Formal Verification of a PowerPC^{TM} Microprocessor

David P. Appenzeller
IBM Microelectronic Burlington
Essex Junction, VT, U.S.A.

Andreas Kuehlmann
IBM Thomas J. Watson Research Center
Yorktown Heights, NY, U.S.A.

Abstract

This paper presents the use of formal methods in the design of a PowerPC microprocessor. The chosen methodology employs two independently developed design views, a register-transfer level specification for efficient system simulation and a transistor-level implementation geared toward maximal processor performance. A BDD-based verification tool is used to functionally compare the two views which essentially validates the transistor-level implementation with respect to any functional simulation/verification performed at the register-transfer level. We show that a tight integration of the verification approach into the overall design methodology allows the formal verification of complex microprocessor implementations without compromising the design process or performance of the resulting system.

1 Introduction

Formal methods have successfully been used on various levels of abstraction ranging from high-level property checking to Boolean comparison of two combinatorial circuits. Basically, the application area of formal verification for practical designs is characterized by three main factors: (1) The size of the design under examination, (2) the abstraction level of the properties to be verified, and (3) the degree of automation provided by the verifier. Due to the exponential complexity of the general verification problem current approaches cannot explore all three factors at the same time. Therefore, various methods have been developed that are tuned to different areas in this complexity/abstraction/automation domain.

As an example, symbolic model checking [1] is capable of verifying high-level design properties expressed as formulas of temporal logic. For testing complex sequential system characteristics, these formulas are very powerful and can identify serious design errors, such as dead lock situations of communicating protocol machines. However, unless specific properties of the circuit structure can be exploited, the application of model checking is limited to designs with a few hundred state registers. Even though model checking is not practical for full scale microprocessor verification, it can successfully complement an existing verification methodology for specific subsystems, such as bus controllers.

[1] This paper is published as Report RC (19971), IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, March 1995

As compared to high-level model checking, less abstraction of the properties to be proven can significantly increase the size of the designs which can be formally verified. For example, an exhaustive functional comparison of different design views implicitly validates properties confirmed on one (preferably abstract) view for all other (typically more detailed) representations. As an example, in [2] the application of SFG-tracing is presented to formally compare circuits synthesized by the Cathedral system with the original input specification.

In this paper, we discuss the application of formal verification in the design process of a PowerPC microprocessor. The methodology employed two independently developed design views. The first view, a register-transfer level (RTL) specification, was highly tuned for maximum simulation performance and exposed to extensive simulation for confirming the compliance with the PowerPC architecture. The second view was a system implementation and was primarily custom designed on the transistor-level to achieve optimal system performance. The verification program, Verity [3], was applied to exhaustively prove the functional equivalence of these two design representations.

An industrial design environment necessitates the following challenges for formal verification, which need to be addressed in the chosen design methodology:

- The verification approach must be suited for a distributed design environment in terms of time and space. It is not acceptable to always wait for a complete design point before verification can be applied. Verifying partial circuits must be a continual part of the development process which might last for several years. Further, typical development teams for microprocessors consist of 50 to 100 designers. The verification method must consider a corresponding division of the development work, different design styles and skill levels, and varying progress in completing the individual pieces.

- The verification tool must be both reliable and predictable. Due to the algorithmic complexity, our approach for verifying large systems is based on an equivalent partitioning of the two design views being compared. It is crucial to confirm as early as possible that a given partitioning is feasible for the verifier. Late changes of either design model might have a significant impact on the overall design schedule.

- The verification tool must handle various design and circuit styles. In the given case, manual circuit design in conjunction with logic synthesis and PLA generators were used. The resulting system implementation consists of static and dynamic CMOS circuits mixed with gate-level logic.

- To maximize its application during manual circuit design, the verifier needs to work interactively with a fast response for the majority of the designs. Further, a strong debugging assistance for efficient error location and correction is as important as uncovering functional miscompares.

The rest of the paper is organized as follows: Section 2 summarizes the basic concepts of the verification tool, Verity. Section 3 describes the overall design methodology with respect to the verification approach. Section 4 elaborates on the circuit design style using formal verification. Section 5 presents various statistics about the tool usage over the project duration.

2 Verity

In this section we discuss those concepts of the verification tool Verity that are significant for the presented PowerPC verification methodology. A detailed description of the applied algorithms and methods can be found in [3].

Verity was designed for functional verification of large transistor and gate-level circuits. It uses Reduced Ordered Binary Decision Diagrams (ROBDD) [4] for a canonical representation of logical functions and employes various heuristic ordering algorithms including dynamic variable ordering [5]. The following techniques have proven to be a prerequisite for the practical verification of large systems:

Programmable mixed-mode extractor:
The extraction of the logical system function is based on a mixed switch/gate-level circuit representation. This general scheme allows the verification of circuits at various design stages and abstraction levels. The actual extraction rules are programmable and can be adapted to a wide variety of circuit styles such as static or multi-phase dynamic circuit techniques. Further, a set of programmable consistency checks validates the extraction model and is used to uncover unwanted circuit situations such as collisions at nets that are simultaneously driven by both logical values.

Combinatorial verification model:
Verity does not address the general sequential verification problem. It is based on a verification model in which corresponding registers of both design views are to be identified. This restriction can impact the composition of the RTL specification. However, the limitation to combinatorial equivalence enables the verification of more complex systems. Further, it significantly improves the ability to predict whether a given circuit partition can be handled by the tool. In many practical cases registers/latches used in the circuit implementation have modified interfaces with respect to the original specification. Verity uses a general method for matching such differences by supporting user-defined glue logic.

Hierarchical verification methodology:
The verification of large systems is based on an identical partitioning of the two design views being compared. Typically, the top part of the given design hierarchy is taken as the verification partitioning. Its granularity must guarantee that each piece can successfully pass Verity. Depending on the functionality, circuit pieces containing up to 25,000 transistors can be handled. Design parts that have been verified are excluded on the next higher verification level by a *black-boxing* scheme. Black-box inputs are treated as verification outputs and the incoming functions of the two views are compared. Similarly, black-box outputs are converted into verification inputs and driven by independent variables.

Logical boundary assertions:
A partitioned verification approach requires a general mechanism to specify and validate logical boundary assertions. Such assertions describe the set of valid signal patterns at circuit boundaries that occur in normal chip operation. In conjunction with the hierarchical verification approach, Verity uses boundary assertions to restrict the verification space. For each circuit to be verified, the user specifies input constraints and output tests that are used accordingly while applying Verity. When the circuit is black-boxed at the next higher verification level, the output tests are converted into constraints, effectively restricting the set of possible patterns of the output variables. Similarly, the input constraints are converted into tests that are validated for the arriving signal values.

Error diagnosis:
In case of logical miscompares, failing boundary assertions or consistency checks, Verity produces counter example patterns that exercise the unwanted situation. Multiple verification problems are grouped together if they can share the same counter example. This reduces the debugging information and helps the designer to focus on common design problems. Further, Verity applies an efficient error diagnosis algorithm which classifies circuit nets according to their probability of causing the error(s) [6].

3 Design Process of the PowerPC Chip

In order to fully understand the practical implications of applying formal verification to a large design, it is first necessary to describe the framework in which the verification tool operates. For this project, due to tight schedule demands, many design tasks were commenced in parallel. Similarly, the overall design methodology was constantly updated and improved to reflect the current state of the design process.

3.1 Parallel Design Practice

In a practical microprocessor development project, it cannot be assumed that all previous design steps are completed before the next step begins. For example,

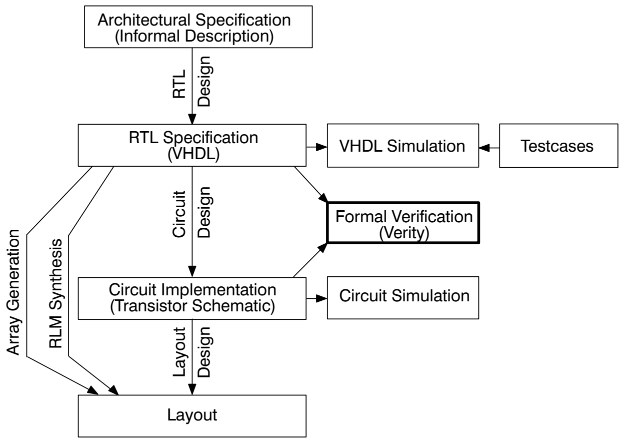

Figure 1: General design flow used for the PowerPC microprocessor from a functional verification point of view.

in order to build and verify a circuit, the RTL specification describing its function might not necessarily be completed, nor fully tested. In the given project, a parallel development strategy was implemented that spanned all design activities from the RTL specification to the layout implementation.

In order to regularly synchronize the design activities, a snapshot methodology was applied. A snapshot was defined as a consistent and structurally intact design point which did not necessarily implement the complete or correct function. For example, once the RTL specification for a specific instruction had been coded, a snapshot was taken and delivered to the circuit and physical design team for schematic and layout implementation. The snapshot technique was also used for the fine tuning of the overall design methodology as new tools or methods could be exposed to stable data before they were released for general application.

The general design flow of the PowerPC development is shown in Figure 1. The primary input to the design process was the PowerPC micro-architecture specification, a plain English description of the functional details for each machine instruction, register, and other details of the microprocessor. Starting from this specification, five distinct and parallel processes were undertaken which included: (1) RTL design, (2) circuit design, (3) functional verification, (4) timing analysis and correction, and (5) layout design, floorplanning, placement and wiring. For the purposes of this paper, only the first three items are of particular interest and are explained below.

3.2 RTL Design

This initial design step included the development of the RTL specification from the architectural specification. The resulting VHDL source represented the first formal and complete model of the microprocessor. The RTL hierarchy was structured into three distinct layers, each comprising one or more hierarchy levels:

Chip layer: This level represents the entire PowerPC description.

Functional unit layer: This layer represented the first order partitioning of the processor into functional blocks such as integer, branch, and dispatch units. The typical unit size was about 25,000 executable lines of VHDL representing about 200K transistors.

Component layer: Each functional unit was composed of multiple components. Depending on the component size and functionality, several hierarchy levels were used for their VHDL models. The average component source contained 7000 executable lines of VHDL which corresponded to about 60,000 transistors.

The specific structure of these hierarchy layers was designed based on three factors: (1) Designers ability to effectively manage a particular hierarchy node, (2) complexity limitations of the physical design tools, and (3) complexity constraints imposed by Verity for hierarchical formal verification.

3.3 Circuit Design

Given the hierarchical RTL specification as described above, the circuit implementation started at the component level. For each RTL component a corresponding circuit implementation was designed, while the hierarchy information for the chip and functional unit layers was directly adopted from the RTL model.

Due to performance requirements, the majority of the circuit components were custom designed at the transistor-level. A significant portion of these designs utilized static CMOS techniques, including pass-transistor and transmission-gate circuits. Performance critical components, such as Content Addressable Memory (CAM) cells and ROM designs were implemented by precharge logic. In a few cases, logic synthesis in conjunction with standard cell layout techniques and array generators for PLAs and ROMs were applied to automatically generate circuit components from the VHDL source.

3.4 Functional Verification

The verification methodology used in this PowerPC project employed a variety of techniques and tools to implement RTL simulation, circuit simulation, timing analysis, and formal verification. For the purposes of this paper, both RTL and circuit functional verification is described covering pattern-based simulation, and formal verification.

Pattern-based simulation: A variety of test cases were simulated on the VHDL and circuit-level models, and the response was compared with the nominal behavior. The set of test cases for VHDL simulation included manually designed test programs and randomly generated test sequences which could be biased toward specific targets. Similarly, stimuli for circuit simulation included: (1) Manual stimulus patterns, (2) input stimuli generated by fault model pattern generators and (3) simulation sequences captured during system simulation for a particular component. Functional simulation was applied on individual components and on the system level containing the entire microprocessor. Overall, to verify the correctness of the RTL and circuit-level specifications, extensive simulation on a large cluster of workstations was performed.

Formal Verification: Verity was applied throughout the design process to check the consistency between the RTL specification and the transistor-level circuit implementation. This formal comparison step was based on the hierarchical system partitioning and was tightly incorporated into the snapshot methodology. Therefore, is was possible to apply the tool continuously at all stages of the design cycle.

4 Practical Circuit Design with Verity

Verity was fully integrated into the circuit design environment, allowing the user to invoke the tool automatically from the schematic entry system. Additional design information such as logical boundary conditions, specific functional tests, and verification options were entered once for each design and then reused in successive verification runs. In addition to the interactive application of Verity, batch submission and version control mechanisms automatically verified updated parts of entire subtrees of the design hierarchy.

With respect to the hierarchy partitioning described above, the application of Verity was distinguished between the component layer and the chip/functional unit layer. For the components, the circuit and RTL models were functionally compared and the logical boundary conditions were validated. Since the structural interconnection of the circuit components were adopted from the RTL model, a functional comparison for the chip and functional unit layers was redundant. For full chip verification, it was sufficient to validate the consistency of the logical boundary conditions on these levels.

Depending on the complexity, the component verification was performed either flat or hierarchically. In the flat case no restriction were implied on the descriptions of the components. For hierarchical verification, the top part of the circuit and RTL hierarchy had to match and, if necessary, consistent logical boundary conditions had to be specified for the inputs and outputs of the subcomponents.

Advantages of Designing with Verity:
Because of its tight integration into the schematic design process, Verity was essentially a push button tool quickly reporting functional correctness of the current design state. The initial setup of the verification data for a particular design required some effort for specifying the boundary conditions and the appropriate verification options (typically 10 to 15 minutes). However, since the data was reused for subsequent verification runs, the repeated application of Verity was far more productive (3 to 4 minutes for an average circuit). The efficiency of formal verification shifted the overall circuit design paradigm from a *design-then-test* style more towards a *trial-and-error* mode in which Verity was used as a design engine testing iterative design attempts.

Disadvantages of Designing with Verity:
Although Verity could handle large circuits, components often needed to be verified hierarchically, with the restrictions described above. In a few cases where circuit and RTL designs were started before formal verification was introduced, a significant amount of repartitioning was needed to make the component pass formal verification. In general, this could have been avoided by a close collaboration between the RTL and circuit designer. However, due to tight schedule demands, this could not always be achieved.

5 Verification Statistics

The entire PowerPC chip was implemented in 139 custom designed components, 38 random logic macros (RLM) and 65 PLA's. Overall, Verity was applied to 113 custom components and to all RLM's and PLA's, totalling 89 % of the chip components. 26 components were not formally verified because 18 of them contained large storage arrays (ROM's, cash-arrays) and the remaining 8 components were too complex and could not be repartitioned due to schedule constraints. These circuits were verified by extensive simulation. In total, 39 circuit designers invoked Verity 7920 times, totalling 746 CPU hours.

In the following, various verification statistics are discussed which were collected over the duration of the project. We use *circuit* to denote a design part which was verified as a single entity. The reported numbers correspond to the flat circuit structure excluding the black-boxed components. Figures 2 and 3 detail the computing resource required for verifying all circuits. For the verification, a pool of remote RS/6000, model 580/590 workstations was used for job submission including a dedicated machine loaded with 2 GBytes of real memory. It is notable that 95 % of the circuits could be verified within 800 CPU seconds using less than 30 MBytes of memory. This attests to the push-button characteristic of the verification approach for the majority of applications. For the remaining few cases, it was acceptable to invest additional computing resources or to have a dedicated verification engineer investigate alternative tool options to successfully verify the circuit.

The following three figures are used to illustrate the continuous application of formal verification during the project. Figure 4 shows the number of Verity runs per circuit and Figure 5 displays their distribution over the project duration. It is evident, that instead of applying Verity at the end of the design cycle only, formal verification was a constant part of the entire

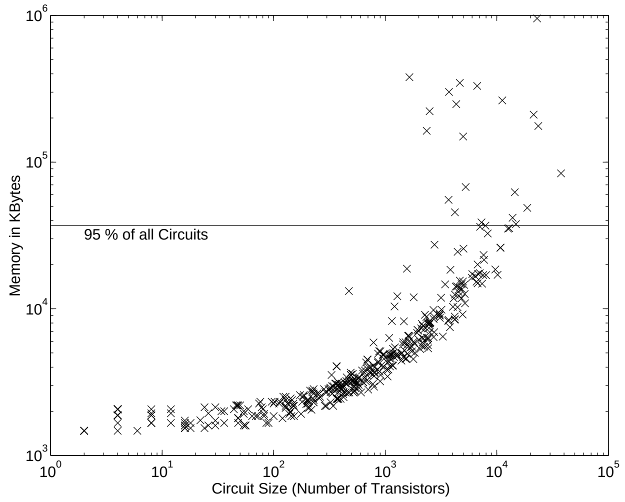

Figure 2: Memory usage versus design complexity for all verified circuits.

development process. To demonstrate the *trial-and-error* mode in using Verity as a design engine, Figure 6 reports the application success rate over the project duration. Here, verification success is referred to as confirmed functional equivalence including successful tests of boundary assertions.

In order to illustrate the hierarchical verification approach, Figure 7 shows histograms of the black-box usage and the corresponding application of boundary conditions. It is notable that about 80 % of the boundary assertions could be validated by applying hierarchical verification. The remaining assertions were either of a sequential nature or they span large portions of the design that could not be handled flat. RTL simulation was used to validate the remaining assertions based on the given set of test cases.

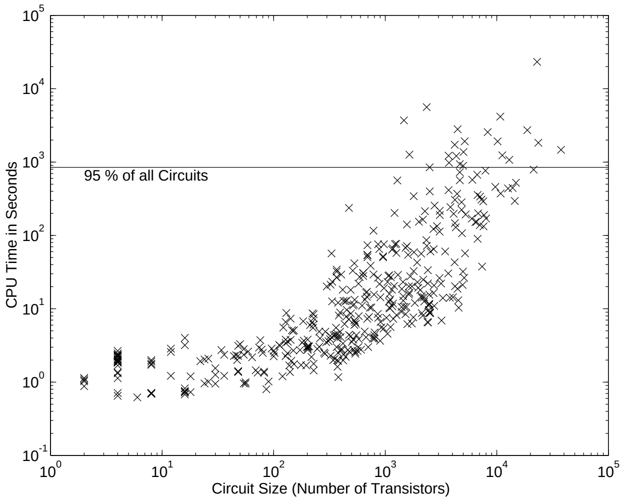

Figure 3: Runtime versus design complexity for all verified circuits.

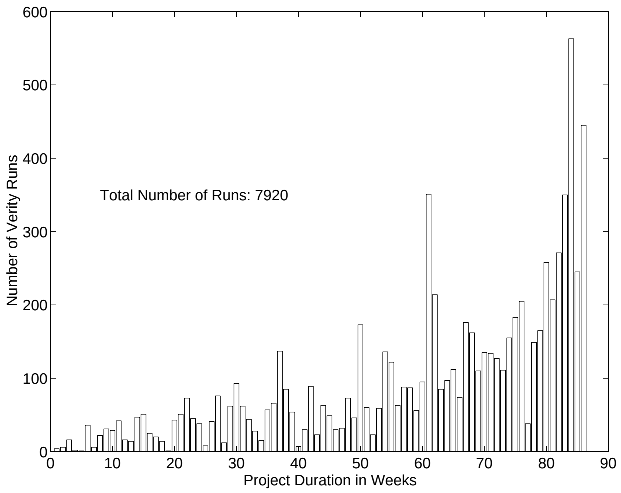

Figure 4: Application of Verity during the project.

6 Conclusions

In this paper we presented the application of formal verification to the development of a PowerPC microprocessor. The design methodology employed a RTL specification and a transistor-level implementation which were formally compared for functional equivalence. The chosen approach applied a combinatorial verification model, where corresponding registers of the two design views had to be identified. It was shown that, using this model in conjunction with a hierarchical verification scheme, complex microprocessor systems can be formally verified.

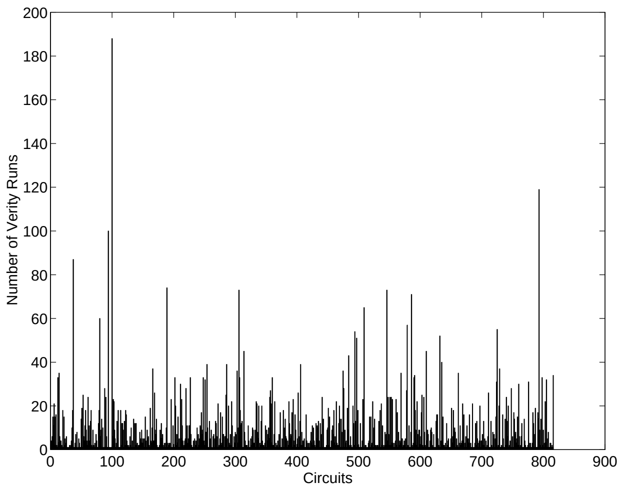

Figure 5: Number of Verity runs per circuit.

A verification approach based on an equivalent system partitioning required the users to specify logical boundary assertions which need to be validated during hierarchical verification. Although most of the assertions could be validated, sequential boundary assertions and those spanning large portions of the system could not be handled. For the complete verification of large systems this remains an open problem which needs to be addressed in future research activities.

A tight integration of the verification tool into the design environment allowed users to continuously prove the correctness of the circuit implementation throughout the design process. Due to efficient tool usage, a shift of the application from a *design-then-verify* style to a *trial-and-error* design style could be observed.

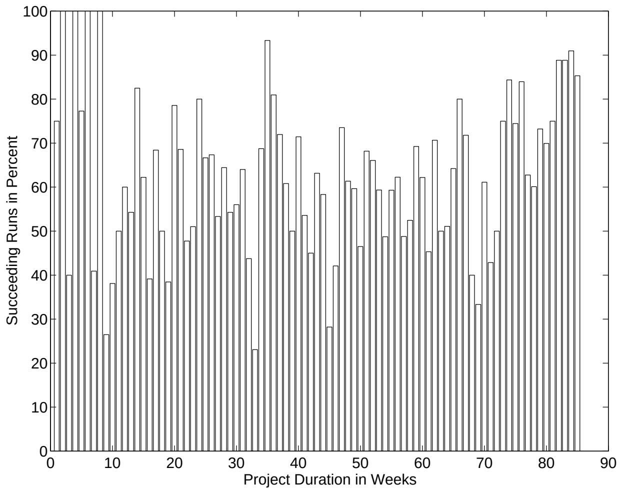

Figure 6: Success rate of Verity runs during the project.

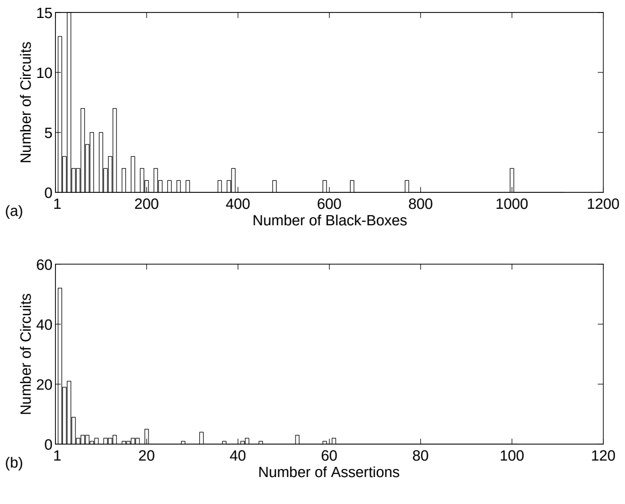

Figure 7: (a) Histogram of using black-boxes, (b) Histogram of using logical boundary assertions for all circuits.

7 Acknowledgements

The authors would like to thank Kurt Carpenter, Elizabeth Bouldin, and Greg Rodgers from IBM Burlington for their signficant support to incorporate formal verification into the PowerPC verification methodology. They also wish to thank David LaPotin, Florian Krohm, Arjen Mets, and Mark Williams from IBM, Arvind Srinivasan, currently at Mentor Graphics, and Geert Janssen from the Technical University Eindhoven for their invaluable contributions to fine tune Verity for the PowerPC project.

References

[1] K. L. McMillan, *Symbolic Model Checking.* Boston, MA: Kluwer Academic Publishers, 1993.

[2] M. Genoe, L. Claesen, E. Verlind, F. Proesmans, and H. D. Man, "Automatic formal verification of Cathedral-II circuits from transistor switch level implementations up to high level behavioral specifications by the SFG-tracing methodology," in *Proceedings of The European Conference on Design Automation*, (Brussels, Belgium), pp. 54–58, IEEE, February 1992.

[3] A. Kuehlmann, A. Srinivasan, and D. P. LaPotin, "Verity - a formal verification program for custom CMOS circuits," *IBM Journal of Research and Development*, vol. 39, pp. 149–166, January/March 1995.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, pp. 677–691, August 1986.

[5] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 42–47, IEEE, November 1993.

[6] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin, "Error diagnosis for transistor-level verification," in *Proceedings of the 31th ACM/IEEE Design Automation Conference*, (San Diego, CA), pp. 218–224, IEEE, June 1994.