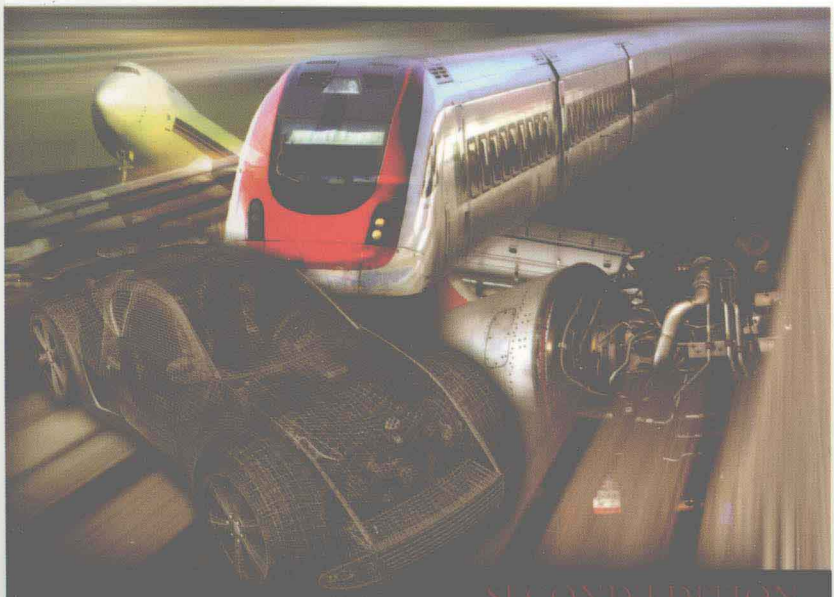
HZ BOOKS 华章教育
经 典 原 版 书 库
ELSEVIER

嵌入式计算系统设计原理

（英文版 · 第2版）

COMPUTERS AS COMPONENTS

PRINCIPLES OF EMBEDDED COMPUTING SYSTEM DESIGN



SECOND EDITION

WAYNE WOLF

MK

（美）Wayne Wolf 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

嵌入式计算系统设计原理

（英文版 · 第2版）

Computers as Components

Principles of Embedded Computing System Design

(Second Edition)

（美）Wayne Wolf 著

机械工业出版社
China Machine Press

Wayne Wolf: Computers as Components: Principles of Embedded Computing System Design, Second Edition (ISBN 978-0-12-374397-8).

ISBN: 978-981-272-293-5

Printed in China by China Machine Press under special arrangement with Elsevier (Singapore) Pte Ltd. This edition is authorized for sale in China only, excluding Hong Kong SAR and Taiwan.

本书版权登记号：图字：01-2008-4175

图书在版编目（CIP）数据

嵌入式计算系统设计原理（英文版·第2版）/（美）沃尔夫（Wolf, W.）著. —北京：机械工业出版社，2008.12
（经典原版书库）
书名原文：Computers as Components: Principles of Embedded Computing System Design, Second Edition

ISBN 978-7-111-25360-0

Ⅰ.嵌… Ⅱ.沃… Ⅲ.微型计算机－系统设计－英文 Ⅳ.TP36

中国版本图书馆CIP数据核字（2008）第161446号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）
责任编辑：迟振春
三河市明辉印装有限公司印刷·新华书店北京发行所发行
2009年1月第1版第1次印刷
170mm×242mm · 33印张
标准书号：ISBN 978-7-111-25360-0
定价：75.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章分社较早意识到“出版要为教育服务”。自1998年开始，华章分社就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章分社欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

华章教育

About the Author

Wayne Wolf is Professor, Rhesea "Ray" P. Farmer Distinguished Chair in Embedded Computing, and Georgia Research Alliance Eminent Scholar at the Georgia Institute of Technology. Before joining Georgia Tech, he was with Princeton University and AT&T Bell Laboratories in Murray Hill, New Jersey. He received his B.S., M.S., and Ph.D. in electrical engineering from Stanford University. He is well known for his research in the areas of hardware/software co-design, embedded computing, VLSI CAD, and multimedia computing systems. He is a fellow of the IEEE and ACM. He co-founded several conferences in the area, including CODES, MPSoC, and Embedded Systems Week. He was founding co-editor-in-chief of *Design Automation for Embedded Systems* and founding editor-in-chief of *ACM Transactions on Embedded Computing Systems*. He has received the ASEE Frederick E. Terman Award and the IEEE Circuits and Society Education Award. He is also co-series editor of the Morgan Kaufmann Series in Systems on Silicon.

Foreword to The First Edition

Digital system design has entered a new era. At a time when the design of microprocessors has shifted into a classical optimization exercise, the design of embedded computing systems in which microprocessors are merely components has become a wide-open frontier. Wireless systems, wearable systems, networked systems, smart appliances, industrial process systems, advanced automotive systems, and biologically interfaced systems provide a few examples from across this new frontier.

Driven by advances in sensors, transducers, microelectronics, processor performance, operating systems, communications technology, user interfaces, and packaging technology on the one hand, and by a deeper understanding of human needs and market possibilities on the other, a vast new range of systems and applications is opening up. It is now up to the architects and designers of embedded systems to make these possibilities a reality.

However, embedded system design is practiced as a craft at the present time. Although knowledge about the component hardware and software subsystems is clear, there are no system design methodologies in common use for orchestrating the overall design process, and embedded system design is still run in an *ad hoc* manner in most projects.

Some of the challenges in embedded system design come from changes in underlying technology and the subtleties of how it can all be correctly mingled and integrated. Other challenges come from new and often unfamiliar types of system requirements. Then too, improvements in infrastructure and technology for communication and collaboration have opened up unprecedented possibilities for fast design response to market needs. However, effective design methodologies and associated design tools have not been available for rapid follow-up of these opportunities.

At the beginning of the VLSI era, transistors and wires were the fundamental components, and the rapid design of computers on a chip was the dream. Today the CPU and various specialized processors and subsystems are merely basic components, and the rapid, effective design of very complex embedded systems is the dream. Not only are system specifications now much more complex, but they must also meet real-time deadlines, consume little power, effectively support complex real-time user interfaces, be very cost-competitive, and be designed to be upgradable.

Wayne Wolf has created the first textbook to systematically deal with this array of new system design requirements and challenges. He presents formalisms and a methodology for embedded system design that can be employed by the new type of "tall-thin" system architect who really understands the foundations of system design across a very wide range of its component technologies.

Moving from the basics of each technology dimension, Wolf presents formalisms for specifying and modeling system structures and behaviors and then clarifies these

ideas through a series of design examples. He explores the complexities involved and how to systematically deal with them. You will emerge with a sense of clarity about the nature of the design challenges ahead and with knowledge of key methods and tools for tackling those challenges.

As the first textbook on embedded system design, this book will prove invaluable as a means for acquiring knowledge in this important and newly emerging field. It will also serve as a reference in actual design practice and will be a trusted companion in the design adventures ahead. I recommend it to you highly.

Lynn Conway
Professor Emerita, Electrical Engineering and
Computer Science University of Michigan

Preface to The Second Edition

Embedded computing is more important today than it was in 2000, when the first edition of this book appeared. Embedded processors are in even more products, ranging from toys to airplanes. Systems-on-chips now use up to hundreds of CPUs. The cell phone is on its way to becoming the new standard computing platform. As my column in *IEEE Computer* in September 2006 indicated, there are at least a half-million embedded systems programmers in the world today, probably closer to 800,000.

In this edition I have tried to both update and revamp. One major change is that the book now uses the TI TMS320C55x™ (C55x) DSP. I seriously rewrote the discussion of real-time scheduling. I have tried to expand on performance analysis as a theme at as many levels of abstraction as possible. Given the importance of multiprocessors in even the most mundane embedded systems, this edition also talks more generally about hardware/software co-design and multiprocessors.

One of the changes in the field is that this material is taught at lower and lower levels of the curriculum. What used to be graduate material is now upper-division undergraduate; some of this material will percolate down to the sophomore level in the foreseeable future. I think that you can use subsets of this book to cover both more advanced and more basic courses. Some advanced students may not need the background material of the earlier chapters and you can spend more time on software performance analysis, scheduling, and multiprocessors. When teaching introductory courses, software performance analysis is an alternative path to exploring microprocessor architectures as well as software; such courses can concentrate on the first few chapters.

The new Web site for this book and my other books is http://www.waynewolf.us. On this site, you can find overheads for the material in this book, suggestions for labs, and links to more information on embedded systems.

ACKNOWLEDGMENTS

I would like to thank a number of people who helped me with this second edition. Cathy Wicks and Naser Salameh of Texas Instruments gave me invaluable help in figuring out the C55x. Richard Barry of freeRTOS.org not only graciously allowed me to quote from the source code of his operating system but he also helped clarify the explanation of that code. My editor at Morgan Kaufmann, Chuck Glaser, knew when to be patient, when to be encouraging, and when to be cajoling. (He also has great taste in sushi restaurants.) And of course, Nancy and Alec patiently let me type away. Any problems, small or large, with this book are, of course, solely my responsibility.

Wayne Wolf
Atlanta, GA, USA

Preface to The First Edition

Microprocessors have long been a part of our lives. However, microprocessors have become powerful enough to take on truly sophisticated functions only in the past few years. The result of this explosion in microprocessor power, driven by Moore's Law, is the emergence of embedded computing as a discipline. In the early days of microprocessors, when all the components were relatively small and simple, it was necessary and desirable to concentrate on individual instructions and logic gates. Today, when systems contain tens of millions of transistors and tens of thousands of lines of high-level language code, we must use design techniques that help us deal with complexity.

This book tries to capture some of the basic principles and techniques of this new discipline of embedded computing. Some of the challenges of embedded computing are well known in the desktop computing world. For example, getting the highest performance out of pipelined, cached architectures often requires careful analysis of program traces. Similarly, the techniques developed in software engineering for specifying complex systems have become important with the growing complexity of embedded systems. Another example is the design of systems with multiple processes. The requirements on a desktop general-purpose operating system and a real-time operating system are very different; the real-time techniques developed over the past 30 years for larger real-time systems are now finding common use in microprocessor-based embedded systems.

Other challenges are new to embedded computing. One good example is power consumption. While power consumption has not been a major consideration in traditional computer systems, it is an essential concern for battery-operated embedded computers and is important in many situations in which power supply capacity is limited by weight, cost, or noise. Another challenge is deadline-driven programming. Embedded computers often impose hard deadlines on completion times for programs; this type of constraint is rare in the desktop world. As embedded processors become faster, caches and other CPU elements also make execution times less predictable. However, by careful analysis and clever programming, we can design embedded programs that have predictable execution times even in the face of unpredictable system components such as caches.

Luckily, there are many tools for dealing with the challenges presented by complex embedded systems: high-level languages, program performance analysis tools, processes and real-time operating systems, and more. But understanding how all these tools work together is itself a complex task. This book takes a bottom-up approach to understanding embedded system design techniques. By first understanding the fundamentals of microprocessor hardware and software, we can build powerful abstractions that help us create complex systems.

A NOTE TO EMBEDDED SYSTEM PROFESSIONALS

This book is not a manual for understanding a particular microprocessor. Why should the techniques presented here be of interest to you? There are two reasons. First, techniques such as high-level language programming and real-time operating systems are very important in making large, complex embedded systems that actually work. The industry is littered with failed system designs that didn't work because their designers tried to hack their way out of problems rather than stepping back and taking a wider view of the problem. Second, the components used to build embedded systems are constantly changing, but the principles remain constant. Once you understand the basic principles involved in creating complex embedded systems, you can quickly learn a new microprocessor (or even programming language) and apply the same fundamental principles to your new components.

A NOTE TO TEACHERS

The traditional microprocessor system design class originated in the 1970s when microprocessors were exotic yet relatively limited. That traditional class emphasizes breadboarding hardware and software to build a complete system. As a result, it concentrates on the characteristics of a particular microprocessor, including its instruction set, bus interface, and so on.

This book takes a more abstract approach to embedded systems. While I have taken every opportunity to discuss real components and applications, this book is fundamentally not a microprocessor data book. As a result, its approach may seem initially unfamiliar. Rather than concentrating on particulars, the book tries to study more generic examples to come up with more generally applicable principles. However, I think that this approach is both fundamentally easier to teach and in the long run more useful to students. It is easier because one can rely less on complex lab setups and spend more time on pencil-and-paper exercises, simulations, and programming exercises. It is more useful to the students because their eventual work in this area will almost certainly use different components and facilities than those used at your school. Once students learn fundamentals, it is much easier for them to learn the details of new components.

Hands-on experience is essential in gaining physical intuition about embedded systems. Some hardware building experience is very valuable; I believe that every student should know the smell of burning plastic integrated circuit packages. But I urge you to avoid the tyranny of hardware building. If you spend too much time building a hardware platform, you will not have enough time to write interesting programs for it. And as a practical matter, most classes do not have the time to let students build sophisticated hardware platforms with high-performance I/O devices and possibly multiple processors. A lot can be learned about hardware by measuring and evaluating an existing hardware platform. The experience of programming

complex embedded systems will teach students quite a bit about hardware as well—debugging interrupt-driven code is an experience that few students are likely to forget.

A home page for the book (www.mkp.com/embed) includes overheads, instructor's manual, lab materials, links to related Web sites, and a link to a password-protected ftp site that contains solutions to the exercises.

ACKNOWLEDGMENTS

I owe a word of thanks to many people who helped me in the preparation of this book. Several people gave me advice about various aspects of the book: Steve Johnson (Indiana University) about specification, Louise Trevillyan and Mark Charney (both IBM Research) on program tracing, Margaret Martonosi (Princeton University) on cache miss equations, Randy Harr (Synopsys) on low power, Phil Koopman (Carnegie Mellon University) on distributed systems, Joerg Henkel (NEC C&C Labs) on low-power computing and accelerators, Lui Sha (University of Illinois) on real-time operating systems, John Rayfield (ARM) on the ARM architecture, David Levine (Analog Devices) on compilers and SHARC, and Con Korikis (Analog Devices) on the SHARC. Many people acted as reviewers at various stages: David Harris (Harvey Mudd College); Jan Rabaey (University of California at Berkeley); David Nagle (Carnegie Mellon University); Randy Harr (Synopsys); Rajesh Gupta, Nikil Dutt, Frederic Doucet, and Vivek Sinha (University of California at Irvine); Ronald D. Williams (University of Virginia); Steve Sapiro (SC Associates); Paul Chow (University of Toronto); Bernd G. Wenzel (Eurostep); Steve Johnson (Indiana University); H. Alan Mantooth (University of Arkansas); Margarida Jacome (University of Texas at Austin); John Rayfield (ARM); David Levine (Analog Devices); Ardsher Ahmed (University of Massachusetts/Dartmouth University); and Vijay Madisetti (Georgia Institute of Technology). I also owe a big word of thanks to my editor, Denise Penrose. Denise put in a great deal of effort finding and talking to potential users of this book to help us understand what readers wanted to learn. This book owes a great deal to her insight and persistence. Cheri Palmer and her production team did an excellent job on an impossibly tight schedule. The mistakes and miscues are, of course, all mine.

List of Examples

Application Example 1.1 BMW 850i brake and stability control system 3
Example 1.1 Requirements analysis of a GPS moving map 15
Example 2.1 Status bit computation in the ARM 62
Example 2.2 C assignments in ARM instructions 67
Example 2.3 Implementing an `if` statement in ARM 70
Example 2.4 Implementing the C switch statement in ARM 71
Application Example 2.1 FIR filters 72
Example 2.5 An FIR filter for the ARM 72
Example 2.6 Procedure calls in ARM 75
Application Example 3.1 The 8251 UART 92
Example 3.1 Memory-mapped I/O on ARM 94
Example 3.2 Busy-wait I/O programming 95
Example 3.3 Copying characters from input to output using busy-wait I/O 96
Example 3.4 Copying characters from input to output with basic interrupts 98
Example 3.5 Copying characters from input to output with interrupts and buffers 99
Example 3.6 Debugging interrupt code 103
Example 3.7 I/O with prioritized interrupts 106
Example 3.8 Direct-mapped vs. set-associative caches 117
Example 3.9 Execution time of a for loop on the ARM 127
Application Example 3.2 Energy efficiency features in the PowerPC 603 130
Application Example 3.3 Power-saving modes of the StrongARM SA-1100 132
Application Example 3.4 Huffman coding for text compression 134
Example 4.1 A glue logic interface 176
Application Example 4.1 System organization of the Intel StrongARM SA-1100 and SA-1111 182
Programming Example 4.1 Breakpoints 185
Example 4.2 A timing error in real-time code 187
Example 4.3 Performance bottlenecks in a bus-based system 193
Programming Example 5.1 A software state machine 210
Programming Example 5.2 A circular buffer implementation of an FIR filter 213
Programming Example 5.3 A buffer-based queue 214
Example 5.1 Generating a symbol table 223
Example 5.2 Compiling an arithmetic expression 229
Example 5.3 Generating code for a conditional 231
Example 5.4 Loop unrolling 238
Example 5.5 Register allocation 240

Example 5.6 Operator scheduling for register allocation 243
Example 5.7 Data-dependent paths in `if` statements 251
Example 5.8 Paths in a loop 252
Example 5.9 Cycle-accurate simulation 256
Example 5.10 Data realignment and array padding 260
Example 5.11 Controlling and observing programs 268
Example 5.12 Choosing the paths to test 270
Example 5.13 Condition testing with the branch testing strategy 273
Application Example 6.1 Automotive engine control 296
Application Example 6.2 A space shuttle software error 300
Example 6.1 Utilization of a set of processes 304
Example 6.2 Priority-driven scheduling 309
Example 6.3 Rate-monotonic scheduling 317
Example 6.4 Earliest-deadline-first scheduling 320
Example 6.5 Priority inversion 324
Example 6.6 Data dependencies and scheduling 325
Example 6.7 Elastic buffers as shared memory 326
Programming Example 6.1 Test-and-set operations 328
Example 6.8 Scheduling and context switching overhead 330
Example 6.9 Effects of scheduling on the cache 332
Example 7.1 Performance effects of scheduling and allocation 365
Example 7.2 Overlapping computation and communication 366
Example 7.3 Buffers and latency 368
Example 8.1 Data-push network architectures 405
Example 8.2 Simple message delay for an I^2C message 414
Application Example 8.1 An Internet video camera 420
Application Example 9.1 Loss of the Mars Climate Observer 439
Example 9.1 Concurrent engineering applied to telephone systems 444
Application Example 9.2 The TCAS II specification 451
Example 9.2 CRC card analysis 456
Application Example 9.3 The Therac-25 medical imaging system 458

Contents

About the Author iv
Foreword to The First Edition v
Preface to The Second Edition vii
Preface to The First Edition viii
List of Examples xi

CHAPTER 1 Embedded Computing 1
Introduction 1
1.1 Complex Systems and Microprocessors 1
1.1.1 Embedding Computers 2
1.1.2 Characteristics of Embedded Computing Applications 4
1.1.3 Why Use Microprocessors? 6
1.1.4 The Physics of Software 8
1.1.5 Challenges in Embedded Computing System Design 8
1.1.6 Performance in Embedded Computing 10
1.2 The Embedded System Design Process 11
1.2.1 Requirements 12
1.2.2 Specification 17
1.2.3 Architecture Design 18
1.2.4 Designing Hardware and Software Components 20
1.2.5 System Integration 20
1.3 Formalisms for System Design 21
1.3.1 Structural Description 22
1.3.2 Behavioral Description 27
1.4 Model Train Controller 30
1.4.1 Requirements 31
1.4.2 DCC 32
1.4.3 Conceptual Specification 34
1.4.4 Detailed Specification 37
1.4.5 Lessons Learned 44
1.5 A Guided Tour of This Book 45
1.5.1 Chapter 2: Instruction Sets 46
1.5.2 Chapter 3: CPUs 46
1.5.3 Chapter 4: Bus-Based Computer Systems 46

1.5.4 Chapter 5: Program Design and Analysis 47
1.5.5 Chapter 6: Processes and Operating Systems 48
1.5.6 Chapter 7: Multiprocessors 49
1.5.7 Chapter 8: Networks 50
1.5.8 Chapter 9: System Design Techniques 50
Summary 51
Further Reading 51
Questions 52
Lab Exercises 53

CHAPTER 2 Instruction Sets 55

Introducton 55
2.1 Preliminaries 55
2.1.1 Computer Architecture Taxonomy 55
2.1.2 Assembly Language 58
2.2 ARM Processor 59
2.2.1 Processor and Memory Organization 60
2.2.2 Data Operations 61
2.2.3 Flow of Control 69
2.3 TI C55x DSP 76
2.3.1 Processor and Memory Organization 76
2.3.2 Addressing Modes 78
2.3.3 Data Operations 82
2.3.4 Flow of Control 83
2.3.5 C Coding Guidelines 85
Summary 86
Further Reading 86
Questions 86
Lab Exercises 89

CHAPTER 3 CPUs 91

Introduction 91
3.1 Programming Input and Output 91
3.1.1 Input and Output Devices 92
3.1.2 Input and Output Primitives 93
3.1.3 Busy-Wait I/O 95
3.1.4 Interrupts 96
3.2 Supervisor Mode, Exceptions, and Traps 110
3.2.1 Supervisor Mode 111
3.2.2 Exceptions 111
3.2.3 Traps 112
3.3 Co-Processors 112

3.4 Memory System Mechanisms 113
3.4.1 Caches 113
3.4.2 Memory Management Units and Address Translation 119
3.5 CPU Performance 124
3.5.1 Pipelining 124
3.5.2 Caching 128
3.6 CPU Power Consumption 129
3.7 Design Example: Data Compressor 134
3.7.1 Requirements and Algorithm 134
3.7.2 Specification 136
3.7.3 Program Design 139
3.7.4 Testing 145
Summary 147
Further Reading 147
Questions 148
Lab Exercises 151

CHAPTER 4 Bus-Based Computer Systems **153**
Introduction 153
4.1 The CPU Bus 153
4.1.1 Bus Protocols 154
4.1.2 DMA 160
4.1.3 System Bus Configurations 162
4.1.4 AMBA Bus 165
4.2 Memory Devices 166
4.2.1 Memory Device Organization 166
4.2.2 Random-Access Memories 167
4.2.3 Read-Only Memories 169
4.3 I/O devices 169
4.3.1 Timers and Counters 169
4.3.2 A/D and D/A Converters 171
4.3.3 Keyboards 171
4.3.4 LEDs 173
4.3.5 Displays 173
4.3.6 Touchscreens 175
4.4 Component Interfacing 175
4.4.1 Memory Interfacing 176
4.4.2 Device Interfacing 176
4.5 Designing with Microprocessors 177
4.5.1 System Architecture 177
4.5.2 Hardware Design 179
4.5.3 The PC as a Platform 180

4.6 Development and Debugging 183
4.6.1 Development Environments 183
4.6.2 Debugging Techniques 184
4.6.3 Debugging Challenges 187
4.7 System-Level Performance Analysis 189
4.7.1 System-Level Performance Analysis 189
4.7.2 Parallelism 194
4.8 Design Example: Alarm Clock 196
4.8.1 Requirements 196
4.8.2 Specification 198
4.8.3 System Architecture 200
4.8.4 Component Design and Testing 203
4.8.5 System Integration and Testing 204
Summary 204
Further Reading 205
Questions 205
Lab Exercises 207

CHAPTER 5 Program Design and Analysis 209
Introduction 209
5.1 Components for Embedded Programs 210
5.1.1 State Machines 210
5.1.2 Stream-Oriented Programming and Circular Buffers 212
5.1.3 Queues 213
5.2 Models of Programs 215
5.2.1 Data Flow Graphs 215
5.2.2 Control/Data Flow Graphs 217
5.3 Assembly, Linking, and Loading 220
5.3.1 Assemblers 222
5.3.2 Linking 225
5.4 Basic Compilation Techniques 227
5.4.1 Statement Translation 229
5.4.2 Procedures 233
5.4.3 Data Structures 234
5.5 Program Optimization 236
5.5.1 Expression Simplification 236
5.5.2 Dead Code Elimination 237
5.5.3 Procedure Inlining 237
5.5.4 Loop Transformations 238
5.5.5 Register Allocation 239
5.5.6 Scheduling 244
5.5.7 Instruction Selection 246