Guest Editorial: Special Issue on Models and Methodologies for Co-Design of Embedded Systems

This special issue is based on innovative ideas presented and discussed during the first ACM/IEEE Conference on Formal Methods and Models for Co-Design (MEMOCODE) held at Mont Saint Michel in France during the summer of 2003. Selected papers from the conference were invited for this special issue together with an open call for papers soliciting novel contributions on the topics of this conference. Rigorous reviews of 12 submissions led to the selection of four papers for this special issue. In this editorial statement, we outline the premise and the context of this special issue, and give a short introduction to the theme under consideration and briefly introduce the papers selected. We also thank the authors who submitted their contributions to this special issue, and all the reviewers without whose dedication and hard work toward ensuring the quality of the selections, editing this special issue would have been impossible.

Shortening *time to market* demands for complex electronic equipments, together with an ever increasing dependence on tightly coupled hardware–software systems for their operations, dynamically shifting boundaries between the hardware and software, heightening requirements for *performance* and *QoS guarantees*, *reliable operation*, and *accommodation of various consumer facing applications* have led researchers and systems design companies to invent innovative system design flows and techniques.

High-level modeling of hardware and embedded hardware–software systems, formal analysis techniques for hardware–software partitioning, formal techniques for system verification, and abstraction-based techniques for reducing complexity of such analyses are but a few examples of such innovations brought forth during the past decades. An often cited issue confronting system designers and companies today is the exponentially rising system complexities resulting from *Moore's law* and the resulting widening *productivity gap* in the face of various constraints.

The techniques and tools for circumventing these problems are not necessarily based on traditional system design flows which included hardware design followed by software development, but on a codevelopment flow, that can be immensely expedited by formal modeling techniques and formal design methodologies. Formal models of computation allow designers to work with concrete and analyzable mathematical concepts. These also help predict performance trade-offs, enable verifiability, and facilitate other functions in the design flow.

While investigating these flows, tools, methodologies, and models, we talked to various researchers from hardware industry, consumer electronics systems industry, as well as researchers innovating in the field of embedded software design and software engineering. It became quite apparent that embedded software engineering researchers speak a very different language, while often being confronted with very similar issues.

This situation is not surprising, because today's mission critical software systems are often concurrent, event driven, and certainly exploit the shift in boundaries of hardware and software. In certain fields of formal modeling and methodology such as in the theory of formal verification, researchers from hardware design as well as from software design fields have identified commonalities in the recent past, have attended the same conferences, and spoken the same language such as temporal logic, model checking, and automata theoretic methods and so on.

Our mission in building the new ACM/IEEE forum on formal methods and models for codesign has been to go beyond this limited interaction between the two fields which are both in need of innovations in design flow, methodology, and tools that would allow them to cope with the increasing design complexity and to effectively exploit the trends in hardware–software systems. Formal verification is but one issue confronting the two communities; there are many more interesting interactions that can help the two communities immensely.

The MEMOCODE conference is going to have its third meeting in July 2005. However, due to the rigor of the review process, this special issue dedicated to the first incarnation of this forum is coming out now. Nevertheless, we feel that this special issue will do its part in drawing attention to the commonalities between the two apparently distinct fields of research and, at the same time, will expose the readers to the four very interesting research work reported in the four papers we selected, that are summarized below. But before we summarize them, it might be timely to say a few words about the current activities in formal models and methodology areas.

Numerous programming languages, tools, and frameworks have been proposed in the past to design, simulate, and validate heterogeneous systems within an abstract and rigorously defined mathematical model. Recently, attention has shifted to modeling frameworks based on variants of general-purpose programming languages, in response to the growing industry demand for use of higher levels of abstraction in the system design process. Meanwhile, the installed base of existing IP (intellectual property) adds further requirements for the adaptation of existing equipments with new services within complex integrated architectures, calling for appropriate methodological approaches.

Whereas abstract mathematical frameworks are ways to unambiguously model the essence of hardware and software systems, help understand design, implement trustable correctness proofs, effectively predict performances, and other metrics; general-purpose languages facilitate programming, reuse, and gain from the popularity of languages such as C or C++. Still, important gaps need to be filled and bridges to be built between the theory of modeling and the practice of programming.

Programming languages shall benefit the rigorousness of models and the experience of programming practice. This calls for finding a convergence between both approaches. A focus on formal methods (programming and concurrency models, analysis and verification techniques) for hardware–software codesign hence is necessary, because languages with which system designers work are general-purpose ones, and because the only way provably correct systems can be constructed are by technology transfer of research in formal methods.

The first paper in this issue is by Cachera David and Morin-Allory Katell, and the paper is focused on formal verification techniques for safety properties using polyhedral models, which allows them to handle parameterized systems. The design flow, which forms the basis of the application of their technique, comprises a series of semiautomatic rewriting-based transformations from algorithmic system descriptions toward hardware or software implementations, and exploits the knowledge of the transformation steps in verifying the low-level control properties.

The second paper by Chouali Samir, Julliand Jacques, Masson Pierre Alain, and Bellegarde Françoise is a bit differently focused yet also addresses the formal verification problem. They propose an efficient model checking based verification of systems under fairness assumptions by partitioning the reachability graph and verifying by parts, which does not necessarily always work, as can be guessed easily. Therefore, they provide a characterization of properties for which such verification by partition is a sound and complete procedure. This is very useful for formal verification of hardware and software systems with large state space, if the property under verification satisfies their characterization.

The third paper, by William B. Gardner, is a departure from formal verification into the arena of programming and specification languages for effectively constructing executable system specification without departing too much from the known territory of general-purpose programming languages. He provides the implementation of a language based on communicating sequential processes (CSP), where CSP-like syntax is mixed with C++ to provide a way to model the concurrency and reactive aspects of a system (using CSP), and to encode behaviors in vanilla C++, thereby constructing executable concurrent reactive models of systems. Although the author does not extend this presentation toward a full-fledged system design language in the spirit of SystemC or SpecC, it can find a very interesting application in rapid system modeling and simulation.

The fourth and last paper is an interesting application of the model checking technique from formal verification to achieve more than just design verification. This paper, by Roberto Ziller and Klaus Schneider, attacks the problem of synthesizing a supervisor that can constrain the behavior of a system for controllability and coaccessibility, and their approach is a novel one via techniques borrowed from model checking, and shows how innovations in model checking can be transferred to the supervisor synthesis problem in control systems.

As outlined in the beginning of this editorial, our aim is to cross-pollinate methodologies and techniques from two distinct communities in order to solve system design problems, and certainly there are many distinct techniques and research issues that are not represented in this space constrained special issue. We hope that this special issue is just a sampler, and that the future bears much more innovations through the intended cross-coupling of two communities and through building bridges.

Sandeep K. Shukla and Jean-Pierre Talpin
(*Guest editors*)