From Formal Verification to Silicon Compilation

Reprint of a paper from Proceedings of IEEE Compcon, San Francsico CA, February 1991, pp. 450–455

J. Joyce[*], E. Liu, J. Rushby, N. Shankar, R. Suaya, F. von Henke

Computer Science Laboratory, SRI International
333 Ravenswood Ave., Menlo Park, California 94025

Abstract

Formal verification is emerging as a viable method for increasing design assurance for VLSI circuits. Potential benefits include reduction in the time and costs associated with testing and redesign, improved documentation and ease of modification, and greater confidence in the quality of the final product. This paper reports on an experiment whose main purpose was to identify the difficulties of integrating formal verification with conventional VLSI CAD methodology. Our main conclusion is that the most effective use of formal hardware verification will be at the higher levels of VLSI system design, with lower levels best handled by conventional VLSI CAD tools.

1 Introduction

Design errors in digital integrated circuits are ruinous: the devices are manufactured in vast quantities and, unlike software, errors cannot be remedied in the field by patching. Recently discovered errors in the Intel 486 demonstrate that these concerns are not academic. In addition to economic considerations, the use of digital systems in critical applications (for example, flight-critical digital avionics, encryption and other security-critical applications) increases the degree of design assurance necessary.

[*]Department of Computer Science, University of British Columbia, 6356 Agricultural Road, Vancouver, B.C., Canada V6T 1W5

Simulation, the present method for verification of VLSI designs, is stretched to its limits, and new methods are needed for the economical and reliable verification of VLSI designs.

Formal verification, the mathematical demonstration of consistency between a specification and a design, has been applied to critical software systems for some time. It now affords a promising new method for the validation of VLSI designs.

VLSI designs and specifications are generally simpler and more regular than those for software, and the theorem-proving bottleneck that afflicts formal verification of software is a much smaller problem for VLSI designs. Instead, the limitations on the application of formal verification to VLSI designs are those concerned with its integration into engineering practice.

During the summer of 1989, members SRI's Computer Science Laboratory, Cambridge Computer Science Research Centre, and the Integrated Circuit and Systems Program collaborated on the experimental verification and implementation of a very simple microprocessor using three different verification systems and a commercial silicon compiler. This experiment identified several main research issues including the subject of this paper: the interface between formal verification and the capabilities and requirements of VLSI CAD systems.

2 Formal Verification

There have been several applications of formal verification to VLSI designs in the last few years, but the only VLSI device that has been subjected to formal verification and is actually in production is the British Viper microprocessor [11, 10]. This was designed by staff at the UK Royal Signals and Radar Establishment (RSRE) and is intended for civilian and military safety-critical applications (for example, railroad signaling and missile guidance). Aspects of the Viper microprocessor were formally verified by Avra Cohn [7–9] at Cambridge University using a formal verification system called HOL [5, 14] developed by Mike Gordon.

The HOL system is based on higher-order logic. The very advanced EHDM formal verification system [16, 22] developed for the National Computer Security Center by SRI's Computer Science Laboratory, is also based on higher-order logic. Many of the verification techniques developed for HOL can be reproduced in EHDM.

An early and substantial hardware verification experiment performed in the USA was Warren Hunt's verification of the FM8501/2 microprocessor [17] using the Boyer-Moore Theorem Prover [3, 4]. Unlike the higher-order approaches used in HOL and EHDM, the Boyer-Moore Theorem Prover is based on quantifier-free first-order logic.

3 An Experiment

The goal of our collaborative experiment was not to develop new techniques for hardware verification, but to identify principal difficulties preventing its more widespread application, and to explore methods for alleviating those difficulties.

We conducted our investigation using the "Tamarack" microprocessor designed for concrete experimentation. The original design was proposed by Gordon [15] as a verification example; subsequent variants have been used to illustrate a variety of simulation and verification environments [1, 6, 12, 13, 18, 21, 24, 25], and some have been carried through to fabrication [20].

We began our research by examining a verification of Tamarack in the HOL system; we then experimented with alternative specification and verification strategies using HOL as well as two other verification systems, the EHDM system and the Boyer-Moore Theorem Prover. We examined these various specifications and proof fragments from technical viewpoints (e.g., how compact were the specifications, how difficult was it to construct the necessary proofs), and also from the point of view of the potential user of the technology. We also studied the problem addressed in this paper, namely, the problem of converting the descriptions used in our formal specification and verification of Tamarack to those required by modern VLSI CAD systems. We performed the necessary translation for one particular CAD system—the GENESIL silicon compiler [23]—and carried the design down to the level required as input to a VLSI fabrication facility.

Our experiment lead to the identification of three main research issues:

- The use of abstraction and layering to structure and simplify specifications and proofs.

- Technical issues concerning the use of functional as opposed to relational specification styles, the use of explicit quantification and higher-order functions, and finally, the value of various theorem-proving capabilities and strategies.

- The need to consider the capabilities and requirements of VLSI CAD systems when formulating a strategy for formal specification and verification.

These issues are discussed in detail in a forthcoming report [19]. Our experience with the last of these three issues—the interface between formal verification and modern VLSI CAD systems—is briefly summarized in this paper.

4 Interfacing with VLSI CAD

Formal hardware verification is no more than an academic exercise unless verified designs can be turned into functional chips. The input usually required by VLSI fabrication systems is a very low-level description of the geometrical chip layout. Several VLSI CAD systems are available that can produce a layout from a higher-level description of the desired chip. Such systems play a role comparable to compilers and assemblers in software development—indeed, some VLSI CAD systems are referred to as *silicon compilers*. A typical silicon compiler takes a register-transfer level (RTL) design as input and produces a geometrical chip layout as output. Just as a modern programming language is often supported by a complete programming environment, so a silicon compiler CAD system usually provides additional tools such as a simulator, and analyzers for timing and power dissipation.

In order to fabricate a verified VLSI design, it is necessary to translate the formal specification into the notation required by the chosen CAD/CAM system. The shift from one notational system to another necessarily incurs some risk, and it might seem that the best way to minimize the risk is to remain with the purely formal system as long as possible. Thus many papers on hardware verification are concerned with low-level designs—for example, demonstrating the correctness of gate-level implementations of register-transfer level components. In our view, this emphasis on low-level designs is misplaced for the following reasons.

First, a VLSI chip is a physical device and its behavior must ultimately be modeled by the laws of physics, not of logic. Purely logical gate-level models do not account for such important physical considerations as fan-in and fan-out, power dissipation, timing, or the geometrical design rules for the technology concerned. It is perfectly feasible that a logically correct design could fail to function properly owing to the neglect of these or other physical factors.

Second, modern VLSI CAD tools do deal with these physical constraints quite successfully. By employing libraries of tried-and-tested implementations for standard register-transfer level components, by modeling the relevant physical properties, by enforcing design rules, and by using mathematical optimization techniques, such CAD tools deliver reliable, well-optimized, cleanly-routed chip layouts from a register-transfer level design. Of course, there is no formal guarantee that the chip layout produced a VLSI CAD tool correctly implements the register-transfer design provided as its input, but there is a great deal of pragmatic evidence that it will do so. The situation is analogous to that of a conventional software compiler: it is entirely possible that a compiler may generate incorrect code, but the likelihood that a program will fail because of a compiler bug is small (assuming it is a widely-used, mature compiler) compared with the likelihood that a program will fail because of faults in its own design. Similarly, we consider it much more likely that a VLSI chip will fail due to high-level design errors, than due to a flaw in the implementation of, say, an ALU from a mature cell library.

In an ideal world, compilers and VLSI CAD tools would be verified, but in the world as it presently exists we believe that the cause of reliability (not to mention cost-effectiveness) is likely to be best served by using mature, if unverified, high-level compilers and CAD tools, and verifying the source text that is provided to them, rather than in carrying formal verification down to levels where physical properties becomes significant.

Given this perspective, we decided to construct an implementation of Tamarack using GENESIL—a silicon compiler—in order to investigate the issues involved in the transition between a formal specification and the notation required as input to this VLSI CAD tool.

5 GENESIL: A Silicon Compiler

The GENESIL silicon compiler is a system that generates VLSI design layouts from a register-transfer level description of the design.

The GENESIL Function Set is a library of standard RTL components that can be combined to form larger systems. This function set is comprised of various types of functional blocks: independent blocks such as ROMs, PLAs and multipliers; datapath blocks such as ALUs and shifters; random logic blocks including multiplexors, latches, clocks, gates, decoders, flags, and synchronizers.

Instances of these blocks are specified when the user supplies the required parameters to GENESIL. The blocks are then combined and interconnected according to a user-supplied description. The GENESIL compiler checks the design for design and timing errors and produces a chip layout. Apart from the layout, GENESIL also produces a functional model for simulation, a timing model for timing analysis, and a load model for studying power dissipation.

GENESIL supports a CMOS-based design methodology based on synchronous design, transparent level-sensitive latching, and two-phase non-overlapping clocks.

In the GENESIL system, signals are assigned timing attributes according to the triggering clock phase of the devices that they can drive. A Valid-A(t) signal holds its value through the *t*'th falling Clock-A edge, and similarly, a Valid-B(t) signal holds it value through the *t*'th falling Clock-B edge. A Stable-A(t) signal sets up and holds its value through the *t*'th falling Clock-B edge and the following *t+1*'th Clock-A edge. Likewise, a Stable-B(t) signal holds its value through the *t*'th falling Clock-A and Clock-B edges.

GENESIL functions blocks specify the timing attributes of the signals they can accept and produce. The output of *transparent latch* (clocked on Phase-A, say) tracks it input when Clock-A is high and latches the input value at the falling edge. The output of such a latch is held until Clock-A goes high again. Thus, for a transparent latch, the expected input is Valid-A(t) and the output is Stable-B(t). A *D-type flip-flop* latch clocked on Phase-B expects Valid-A(t) input and the input value is available at the output as a Stable-A(t+1) signal, i.e., with a half-cycle delay. Further attributes, such as *direct*, *precharged*, and *tri-state* are applied to bus signals.

The GENESIL signal timing attributes and associated interconnections rules are similar to the strong-typing schemes used in modern programming languages and serve a similar purpose: to clarify the statement of the user's intent and to eliminate a class of errors. In the case of GENESIL, the signal attribute and interconnection rules eliminate a large class of race conditions and timing errors.

The user interacts with GENESIL through an editor that can be used to update existing designs or add new designs. GENESIL displays a form that is specific to the block being edited, and the user selects the appropriate options. Once a design is complete, the GENESIL compiler can be invoked on the design. The user can then examine the layout graphically or simulate its functional behavior.

6 Tamarack in GENESIL

The structural information (i.e., what components are needed and how they are connected together) present in the formal RTL specification of Tamarack is very close to the input required by GENESIL. Many of the components in the RTL specification of Tamarack (e.g., ROM, RAM, registers, the ALU) correspond directly to blocks provided by GENESIL. Only a small amount of random logic was needed in the control unit of the microprocessor—and that was developed directly from a lower-level formal specification using random logic blocks provided by GENESIL.

It took about three days to construct an implementation of Tamarack in GENESIL, and much of that time was spent in learning GENESIL. The principle difficulties encountered were uncertainty about the precise functionality provided by some GENESIL function blocks, and by the plethora of sub-options available (despite copious documentation, formal specifications would have been much more useful).

Another difficulty turned out to be surprisingly troublesome. The formal specification of Tamarack, not unlike the formal specifications of both the FM8501 microprocessor and the Viper microprocessor, is based on a single-phase clocking scheme. Translation of this formal specification into a GENESIL design required a shift from the single-phase scheme to the two-phase scheme supported by GENESIL. Obviously, great care should be exercised when implementing a design that assumes a single-phase clocking discipline within a two-phase discipline. Subtle errors can creep in through clocking mismatches between the driver and the device being driven. One such error occurred in our initial implementation of Tamarack. In the microcoded control unit, the address of the next micro-instruction is latched in the microcode program counter (MPC) on Phase-B but read by the microcode ROM on Phase-A. However, the ROM delivers the corresponding micro-instruction a half-cycle later (on Phase-B) and alters the value latched by MPC. The MPC therefore latches the wrong value for the current micro-instruction address, which in turn affects all future values of MPC.

This error does not violate the GENESIL clocking rules which are automatically checked by the GENESIL system. However, the non-detection of this error this does *not* indicate a shortcoming in these rules; the error is at a higher level in that the resulting circuit is an incorrect implementation of our abstract model of sequential behavior in which every clock cycle is intended to cause a new state to be computed for the system based on the current state and current inputs. This initial implementation of Tamarack resulted in a perfectly valid circuit, but not a circuit with the behavior that we had intended.

This error was detected using the GENESIL functional simulator, and was easily repaired by changing the implementation of the MPC from a D-type flip-flop to the master-slave variety. Of course, an integrated methodology would have started with a verified design employing a two-phase clock.

7 Conclusion

We found the experience of implementing Tamarack in GENESIL very illuminating. It is clear that GENESIL and other silicon compilers provide an effective way to construct VLSI implementations from relatively high-level descriptions. Concerns such as gate-level design, layout, analysis of power consumption, timing and race conditions, conformity to clocking rules, etc., are mostly handled automatically. However, the real-world orientation of GENESIL forced us to encounter issues such as multi-phase clocking, dual-bus datapaths, precharged and tri-state busses, and a much more concrete model of sequential behavior (i.e., the various sub-options concerning the exact functionality and timing of devices such as latches and ROMs) than is normally considered in the formal verification of hardware.

For the most part, work on formal hardware verification has not dealt with these complications, yet it seem they are necessary for systems of realistic performance and physical reliability. We strongly believe that the most effective use of hardware verification will be at the higher levels of VLSI design, with lower levels handled by VLSI CAD tools such as GENESIL. The important research topic then concerns the interface between formally verified register-transfer level specifications and the input required by VLSI CAD tools. The significant interface is the semantic one: for example, the connection between the type system of the formal specification language and the signal types of the CAD tool, and the formal modeling of multi-phase clocking schemes. These present interesting challenges for future research.

Acknowledgements

This work was funded by SRI International.

References

[1] H. Barrow. VERIFY: A program for proving correctness of digital hardware designs. *Artificial Intelligence*, 24:437–491, 1984.

[2] G. Birtwistle and P. A. Subrahmanyam, editors. *VLSI Specification, Verification and Synthesis*. Kluwer Academic Publishers, Boston, MA, 1988.

[3] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, New York, 1979.

[4] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, 1988.

[5] *The HOL System*. Cambridge Computer Science Research Center, SRI International, Cambridge, England, December 1989. Three volumes: Description, Tutorial, and Reference Manual.

[6] A. J. Camilleri. Simulation as an aid to verification using the HOL theorem prover. In D. Edwards, editor, *Proceedings of the IFIP TC10 Working Conference on Design Methodology in VLSI and Computer Architecture*, pages 147–168, Pisa, Italy, September 1989. North-Holland.

[7] A. J. Cohn. A proof of correctness of the VIPER microprocessors: The first level. In Birtwistle and Subrahmanyam [2], pages 27–72.

[8] A. J. Cohn. Correctness properties of the VIPER block model: The second level. In G. Birtwistle and P. A. Subrahmnayam, editors, *Current Trends in Hardware Verification and Theorem Proving*, pages 1–91, New York, NY, 1989. Springer Verlag. Also published as University of Cambridge Computer Laboratory Technical Report No. 134.

[9] Avra Cohn. The notion of proof in hardware verification. *Journal of Automated reasoning*, 5(2):127–139, June 1989.

[10] W. J. Cullyer. High integrity computing. In M. Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lecture Notes in Computer Science*, pages 1–35. Springer Verlag, 1988. (Proceedings, First International Symposium, Warwick, UK, September 1988.).

[11] W. J. Cullyer. Implementing safety critical systems: The VIPER microprocessor. In Birtwistle and Subrahmanyam [2], pages 1–26.

[12] Paul Curzon. *A Structured Approach to the Verification of Low Level Microcode*. PhD thesis, University of Cambridge, 1990.

[13] Bruce S. Davie. *A Formal, Hierarchical Design and Validation Methodology for VLSI*. PhD thesis, Department of Computer Science, University of Edinburgh, October 1988. Technical report CST-55-88.

[14] M. J. C. Gordon. HOL: A proof generating system for higher-order logic. In Birtwistle and Subrahmanyam [2], pages 73–128.

[15] Mike Gordon. Proving a computer correct. Technical Report TR 42, University of Cambridge, Computer Laboratory, 1983.

[16] F. W. von Henke, J. S. Crow, R. Lee, J. M. Rushby, and R. A. Whitehurst. The EHDM verification environment: An overview. In *Proceedings 11th National Computer Security Conference*, pages 147–155, Baltimore, MD, October 1988. NBS/NCSC.

[17] Warren A. Hunt, Jr. Microprocessor design verification. *Journal of Automated reasoning*, 5(4):429–460, December 1989.

[18] Jeffrey Joyce. *Multi-Level Verification of Microprocessor-Based Systems*. PhD thesis, University of Cambridge, December 1989.

[19] Jeffrey Joyce, Erwin Liu, John Rushby, Natarajan Shankar, Roberto Suaya, and Friedrich von Henke. From hardware verification to silicon compilation. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1990. Forthcoming.

[20] Jeffrey J. Joyce. Formal specification and verification of microprocessor systems. *Integration, the VLSI journal*, 7:247–266, September 1989.

[21] Martin Richards. BSPL: A language for describing the behaviour of synchronous hardware. Technical Report 84, Computer Laboratory, Cambridge University, July 1986.

[22] John Rushby, Friedrich von Henke, and Sam Owre. An introduction to formal specification and verification using EHDM. Technical Report SRI-CSL-91-2, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1991.

[23] *GENESIL System: System Description Users Manual*. Silicon Compiler Systems Corporation, 2045 Hamilton Avenue, San Jose, CA, 1988. Order number 110013-3.

[24] John P. Van Tassel. The semantics of VHDL with VAL and HOL: Towards practical verification tools. Master's thesis, Wright State University, 1989.

[25] D. Weise. *Formal Multilevel Hierarchical Verification of Synchronous MOS VLSI Circuits*. PhD thesis, Massachusetts Institute of Technology, 1986.