A Two-Level Formal Verification Methodology using HOL and COSMOS *

Carl-Johan H. Seger and Jeffrey J. Joyce
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z2 Canada

Abstract

Theorem-proving and symbolic simulation are both described as methods for the *formal verification of hardware*. They are both used to achieve a common goal—correctly designed hardware. However, they have different strengths and weaknesses. The main significance of this paper—and its most original contribution—is the suggestion that symbolic simulation and theorem-proving can be combined in a complementary manner. We also illustrate this combined approach on an example that neither tool can handle well.

1 Introduction

Designing complex digital system in VLSI technology usually involves working at several levels of abstraction, ranging from very high level behavioral specifications down to physical layout at the lowest. One of the main difficulties in this process is to verify the consistency of the different levels of abstraction. Simulation is often used as the main tool for "checking" the consistency. Despite major simulation efforts, serious design errors often remain undetected. Consequently, there has been a growing interest in using formal methods to verify the correctness of designs. There are three general approaches to formal hardware verification: theorem-proving, state machine analysis, and symbolic simulation. These methods all have their strengths and weaknesses. In this paper we will illustrate how theorem-proving can be used in conjunction with symbolic simulation to gain a verification methodology that draws on the strengths of each approach.

Most research on formal verification has relied on the use of computer-assisted theorem provers [3, 8, 9, 12, 16] to establish equivalence between different circuit representations. One of the main strengths of the theorem-proving approach is its ability to describe and relate circuit behaviors at many different levels of abstraction. By being able to reason about the circuit at increasingly higher levels

*This research was supported by operating grants from the Natural Sciences and Engineering Research Council of Canada.

of abstraction, we can eventually minimize the semantic gap between the formal high-level specification and the informal, intuitive, specification of the circuit that resides in the mind of the designer.

Unfortunately, theorem-proving based verification requires a large amount of effort on the part of the user in developing specifications of each component and in guiding the theorem prover through all of the lemmas. Also, in order to make the proofs tractable, most attempts at this style of verification have been forced to use highly simplified circuit models.

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. Since a symbolic simulator is based on a traditional logic simulator, it is can use the same, quite accurate, electrical and timing models to compute the circuit behavior. 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 [6] developed a new type of symbolic simulators. Here the simulator establishes the validity of formulas expressed in a very limited, but precisely defined, temporal logic. By limiting the complexity of the logic, great efficiency is obtained. Furthermore, the verification process is highly automated. Unfortunately, the automation obtained by the symbolic simulator comes with a price. First of all, for some behaviors, the computational requirements for carrying out a correctness proof can make the approach infeasible for larger circuits. Secondly, the semantic gap between the intuitive, informal, specification the designer has in mind and the specification used in the symbolic simulator is often quite large.

When tabulating the strengths and weaknesses of theorem-proving and symbolic simulation used for formal hardware verification, it is striking to see how well the two approaches complement each other. Thus, it is very appealing to attempt to integrate them into a two-level combined approach to formal hardware verification. However, in order to achieve this integration, two problems need to be resolved: 1) a mathematically precise interface must be developed so that the rigor of the formal proof is not jeopardized, and 2) a practical interface between the two processes must to be developed. In this paper we focus on the first issue and only briefly mention ongoing work towards solving the second problem.

2 A Two-Level Approach

Symbolic simulation, as achieved by the COSMOS simulator, can be viewed as a highly specialized form of theorem-proving. COSMOS checks the validity of assertions in a specification language (which we call *CL*) with respect to a model structure Ψ. This model structure is a set of infinite state sequences determined by an extracted circuit netlist C and a built-in switch-level and delay model of circuit behavior. When viewed as a theorem-proving system, the COSMOS system can be used to prove theorems of the form, $\Psi \models \mathtt{f}$, where $\mathtt{f}$ is a formula in *CL*.

A rigorous link with general-purpose theorem-proving, in particular, the Cambridge HOL system, is achieved by semantically embedding the specification language *CL* in higher-order logic. The semantic embedding of *CL* in higher-order logic allows *CL* specifications to be expanded into a term of higher-order logic and used to derive higher-level correctness results. That is, the HOL system can be used to prove theorems of the form, $\vdash \mathtt{f} \Longrightarrow \mathtt{t}$, where $\mathtt{f}$ is a formula in *CL* (embedded in higher-order logic) and $\mathtt{t}$ is a term in higher-order logic.

Thus, the two proof results, $\Psi \models \mathtt{f}$ and $\vdash \mathtt{f} \Longrightarrow \mathtt{t}$ constitute a statement of correctness in our combined approach. They are obtained by symbolic simulation and general-purpose theorem-proving respectively.

3 Symbolic Simulation Viewed as Theorem-Proving

The main thesis of this section is that the verification system described in [6], based on the COSMOS symbolic simulator, can be viewed as a proof system. We use the

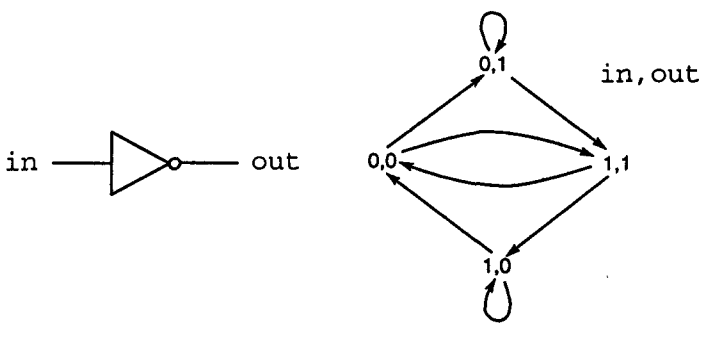

Figure 1: Inverter and corresponding state machine.

simple example of an inverter, shown in Fig. 1(a), to provide the reader with an informal account of our approach. If we assume a binary circuit model and a unit delay simulator, the inverter circuit is accurately described by the state machine shown in Fig. 1(b). The states of the machine are labelled with the current values of the two nodes in the circuit, and a transition in the state machine corresponds to a basic unit of time.

The state machine in Fig. 1(b) implies certain properties. For example, it is easy to see that we can conclude from the state machine that the value on the output is always the complement of the value that was present on the input one time unit ago. Informally, this could be written as:

$$\text{for every state sequence} \quad [(\mathrm{in} = a) \implies \mathbf{X}(\mathrm{out} = \overline{a})]$$

where the **X** is a "next time" operator and $\overline{a}$ denotes the Boolean complement of a. The main result of [6] is that the COSMOS symbolic simulator can be used to prove this kind of statement. In other words, the COSMOS system can be used to prove that the behaviors derived from an extracted netlist, using a sophisticated switch-level and timing model, implies certain formulas described in a logic with precisely defined semantics.

3.1 Circuit State Machine and Trajectories

The circuit model used in [6] is a ternary model, i.e., nodes in the circuit can take on the values 0, 1, and X. The *circuit state machine* corresponding to some circuit $\mathcal{C}$ is a non-deterministic finite state machine $\mathcal{M} = (S, \Delta, \Theta)$, where S is a finite set of states, Δ, the *transition relation*, is a relation on S, and Θ is a function $\Theta: S \rightarrow \{0, 1, X\}^n$ relating every state in S to an assignment of 0s, 1s, and Xs to the nodes of the circuit. Intuitively, if the circuit currently is in the state s^i and $(s^i, s^{i+1}) \in \Delta$, then the circuit can be in the state s^{i+1} one basic time unit later. The circuit state machine is determined by three factors: 1) the extracted netlist, 2) the switch-level model, and 3) the delay model.

Given a circuit state machine, a *state trajectory*, χ, is an infinite sequence of states $s^1, s^2, \ldots$, such that $s^i \in S$ and $(s^i, s^{i+1}) \in \Delta$ for $i \geq 1$. The *circuit trajectory*, $\psi(\chi)$, corresponding to a state trajectory χ is an infinite sequence of ternary state vectors $a^1, a^2, \ldots$, such that $a^i \in \{0, 1, X\}^n$ and $a^i = \Theta(s^i)$ for $i \geq 1$. Informally, a circuit trajectory can be viewed as an infinite sequence of "snap-shots" of the operating circuit taken every unit of time. Finally, let Ψ denote the set of all possible circuit trajectories for a given circuit. Intuitively, Ψ can be viewed as the set of all possible circuit behaviors according to the switch-level and delay model used.

3.2 Logic *CL*

The logic *CL* is defined in terms of another logic called *CL'*. We begin by describing *CL'* and then consider *CL*.

The logic *CL'* is defined over a set of nodes, $\mathcal{N} = \{n_1, \ldots, n_n\}$, and over a set of symbolic Boolean variables, $\mathcal{V}$. The formulas consists of constants (UNCONST), atomic propositions ($n_i = 1$ and $n_i = 0$), conjunction ($\mathtt{f}_1 \wedge \mathtt{f}_2$), case restriction ($e \rightarrow \mathtt{f}$), and next time operations ($\mathbf{X}\mathtt{f}$). In case restriction, ($e \rightarrow \mathtt{f}$), e is a Boolean expression over $\mathcal{V}$ and f is a *CL'* formula. The basic idea is to use a Boolean function to limit the cases for which the *CL'* formula f is of interest.

Let $\mathcal{V}$ be a set of symbolic Boolean variables. An *interpretation*, ϕ, is a function $\phi: \mathcal{V} \rightarrow \mathcal{B}$ assigning a binary value to each symbolic Boolean variable. Let Φ be the set of all possible interpretations, i.e., $\Phi = \{\phi: \mathcal{V} \rightarrow \mathcal{B}\}$.

The truth semantics of a *CL'* formula f is defined relative to an interpretation $\phi \in \Phi$ and a circuit trajectory $\psi = a^1, a^2, a^3, \ldots \in \Psi$. For a precise definition of the truth semantics, see [6]. Informally, the *CL'* formula UNCONST holds for every ϕ and ψ. The formula $n_i = 0$ ($n_i = 1$) holds if and only if $a_i^1 = 0$ ($a_i^1 = 1$). The conjunction of two *CL'* formulas holds if and only if both formulas hold. The *CL'* formula $e \rightarrow \mathtt{f}$ holds if either the Boolean formula denoted by the Boolean expression e evaluates to 0 for interpretation ϕ, or if the *CL'* formula f holds. Finally, $\mathbf{X}\mathtt{f}$ holds for ϕ and circuit trajectory $a^1, a^2, a^3 \ldots \in \Psi$ if and only if f holds for ϕ and circuit trajectory $a^2, a^3 \ldots$.

The verification methodology used by the COSMOS system entails proving *assertions* about the model structure. These assertions, written in the *core logic CL*, are of the form $\mathtt{A} \implies \mathtt{C}$, where the *antecedent* A and the *consequent* C are *CL'* formulas over $\mathcal{N}$ and $\mathcal{V}$. This assertion is true, written $\Psi \models (\mathtt{A} \implies \mathtt{C})$, if and only if for every interpretation, i.e., every assignment of 0s and 1s to the symbolic Boolean variables, and for every possible circuit trajectory, the *CL'* formula C holds or the *CL'* formula A does not hold.

3.3 A Decision Algorithm

A decision algorithm based on ternary symbolic simulation was given in [6] for determining the validity of formulae in *CL*. That is, the algorithm determines whether or not for every interpretation every circuit trajectory satisfying the antecedent A must also satisfy the consequent C. It does this by generating a symbolic simulation sequence corresponding to the antecedent, and testing whether the resulting symbolic state sequence satisfies the consequent. For details, see [6].

4 Semantically Embedding *CL* in Higher-Order Logic

As argued in [14], a major advantage of higher-order logic as a formalism for verifying hardware is the ability to semantically embed more specialized formalisms into this logic. This often results in more concise specifications and easier proofs. Furthermore, as this paper demonstrates, the ability to semantically embed another formalism in higher-order logic, in particular *CL*, provides a means of establishing a rigorous link between general-purpose theorem-provers, such as HOL, and other verification tools, such as COSMOS.

Although full details, including machine-readable syntax, are beyond the scope of this paper, a sketch of how *CL* can be embedded in higher-order logic is given below. As explained earlier, the truth semantics of a CL' formulae is relative to an interpretation ϕ and a circuit trajectory ψ. Thus, operators of CL' are defined as functions of ϕ and ψ. An interpretation ϕ is represented as a function that maps Boolean expressions to Boolean values. Circuit trajectories are also represented by functions—in this case, functions that map position in a sequence to state vectors.

$\vdash_{def}$ UNCONST = $\lambda\ \phi\ \psi.$ true

$\vdash_{def}$ $n\_i = 0$ = $\lambda\ \phi\ \psi.$ (head ψ)[i] = 0

$\vdash_{def}$ $n\_i = 1$ = $\lambda\ \phi\ \psi.$ (head ψ)[i] = 1

$\vdash_{def}$ f1 $\wedge$ f2 = $\lambda\ \phi\ \psi.$ (f1 $\phi\ \psi$) $\wedge$ (f2 $\phi\ \psi$)

$\vdash_{def}$ e $\rightarrow$ f = $\lambda\ \phi\ \psi.$ (ϕ(e)) $\Longrightarrow$ (f $\phi\ \psi$)

$\vdash_{def}$ **X**f = $\lambda\ \phi\ \psi.$ f ϕ (tail(ψ))

$\vdash_{def}$ $\Psi \models$ (A $\Longrightarrow$ B) = $\forall\phi \in \Phi, \forall\psi \in \Psi.$ (A $\phi\ \psi$) $\Longrightarrow$ (C $\phi\ \psi$)

To avoid a proliferation of symbols in this informal account, we have used the same symbol for conjunction and implication in both CL' and higher-order logic, namely, $\wedge$ and $\Longrightarrow$ respectively.

5 An Example

To illustrate our two-level approach to formal hardware verification, consider the circuit shown in Fig. 2. This is a 16-bit instance of a (pseudo) domino-logic design for a circuit that tests whether: 1) input A is greater than input B and, 2) input

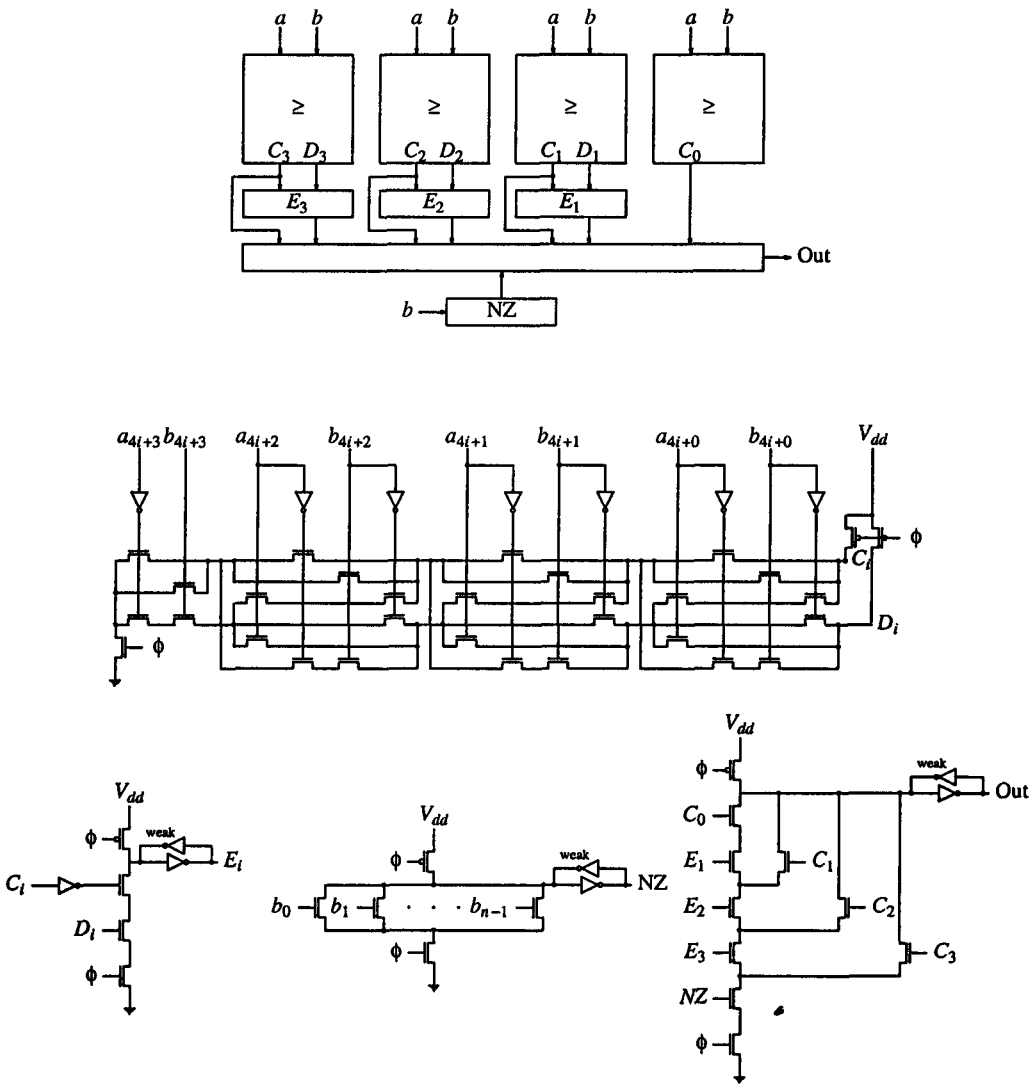

Figure 2: 16-bit circuit for computing $A > B > 0$.

B is greater than zero, when these inputs are interpreted as the unsigned binary representation of two numbers. The goal of formal verification is to relate a top-level specification of this circuit's intended function to a bottom-level specification of its implementation (based on an underlying model of hardware). The top-level specification should be sufficiently abstract to minimize the semantic gap between it and the informal, intuitive, specification of the circuit that resides in the mind of the designer. On the other hand, the bottom-level specification should be an accurate model of the circuit. This includes not only an accurate electrical model but also temporal properties of the circuit.

In the mind of the human specifier, the intended function of the circuit shown in Fig. 2 is intuitively understood in terms of an arithmetic relation, i.e., "the output should be 1 when A is greater than B and B is greater than 0". To minimize the semantic gap, the top-level formal specification should also be stated in terms of an arithmetic relation. At the bottom-level of specification, the actual operation of the circuit shown in Fig. 2 cannot be accurately described by a simple model of circuit behavior. A number of detailed features such as clocking, charge storage, charge sharing, and sized transistors, need to be included in an accurate model of this circuit. Hence, the verification problem, in this particular case, is to relate a top-level specification expressed in terms of an arithmetic relation to a bottom-level specification based on a detailed model of switch level circuit behavior.

Neither symbolic simulation or theorem-proving on their own is able to satisfactorily deal with this verification problem. Symbolic simulation would clearly be unable to support a top-level specification stated in terms of arithmetic relations. Theorem-proving is generally inappropriate for reasoning about detailed circuit behavior. Below, we will outline how this proof could be carried out using our combined verification approach. However, due to space considerations, we will not include the actual code representing the specification to the COSMOS system nor the actual HOL statements.

At the COSMOS level, we first define a function *CmpBitLevel* that takes a size parameter n and two vectors of Boolean variables A and B and returns the Boolean expression representing the bit level "compare and greater than zero operation". We then define a function *timing* that takes a Boolean expression `res` as argument and that essentially describes the timing conditions under which we wish to verify the circuit in Fig. 2. To paraphrase this definition: on the assumption that, 1) the clock signal `phi` is low for 100 time units and then is high for another 100 time units, and 2) the vectors of circuit nodes `a` and `b` are assigned the vectors of symbolic Boolean variables `A` and `B` at time 95 and held stable until time 200, then the circuit node denoted by `out` should be equal to the value `res` from at least time 180 until time 200.

The functional specification expressed by *CmpBitLevel* and the timing conditions expressed by *timing* are combined in the top level COSMOS specification: (*timing* (*CmpBitLevel n A B*));.

The COSMOS system is able to derive the following theorem which states that the above specification is a logical consequence of the finite state machine derived from the extracted netlist of the circuit shown in Fig. 2.

$$\Psi \models (\mathit{timing}\ (\mathit{CmpBitLevel}\ 16\ A\ B))$$

We now wish to derive a more abstract correctness result which expresses correctness at the arithmetic level. Currently, we hand-translate the COSMOS specifications into higher-order logic. Eventually, when the interface language is semantically embedded in higher-order logic, this translation will be a series of expansion steps governed by the inference rules of higher-order logic.

To formally establish a relationship between a bit level correctness result and a higher level correctness result expressed in terms of natural number arithmetic, we need to formally define a relationship between bit vectors and natural numbers. This is expressed by a definition called *BitsToNum* which is a data abstraction function that maps bit vectors to natural numbers. We also define the function *CmpNumLevel* which is the arithmetic level specification of the function we want the circuit in Fig. 2 to compute.

The HOL system can now be used to prove that $\forall n\ A\ B$:

$$(\mathit{CmpBitLevel}\ n\ A\ B) \equiv (\mathit{CmpNumLevel}\ (\mathit{BitsToNum}\ n\ A)(\mathit{BitsToNum}\ n\ B)).$$

Having established this equivalence, we can then derive a generalized correctness result which relates the bit level specification of the compare circuit to an arithmetic level specification for any value of n, i.e., we can show that $\forall n$:

$$(\mathit{timing}\ (\mathit{CmpBitLevel}\ n\ A\ B)) \Longrightarrow (\mathit{timing}\ (\mathit{CmpNumLevel}\ (\mathit{BitsToNum}\ n\ A)(\mathit{BitsToNum}\ n\ B)))$$

Finally, we instantiate this generalized result for $n = 16$ to obtain,

$$(\mathit{timing}\ (\mathit{CmpBitLevel}\ 16\ A\ B)) \Longrightarrow (\mathit{timing}\ (\mathit{CmpNumLevel}\ (\mathit{BitsToNum}\ 16\ A)(\mathit{BitsToNum}\ 16\ B)))$$

which, together with the symbolic simulation result above constitutes a statement of correctness for the circuit in Fig. 2.

6 Conclusions

Different methods of formal verification involve tradeoffs between automation, flexibility, expressibility, and accuracy. We conclude that a promising balance of these tradeoffs can be achieved by using theorem-proving at higher levels and symbolic simulation at lower levels. By embedding the "high-level" specification logic used by COSMOS into HOL, we are able to efficiently verify systems from a very detailed electrical and timing domain up to a very abstract behavioral domain.

We think that this two-level approach will be particularly useful in the case of circuits where there is tight coupling between functional and temporal properties of the circuit level and high level abstractions, e.g., when a gate level or RTL abstraction is not available as an intermediate level. This is especially true in the case of high performance designs. Also, by integrating these two methods, we open up the possibility of verifying mixed software/hardware systems[1, 15].

We are currently in the process of formalizing the interface logic and implementing a compiler for this language in the COSMOS system. This involves not only modifying the existing, informal, compiler in COSMOS, but also to define the precise semantics of the language and proving the correctness of the compilation method.

References

[1] W. Bevier, W. Hunt, J Moore, and W. Young, "An Approach to Systems Verification", *Journal of Automated Reasoning*, Vol. 5, No. 4, November 1989.

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

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

[4] R.E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. on Computers* Vol. C-33, No. 2, February, 1984, pp. 160–177.

[5] R.E. Bryant, "Symbolic Verification of MOS Circuits", *1985 Chapel Hill Conference on VLSI*, May, 1985, pp. 419-438.

[6] R.E. Bryant, and C-J. Seger, "Formal Verification of Digital Circuits Using Symbolic Ternary System Models", *DIMAC Workshop on Computer-Aided Verification*, Rutgers, New Jersey, June 18-20, 1990 (to appear in Springer Verlag's Lecture Notes in Computer Science).

[7] Albert John Camilleri, "Mechanizing CSP Trace Theory in Higher Order Logic", *IEEE Transactions on Software Engineering*, Vol. SE-16, No. 9, September 1990, pp. 993-1104.

[8] Paolo Camurati and Paolo Prinetto, "Formal Verification of Hardware Correctness", *IEEE Computer*, Vol. 21, No. 7, July 1988, pp. 8-19.

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

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

[11] Roger W. S. Hale, *Programming in Temporal Logic*, Ph.D. Thesis, Report No. 173, Computer Laboratory, Cambridge University, July 1989.

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

[13] Jeffrey J. Joyce, "A Verified Compiler for a Verified Microprocessor", Report No. 167, Computer Laboratory, Cambridge University, March 1989.

[14] Jeffrey J. Joyce, "More Reasons Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware", in: P. Subrahmanyam, ed., *Proceedings of a Workshop on Formal Methods in VLSI Design*, 9-11 January 1991, Miami, Florida.

[15] Jeffrey J. Joyce, "Totally Verified Systems: Linking Verified Software to Verified Hardware", in: *Specification, Verification and Synthesis: Mathematical Aspects*, Proceedings of a Workshop, 5-7 July 1989, M. Leeser and G. Brown, eds., Ithaca, N.Y., Springer-Verlag, 1989.

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