Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 174 (2007) 3–7
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

From Error to Error: Logic Debugging in the Many-Core Era

John Moondanos[1]

Formal Technologies
Design Technologies and Solutions
INTEL Corp., M/S SC12-605
2200 Mission College Blvd.
CA 95054, USA

Abstract

Design and manufacturing of present day Multi-Core microprocessors has to overcome major technology obstacles, particularly in the areas of Power and Validation. More specifically, the state of the Art in the area of validation is such that upwards of 30% of the human resources in a modern microprocessor Design Team are dedicated to it. The term *Validation Productivity Gap* has been introduced to describe exactly these ever increasing resource requirements. Formal Verification techniques offer one of the technological approaches for addressing many aspects of this validation gap. Therefore it is of paramount importance to develop technologies and methodologies for reducing the time it takes to debug and rectify Logic Errors that are detected by Formal Verification Techniques. Such developments would offer a much needed productivity improvement both in pre- and post-Silicon validation activities.

Keywords: Formal Verification, Equivalence Checking, Property Verification, Logic Debugging, Logic Circuit Rectification.

1 Introduction

A little over a decade after the Pentium FDIV bug, design engineers everywhere are enjoying the benefits of the widespread use of Formal Property Verification (FPV) and Formal Equivalence Verification (FEV) tools. The significance of the role that these tools have played in bug-free tape-outs over the years cannot be overstated. Unfortunately, debugging the process of identifying and correcting the root cause of a design error seems to remain a labor-intensive undertaking and has become one of the major bottlenecks of FV activities. An engineer still has to notice something, question why it happens, set up hypotheses, and then attempt to confirm or falsify them. As has been correctly stated, there seems to exist a non-algorithmic, psychological component to current debugging methodologies. Unfortunately, the

[1] Email: john.moondanos@intel.com

1571-0661 © 2007 Published by Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2006.12.035

stringent time-to-market requirements at top microprocessor companies do not allow us to continue with this legacy. To quote leading experts in this field, debugging should be just as disciplined, systematic, and quantifiable as other areas of logic design — which means that we should automate it. Fortunately, recent progress in CAD tools has brought breakthrough improvements in terms of the circuit size and complexity that our formal verification tools can handle. This talk focuses exactly on taking advantage of this recent progress to increase designer productivity in formal verification applications by exploiting advanced automated debugging algorithms.

To define the operations involved in debugging a design, let us first review the facets of a design that engineers are examining in this era of multi-core processors. When we design a system we deal mainly with three aspects. Its specification (in terms of properties or a circuit at various levels of abstraction), the actual model and the environment within which the model operates (expressed frequently as a set of dont care conditions). On the other hand the term Design Debugging refers collectively to three separate but complementary activities. **Error explanation** describes any approach to aiding a user in moving from a particular example of specification failure to an understanding of the essence of the failure. **Error Localization** is the more specific task of identifying the faulty core of a system by locating the components that are the root cause of the problem. Finally, **Error Rectification** deals with techniques for actually correcting the problem.

The careful reader will appreciate the importance of automated formal debugging algorithms for additional reasons, other than the fact that they stand to make the logic debugging process less time consuming and less frustrating. In addition to fixing implementation errors, these techniques are also of paramount importance in the process of design revision due to specification changes. A specification change is performed when an error in the high-level specification is found during the design validation process. In practice, as many of us have experienced over the years, a specification error is found late in the design cycle when the logic synthesis or even placement-and-route has been completed. In order to re-use the spent engineering effort, a designer tends to patch the old implementation and make it conform to the new specification, instead of re-synthesizing from scratch. This process known as ECO [1] calls for methodologies that enable very efficient design revisions. Both ECO and error debugging can clearly be formulated as rectification problems in the logic domain.

2 Logic Debugging and Rectification Technologies

Debugging is an interactive and iterative process that requires hands-on involvement of designers, who must identify and resolve the root cause of a failure. One way to accelerate this process is by enhancing the capability of designers to understand and analyze their designs and verification results using advanced, automated analysis algorithms. These algorithms will increase the designers productivity by providing a smaller set of possible error locations and automated ways of correcting the design. These advanced debugging algorithms need to consider:

Trace Techniques: FPV and FEV tools produce counterexamples automatically; however as leading researchers have stated, isolated counterexamples are merely symptoms of the design error and not identifications of the root cause. To increase the value of counterexamples in identifying the root cause of the error, the techniques of this category attempt to:

(i) Localize the cause of the error in an error trace, and

(ii) Produce distinct error traces having different causes.

Such trace based techniques are particularly useful in debugging Formal Properties in the absence of a specification model. In the ideal debugging algorithm architecture we need to include:

(i) **Counter-Example-Minimization:** Help the debugging engineer by producing a minimal length sequence that leads to the same error state.

(ii) **Trace Clustering:** Group together counterexample sequences that seem to reach the same error states due to the same root causes

(iii) **Positive and Negative Trace Generation:** Create traces that are similar (with respect to a distance metric) to a given counterexample that lead to error states (negative traces) and good states (positive traces).

These techniques form the basis of the so called *Delta Debugging* [9] approach, whereby we attempt to root cause the problem in a design by examining its behavior for traces that are slightly different from the original counterexample.

Structural Techniques: These include utilizing gate dominators [2], various schemes of cut-point generation algorithms and other circuit structural information to identify and visualize similarities between the spec and the imp. They also include cone intersection whereby a signal that is not in the intersection of the incorrect outputs fan-in cones cannot be solely responsible for the error and, thus, cannot be the error site. In addition, back-propagation [12] techniques attempt to identify the root cause of an erroneous output by tracing signal paths that are regulated by gates whose inputs are set to the controlling values.

Symbolic Techniques: These techniques [1,3,5] employ reduced ordered BDDs to formulate the necessary and sufficient Boolean conditions for a signal or a set of signals to be the root cause of an error. In addition the play a major role in the rectification process as they encode the function of the new logic that must be implemented to correct the circuit.

Analytical Techniques: These [6,7] employ SAT solvers and the recent advances that have been implemented in the field. In the ideal debugging algorithm architecture we should have a strategy whereby we attempt to identify which gates in the circuit must be changed in order to correct the wrong values we observe at the circuits outputs for set of counterexamples, the size of which is user defined. The intuition behind this approach is that gates which when changed can correct the output of the circuit for a large number of counterexamples, are probably the actual root cause of the design error.

Simulation Based Techniques: These approaches [4,8] serve primarily as filters in the proposed algorithm architecture. They gradually exclude signals from the candidate list of error locations. In general, the main advantage of simulation-based techniques is that they are highly scalable. In many cases they also produce satisfactory diagnosis results [10]. However, simulation based approaches do not provide enough insight about how to fix the error(s). Thus, they are less suitable for applications that require automatic rectification (e.g., the ECO problem). The Sensitization Based Filter complements the output of a gate and attempts to propagate the effect of this change throughout the gates fan-out cone. If this change cannot be propagated to the output the gate is taken from the candidate error list. The Common Error Filter approach attempts to use quick simulation results to potentially identify whether some common gate level errors (e.g. missing/extra inverter/buffer, wrong gate type, etc.) are the root cause of the FPV/FEV runs.

Hybrid Techniques: These include Automatic Test Pattern Generation techniques (ATPG) which in effect combine the technologies underlying both structural and SAT based approaches [11]. The problem of error localization can be cast as an ATPG problem if we consider a D or D-bar to represent the pair (spec, imp) value, rather than the pair (good, faulty) circuit value.

3 Summary

All these techniques need to be improved, further expanded and combined, to accommodate the extremely complicated problems that the design of many-core processors poses for Formal Verification Techniques today. In particular, debugging of failed property runs targeted for protocol verification seems to be a crucial step towards implementation of correct multi-core designs.

References

[1] C.-C. Lin, K.-C. Chen, S.-C. Chang and M. M. Sadowska, *Logic Synthesis for Engineering Change*, Proc. Design Automation Conf., San Francisco, CA, pp. 647-652, Jun. 1995.

[2] H.-T. Liaw, J.-H. Tsaih, and C.-H. Lin, *Efficient automatic dianosis of digital circuits*, Proc. ICCAD, pp. 464-467, 1990.

[3] J. C. Madre, O. Coudert, and J. P Billon, *Automating the diagnosis and the rectification of digital errors with PRIAM*, Proc. ICCAD, pp. 30-33, 1989.

[4] V. Boppana and M. Fujita, *Modeling the unknown! Towards model-independent fault and error diagnosis*, in Proc. Intl. Test Conf., 1998, pp. 1094-1101.

[5] Guanghui Li, Ming Shao, and Xiaowei Li, *Design error diagnosis based on verification techniques*, 12th Asian Test Symposium, pages: 474-477, 2003

[6] Alexander Smith, Andreas Veneris, and Anastasios Viglas, *Design Diagnosis Using Boolean Satisfiability*, Proc. ASP-DAC'04, pp. 218-223, 2004

[7] Moayad Fahim Ali, Andreas Veneris, Sean Safarpour, Magdy Abadir, Rolf Drechsler, and Alexander Smith, *Debugging Sequential Circuits Using Boolean Satisfiability*, Fifth International Workshop on Microprocessor Test and Verification, pp. 44-49, 2004

[8] Debashis Nayak, D.M.H. Walker, *Simulation-Based Design Error Diagnosis and Correction in Combinational Digital Circuits*, Proc. 17TH IEEE VLSI Test Symposium, pages: 70-78, 1999

[9] A. Zeller and R. Hildebrandt, *Simplifying and isolatingfailure-inducing input*, IEEE Trans. Software Engineering,28(2), Febuary 2002.

[10] S.-Y. Huang, K.-T. Cheng, K.-C. Chen and D.-I. Cheng, *Error-Tracer: A Fault Simulation Based Approach to Design Error Diagnosis*, Proc. IEEE Intl Test Conf., Washington, DC, pp. 974-981, Nov. 1997.

[11] M. S. Abadir, J. Ferguson and T. E. Kirkland, *Logic Design Verification via Test Generation*, IEEE Trans. Computers, pp. 138-148, Jan., 1988.

[12] A. Kuehlmann, D. I. Cheng, A. Srinivasan and D. P. LaPotin, *Error Diagnosis for Transistor-level Verification*, Proc. Design Automation Conf., Anaheim, CA, pp. 466-471, Sept. 1992.