Research Needs in Verification

Semiconductor Research Corporation
April 2005

This document describes needs and problems SRC member companies see in the verification of hardware and system designs. It is not intended to restrict research proposals or to discourage innovative thinking, but to point potential university investigators to the challenges members face and to encourage work in these key areas. Successful proposals in response to these needs will advance the state of the art, open new opportunities to SRC members, and provide new ways to assure error-free products. Proposals for new description languages and their compilers, for incremental improvements to existing techniques, or for work without an indication of applicability to real challenges are not encouraged.

SRC members have increased interests in design of systems and subsystems, as well as in materials, lithography, devices, and circuits. Thus, there is a desire by our members to advance verification technology as applied to systems comprised of both software and hardware, and in which the hardware itself is increasingly diverse, composed of multiple independently-developed cores, digital and analog components, processors, memories, a variety of interconnect approaches, systems-on-chip and system-in-package, etc. Successful proposals will address aspects of this challenge in the context of system verification, exploring research in formal, dynamic, and so-called semi-formal or hybrid methods, coverage analysis, and software and hardware verification techniques. Proposers will benefit from indicating present and/or anticipated relationships with SRC member companies.

1) Coverage
In order to have a strong and efficient balance between simulation-based validation and formal verification, which will allow the verification of increasingly more complex designs, we need to capture the "exhaustiveness" or coverage of formal verification. What are meaningful and useful coverage metrics, and what is the relation of design quality to these metrics? How do complexity measures of design relate to coverage? Can we do automatic coverage-directed stimulus generation? Can data mining be used for effective coverage data analysis? How can we generate random test streams yet provide a realistic system test environment that considers all external interfaces and bus devices?

Examples include experiments with different coverage metrics (e.g., various code-coverage metrics, functional coverage models at various levels, etc.) to explore the correlation between the different approaches and types of coverage models with the probability of revealing specific types of bugs in the design, learning-based algorithms that will allow a coverage-directed generation process with feedback, understanding how to use coverage data from a given simulation “cycle” to modify the input parameters for the stimulus generator so that the next simulation “cycle” will produce better coverage results.

2) Hybrid Methods
Since formal verification still has its limitations, we need to develop better hybrid flows for verification based both on simulation and formal technologies. How can formal and dynamic methods be combined to yield verification methodologies for practical verification? What combined methods with increased capacity will allow more fast "yes" and "no" answers to large problems to detect more bugs earlier? How can formal and mathematical methods be used to derive patterns for simulation and the sequencing of test cases? Can model checking and theorem proving be combined effectively?

Research in combined formal and dynamic methods, and in combinations of different verification techniques, must clearly demonstrate how combined methods can address problems not able to be solved, or solved only with unacceptable effort, by single methods alone.

3) Verification and Specification of Software
SRC remains focused on hardware verification, but system verification will of necessity involve software components, particularly as applied to embedded kernels, firmware, auto-generated code, etc. Many complex system designs as well as advanced new protocols are designed in software. The software language can be limited to a relevant subset for each application domain. Abstraction techniques can be adjusted with respect to the interesting set of properties. New specification formalisms may be developed to address well the specification and complexity issues.

Since most new designs are of embedded systems, software/firmware/hardware co-verification is an important topic. In these systems the design of the hardware and software is simultaneous and the correctness of each component depends heavily on the other. Verification should include aspects of functional correctness as well as aspect of power and other implementation concerns.

4) System-level verification
How can properties be successfully specified and verified at the system level? How can "verified" heterogeneous components be assembled into a hardware system so that properties can be proved about the whole system? Are there cost-effective and practical tools for verifying microcode and microarchirectures? What are the approaches for verifying heterogeneous components and their combinations, including mixed signal/analog? Verification of analog, RF, mixed signal, MEMS, and other components would be a great advance for formal methods, but is extremely difficult – analog test is still a challenge. What properties in addition to function (performance, area, power, I/O behavior, timing) can be verified, and how? How can bugs in design be reduced by making designs easier to verify and bugs easier to detect above the RTL level?

How can protocols, memory arrays, and deeply pipelined machines best be modeled and verified? How can constraint satisfaction techniques be used for generation of test vectors for verification of floating point units, and for test-program generation for high-end designs? How can we control synchronization between processor simulation models and pseudo-random generators? How can we generate random test streams

yet provide a realistic system test environment that considers all external interfaces and bus devices? Can high-level models be used for functionality verification as well as other aspects of processor modeling such as performance tuning and power estimation? Can a high-level computation model can be used to accurately model circuit switching activities, and to design clock-gating strategies?

Increasing design complexity has made it difficult to ensure that design constraints are respected during every refinement step in the design process. Research is needed into methods and tools that enable requirements and assumptions to be captured precisely and mapped down into lower-level checks. This process should start with high-level requirements and continue down through system modeling, RTL coding, and netlist implementation. These checks should make use of property checking technology where possible and otherwise into simulation monitors or reviewable items (e.g., for non-functional requirements).

5) Verification Reuse and Verification IP

Investigations are needed on including verification IP in design IP to enable reuse at the system level, using consistent assertions across all IP used in a system-on-chip, reusing design IP stimulus generation at the system level, and using design IP reference models to predict behavior at the system-level. Topics include reuse of verification, and verification standards for design IP. Incompatible components are very difficult to debug when they are integrated. Testing individually formally defined specifications can reduce the system level debugging time. Research topics include applying assertions for components at the SoC level, and re-using stimulus or transaction generators from a lower level again at a higher level, and using formal rules described as assertions to generate stimulus for the SOC.

6) Verification Core Technologies

Significant improvements to verification core technologies, particularly SAT-based methods, applicable to member verification needs mentioned in this document are needed. How can the SAT state space be limited by problem information? How can the “implied intent” of a design be captured and specifications be made easier? Can information in designs or circuits being verified be used to improve SAT solvers? Can parallel techniques be employed effectively? Can temporal expressions be used as drivers? What are the most promising verification techniques combining BDDs, SAT, ATPG-based methods, and new approaches? How can verification methods and algorithms be targeted to the properties to be verified? What are formal debugging aids that support fast error localization? Are there promising bounded model checking satisfiability solving algorithms?

SRC Task Force in Verification 2005

John Murphy	AMD	Robert Nguyen	Intel
Bob Kurshan	Cadence	Limor Fix	Intel
Ken Albin	Freescale	George Gorman	LSI Logic
Yaron Wolfsthal	IBM	Dale Edwards	SRC
Danny Geist	IBM	David Yeh	SRC
Jun Sawada	IBM	Bill Joyner	SRC