ELSEVIER

Joseph A. Fisher • Paolo Faraboschi • Cliff Young

Embedded Computing

A VLIW Approach to Architecture, Compilers, and Tools

MK MORGAN KAUFMANN

Publisher	Denise E. M. Penrose
Publishing Services Manager	Simon Crump
Senior Production Editor	Angela Dooley
Editorial Assistant	Valerie Witte
Cover Design	Hannus Design
Cover Image	Santiago Calatrava's Alamillo Bridge
Text Design	Frances Baca Design
Composition	CEPHA
Technical Illustration	Dartmouth Publishing
Copyeditor	Daril Bentley
Proofreader	Phyllis Coyne & Associates
Indexer	Northwind Editorial
Interior printer	The Maple-Vail Manufacturing Group
Cover printer	Phoenix Color, Inc.

Morgan Kaufmann Publishers is an imprint of Elsevier. 500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

Cover image: Santiago Calatrava's Alamillo Bridge blends art and engineering to make architecture. While his design remains a modern, cable-stayed bridge, it simultaneously reinvents the category, breaking traditional assumptions and rearranging structural elements into a new form that is efficient, powerful, and beautiful. The authors chose this cover image for a number of reasons. Compiler engineering, which is at the heart of modern VLIW design, is similar to bridge engineering: both must be built to last for decades, to withstand changes in usage and replacement of components, and to weather much abuse. The VLIW design philosophy was one of the first computer architectural styles to bridge the software and hardware communities, treating them as equals and partners. And this book is meant as a bridge between the VLIW and embedded communities, which had historically been separate, but which today have complementary strengths and requirements.

ADVICE, PRAISE, AND ERRORS: Any correspondence related to this publication or intended for the authors should be addressed to FFY@VLIW.org. Information regarding error sightings is also encouraged and can be sent to mkp@mkp.com.

Library of Congress Cataloging-in-Publication Data
ISBN: 1-55860-766-8

For information on all Morgan Kaufmann publications,
visit our Web site at www.mkp.com or www.books.elsevier.com.

Printed in the United States of America
04 05 06 07 08 5 4 3 2 1

To my wife Elizabeth, our children David and Dora,
and my parents, Harry and the late Susan Fisher.
And to my friend and mentor, Martin Davis.
Josh Fisher

To the memory of my late parents Silvio and Gina,
to my wife Tatiana and our daughter Silvia.
Paolo Faraboschi

To the women of my family:
Yueh-Jing, Dorothy, Matilda, Joyce, and Celeste.
Cliff Young

To Bob Rau, a VLIW pioneer and true visionary,
and a wonderful human being.
We were privileged to know and work with him.
The Authors

About the Authors

JOSEPH A. FISHER is a Hewlett-Packard Senior Fellow at HP Labs, where he has worked since 1990 in instruction-level parallelism and in custom embedded VLIW processors and their compilers. Josh studied at the Courant Institute of NYU (B.A., M.A., and then Ph.D. in 1979), where he devised the trace scheduling compiler algorithm and coined the term *instruction-level parallelism*. As a professor at Yale University, he created and named VLIW architectures and invented many of the fundamental technologies of ILP. In 1984, he started Multiflow Computer with two members of his Yale team. Josh won an NSF Presidential Young Investigator Award in 1984, was the 1987 Connecticut Eli Whitney Entrepreneur of the Year, and in 2003 received the ACM/IEEE Eckert-Mauchly Award.

PAOLO FARABOSCHI is a Principal Research Scientist at HP Labs. Before joining Hewlett-Packard in 1994, Paolo received an M.S. (Laurea) and Ph.D. (Dottorato di Ricerca) in electrical engineering and computer science from the University of Genoa (Italy) in 1989 and 1993, respectively. His research interests skirt the boundary of hardware and software, including VLIW architectures, compilers, and embedded systems. More recently, he has been looking at the computing aspects of demanding content-processing applications. Paolo is an active member of the computer architecture community, has served in many program committees, and was Program Co-chair for MICRO (2001) and CASES (2003).

CLIFF YOUNG works for D. E. Shaw Research and Development, LLC, a member of the D. E. Shaw group of companies, on projects involving special-purpose, high-performance computers for computational biochemistry. Before his current position, he was a Member of Technical Staff at Bell Laboratories in Murray Hill, New Jersey. He received A.B., S.M., and Ph.D. degrees in computer science from Harvard University in 1989, 1995, and 1998, respectively.

Foreword

Bob Colwell, R & E Colwell & Assoc. Inc.

There are two ways to learn more about your country: you can study it directly by traveling around in it or you can study it indirectly by leaving it. The first method yields facts and insights directly in context, and the second by contrast.

Our tradition in computer engineering has been to seldom leave our neighborhood. If you want to learn about operating systems, you read an OS book. For multiprocessor systems, you get a book that maps out the MP space.

The book you are holding in your hands can serve admirably in that direct sense. If the technology you are working on is associated with VLIWs or "embedded computing," clearly it is imperative that you read this book.

But what pleasantly surprised me was how useful this book is, even if one's work is not VLIW-related or has no obvious relationship to embedded computing. I had long felt it was time for Josh Fisher to write his magnum opus on VLIWs, so when I first heard that he and his coauthors were working on a book with VLIW in the title I naturally and enthusiastically assumed this was it. Then I heard the words "embedded computing" were also in the title and felt considerable uncertainty, having spent most of my professional career in the general-purpose computing arena. I thought embedded computing was interesting, but mostly in the same sense that studying cosmology was interesting: intellectually challenging, but what does it have to do with me?

I should have known better. I don't think Josh Fisher can write boring text. He doesn't know how. (I still consider his "Very Long Instruction Word Architectures and the ELI-512" paper from ISCA-10 to be the finest conference publication I have ever read.) And he seems to have either found like-minded coauthors in Faraboschi and Young or has taught them well, because *Embedded Computing: A VLIW Approach to Architecture, Tools and Compilers* is enthralling in its clarity and exhilarating in its scope. If you are involved in computer system design or programming, you must still read this book, because it will take you to places where the views are spectacular, including those looking over to where you usually live. You don't necessarily have to agree with every point the authors make, but you *will* understand what they are trying to say, and they *will* make you think.

One of the best legacies of the classic Hennessy and Patterson computer architecture textbooks is that the success of their format and style has encouraged more books like theirs. In *Embedded Computing: A VLIW Approach to Architecture, Tools and Compilers*, you will find the pitfalls, controversies, and occasional opinion sidebars that made

H&P such a joy to read. This kind of technical exposition is like vulcanology done while standing on an active volcano. Look over there, and see molten lava running under a new fissure in the rocks. Feel the heat; it commands your full attention. It's immersive, it's interesting, and it's immediate. If your Vibram soles start melting, it's still worth it. You probably needed new shoes anyway.

I first met Josh when I was a grad student at Carnegie-Mellon in 1982. He spent an hour earnestly describing to me how a sufficiently talented compiler could in principle find enough parallelism, via a technique he called trace scheduling, to keep a really wild-looking hardware engine busy. The compiler would speculatively move code all over the place, and then invent more code to fix up what it got wrong. I thought to myself "So *this* is what a lunatic looks like up close. I hope he's not dangerous." Two years later I joined him at Multiflow and learned more in the next five years than I ever have, before or since.

It was an honor to review an early draft of this book, and I was thrilled to be asked to contribute this foreword. As the book makes clear, general-purpose computing has traditionally gotten the glory, while embedded computing quietly keeps our infrastructure running. This is probably just a sign of the immaturity of the general-purpose computing environment (even though we "nonembedded" types don't like to admit that). With general-purpose computers, people "use the computer" to do something. But with embedded computers, people accomplish some task, blithely and happily unaware that there's a computer involved. Indeed, if they had to be conscious of the computer, their embedded computers would have already failed: antilock brakes and engine controllers, for instance. General-purpose CPUs have a few microarchitecture performance tricks to show their embedded brethren, but the embedded space has much more to teach the general computing folks about the bigger picture: total cost of ownership, who lives in the adjacent neighborhoods, and what they need for all to live harmoniously. This book is a wonderful contribution toward that evolution.

Bob Colwell
June 17, 2004

Preface

Welcome to our book. We hope you enjoy reading it as much as we have enjoyed writing it. The title of this book contains two major keywords: *embedded* and *VLIW* (very long instruction word). Historically, the embedded computing community has rarely been related to the VLIW community. Technology is removing this separation, however. High-performance techniques such as VLIW that seemed too expensive for embedded designs have recently become both feasible and popular. This change is bringing in a new age of embedded computing design, in which a high-performance processor is central. More and more, the traditional elements of nonprogrammable components, peripherals, interconnects, and buses must be seen in a computing-centric light. Embedded computing designers must design systems that unify these elements with high-performance processor architectures, microarchitectures, and compilers, as well as with the compilation tools, debuggers, and simulators needed for application development.

Since this is a book about embedded computing, we define and explore that world in general, but with the strongest emphasis on the processing aspects. Then, within this new world of embedded, we show how the VLIW design philosophy matches the goals and constraints well. We hope we have done this in a way that clearly and systematically explains the unique problems in the embedded domain, while remaining approachable to those with a general background in architecture and compilation. Conversely, we also need to explain the VLIW approach and its implications and to point out the ways in which VLIW, as contrasted with other high-performance architectural techniques, is uniquely suited to the embedded world.

We think this book fills a hole in the current literature. A number of current and upcoming books cover embedded computing, but few of them take the combined hardware–software systems approach we do. While the embedded computing and digital signal processing (DSP) worlds seem exotic to those with general-purpose backgrounds, they remain *computing*. Much is common between general-purpose and embedded techniques, and after showing what is common between them, we can focus on the differences. In addition, there is no standard reference on the VLIW approach. Such a book has been needed for at least a decade, and we believe that a book explaining the VLIW design philosophy has value today. This book should be useful to engineers and designers in industry, as well as suitable as a textbook for courses that aim at seniors or first-year graduate students.

While considering the mission of our book, we came up with three different possible books on the spectrum from VLIW to embedded. The first is the previously mentioned book, purely about VLIW. The second is a book about high-performance approaches

to the embedded domain, with equal emphasis on VLIW, Superscalar, digital signal processor (DSP), micro-SIMD (Single Instruction Multiple Data), and vector techniques. Our book (the third option) strikes a balance: it focuses on the VLIW approach to the embedded domain. This means we give lighter treatment to the alternative approaches but spend additional effort on drawing the connections between VLIW and embedded. However, large parts of the information in our book overlap material that would go into the other two, and we think of this book as valuable for those with a strong interest in embedded computing but only a little interest in VLIW, and vice versa.

Along the way, we have tried to present our particularly idiosyncratic views of embedded, VLIW, and other high-performance architectural techniques. Most of the time, we hope we have impartially presented facts. However, these topics would be terribly dry and boring if we removed all controversy. VLIW has become a significant force in embedded processing and, as we make clear, there are technical and marketing reasons for this trend to continue. We will wear our biases on our sleeves (if you can't tell from the title, we think VLIW is the correct hammer for the embedded nail), but we hope to be honest about these biases in areas that remain unresolved.

Content and Structure

When we first wrote the outline for this book, the chapters fell into three major categories: *hardware*, *software*, and *applications*. Thus, the outline of the book correspondingly had three major parts. As we have written and rewritten, the organization has changed, pieces have migrated from one chapter to another, and the clean three-part organization has broken down into a set of chapters that only roughly matches the original tripartite structure. The unfortunate truth of modern computer architecture is that one cannot consider any of hardware, software, or applications by themselves.

This book really has two introductory chapters. Chapter 1 describes the world of embedded processing. It defines embedded processing, provides examples of the various types of embedded processors, describes application domains in which embedded cores are deployed, draws distinctions between the embedded and general-purpose domains, and talks about the marketplace for embedded devices. The second introductory chapter, Chapter 2, defines *instruction-level parallelism* (ILP), the primary technique for extracting performance in many modern architectural styles, and describes how compilation is crucial to any ILP-oriented processor design. Chapter 2 also describes the notion of an architectural style or design philosophy, of which VLIW is one example. Last, Chapter 2 describes how technology has evolved so that VLIW and embedded, once vastly separate domains, are now quite suited to each other.

Chapters 3 through 5 constitute the purely "hardware"-related part of the book. Chapter 3 describes what we mean when we say architecture or instruction-set architecture (ISA), defines what a VLIW ISA looks like, and describes in particular how VLIW architectures have been built for embedded applications. Chapter 3 also describes instruction set *encoding* at two levels. From a high-level perspective, Chapter 3 revisits the notion of design philosophy and architectural style with respect to how that style affects the way operations and instructions are encoded under each design philosophy.

At a detailed level, Chapter 3 describes the particular issues associated with VLIW operation and instruction encoding.

Chapter 4 might be seen as a continuation of the previous chapter, but instead of describing ISA design as a whole (with a view across various ISA styles), Chapter 4 examines the hardware structures (such as the datapath, memory, register files, and control units) necessary to all modern processors. Chapter 4 pays particular attention to how these structures differ in the embedded domain from their general-purpose counterparts.

The next chapter explores microarchitecture, the implementation of techniques within a given ISA. Chapter 5 can be seen as largely paralleling Chapter 4 in subject matter, but it considers how to *implement* each piece of functionality rather than how to *specify* that work be done within an ISA. Chapter 5 is informed by the technological constraints of modern design; that is, wires are expensive, whereas transitors are cheap. The chapter also (very briefly) considers power-related technological concerns.

Chapter 6 fits poorly into either the hardware and software categories, as both topics occur in each of its sections. Chapter 6 begins with a description of how a system-on-a-chip (SoC) is designed. Most modern embedded systems today are designed using the SoC methodology. Chapter 6 continues with how processor cores integrate with SoCs. Then it describes simulation methodologies for processor cores, followed by simulation techniques for entire systems. Last, Chapter 6 describes validation and verification of simulators and their systems. It might be best to view Chapter 6 as a bridge between the hardware and software areas, or perhaps its integration of the two serves as a good illustration of the complexities involved in building hardware/software *systems*.

The next three chapters emphasize the software area, although reading them will make it clear that they are infused with hardware-related topics in a number of ways. Chapter 7 describes the entire *toolchain*: the suite of software programs used to analyze, design, and build the software of an embedded system. Chapter 7 also describes a number of embedded- and DSP-specific code transformations.

Chapter 8 describes a subset of the compiler optimizations and transformations in an industrial-strength ILP-oriented compiler. This book is not a compiler textbook. Our goal in this chapter is to paint a balanced picture of the suite of optimizations — including their uses, complexities, and interactions — so that system designers will understand the nature of compilation-related issues, and so that compiler designers will know where else to look.

Chapter 9 covers a broad range of topics that often fall between the cracks of traditional topics, but are nonetheless important to building a working system. Chapter 9 details issues about exceptions, application binary interfaces (ABIs), code compression, operating systems (including embedded and real-time variants), and multiprocessing. Many of these topics have a strong software component to them, but each also interacts strongly with hardware structures that support the software functionality.

The last two chapters focus on applications. Chapter 10 begins by discussing programming languages for embedded applications, and then moves on to performance, benchmarks, and tuning. Then it continues to scalability and customizability in embedded architectures, and finishes with detail about customizable processors.

Chapter 11 visits a number of embedded applications at a variety of levels of detail. We spend the most time on digital printing and imaging, and telecommunications, and less time on other areas, such as automotive, network processing, and disk drives.

While writing this book, it became clear that there are a large number of terms with overlapping and conflicting meanings in this field. For example, *instruction* can mean operation, bundle, parallel issue group, or parallel execution group to different subcommunities. Wherever possible, we use the terms as they are used in the architecture field's dominant textbook, John Hennessy and Dave Patterson's *Computer Architecture: A Quantitative Approach*. The Glossary lists alternate definitions and synonyms, and indicates which terms we intend to use consistently.

The VEX (VLIW Example) Computing System

Lest we be accused of writing an armchair textbook (like those scientists of the nineteenth century who deduced everything from first principles), our book ships with an embedded-oriented VLIW development system. We call this system VEX, for "VLIW Example." We hope it is even more useful to our readers than its textbook ancestors, MIX and DLX, were for their readers. VEX is based on production tools used at HP Labs and other laboratories. It is a piece of real-world VLIW processor technology, albeit simplified for instructional use.

VEX is intended for experimental use. It includes a number of simulators, and its tools allow hardware reconfiguration and both manual and automated design-space exploration. Code, documentation, and samples can be downloaded from the book's Web site at *http://www.vliw.org/book*. VEX examples and exercises occur throughout the book. The Appendix describes the VEX instruction set architecture and tool chain.

Audience

We assume a basic knowledge of computer architecture concepts, as might be given by some industrial experience or a first undergraduate course in architecture. This implies that you know the basic techniques of pipelining and caching, and that the idea of an instruction set is familiar. It helps but is not a requirement that you have some background in compilation, or at least that you believe an optimizing compiler might be useful in producing fast code for modern machines. For reasons of space, we touch on those fundamentals related to this text and for more basic information refer you to more basic architecture and compilation textbooks. Patterson and Hennessy's undergraduate architecture textbook, *Computer Organization and Design*, and Appel's polymorphic set of undergraduate compiler books, *Modern Compiler Implementation in C, Java*, and *ML* are fine places to start.

There are four likely types of readers of our book. For those trying to bridge the embedded and high-performance communities, we believe this book will help. Designers of general-purpose systems interested in embedded issues should find this book a useful introduction to a new area. Conversely, those who work with existing embedded and/or DSP designs but would like to understand more about high-performance computing in

general, and VLIW in particular (as these technologies become a central force in the embedded domain) are also part of our audience. Third, the book should serve well as a general reference on all aspects of the VLIW design style, embedded or general-purpose. Last, this book should be usable in a senior undergraduate or graduate-level computer architecture course. It could be the main textbook in an embedded-specific course, and it could be used to supplement a mainstream computer architecture text.

Cross-cutting Topics

From the chapter organization of our book, you can see that we have organized it horizontally, in effect by different traditional layers between fields: hardware versus software, with various chapters dealing with issues within the hardware area (such as ISA, microarchitecture, and SoC). However, some ("vertical") topics cut across multiple layers, making them difficult to explain in a single place, and unfortunately necessitating forward references. These topics include clustering, encoding and fetching, memory access, branch architecture, predication, and multiprocessing and multithreading. This section points out where the cross-cutting topic threads can be found, so that readers can follow a single thread through multiple layers.

Clusters, or groupings of register files and functional units with complete connectivity and bypassing, are described from an instruction-set encoding perspective in Section 3.5, "VLIW Encoding," as a structure in hardware design in Section 4.2, "Registers and Clusters," with respect to branches in Section 4.4, "Branch Architecture," from an implementation perspective in Section 5.1, "Register File Design," as a compiler target in Section 8.2, "Scheduling," and with respect to scalability in Section 10.3, "Scalability and Customizability."

Encoding and its dual problem of decoding occur as general topics in Chapters 3 and 5. However, the specific physical issue of dispatching operations to clusters and functional units is treated more specifically in Sections 3.5, "VLIW Encoding" and Section 5.3 "VLIW Fetch, Sequencing and Decoding." There are also correspondingly detailed discussions of encoding and ISA extensions in Sections 3.6, "Encoding and Instruction Set Extensions" and Section 5.3 "VLIW Fetch, Sequencing and Decoding."

The architectural view of predication is introduced in Section 4.5.2, "Predication." Microarchitectural support for predication, and in particular its effect on the bypass network, is described in Section 5.4.4, "Predication and Selects." Compiler support for predication is discussed throughout Chapter 8, and in particular appears in Section 8.2.1, "Acyclic Region Types and Shapes," in Section 8.2.5, "Loop Scheduling," and in Section 8.4.2, "Predicated Execution."

Multiprocessing, or using multiple processor cores (either physical or virtual) in a single system, is discussed as a pure memory-wiring problem in Section 5.5.4, "Memories in Mutliprocessor Systems," with respect to SoC design in Section 6.2.2, "Multiprocessing on a chip," and with respect to the run-time system in Sections 9.4.3, "Multiple Flows of Control" and Section 9.5, "Multiprocessing and Multithreading."

How to Read This Book

The most obvious reading advice is to read the book from cover to cover, and then read it again. This is not particularly useful advice, and thus the following outlines how we think various types of readers might approach the book.

To use this book as the main text in a senior or graduate course, we recommend using each chapter in order. The other possibility would be to jump immediately to the software section after a prerequisite in hardware from another course. If the book is supplementary material to another architecture or compilation textbook, various chapters (e.g., those on microarchitecture, simulation, and application analysis) will be especially appropriate as selective reading.

If you already know a lot about VLIWs, much of the introductory chapter on VLIWs (Chapter 2) and most of the compiler details in Chapters 7 and 8 will be familiar. We recommend focusing on Chapters 3 through 5 (on ISA, structure, and microarchitecture, respectively), and also scanning for topics that are unique to embedded systems. The information about other parts of the development toolchain in Chapter 7 will still be relevant, and the application-related chapters (10 and 11) will be relevant in any case.

If you already work in an embedded or DSP-related field, the embedded-specific parts of the hardware-oriented chapters (3 through 5) will be familiar to you, and some or all of the application examples in Chapter 11 will be familiar. Depending on your specialization, the SoC part of Chapter 6 may be familiar, but the simulation and verification parts of that chapter will be especially valuable. Pay close attention to the importance of ILP compilation and the pitfalls associated with compilers, covered in Chapters 7 and 8.

If you have a general-purpose architecture background, many parts of Chapters 3 through 5 will be familiar, as will the sections on the software development toolchain in Chapter 7. Try reading them, and skim where it seems appropriate. Parts of Chapter 8 (on compilation) may be skimmed, depending on your particular expertise. The final chapter, dealing with application examples, pulls together many of the principles of the book, so they're worth spending the time to read.

We greatly admire the textbooks of Dave Patterson and John Hennessey, and we adopted some of their organizational ideas. Like them, we include sidebars on "fallacies" and "pitfalls." We also added sidebars we call "controversies." These comment on issues too unsettled to fall into one of the former categories. Our equivalents of their "Putting It All Together" sections have been grouped in Chapter 11. These application examples play the same role in our book that example instruction set architectures such as MIPS, the Intel x86, the DEC VAX, and the IBM 360/370 play in Hennessy and Patterson [2004].

Because our book emphasizes embedded processing, there are sections and sidebars that focus on "embedded-specific topics." As in general-purpose work, performance remains a central theme, but the embedded world adds additional optimization goals for power/heat, space/size, and cost. Each of these topics receives special emphasis in dedicated sections.

The book does not cover the entire space of embedded systems and tries to remain within a rather fuzzy set of boundaries. On the hardware and modeling side, we never

descend below the architecture and microarchitecture level, and only provide pointers to relevant literature on ASIC design, CAD tools, logic design techniques, synthesis, verification, and modeling languages. Although reconfigurable computing is of increasing importance in the embedded domain, of necessity we give it less time than it deserves. In the chapters dedicated to compiler technology, we focus largely on VLIW-specific and embedded-specific techniques for regular architectures. For example, we do not cover front-end-related issues (lexical analysis, parsing, and languages) nor "traditional" scalar optimizations, both of which can be found in the abundant compiler literature. When talking about system software and simulation, our boundary is the operating system, whose role we discuss but whose technology we only skim (this also applies to programming languages). We spend very little of the book discussing real time. Finally, in the application sections, we cover only the most relevant aspects of some of the underlying algorithms, but always with an eye to their computing requirements and the interaction with the rest of the system.

Each chapter is accompanied by a set of exercises. Following widespread practice, especially difficult exercises are marked with chili pepper symbols. A single 🌶 means that an exercise requires some materials not included in this book. Two 🌶🌶 indicate that the exercise is something of a project in scope. Three 🌶🌶🌶 mark those places where we weaseled out of writing the section ourselves, and left the actual work to the reader.[1] Throughout the book we use several well-known acronyms, whose definitions and explanations we collect in the glossary.

1. If you do a good job, please send us your text for our next edition.

Figure Acknowledgments

Figures 1.3, 6.9 adapted from Texas Instruments Incorporated.

Figures 2.4, 11.1 courtesy, Hewlett-Packard Company.

Figure 5.11 adapted from Montanaro et al. *IEEE Journal of Solid-State Circuits*, volume 31, number 11, November 1996, pages 1703–1711.

Figure 5.12 adapted from Sukjae Cho, University of Southern California, Information Sciences Institute, *http://pads.east.isi.edu/presentations/misc/sjcho-pm-report.pdf.*

Figure 6.2 adapted from Cirrus Logic, Inc.

Figure 6.4 adapted from Cadence Design Systems, Inc.

Figure 6.5 adapted from IBM Corporation and ARM Ltd.

Figure 6.8 courtesy of Altera Corporation. Altera is a trademark and service mark of Altera Corporation in the United States and other countries. Altera products are the intellectual property of Altera Corporation and are protected by copyright laws and one or more U.S. and foreign patents and patent applications.

Figures 11.3, 11.4 adapted from Kipphan, Helmut. *Handbook of Print Media: Technologies and Manufacturing Processes.* Springer-Verlag, 2001.

Figure 11.18, courtesy of Siemens VDO Automotive.

Figure 11.19 adapted from Balluchi, Andrea; Luca Benvenuti, Di Benedetto, Maria Domenica; Pinello, Claudio; Sangiovanni-Vincentelli, Alberto Luigi. *Automotive Engine Control and Hybrid Systems: Challenges and Opportunities. Proceedings of the IEEE*, 88(7):888–912, July 2000.

Figures 11.21, 11.22 adapted from Intel Corporation.

Acknowledgments

Our first praise and thanks must go to our editor, Denise Penrose, of Morgan Kaufmann Publishers (a division of Elsevier). She has happily, patiently, and diligently assisted us through the various stages of writing, and she has tolerated missed deadlines, slipped schedules, and annoyingly inconvenient phone calls beyond all reason. We also thank the rest of the enormously talented team at Morgan Kaufmann (Elsevier) — both the folks behind the scenes, and Angela Dooley, Emilia Thiuri and Valerie Witte, who we had the pleasure of dealing with directly.

Next, we thank our reviewers: Erik Altman, IBM; Eduard Ayguade, Universitat Politècnica de Catalunya; Alan Berenbaum, Agere Systems; Peter Bosch, Bell Labs; Dan Connors, University of Colorado; Bob Colwell, R&E Colwell & Assoc. Inc.; Gene Frantz, Texas Instruments; Rajiv Gupta, University of Arizona; John Hennessy, Stanford University; Mark Hill, University of Wisconsin-Madison; Tor Jeremiassen, Texas Instruments; Norm Jouppi, HP Labs; Brian Kernighan, Princeton University; Jack Kouloheris, IBM Research; Richard Lethin, Reservoir Labs, Inc.; Walid Najjar, University of California, Riverside; Tarun Nakra, IBM; Michael D. Smith, Harvard University; Mateo Valero, Universitat Politècnica de Catalunya.

They constructively criticized our work, sometimes contributing technical material themselves, and they vastly improved both the details and the overall shape of this book. The turning point in our work came when we first saw the reviews and realized that despite assigning us much more work to do, our reviewers believed we were building something good. We thank them also for their patience with the early and incomplete versions we shipped them. Bob Colwell deserves special mention. His combination of precision, technical mastery, and willingness to flame made his reviews both a delight to read and a major source of improvements to our book.

Two other people helped to better tie our book together. Kim Hazelwood performed redundancy elimination and cross-linking on our text. Mark Toburen compiled, sorted, and double checked our bibliography and bibliographic references.

Many other individuals helped with specific technical points. Peter Bosch and Sape Mullender helped with the history of real-time schedulers. Andrea Cuomo and Bob Krysiak educated us about the embedded marketplace. Giuseppe Desoli's work at HP Labs inspired us in many ways when discussing applications analysis, optimization, and fine-tuning techniques. Gene Frantz helped us with the standards battles in DSPs using saturating arithmetic. Stefan Freudenberger designed and wrote the run-time architecture of the Lx/ST200, the starting point for the VEX run-time architecture.

Fred (Mark Owen) Homewood helped us with his insightful views on microarchitecture and VLSI design. Dong Lin checked our work describing networks and network processors. Josep Llosa gave instrumental advice about the discussion of modulo scheduling. C. K. Luk gave us advice on the state of the art in compiler-directed prefetching. Scott Peterson was an invaluable source of information for anything related to the legal aspects of intellectual property and the complex legal ramifications of open-source licenses. Dennis Ritchie improved our discussion of C99. Miami Beach architect Randall Robinson helped us with the role of design philosophies as seen by true architects (i.e., those who build buildings and landscapes, not those who build chips). Chris Tucci informed us about the economics of monopolies, and their effect on innovation. Bob Ulichney reviewed our various descriptions of image-processing pipelines. Gary Vondran and Emre Ozer helped us with their evaluations of code compression and code layout techniques.

A special mention goes to Geoffrey Brown, who was one of the originators of the idea of this book, and who made significant contributions to the initial definition of the book's topics. As we were starting to write the book, Geoff decided to follow other career paths, but many of his initial ideas are still reflected in the book's organization and content.

At the beginning of this project, a number of computer science authors gave us extremely useful advice about how to write a book. Al Aho, Jon Bentley, Brian Kernighan, Rob Pike, and Dennis Ritchie gave freely of their time and wisdom. Their suggestions (especially about carefully choosing coauthors) were invaluable to us. Arthur Russell and his associates reviewed our contract, and Martin Davis helped us with his experiences with publishers.

A number of individuals and organizations loaned us infrastructure while we were writing this book. Glenn Holloway, Chris Kells, John Osborn, Chris Small, and Michael D. Smith variously loaned us machines and conference rooms in which to work. We also drew on the resources of Bell Labs in Murray Hill, HP Barcelona, HP Labs Cambridge, HP Cambridge Research Laboratory, and HP Glastonbury.

Our enlightened and flexible managers at Bell Labs, DE Shaw, and HP deserve particular thanks. Al Aho, Wim Sweldens, Rob Pike, and Eric Grosse continued the Bell Labs tradition of allowing authors to write as part of their day jobs. Dick Lampman, Patrick Scaglia, and Rich Zippel supported Josh and Paolo's work on the book, in the tradition of fostering technical excellence at HP Labs. Ron Dror and David Shaw gave Cliff the flexibility to complete this work.

Last and most important, we thank our wives, Elizabeth, Tatiana, and Joyce, for their support, encouragement, tolerance, and patience. Elizabeth has long been used to life like this, but Paolo and Cliff are particularly amazed that Tatiana and Joyce (respectively) married them *while we were working on the book*. We question their judgment, but we are grateful for their grace.

Contents