Practical Considerations in Formal Equivalence Checking of PowerPC[1] Microprocessors

Arun Chandra*, Li-Chung Wang♦, and Magdy Abadir♦

*IBM, Somerset Design Center, Austin, TX 78730
♦Motorola, Somerset Design Center, Austin, TX 78730

Abstract

Recently, formal verification is becoming more a part of the VLSI design methodology. Formally verifying a design guarantees 100% coverage and negates the need to do simulation. Theoretically, 100% coverage is very appealing and formal verification looks to be the panacea to solve the coverage problem. However, there are many practical considerations in deploying formal verification in real design environments. These considerations if not evaluated can lead to ineffective and even erroneous formal verification methodologies.

In this paper we show how to make formal verification a successful part of a design methodology by paying attention to practical considerations and knowing the limitations of formal verification. We show the errors that can result by making over generalized assumptions and how they can be avoided. We do this in the context of the design of PowerPC microprocessors. We limit ourselves to a formal verification technique commonly used in our design methodology--boolean equivalence checking

1. Introduction

Design Verification is an important problem in the VLSI design area. Design Verification is checking to see whether the system performs its intended function. Design verification techniques can broadly be classified into at least three distinct areas. The first uses simulation to functionally verify designs. Simulation involves running a preselected set of patterns on a VLSI design and comparing its response with an expected response in order to verify the correctness of the design. Simulation is usually done using a high-level (RTL) view of a design. This type of verification is usually termed *functional verification*. Functional verification techniques rely upon automated random pattern generation techniques. Also, coverage is never 100% and to achieve high levels of coverage unreasonable amount of simulation cycles (in the order of billions) have to be run. *Timing verification* (or timing analysis) primarily involves estimating resistances and capacitances and calculating path delays and verifying that the delay characteristics of a circuit meet prespecified criteria.

Finally, *formal verification* is primarily used for equivalence checking, constraint checking, and model checking. *Equivalence checking* is proving the equivalence of two pieces of logic, or proving the equivalence between two different views of the same logic. *Constraint checking* is proving the satisfiability of a prespecified constraint. *Model checking* involves checking the state transition properties of a design. Formal techniques have 100% coverage and negate the need for simulation.

Formal verification techniques are very appealing and theoretically promise a very short verification time and 100% coverage. However, in real design environments one must evaluate the practical considerations involved in deploying formal verification techniques. If formal verification is deployed naively it can lead to longer verification times and erroneous situations. *In erroneous situations real design bugs can be masked or long times are spent chasing false bugs.*

2. Formal Verification Methodology

Figure 1 shows the general formal verification methodology in use. *Equivalence checking* can prove the equivalence between the RTL and the schematic for both custom and synthesized circuits. Checking between the RTL and the VLSI circuit proves that what is being simulated is being physically built. Additionally, it can prove the equivalence between the RTL and the gate-level view to be fed into an ATPG tool for testing purposes. Model checking and constraint checking is usually done on the RTL view of a design.

Equivalence and model checking tools use some form of Ordered Binary Decision Diagram (OBDD) packages [Bryant 86] as the core. As a result, the success of formal verification especially with respect to the size of the circuit that can be handled depends upon the algorithm used in the OBDD packages.

0-8186-8409-7/98 $10.00

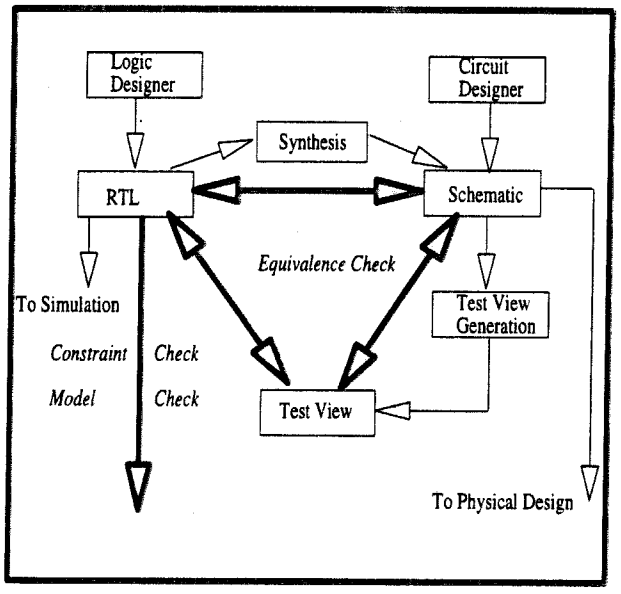

Figure 1 Formal Verification Methodology

3. Practical Considerations

In this section we show the practical considerations involved in deploying formal equivalence checking in real environments. We do this in the context of the design of PowerPC Microprocessors. The major issues we cover are: 1) Label Correspondence, 2) Sequential Designs, 3) Constraints, 4) Circuit Elements, 5) Dynamic Logic, and 6) Capacity.

3.1 Label Correspondence

In general, a well defined and good labeling mechanism makes debugging designs easier. In the practical implementation of formal equivalence checking, label correspondence can become a weak link. Label correspondence is an issue because often the circuit designer uses different labels than the logic designer. Also, some synthesis tools modify labels during synthesizing a gate-level or transistor-level view.

The label mismatches that cause immediate problems are the mismatches on primary inputs or outputs. Equivalence checking with label mismatches is usually performed using a manual label correspondence mechanism. There is a possibility of a manual label correspondence mechanism incorrectly masking true errors. This is shown in Figure 2. In this figure, the incorrect correspondence in the correspondence file, between pin A in the RTL and pin C in the schematic and vice-versa, leads to the incorrect matching of two inequivalent functions (A.B + C vs. A + B.C).

Another problem with label mismatches are that they lead to false bugs which are then tracked to the label correspondence mechanism. This ultimately leads to long verification times and designer frustration.

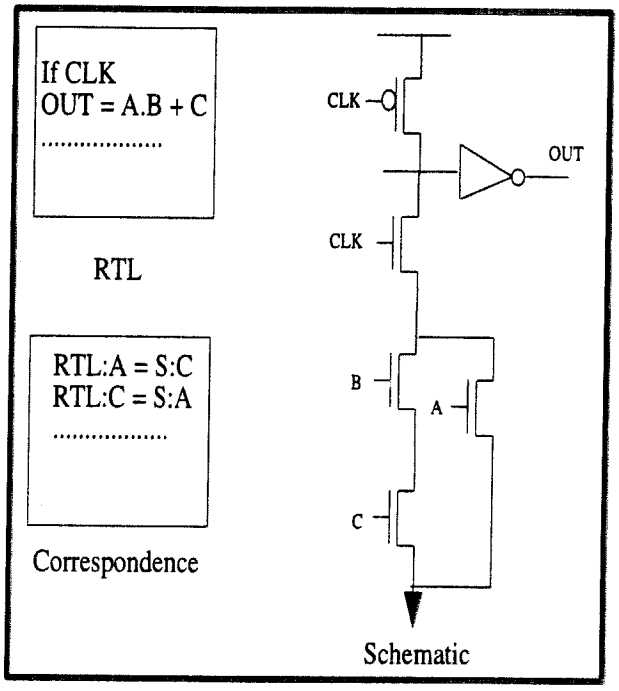

Figure 2 Incorrect Labels Masking Logic Errors

3.2 Dealing With Sequential Designs

Sequential designs are a challenge for equivalence checking. Equivalence checking tools are primarily designed for combinatorial circuits. In modern superscalar microprocessors like the PowerPC, arrays and latches are frequently present in critical custom design circuits. As a result, concentrating on purely the combinatorial circuits leaves a huge hole in the formal verification methodology. When latches or arrays are encountered, equivalence checking tools must be used in a special fashion. In what follows we will cover the issues involved in formally verifying circuits with array and latches.

In circuits with latches or arrays, most tools formally verify the combinational logic around the latches. The latches can then functionally verified using simulation. There are two primary techniques that can be used to verify circuits with sequential elements. The first common approach is the blackboxing approach. The latches are blackboxed and therefore the latch logic (feedback loop) is ignored. As a result of this blackboxing, the formal verification problem is actually split into two phases. Verifying the logic before the latch and verifying the logic after the latch.

Blackboxing opens up a very important practical problem of latch correspondence. Latch correspondence is practically only feasible if the two models follow the same hierarchy and the names of the internal nets which are the primary inputs and outputs from the blackboxed circuits are the same. This is usually not the case. Without this many false errors are reported by an equivalence checking mechanism. Also, as latches are blackboxed, false errors

can result between two models if they are equivalent after two levels of latches. Figure 3 shows an example of this. The design shown in this figure has nine latches (A..I), two And gates, and one Or gate. In this figure, equivalence checking fails because even though both the designs have the same final function, they are functionally different at intermediate points (F). As a result, the equivalence checking mechanism gives false errors in both the combinatorial cone checking phases.

It is interesting to note that latch correspondence is generally easier in scan-based designs as many latches can be identified by following the scan chain. In the absence of scan logic, the latch correspondence problem is much harder.

Circuits with latches can also be verified using cutpoints. A cutpoint is identified in the feedback loop of the latch. The equivalence checking methodology then reduces to verifying the combinational logic before the cutpoint and verifying the combinational logic after the cutpoint. The most important practical issue here is the designer must identify, specify, and correspond these cutpoints in the two models. Incorrect cutpoint specification or correspondence can lead to false errors and in some rare cases true error masking.

Even though the cutpoint approach and the blackbox approach are semantically equivalent, the cutpoint approach is easier to implement as it avoids the need to restructure the design to enable black boxing.

In some cases modifications are made to the RTL-level model to improve simulation performance. A typical example is the use of a transparent latch to improve simulation time. However, with transparent latches, latch correspondence fails.

For embedded arrays it is impractical to use traditional static formal equivalence checking tools. Using plain simulation or the more powerful formal method of Symbolic Trajectory Evaluation [Ganguly 96], [Pandey 96], [Seger 95] is a much better and less error prone approach. If the array is part of a bigger cell to be verified then the only solution is to *blackbox the entire array.*

3.3 Constraint Handling

Steering logic implemented with pass transistors is used abundantly in VLSI circuits. Circuits using pass transistor logic (e.g. Multiplexers) have incomplete truth tables. As a result, to avoid reporting false errors, formal logic verification tools must be provided with *constraint information.* An example of a constraint on a Multiplexer is the *orthogonality constraint*--exactly one select line should be on at any one time. If the orthogonality constraint is not provided false signal collision errors or floating line errors will be reported.

Constraints are usually provided for by manual statements. This manual provision of constraints has to be done with utmost care to specify the correct constraints. Incorrect constraints can lead to longer verification times and more seriously, can also lead to bugs being masked. An example of incorrect constraint statements (S1 is orthogonal to S2, S3 is 0), leading to an inversion error being masked is shown in Figure 4. In this figure because S3 is set to 0, the incorrect path will never be checked.

In many complex transistor-level circuits finding the complex set of constraints to make the behavior of the circuit formally verifiable is a non-trivial error prone task which makes it another weak link which must be addressed by a good formal verification methodology.

A good methodology for equivalence checking with constrained circuits is to use hierarchy to check with and test constraints. A circuit can be checked with a set of constraints. Following that, at a next higher level where the circuit is used, the set of inputs feeding the circuit can be tested to see if they obey these constraints. This methodology does add to the complexity of the formal verification process. Further, you cannot resolve the constraints past latch boundaries and more seriously cannot address the fact of the constraints being correct.

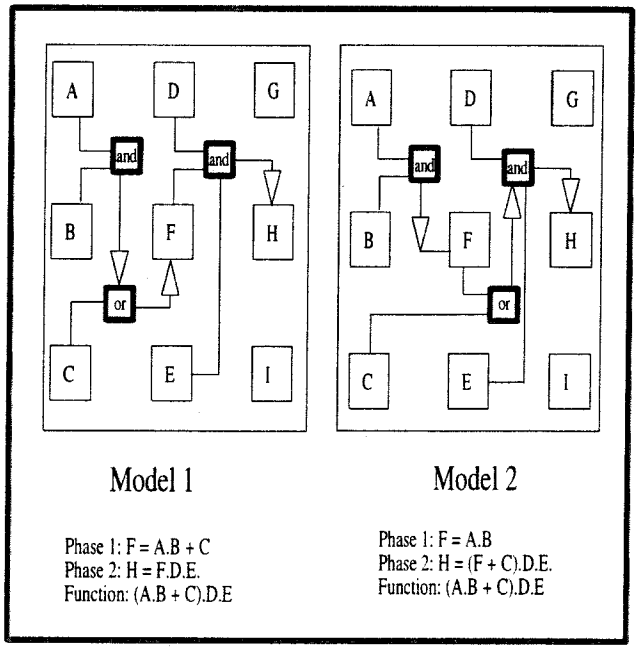

Figure 3 False Errors in Sequential Circuits

3.4 Handling Transistor Circuit Elements

Formal equivalence checking is very beneficial at the transistor level because it finds errors introduced by circuit designers. However, if formal equivalence checking has to be used successfully at the transistor level, the handling of transistor circuit elements must be addressed. Without the above, any formal verification methodology is bound to fail due to long verification times and error masking.

Circuit elements like tristate buffers and resistive transistors affect the way a transistor level circuit is formally verified. Tristate buffers have a valid high

impedance (or Z) state. Most formal verification techniques cannot deal with any arbitrary state other than 1 and 0. As a result, all the logic downstream from tristate buffers cannot be validated. This is because there is no way to propagate the Z value down a piece of combinational logic. A way to get past tristate buffers is to set them in the enable state. The only problem with this approach is that it can become tedious for large designs. Also, it masks errors caused by the tristate buffers in the disable state.

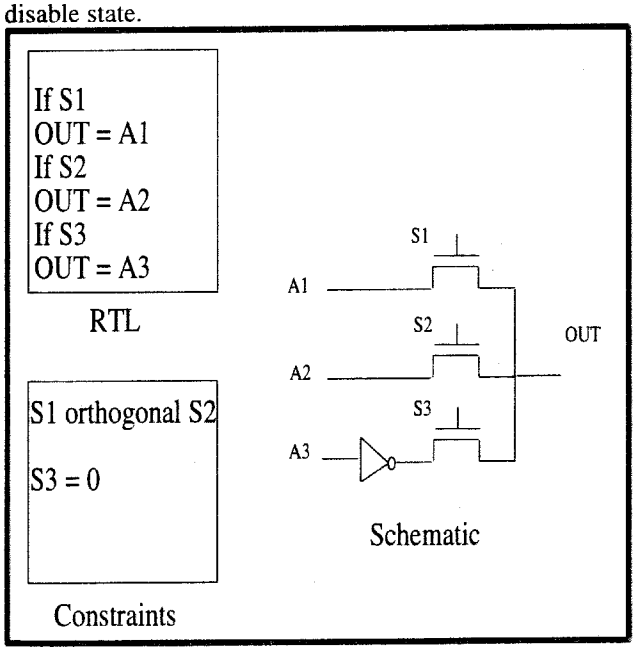

Figure 4 Incorrect Constraints Masking Errors

Another common circuit element is a resistive transistor which has low gain. A resistive transistor prevents floating conditions by pulling up a signal to a high value or pulling down a signal to a low value if the signal is not being driven. In the case that the signal is being driven, the driven signal value overrides the effect of the resistive transistor. As a result, the resistive transistors do not interfere with the logical functionality of a circuit. For formal equivalence checking, usually a resistive property has to be added onto the transistor to enable the correct interpretation of the transistor-level circuit.

Finally, a common circuit element is the keeper. A keeper keeps the previous value of the circuit. Keepers are also called rail-pullers. Keepers are like half latches and have loops and introduce sequentiality. As a result, formal verification mechanisms must be provided with relevant information if formal verification is to be performed on circuits which contain keepers. Keeper transistor are usually ignored because they do not influence the base functionality of a circuit. However, ignoring keeper circuits creates a hole and errors in the keeper circuit and errors caused by the keeper circuit get masked.

Explicitly declared properties like the resistive and keeper property are dangerous because they can override the actual transistor-level behavior of the circuit. Figure 5 shows an example of this. In this figure, an error is masked if transistor P1 is not actually resistive. Inferred properties (e.g. via pattern matching or W/L ratios) are the right way to go. However, inferring properties is non-trivial.

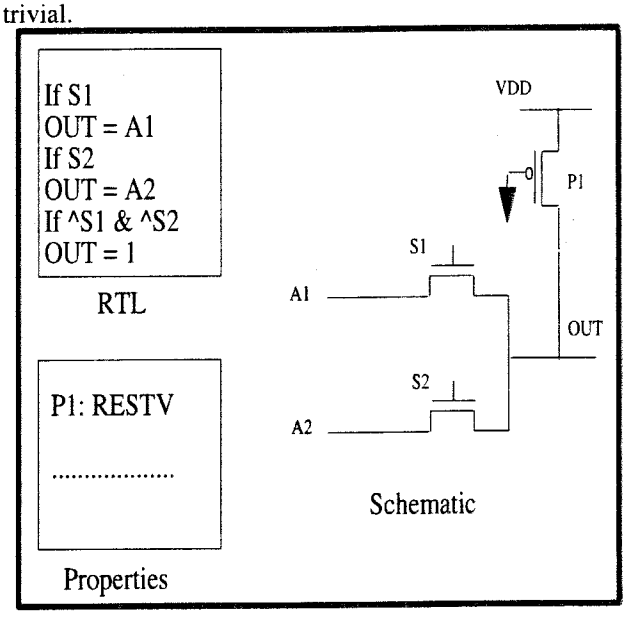

Figure 5 Incorrect Circuit Element Representation

3.5 Dealing With Dynamic Logic

Dynamic logic introduces pseudo-sequentiality into a circuit and has to be handled in a special fashion. As opposed to static logic, dynamic logic has 2 to 4 phases each of which need to be checked for full coverage. Formally verifying dynamic logic then reduces to the problem of verifying sequentiality in circuits. However, in practice simplifying assumptions are made to verify dynamic logic.

A common assumption made is to do an equivalence check in the evaluate phase only. This can be done by asserting the precharge clocks to be enabled. However, this leaves holes in the formal equivalence checking methodology in that the errors in the precharge phase are masked. Figure 6 shows how errors (P transistor gated by B) in dynamic circuits can escape if the precharge phase is not verified. In this figure, the CLK is a precharge clock.

Another way to deal with dynamic logic is to verify the precharge phase and the evaluate phase separately. This methodology makes the equivalence checking complex especially for 4-phase dynamic circuits. Also, this methodology does not guarantee that all errors will be caught and is some rare cases errors can be missed. Usually, some amount of customization is needed to use formal checking for dynamic logic.

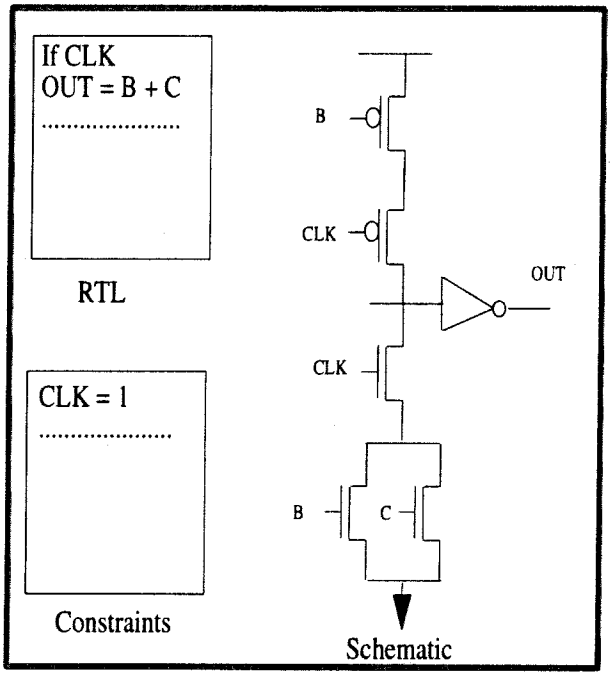

Figure 6 Errors Masked in Dynamic Logic

Finally, dynamic latches cause a problem when used in conjunction with cutpoints (Section 3.2). The weak inverters in dynamic latches must be explicitly declared. However, this leads to the same problem of incorrect property specification.

3.6 The Capacity Problem

For complex industrial designs formal equivalence checking is computationally very complex, hence it is necessary to partition the designs. Partitioning the design is a non-trivial task. Partitioning designs is usually done by introducing cutpoints. Cutpoints are nets which break up the circuit and then formal verification takes place by verifying the partitions of circuit. However, there is a possibility that the partitions individually might require one to check functionality that is never used in the whole circuit and thus leads to false errors.

Designers have to make the circuit partitionable and provide a set of cutpoints for complex circuit verification. The challenging aspect of cutpoint assignment is that the cutpoints should correspond between the views. Incorrect cutpoint specification or correspondence leads to long verification times with many false starts. The need for cutpointing is alleviated if one uses good hierarchical verification techniques.

4. Practical Formal Verification Methodology

For formal equivalence checking of PowerPC microprocessors we use primarily a OBDD-based approach. Also, we use the equivalence checking primarily for combinatorial designs. We follow Design for Verifiability guidelines to make certain we avoid the common pitfalls shown in the previous section. Our equivalence checking approach is hierarchical and we make use of blackboxing frequently. This hierarchical approach alleviates the capacity problem and we can verify large designs using this approach.

In general, the hierarchical approach is accomplished using a bottom up approach as follows:

- Determine the Equivalence of the Lower-Level Cells
- Black Box the Lower-Level Cells
- Verify Equivalence of Interconnects in Higher Cells
- Verify Constraints Between High Level Cells

A key component of our formal equivalence checking methodology is keeping a *strict audit* on the label correspondence and constraints files. All label correspondence and respective constraints are audited by designer and verification engineers by file walkthroughs. Constraints are especially audited for pass gate logic circuits.

For sequential designs, we use the cutpoint approach because of ease of use as compared to the blackbox approach. In most cases this approach works well. However, in some cases we get false errors. In these cases, we use simulation to prove that the errors indicated by the equivalence checking methodology are not true errors. For sequential designs, this is the best possible approach as there is no way to ensure no false errors. Also, for formal equivalence checking we do not allow transparent latches in the RTL.

As we formally verify many transistor-level circuits we have to handle certain circuit elements. Each circuit element is handled in a special fashion in our formal verification process. Tristates, resistive transistors, and keepers are required to be labeled as such. The formal equivalence checking then proceeds with these pre-specified transistor properties. Resistive transistors are interpreted as weak transistors and, using rules, the functionality of the circuit is derived. In our methodology we ignore keepers. Keeper circuitry is verified by simulation. Finally, we use constraints to verify tristated circuits. Additionally, these special transistor properties are audited by both designer and verification engineer by file walkthroughs. Also, the danger of using these properties is reduced because other tools use these same properties for different reasons.

We verify both the phases of most 2-phase dynamic circuits. For very complex dynamic circuits we use constraints. Also, we appropriately label weak inverters in dynamic latches and audit the properties. Figure 7 shows our overall formal equivalence checking methodology. The additional steps we take makes it a success and eliminates the holes in the methodology. Also, we continue to refine our methodology to make it more robust. This ensures continuous improvement.

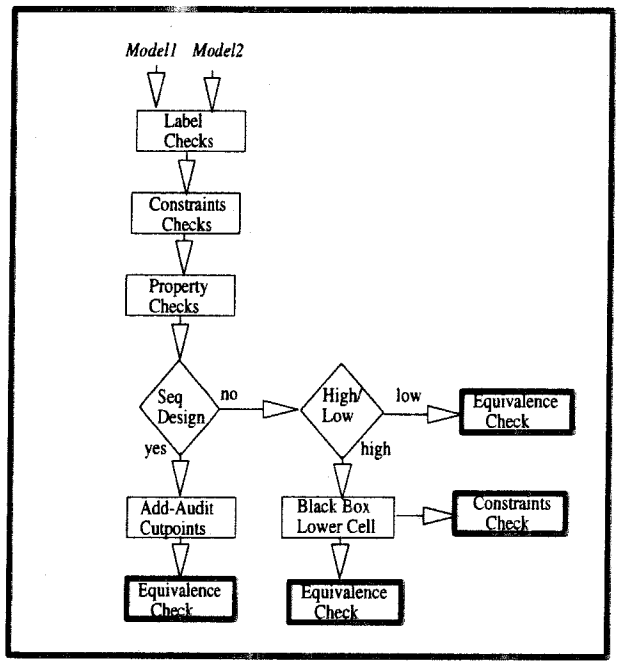

Figure 7 Overall Formal Verification Methodology

5. Conclusions

In this paper we show how to make formal verification a successful part of a design methodology by paying attention to practical considerations and knowing the limitations of formal verification. We show the errors that can result by making over generalized assumptions and how they can be avoided. We do this in the context of the design of PowerPC microprocessors. We limit ourselves to formal equivalence checking which is used commonly in our design flow.

Concluding, there are many practical considerations in deploying formal verification in real design environments. These considerations if not evaluated can lead to ineffective and even erroneous formal verification methodologies.

References

[Abraham 96] Abraham, J. 1996. "Formal Hardware Verification for Engineers," Tutorial Notes.

[Bryant 86] Bryant, R. E. 1986. "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35(8), pp. 677 - 691.

[Ganguly 96] Ganguly, N., M. S. Abadir, and M. Pandey. 1996. "PowerPC Array Verification Using Formal Verification Techniques," *International Test Conference*, pp. 857 - 864.

[Kuehlmann 95] Kuehlmann, A., A. Srinivasan, and D. P. LaPotin. 1995. "Verity - A Formal Verification Program for Custom CMOS Circuits," *IBM Journal of Research and Development*, Vol. 49, pp. 149 - 166.

[Malley 95] Malley, C., and M. Dieudonne. 1995. "Logic Verification Methodology for PowerPC Microprocessors," *Proceedings 32nd ACM/IEEE Design Automation Conference*, pp. 234 -240.

[Pandey 96] Pandey, M., R. Raimi, D. L. Beatty, and R. L. Bryant. 1996. "Formal Verification of PowerPC Arrays Using Symbolic Trajectory Evaluation, *Proceedings 33rd ACM/IEEE Design Automation Conference*, pp. 649 - 654.

[Seger 95] Seger, C., and R. E. Bryant. 1995. "Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories," *Formal Methods in System Design*, Vol. 6, pp. 147 - 189.

[Seger 93] Seger, C. 1993. "Voss - A Formal Hardware Verification System - Users Guide," Technical Report 93-45, Department of Computer Science, University of British Columbia.

Footnotes

[1] PowerPC is a Trademark of the IBM Corporation in the United States, or other countries, or both.