Roopak Sinha · Parthasarathi Roop
Samik Basu

Correct-by-Construction Approaches for SoC Design

Springer

Roopak Sinha • Parthasarathi Roop • Samik

Correct-by-Construction Approaches for SoC Design

Springer

Roopak Sinha
Electrical and Computer Engineering
The University of Auckland
Auckland, New Zealand

Parthasarathi Roop
Electrical and Computer Engineering
The University of Auckland
Auckland, New Zealand

Samik Basu
Department of Computer Science
Iowa State University
Ames, IA, USA

ISBN 978-1-4614-7863-8 ISBN 978-1-4614-7864-5 (eBook)
DOI 10.1007/978-1-4614-7864-5
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013944687

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Correct-by-Construction Approaches for SoC Design

Roopak: To Anaya and Dhrov.

Partha: To Bapa, Maa and Guruji.

Samik: To my parents.

Foreword

It is always a good idea to begin a foreword for a computer science book with Moore's law! Indeed, this self-fulfilling prediction has led to a miniaturization such that 2012s processors have up to five billions transistors, such as the Intel's Xeon Phi. However, this miniaturization comes with a proportional complexity price, no engineer being able to conceive a processor with that many transistors. It was therefore natural to use a divide-and-conquer approach, which resulted first in the advent of multi-core processors and system-on-a-chip (SoC) and finally multi-processor-SoC (MPSoC). This is where the field of processor design (hardware) meets the field of component-based design (software). In both fields, the issue is quite simple: how can an engineer design a huge and complex system (be it an MPSoC or a large software system) from simpler and already validated components, possibly third-party? This looks very appealing, but it raises several crucial challenges, which are precisely what this book is about: the design, verification, and validation of systems composed of many components, such that global properties can be preserved and such that mismatches of any kind between the components can be avoided (mismatches either with respect to the control signals, to the clocks, or to the data exchanged between the components). The methods and tools presented in this book will prove essential for designing the next generation of MPSoCs, ever more powerful, ever more embedded, and ever more reliable.

Grenoble, France — Dr. Alain Girault

Preface

A rapid surge in the consumer electronics revolution of the past decade may be attributed to the advancement in the automated design techniques for systems-on-chips (SoCs). SoCs are excellent metaphors for well-known computing terminologies such as *abstraction* and *reuse*, which have been in vogue for many decades both in industry and in academia. The mobile phone stands out as the significant benefactor of the abstraction and reuse approaches in computing. One key ingredient for the success of the mobile phone has been not only the advancement of RF technology but also the rapid evolution of the phone as a multi-functional, user friendly device that can perform wide-ranging tasks from movie making, transport planning to being a portable medical device. Would this have been possible without rapid advancement in SoC design techniques that incorporated state-of-the-art multicore processors, low-power design and many pre-verified components and buses?

The impetus for this monograph has been twofold. Firstly, Partha came across the reuse methodology manual (RMM) as a graduate student. RMM-like design methodology, and its variants practiced in industry, has been very successful in dealing with many challenges of SoC design. However, RMM and similar design approaches state that the system-level verification effort is still a major challenge as it requires significantly higher effort in comparison to other phases of SoC design. This motivated Partha and Roopak in early 2004 to examine this issue. Roopak worked on this problem from 2004 to 2008 and proposed the use of rigorous techniques such as model checking to aid the system-level verification process. They worked jointly with Samik who helped shape this research by leveraging many interesting ideas from software engineering. This joint effort led to many publications in embedded system conferences such as design automation and test in Europe (DATE) and VLSI Design. This research and associated publications form the main foundations of this book.

A second impetus of this monograph is motivated by the rapid adoption of SoCs in safety-critical applications in robotics, automotive and medical devices. Here, a major concern is the overall functional safety of the product, i.e., the device has to be designed to consider all possible risks and safety functions that have been integrated

to mitigate these risks. We noticed that the research we undertook since 2004 may be ideal not only for reducing the system-level verification effort but also for helping safety analysis and associated certification.

While our work is already published and hence available to expert researchers, we felt that a monograph on this topic of system-level verification of SoCs may have wider interest. Firstly, we wanted to make these research results available to researchers, designers and students alike. Hence, we embarked on a pedagogic presentation of formal methods that is widely accessible. We also simplified the process of requirement elicitation of SoCs through natural language-based boiler plates. This aspect of our work was never published before and is a significant addition to facilitate the adoption of formal methods in industry. Secondly, we developed this monograph with practitioners in mind. ARM is one of the most widely adopted platforms for SoCs (http://www.arm.com/community/soc/index.php). Hence, we used ARM and AMBA-based examples throughout this text to motivate our approach.

Last but not the least, our work has been accomplished not in isolation but due to research efforts of many researchers, who have also published on related topics and have proposed many concepts that we have either extended or reused. As it is often stated, "we stood on the shoulder of other researchers", to develop the proposed methodology. We have carefully examined and presented the latest status of this research on the use of formal methods in SoC design and system-level verification (see Chap. 7 for a state-of-the-art review of related research). This may be helpful for starting graduate students, SoC designers and researchers to access such related research quickly.

This being the first version of the book, we envisage that there are some errors and omissions. We welcome any feedback in this regard. We are also developing supplementary material such as exercises and lecture notes for use of this text in advanced undergraduate or postgraduate classes.

Organization and Reading Guide

This book is not a traditional book on formal methods or formal verification. In these books, a set of formal techniques are described, in an algorithmic manner. Our book, in contrast, is developed for engineers so that they may use a design methodology that is based on sound mathematical principles. Hence, it is more a book on system design rather than system analysis, though the latter is built into the design process.

The book is organized into seven chapters. Each chapter deals with some key concepts related to the correct-by-construction design approach for SoCs. Figure 1 provides an overview of the chapters and the main ideas conveyed in them. Each chapter has a set of key concepts (presented in cyan) and the main idea (presented in red).

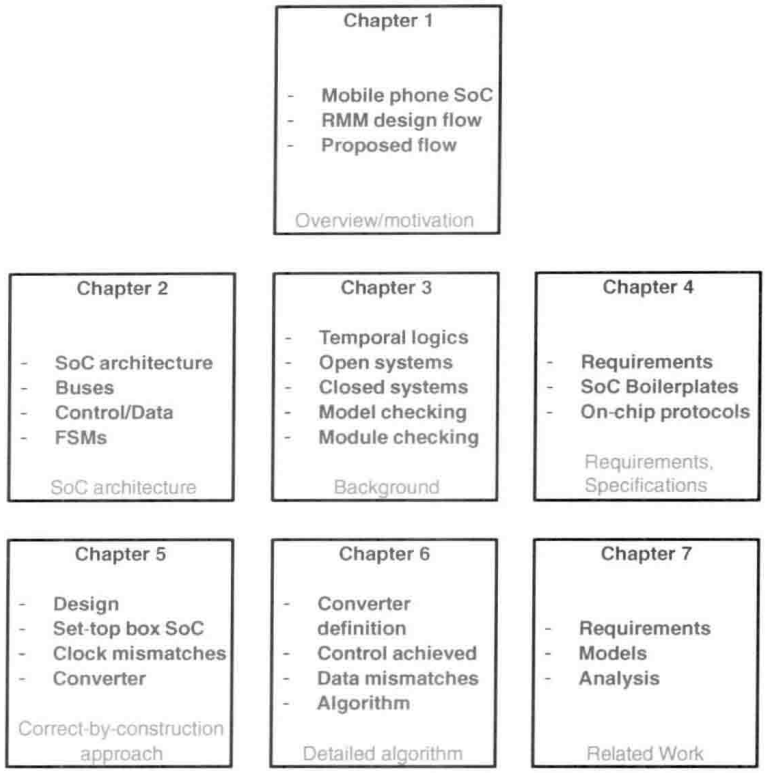

Fig. 1 Key concepts of the different chapters

Chapter 1 provides the overview of SoC design and explains one particular approach based on the reuse methodology manual (RMM) [KB02]. We then motivate the need for the proposed approach, especially to tackle the problem of system-level verification. The key concepts presented in this chapter include the "inside view" of a mobile-phone SoC, the RMM design flow for SoCs and a proposed design flow for the correct-by-construction approach. This chapter may be viewed as the "overview/motivation" chapter.

Chapter 2 provides details of SoC internals, including key concepts such as the SoC architecture based on ARM, internals of buses used, the need for capturing the control-flow and data-flow in on-chip communication protocols of SoC components (also called IPs). This chapter may be viewed as the "SoC architecture" chapter.

Chapter 3 equips the reader with the necessary background material needed for later chapters. Key concepts covered in this chapter include the distinction between closed and open systems, temporal logic (to capture specifications), model checking (to verify a closed system) and module checking (to verify open systems). This chapter may be viewed as the "background" chapter.

Chapter 4 provides the concept of SoC boiler plates. These provide an approach by which engineers can capture correctness criteria of an SoC at the system-level using "structured English" requirements. We then show how these can be automatically mapped to temporal logic formula. This chapter also introduces a type of finite state machine called synchronous Kripke structures (SKS). These are used for the formal representation of on-chip communication protocols of the IPs. This chapter may be viewed as the "requirements/specification" chapter.

Chapter 5 provides a SoC design approach, where the on-chip protocols are described as SKS and requirements are captured as boiler plates. It then develops an approach called oversampling to bridge the clock mismatches between IPs. Finally, it uses an approach based on "converter synthesis" to propose the design methodology. The concepts in this chapter are illustrated using a set-top box example. This chapter may be viewed as the "correct-by-construction design methodology" chapter.

Chapter 6 provides further details of the converter synthesis algorithm. Key concepts covered in this chapter include data-buffers and data-related properties, converter definition and control and the converter generation algorithm. We provide classifications of the inputs and outputs of a converter. Then the converter is formalized and its control actions are described using an example. The details of this algorithm with an appropriate illustration appear in Appendix A. This chapter may be viewed as "converter synthesis" chapter.

Chapter 7 summarizes related work. We discuss the system-level verification literature and SoC design literature. Key concepts covered in this chapter include literature related to requirements, modelling and analysis.

Chapters 1–3 and 7 are self-contained and may be read in any order. Chapters 4–6, on the other hand, will make more sense if read in sequence. Here are some possible/suggested reading orders:

- $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 6 \Rightarrow 7$. This is a very conventional flow and we have organized the chapters so that a reader can read the book top-down using this flow. This can also form the basis for any SoC design course, where these chapters may be covered over one semester, in this sequence.
- $1 \Rightarrow 7 \Rightarrow 3 \Rightarrow 2 \Rightarrow 4 \Rightarrow 5 \Rightarrow 6$. This order allows the reader to get good understanding of related work and the formal background (Chap. 3), before exploring the rest of the technical details. This is good for readers, who want a good grasp of both the background and the formal methods before progressing further.
- $1 \Rightarrow 2 \Rightarrow 7 \Rightarrow 4 \Rightarrow 5 \Rightarrow 6$. This order is suitable for more astute readers, who would like to skip the introductory material related to model/module checking and temporal logic. The contents of Chap. 7 can be suitably intertwined with the other chapters. For example, when reading Chap. 4, related work on modelling may be useful. Similarly, when reading Chaps. 5 and 6, related work on analysis presented in Chap. 7 may be useful.

The technical details of the converter generation algorithm are presented in Chap. 6 and Appendix A. An algorithm for convertibility verification and associated converter synthesis is presented in Appendix A. This forms the main basis of the proposed design methodology. However, Chap. 5 is fairly self-contained and provides the key idea of SoC design at a high level. Hence, it may be feasible that some readers may skip the technical details of Chap. 6 and Appendix A.

Auckland, New Zealand — Roopak Sinha
Auckland, New Zealand — Parthasarathi Roop
Ames, IA, USA — Samik Basu

Acknowledgements

We would like to acknowledge the following individuals, who have been either our collaborators, co-authors or people with whom we have discussed regarding the proposed research.

- Dr. Alain Girault, INRIA, Grenoble
- Dr. Gregor Goessler, INRIA, Grenoble
- Dr. S. Ramesh, General Motors R&D, USA
- Prof. Arcot Sowmya, UNSW, Australia
- Prof. Zoran Salcic, University of Auckland, New Zealand
- Dr. Karin Avnit, UNSW, Australia

Acronyms

- SoC: System-on-a-chip. Plural: SoCs (systems-on-chip)
- IP: Intellectual property (block)
- FSM: Finite state machine
- SKS: Synchronous Kripke structures
- CTL: Computation tree logic
- AMBA: Advanced microcontroller bus architecture

Contents

Chapter 1
System-on-a-Chip Design

A rapid surge in the consumer electronics revolution of the past decade has been fuelled by a design methodology that started in the mid 1990s called *core based design* [GZ97]. During this period, hardware design methods based on logic synthesis techniques [GDWL92] were exacerbating the design productivity gap i.e., the productivity of human designers lagged behind the projections made by Moore's law. It was predicted that for an average engineer, completing a design for a chip in the year 2001 would require around 500 years [KB02]. Hence, there was a lot of impetus in the mid 1990s to develop alternative design techniques that were inspired by the concept of design reuse that was well known for software. Reusable software components, also called "abstract interfaces", were introduced as early as 1977 [Par77]. Subsequently, Goldberg introduced the concept of object-oriented programming as an alternative paradigm to structured programming. Ever since, technologies such as CORBA, .NET, and more recently web-services encapsulate a piece of software as a reusable *component* and provide a range of methods for component reuse and composition [OSB11].

The hardware design community, around the early nineties, started exploring ways to bridge the design productivity gap. The concept of core based design was developed to reuse pre-designed and pre-verified cores such as microprocessors, interface controllers, network controllers, ports, buses and timers. The main objective was to create a design using existing cores rather than creating a full-custom design. Cores could be kept at different levels of abstractions such as:

1. A *hard core* is a pre-designed block with minimal scope for modification. It is fully designed with placement and routing completed (physical design steps following high level synthesis).
2. A *soft core* is a synthesizable HDL description.
3. A *firm core* is at an intermediate level between hard and soft cores. It can be in register transfer level (RTL) or netlist form.

As the appetite for consumer electronic devices such as mobile phones increased, time to market pressures also increased significantly. Hence, a design paradigm for reusable cores that meet minimum quality standards [GLK+99] was needed.

R. Sinha et al., *Correct-by-Construction Approaches for SoC Design*,
DOI 10.1007/978-1-4614-7864-5_1, © Springer Science+Business Media New York 2014