Challenges in the Formal Verification of Complete State-of-the-Art Processors

Nathaniel Ayewah, Nikhil Kikkeri and Peter-Michael Seidel
Southern Methodist University, Computer Science and Engineering, Dallas, TX 75205
{ayewah,nikhil,seidel}@engr.smu.edu

Sven Beyer*
OneSpin Solutions GmbH, Munich, Germany
Sven.Beyer@onespin-solutions.com

Abstract

Research on formal hardware verification has made steady progress in developing methodologies and tools that try to cope with the growing complexities of systems. Despite of case studies that demonstrate the applicability of formal methods to selected contemporary processor designs, the current state in formal hardware verification is far from being considered practical for systems of the complexity of complete contemporary processor designs.

It is our goal to improve the practicality of current formal verification methods for complete state-of-the-art processor designs. The recent success in the complete formal verification of the VAMP can be considered pioneering for reaching design complexities close to this range. We dissect the VAMP verification effort in detail with the goal to identify the main technical and organizational challenges and the major productivity bottlenecks of the verification process. This is done in particular to search for opportunities of increased levels of automation. As part of our efforts we are developing the VAMPExplorer, a tool that provides an intuitive interface to the specification, the implementation and the verification of the VAMP. The VAMPExplorer visualizes the general implementation and verification structure and improves accessibility to expert and non-expert users.

1. Introduction

Most current digital systems are far too complex to guarantee their correct design just by non-formal verification techniques such as exhaustive simulation. Formal verification techniques can, in principle, offer better scalability of the verification effort with the circuit size and it seems that recent advances in formal verification methodologies and tools have enabled the applicability to more practical design options. Various projects have been successful in showing the correctness of aspects of processor designs using formal techniques (e.g. [6,8,13,20,23]). However, most studies focus on high-level descriptions or on selected subcomponents of a system and make idealistic assumptions about some parts of the design. Details of the implementation are usually ignored and hardly any project design is synthesized or fully implemented. Formal verification efforts are also increasingly finding their way into commercial design efforts (e.g. [1, 7, 14, 16, 18]). But even if in these cases the corresponding designs have an implementation, it is not usually the implementation that is formally verified, but it is some high-level description or only parts of the design. A full consideration of all system levels from specification to a synthesizable implementation description is necessary to consider it the complete verification of a processor. The authors of [12] have been generalizing the limitations of current verification techniques by stating that the complete functional verification of microprocessor designs could not even be achieved. Although this statement does not hold in this generality as shown by the recent success in the complete verification effort of the VAMP [2, 3], it is generally assumed that complete formal verification is still far from being considered practical for designs of the complexity of contemporary microprocessors.

The complete formal verification of the VAMP can be considered pioneering for reaching design complexities close to this range. The VAMP has been verified using PVS [17] with a high degree of interactive intervention. A premier motivation in our analysis and presentation of the VAMP verification effort is the search for opportunities of increased levels of automation and the increased awareness and communication about verified open source reference designs and challenges in complete processor verification.

*Work funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft [22] project under grant 01 IS C38 while the author was with Saarland University, Saarbrücken, Germany

As part of this effort we are developing the VAMPExplorer, a tool that provides an intuitive interface to the specification, the implementation and the verification of the VAMP. The VAMPExplorer visualizes the general implementation and verification structure and simplifies quick location and comparison of code fragments for improved accessibility.

In Section 2 we provide a description of the implementation and proof of the VAMP. In Section 3 we overview alternative processor verification efforts and discuss their restrictions. In section 4 we dissect the challenges in the VAMP verification effort. In section 5 we give a short overview of our efforts in developing the VAMPExplorer, before we finally conclude in section 6.

2. Description of the VAMP

For a detailed description of the VAMP, we refer to [2,3]. We think that the VAMP project [4] can be considered a reference project for processor verification for several reasons.

With the help of the theorem-prover PVS [17], the VAMP is formally verified on the gate level, i.e., the verified PVS design is automatically translated to Verilog HDL for synthesis on an FPGA. The complete sources of both design and proofs are available online at the project website [4]. The VAMP offers a full DLX instruction set [15] with about 100 instructions including floating point extensions. The implementation contains a Tomasulo scheduler with reorder buffer supporting multiple results, issuing into the reorder buffer, and reordering of instructions in execution units.

Design and verification are structured hierarchically resulting in many verified modules. This means that the correctness criteria of the modules are not only what one thinks they should be, but they *formally* are what is needed for the complete verification of the VAMP. Correctness criteria for modules that almost fit together are of no use. Just consider the difference between i) an aeroplane with yourself on board crashing due to some interplay of its modules that was not foreseen by its engineers because it was not modeled formally and ii) you safely landing at your desired holiday resort.

The VAMP is not intended as a prototype that demonstrates feasibility of new verification techniques. On the contrary, the VAMP project "just" used slight refinements of existing techniques and applied them in an engineering process to the formal verification of a complete microprocessor. We therefore suggest using modules of the VAMP as benchmark suites for new verification methodologies instead of some purely academic examples.

In the VAMP project, almost 90000 interactive proof steps were carried out for about 2500 lemmas; the overall design and verification effort was about eight man years.

The generated Verilog code of the VAMP contains 866 modules in more than 100000 lines of code. The PVS sources of the design alone have roughly 10000 lines of code. The VAMP contains more than 8800 register bits. The Xilinx software reports an equivalent gate count of about 1.5 million. Note that this number also covers the instruction and data cache, i.e., 24 KB.With a cost of 4 gates per memory bit, this results in about 0.8 million gates for memories, i.e., slightly more than half the overall gate count. As a comparison, an Intel Pentium of the first generation has a gate count of about 3.1 million including 16 KB of caches, i.e., 0.5 million memory gates.

The overall correctness proof of the VAMP is structured hierarchically. At the lowest level, there is the gate-level implementation of the VAMP. About 5% of the overall proof effort were spent on this step.

As a next step, complete execution units are verified and abstracted from with uninterpreted functions. Carrying out this module verification and replacing the execution units by their specification function leads to an abstract implementation of the overall processor with uninterpreted functions as execution units. More than 60% of the proof effort was spent for the complex execution units.

The verification of a parameterized Tomasulo algorithm cost about 15% of the verification effort. In the next step, the actual data management of the pipeline is verified, i.e., the abstract pipeline implementation is mapped to a programmer's model without interrupts which carries out one instruction per step which was just as complex as the verification of the Tomasulo algorithm. In the final step with about 3% of effort, interrupt support is added.

3. Other Processor Verification Projects

Most of the features of the VAMP have also been covered by other verification projects; however, these projects usually focus on single modules, make heavy restrictions, or use strong abstractions.

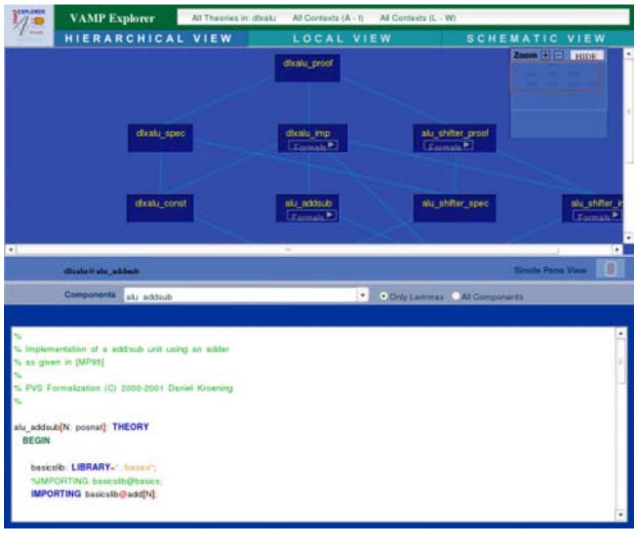
Many formal verification projects focus on Microprocessors with in-order scheduling, one or several pipelines including result forwarding, stalling, and interrupt mechanisms [6, 25]. The verification of the very simple, non-pipelined FM9001 processor is reported in [5]. Using the flushing method from [6] and uninterpreted functions for modeling functional units, superscalar processors with multicycle execution units, exceptions and branch prediction [25] have been verified by automatic BDD based methods.

The formal verification of Tomasulo schedulers with reorder buffers for the support of precise interrupts [8, 10, 13, 19] is more complex. Using theorem proving, Sawada and Hunt [11, 19, 20] show the correctness of an entire out-of-order processor, precise interrupts, and a store buffer for the memory unit. They also consider self-modifying code.

Except for the work on the FM9001 processor [5], *none* of the papers quoted above states that the verified design has been implemented. *All* results cited above use several simplifications and abstractions. The realized instruction set is restricted to 11 instructions [19] or even only six [6].

Sometimes, non-implementable constructs are used in the processors: e.g., Hosabettu et.al. [10] use tags from an infinite set. The highly automated Tomasulo proof of McMillan [13] neither covers multiple results sharing the same tag, specialized execution units or reservation stations nor direct issuing into the reorder buffer.

The verification of pipelines or Tomasulo schedulers with *instantiated* execution units has not been reported apart from the VAMP project [2, 3]. Indeed, in [24] the authors state: "An area of future work will be to prove that the correctness of an abstract term-level model implies the correctness of the original bit-level design."



```
%
% Implementation of a add/sub unit using an adder
% as given in [MP95]
%
% PVS Formalization (C) 2000-2001 Daniel Kroening
%

alu_addsub[N: posnat]: THEORY
  BEGIN

    basicslib: LIBRARY="../basics";
    %IMPORTING basicslib@basics;
    IMPORTING basicslib@add[N];
```

Figure 1. Screen-shot of the VAMPExplorer

4. Challenges in the Verification of the VAMP

The aim of the VAMP project was the design and verification of a complete state-of-the-art microprocessor without any abstractions or restrictions. Most of the difficulties faced in the course of the project stemmed from this fact.

Intense cooperation between the project members was needed in oder to define interfaces between the modules that not only looked good, but would actually allow for an efficient design and their formal integration into the overall correctness proof. Different modules were then designed and verified in parallel. Since synthesis of the design had been one of the initial project objectives, all these modules had to be verified down to the gate level.

When all the different modules were integrated to the VAMP proof, the full DLX instruction set with about 100 instructions was realized. Finally, the VAMP project also carried PVS to its limits concerning complexity. Type checking of the overall proof alone took two hours of CPU time in 2003; actually rerunning all the proofs several days.

The VAMP project basically employed purely interactive proofs in PVS with little use of strategies. However, 90000 interactive proof steps are obviously too much for the VAMP project. We believe that with integration of the right automated tools, at least three quarters of the interactive proof steps will become obsolete since they seem trivial. Only with this gain in productivity, the verification of entire systems will become feasible, e.g., the combination of hardware and system software needed in order to provide virtual memory to user processes [9]. We therefore provide selected challenges for the automated verification community at the VAMPExplorer website [21].

The VAMP design is cleanly modularized into the core, the execution units, and the memory system. Therefore, it is possible to plug in a different memory system or more optimized execution units by only verifying their local correctness and combining it with the remaining part of the original VAMP correctness proof. In addition, the memory system and its proof is parameterized, e.g., on the associativity and size of the caches. Both the modularization and the parameterization make the VAMP proof quite generic.

Since the proof is modularized, it is possible to switch the VAMP core with some other scheduling algorithm. Note, however, that this would also require the proof of a gate-level *implementation* of the scheduling algorithm.

The VAMP uses the delayed PC scheme for instruction fetch. Integrating branch prediction instead would require a considerable effort on the core proof since the correctness of instruction cancellation is proved for interrupts only as the very last step of the overall proof. Cancellation due to misprediction, however, happens frequently even without interrupts, i.e., at the lowest level of the proof.

While the memory system is parameterized, there are some extensions that would require a large effort. Consider, e.g., non-blocking caches. On the other hand, for the addition of store buffers, the existing memory system could just be re-used; some additional effort would only be needed to show the correctness of forwarding from the store buffers.

As long as design optimizations keep the overall structure of the VAMP, an adaption of the proof is fairly easy to carry out. However, re-balancing the design by introducing an additional pipeline stage for timing reasons or power-saving optimizations may require a huge proof effort.

5. The VAMPExplorer

The complexity of the VAMP project makes it difficult to identify structural information or to navigate through specific details. The goal is to establish the open source

VAMP descriptions as a reference design for complete processor verification to a broad audience and encourage modular verification of selected properties/components with alternative, preferably automated, verification techniques. We have a focus on providing an intuitive graphical interface for identifying and navigating through VAMP theories and their relations while allowing concurrent access and summary of different aspects (specification, PVS implementation, synthesized Verilog Implementation, properties and related proofs) of a single component.

A high degree of automation is involved in the data and property extraction from the PVS files and the preparation of the dynamic hierarchical structural views. We chose to use Macromedia Flash as our web front-end because of its broad web accessibility. The VAMPExplorer is available online at [21].

6. Conclusion

It is our goal to improve the practicality of complete formal verification for complex state-of-the-art processors. For this purpose we discuss the details and and the generalization of challenges that occur in the complete formal verification effort of the VAMP as one of the pioneering examples in complete formal processor verification. We suggest the use of the VAMP sources as a reference design and benchmark suite for the (automatic) formal verification of components as they are integrated in a complex modular processor design. For improved accessibility we develop the VAMP-Explorer GUI for an intuitive access to the VAMP sources. Future work includes extensions of the VAMPExplorer to enhanced automatically generated schematic views and the handling of even more complex proof structures [22].

References

[1] S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal. Model Checking at IBM. In *Formal Methods in Systems Design*, pages 101–108. Kluwer Academic Publishers, 2003.

[2] S. Beyer. *Putting it all together – Formal Verification of the VAMP*. PhD thesis, Saarland University, Germany, 2005.

[3] S. Beyer, C. Jacobi, D. Kröning, D. Leinenbach, and W. Paul. Instantiating uninterpreted functional units and memory system: functional verification of the VAMP. In *CHARME*, volume 2860 of *LNCS*. Springer, 2003.

[4] S. Beyer, C. Jacobi, D. Leinenbach, and W. J. Paul. The VAMP project. Website, 2003. http://www-wjp.cs.uni-sb.de/projects/verification.

[5] B. C. Brock and W. A. Hunt. The DUAL-EVAL hardware description language and its use in the formal specification and verification of the FM9001 microprocessor. *Formal Methods in System Design*, 11:71–107, July 1997.

[6] J. Burch and D. Dill. Automatic verification of pipelined microprocessors control. In *CAV*, volume 818 of *LNCS*, pages 68–80. Springer, 1994.

[7] M. Cornea-Hasegan. IA-64 floating point operations and the IEEE standard for binary floating-point arithmetic. *Intel Technology Journal*, Q4, 1999.

[8] W. Damm and A. Pnueli. Verifying out-of-order executions. In *CHARME*, pages 23–47. Chapman & Hall, 1997.

[9] M. Hillebrand. *Address Spaces and Virtual Memory: Specification, Implementation, and Correctnesss*. PhD thesis, Saarland University, Germany, 2005.

[10] R. Hosabettu, M. Srivas, and G. Gopalakrishnan. Proof of correctness of a processor with reorder buffer using the completion functions approach. In *CAV*, volume 1633 of *LNCS*, pages 47–59. Springer, 1999.

[11] W. A. Hunt and J. Sawada. Verifying the FM9801 microarchitecture. *IEEE Micro*, pages 47–55, May-June 1999.

[12] S. Mangelsdorf, R. Gratias, R. Blumberg, and R. Bhatia. Functional verification of the HP PA 8000 processor. *Hewlett-Packard Journal*, 1-13, 1997.

[13] K. McMillan. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In *CAV*, volume 1427 of *LNCS*. Springer, 1998.

[14] J. Moore, T. Lynch, and M. Kaufmann. A mechanically checked proof of the AMD5K86 floating point division program. *IEEE Transactions on Computers*, 47(9):913–926, 1998.

[15] S. M. Müller and W. J. Paul. *Computer Architecture. Complexity and Correctness*. Springer, 2000.

[16] J. O'Leary, X. Zhao, R. Gerth, and C.-J. H. Seger. Formally verifying IEEE compliance of floating-point hardware. *Intel Technology Journal*, Q1, 1999.

[17] S. Owre, N. Shankar, and J. M. Rushby. PVS: A prototype verification system. In *CADE 11*, volume 607 of *LNAI*, pages 748–752. Springer, 1992.

[18] D. M. Russinoff. A case study in formal verification of register-transfer logic with ACL2: The floating point adder of the AMD Athlon processor. In *FMCAD*, volume 1954 of *LNCS*. Springer, 2000.

[19] J. Sawada and W. A. Hunt. Processor verification with precise exceptions and speculative execution. In *CAV*, volume 1427 of *LNCS*. Springer, 1998.

[20] J. Sawada and W. A. Hunt. Verification of the FM9801 microprocessor: An out-of-order microprocessor model with speculative execution, exceptions, and self-modifying code. *Formal Methods in Systems Design*, 20(2):187–222, March 2002.

[21] P.-M. Seidel and N. Ayewah. The VAMP explorer. Website, 2005. http://engr.smu.edu/~seidel/VAMPExplorer/.

[22] The Verisoft Consortium. The Verisoft project. Website, 2003. http://www.verisoft.de.

[23] M. Velev. Comparative study of strategies for formal verification of high-level processors. In *ICCD*, pages 119–124, 2004.

[24] M. Velev and R. Bryant. Superscalar processor verification using efficient reductions of the logic of equality with uninterpreted functions to propositional logic. In *CHARME*, volume 1703 of *LNCS*, 1999.

[25] M. Velev and R. Bryant. Formal verification of superscale microprocessors with multicycle functional units, exception, and branch prediction. In *DAC*. ACM, 2000.