An Introductory Textbook on Cyber-Physical Systems *

Edward A. Lee
EECS Department
University of California, Berkeley
Berkeley, CA, USA
eal@eecs.berkeley.edu

Sanjit A. Seshia
EECS Department
University of California, Berkeley
Berkeley, CA, USA
sseshia@eecs.berkeley.edu

ABSTRACT

We introduce a textbook that strives to identify and introduce the durable intellectual ideas of embedded systems as a technology and as a subject of study. The emphasis is on modeling, design, and analysis of cyber-physical systems, which integrate computing, networking, and physical processes. The book is intended for students at the advanced undergraduate level or the introductory graduate level, and for practicing engineers and computer scientists who wish to understand the engineering principles of embedded systems. It is also an experiment in publishing. The book is available free in electronic form, in the form of PDF file designed specifically for on-line reading. Specifically, the layout is optimized for medium-sized screens, particularly the iPad and forthcoming tablets. Extensive use of hyperlinks and color enhance the online reading experience. A print version will be available through a print-on-demand service, enabling rapid evolution and immediate correction of errors. See http://LeeSeshia.org.

1. INTRODUCTION

The most visible use of computers and software is processing information for human consumption. We use them to write books (like the one presented here), search for information on the web, communicate via email, and keep track of financial data. The vast majority of computers in use, however, are much less visible. They run the engine, brakes, seatbelts, airbag, and audio system in your car. They digitally encode your voice and construct a radio signal to send it from your cell phone to a base station. They control your microwave oven, refrigerator, and dishwasher. They run printers ranging from desktop inkjet printers to large industrial high-volume printers. They command robots on a factory floor, power generation in a power plant, processes in a chemical plant, and traffic lights in a city. They search for microbes in biological samples, construct images of the inside of a human body, and measure vital signs. They process radio signals from space looking for supernovae and for extraterrestrial intelligence. They bring toys to life, enabling them to react to human touch and to sounds. They control aircraft and trains. These less visible computers are called **embedded systems**, and the software they run is called **embedded software**.

Despite this widespread prevalence of embedded systems, computer science has, throughout its relatively short history, focused primarily on information processing. Only recently have embedded systems received much attention from researchers. And only recently has the community recognized that the engineering techniques required to design and analyze these systems are distinct. Although embedded systems have been in use since the 1970s, for most of their history they were seen simply as small computers. The principal engineering problem was understood to be one of coping with limited resources (limited processing power, limited energy sources, small memories, etc.). As such, the engineering challenge was to optimize the designs. Since all designs benefit from optimization, the discipline was not distinct from anything else in computer science. It just had to be more aggressive about applying the same optimization techniques.

Recently, the community has come to understand that the principal challenges in embedded systems stem from their interaction with physical processes, and not from their limited resources. The term **cyber-physical systems** (**CPS**) was coined by Helen Gill at the National Science Foundation in the U.S. to refer to the integration of computation with physical processes. In CPS, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. The design of such systems, therefore, requires understanding the joint dynamics of computers, software, networks, and physical processes. It is this

*This work was supported in part by NSF CAREER grant #0644436, an Alfred P. Sloan Research Fellowship, and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #CCR-0225610 (ITR), #0720882 (CSR-EHS: PRET), #0647591 (CSR-SGER), #0931843 (ActionWebs) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyc), and the following companies: Agilent, Bosch, National Instruments, Thales, and Toyota.

Workshop on Embedded Systems Education October 28, 2010, Scottsdale, USA.

study of *joint* dynamics that sets this discipline apart.

When studying CPS, certain key problems emerge that are rare in so-called general-purpose computing. For example, in general-purpose software, the time it takes to perform a task is a *performance* issue, not a *correctness* issue. It is not incorrect to take longer to perform a task. It is merely less convenient and therefore less valuable. In CPS, the time it takes to perform a task may be critical to correct functioning of the system. In the physical world, as opposed to the cyber world, the passage of time is inexorable.

In CPS, moreover, many things happen at once. Physical processes are compositions of many things going on at once, unlike software processes, which are deeply rooted in sequential steps. Abelson and Sussman [1] describe computing as "procedural epistemology," knowledge through procedure. In the physical world, by contrast, processes are rarely procedural. Physical processes are compositions of many concurrent processes. Measuring and controlling the dynamics of these processes by orchestrating actions that influence the processes are the main tasks of embedded systems. Consequently, concurrency is intrinsic in CPS. Many of the technical challenges in designing and analyzing embedded software stem from the need to bridge an intrinsically sequential semantics with an intrinsically concurrent physical world.

Today, getting computers to work together with physical processes requires technically intricate, low-level design. Embedded software designers are forced to struggle with interrupt controllers, memory architectures, assembly-level programming (to exploit specialized instructions or to precisely control timing), device driver design, network interfaces, and scheduling strategies, rather than focusing on specifying desired behavior. The sheer mass and complexity of these technologies tempts us to focus an introductory course on mastering them. But a better introductory course would focus on how to model and design the joint dynamics of software, networks, and physical processes. Such a course would present the technologies only as today's (rather primitive) means of accomplishing those joint dynamics. The book presented here is our attempt at a textbook for such a course.

2. RELATED BOOKS

Most texts on embedded systems focus on the collection of technologies needed to get computers to interact with physical systems [2, 3, 4, 9, 14, 16, 19, 22, 23]. Others focus on adaptations of computer-science techniques (like programming languages, operating systems, networking, etc.) to dealing with technical problems in embedded systems [5, 6, 18]. While these implementation technologies are (today) necessary for system designers to get embedded systems working, they do not form the intellectual core of the discipline. The intellectual core is instead in models and abstractions that conjoin computation and physical dynamics.

A few textbooks offer efforts in this direction. Jantsch [8] focuses on concurrent models of computation, Marwedel [12] focuses on models of software and hardware behavior, and Sriram and Bhattacharyya [20] focus on dataflow models of signal processing behavior and their mapping onto programmable DSPs. These are excellent starting points. Models and concurrency (such as dataflow) and abstract models of software (such as Statecharts) provide a better starting point than imperative programming languages (like C), interrupts and threads, and architectural annoyances that a designer must work around (like caches). These texts, however, are not suitable for an introductory course. They are either too specialized or too advanced or both. This book is our attempt to provide an introductory text that follows the spirit of focusing on models and their relationship to realizations of systems.

The variety of textbooks on embedded systems that have appeared in recent years is surprising, often reflecting the perspective of a more established discipline that has migrated into embedded systems, such as VLSI design, control systems, signal processing, robotics, real-time systems, or software engineering. Some of these books complement the one we present nicely. We strongly recommend them to the reader who wishes to broaden his or her understanding of the subject.

Specifically, Patterson and Hennessey [17], although not focused on embedded processors, is the canonical reference for computer architecture, and a must-read for anyone interested embedded processor architectures. Sriram and Bhattacharyya [20] focus on signal processing applications, such as wireless communications and digital media, and give particularly good coverage to dataflow programming methodologies. Wolf [23] gives an excellent overview of hardware design techniques and microprocessor architectures and their implications for embedded software design. Mishra and Dutt [13] give a view of embedded architectures based on architecture description languages (ADLs). Oshana [15] specializes in DSP processors from Texas Instruments, giving an overview of architectural approaches and a sense of assembly-level programming.

Focused more on software, Buttazzo [5] is an excellent overview of scheduling techniques for real-time software. Liu [11] gives one of the best treatments yet of techniques for handling sporadic real-time events in software. Edwards [6] gives a good overview of domain-specific higher-level programming languages used in some embedded system designs. Pottie and Kaiser [18] give a good overview of networking technologies, particularly wireless, for embedded systems.

No single textbook can comprehensively cover the breadth of technologies available to the embedded systems engineer. We have found useful information in many of the books that focus primarily on today's design techniques [2, 3, 4, 7, 9, 14, 16, 19].

3. THEME OF THE BOOK

The major theme of our book is on models and their relationship to realizations of systems. The models we study are primarily about dynamics, the evolution of a system state in time. We do not address structural models, which represent static information about the construction of a system, although these too are important to embedded system design.

Working with models has a major advantage. Models can have formal properties. We can say definitive things about models. For example, we can assert that a model is deterministic, meaning that given the same inputs it will always produce the same outputs. No such absolute assertion is possible with any physical realization of a system. If our model is a good abstraction of the physical system (meaning that it omits only inessential details), then the definitive assertion about the model gives us confidence in the physical realization of the system. Such confidence is enormously valuable, particularly for embedded systems where malfunc-

tions can threaten human lives. Studying models of systems gives us insight into how those systems will behave in the physical world.

Our focus is on the interplay of software and hardware with the physical environment in which they operate. This requires explicit modeling of the temporal dynamics of software and networks and explicit specification of concurrency properties intrinsic to the application. The fact that the implementation technologies have not yet caught up with this perspective should not cause us to teach the wrong engineering approach. We should teach design and modeling as it should be, and enrich this with a *critical* presentation of how to (partially) accomplish our objectives with today's technology. Embedded systems technologies today, therefore, should not be presented dispassionately as a collection of facts and tricks, as they are in many of the above cited books, but rather as stepping stones towards a sound design practice. The focus should be on what that sound design practice is, and on how today's technologies both impede and achieve it.

Stankovic et al. [21] support this view, stating that "existing technology for RTES [real-time embedded systems] design does not effectively support the development of reliable and robust embedded systems." They cite a need to "raise the level of programming abstraction." We argue that raising the level of abstraction is insufficient. We have to also fundamentally change the abstractions that are used. Timing properties of software, for example, cannot be effectively introduced at higher levels of abstraction if they are entirely absent from the lower levels of abstraction on which these are built.

We require robust and predictable designs with repeatable temporal dynamics [10]. We must do this by building abstractions that appropriately reflect the realities of cyber-physical systems. The result will be CPS designs that can be much more sophisticated, including more adaptive control logic, evolvability over time, improved safety and reliability, all without suffering from the brittleness of today's designs, where small changes have big consequences.

In addition to dealing with temporal dynamics, CPS designs invariably face challenging concurrency issues. Because software is so deeply rooted in sequential abstractions, concurrency mechanisms such as interrupts and multitasking, using semaphores and mutual exclusion, loom large. We therefore devote considerable effort in this book to developing a critical understanding of threads, message passing, deadlock avoidance, race conditions, and data determinism.

4. ORGANIZATION OF THE BOOK

As shown in Figure 1, this book is divided into three major parts, focused on **modeling**, **design**, and **analysis**. Modeling is the process of gaining a deeper understanding of a system through imitation. Models imitate the system and reflect properties of the system. Models specify **what** a system does. Design is the structured creation of artifacts. It specifies **how** a system does what it does. Analysis is the process of gaining a deeper understanding of a system through dissection. It specifies **why** a system does what it does (or fails to do what a model says it should do).

The three parts of the book are relatively independent of one another and are largely meant to be read concurrently. Strong dependencies between chapters are shown with arrows in black. Weak dependencies are shown in grey. When there is a weak dependency from chapter i to chapter j, then j may mostly be read without reading i, at most requiring skipping some examples or specialized analysis techniques. A systematic reading of the text can be accomplished in seven segments, shown with dashed outlines. Each segment includes two chapters, so complete coverage of the text is possible in a 14 week semester, assuming each of the seven modules takes two weeks. We now briefly describe the three main parts.

4.1 Modeling

This part of the book focuses on models of dynamic behavior. It begins with a light coverage of the modeling of physical dynamics, specifically focusing on continuous dynamics in time. It then talks about discrete dynamics, using state machines as the principal formalism. It then combines the two with a discussion of hybrid systems. The fourth chapter focuses on concurrent composition of state machines, emphasizing that the semantics of composition is a critical issue that designers must grapple with. The fifth chapter gives an overview of concurrent models of computation, including many of those used in design tools that practitioners frequently leverage, such as Simulink and LabVIEW.

In this part of the book, we define a **system** to be simply a combination of parts that is considered a whole. A **physical system** is one realized in matter, in contrast to a conceptual or **logical system** such as software and algorithms. The **dynamics** of a system is its evolution in time: how its state changes. A **model** of a physical system is a description of certain aspects of the system that is intended to yield insight into properties of the system. In this text, models have mathematical properties that enable systematic analysis. The model imitates properties of the system, and hence yields insight into that system.

A model is itself a system. It is important to avoid confusing a model and the system that it models. These are two distinct artifacts. A model of a system is said to have high **fidelity** if it accurately describes properties of the system. It is said to **abstract** the system if omits details. Models of physical systems inevitably *do* omit details, so they are always abstractions of the system. A major goal of this text is to develop an understanding of how to use models, of how to leverage their strengths and respect their weaknesses.

A cyber-physical system (CPS) is a system composed of physical subsystems together with computing and networking. Models of cyber-physical systems must include all three parts. The models will typically need to represent both **static properties** (those that do not change during the operation of the system) and dynamics.

Each of the modeling techniques described in this part of the book is an enormous subject, much bigger than one chapter, or even one book. In fact, such models are the focus of many branches of engineering, physics, chemistry, and biology. Our approach is aimed at engineers. We assume some background in mathematical modeling of dynamics (calculus courses that give some examples from physics are sufficient), and then focus on how to compose diverse models. This will form the core of the cyber-physical system problem, since joint modeling of the cyber side, which is logical and conceptual, with the physical side, which is embodied in matter, is the core of the problem. We therefore make no attempt to be comprehensive, but rather pick a few modeling techniques that are widely used by engineers and well

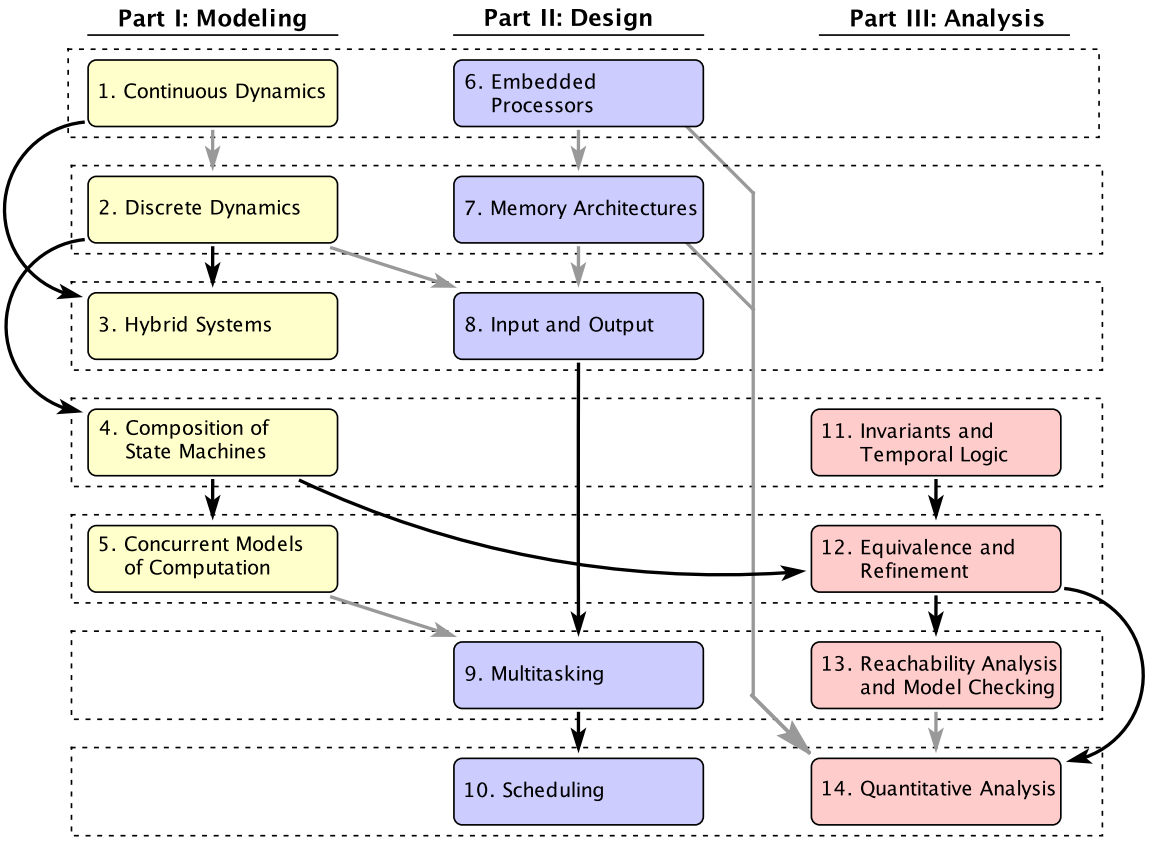

Figure 1: Map of the book with strong and weak dependencies between chapters.

understood, review them, and then compose them to form a cyber-physical whole.

4.2 Design

The second part of the book has a very different flavor, reflecting the intrinsic heterogeneity of the subject. This part focuses on the design of embedded systems, with emphasis on the role they play *within* a CPS. Chapter 6 discusses processor architectures, with emphasis on specialized properties most suited to embedded systems. Chapter 7 describes memory architectures, including abstractions such as memory models in programming languages, physical properties such memory technologies, and architectural properties such as memory hierarchy (caches, scratchpads, etc.). The emphasis is on how memory architecture affects dynamics. Chapter 8 is about the interface between the software world and the physical world. It discusses input/output mechanisms in software and computer architectures, and the digital/analog interface, including sampling. Chapter 9 introduces the notions that underly operating systems, with particularly emphasis on multitasking. The emphasis is on the pitfalls of using low-level mechanisms such as threads, with a hope of convincing the reader that there is real value in using the modeling techniques covered in the first part of the book. Chapter 10 introduces real-time scheduling, covering many of the classic results in the area.

In all chapters in this part, we particularly focus on the mechanisms that provide concurrency and control over timing, because these issues loom large in the design of cyber-physical systems. When deployed in a product, embedded processors typically have a dedicated function. They control an automotive engine or measure ice thickness in the Arctic. They are not asked to perform arbitrary functions with user-defined software. Consequently, the processors, memory architectures, I/O mechanisms, and operating systems can be more specialized. Making them more specialized can bring enormous benefits. For example, they may consume far less energy, and consequently be usable with small batteries for long periods of time. Or they may include specialized hardware to perform operations that would be costly to perform on general-purpose hardware, such as image analysis. Our goal in this part is to enable the reader to *critically* evaluate the numerous available technology offerings.

One of the goals in this part of the book is to teach students to implement systems while *thinking across traditional abstraction layers* — e.g., hardware *and* software, computation *and* physical processes. While such cross-layer thinking is valuable in implementing systems in general, it is particularly essential in embedded systems given their heterogeneous nature. For example, a programmer implementing a control algorithm expressed in terms of real-valued quantities must have a solid understanding of computer arithmetic (e.g., of fixed-point representations) in order to create a reliable implementation. Similarly, an implementor of automotive software that must satisfy real-time constraints must be aware of processor features – such as pipelining and caching – that can affect the execution time of tasks and hence the real-time behavior of the system. Likewise, an implementor of interrupt-driven or multi-threaded software must understand the level of atomicity provided by the

underlying software-hardware platform and use appropriate synchronization constructs to ensure correctness. Rather than doing an exhaustive survey of different implementation methods and platforms, this part of the book seeks to give the reader an appreciation for such cross-layer topics, and uses homework exercises to facilitate a deeper understanding of them.

4.3 Analysis

Every system must be designed to meet certain requirements. For embedded systems, which are often intended for use in safety-critical, everyday applications, it is essential to certify that the system meets its requirements. Such system requirements are also called **properties** or **specifications**. The need for specifications is aptly captured by the following quotation (paraphrased from [24]):

> "A design without specifications cannot be right or wrong, it can only be surprising!"

This part of the book focuses on precise specifications of properties, on techniques for comparing specifications, and on techniques for analyzing specifications and the resulting designs. Reflecting the emphasis on dynamics in the text, Chapter 11 focuses on temporal logics, which provide precise descriptions of dynamic properties of systems. These descriptions are treated as models. Chapter 12 focuses on the relationships between models. Is one model an abstraction of another? Is it equivalent in some sense? Specifically, that chapter introduces type systems, as a way of comparing static properties of models, and language containment and simulation relations as a way of comparing dynamic properties of models. Chapter 13 focuses on techniques for analyzing the large number of possible dynamic behaviors that a model may exhibit, with emphasis on model checking as a technique for exploring such behaviors. Chapter 14 is about analyzing quantitative properties of embedded software, such as finding bounds on resources consumed by programs. It focuses particularly on execution time analysis, with some introduction to others such as energy and memory usage.

In present engineering practice, it is common to have system requirements stated in a natural language such as English. It is important to precisely state requirements to avoid ambiguities inherent in natural languages. The goal of this part of the book to help replace descriptive techniques with *formal* ones, which we believe are less error prone.

Importantly, formal specifications also enable the use of automatic techniques for formal verification of both models and implementations. This part of the book introduces readers to the basics of formal verification, including notions of equivalence and refinement checking, as well as reachability analysis and model checking. In discussing these verification methods, we take a *user's view* of them, discussing, for example, how model checking can be applied to find subtle errors in concurrent software, or how reachability analysis can be used in computing a control strategy for a robot to achieve a particular task.

4.4 Missing Sections

Version 1.0 of the book will not be complete. It is arguable, in fact, that complete coverage of embedded systems in the context of CPS is impossible. Specific topics that we cover in the undergraduate Embedded Systems course at Berkeley[1] and hope to include in version 2.0 or later versions of the book include sensors and actuators, networking, fault tolerance, simulation techniques, control systems, and hardware/software codesign. If you are an author interested in contributing, please contact us at authors@LeeSeshia.org.

4.5 Intended Audience

This book is intended for students at the advanced undergraduate level or the introductory graduate level, and for practicing engineers and computer scientists who wish to understand the engineering principles of embedded systems. We assume that the reader has some exposure to machine structures (e.g., should know what an ALU is), computer programming (we use C throughout the text), basic discrete mathematics and algorithms (e.g., graph traversal through depth-first or breadth-first search), and at least an appreciation for signals and systems (what it means to sample a continuous-time signal, for example).

5. PUBLICATION STRATEGY

Recent advances in technology are fundamentally changing the technical publishing industry. Almost every aspect of how academics, teachers, and intellectuals communicate is in flux, and we believe that the landscape in the 21st century will be very different from that of the 20th century. In response to this, the book we introduce here is an experiment in publishing. With apologies to the many hard-working men and women in the traditional publishing industry, we hope that if this approach is successful, that it will be followed by other authors.

The book is available free in electronic form from the website http://LeeSeshia.org. In recognition that there is real value in a tangible manifestation on paper, something you can thumb through, something that can live on a bookshelf to remind you of its existence, the book will also be available in print form. Our plan is to use a print-on-demand service, which has the advantages of dramatically reduced cost to the reader and the ability to quickly and frequently update the version of the book to correct errors and discuss new technologies.

The reduced revenue stream that results from this publication strategy, of course, has disadvantages. But prior experience of these authors indicate that it is rare for authors of technical books to even earn minimum wage for their efforts. Remuneration is clearly not the main goal. The main goals seem to be communication, education, and impact. We believe that our strategy enhances all three goals.

Consider for example the opportunities afforded by a focus on on-line dissemination. The electronic version is a PDF file designed specifically for on-line reading. The layout is optimized for medium-sized screens, particularly the iPad and forthcoming tablets. Extensive use of hyperlinks and color enhance the online reading experience. Links to external websites are included directly at the relevant point.

We attempted to adapt the book to e-book formats, which, in theory, enable reading on various sized screens, attempting to take best advantage of the available screen. However, like HTML documents, e-book book formats are a reflow technology, where page layout is recomputed on the fly. The results are highly dependent on the screen size and prove ludicrous on many screens and suboptimal on all. As a conse-

[1] See http://chess.eecs.berkeley.edu/eecs149/

quence, we have opted for controlling the layout, and we do not recommend attempting to read the book on an iPhone.

As of this writing, a preliminary version of the book is available at http://LeeSeshia.org. We plan a complete release of version 1.0 by the end of summer, 2010, with an associated print-on-demand version available in the Fall.

6. ACKNOWLEDGEMENTS

The authors gratefully acknowledge contributions and helpful suggestions on the text from Elaine Cheong, Gage Eads, Stephen Edwards, Shanna-Shaye Forbes, Jeff Jensen, Wenchao Li, Isaac Liu, Slobodan Matic, Steve Neuendorffer, Minxue Pan, Hiren Patel, Jan Reineke, Chris Shaver, Stavros Tripakis, Pravin Varaiya, Maarten Wiggers, and the students in UC Berkeley's EECS 149 class, particularly Ned Bass and Dan Lynch.

7. REFERENCES

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, second edition, 1996.

[2] M. Barr and A. Massa. *Programming Embedded Systems*. O'Reilly, 2nd edition, 2006.

[3] A. S. Berger. *Embedded Systems Design: An Introduction to Processes, Tools, & Techniques*. CMP Books, 2002.

[4] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX*. Addison-Wesley, 3d edition, 2001.

[5] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, second edition, 2005.

[6] S. A. Edwards. *Languages for Digital Embedded Systems*. Kluwer Academic Publishers, 2000.

[7] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design - Modeling, Synthesis, and Verification*. Springer, 2009.

[8] A. Jantsch. *Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2003.

[9] R. Kamal. *Embedded Systems: Architecture, Programming, and Design*. McGraw Hill, 2008.

[10] E. A. Lee. Computing needs time. Technical Report UCB/EECS-2009-30, EECS Department, University of California, Berkeley, February 18 2009.

[11] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[12] P. Marwedel. *Embedded System Design*. Kluwer Academic Publishers, 2003.

[13] P. Mishra and N. D. Dutt. *Functional Verification of Programmable Embedded Processors - A Top-down Approach*. Springer, 2005.

[14] T. Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Elsevier, 2005.

[15] R. Oshana. *DSP Software Development Techniques for Embedded and Real-Time Systems*. Embedded Technology Series. Elsevier, 2006.

[16] J. S. Parab, V. G. Shelake, R. K. Kamat, and G. M. Naik. *Exploring C for Microcontrollers*. Springer, 2007.

[17] D. A. Patterson and J. L. Hennessey. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[18] G. Pottie and W. Kaiser. *Principles of Embedded Networked Systems Design*. Cambridge University Press, 2005.

[19] D. E. Simon. *An Embedded Software Primer*. Addison-Wesley, 2006.

[20] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC press, 2nd edition, 2009.

[21] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *Computer*, pages 23–31, 2005.

[22] J. W. Valvano. *Embedded Microcomputer Systems - Real Time Interfacing*. Thomson, 2nd edition, 2007.

[23] W. Wolf. *Computers as Components: Principles of Embedded Computer Systems Design*. Morgan Kaufman, 2000.

[24] W. Young, W. Boebert, and R. Kain. Proving a computer system secure. *Scientific Honeyweller*, 6(2):18–27, July 1985.