Message from the Chairs

“HW/SW Codesign will be one of the most important design problems,” thus pined the #1 prediction from the Circuits and Network Design (CANDE) group in its 1991 forecast of the leading technical developments in five years. From its early promise to over a decade of technical sessions and even conferences dedicated to HW/SW Codesign, we have seen numerous articles addressing various aspects of the HW/SW systems modeling, programming, partitioning, co-verification along with numerous case studies in variously constrained embedded and/or realtime systems. More recently, the research has expanded to heterogeneous and hybrid (e.g., continuous, discrete time) systems. Taken together these contributions reflect hopes, promises and occasional disappointments of an emerging community of re-searchers with the overarching goal of handling the ever-increasing complexity (defined by the size and diversity of function) of microelectronic system-chips. Driven by the Moore’s Law, the process technology continues to march along its scripted path outlined in the International Technology Roadmap for Semiconductors (ITRS). However, our capability to design (and verify) these complex chips continues to lag behind creating the so-called “design productivity gap.” We have now arrived at a situation where design technology is beginning to be a limiter on progress of microelectronic system capabilities: increasingly a greater share of the performance is left “on the table” by the automated tool-driven design flows compared to their custom designed counterparts used in very high volume designs.

Recently, modeling frameworks based on general-purpose programming language variants (e.g., SystemC) have been viewed by many as a solution to fill the design productivity gap by enabling use of a higher abstraction level in the circuit (and software) synthesis process in embedded systems. However, as we begin to understand the complexity of the system design task, formal modeling and verification have taken on increased importance both in academia as well as industry practice. Even as we learn to model complete systems in emerging software and component frameworks from SystemC to Ptolemy II, the need to get a handle on the formal specification and correctness grows. The drive towards standardization in recent years is particularly notable both for specification (e.g., SystemC) but also for verification (e.g., constraint specification languages such as SUGAR).

Whereas abstract frameworks provide ways to unambiguously model the hardware/software systems, help understand the design, implement formal correctness proofs, predict performances and other metrics; general-purpose languages facilitate programming, reuse and gain from the popularity of C, C++ like languages. However, as we gather the momentum on use of “standards-based” specification and verification methods, we are beginning to see gaps between the modeling theory and programming practice. To address this gap, early in 2002 we initiated an e-mail discussion with a group of leading re-searchers in embedded system and formal methods area regarding barriers to adoption of high-level design, validation methods in embedded systems particularly, in the context of hardware/software codesign. The resulting discussion was quite an eye-opener: while discussing fairly abstract models and techniques, the participants converged on five different technology needs for the system design houses today. Expanded summary of these discussions appears on the conference website, we capture the essential ideas below:

1. Conformance Checking: While companies deliver high-level models of their chips even before RTL is written, there is no known formal way to compare these models against RTL implementations. Are some high-level models more amenable to automated equivalence checking than others? What is an appropriate notion of equivalence? Is there a methodology driven design style that can make the equivalence problem easier?

2. Hierarchical Verification: When using pre-designed IP blocks, if the IP blocks are known to be correct, how does one define and verify correctness of the entire system composed of the IP blocks? How can block-level assumptions regarding the environment be validated or composed? What happens if some internal details of an IP block are not known? What is the correct approach for re-use of verification information in IP based design?
3. Abstraction versus Optimization: By definition, a high-level model abstracts away many implementation details to achieve efficiency in design, synthesis and verification tasks. This efficiency in methodology sometimes runs counter to the quality of design and begs the following question – how much of a “methodology enforced” compromise in design quality (performance, size) is acceptable? Where is the threshold of pain in abstraction (design productivity, correctness), in optimization (design quality)?

4. Incremental Verification: Platforms, as in platform-based design, provide a basis for reuse of pre-verified hardware and software components while restricting the range of architectural choices. How do we provide incremental verification to ensure ‘rules’ of the platform have been observed when additional hardware is added? How can we guarantee that a function mapped onto that platform still operates as intended? How do we ensure that code coverage and functional test goals are achieved for the composed design?

5. Post-production Patchability: Designers may add mechanisms (using flash memory, FPGA, CPLD etc.) to allow after-fabrication modification of function, programmability or correction (as is commonly done for memory blocks). Formally, verification seeks to model an implementation and determine if a set of pre-specified properties hold for the model. What is the appropriate analog if the implementations are allowed to contain subsystems with large but bounded programmability? What properties provide for sound specification and design given this freedom? What is the right coverage metric? Can programmable interface protocols support this patchability? What are different types and levels of programmability that could be inexpensively added to a system to increase the flexibility?

From these discussions, we converged on building a forum, “Formal Methods and Models for Codesign,” or MEMOCODE, with the goal to bring the theory and practice together for embedded systems design, design modeling and design verification. In response to the call for papers, we received a total of 63 papers out of which 21 papers were selected for full presentations and 5 papers for short presentations after a thorough review process that included, on average, 4 to 5 reviews per paper. The technical program at MEMOCODE is actively put together through discussions within the entire technical program committee of the conference. Many of the conference sessions and panels map nicely to the grass roots discussions among the community we had earlier. To supplement the program, we have invited leading experts to share with us their view of the progress and challenges in the area.

We would like to take this moment to especially thank members of the technical program committee for their dedication, sincerity and fairness in the paper review process and their feedback in putting together this program. Our thanks to Elisabeth Lebret for her help with local arrangements. Many thanks to Patty Bladh for pulling together the digest and to Danielle Martin, Publications Coordinator at IEEE Computer Society Press, for its timely publication. We also thank IEEE Computer Society, IEEE Circuits and Systems Society, ACM Special Interests Group on Design Automation, CNRS, INRIA, IRISA, and University of Rennes for sponsorship of this event. But most importantly, we thank you, the attendees, for your participation in the technical program. We hope you like the program!

Mont Saint-Michel, France

Sandeep Shukla, Jean-Pierre Talpin
Technical Program Chairs

Rajesh K. Gupta
General Chair