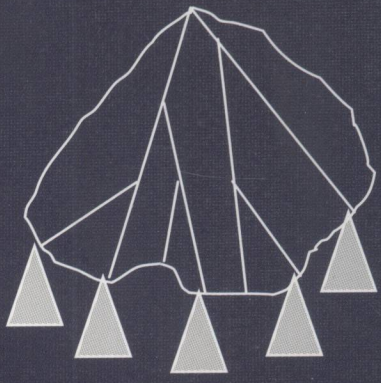
Marco Bernardo
Alessandro Cimatti (Eds.)

LNCS 3965 Tutorial

Formal Methods for Hardware Verification

6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006
Bertinoro, Italy, May 2006
Advanced Lectures



Springer

Marco Bernardo Alessandro Cimatti (Eds.)

Formal Methods for Hardware Verification

6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006
Bertinoro, Italy, May 22-27, 2006
Advanced Lectures

Springer

Volume Editors

Marco Bernardo
Università degli Studi di Urbino "Carlo Bo"
Istituto di Scienze e Tecnologie dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy
E-mail: bernardo@sti.uniurb.it

Alessandro Cimatti
Istituto Trentino di Cultura (ITC)
Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Loc. Panté, Povo, 38050 Trento, Italy
E-mail: cimatti@irst.itc.it

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, D.3, F.3, C.3, C.2.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-34304-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34304-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11757283 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3965

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

This volume presents a set of papers accompanying the lectures of the sixth edition of the International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM).

This series of schools addresses the use of formal methods in computer science as a prominent approach to the rigorous design of computer, communication and software systems. The main aim of the SFM series is to offer a good spectrum of current research in foundations as well as applications of formal methods, which can be of help for graduate students and young researchers who intend to approach the field.

SFM 2006 was devoted to formal techniques for hardware verification and covered several aspects of the hardware design process, including hardware design languages and simulation, property specification formalisms, automatic test pattern generation, symbolic trajectory evaluation, BDD-based and SAT-based model checking, decision procedures, refinement, theorem proving, and the verification of floating point units.

The opening paper by Bombieri, Fummi, and Pravadelli provides a general view on simulation-based modeling and verification strategies for developing embedded systems. In particular, the paper is focussed on describing state-of-the art co-simulation approaches and verification strategies based on fault simulation and assertion checking.

The paper by Drechsler and Fey reviews the basic concepts and algorithms for the postproduction test of integrated circuits. The then authors present an advanced SAT-based tool for automatic test pattern generation.

The paper by Claessen and Roorda concentrates on simulation-based model-checking techniques, which do not need to represent the states of the design, but only the values that flow through each signal. In particular, the authors introduce a high-performance simulation-based model-checking technique called symbolic trajectory evaluation.

The paper by Cabodi and Murciano overviews binary decision diagrams (BDD) and their application in formal hardware verification. The paper by Gupta, Ganai, and Wang illustrates instead a promising alternative to BDD-based symbolic model-checking methods that relies on Boolean satisfiability (SAT).

The paper by Cimatti and Sebastiani deals with decision procedures for verification problems that can be represented as satisfiability problems in some decidable fragments of first-order logic. The authors focus on integration techniques for combining technology for propositional satisfiability and solvers able to deal with the theory component.

The paper by Manolios addresses theorem-proving systems and shows how they can be employed to model and verify hardware using refinement. Theorem

proving is considered also in the closing paper by Harrison, where it is used for the verification of floating-point algorithms.

We believe that this book offers a comprehensive view of what has been done and what is going on worldwide in the field of formal methods for hardware verification. We wish to thank all the lecturers and all the participants for a lively and fruitful school. We also wish to thank the entire staff of the University Residential Center of Bertinoro (Italy) for the organizational and administrative support.

May 2006

Marco Bernardo and Alessandro Cimatti
SFM 2006 Directors

Lecture Notes in Computer Science

For information about Vols. 1–3894

please contact your bookseller or Springer

Vol. 3994: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part IV. XXIX, 1094 pages. 2006.

Vol. 3993: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part III. XXX, 1138 pages. 2006.

Vol. 3992: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part II. XXIX, 1121 pages. 2006.

Vol. 3991: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part I. CCXX, 1090 pages. 2006.

Vol. 3987: M. Hazas, J. Krumm, T. Strang (Eds.), Location- and Context-Awareness. X, 289 pages. 2006.

Vol. 3986: K. Stølen, W.H. Winsborough, F. Martinelli, F. Massacci (Eds.), Trust Management. XIV, 474 pages. 2006.

Vol. 3984: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part V. XXV, 1045 pages. 2006.

Vol. 3983: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part IV. XXVI, 1191 pages. 2006.

Vol. 3982: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part III. XXV, 1243 pages. 2006.

Vol. 3981: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part II. XXVI, 1255 pages. 2006.

Vol. 3980: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part I. LXXV, 1199 pages. 2006.

Vol. 3979: T.S. Huang, N. Sebe, M.S. Lew, V. Pavlović, T. Kölsch, A. Galata, B. Kisačanin (Eds.), Computer Vision in Human-Computer Interaction. XII, 121 pages. 2006.

Vol. 3978: B. Hnich, M. Carlsson, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 179 pages. 2006. (Sublibrary LNAI).

Vol. 3976: F. Boavida, T. Plagemann, B. Stiller, C. Westphal, E. Monteiro (Eds.), Networking 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems. XXVI, 1276 pages. 2006.

Vol. 3970: T. Braun, G. Carle, S. Fahmy, Y. Koucheryavy (Eds.), Wired/Wireless Internet Communications. XIV, 350 pages. 2006.

Vol. 3968: K.P. Fishkin, B. Schiele, P. Nixon, A. Quigley (Eds.), Pervasive Computing. XV, 402 pages. 2006.

Vol. 3967: D. Grigoriev, J. Harrison, E.A. Hirsch (Eds.), Computer Science – Theory and Applications. XVI, 684 pages. 2006.

Vol. 3966: Q. Wang, D. Pfahl, D.M. Raffo, P. Wernick (Eds.), Software Process Change. XIV, 356 pages. 2006.

Vol. 3965: M. Bernardo, A. Cimatti (Eds.), Formal Methods for Hardware Verification. VII, 243 pages. 2006.

Vol. 3964: M. Ü. Uyar, A.Y. Duale, M.A. Fecko (Eds.), Testing of Communicating Systems. XI, 373 pages. 2006.

Vol. 3962: W. IJsselsteijn, Y. de Kort, C. Midden, B. Eggen, E. van den Hoven (Eds.), Persuasive Technology. XII, 216 pages. 2006.

Vol. 3960: R. Vieira, P. Quaresma, M.d.G.V. Nunes, N.J. Mamede, C. Oliveira, M.C. Dias (Eds.), Computational Processing of the Portuguese Language. XII, 274 pages. 2006. (Sublibrary LNAI).

Vol. 3959: J.-Y. Cai, S. B. Cooper, A. Li (Eds.), Theory and Applications of Models of Computation. XV, 794 pages. 2006.

Vol. 3958: M. Yung, Y. Dodis, A. Kiayias, T. Malkin (Eds.), Public Key Cryptography - PKC 2006. XIV, 543 pages. 2006.

Vol. 3956: G. Barthe, B. Gregoire, M. Huisman, J.-L. Lanet (Eds.), Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. IX, 175 pages. 2006.

Vol. 3955: G. Antoniou, G. Potamias, C. Spyropoulos, D. Plexousakis (Eds.), Advances in Artificial Intelligence. XVII, 611 pages. 2006. (Sublibrary LNAI).

Vol. 3954: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part IV. XVII, 613 pages. 2006.

Vol. 3953: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part III. XVII, 649 pages. 2006.

Vol. 3952: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part II. XVII, 661 pages. 2006.

Vol. 3951: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part I. XXXV, 639 pages. 2006.

Vol. 3950: J.P. Müller, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VI. XVI, 249 pages. 2006.

Vol. 3947: Y.-C. Chung, J.E. Moreira (Eds.), Advances in Grid and Pervasive Computing. XXI, 667 pages. 2006.

Vol. 3946: T.R. Roth-Berghofer, S. Schulz, D.B. Leake (Eds.), Modeling and Retrieval of Context. XI, 149 pages. 2006. (Sublibrary LNAI).

Vol. 3945: M. Hagiya, P. Wadler (Eds.), Functional and Logic Programming. X, 295 pages. 2006.

Vol. 3944: J. Quiñonero-Candela, I. Dagan, B. Magnini, F. d'Alché-Buc (Eds.), Machine Learning Challenges. XIII, 462 pages. 2006. (Sublibrary LNAI).

Vol. 3943: N. Guelfi, A. Savidis (Eds.), Rapid Integration of Software Engineering Techniques. X, 289 pages. 2006.

Vol. 3942: Z. Pan, R. Aylett, H. Diener, X. Jin, S. Göbel, L. Li (Eds.), Technologies for E-Learning and Digital Entertainment. XXV, 1396 pages. 2006.

Vol. 3941: S.W. Gilroy, M.D. Harrison (Eds.), Interactive Systems. XI, 267 pages. 2006.

Vol. 3940: C. Saunders, M. Grobelnik, S. Gunn, J. Shawe-Taylor (Eds.), Subspace, Latent Structure and Feature Selection. X, 209 pages. 2006.

Vol. 3939: C. Priami, L. Cardelli, S. Emmott (Eds.), Transactions on Computational Systems Biology IV. VII, 141 pages. 2006. (Sublibrary LNBI).

Vol. 3936: M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, A. Yavlinsky (Eds.), Advances in Information Retrieval. XIX, 584 pages. 2006.

Vol. 3935: D. Won, S. Kim (Eds.), Information Security and Cryptology - ICISC 2005. XIV, 458 pages. 2006.

Vol. 3934: J.A. Clark, R.F. Paige, F.A. C. Polack, P.J. Brooke (Eds.), Security in Pervasive Computing. X, 243 pages. 2006.

Vol. 3933: F. Bonchi, J.-F. Boulicaut (Eds.), Knowledge Discovery in Inductive Databases. VIII, 251 pages. 2006.

Vol. 3931: B. Apolloni, M. Marinaro, G. Nicosia, R. Tagliaferri (Eds.), Neural Nets. XIII, 370 pages. 2006.

Vol. 3930: D.S. Yeung, Z.-Q. Liu, X.-Z. Wang, H. Yan (Eds.), Advances in Machine Learning and Cybernetics. XXI, 1110 pages. 2006. (Sublibrary LNAI).

Vol. 3929: W. MacCaull, M. Winter, I. Düntsch (Eds.), Relational Methods in Computer Science. VIII, 263 pages. 2006.

Vol. 3928: J. Domingo-Ferrer, J. Posegga, D. Schreckling (Eds.), Smart Card Research and Advanced Applications. XI, 359 pages. 2006.

Vol. 3927: J. Hespanha, A. Tiwari (Eds.), Hybrid Systems: Computation and Control. XII, 584 pages. 2006.

Vol. 3925: A. Valmari (Ed.), Model Checking Software. X, 307 pages. 2006.

Vol. 3924: P. Sestoft (Ed.), Programming Languages and Systems. XII, 343 pages. 2006.

Vol. 3923: A. Mycroft, A. Zeller (Eds.), Compiler Construction. XIII, 277 pages. 2006.

Vol. 3922: L. Baresi, R. Heckel (Eds.), Fundamental Approaches to Software Engineering. XIII, 427 pages. 2006.

Vol. 3921: L. Aceto, A. Ingólfsdóttir (Eds.), Foundations of Software Science and Computation Structures. XV, 447 pages. 2006.

Vol. 3920: H. Hermanns, J. Palsberg (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XIV, 506 pages. 2006.

Vol. 3918: W.K. Ng, M. Kitsuregawa, J. Li, K. Chang (Eds.), Advances in Knowledge Discovery and Data Mining. XXIV, 879 pages. 2006. (Sublibrary LNAI).

Vol. 3917: H. Chen, F.Y. Wang, C.C. Yang, D. Zeng, M. Chau, K. Chang (Eds.), Intelligence and Security Informatics. XII, 186 pages. 2006.

Vol. 3916: J. Li, Q. Yang, A.-H. Tan (Eds.), Data Mining for Biomedical Applications. VIII, 155 pages. 2006. (Sublibrary LNBI).

Vol. 3915: R. Nayak, M.J. Zaki (Eds.), Knowledge Discovery from XML Documents. VIII, 105 pages. 2006.

Vol. 3914: A. Garcia, R. Choren, C. Lucena, P. Giorgini, T. Holvoet, A. Romanovsky (Eds.), Software Engineering for Multi-Agent Systems IV. XIV, 255 pages. 2006.

Vol. 3911: R. Wyrzykowski, J. Dongarra, N. Meyer, J. Waśniewski (Eds.), Parallel Processing and Applied Mathematics. XXIII, 1126 pages. 2006.

Vol. 3910: S.A. Brueckner, G.D.M. Serugendo, D. Hales, F. Zambonelli (Eds.), Engineering Self-Organising Systems. XII, 245 pages. 2006. (Sublibrary LNAI).

Vol. 3909: A. Apostolico, C. Guerra, S. Istrail, P. Pevzner, M. Waterman (Eds.), Research in Computational Molecular Biology. XVII, 612 pages. 2006. (Sublibrary LNBI).

Vol. 3908: A. Bui, M. Bui, T. Böhme, H. Unger (Eds.), Innovative Internet Community Systems. VIII, 207 pages. 2006.

Vol. 3907: F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J.H. Moore, J. Romero, G.D. Smith, G. Squillero, H. Takagi (Eds.), Applications of Evolutionary Computing. XXIV, 813 pages. 2006.

Vol. 3906: J. Gottlieb, G.R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization. XI, 293 pages. 2006.

Vol. 3905: P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.), Genetic Programming. XI, 361 pages. 2006.

Vol. 3904: M. Baldoni, U. Endriss, A. Omicini, P. Torroni (Eds.), Declarative Agent Languages and Technologies III. XII, 245 pages. 2006. (Sublibrary LNAI).

Vol. 3903: K. Chen, R. Deng, X. Lai, J. Zhou (Eds.), Information Security Practice and Experience. XIV, 392 pages. 2006.

Vol. 3902: R. Kronland-Martinet, T. Voinier, S. Ystad (Eds.), Computer Music Modeling and Retrieval. XI, 275 pages. 2006.

Vol. 3901: P.M. Hill (Ed.), Logic Based Program Synthesis and Transformation. X, 179 pages. 2006.

Vol. 3900: F. Toni, P. Torroni (Eds.), Computational Logic in Multi-Agent Systems. XVII, 427 pages. 2006. (Sublibrary LNAI).

Vol. 3899: S. Frintrop, VOCUS: A Visual Attention System for Object Detection and Goal-Directed Search. XIV, 216 pages. 2006. (Sublibrary LNAI).

Vol. 3898: K. Tuyls, P.J. 't Hoen, K. Verbeeck, S. Sen (Eds.), Learning and Adaption in Multi-Agent Systems. X, 217 pages. 2006. (Sublibrary LNAI).

Vol. 3897: B. Preneel, S. Tavares (Eds.), Selected Areas in Cryptography. XI, 371 pages. 2006.

Vol. 3896: Y. Ioannidis, M.H. Scholl, J.W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, C. Boehm (Eds.), Advances in Database Technology - EDBT 2006. XIV, 1208 pages. 2006.

Vol. 3895: O. Goldreich, A.L. Rosenberg, A.L. Selman (Eds.), Theoretical Computer Science. XII, 399 pages., 2006.

¥359.00元

Table of Contents

Hardware Design and Simulation for Verification

Nicola Bombieri, Franco Fummi, and Graziano Pravadelli

Università di Verona, Strada le Grazie 15, 37134 Verona, Italy
{bombieri, fummi, pravadelli}@sci.univr.it

Abstract. The development of more and more complex embedded systems constitutes a very challenging task for EDA experts, due to their HW/SW-mixed nature joint to the high demand for quality and reliability. Recently, both industrial engineers and academic researchers have developed a very large number of techniques for dynamic verification in terms of co-simulation, which, in particular, address the different nature of hardware and software components of an embedded system. However, a widely accepted methodology does not exist. Thus, this paper is intended to provide a general view on simulation-based modeling and verification strategies for developing embedded systems. In particular, the paper is focussed on describing state-of-the art co-simulation approaches and verification strategies based on fault simulation and assertion checking.

1 Introduction

An embedded system can be defined as a computer that is a component in a large system and that relies on its own microprocessor [1, 2]. Thus, it can be viewed as a mix of cooperating hardware and software parts, which are able to provide a wider and more adaptable set of complex functionality with respect to ASIC and ASIP, without requiring the large amount of resources needed by general purpose systems. Examples of embedded systems include controllers for industrial processes, automotive appliances, medical devices, multimedia portable systems, data acquisition systems, etc. The main characteristic of embedded systems is the reactivity: they must continuously react to asynchronous input events. Furthermore, since such systems are particularly suited in real-time contexts, where tasks must be performed within a given deadline, predictability (determinism) can become a key issue. In such application domains, the adaptability is required too. In fact, when determinism is required, it must be preserved also when the system is operating in a highly non-deterministic environment.

Even if embedded systems historically operate with bounded resources, as memory and computational power, nowadays they are increasing their resources, leveraging on the improvements of silicon technology. In fact, technology scaling always offers new opportunities and new challenges to system designers. Moore's law predicts a doubling on systems complexity (expressed as the number of transistors per integrated circuits) every couple of years. Chips composed of tens of million of gates, and therefore of more than a hundred million transistors, are

M. Bernardo and A. Cimatti (Eds.): SFM 2006, LNCS 3965, pp. 1–29, 2006.

today feasible in commercial production lines with a 90 nm technology. Thus, a system that yesterday was developed as a set of several chips connected on a printed board, nowadays can be developed in a single chip, composed of several complex subsystems integrated on the same silicon die. Such a system is known as *System-on-a-Chip* (SoC) and it represents a strong paradigm for embedded systems [3]. Several advantages make this way to develop a system very attractive for system developers. When the system complexity increases, the number of pins also tends to increase, so it becomes simpler and cheaper to connect many subsystems on a single chip than several chips on a printed board. Furthermore, the on-chip wire capacitances are smaller than their on-board counterparts, and this implies higher performance and lower energy requirements.

On the other side, developing a single, very complex, SoC poses many challenges for the developers. First, when the technology scales, designers have to face new issues, like the short channel effects [4] and the crosstalking problem [5]. Even if such problems are usually faced by foundries, the system developers must be aware of them, because high-level design choices can have a strong impact on lower levels of abstraction.

Another issue is represented by the power quest [6]. The energy budget for embedded systems is usually strictly limited. Those systems are often battery-based, and the improvements on the battery capacitance cannot keep pace with the increase on the system complexity. Thus, to obtain a usable system (in terms of activity time), the designer must take into account the optimization of the energy consumption. Such an optimization is pressing also for the higher power density involved in modern integrated circuits. As the gate size reduces and the power consumption increases, the power density to dissipate strongly increases. Current high-performance systems, as the state of the art microprocessors, have already reached very high power densities, and, in the near future, power density is expected to increase even more. Those levels of power density imply a high quantity of heat to dissipate and this fact causes an increasing cost of the package to use. Moreover, the higher temperature of functioning has a direct impact on the reliability and on the life time of the systems. Therefore, the energy consumption minimization of a system is nowadays a key issue for the developer, which has to keep it under control at every stage of the design process.

Also the development time spent is a key factor that must be accurately considered when an embedded system is designed. The growing complexity of the development of such systems pushes towards component reuse [2]. Designers are bound to use use pre-designed subsystems, called *Intellectual Property* (IP) components, as far as possible. Such components can be of several kinds, ranging from cell libraries over blocks which perform a standard task (e.g., MPEG decoder, USB controller, etc.), to very complex components, like processor cores (ARM, MIPS and other families of microprocessors are commercially available as pre-designed IP cores). IP components are specified at several levels of abstraction, so that they can be used during all steps of the system design flow. Moreover, they are often customizable and configurable, thus designers can use them in their own systems, tuning them according to their needs.

The modern approach of design reuse introduces the concept of *platform* [7]. A platform is a fully defined interconnection structure and a collection of customizable IP blocks. A developer can start from an available platform and configure it choosing the parameters for the given IP blocks, adding new hardware devices, and removing useless IP blocks. A platform is then conceived as a highly reusable system that a designer can adapt to his own needs and purposes.

The need of taking into account all the previous challenges, and the intrinsic heterogeneous nature of embedded systems (HW and SW) makes the development of such systems a harder task compared with more traditional digital systems. In particular, the high demands for quality and reliability for embedded systems have led to complementary quality assurance efforts: hardware engineers have developed techniques for verification in terms of co-simulation, which, in particular, addresses the different nature of hardware and software components. Thus, these techniques are tailored for design and verification flows which comprises dedicated models for the hardware and the software parts.

In this context, the paper is intended to provide a review of design and verification techniques based on simulation for developing embedded systems. The paper is organized as follows. Section 2 describes a typical embedded system design flow. Section 3 is devoted to present techniques for simulation and co-simulation. Section 4 focuses on verification approaches which exploit testbenches and assertions, and the related issues. Section 5 reports some experimental results for the verification techniques presented in Section 4. Finally, concluding remarks are summarized in Section 6.

2 Design Modeling

The design of an embedded system is a very challenging task which involves the cooperation of different experts: system architects, SW developers, HW designers, verification engineers, etc. Each of them operates on different views of the system starting from a very abstract informal specification and refining the model through the abstraction layers reported. At every level of abstraction, a model of an embedded system can be viewed as a black box that processes the information received at its inputs to produce corresponding outputs. This I/O mapping defines the behavior of the system.

Figure 1 represents the classical design modeling flow where system level is refined by applying the new *transactional level modeling* (TLM) style [8]. A TLM-based design flow starts from an abstract system description and it evolves toward more detailed implementations till it gets to RTL. In particular, verification activity involves three main phases: first, the design implemented at the higher abstraction level is validated considering the system functionality; then, once the design is optimized following architecture exploration and performance analysis, it is validated taking into account the temporal behavior. Finally, whenever a step of the refinement flow implies a change in the system design, a further verification check is required in order to preserve the *golden model* functionality ascertained at the preceding step.

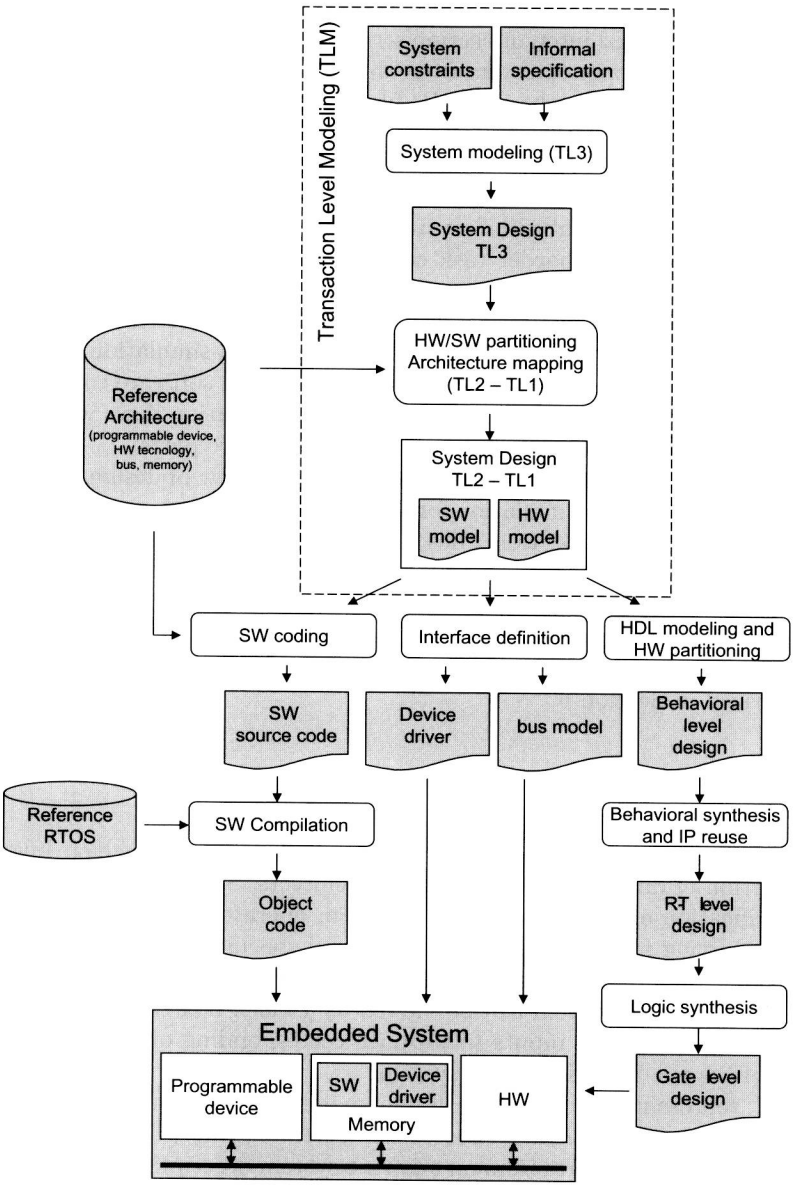


Fig. 1. Embedded system design flow

In spite of its name, *transaction-level* does not denote a *single* level of description; rather, it refers to a group of abstraction levels, each varying in the degree of functional or temporal details used and expressed.

A common agreement on terminology for TLM levels is still missing so far. Different interpretations (and terminology for the same concepts) have been

Level	Use	Features
TL3	Executable specifications and first level of functional partitioning of data and control. System proof of concepts.	➢ Implementation architecture-abstract. ➢ Untimed functionalities modeling. ➢ Event-driven simulation semantics. ➢ Point-to-point Initiator-Target connection. ➢ Abstract data types.
TL2	Hardware architectural performance and detailed behavior analysis. HW/SW partitioning and co-development. Cycle performance estimation.	➢ Mapping ideal architecture into resource-constrained world. ➢ Memory/Register map accurate. ➢ Event driven simulation with time estimation. ➢ Bit-width and transfer-size constrained data types to allow mapping to bus bursts or fragments of bursts. ➢ Split, pipelined with time delays.
TL1	Detailed analysis and low level SW development. Modeling CA interfaces for abstract simulation models of IP blocks such as embedded processors. CA performance simulation.	➢ Clock-accurate protocols mapped to the chosen HW interfaces and bus structure. ➢ Interface pin are hidden. ➢ Byte-accurate data Transactions have internal structure (protocols, data, clock). ➢ Transactions map directly to bus cycles. ➢ Parametizable to model different bus protocol and signal interfaces.

Fig. 2. TLM levels use and features

proposed by both industry and academia [8, 9, 10, 11]. However, factoring out common elements, key concepts are:

1. To implement a system at higher level means to implement the system in a more abstract way, that is to leave implementation details in order (mainly) to speed-up simulation for functional verification purposes.
2. To implement a system at lower level means to add implementation details to the system in order to simulate it in a more accurate way (for performance analysis purpose).

Hence, taking OCP-IP definition as reference, main use and features of every TLM level (e.g., TL3, TL2, and TL1) are summarized in Figure 2.

SystemC, as a broad-range level of abstraction modeling language, well addresses TLM. However, lack of established standards and methodologies means that each organization adopting TLM has to invent its own usage methodologies and API's. In addition to this redundant cost, these methodologies easily differ, making IP exchange and reuse more difficult. In this context, OSCI TLM library [12] based on SystemC represents a valuable set of templates and implementation rules aiming at establishing a reference for TLM API's implementation.

A typical TLM-based SoC design flow consists of the following steps.

- **System modeling (TL3).** Informal specification and system constraints are analyzed to provide a *system level model* of the design. At this level, there is no distinction between the HW components and the embedded SW. Indeed, the embedded system is considered as an interconnection of independent functional blocks which communicate by using blocks of words (messages) or shared memory. Implementation details like communication

protocols, delay analysis, computation algorithms, etc. are not taken into consideration. The most important issues of system modeling are represented by *efficiency* (i.e., the model must be quickly developable), *flexibility* (i.e., the model must be easily adaptable to explore different design implementations), and *functionality* (i.e., the behavior of the system must reflect the informal specification and it must satisfy the system constraints). Finite state machines (FSMs) [13], Labeled Transition Systems [14], Kripke structures [15], Petri nets [16, 17], process networks [18], etc. are valuable alternatives to formally model the functionality of embedded systems at such a level. The adoption of such semantic models makes the system level a good target for formal verification issues. However, to evaluate different architectural alternatives and to carry out performance analysis, semantic models are typically translated into simulatable descriptions. In this context, SystemC [19] is a very suitable language for system level modeling[1]: it joins the flexibility of C++ and the standard features of the traditional hardware description languages (HDLs), like VHDL [20], Verilog [21], etc.

- **HW/SW partitioning and architecture mapping (TL2, TL1).** The system level description is then mapped onto an architecture to obtain a transactional level model. This requires to decide which tasks will be implemented by SW and which ones by HW. The partitioning is actually a critical design choice, since there is no unique way to decide which task must be mapped into HW and which ones into SW. Moreover, some decisions about the configuration of the final system must be taken. In particular, the designers must select the following components:
  - the programmable device where the SW will run;
  - the memory model;
  - the HW/SW communication architecture and the bus typology;
  - the HW technology (ASIC, FPGA, etc.) where HW tasks will be mapped.

  HW/SW partitioning and architecture mapping provides a *transactional level model* where the communication is completely separated from computation. The focus is on the data rather than on the way transfer is executed. At this level, simulation is used intensively for evaluating different architectures. Thus, the transactional model aims at minimizing the amount of events and the information processed during simulation in order to reduce the verification time. SystemC represents an attractive alternative also at the transactional level, since it allows one to describe very accurately both SW and HW components. In this way, HW/SW partitioning is simplified, because functional tasks can be moved from SW to HW and viceversa without the need of code translations, which are required when two different languages are used to model SW and HW components.

[1] SystemC is a C++ class library which can be used to define methodologies to effectively model software algorithms, hardware architectures, and HW/SW interfaces. The class library includes a simulation engine (the *SystemC kernel*) that can be linked with the user descriptions. This allows us to obtain a single executable which exhibits the behavior of the modeled system.

- **SW coding.** After partitioning, SW and HW parts follow a different design flow. In particular, SW tasks are implemented by using a programming language (C/C++ represents an immediate solution when SystemC is adopted at system and transactional levels), and typical SW engineering techniques are used to optimize the resulting code. At this level, SW developers consider the HW part as a black box which communicates with SW via device drivers. Thus, SW coding must take into account constraints depending on the programmable device selected during the architecture mapping, and constraints depending on the communication interface.
- **SW compilation.** After the coding, the SW is compiled to object code. The compilation process generally depends on a Real Time Operating System (RTOS) which is selected to take care of load distribution, task scheduling, and communication with the HW interface. However, in some cases, RTOS may be absent, and the SW directly interacts with the device driver.
- **Interface definition.** Splitting the design tasks in HW components and pieces of SW introduces the need for an interface between the two parts that, often, is not specified in the initial requirements. This interface has to translate the timing information from the SW to the HW, and viceversa, because HW and SW rely on very different timing models. The HW model is typically event driven, while the SW model is cycle based, assuming it is executed by a programmable device. For this reason, the design of the interface between SW and HW parts is one of the most challenging task in the embedded system design flow. It requires to implement the device drivers for the programmable devices where the SW runs, and the communication bus to connect HW components, memory and programmable devices. The device drivers represent the interface between the RTOS and the HW components. Its purpose consists of hiding the HW to the SW layers by providing a set of functions to control the operation of the peripheral devices. Complementary, the purpose of the bus consists of defining the communication protocol taking into account many parameters like cost, bandwidth, reliability, etc..
- **HDL modeling and HW partitioning.** The HW model generated at the transactional level must be refined and optimized by executing different synthesis steps to obtain a gate-level description. Historically, the highest level of abstraction for HW components is represented by the *behavioral level*. The HW model is implemented by using an HDL focussing on the logic function of the HW components and ignoring implementation details. Moreover, the HW model is possibly partitioned into various interacting modules that better characterize the different HW units. Some books dealing with Electronic Design Automation (EDA) [22] make a more accurate classification and refer to this level as the *functional level*, while a behavioral level model is intended as a functional representation of the design coupled with a description of the associated timing relations. Any of these two abstraction levels keeps the complexity of digital system models quite low, allowing their rapid simulation.
- **Behavioral synthesis and IP reuse.** The functional/behavioral model of each HW component is further refined into a *Register Transfer Level (RTL)*

model by means of behavioral synthesis. The functionality of the design is decomposed and represented by a structural connection of combinational and sequential components (generally described as finite state machines with datapath (FSMD) [23]). At this level, IP reuse is performed too. Thus, already existing components are connected with new ones to provide the final RTL model. IP reuse sensibly decreases the time-to-market and it allows designers to concentrate the effort in implementing the very critical functionality of the system.

- **Logic Synthesis.** Finally, logic synthesis is used to translate the RTL model to a *gate-level* model, where the design is mapped into a structural view of primitive components (AND, OR, flip-flop, etc.) from which the physical mask can be easily generated to physically produce the circuit.

A verification/testing phase is mandatory after each step of the embedded system design flow to avoid the propagation of errors between the different abstraction levels. Indeed, synthesis is a dangerous process since it may introduce further bugs. This can be due to different causes: incorrect use of synthesis tools, incorrect code writing style that may prevent the synthesis tool to adequately infer the required logic, bugs of the synthesis tool, etc..

Thus, most of the publications focusing on the field of EDA start claiming the importance of verification [24] and testing [22] for shipping successful embedded systems. While the purpose of testing is to verify that the design was manufactured correctly, verification aims at ensuring that the design meets its functional intent before manufacturing. In particular, functional verification of embedded systems is the process of ensuring that the logical design of the system satisfies the architectural specification by detecting and removing every possible design error. As digital systems become more complex with each generation, verifying that the behavior is correct has become a very challenging task. Between 60% and 80% of the design group effort is now dedicated to verification [24]. The trend is particularly crucial for embedded systems, which are composed of a heterogeneous mix of hardware and software modules, and where the presence of design errors in the early phases of the design flow may lead to a complete failure of time-to-market fulfillment. In this context, both formal verification and simulation-based verification represent effective solutions to remove design errors.

There exist formal verification approaches to deal with the analysis and check of each of the grey products in Fig. 1. On the contrary, since one needs executable specifications to do simulation, simulation-based verification plays a predominant role in the later stages of the design process, i.e. once a design is already available, while formal verification must do most of the work at the border with the higher levels of abstraction. Typical abstraction-bridging verification tasks include checking a system level design vs. constraints or abstract specifications, checking the behavioral level design vs. partial models, checking component design vs. behavioral properties. Of course, the different approaches cover different aspects, thus they belong to different communities of scientists (like requirement engineering at the specification level, code analysis around a compilation task,