

An Integrated Methodology for SoC Design, Verification, and Application Development

Amir Hekmatpour
IBM Microelectronics
Research Triangle Park
North Carolina, USA
rima@us.ibm.com

Robert Devins
IBM Microelectronics
Essex Junction
Vermont, USA
rdevins@us.ibm.com

Dave Roberts
IBM Microelectronics
Research Triangle Park
North Carolina, USA
gdrobert@us.ibm.com

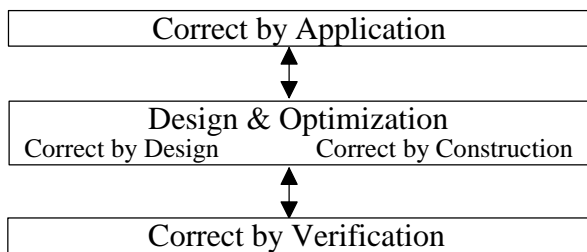
Michael Hale
IBM Microelectronics
Research Triangle Park
North Carolina, USA
mhale@us.ibm.com

Abstract

Today's highly complex System-on-a-Chip (SoC) designs demand a comprehensive and integrated design, verification, and application development environment. The methodology presented in this paper in conjunction with the IBM Blue Logic design methodology, tools, and services streamline this process to achieve first-time-right designs. It provides an improved co-simulation environment and a block-level assertion-based functional verification methodology in conjunction with Correct by Design and Correct by Construction design practices.

1. Introduction

To better manage the increasing application development cycle and cost, an early software/hardware co-simulation capability is provided to optimize applications and to qualify the system architecture. To address the increasing design complexity and the heterogeneous design characteristics of a SoC (i.e. CPU, IPs, BUS, Memory), a coverage-driven assertion-based method and flow is developed for block-level functional verification. To improve the overall design quality and shorten the design cycle, Correct by Design and Correct by Construction practices are incorporated into the development process.



The methodology provides a simulation environment that leverages open, industry standard modeling languages in conjunction with advanced simulation and acceleration technology. It enables pre-silicon software development, analysis, and verification. The central component of the open simulation environment is the In-Circuit Simulation (IC-Sim). IC-Sim is a transaction level interface between the simulator (cycle-based logic simulator/accelerator or PLI event simulator) and client software programs. IC-Sim adapts various CPU, IP and real-world model abstractions to the hardware assisted simulation accelerator or PLI

simulators, providing a board-level simulation model of a system (Virtual Board). Application developers are then able to execute large system software in a timely manner, very early in the design cycle. Transaction level models of new cores and IPs developed in C/C++ or SystemC are utilized in conjunction with the existing HDL (Verilog, VHDL) platforms.

In today's assertion-based verification tools and methods, the verification bottleneck of traditional simulation-based verification (test generation, simulation, coverage analysis) has been shifted but not eliminated. Defining assertions, ensuring their completeness and accuracy and maintaining a large number of assertions throughout the architectural and specification changes have proved to be the new verification bottleneck. Our block-level assertion-based methodology improves functional and interconnect verification where improper and unexpected design behavior can be caught close to the source of the bug (faulty IP, Core, Functional Block) in a timely fashion. Design integration is also improved through correct IP usage and transaction protocol checking and enforcement. The coverage-driven methodology ensures consistent verification quality across the SoC design flow. The assertion-based methodology enables static (Formal) as well as dynamic (Simulation-based) verification and provides interface, interconnect and protocol validation.

The co-simulation environment and the block-level verification methodology are integrated with a set of tools and techniques from IBM Research and leading EDA vendors to provide Correct by Design and Correct by Construction capabilities, for a comprehensive SoC design and verification environment.

2. System Level Verification - Correct by Application

The Correct by Application (CbA) philosophy states that a system is functional as long as the target application for that system is functional. CbA raises the verification abstraction level to the final functionality of the system as a whole. In this view, software ultimately determines the functionality of a system, so when software is "verified", hardware is also verified. IBM has been providing early capability to run application code snippets on an ongoing SoC design using Test Operating System [1].

Current advances in system level design (SLD) enable early hardware/software and architectural analysis using modeling environments such as SystemC [2]. Often, not all SoC models are available in SystemC, especially if a SoC is generated from a derivative chip. Initially, new design components are represented as SLD models, and need to be included in an existing platform simulation environment. The key to using CbA in a methodology is having a high performance hybrid simulation environment to effectively enable software execution on an early hardware model. Design and verification components are allowed from many sources; from HDLs to C-based behavioral code.

2.1 Open Simulation Environment

Open simulation is a software driven methodology that leverages open, industry standard modeling languages and behaviors in conjunction with cycle simulation and acceleration technology. Its purpose is an environment for pre-silicon software development, analysis, and high level chip functional verification.

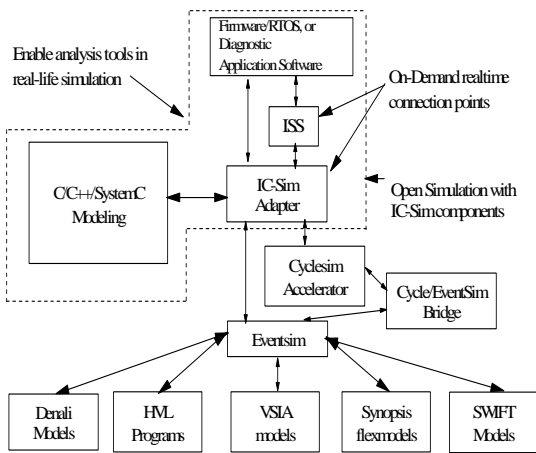


Figure 1. Open Simulation Platform

2.2 In Circuit Simulation

The central component of the Open Simulation methodology is the In-Circuit Simulation (IC-Sim) adapter. IC-Sim is a transaction based interface adapter between a cycle-based logic simulator/accelerator or PLI event simulator, and client software programs. Mixed (cycle and event) simulation is accomplished by using a bridge component that maps the design to both cycle simulation and event simulation. Mixed simulation opens the environment to all kinds of event-based verification tools and models. To accomplish this kind of hybrid modeling, IC-Sim has two distinct, simultaneous views: the CPU view and the Modeling view.

In the CPU view, the adapter delivers software originated transactions (loads, stores and cache-ops) to bus models in a design, and delivers responses and exceptions back to the software clients. Using this interface, software applications, either native (as a workstation program) or running in an Instruction Set Simulator (ISS), interact with pre-silicon hardware.

In the modeling view, the adapter interacts with either pin-accurate or transactional level models (TLM) C, C++ or SystemC models. This allows system designers to experiment with external real-world stimulus models, and internal TLM-based core representations.

Using Bus Functional Models (BFMs) in cycle acceleration, along with an ISS, a Real Time Operating System (RTOS) may be booted in the simulation environment. IC-Sim allows measurement of the properties of embedded software - useful for architectural, performance and power analysis based on real life scenarios.

2.3 Using Open Simulation in System Verification

As SOC and other ASIC designs become more complex, a top down, high level, Correct by Application (CbA) approach is needed to provide the ultimate verification scenario. There is significant work being done in the industry to use embedded software to help verify that the design constraints of the embedded system are correct [3]. Such an approach complements, but does not eliminate other forms of verification, such as design validation, interconnection and assertion testing.

In order to effectively develop application software and use it to verify system functionality, the design under test must be immersed in a *real world* simulation environment. Open Simulation enables the creation of a virtual representation of a physical board (Virtual-Board or Vboard), as shown in the following figure. By using various instantiations of IC-Sim adapters, and combinations of HDL and C-based models, a fully synchronous real world simulation is formed, and the Vboard closely approximates the physical board. Because Vboard is a set of simulation models, extreme visibility to all aspects of the design is possible. Debuggers are attached to both the hardware and software, and at points in between to monitor activity.

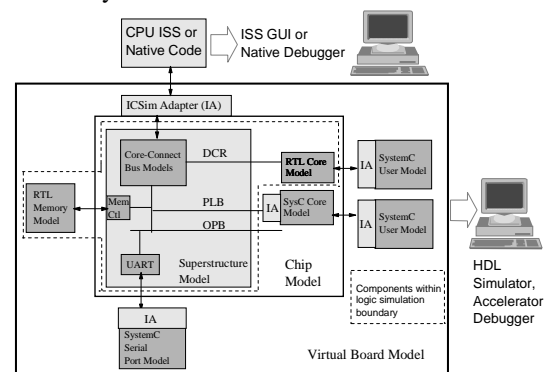


Figure 2. Vboard Co-Modeling System

The hybrid nature of the IC-Sim adapters allows a mix and match of various model abstractions during the design cycle, while maintaining the speed advantage of TLM and hardware acceleration.

In this Vboard example, the superstructure [4] along with external memory and one internal chip core exist as HDL

components. One new internal core is introduced and attached using an IC-Sim adapter instance. Three external real world testbench models are connected using the IC-Sim adapters. The CPU is represented as a BFM connected to a native application program or an ISS. Because of the abstract configuration of IC-Sim models, it is not necessary to rewrite any external models as the internal models are converted from C to HDL. Very high visibility and control is achieved as both hardware and software development occurs. When the Vboard is finally mapped into a physical board, designers have high confidence that the system architecture is correct, and the hardware and software design is largely functional. Because of the close mapping of physical to virtual and high visibility of system activity, the Vboard can be used to recreate and debug problems found in the lab.

3. Assertion-Based Functional Verification

Current state-of-the-art functional verification tools and methods encompass two categories: Formal verification and Assertion-based verification (ABV). Our system is directly related to ABV but the generated assertions can be utilized in Dynamic Verification (Simulation), Static verification (Formal), or Dynamic Formal Verification (Semi-Formal).

Current ABV tools and methods provide mechanisms for designers to define assertions in one or more of commonly used languages (PSL, Verilog, VHDL, C, e, ...) [5]. These assertions (checkers, monitors) are then folded into the verification test bench and exercised during verification [6][7]. The Verification bottleneck of traditional simulation-based verification (test generation, simulation, coverage analysis) has been shifted but not eliminated. Defining assertions, ensuring their completeness and accuracy and maintaining a large number of assertions throughout the architectural and specification changes has proved to be the new verification bottleneck.

Continuing pressure on the design cycle reduction and more frequent design architecture and specification changes require an automated assertion generation and maintenance mechanism. Similar to the Design Synthesis, as the design specification or implementation changes, designers should be able to automatically assess the existing assertions, generate any missing ones (for new design spec or implementation) and update those impacted by design changes and/or specific implementation. This feature is not available in any of the existing ABV tools and methodologies.

3.1 Related Work

Several EDA companies such as Mentor Graphics (0-in), @HDL, Cadence (Verplex), Real Intent, and Atrenta have addressed this problem by providing static design analysis tools and checker libraries. However, they do not provide a mechanism for the user to customize the checker library during verification process nor do they provide any automatic general purpose assertion generation capability (i.e. assertions that can be used outside of their own

proprietary tools and libraries) [8]. These tools and methods either generate low level assertions within their own black-box environment with no visibility to the user (e.g. 0-In CheckList, @HDL Verifier, Verplex BlackTie, Atrenta SPYGlass,...) or require the user to manually define and maintain verification assertions (e.g. 0-In Search, 0-In Confirm, Verplex Conformal, IBM Arctic) [9].

In general, ABV tools and techniques provides assertion libraries and modeling languages, so designers can select (use as is) or define assertions and then verify the assertions either via simulation or formal analysis [8] (Figure 3a).

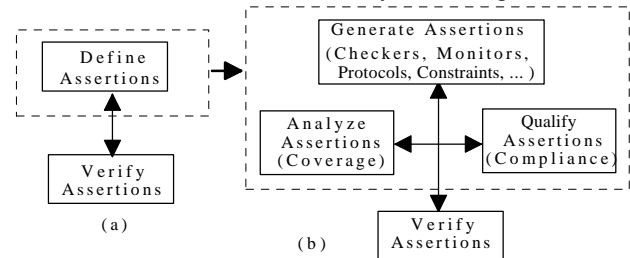


Figure 3: ABV and Analysis for SoC Validation

3.2 Block-level Assertion Generation

Our approach provides a model-based automatic generation of user-directed and design-biased assertions for ABV, as well as Formal Verification. The assertions can be generated in one of the supported languages either embedded in the design HDL or provided as a separate design module. The model-based architecture applies selected assertion schemas to the corresponding HDL blocks. It also enables the designer to qualify the generated assertions for methodology or standards compliance, as well as analyze their quality and completeness as shown in Figure 3b.

User-defined biases and preferences, as well as design constraints and verification environment requirements are taken into consideration in generating, optimizing and maintaining a design assertion schema library. The automated process provides mechanisms for design and verification engineers to describe their preferences, as well as to include their own hand coded or third-party assertions. Existing assertion schemas can be updated for a fully directed and customized assertion generation.

The generation process starts with an assertion schema either directly selected from the library, modified by the user, or fully defined by the user. Analysis of the assertion schema identifies tasks to be performed, impacted design constructs, and constraints to be solved. A process model (tasks and actions) is generated from the assertion schema rather than hard-coded into system, as is the case with existing tools and methods. Our approach allows what-if analysis and provides a generation and inferencing mechanism for defining custom assertion coding techniques and applies it to the design under test. Verification engineers can make changes to assertion schemas and regenerate assertions for visual inspection or resimulation/analysis to quantify the impact of changes.

Figure 4 gives an overview of the assertion generation process. Design specification (HDL) is parsed and the design attributes corresponding to each class of assertions are identified. Signal names and block attributes are then extracted and embedded into the corresponding assertion model instance. The completed assertion model is then instantiated and the corresponding assertion object codes are either inserted into the design specification or compiled to a stand-alone verification module.

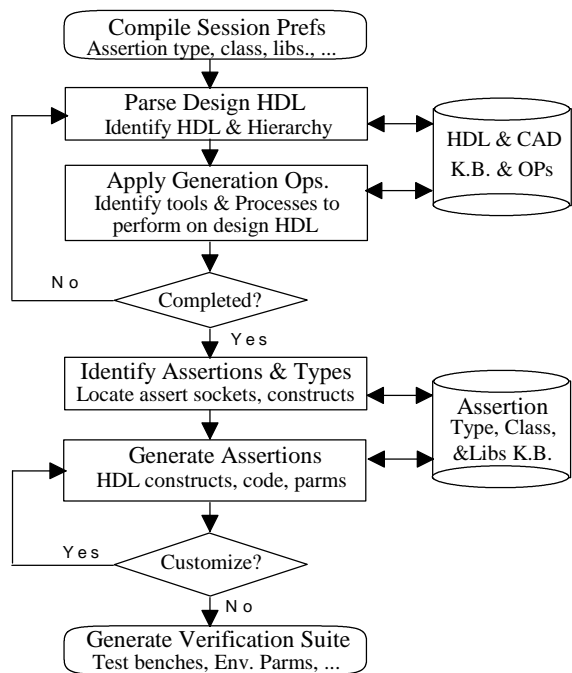


Figure 4. Assertion Generation Process

3.3 Model-Based Assertion Generation

An overview of the model-based system architecture is shown in Figure 5. The generation and optimization components of the architecture were described in detail in the previous sections. In this section we provide an overview of the system architecture.

A model-based system operates on a model of the structure and behavior of a device or the function that a system is designed to simulate. Observed behavior (what the device is actually doing) is compared with predicted behavior (what the device should do). The differences between observed behavior and predicted behavior are identified as discrepancies, indicating potential defects. The inference component of such a model-based system (e.g. its model-based reasoning engine) is then initiated to diagnose the nature and location of any defects.

A model-based system is usually comprised of several independent components (i.e. models, methods, inference). The final result is based on and influenced by all relevant models and methods. Changes in the inference will impact the quality of the results for the same set of models and changes in any of the models will impact the result of such a

system even if the inference and generation methods remain the same. Ensuring the integrity and predictable quality of solutions is an important ongoing activity in development and maintenance of such systems. Providing feedback and guidance to users of such a system on proper utilization and adjustments required to existing methods and procedures (i.e. what adjustments users have to make to the models or methods they have already developed) is another important ongoing activity in such an environment

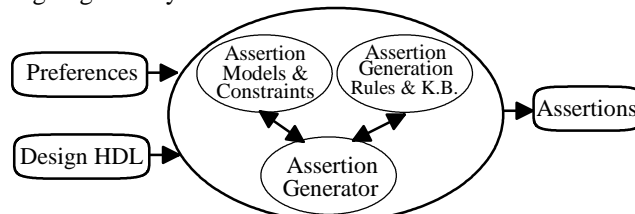


Figure 5. Model-Based Architecture

4. Correct by Design

We view “Correct by Design” (CbD) to be a mindset of practical, usable disciplines which contribute to IBM’s history of “First Time Right” in SoC engagements. The basic principle behind these disciplines is to focus on putting effort up front to prevent design problems, rather than applying effort later to detect them.

The main CbD techniques applied to SoC development at IBM can be reduced to four simple concepts: setup & infrastructure, abstraction, single source, and IP reuse.

4.1 Setup and Infrastructure

Before any SoC development work begins, proper facilities are put in place to manage the project’s data. Internal development standards are followed for the setup of the project work area; revision management; problem tracking; and the simulation analysis/regression system. These are then coupled with project-wide conventions (such as naming and connectivity) to form a base of clear and effective communication between architects, designers, and verifiers. The result is being able to effectively “bring together the right mix of expertise and design resources to the project” as described in the Design Management section of [10].

4.2 Abstraction

Once the product definition is complete for an SoC, a System Level Design (SLD) process, such as described in Synopsys’ reference manual [11], is followed. This is done in SystemC, SystemVerilog, or other modeling language. The elements comprising the system are abstracted to a high level. Key system attributes of concern, such as bus usage, resource availability, and data transforms are retained while discarding details below the transaction level.

The effects of architectural and software choices on characteristics such as bandwidth, power, and resource availability can then be analyzed. This is done under the context of the design’s true operating environment while running real system software. Once all design constraints

are met, the architecture choice is solidified and the SLD becomes an executable specification of intended function.

The SLD is then used through the rest of the design cycle. When the SLD indicates that new logic functionality needs to be developed, design assist tools are used. If it is a hardware algorithm, logic is synthesized from the behavioral descriptions. This reduces intermediate steps and brings the result closer to the intent. This is coupled with HW IDEs (such as SummitHDL) which help improve visibility and control complexity. The SLD operating environment becomes the core of the test bench. The models are used as golden references as well as to accelerate simulation. ICSim/Vboard allows dynamic, “at will” substitutions of core representations (RTL/gate/transaction) as the design progresses while always furnishing the ability to conduct software development.

There is little consensus among IP vendors today concerning what characteristics deserve to be modeled in SLD descriptions and analysis tools. IBM Research is currently leading the way in developing a set of standards as they reflect upon the unique requirements of SoC development.

4.3 Single Source

When possible, a single source of information is used to automatically generate other forms of that information. This allows equivalent descriptions to remain in synchronization and reduces manual error injection. IBM Haifa pioneered generating design and verification templates directly from the textual specification.

After key tables and diagrams in the specification are complete, programs parse the spec and generate a project starting point. This consists of design and verification templates which match the information in the specification exactly. These templates contain key information such as register names/offsets/definitions, interrupt vectors, reset information, I/O definitions, and top level interconnects. This ensures the C, SystemC, documentation, Verilog, and VHDL all agree on these definitions. Whenever updates are required, the generated sections of the other files can be respun quickly. This automated generation also allows the design and simulation process to begin sooner as it decreases the time required for infrastructure development. Externally, a more limited, yet similar process has been presented to the Synopsys User's Group (SNUG) [12].

4.4 IP Use

Lastly, the final component of CbD isn't even created by the SoC design team, it is developed by those who design and develop the IBM Blue Logic IP collection. Considerable thought and effort has been spent in the creation of key core development and release manuals who define the standards to which all cores in the library must adhere. Exacting standards are defined on verification, interconnect, naming and signal convention, reset, control registers, and so on to ensure that all cores are truly plug and play.

Peer review and automated processes occur throughout the core development cycle to ensure compliance to these rules and to further core quality.

5. Correct By Construction

Correct by Construction (CbC) is “becoming design mantra.” as quoted in a commentary in the EETimes by Max Lloyd [13]. The article describes a physical synthesis approach for Structured ASICs. But the principle, “reduce or, better yet, eliminate design iterations” can be applied at many levels in the design practice. For SoCs, CbC is about encapsulating IP integration into a definable and repeatable process. In SoC design, it is assumed all IPs have been fully verified and only the integration, glue logic and/or additional functions need to be verified. IP Integration has not classically been seen as a bottleneck, because one or two well trained engineers can do the integration in a short amount of time. However, integration errors are often a root cause of many verification issues, which have taken considerable time and effort to identify. This integration has been eased somewhat over the past few years by the introduction of On Chip Bus Architectures such as CoreConnect [14] and AMBA. However, intimate knowledge of the bus architecture is required for proper IP integration.

Correct by Construction is good design practice for SoC because it seeks to automate the integration process and allows a SoC designer to concentrate on exploring different configurations quickly. CbC allows integration by engineers that do not need extensive knowledge of the underlying bus topology. As discussed in the Correct by Design practices SoC design is more about exploration than about integration. Enabling Correct by Construction methods lifts the burden for a SoC designer. One graphical interface enables the designer to create a design, explore different configurations of bus topology, bus type, memory maps, and IP content. The actual views necessary are generated from this one interface to simulate and evaluate different configurations.

5.1 Contrasting Methods

A contrasting method to a graphical interface is to use a Platform Based Design (PBD) approach. IBM's Customizable Control Processor (CCP) [4] could be considered an extension to the PBD approach. In CCP a great deal of implementation detail is designed and verified, thus enabling the designer to concentrate on the derivative portion of a new CCP-based design. A PBD approach is less flexible, but has more decisions made up front. This could be considered a Correct by Construction method because a large portion of the design is pre-stitched and verified. Besides being less flexible there is still the rest of the design that still needs to be created. A hardened PBD approach such as CCP could benefit from the addition of graphical interface if the CCP is effectively considered IP, just like the individual functions. This would not change the hardened portion of the design but burden of defining the

derivative portion of the design would be eased by allowing different configurations to be explored more quickly.

5.2 CbC Tools

So far the discussion has been about why, but there needs to be significant investments in the “how” in order for Correct by Construction to be successful. First, a tool must be used to create a representation of the design. There are currently a limited number of tools available that offer this functionality today. IBM has an internally developed tool called “CORAL” [15] and Mentor Graphics is deploying a tool called Platform Express [16]. Both tools are similar in that they require each IP to be described in a format foreign to the design community. CORAL uses the term Virtual Component and utilizes a proprietary format while Platform Express produces a schema that defines an XML construct. Both formats require some specific information about the IP and the On chip bus to be described to the tool in order to correctly construct a desired design in a specific output format.

5.3 Application to SoC design Process

All of the information necessary to construct the design is also beneficial to Correct by Verification and Correct by Application methodologies, but only because it enables Correct by Design practices. A specific example of assisting the CbA methodology is that when the design is created, the memory map is known. One of the desired outputs of CbC might be to create a memory map definitions file in an ANSI C format to be used by the drivers of the application software. An example of enhancing the CbV methodology is the same design will have connections with known bus connections, certain bus assertions could be defined for each IP based on its intent in the design. Although enabling Correct by Construction may require a considerable amount of effort and resources, it improves design productivity and reuse. It can be viewed as a method of encapsulating a significant deal of the IP information into a format that can be leveraged by various tools. The information that is needed by Correct by Construction is effectively a parsable specification that enables tools that can integrate IP at an SoC level, create documentation, and even aid in creating simulation environments.

6. Summary and Conclusion

We introduced an integrated methodology for SoC design and verification. Its open simulation environment supports industry standard modeling languages and provides a virtual board system model enabling pre-silicon software development, analysis, and verification. The block-level automatic assertion generation improves the functional verification and provides efficient interconnect analysis and protocol compliance checking. The correct by design and correct by construction tools and techniques complement the above co-simulation and verification methods by enabling and enforcing coherent and efficient design practices.

References

- [1] R. Devins, "SOC Verification Software - Test Operating System," Electronic Design Processes Workshop, 2001.
- [2] Ramaswamy Ramaswamy, Russell Tessier, "The Integration of SystemC and Hardware-assisted Verification," www.ecs.umass.edu/ece/tessier/systemc-fpl02.pdf.
- [3] Alberto Sangiovanni-Vincentelli, Grant Martin, "A vision for embedded software," Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems, 2001.
- [4] C. Ross Ogilvie, et al. "Simplifying SoC design with the Customizable Control Processor," Proc. ICCD, Oct. 2003.
- [5] T. Fitzpatrick, H. Foster, E. Marschner, P. Narain, "Introduction to Accellera's assertion efforts," EEdesign, June 2, 2002.
- [6] Synopsys, Inc., "Assertion-Based Verification," <http://www.synopsys.com>, May 2002.
- [7] H. Foster, "Improving Verification through Property Specification," Design and Reuse Industry Articles.
- [8] J. Emmitt, "Verifying complex SoCs with OVL," Electronic Engineering Design, January 28, 2002.
- [9] 0-In Design Automation, Inc., "Black & White Assertion-Based Verification Flow," The Verification Monitor, www.0-in.com, May 2002.
- [10] P. Loughhead, "The Multiple Dimensions of Design Capture," Nov. 2002, www.altium.com/learningguides/dxp/wp_multidimensionaldesign.pdf.
- [11] Synopsys, Inc. "CoCentric SystemC Compiler Behavioral User and Modeling Guide," 2000.
- [12] W. Snyder "505 Registers or Bust," 2001 Boston SNUG, www.veripool.com/vregs.html.
- [13] Max Lloyd, "Correct-by-construction becoming design mantra," <http://www.eedesign.com/columns/eda/OEG20031117S0056>.
- [14] Imed Moussa and Thierry Roudier, "IP Modeling and Reuse for SoC Design Using Standard Bus," <http://www.us.design-reuse.com/articles/article4837.html>.
- [15] R. A. Bergamaschi, et. al, "Automating the Design of Systems-on-Chip Using Cores," *IEEE Design & Test Magazine*, September, 2001.
- [16] Mentor Graphics, "Platform Express Technical Background," http://www.mentor.com/soc/techpapers/?navtabs=soc_1:1,so_c_2verification:2&nav=platform_express#plat.

Acknowledgments

Authors are grateful to Randal Pratt, Mark Kautzman, Ray Schuppe, Azadeh Salehi, Frank Neuffer, James Coulter, Bob Lynch, and Dwight Sullivan for their technical critique and suggestions on development of the “IBM SoC Methodology” and their review and comments on early drafts of this paper and the associated “IBM SoC Methodology White Paper”.

Copyrights

PowerPC is a registered trademark of the International Business Machines Corporation.