# Formal Verification of the VAMP Microprocessor
# Project Status

Christoph Berg*, Sven Beyer, Christian Jacobi,
Daniel Kröning*,**, and Dirk Leinenbach

Saarland University, Computer Science Department
66123 Saarbrücken, Germany
{cb,sbeyer,cj,kroening,dirkl}@cs.uni-sb.de
Tel. +49-681-302-4129, Fax -4290

February 25, 2002

## 1 Introduction

Microprocessors are in use in many safety-critical environments, such as cars or planes. We therefore consider the correctness of such components as a matter of vital importance. Testing microprocessors is limited by the huge state space of modern microprocessors. We therefore think formal verification is the sole way to obtain a correctness guarantee.

At Saarland University, we are currently working on a project aiming to formally verify the correctness of a complete microprocessor called VAMP. The VAMP (Verified Architecture Microprocessor) is a variant of the DLX processor [11]. It features a Tomasulo-scheduled 5-stage pipeline, precise interrupts, delayed branch, virtual memory management, cache memory, and a fully IEEE compliant dual-precision floating point unit that handles denormals and exceptions entirely in hardware. The specification and verification is performed on the gate level using the PVS theorem proving system [25]. Our group has developed a tool which automatically translates hardware specifications from the PVS language to Verilog HDL. This enables us to translate the VAMP to Verilog and synthesize it on a Xilinx FPGA [7].

This paper provides an overview of the VAMP project. We sketch the proof techniques used in the verification of the different VAMP components.

*Related Work.* The designs verified in the VAMP project are based on [9, 23]. The designs and paper-and-pencil proofs in [23] served as guidelines during the formal verification.

Processor scheduler and hardware verification is considered in [2, 3, 12, 13, 19, 27]. Our verification project is the first to prove the correctness of a gate level description of a microprocessor of such high complexity. Other formalizations of the IEEE standard using theorem provers are [10, 20]. Floating point hardware is verified in [1, 8, 21, 24, 26]. In contrast to our verification project, these do not treat denormal numbers and exceptions.

* Work supported by the DFG graduate program "Leistungsgarantien für Rechnersysteme"
** Currently at Carnegie Mellon University

## 2 Tomasulo Scheduler Verification

Out-of-order execution allows for high performance even in case of long latency instructions such as floating point or memory instructions. One of the most popular out-of-order execution algorithm is the Tomasulo scheduling algorithm [28]. It provides CPI rates down to 1.1 on a single-instruction issue machine [22]. The proof is described in detail in [18], we only sketch the proof idea here.

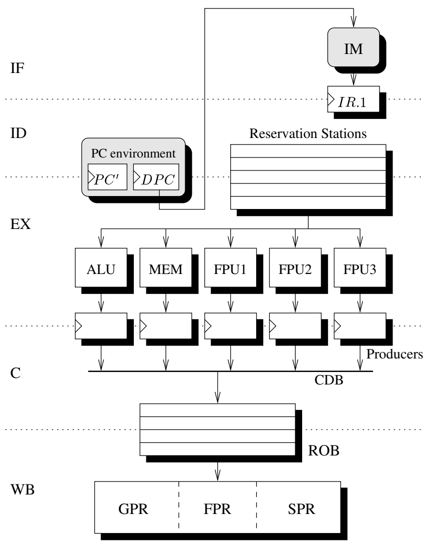


**Fig. 1.** VAMP microprocessor overview

**Data Consistency.** We show the correctness of a DLX implementation with Tomasulo scheduler by defining an abstract machine $aS$ that processes one instruction with each transition. The configuration of this machine consists only of the ISA registers $PC$, register files, and main memory. The proof is split in two parts: 1) We show that a machine implementing the Tomasulo protocol simulates the abstract machine $aS$. This is the hardest part of the proof. 2) We then show that the DLX implementation with Tomasulo scheduler implements the Tomasulo protocol.

**Scheduling Functions.** We need a formal way to state that "instruction $I_i$ is being issued during cycle $T$" or "instruction $I_i$ is being dispatched during cycle $T$". We do this in analogy to [23] using a *scheduling function*.

*Issue.* We recursively define a function $sIissue$ that maps a cycle $T$ to the index $i$ of the instruction $I_i$ that is in the issue stage. Since we issue in program order, that number increases by one in case of an issue and stays unmodified otherwise. We start with instruction $I_0$. Let the predicate $issue(T)$ denote the fact that we issue an instruction during cycle $T$.

$$sIissue(T) := \begin{cases} 0 & : T = 0 \\ sIissue(T-1) + 1 & : issue(T-1) \\ sIissue(T-1) & : \text{otherwise} \end{cases}$$

*Reservation Stations.* We also desire a way to define the instruction in a given reservation station $rs$ during a given cycle $T$. We do this by defining a schedule function $sIRS(rs, T)$ for the reservation stations. Instructions are put in a reservation station during issue. In case an instruction is issued into reservation station $rs$, we take the value of $sIissue(T-1)$. Otherwise, the value of $sIRS(rs, T)$ remains unchanged.

$$sIRS(rs, T) := \begin{cases} 0 & : T = 0 \\ sIissue(T-1) & : issue(T-1) \text{ into reservation station } rs \\ sIRS(rs, T-1) & : \text{otherwise} \end{cases}$$

We define similar functions for the reorder buffer entries, for the FUs and for the producers. Furthermore, we define a function that provides the index of the instruction on the CDB.

**Proving the Forwarding Correct.** The most important part of the data consistency proof is proving the forwarding correct. Forwarding is done by reading from the CDB or from the ROB. Both the ROB and the CDB use tags to identify the results. The first step is to show the following property on the tag that is used to identify the instruction that produces the desired result: the tag is shown to be the tag of the last instruction that wrote the register to be forwarded. The next step is to show the following central claim: Consider two instructions $I_i$ and $I_j$ that are already issued but not terminated yet. Let those instructions have the same tag. Then the instructions are the same. This allows concluding that the data we read is the result of the last instruction writing the desired register.

**Liveness.** We propose the following liveness criterion for the Tomasulo machine with reorder buffer: we will show that all instructions will eventually terminate. We omit the details due to lack of space and refer the reader to [18].

## 3 Memory Management Unit

The VAMP includes a memory management unit (MMU) with virtual memory and parameterized $k$-way data and instruction caches. The cache verification is finished, the MMU will be completed within the next months [6]. We omit the details due to lack of space.

## 4 Floating Point Unit

To exploit the benefits of the Tomasulo scheduler, there are three FPUs in the VAMP processor. We have one FPU that handles addition/subtraction, one that handles multiplication/division, and one for conversion and other instructions. Figure 2 depicts the overall structure of the floating point units. Special cases, e.g., operations on $\pm\infty$, are handled separately and bypass the computation unit, where the actual operation is implemented. The results of the operations are passed to the rounder, where they are rounded to representable IEEE numbers.

The verification of the FPU is split into two independent parts: the correctness of the combinatorial data paths (unpacker, the computation unit implementing the operation to be performed, and the rounder), and the correctness of the pipeline control for the different units. The combinatorial design of the FPU is taken from [23]. The paper-and-pencil proofs in [23] served as guidelines for the formal verification in PVS. We describe the formal verification in detail in [4].

**Combinatorial FPU Data Paths.** The FPU is correct if it obeys the IEEE standard 754 [14]. This standard is an informal description of floating point arithmetic. We therefore have to give a formal version of the standard that can be used for theorem proving. We have formalized the standard in PVS [4, 15]. The formalization of the IEEE standard bases on [9, 20, 23].

For each of the different rounding and precision modes in the standard, we define a function $rd$ that rounds real numbers to values representable within the IEEE floating point format. The correctness criterion we prove using PVS then is: for each operation $\circ \in \{+, -, \times, \div\}$ on numbers $a$ and $b$ (including denormals), the FPU output is $f = rd(a \circ b)$.

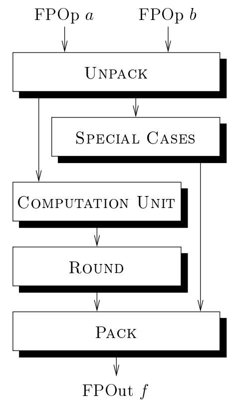

**Fig. 2.** The FPU data paths

To allow the computation units to be implemented independently of the rounding mode, we introduce an equivalence relation $\equiv_\alpha$, where two numbers $x$ and $y$ are equivalent iff they round to the same value [9, 23]. This includes denormal numbers. The computation unit then computes a value that is $\alpha$-equivalent to the precise result which the rounder rounds to the correct output. The concept of $\alpha$-equivalence allows us to reduce significantly the complexity of the verification because computation unit and rounder can be verified independently.

The circuit hierarchy is verified in a bottom-up approach. On the gate level, the smallest modules consist of a few gates. In PVS, the correctness criterion is proved using a combination of induction and automatic case analysis [5]. The modules are combined to form ever larger modules, whose correctness is a consequence of the correctness criteria of the components. At the end, the correctness of the whole unit is achieved with a practicable complexity of the proof structure.

**Pipelining the FPUs.** In order to implement the units with reasonable cycle time, one has to insert pipelining registers. The pipelined units are then incorporated into the Tomasulo CPU. In order to work properly in the Tomasulo framework, the FUs have to obey the following correctness properties:

*Liveness.* Each instruction dispatched into the FU has to complete eventually, and the data is computed correctly. Formally, this means if tag $tg$ is dispatched at time $t$, then the tag $tg$ is output at some later time $t' \geq t$, and the data output at time $t'$ is correct with respect to the data input at time $t$.

*Consistency.* Each tag $tg$ which the FU outputs at some time must have been dispatched into the unit at some time earlier, and the tag $tg$ must not have been returned by the unit in the intermediate time. Consistency depends on the *tag uniqueness* property: the Tomasulo scheduler must not issue an instruction with tag $tg$ that is currently in use, i.e., that has been dispatched but not yet returned. Tag uniqueness is proved during the verification of the Tomasulo scheduling algorithm. Together, liveness and consistency guarantee that for each instruction dispatched into the FU, there exists *exactly* one return, and the data returned is correct.

The FUs may process multiple instructions simultaneously, and instructions need not be returned in the same order they were dispatched. All of our FPUs use these features. For example, operations on special operands (e.g., $\pm\infty$) bypass the computation unit and may leave the FU after only one cycle, while other instructions dispatched before are still processed in the computation unit (see fig. 2).

We have developed a new methodology for the verification of complex pipelines using a combination of theorem proving and model checking [16, 17]. We are not aware of any other verification projects where pipelined datapaths with branches and cycles in the pipeline structure the are formally verified.

## 5 Translation to Verilog HDL

Our group has implemented a translation tool that converts the PVS hardware specification to Verilog HDL [7]. This tool has been used to implement the VAMP FPU on a Xilinx FPGA (hosted on a PCI board). We have tested the implementation with several hundred thousand test vectors without encountering a bug in the VAMP FPU. The complete VAMP processor will be implemented on the FPGA within a few weeks.

## References


1. M. D. Aagaard and C.-J. H. Seger. The formal verification of a pipelined double-precision IEEE floating-point multiplier. In *ICCAD*, pages 7–10. IEEE, Nov. 1995.
2. Arvind and X. Shen. Using term rewriting systems to design and verify processors. *IEEE Micro Special Issue on Modeling and Validation of Microprocessors*, 19(3):36–46, May 1999.
3. S. Berezin, A. Biere, E. Clarke, and Y. Zhu. Combining symbolic model checking with uninterpreted functions for out-of-order processor verification. In G. Gopalakrishnan and P. Windley, editors, *FMCAD*, volume 1522 of *LNCS*, pages 369–386. Springer-Verlag, 1998.
4. C. Berg and C. Jacobi. Formal verification of the VAMP floating point unit. In *CHARME 2001*, volume 2144 of *LNCS*, pages 325–339. Springer, Sept. 2001.
5. C. Berg, C. Jacobi, and D. Kröning. Formal verification of a basic circuits library. In *IASTED International Conference on Applied Informatics*. ACTA Press, Feb. 2001.
6. S. Beyer. Formal verification of a cache memory interface. Submitted for publication, 2002.
7. S. Beyer, C. Jacobi, D. Kröning, and D. Leinenbach. Correct hardware by synthesis from PVS. Submitted for publication, 2002.
8. Y.-A. Chen and R. E. Bryant. Verification of floating point adders. In *CAV'98*, volume 1427 of *LNCS*, 1998.
9. G. Even and W. J. Paul. On the design of IEEE compliant floating point units. In *Proceedings of the 13th Symposium on Computer Arithmetic*. IEEE Computer Society Press, 1997.
10. J. Harrison. A machine checked theory of floating point arithmetic. In *TPHOL '99*, volume 1690 of *LNCS*. Springer, 1999.
11. J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, second edition, 1996.

12. T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *CAV'98*, volume 1427 of *LNCS*, pages 440–451. Springer-Verlag, 1998.
13. R. Hosabettu. *Systematic Verification of Pipelined Microprocessors*. PhD thesis, University of Utah, Department of Computer Science, 2000.
14. Institute of Electrical and Electronics Engineers. *ANSI/IEEE standard 754–1985, IEEE Standard for Binary Floating-Point Arithmetic*, 1985.
15. C. Jacobi. Formal verification of a theory of ieee rounding. In *Suppl. Proc. TPHOLs 2001*, 2001. Informatics Research Report EDI-INF-RR-0064, Univ. Edinburgh, UK.
16. C. Jacobi. Formal verification of complex out-of-order pipelines by combining model-checking and theorem-proving. Submitted for publication, 2002.
17. C. Jacobi. *Formal Verification of a Fully IEEE Compliant Floation Point Unit*. PhD thesis, Saarland University, Computer Science Department, April 2002.
18. D. Kröning. *Formal Verification of Pipelined Microprocessors*. PhD thesis, Saarland University, Computer Science Department, 2001.
19. K. L. McMillan. Circular compositional reasoning about liveness. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods: IFIP WG 10.5 Advanced Research Working Conference, CHARME '99*, pages 342–345. Springer-Verlag, 1999.
20. P. S. Miner. Defining the IEEE-854 floating-point standard in PVS. Technical Report TM-110167, NASA Langley Research Center, 1995.
21. 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.
22. S. M. Mueller, H. Leister, P. Dell, N. Gerteis, and D. Kroening. The impact of hardware scheduling mechanisms on the performance and cost of processor designs. In *15th GI/ITG Conference 'Architektur von Rechensystemen' ARCS'99*, pages 65–73. VDE Verlag, 1999.
23. S. M. Mueller and W. J. Paul. *Computer Architecture. Complexity and Correctness*. Springer, 2000.
24. 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.
25. S. Owre, N. Shankar, and J. M. Rushby. PVS: A prototype verification system. In *CADE'92*, volume 607 of *LNAI*, pages 748–752. Springer, 1992.
26. D. M. Russinoff. A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor. *LMS Journal of Computation and Mathematics*, 1:148–200, 1998.
27. J. Sawada and W. A. Hunt. Results of the verification of a complex pipelined machine model. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods: IFIP WG 10.5 Advanced Research Working Conf., CHARME '99*, pages 313–316. Springer, 1999.
28. R. M. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. In *IBM Journal of Research and Development*, volume 11 (1), pages 25–33. IBM, 1967.