Formal Hardware Verification of Digital Circuits

J. Joyce and C-J. Seger

Department of Computer Science

University of British Columbia

Vancouver, B.C.

Canada V6T 1W5

Abstract- **The use of formal methods to verify the correctness of digital circuits is less constrained by the growing complexity of digital circuits than conventional methods based on exhaustive simulation. This paper briefly outlines three main approaches to formal hardware verification: symbolic simulation, state machine analysis, and theorem-proving.**

1 Introduction

Advances in VLSI fabrication technology have greatly outstripped 'verification capacity' — that is, the capacity of conventional methods for demonstrating that the design of a circuit is correct with respect to a specification of its requirements.

Verification capacity has fallen behind fabrication technology because conventional verification methods do not scale with complexity. These methods are generally based on simulation — they do not scale because the number of simulation cases is likely to increase exponentially if one attempts to maintain the same degree of coverage.

Considerable effort has been made to increase, in a brute-force manner, what coverage can be achieved with simulation. One approach is to distribute the simulation cases over a large number of machines running identical versions of the simulation model. Another brute-force approach has been the development of special-purpose simulation hardware to increase the speed of a simulation by several orders of magnitude. However, these techniques do not offer a satisfactory, long-term solution for verifying digital designs by exhaustive simulation because, in general, the number of simulation cases grows exponentially with the number of components in a design.

Of course, it may be argued that it is not really necessary, for any practical purpose, to exhaustively simulate a design in order to detect every error in a design. Instead, it would be argued that it is only reasonable to simulate the design for a feasible number of representative cases. However, this assumes that there is general-purpose, systematic method for finding a truely representative set of simulation cases. Although one can easily imagine a systematic way of generating some obvious cases, it is clear that digital systems often fail at the "confluence of unrelated or seemingly unrelated events" [25].

Formal methods offer considerable hope for verification techniques which are better able to scale with the complexity of VLSI designs. We can identify three distinct approaches to formal hardware verification, namely, symbolic simulation, state machine analysis, and theorem-proving. These formal approaches to hardware verification are better able to scale

with the complexity of VLSI designs because they exploit powerful tools of mathematics rather than brute-force. A good example is the use of the mathematical induction which is a mainstay of the theorem-proving approach to formal hardware verification. All of these approaches are supported by software tools — many of which have been under constant development for the last decade or longer.

2   The Symbolic Simulation Approach

The concept of symbolic simulation was first proposed by researchers in the late 1970's as a method for evaluating register transfer language representations [11]. The early programs were very limited in their analytical power since their symbolic manipulation methods were weak. Consequently, symbolic simulation did not evolve much further until more efficient methods of manipulating symbols emerged. The development of Ordered Binary Decision Diagrams (OBDDs) for representing Boolean functions [8] radically transformed symbolic simulation.

The first "post-OBDD" symbolic simulators were simple extensions of traditional logic simulators [7]. In these symbolic simulators the input values could be arbitrary Boolean expressions over some Boolean variables rather than only 0's, 1's (and possibly X's) as in traditional logic simulators. Consequently, the results of the simulation were not single values but rather Boolean functions describing the behavior of the circuit for the set of all possible data represented by the Boolean variables. To illustrate this idea, consider the (pseudo) Domino CMOS circuit shown in Fig. 1. If the circuit is clocked correctly, the inputs are stable long enough before the clock goes high, and the inputs and clock signal are then kept stable, the output node should eventually change to 1 if and only if the number represented by the 4-bit binary input vector a is greater than the number represented by the 4-bit binary input vector b, and both numbers are greater than zero. In a simple OBDD-based symbolic simulator we would simply apply the Boolean input variables at the correct time and in the end compare the value on the output node with the Boolean function:

$$(a_3\bar{b}_3 + \overline{(a_3 \oplus b_3)}(a_2\bar{b}_2 + \overline{(a_2 \oplus b_2)}(a_1\bar{b}_1 + \overline{(a_1 \oplus b_1)}a_0\bar{b}_0)))(b_3 + b_2 + b_1 + b_0)$$

A verifier based on symbolic simulation applies logic simulation to compute the circuit's response to a series of stimuli chosen to detect all possible design errors. When a circuit has been "verified" by simulation, this means that any further simulation would not uncover any errors. Hence, the problem of verifying the correctness of a design becomes one of simulating a large number of input patterns. Selecting such a set of simulation patterns is a nontrivial task, since errors that arise during the design process cannot be easily characterized. Designer's misconceptions, incomplete or inconsistent specifications, and carelessness on the part of the designer can cause the resulting circuit to behave unpredictably. Worst of all, it may misbehave only under unusual combinations of circumstances. Rather than trying to postulate a simplified "fault model" for design errors, it is more appropriate to
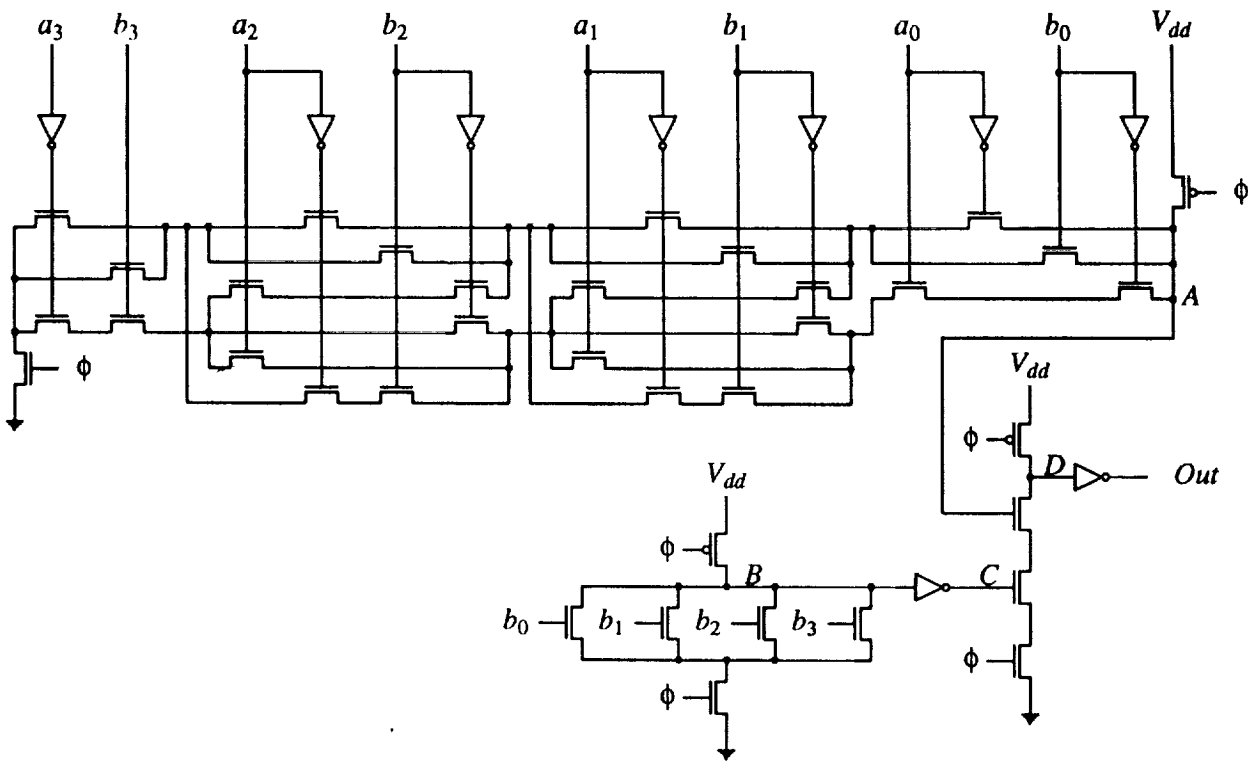
Figure 1: Circuit for computing $A > B > 0$.

adopt a philosophy that design verification must work against a malicious adversary. That is, given a proposed simulation test, the adversary will attempt to create a circuit that does not fulfill the specification, yet passes the test. A circuit is considered "correct" only if no adversary can defeat the simulation test. Thus, when a circuit has been "verified" by simulation, this means that any further simulation would not uncover any errors.

Since a symbolic simulator is based on a traditional logic simulator, it can use the same, quite accurate, electrical and timing models to compute the circuit behavior. For example, a detailed switch-level model, capturing charge sharing and subtle strengths phenomena, and a timing model, capturing bounded delay assumptions, are well within reach. Also—and of great significance—the switch-level circuit used in the simulator can be extracted automatically from the physical layout of the circuit. Hence, the correctness results will link the physical layout with some higher level of specification.

Recently, Bryant and Seger [10] developed a new generation of symbolic simulator based verifier. Here the simulator establishes the validity of formulas expressed in a very limited, but precisely defined, temporal logic. This temporal logic allows the user to express properties of the circuit over *trajectories*: bounded-length sequences of circuit states. The verifier checks the validity of these formulas by a modified form of symbolic simulation. Further, by exploiting the 3-valued modeling capability of the simulator, where the third logic value X indicates an unknown or indeterminate value, the complexity of the symbolic

manipulations is reduced considerably.

This verifier supports a verification methodology in which the desired behavior of the circuit is specified in terms of a set of assertions, each describing how a circuit operation modifies some component of the (finite) state or output. The temporal logic allows the user to define such interface details as the clocking methodology and the timing of input and output signals. The combination of timing and state transition information is expressed by an assertion over state trajectories giving properties the circuit state and output should obey at certain times whenever the state and inputs obey some constraints at earlier times.

This form of specification works very well for circuits that are normally viewed as state transformation systems, i.e., where each operation is viewed as updating the circuit state. Using a prototype system, a simple 32-bits microprocessor and a significant portion of a modern 32 bit RISC microprocessor have been verified. These circuits contained around 15,000 transistors and the verification effort required less than two hours on a MIPS Magnum 3000 workstation. The complete verification process including developing the specification, deriving the circuit description, and carrying out the symbolic ternary simulation, took less than a person-week.

3   State Machine Analysis

A second approach to formal hardware verification is *state machine analysis*. This approach uses algorithmic techniques to decide whether a finite state machine satisfies a set of user-specified properties. In this brief overview, we focus on just one particular approach to state machine analysis called *model-checking*. Other approaches to state machine analysis include those based on *language containment tests* [23].

We use the example of a simple handshaking protocol, illustrated by the timing diagram in Figure 2, to describe the state-machine analysis approach to formal hardware verification.
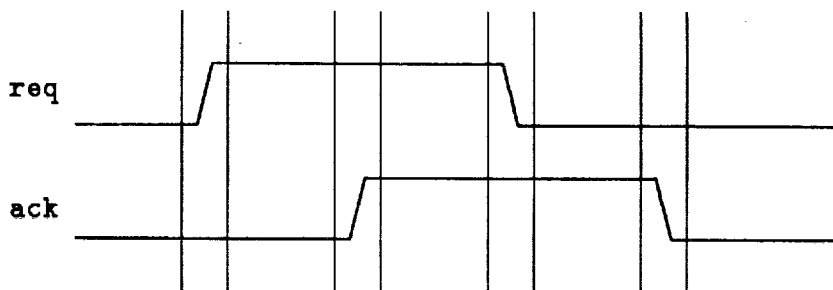

Figure 2: Timing diagram for simple handshaking protocol.

This protocol could be implemented either in software or directly in hardware. A 'correct' implementation of this protocol must satisfy the following properties:

"whenever the request signal becomes true, it must remain true
until it is acknowledged"

"every request must eventually be acknowledged"

"whenever the acknowledgement signal becomes true, it must remain true
until the request signal returns to false"

"the request signal will eventually return to false after
the request is acknowledged"

"whenever the request signal is false, it will remain false until
the acknowledgement signal is also false"

"the acknowledgement signal will eventually return to false after
the request signal returns to false"

"once false, the acknowledgement signal will remain false until
there is a request"

"whenever the acknowledgement signal is false,
there will eventually be a request"

These properties can be translated one-by-one into temporal logic. The symbols $\cup$, $\diamond$, $\sim$ and $\longrightarrow$ can be informally read as "until", "eventually", "not" and "implies".

$$(\text{req} \longrightarrow (\text{req} \cup \text{ack}))$$

$$(\text{req} \longrightarrow (\diamond\text{ack}))$$

$$(\text{ack} \longrightarrow (\text{ack} \cup (\sim\text{req})))$$

$$(\text{ack} \longrightarrow (\diamond(\sim\text{req})))$$

$$((\sim\text{req}) \longrightarrow ((\sim\text{req}) \cup (\sim\text{ack})))$$

$$((\sim\text{req}) \longrightarrow (\diamond(\sim\text{ack})))$$

$$((\sim\text{ack}) \longrightarrow ((\sim\text{ack}) \cup \text{req}))$$

$$((\sim\text{ack}) \longrightarrow (\diamond\text{req}))$$

A program for automatic state machine analysis would take, as input, a machine-readable list of formally specified properties such as the eight properties listed above. The analyzer program would also take, as input, a machine-readable description of a finite state machine, for example, a model of a candidate implementation of the handshaking protocol. The analyzer program would then generate either the answer "yes", meaning that the state machine does indeed satisfy all of the properties supplied by the user, or the answer "no", meaning that the machine fails to satisfy at least one of these properties. When the outcome is "no", the analyzer may also produce helpful information about how the state machine fails to satisfy a particular property, i.e., a counter-example.

State-machine analysis is bounded by the number of states in the finite state machine. Early state machine analysis techniques relied on the explicit enumeration of states which,

reportedly, limits the use of these techniques to systems with between 10^3 and 10^6 reachable states. Unfortunately, the number of states in a system may grow exponentially with the number of concurrent components in the system. To deal with this "state explosion problem", several groups [3,9,16] have investigated ways to represent a state space symbolically rather than explicitly. A popular candidate for the symbolic representation of states are OBDD's — mentioned earlier in connection with the symbolic simulation approach. Using this symbolic approach, it is reported that state machine analysis can be applied in practice to systems with in excess of 10^{20} states [9].

State machine analysis techniques have been applied to several commercial designs. These techniques were used to discover several possible execution sequences leading to failure in a design for the cache consistency protocol of the Encore Gigamax multiprocessor [24]. Another approach to state machine analysis (based on language containment) has been used by AT&T in the design of a packet layer controller chip [23].

4 Theorem Proving

A third approach to formal hardware verification, computer-assisted *theorem-proving*, is based on the construction of a proof in formal logic. This proof is a formal argument that a hardware design, based on some model of the primitive components, satisfies a formal specification of its requirements. Figure 3 shows an example of a formal proof establishing the correctness of the two-component design shown in Figure 4.

1.	ANDGate_IMP (i1,i2,outp)	[from above circuit diagram]
2.	∃x. NANDGate (i1,i2,x) ∧ NOTGate (x,outp)	[by def. of ANDGate_IMP]
3.	NANDGate (i1,i2,x) ∧ NOTGate (x,outp)	[strip off "∃x."]
4.	NANDGate (i1,i2,x)	[left conjunct of line 3]
5.	x = ¬(i1 ∧ i2)	[by def. of NANDGate]
6.	NOTGate (x,outp)	[right conjunct of line 3]
7.	outp = ¬x	[by def. of NOTGate]
8.	outp = ¬(¬(i1 ∧ i2))	[substitution, line 5 into 7]
9.	outp = (i1 ∧ i2)	[simplify, ¬¬t = t]
10.	ANDGate (i1,i2,outp)	[by def. of ANDGate]
11.	ANDGate_IMP (i1,i2,outp) ⟹ ANDGate (i1,i2,outp)	[discharge assumption, line 1]

Figure 3: Formal proof of correctness for an AND-Gate.

Figure 4: Implementation of an AND-gate from an NAND-gate and an inverter.

Both ordinary human reasoning and formal proof can be used to show that a specific conclusion follows from a certain set of assumptions by accepted laws of reasoning. However, formal proof is a purely syntactic process. A proof is formally defined as a sequence of lines (such as the numbered sequence of lines in Figure 3) where each line follows from a previous line by rule of inference. There are only a finite number of primitive inference rules (in fact, usually a very small number of primitive rules). The validity of any particular line in a proof can be decided by a purely syntactic test based on checking to see if any one of the primitive inference rules can be used to justify that particular line in the proof.

Unlike ordinary human reasoning, which is notoriously error-prone, formal proof is extremely rigorous. Indeed, its main advantage is that it can be mechanically checked. The main disadvantage of formal proof, compared to ordinary human reasoning, is that formal proof is overwhelmingly tedious. The very simple proof in Figure 3 has just eleven lines, but a formal proof of correctness for a real design (such as a simple microprocessor) may involve several million primitive inference steps. Fortunately, there has been considerable progress made towards the partial automation of formal proof. A very large fraction of the actual line-by-line inference steps in a formal proof can be generated automatically by computer-based theorem-prover.

A digital circuit can be "verified" using a theorem-prover by generating a theorem which states that the formal specification of a design logically *satisfies* a formal specification of its intended behaviour (i.e. a high level model). The exact meaning of "satisfies" is stated unambiguously as a mathematical relationship between the two levels of formal specification. In the very simple example shown in Figure 3, logical implication, $\Longrightarrow$, is used to express the relationship between the implementation of the AND-gate and its behavioural specification:

```
ANDGate_IMP (i1,i2,outp)  ==>  ANDGate (i1,i2,outp)
```

The theorem-proving approach to formal hardware verification is a *structural* approach in contrast to both symbolic simulation and state-machine analysis which are *behavioural* approaches. The latter two approaches, symbolic simulation and state-machine analysis, both apply verification techniques to a 'flat' design — they do not require additional details about the hierarchical structure of the design. On the other hand, theorem-proving can only be applied 'in the large' to a hierarchically structured design. In a theorem-proving approach, the design is verified hierarchically. The proof hierarchy is generally a reflection of the hierarchical structure of the design. For example, the bottom level of this hierarchical process may involve the formal verification of simple RTL (Register Transfer

Level) components composed from primitives such as CMOS transistors. Each kind of component only has to be verified once — in contrast to other approaches which verify every instance of that particular component. At higher levels in the verification hierarchy, each instance of a particular component uses the single verification result obtained from a lower level.

The direct re-use of a verification result for multiple identical instances of a particular component is a very simple form of how a single verification result can be re-used. Theorem-proving approaches also allow *generic* specifications to be formally verified where a specification is parameterized is by scalar values (e.g., the number of bits in a RTL component) or even by data types and operations [21]. A single generic verification result can be instantiated for different parameter values; for example, the generic specification of an n-bit multiplier can be instantiated for different values of n, e.g., a 16-bit multiplier, a 32-bit multiplier.

The theorem-proving approach relies heavily upon (and benefits greatly from) a number of mathematical tools. This includes, as with symbolic simulation, the ability to represent data symbolically. Mathematical induction is also critical for scaling with the increasingly complexity of circuit designs.

A distinct advantage of the theorem-proving approach to formal hardware verification is the ability to verify digital systems with respect to higher algebraic levels. For example, the correctness of arithmetic hardware can be stated directly in terms of arithmetic operations on natural numbers rather than Boolean operations on bit-vectors. This is often referred to as *data abstraction* — an illustrative example of this technique is given by Chin [12] in verifying arithmetic hardware for signal processing applications. Other kinds of abstraction include *temporal abstraction* which is a technique for relating computational behaviour at increasingly abstract time scales.

Among the best known interactive theorem-provers are the Boyer-Moore Theorem Prover [4] and the Cambridge HOL System [18,19]. The Boyer-Moore Theorem Prover has been used by researchers at Computational Logic Inc. to develop an multi-level proof of correctness for a complete computer system including both hardware and software levels [1,2]. The Cambridge HOL System has been used by researchers at Cambridge University to verify aspects of the commercially-available Viper microprocessor designed by the British Ministry of Defence for safety-critical applications [6,13,14,15].

5   Summary and Future Directions

This paper has briefly described three main approaches to formal hardware verification: symbolic simulation, state-machine analysis and theorem-proving. There have already been some trial applications of these verification techniques to real commercial designs (as mentioned earlier) and there is evidence of increasing industrial interest in these techniques.

Many research efforts in this area are now focussed on the issue of integrating formal verification techniques with conventional CAD. For example, researchers at Cambridge University are investigating the use of conventional HDL's (Hardware Description Lan-

guages) such as Ella and VHDL as specification languages for theorem-proving techniques [5]. Another example is work that investigates links between formally verified hardware specifications and conventional CAD tools such as silicon compilers [22].

It is unlikely that any one of the three verification approaches described in this paper offers, by itself, a "complete" approach to verifying digital hardware. However, we believe that a "complete" approach may be achieved by some combination of the three approaches described here. We are currently developing a hybrid approach that based on a combination of symbolic simulation (using the COSMOS system) and theorem-proving (using the Cambridge HOL system). The objective of our research is a hybrid formal verification methodology (and supporting tools) which combines the complementary advantages of theorem-proving and symbolic simulation. This methodology would allow a very abstract specification of a digital system (specified with the full expressive power of higher-order logic) to be verified with respect to a switch-level model of a CMOS digital circuit. Initial progress on the development of a "mathematical interface" for this hybrid approach is reported in [26].

Acknowledgements

References

[1] W. R. Bevier, W. A. Hunt, Jr., and W. D. Young, Towards Verified Execution Environments, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 27-29 April 1987, Oakland, California Computer Society Press, Washington, D.C., 1987 pp. 106-115. Also Report No. 5, Computational Logic, Inc., Austin, Texas, February 1987.

[2] W. Bevier, W. Hunt, J Moore, and W. Young, An Approach to Systems Verification, *Journal of Automated Reasoning*, Vol. 5, No. 4, November 1989. Also Report No. 41, Computational Logic, Inc., Austin, Texas, April 1989.

[3] S. Bose and A. Fisher, Automatic Verification of Synchronous Circuits using Symbolic Logic Simulation and Temporal Logic, *Proceedings of the IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Designs*, Houthalen, Belgium, 1989.

[4] R. S. Boyer and J S. Moore, A Computational Logic, Academic Press, 1979.

[5] R. Boulton, M. Gordon, J. Herbert, and J. Van Tassel, The HOL Verification of ELLA Designs, *Proceedings of the ACM 1991 International Workshop on Formal Methods in VLSI Design*, P. Subrahmanyam, ed., Miami, Florida, 9-11 January 1991

[6] B. Brock and W. A. Hunt, Jr., Report on the Formal Specification and Partial Verification of the VIPER Microprocessor, Report No. 46, Computational Logic, Inc., Austin, Texas, January 1990.

[7] R.E. Bryant, Symbolic Verification of MOS Circuits. *1985 Chapel Hill Conference on VLSI*, Fuchs, H., Ed. Computer Science Press, Rockville, MD, 1985, pp. 419–438.

[8] R.E. Bryant, Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. C-35, No. 8, December 1986, pp. 677–691.

[9] J. Burch, E. Clarke, and K. McMillan, Symbolic Model Checking: 10^{20} States and Beyond. Manuscript, 1990.

[10] R.E. Bryant and C-J. Seger, Formal Verification of Digital Circuits Using Symbolic Ternary System Models, *Computer-Aided Verification '90*, eds. E.M. Clarke, R.P. Kurshan, American Mathematical

[11] W.C. Carter, W.H. Joyner, Jr., and D. Brand, Symbolic Simulation for Correct Machine Design, *16th ACM/IEEE Design Automation Conference*, 1979, pp. 280–286.

[12] S-K. Chin, Synthesis of Arithmetic Hardware Using Hardware Metafunctions, *IEEE Trans. CAD*, Vol. 9, No. 8, August 1990, pp. 793-803.

[13] Avra Cohn, A Proof of Correctness of the Viper Microprocessor: The First Level, *VLSI Specification, Verification and Synthesis*, G. Birtwistle and P. Subrahmanyam, eds., Kluwer Academic Publishers, Boston, 1988, pp. 27-71. Also Report No. 104, Computer Laboratory, Cambridge University, January 1987.

[14] Avra Cohn, Correctness Properties of the Viper Block Model: The Second Level, *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. Subrahmanyam, eds., Springer-Verlag, 1989, pp. 1-91. Also Report No. 134, Computer Laboratory, Cambridge University, May 1988.

[15] Avra Cohn, The Notion of Proof in Hardware Verification, *Journal of Automated Reasoning*, Vol. 5, May 1989, pp. 127-139.

[16] O. Coudert, C. Berthet, and J. C. Madre. Verification of Synchronous Sequential Machines basd on Symbolic Execution, *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.

[17] M. J. C. Gordon, Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware, *Formal Aspects of VLSI Design, Proceedings of the 1985 Edinburgh Conference on VLSI*, G. Milne and P. Subrahmanyam, eds., North-Holland, 1986, pp. 153-177.

[18] M.J.C. Gordon, Mechanizing Programming Logics in Higher Order Logic, *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. Subrahmanyam, eds., Springer-Verlag, 1989, pp. 387-439. Also Report No. 145, Computer Laboratory, Cambridge University, September 1988.

[19] M.J.C. Gordon et al., The HOL System Description, Cambridge Research Centre, SRI International, Suite 23, Miller's Yard, Cambridge CB2 1RQ, England.

[20] Warren A. Hunt, FM8501, A Verified Microprocessor, Ph.D. Thesis, Report No. 47, Institute for Computing Science, University of Texas, Austin, December 1985.

[21] Jeffrey J. Joyce, Generic Specification of Digital Hardware, *Proceedings of Workshop on Digital Circuit Correctness*, M. Sheeran and G. Jones, eds., September 1990, Oxford.

[22] J. Joyce, E. Liu, J. Rushby, N. Shankar, R. Suaya, and F. von Henke, From Formal Verification to Silicon Compilation, Proceedings of *COMPCON 91*, San Francisco, California, 26-27 February 1991.

[23] R. Kurshan, Automaton (State-Machine)-Based Analysis, *Tutorial on Formal Verification of Hardware*, DAC'91, 21 June 1991, San Francisco.

[24] K. McMillan and J. Schwalbe, Formal Verification of the Gigamax Cache Consistency Protocol, manuscript, 1990.

[25] P. G. Neuman, The Computer-Related Risk of the Year: Weak Links and Correlated Events, Proceedings of *COMPASS '91*, Washington, D.C., 24-27 June 1991.

[26] C.-J. Seger and J. J. Joyce, A Two-Level Formal Verification Methodology using HOL and COSMOS, *Proceedings of the Third Workshop on Computer Aided Verification*, K. Larsen and A. Skou, eds., Aalborg, Denmark, 1-4 July 1991. pp. 380-391.