Mechanical Mathematical Methods for Microprocessor Verification

Warren A. Hunt, Jr.

Department of Computer Sciences
1 University Station, M/S C0500
The University of Texas
Austin, TX 78712-0233, USA
hunt@cs.utexas.edu

Abstract. The functional verification of microprocessor designs continues to represent one of the difficult challenges confronting the design of commercial microprocessors. In addition, test logic, transient errors, and power considerations complicate the problems by creating additional complexity and constraints on design solutions. Rigorious mechanized mathematics, often called formal methods, are being used to assist with functional verification and its use has spread to ensuring that test coverage and power limitations are met. While the successes have been notable, the wide-spread use of mathematical methods is still limited. Here we give a brief introduction to formal microprocessor verification, and then we present some scientific and engineering issues that need addressing to bring formal methods into the mainstream.

1 Introduction

The specification and formal verification of microprocessors represents an evolving science and a difficult to reach goal. There have been many efforts to verify abstractions of microprocessor designs, each with its own specific abstractions and detail. A number of related approaches have been reported, each with a slightly different twist, but in spirit similar. We discuss how we have partitioned the verification of a microprocessor, and make some remarks about why partitioning seems essential. This discussion will transform into a vision that an integrated capabilities a general-purpose verification system should possess. Finally, we present some research problems we believe need addressing if mechanical mathematical formal methods are to become mainstream desigin tools for industrial-sized designs.

We discuss several approaches that are typical for specifying the correctness of microprocessors. Even though many successful proofs have been done, there is no general agreement about what a suitable correctness statement is for microprocessors is, especially super-scalar, pipelined microprocessor designs that include memory delays, exceptions, memory management, and external interrupts. We discuss why the correctness specification of a processor possessing external interrupts can be particularly vexing. Instead of attempting to summarize all of the outstanding work that has been done, we refer the interested

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 523–533, 2004.

reader to Aagaard, et. al., paper *A Framework for Microprocessor Correctness Statements* [1], where an extensive reference list can be found.

To illustrate the kinds of integrated capabilities that a general-purpose verification system should have we describe a hypothetical verification tool **FMaAT** (Formal Modeling and Analysis Tool). Our description of **FMaAT** includes an integrated environment of language, database, checkers, provers, and autonomic regression and analysis routines. From our perspective, it is key that all of the design information and design meta data be represented in a unified database so that all manner of analysis can operate on the model of the design.

The excellent work performed by the formal verification community has shown the possibility of verifying microprocessor models, but we are far away from being able to apply our mechanical tools to complete industrial designs. Of course, we use mechanical tools all the time, and some, for instance, Boolean equivalence checkers, are regularly used on very large parts of modern designs. But we still seem far away from being able to process a complete high-level formal description of a microprocessor which includes all of its associated safety and liveness properties properties and (bi-)simulation relations that ultimately need to be checked on the actual RTL design description or an equivalent abstraction of the transistor-level design. If mechanical tools based on formal methods are to be used broadly, then it must be possible to compile and build a model of all specification levels.

We summarize a correctness statement that is sufficient to specify the correctness of a processor with exceptions, supervisor/user modes, memory delays, branch prediction and external interrupts. We present this specification in Section 2, and we discuss hierarchical verification in Section 3. We present a snapshot of what kinds of information that should be included with a design specification, and we argue in Section 5 that complete design data be available to all tools for manipulating and analyzing designs. We postulate what characteristics the **FMaAT** system should include to permit the wide-spread use of formal verification techniques in the design process. In Section 6 present some engineering obstacles that need addressing to ensure that formal analysis techniques are suitable for general use.

We note that this paper was written to provide some background for a microprocessor verification tutorial given by the author at this conference. Therefore, this paper is written in a conversational style, and it is provided as a partial record of what was presented and as a basis for further discussion. This paper was written not to explore a single topic in depth, but to present a vision for microprocessor verification techniques and tools. We conclude by describing areas of research we think need attention if formal mathematical verification techniques are to become mainstream.

2 Correctness Diagrams

The specification of correctness of a microprocessor can itself be subtle. This is obviously critical as it does no good to prove a vacuous or flawed theorem. In fact, there isn't general agreement about what suitable correctness is. Instead of

attempting to justify a particular approach, we present several correspondence diagrams and conclude why we chose an approach we took.

Consider the correctness diagram in Figure 1. This diagram is meant to indicate that a micro-architecture (MA) design requires some number of steps to execute one ISA step; this is a slight generalization of Burch and Dill's approach [5] as it do not restrict the MA to only a single step. This was the correctness diagram was used to state the correctness of the FM8501 microprocessor [8]. This diagram, together with induction, can be used to show the correctness of any finite sequence of instructions. Such relationships between single and multi-step sequences has analyzed with the Microbox Framework [2].

The diagram in Figure 1 shows the ISA state being a proper subset of the MA state, and the correctness statement indicates that the MA state can be projected into a corresponding ISA state at certain points. The arrow moving from MA states to the ISA states are projection (abstraction) functions. In non-pipelined microprocessor implementations, these abstraction functions are simple projections but in the case of pipelines microprocessors, the MA implementation is often used to flush (or retire) in-flight instructions so a simple projection function can be used.

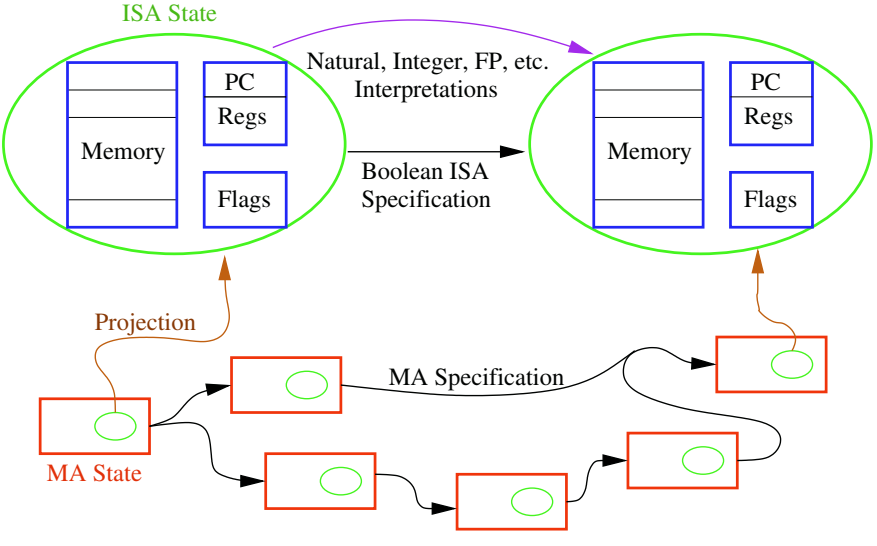

Fig. 1. Simple Microprocessor Correctness Diagram

Figure 1 is not general enough to verify processors that have non-deterministic external interrupts. What is the problem? In a modern microprocessor, (almost) all in-flight instructions are immediately flushed to keep the interrupt latency small, and to prevent a subsequent instruction from creating yet another exception. Consider an external interrupt event interrupting the first MA transition in the sequence of MA transitions. Normally, flushing from a particular state would permit the in-flight instructions to complete, but with an interrupt

most, if not all, in-flight instructions are flushed without completion. The typical Birch-Dill flushing process lets all in-flight instructions complete before a projection is performed, but with an external interrupt in-flight instructions that will normally complete from an earlier MA state will be flushed. If we then project the corresponding ISA state from the MA state just after an interrupt, this may actually produce a state which, in some sense, is earlier in time than the state produced by letting the MA implementation only finish in-flight instructions.

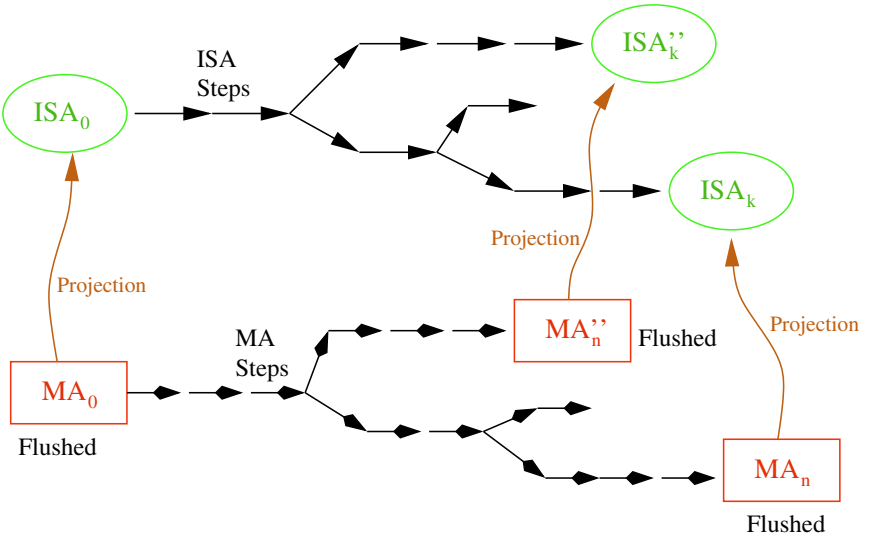

Fig. 2. Super-scalar Microprocessor Correctness Diagram with Interrupts

To provide for asynchronous external interrupts in the correctness statement, it is necessary for the ISA specification to accept external interrupts as well. That is, the different execution paths that can be taken by the MA-level machine must also be possible at the ISA level; otherwise, the ISA specification cannot be kept in correspondence with the MA-level implementation. Such a generalized correctness statement is pictured in Figure 2, where bifurcations represent possible execution path changes due to interrupts. Of course, the design of MA-level flushing mechanism must operate in such a manner that the diagram can be used. This diagram was the correctness diagram for the FM9801 microprocessor verification [11, 15–17]; the flushing mechanism in the FM9801 essentially "throws away" partially completed work, so it can quickly respond to external interrupts, and this mechanism prevents the problem mentioned above. Even though we believe we used correctness diagram in a sound manner, Manolios showed that it is possible to satisfy this type of diagram with trivial, wrong implementations [12]. Correctness statements can be very subtle.

In a companion paper [14], we describe another correctness diagram suitable for microprocessors with sophisticated features and external interrupts. All these correctness statements are designed to be hierarchical. We can construct another

commuting diagram on top of these correctness statements, building a stack of verified layers [3]. In the next section, we discuss breaking down an individual commuting diagram proof into pieces.

3 Compositional Verification

The internal designs of modern microprocessors are very complex. In fact, the designs are so complex that attempting to just use symbolic simulation to satisfy one of the correctness specifications given in the previous section will not work. Instead a proof has to be broken down into smaller pieces so that can be later composed to produce a complete proof of the desired result.

There are many ways that a proof can be broken down, but whether the sublemmas are either single-step properties (or invariants) or multi-step properties certainly changes the ease with which the sublemmas can be composed. Single-step properties are easy to compose – so long as state-space restrictions are met. Multi-step properties are often much more difficult to compose because their environmental requirements and also their (possibly multi-step) conclusions may be very difficult to "stitch" together. Why? The assumptions (usage environments) can be very complex, and it may be difficult or impossible to satisfy both sets of assumptions simultaneously.

Composing two or more individual multi-step properties, or other multi-step invariants, into a single lemma may also produce a conclusion that itself is so specialized that is may difficult to state or prove. For instance, consider composing a four or more step property with a two-step property. Just how should the environments be aligned so the lemmas can be composed? Should the conclusion be about two steps or four steps? With single-step invariants, this is a much easier task as only environmental restrictions on the (reachable) states need to be considered when composing results. We used such single-step invariants to prove the correctness of the FM8501, FM8502, FM9001, Motorola's Complex Arithmetic Processor DSP, and the FM9801 microrprocessors [8, 9, 4, 10].

Industry has equiped itself with several kinds of FSM exploration tools, most notably model checkers and (G)STE engines. These tools have been used to great advantage, exploring various particular design questions in great depth. However, these tools can only be practically applied to subsets of modern microprocessor designs. Generally, the results from these tools provide multi-step property verifications, and as such, are very difficult to compose. This is a critical issue. When automatic FSM exploration tools are available, they can be put to profitable use on many parts of a design, but as the designs grow, and the number of different properties grows, the re-assembly process become extremely complicated.

We believe, if properties verified with FSM exploration tools are to be composed, then either a high-level theorem prover should be used to decompose the proof obligations into pieces that can be checked in a manner similar to McMillan's [13] approach or FSM property tools should be used to prove one-step invariant properties. In this way, the discoveries made with FSM exploration may be safely composed.

Even in light of the difficulties of composing results derives from various FSM exploration tools and checkers, we note that these tools have discoverd many design flaws and checked many complex properties. We have no doubt of the importance of FSM exploration tools as they are used today. These tools are productive and they will continue to be important, and the use of these tools has help defined what the state-of-the-art-of-the-practice is.

4 Current Practice

The state-of-the-art-of-the-practice of the use of formal mathematical methods with mechanically-implemented implementations varies from design to design. My impressions with the industrial use of formal methods comes from time I spent working for IBM Research in Austin, Texas, but through conversations with my colleagues at other companies, my experiences seem typical.

The application of formal methods to design projects varies widely, for many practical reasons: size and importance of project, available tools and people, confidence of architects and managers in the available technology, degree of integration of formal methods tools in the tool flow, size of the company, and duration of the project. The use of formal methods is primarily centered in a dozen or so of the largest companies, probably due to the cost of creating a critical mass of infrastructure (meaning tools, people, and design practice). Even in the larger companies, there are great differences in the degree to which such techniques are deployed. Many smaller projects did not use formal methods, while in larger design efforts, there may be a dozen or more people.

From the project management point of view, the use of formal methods represents a vexing challenge, due to the lack of available metrics to know when its use is efficient and sufficient. This is not so different from simulation, but over many years management has developed "coverage" metrics that help them gauge when there has been "enough" simulation. For complete designs, such intuition has not been developed for mechanical mathematical methods. We are optimistic that coverage metrics can be defined because with a proof, it is very clear what has been proved and what has been assumed.

5 Challenges

We see a number of challenges to improve mechanical mathematics tools so that they are regularly used on commercial-sized designs. First, we recognize that such tools are already being deployed, and are regularly being used to examine large parts of modern designs. For instance, equivalence checkers are now being regularly being used to ensure that low-level gate and transistor-level design specifications implement Boolean RTL micro-architectural descriptions. In addition, model checking [6] and (G)STE [18] are being successfully to validate various design elements. Theorem proving systems have been applied in niche areas, such as floating-point algorithms, and they may provide the "glue" to

bind together different tools. Even so, we don't think of formal tools being part of the "model build" that is often done each evening during the design process.

After developing and using mathematical modeling and analysis tools in a commercial environment, we have identified several challenges that we believe need to be addressed before mathematical analysis will regularly occur at the higher levels of the design hierarchy. So, if we were to build a system called **FMaAT** (Formal Modeling and Analysis Tool), we would like it to have a number of properties.

- **FMaAT** needs to be able to read, compile, and "model build" the entire design specification. **FMaAT** should be able to read the entire design, and represent such a design as a formal object. Engineers do not trust tools that cannot manipulate the actual design specification. If one were to print the complete RTL description for a modern microprocessor, it may well require 30,000 or more pages.
- The **FMaAT** system must, in all respects, operate in a hierarchical manner. Every design is yet just another piece of an even larger design in the future.
- **FMaAT** needs to contain all embedded annotations. That is, if a module has a requirement that its inputs are one-hot and that its outputs are active-low, then this data must be included in the original design specification and it must also have a representation as a formal object that **FMaAT** can inspect, manipulate, and subject to analysis.
- **FMaAT** needs to be able to act as a database engine that allows every design module and interface to be identified. It must be possible to uniquely identify every primitive element, interface, and wire. A completely precise and unique naming convention is a requirement.
- Each time a change is made to the design, the effects of the change should be automatically propagated to the **FMaAT** database so when an analysis is requested, only the relevant parts are subjected to analysis.
- **FMaAT** needs to be able to compute cones-of-influence, bus conflicts, improper connections, and other user-definable queries.
- **FMaAT** must have a command-line interface. In a big design project, tools are always "taped" together with scripting languages to overcome deficiencies in the design flow. **FMaAT**'s command-line interface should itself be described formally.
- There has to be a way to re-run all checkers, simulators, etc., automatically whenever there is any change to the design. Automatic regression verification is a must.
- There must not be any way to get a false positive. There must be provisions for ensure vacuity checking for analysis requests.
- If possible, **FMaAT** should have some kind of analysis "coverage metrics". If included, then a formal definition of coverage should also be included, so that some kind of qualitative assessment can be made as to the thoroughness of an analysis.

- Along with having all of the design and associated property specifications directly available in a single database, it is critical that there be a semantics that allows the various tools (and the results derived from these tools) to be safely composed. This can only be done if **FMaAT** contains a general-purpose theorem prover.
- **FMaAT** must provide a means to write a truly rigorous high-level specification. System-C and System Verilog are not a long-term answer; in fact, these languages are creating yet more problems.
- A purely functional verification system is not sufficient. **FMaAT** must a way to specify non-functional properties such as power requirements, circuit sizes, wire types, physical location data, environmental requirements, and other critical design properties. And for each such property, suitable checkers and verifiers will need to be provided.
- **FMaAT** must deal with a distributed design process. No project of a significant size is all done in a single place.
- Finally, **FMaAT** must safely extensible; that is **FMaAT** should be no more difficult to extend than Emacs, but **FMaAT** should impose a discipline that ensures that extensions do not render existing checker and verifiers unsound.

A tool like **FMaAT** will require a much more general language than those commercially available, such as Verilog and VHDL and their derivatives. The limitations of the available languages are causing the specification problem to actually become worse because designers are forced to record their design properties as comments or in external files. The available existing design languages do not have associated specification languages. A community-wide effort has resulted in the Accellera [7] standardized property specification language, but even this language does not have a formal relationship to the systems (e.g., designs coded in Verilog or VHDL) it is meant to specify.

Future system design languages need to have fully integrated specification languages and fully integrated annotation languages. In this way, the analysis tools (checkers, simulators, theorem provers) can all get access to any or all of the design artifact, thus providing a unifying framework for designs and their specifications. And all such analysis tools must be defined using the same semantic foundation so results from one analysis tools can be immediately used by other analysis tools.

6 Research Problems

There are many technical and engineering challenges that remain before mechanical formal mathematical methods become fully integrated into the commercial design flow. We discuss these obstacles with the hope that our community will help solve these problems.

To make a system like **FMaAT** will require fundamental changes to the infrastructure of commercial design environment. With careful planning and execution, it should be possible to incrementally improve commercial design tools

so that their foundation is suitable to allow the wide-spread use of mechanical formal methods.

- New formally specified design and annotation languages need to be defined that provide a semantically unified framework for designs and all associated specifications. These languages need to include mechanisms to represent all of the design "meta" data directly as a part of the design specification. For instance, that some inputs are "one-hot" and some outputs are "active-low" needs to be captured just as some safety or liveness property. All of this data needs to be expressible in this language. Such a design language also needs to be general enough to express non-functional properties such as area and power constraints. These languages must be hierarchical.
- To reduce power consumption, there is going to be a greater use of asynchronous circuit elements along with circuits that can trade execution speed with power requirements. Specification and analysis of mixed circuit types will be necessary, and we need to develop modeling and verification techniques capable of supporting designs with a mixture of digital, asynchronous, and analogue circuits.
- Typical two- and four-valued simulators need to be extended to symbolic simulator; that is, there should be a single simulation environment general enough to perform simulation with a mixed set of constants and symbolic variables. Moving from one simulation environment to another is error prone and confusing.
- All of the analysis tools (e.g., equivalence and model checkers, (G)STE engines, reachability analysis, theorem provers) should all read the same design data and all follow the same semantics. In other words, we must achieve integration between the various analysis tools so results from one tool can be used elsewhere. Implementations of these tools have chosen their own logics; we need some kind of unification so results can be shared and reused among the various tools.
- A new suite of non-functional checkers (e.g., for power, area, redundancy management) need to be developed. These checkers should receive the same level of rigor and development as existing formal analysis tools.
- Post-silicon design approaches need to be integrated into the design process. Post-silicon debugging tools (e.g., logic analyzers) have not improved much in the last decade and the amount of visibility continues to decrease as implementations become more and more integrated. Future formal design languages must be general enough to also permit the specification of the supporting chip sets and the systems themselves.
- Autonomic systems that automatically (re-)run all checkers and provers should be automatically started any time any part of a design is changed. This is necessary as no large design effort is now done in a single location. This system, if you will, is a *super-CVS*, ensuring that all design properties are pro-actively analyzed.
- Formal approaches to (microprocessor) security need to be developed. Future processors will be systems-on-a-chip, and the specification of security features and analysis of security properties is going to be critical.

These are some of the issues that need solving to broaden the impact of mathematics on the design of microprocessors, and computing systems in general. A sustained, long-term effort will be required to extend the state-of-the-art-of-the-practice.

7 Conclusion

The functional verification of microprocessor-sized designs will continue push the state-of-the-art and the state-of-the-practice of mathematical formal methods. As the complexity and sheer number of microprocessors continues to increase, we see no practical alternative to the use of formal mathematics supported by mechanical reasoning tools. Mathematics is the only technique that can scale with the ever increasing size and complexity, and mechanized mathematical specification and proof are the only practical infrastructure for correct, reliable, and secure microprocessor.

We would like computing systems to be specified by a *formula manual*, a complete precise set of formulas that exactly specifies computing systems (whether hardware, software, or both). We want mathematically specified, mechanically checked computing systems. Systems are increasing in complexity faster than our ability to manage them or control them. If we are aggressive, maybe we can achieve this vision on small commercial designs, e.g, cell telephones, pagers, routers, etc. Our ability to field secure, correct systems is based on our ability to specify and validate our computing, networking, and control systems.

The use of mathematical formal methods will continue to broaden. It is the most economical method to ensure correctness, reliability, power usage, and security, of future designs. No other analytical techniques known to us will be able to scale with future design requirements. We are impressed with the progress and we look forward the challenge of extending the use of mathematics for design.

References

1. M. Aagaard, B. Cook, N. Day, and R. Jones. A Framework for Microprocessor Correctness Statements. In *CHARME 2001*, LNCS 2144, pages 433-448, Springer Verlag, 2001.
2. M. Aagaard, N. Day, and M. Lou. Relating Multi-step and Single-Step Microprocessor Correctness Statements. In *Formal Methods in CAD, FMCAD 2002*, LNCS 2517, pages 123-141, Springer Verlag, 2002.
3. W. R. Bevier, W. A. Hunt, J S. Moore and W. D. Young. An Approach to Systems Verification. In *Journal of Automated Reasoning*, Volume 5, November, 1989.
4. B. C. Brock and W Hunt, Jr. Formal Analysis of the Motorola CAP DSP. In *Industrial-Strength Formal Methods*, edited by Mike Hinchey and Jonathan Bowen, Springer-Verlag, 1999.
5. J. R. Burch and D. L. Dill. Automatic Verification of Pipelined Microprocessor Control. In *Computer Aided Verification, CAV '94*, LNCS 818, pages 68-80, Springer Verlag, 1994.
6. E. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Press, 1999.

7. M. Gordon, J. Hurd, and K. Slind. Executing the Formal Semantics of the Accellera Property Specification Language by Mechanized Theorem Proving. In *CHARME 2003*, LNCS 2860, pages 200-215, Springer Verlag, 2003.
8. W. Hunt, Jr. FM8501: A Verified Microprocessor, LNAI Number 795, Springer-Verlag, 1994.
9. W. Hunt, Jr. and B. Brock. A Formal HDL and Its Use in the FM9001 Verification. In C.A.R. Hoare and M.J.C. Gordon, editors, Mechanized Reasoning and Hardware Design, pages 35-48, Prentice-Hall International Series in Computer Science, Engle wood Cliffs, N.J., 1992.
10. W. Hunt, Jr. and J. Sawada. Verifying the FM9801 Microarchitecture. In *IEEE Micro*, IEEE Press, pages 47–55, May-June, 1999.
11. M. Kaufmann and J S. Moore. ACL2: An Industrial Strength Version of Nqthm. Proceedings of the *Eleventh Annual Conference on Computer Assurance (COMPASS-96)*, pages 23-34, IEEE Computer Society Press, June 1996.
12. P. Manolios. Correctness of Pipelined Machines. In *Formal Methods in Computer-Aided Design, FMCAD 2000*, LNCS 1954, pages 161-178, Springer-Verlag, 2000.
13. K. L. McMillan. A Methodology for Hardware Verification Using Compositional Model Checking. In the *Science of Computer Programming*, Volume 37, Number 1-3, pages 279-309, 2000.
14. S. Ray and W. A. Hunt, Jr. Deductive Verification of Pipelined Machines Using First-Order Quantification. *Computer-Aided Verification, CAV 2004*, LNCS 3114, Springer Verlag, 2004.
15. J. Sawada and W. Hunt, Jr. Trace Table Based Approach for Pipelined Microprocessor Verification. *Computer-Aided Verification, CAV'97*, LNCS 1254, pages 364-375, Springer Verlag, 1997.
16. J. Sawada and W. Hunt, Jr. Processor Verification with Precise Exceptions and Speculative Execution. *Computer Aided Verification, CAV'98*, LNCS 1427, pages 135-146, Springer Verlag, 1998.
17. J. Sawada and W. Hunt, Jr. Verification of the FM9801 Microprocessor: An Out-of-order Microprocessor Model with Speculative Execution, Exceptions, and Self-Modifying Code. In *Formal Methods in Systems Design*, Kluwer Academic Publishers, Volume 20, Number 2, pages 187–222, March, 2002.
18. J. Yang and C. Seger. Generalized Symbolic Trajectory Evaluation — Abstraction in Action. In *Formal Methods in CAD, FMCAD 2002*, LNCS 2517, pages 70-87, Springer Verlag, 2002.