
 VHDL, 350, 356
 VHSIC (Very High Speed Integrated Circuit) program, 356
 VHS tapes, 161
 VI (Visual Interface), 363
 vias, 180, 263, 263^f, 265, 298
 versus holes, 264–5
 hybrids, 278
 via-screen, 299
 vigesimal (base-20) system, 76
 Visual Interface(VI), 363
 VLSI (Very-Large Scale Integration), 188
 volatile, 194
 Volta, Count Alessandro, 19^f
 voltage, 18–19
 volts, 19, 19^f
von Neumann machines, 168^f
 von Siemens, Ernst Werner, 30
 von Zeppelin, Count Ferdinand, 41^f

W

Wafer-Level Chip-Scale Package (WL-CSP), 295
 wafer probing, 181

wafers, 175
 wagon wheel pattern, 269^f
 Walter Brattain, 39
 water clock, 72
 water molecule, 14
 water tank representation
 of capacitance, 22
 of voltage, current, and resistance, 18–19
 Watt, James, 19^f
 watts, 19, 19^f
 waveforms, 4, 5, 6
 wave soldering, 259–60
 weasel words, 361
 wells (in multichip modules), 301, 302
 wetware, 308^f
 Wilder, John Tukey, 78
 William of Ockham, 107^f
 Williams, R. Stanley, 28
 William Shockley, 39, 429
 Winston Churchill, 342
 wire bonding, 287
 wire bonds versus flip-chip
 technique, 293–4
 word, 79^f, 195–6

X

X (“don’t know”) states, 137, 140, 145
 X Architecture, 180^f
 “X” character, 87, 123

Xilinx, 227
 XNOR gate/function, 54, 54^{fp}, 54–562–3, 101, 126
 pass-transistor implementation, 63–5, 424
 XNORs
 even number of, 391
 odd number of, 392
 Reed-Müller implementations, 392
 XOR gate/function, 52^{fp}, 53, 62–3, 100–1, 456–7
 pass-transistor implementation, 424
 pass-transistor implementations, 63–5

Y

yocto, 31
 yotta, 31

Z

Z character, 131
 zepto, 31
 zero
 origin of, 74–5
 zetta, 31
 Zone, 312