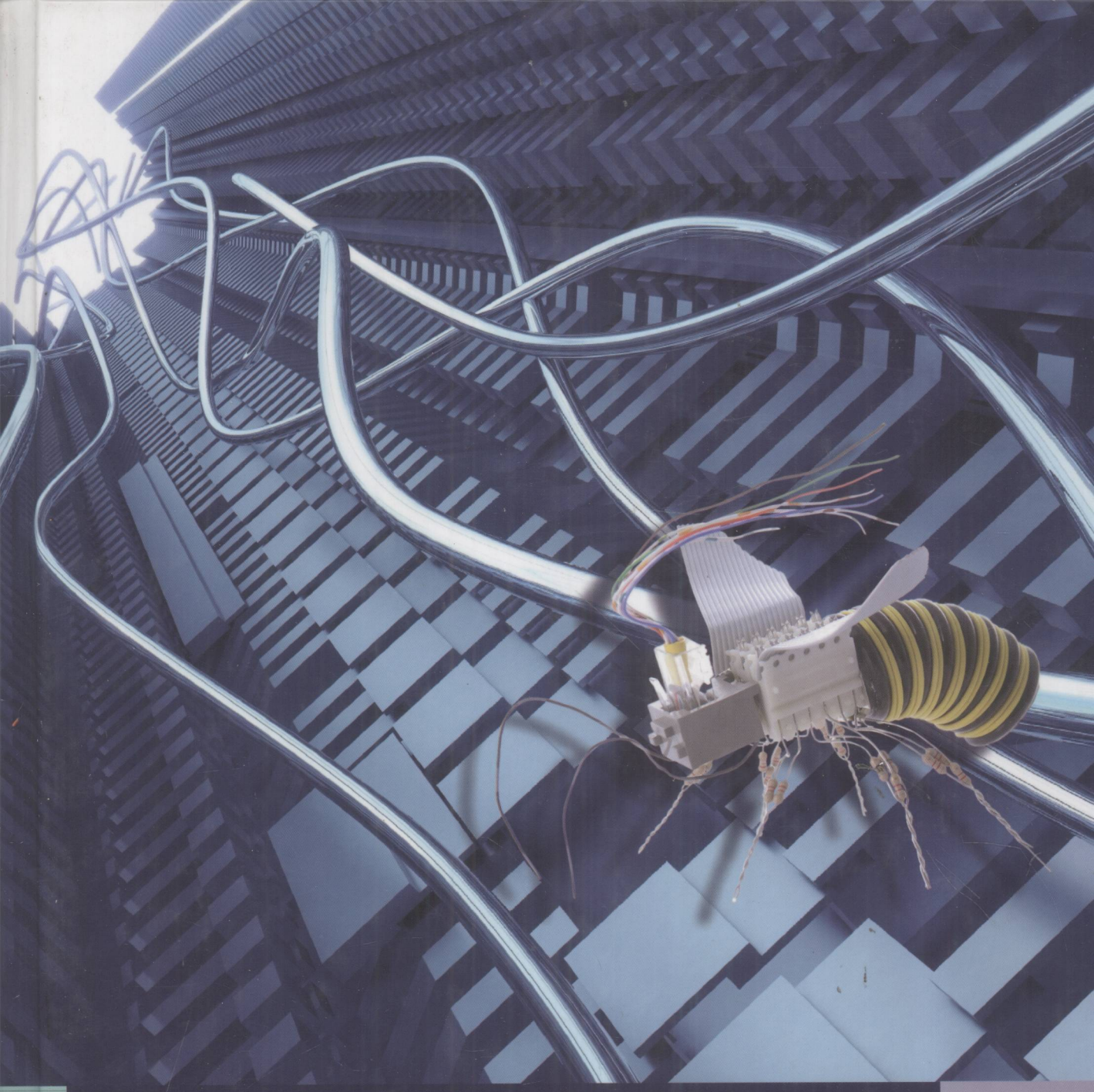

EMBEDDED SYSTEMS AND SOFTWARE VALIDATION

SYSTEMS ON SILICON

ABHIK ROYCHOUDHURY

MK®

Embedded Systems and Software Validation

Abhik Roychoudhury

Department of Computer Science
National University of Singapore

ELSEVIER

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

Morgan Kaufmann Publishers is an imprint of Elsevier

MORGAN
KAUFMANN

Morgan Kaufmann Publishers is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data
Roychoudhury, Abhik.
Embedded systems and software validation / Abhik Roychoudhury.
p. cm. – (The Morgan Kaufmann series in systems on silicon)
Includes bibliographical references and index.
ISBN 978-0-12-374230-8 (hardcover : alk. paper)
1. Embedded computer systems–Design and construction. 2. Embedded computer systems–Testing. 3. Computer software–Testing. I. Title.
TK7895.E42R72 2009
004.1–dc22

2009011196

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

ISBN 13: 978-0-12-374230-8

Printed and bound in United States of America
09 10 9 8 7 6 5 4 3 2 1

Embedded Systems and Software Validation

The Morgan Kaufmann Series in Systems on Silicon

Series Editor: Wayne Wolf, Georgia Institute of Technology

The Designer's Guide to VHDL, Second Edition
Peter J. Ashenden

The System Designer's Guide to VHDL-AMS
Peter J. Ashenden, Gregory D. Peterson, and Darrell A. Teegarden

Modeling Embedded Systems and SoCs
Axel Jantsch

ASIC and FPGA Verification: A Guide to Component Modeling
Richard Munden

Multiprocessor Systems-on-Chips
Edited by Ahmed Amine Jerraya and Wayne Wolf

Functional Verification
Bruce Wile, John Goss, and Wolfgang Roesner

Customizable and Configurable Embedded Processors
Edited by Paolo Ienne and Rainer Leupers

Networks-on-Chips: Technology and Tools
Edited by Giovanni De Micheli and Luca Benini

VLSI Test Principles & Architectures
Edited by Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen

Designing SoCs with Configured Processors
Steve Leibson

ESL Design and Verification
Grant Martin, Andrew Piziali, and Brian Bailey

Aspect-Oriented Programming with e
David Robinson

Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation
Edited by Scott Hauck and André DeHon

System-on-Chip Test Architectures
Edited by Laung-Terng Wang, Charles Stroud, and Nur Touba

Verification Techniques for System-Level Design
Masahiro Fujita, Indradeep Ghosh, and Mukul Prasad

VHDL-2008: Just the New Stuff
Peter J. Ashenden and Jim Lewis

On-Chip Communication Architectures: System on Chip Interconnect
Sudeep Pasricha and Nikil Dutt

Embedded DSP Processor Design: Application Specific Instruction Set Processors
Dake Liu

Processor Description Languages: Applications and Methodologies
Edited by Prabhat Mishra and Nikil Dutt

Three-dimensional Integrated Circuit Design
Vasilis F. Pavlidis and Eby G. Friedman

Electronic Design Automation: Synthesis, Verification, and Test
Edited by Laung-Terng Wang, Kwang-Ting (Tim) Cheng, Yao-Wen Chang

Embedded Systems and Software Validation
Abhik Roychoudhury

To Jishnu

Acknowledgments

This book owes a lot to all my students, colleagues, and co-workers. It is by working with them over the past decade that I have discovered the issues and challenges in the field of embedded systems validation. So, first and foremost, I must thank them all.

I have written this book off and on, in the course of my teaching and research work at the National University of Singapore (NUS). Funding from a University Research Council project at NUS is gratefully acknowledged.

A leave from NUS in 2007 to the Indian Institute of Science (IISc) infused in me the energy to start writing the book. The calm environs of the IISc campus helped set the mood for writing this book.

The support of Elsevier staff was instrumental in ensuring that the book has proceeded on schedule.

Finally, playing with my 5-year-old son Jishnu allowed me to absorb the pressures of writing the book in the midst of various deadlines and commitments. Thanks, Jishnu!

Singapore
19 January 2009

Preface

This book attempts to cover the issues in validation of embedded software and systems. There are many books on this topic, as a Web search with the appropriate search terms will reveal. So, why this book?

There are several ways to answer the question. The first, most direct answer is that the current books mostly deal with the programming and/or co-design of embedded systems. Validation is often discussed almost as an afterthought. In this book, we treat validation as a first-class citizen in the design process, weaving it into the design process itself.

The focus of our book is on validation, but from an embedded software and systems perspective. The methods we have covered (testing/model-checking) can also be covered from a completely general perspective, focusing only on the techniques, rather than on how they fit into the system design process. But we have not done so. Even though the focus of the book is on validation methods, we clearly show how it fits into system design. As an example, we present and discuss the model-checking method twice in two different ways — once at the level of system model (Chapter 2) and again at the level of system implementation (Chapter 5).

Finally, being rooted in embedded software and systems, the focus of our book is not restricted to functionality validation. We have covered at least two other aspects — debugging of performance and communication behavior. As a result, this book contains analysis methods that are rarely found in a single book — testing (informal validation), model checking (formal validation), worst-case execution time analysis (static analysis for program performance), schedulability analysis (system level performance analysis), and so on — all blended under one cover, with the goal of reliable embedded system design.

As for the chapters of the book, Chapter 1 gives a general introduction to the issues in embedded system validation. Differences between functionality and performance validation are discussed at a general level.

Chapter 2 discusses model-level validation. It starts with generic discussions on system structure and behavior, and zooms into behavioral modeling notations such as finite-state machines (FSMs) and message sequence charts (MSCs). Simulation, testing, and formal verification of these models are discussed. We discuss model-based testing, where test cases generated from the model are tried out on the system implementation. We also discuss property verification, and the well-known model-checking method. The chapter ends with a nice hands-on discussion of practical validation tools such as SPIN and SMV. Thus, this chapter corresponds to *model-level debugging*.

Chapter 3 discusses the issues in resolving communication incompatibilities between embedded system components. We discuss different strategies for resolving such incompatibilities, such as endowing the components with appropriate interfaces, and/or constructing a centralized communication protocol converter. Thus, this chapter corresponds to *communication debugging*.

Chapter 4 discusses system-level performance validation. We start with software timing analysis, in particular worst-case execution time (WCET) analysis. This is followed by the estimation of time spent as a result of different interferences in a program execution — from the external environment, or from other executing programs on the same or different processing elements. Suitable analysis methods to estimate the time due to such interferences are discussed. We then discuss mechanisms to combat execution-time unpredictability via system-level support. In particular, we discuss compiler-controlled memories or scratchpad memories. The chapter concludes with a discussion on time predictability issues in emerging applications. Thus, this chapter corresponds to *performance debugging*.

Chapter 5 discusses functionality debugging of embedded software. We discuss both formal and informal approaches, with almost equal emphasis on testing and formal verification. The first half of the chapter involves validation methods built on testing or dynamic analysis. The second half of the chapter concentrates on formal verification, in particular software model checking. The chapter concludes with a discussion on combining formal verification with testing. Thus, this chapter corresponds to *software debugging*.

Apart from some debugging/validation methods being common to Chapters 2 and 5, the readers may try to read the chapters independently. A senior undergraduate or graduate course on this topic may, however, read the chapters in sequence, that is, Chapters 2, 3, 4, 5.

ABOUT THE AUTHOR

Abhik Roychoudhury received his M.S. and Ph.D. in Computer Science from the State University of New York at Stony Brook in 1997 and 2000, respectively. His research has focused on formal verification and analysis methods for system design, with focus on embedded software and systems. In these areas, his research group has been involved in building practical program analysis and software productivity tools that enhance software quality as well as programmer productivity. Two meaningful examples of such endeavors are the JSlice dynamic analysis tool for Java program debugging, and the Chronos static analysis tool for ensuring time-predictable execution of embedded software. His awards include a 2008 IBM Faculty Award. Since 2001, Abhik has been at the School of Computing in the National University of Singapore, where he is currently an Associate Professor.

Contents

CHAPTER

Introduction 1

Embedded software and systems have come to dominate the way we interact with computers and computation in our everyday lives. Computers are no longer isolated entities sitting on our desks. Instead, they are nicely woven and integrated into our everyday lives via the gadgets we directly or indirectly use — mobile phones, washing machines, microwaves, automotive control, and flight control. Indeed, embedded systems are so pervasive, that they perform the bulk of the computation today — putting forward "embedded computing" as a new paradigm to study. In this book, we focus on validation of embedded software and systems, for developing embedded systems with reliable functionality and timing behavior.

Not all embedded systems are safety-critical. One one hand, there are the safety-critical embedded systems such as automobiles, transportation (train) control, flight control, nuclear power plants, and medical devices. On the other hand, there are the more vanilla, or less safety-critical, embedded systems such as mobile phones, HDTV, controllers for household devices (such as washing machines, microwaves, and air conditioners), smart shirts, and so on. Irrespective of whether an embedded system is safety-critical or not, the need for integrating validation into every stage of the design flow is clearly paramount. Of course, for safety-critical embedded systems, there is need for more stringent validation — so much so that formal analysis methods, which give mathematical guarantees about functionality/timing properties of the system, may be called for at least in certain stages of the design.

Our focus in this book is on validation methods, and how they can be woven into the embedded system design process. Before proceeding further, let us intuitively explain some common terminologies that arise in validation — *testing*, *simulation*, *verification*, and *performance analysis*.

- *Testing* refers to checking that a system behaves as expected for a given input. Here the system being checked can be the actual system that will be executed. However, note that it is only being checked for a given input, and not all inputs.

Embedded Systems and Software Validation

- *Simulation* refers to running a system for a given input. However, simulation differs from actual system execution in one (or both) of the following ways.
  - The system being simulated might only be a model of the actual system to be executed. This is useful for functionality simulation — check out the functionality of a system model for selected inputs before constructing the actual system.
  - The execution platform on which the system is being simulated is different from the actual execution platform. This situation is very common for performance simulations. The execution platform on which the actual system will be executed may not be available, or it might be getting decided through the process of performance simulations. Typically, a software model of the execution platform might be used for performance simulations.
- *Formal verification* refers to checking that a system behaves as expected for all possible inputs. Because exhaustive testing is inefficient or even infeasible, verification may be achieved by statically analyzing a system model (which may be represented by a structure such as a finite-state machine).
- Finally, we note that formal verification methods have conventionally been used for giving strict mathematical guarantees about the functionality of a system. However, to give strict guarantees about performance (for example, to give an upper bound on the execution time of a given software), one needs to employ mathematical analysis techniques for estimating performance. Such techniques often go by the name of *performance analysis*.

In order to see what the possibilities and opportunities are in terms of integrating validation into embedded system design flows, we can look at the automobile industry. It is widely recognized that automotive electronics is a wide market, with more and more functionalities in modern-day cars being software-controlled. Indeed, innovations in automotive software can bring about new designs, a point often articulated by car manufacturers themselves. The by-now famous quotes such as "more than 90% of the innovation in a modern-day car is from the software" stand testimony to the importance of embedded software/systems in the design of a modern-day car. Naturally, because of the importance of the various car components (brakes, airbags, etc.) functioning "correctly" during the driving of a car, rigorous validation of the hardware/software controlling these components is crucial. In other words, reliable and robust embedded system design flows that integrate extensive debugging/validation are a *must*.

To see further the importance of validation in embedded systems for automobiles, we can delve deeper into the various components of a car, which can be computer-controlled. Roughly speaking, these can be divided into three categories — engine features, cabin features, and entertainment. Clearly, the engine features are

the most safety-critical and the features related to in-vehicle entertainment are the least safety-critical. The engine features include critical features such as the brake and steering wheel; usually these features involve hard real-time constraints. The cabin features include less critical (but important) features such as power windows and air conditioning. The entertainment or infotainment features include control of in-car devices such as GPS navigation systems, CD player, and in-car television, as well as communication between these devices. Clearly, the computing component controlling the engine features (such as brakes) needs very rigorous validation — to the extent that the behavior of these computing components could be subjected to formal modeling and verification. For the cabin features, we at least need modeling and extensive testing of the computing components controlling the cabin features. For the infotainment features, we need performance analysis methods to ensure that the soft real-time constraints are satisfied.

Thus, as we can see from the discussion on the specific domain of automotive software, validation of different kinds are required for a complex embedded system. For the more safety-critical parts of the system, rigorous modeling and formal verification may be needed. For the less safety-critical parts, more extensive testing may be sufficient. Moreover, for the parts of the system controlling or ensuring real-time responses to/from the environment, detailed performance validation needs to be carried out. Thus, the validation methods we employ can range from formal methods (such as model checking) to informal ones (such as testing). Moreover, the level of abstraction at which we employ the validation may vary — model-level validation; or high-level implementation validation (where we consider only the inter-component behavior without looking inside the components); or low-level implementation validation (where we also look inside the system components). Finally, the criteria for validation may also vary — we may perform validation at different levels, to check for functionality errors, timing errors, and so on.

Figure 1.1 visually depicts the intricacies of embedded system validation. In particular, Figure 1.1a shows the different levels (model/implementation) and criteria (performance/functionality) of system validation.

Figure 1.1b illustrates the complications in functionality validation. For an embedded system that we seek to construct, we may design and elaborate it at different levels of details (or different levels of abstraction). If we are seeking functionality validation, then the higher the level of detail, the lower the formality of the validation method. Thus, for system design at higher levels of abstraction, we may try out fully formal validation methods. On the other hand, as we start fleshing out the implementation details of the system under construction, we may settle for more informal validation methods such as extensive testing.

As opposed to functionality validation, the picture appears somewhat different for timing validation — see Figure 1.1c. As is well understood, embedded systems

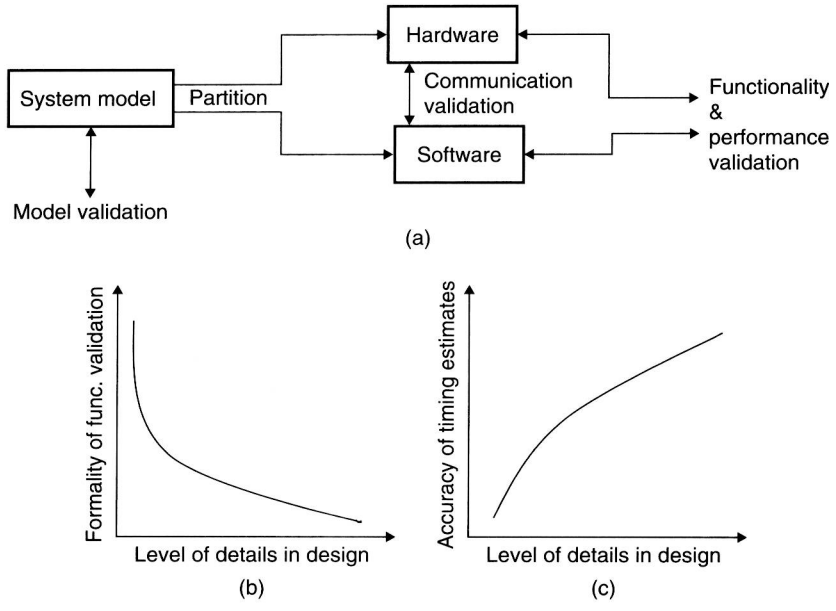

Figure 1.1

Issues in functionality and timing validation of embedded systems.

often incorporate hard or soft real-time constraints on interaction of the system with its physical environment — or, for that matter, interactions between the different components of the system. Hence, timing validation involves developing accurate estimates of the "system response time" (in response to some event from the environment). Clearly, as the details of the embedded system are fleshed out, we can develop more accurate timing estimates and, in that sense, perform more detailed timing validation.

Thus, Figure 1.1 shows the issues in validating functionality versus validating timing properties — both of which are of great importance in embedded system design flows. Two different aspects are being highlighted here:

- Formal verification of functionality is better conducted at higher levels of abstraction. As we start considering lower level details, formal approaches do not scale up, and informal validation methods such as testing come into play.
- For performance validation, as we consider lower level details, our performance estimates are more accurate.

The reader should note that other criteria along which embedded system validation may proceed, such as estimating the energy or area requirements of a system,

also have certain basic similarities with timing validation. As the system design is elaborated in more detail, we can form a better idea about its timing, energy, and area requirements.

In the following chapters, we study embedded software/systems validation from various angles:

- Model-level validation (mostly functionality) — Chapter 2
- Implementation-level validation
  - High-level validation of intercomponent communication — Chapter 3
  - Low-level implementation validation
    - Performance debugging — Chapter 4
    - Functionality debugging — Chapter 5

So, let us get on with the ride — studying various debugging/validation methods for design of reliable embedded software and systems.