Distinguishing Formulas for Free

Tiziana Margaria* Bernhard Steffen

Lehrstuhl für Informatik II
RWTH Aachen
5100 Aachen
Germany

tel: +49 241 8021.201
fax: +49 241 8021.229

{*tiziana,bus*}@*informatik.rwth-aachen.de*

Aachener Informatikberichte
Nr. 92 – 23

Abstract

A system for the efficient verification of the input/output correctness of Finite State Machines with data path and control unit is presented. This system, which is based on First-Order Logic theorem proving, is unique in automatically providing distinguishing formulas expressing all test patterns that witness a behavioural difference between two descriptions. Experimental results illustrate the test pattern generation feature for stuck-at faults, functional faults, and initialization faults, and the efficiency which results from the compositionality of the verification.

Keywords: High Level Design Tools - design verification.

*This work has been supported by the Graduiertenkolleg at the RWTH Aachen, Germany.

1 Introduction

Traditional symbolic verification methods for Finite State Machines of the complexity of a microprocessor either separately address the data path and the control unit (eg. [CMPr91, NKFT90, Vemu89]), or only reduced subsets of Register Transfer level languages [Fili91, BEFP89]. To extend the verification domain to higher level specifications and to bridge the gap towards high-level testing more flexible approaches are required.

In this paper we present a general purpose verification system, which is unique in

- capturing circuit descriptions that contain both gate level and Register Transfer level primitives, by uniformly interpreting them in terms of First-Order Logic formulas. This will enable us to reduce the proof of input/output correctness to consistency checking in a First-Order Logic, which can be done automatically by means of theorem proving,
- automatically providing distinguishing formulas expressing all (sequences of) input patterns that witness a difference between two descriptions. These formulas, which arise as a byproduct during the verification, allow to generate test patterns in linear-time. The approach addresses a wide class of faults: besides the traditional gate-level stuck-at faults, also faults in the initialization as well as high-level functional faults are considered.

Experimental results with the system illustrate the test pattern generation, and the efficiency which results from compositional verification. This allows us to deal with huge Finite State Machines. As an example, the 256-bit counter considered in Section 3 has $2^{256} \approx 10^{80}$ states. Our verification system is the only general purpose system that can deal with circuits of that size.

Besides yielding efficiency, our approach enables *mixed-mode* verification, i.e., each description may contain both behavioural and structural portions.

Our verification system is based on OTTER, a resolution-based theorem prover for First-Order Logic with equality [McCu90]. It proves its goal by *refutation.* Proofs proceed in three steps: first, the semantic expressions for the specification and the implementation are rewritten to yield functions of primary inputs only. Second, they are simplified and canonicalized in a Binary Decision Diagram-like structure [Brya86]. Third, the proof by refutation is executed mainly by rewriting, which guarantees efficiency. Weighting and ordering strategies help choosing the best rewriting strategy, while a user defined lexical ordering on symbols supports the canonicalization of expressions.

Related Work: Current hardware verification systems can be classified in

- very efficient BDD-based *special purpose* tools, like e.g. the COSMOS symbolic simulator [Brya89, FiBr89] or the SIAM industrial system [CoBM89, MaBi88], which are integrated in the hardware design cycle, but in general capture only a specific abstraction level and/or a specific class of verification problems, and in
- *general purpose* tools, including systems based on First-Order Logic, like the Boyer-Moore Prover [BoMo88] or the Larch Prover [GaGu91] which support automatic induction, or on Higher-Order Logics, like e.g. the Nuprl system [Cons86] or the HOL System [Gord88]. These very ambitious tools are tailored for sophisticated reasoning at higher abstraction levels, however they are too inefficient for simpler goals like the ones considered here. Moreover they require extensive user interaction and a deep knowledge of the underlying theory, which restricts the number of possible users.

Our system can be regarded as a compromise between the two categories: it is general purpose, e.g. it uniformly captures arbitrary abstraction levels, it naturally supports hierarchical proofs, and it is efficient. Its surprisingly good performance in hardware verification is mainly due to the simple conceptual structure of our system, which offers a variety of optimized basic proof techniques, from which efficient application-specific proof strategies can be built. Moreover, the proofs are completely automatic. Thus the system can directly be used by designers.

After presenting the verification framework in Section 2, Section 3 illustrates the impact of hierarchical verification and the derivation of distinguishing formulas by means of examples.

2 The semantic framework

Our notion of correctness of a circuit's implementation with respect to a specification is based on the following notion of behavioural equivalence, which establishes our specification-implementation relation:

> *Two descriptions D_1 and D_2, with boolean primary inputs $\overline{x} = (x_1, \ldots, x_n)$ and $\overline{y} = (y_1, \ldots, y_n)$, respectively, are called* behaviourally equivalent, $D_1 \approx^B D_2$, *iff, starting from equivalent initial states, they produce the same output whenever the input variables are assigned equally.*

The verification method we propose is based upon the idea that the behavioural equivalence between descriptions can be characterized in a specific derived theory of First Order Logic. This theory is given by the combination of the axiomatizations for the two descriptions, together with the *thesis*, i.e. a formalization of the claim that both descriptions are behaviourally equivalent. Figure 1 illustrates the verification method.

Altogether, verifications in our framework proceed in three steps: i) axiomatizing the specification and implementation descriptions in First-Order Logic, ii) enriching the axiomatizations of the two description by adding equations that formalize input/output equivalence, and iii) computing a normal form of the thesis.

Once the semantic extraction criteria have been individuated, the first two steps are simple mappings from the HDL descriptions into the corresponding logic formulas, and are achieved by translation. The third step can then be carried out entirely automatically by means of OTTER.

2.1 Semantic Extraction

The semantics of hardware descriptions is given in terms of First-Order theories, i.e., as axiomatizations in First-Order Logic with equality similarly to [Evek85]. These axiomatizations are composed of an *equational* characterization of all the HDL language constructs that express the semantics of hardware specific function and predicate symbols, and of some additional equations characterizing the specific properties of the description under consideration.

The axiomatization of an HDL consists of an axiomatization of *First-Order Logic* with equality, together with three sets of equations characterizing the constructs of the *gate level*, the *RT level* and the *specific HDL* [Marg91]:

1. First-Order Logic with equality: its axiomatization is predefined in OTTER,
2. gate level semantics: it is defined on the basis of a *boolean algebra*, a *time model*, and a *substitution mechanism* by pattern matching to generate boolean expressions from circuit equations. The access to past values of signals models sequential behaviour (either

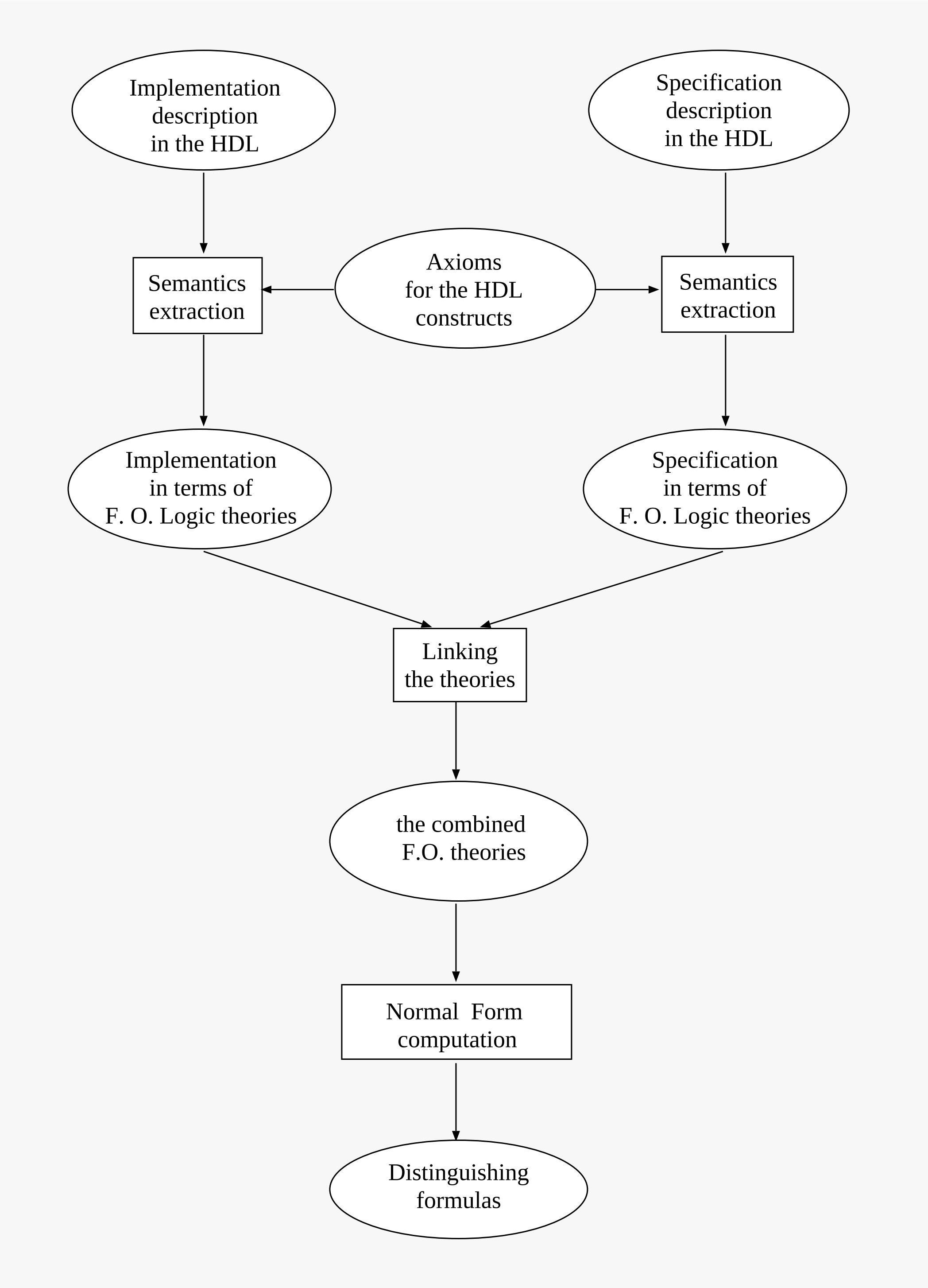

Figure 1: The verification process

asynchronous, as pure before/after relation, or synchronous, with reference to a clocking scheme).

3. **RT level semantics**: it is defined as a compositional extension of the gate level semantics. To represent registers, memories, and PLA's, polymorphic *list selectors* and *constructors* are introduced. *Conditional* primitives express control statements.

4. **HDL-specific theories**: they are derived by associating a First-Order Logic formula to each language primitive, in order to express its intended semantics in terms of the previously defined set of gate and RT level primitives.

The four portions of the axiomatization for an HDL specify the meaning of disjoint sets of constructs. Thus the consistency of the whole axiomatization follows from their individual consistency. This can be proved incrementally. In fact, the consistency of the first three portions has been proved formally once and forever by means of OTTER.

We also proved our axiomatization of the RT level subset of VHDL [SaWa88] to be consistent. Of course, our approach can also be applied to other hardware description languages. In particular, in [Marg91] the consistency of the axiomatization of $ART*$ [PrGa85], a more expressive Register Transfer level nonprocedural HDL has been shown.

The axiomatization of a description consists of additional equations that characterize the behaviour of the corresponding circuit. Each description statement is interpreted as a nested composition of the semantic primitives of the HDL. The resulting equation set consists of boolean operators, list operators, accessors to past values and nested if constructs. This axiomatization can be derived automatically by translation into the logic [Marg91].

Once the logical theories are expressed in terms of the syntax of OTTER, the proof, which only involves the semantic level, can be done entirely automatically.

2.2 The Proof Method

In order to establish the link between a pair of theories representing the two descriptions, it suffices to constrain them to share the same input variables and to relate their initial states. The requirement that the desired equivalence relation must then hold for all input configurations (handled symbolically), constitutes the goal (or thesis).

Given a circuit with n inputs and m outputs, the axiomatizations of its specification Spec and implementation Impl, and denoting the output functions for the i-th output by $\mathcal{O}_{Spec_i}(\overline{x})$ and $\mathcal{O}_{Impl_i}(\overline{y})$, the thesis *consists of m equations of the form*

$$\mathcal{O}_{Spec_i}[\overline{v}/\overline{x}] \;\oplus\; \mathcal{O}_{Impl_i}[\overline{v}/\overline{y}] \;=\; 0$$

where $\oplus$ denotes the boolean difference function and $\overline{v}$ is a symbolic input vector. We will abbreviate the i-th boolean difference expression by $\mathcal{T}_i$.

This system of equations, which will constitute the set of support for the *set of support* optimization strategy [ChLe73], establishes the desired link between the two theories and combines them into a global theory.

The thesis defines a rewriting system, when regarded as a set of (left-to-right) directed equalities between terms that are Boolean functions represented in if-then-else form. This rewriting system is *confluent*, and, after limiting the application of the commutative laws with respect to some ordering on the set of symbols, it is also *terminating*. Thus the following two-steps rewriting process,

1. all boolean operators are moved top-down on the syntax tree towards the leaves

2. common subexpressions are then pruned bottom-up,

always delivers a unique normal form, which can be automatically transformed in linear time into a Binary Decision Diagram (BDD)[Brya86]. Thus we obtain that

$$D_{Spec} \approx^{B} D_{Impl} \quad \text{iff} \quad \mathcal{T}_i \quad \text{is rewritten to} \quad 0 \quad \text{for all } i.$$

This reduces the verification procedure to the computation of the normal form of for the $\mathcal{T}_i$'s, which can be done efficiently by pure rewriting.

In contrast to other hardware verification systems, our system efficiently provides a useful form of *diagnostic information* without any additional computation effort whenever the specification and the implementation behave differently.

Theorem 2.1 (Distinguishing formulas)

$BDD(\mathcal{T}_i)$ characterizes the complete set of counterexamples for the input/output equivalence at output i. In particular, it reduces to the single node "0" iff $D_{Spec} \approx^B D_{Impl}$.

Since we consider input/output equivalence, and start from an RT level specification but also allow gate level portions in the descriptions, any inconsistency that may arise at the RT (or gate) level can be revealed by means of a distinguishing formula. This includes the traditional single stuck-at faults on wires and incorrect initializations, but also more complex faults, like wrong or missing components and incorrect wirings, which are functional faults one would like to detect already at the RT level.

The distinguishing information is given in BDD format, whose minimality guarantees:

Corollary 2.2 (Test Pattern Generation)

$BDD(\mathcal{T}_i(\overline{v}))$ allows us to obtain each distinguishing pattern in $O(|n|)$ time, where n is the number of variables.

3 Application

To demonstrate the dramatic impact of hierarchical verification and to illustrate the distinguishing formulas, a counter with enable and synchronous clear has been chosen.

The *specification* defines the behaviour of the circuit in terms of its input/output function. It specifies a control part, which selects one of the four possible operations (*clear, parallel_load, increment, no_op*) according to the values of the control signals, and a data path consisting of a data register with synchronous clear.

The RT level *implementation* is a cascaded composition of 4-bit modules, each corresponding to a TTL SN74LS163 synchronous up-counter (see Fig. 2). Its semantic description simply relates the calls to instances of the functional blocks to their (structural or behavioural) body.

3.1 Compositional verification

The combination of *syntactic data abstraction*, which avoids slicing, thus simplifying data path expressions, and *compositionality in the semantics*, which allows a mixed-mode verification, where some structural portions are included at need in a basically behavioural semantics has a dramatic impact on the performance ([Marg92]). This compositional technique for the verification of an implementation wrt. a specification has now been extended to hierarchies of verifications.

Considering counters of increasing word size, the impact of structuring RT-level proofs in a hierarchical way is shown by comparing two verification strategies:

- In a **flat strategy**, the behavioural description is directly compared with a totally flattened RT-level structural description. Its components are cascaded 4-bit modules, viewed as atomic units.

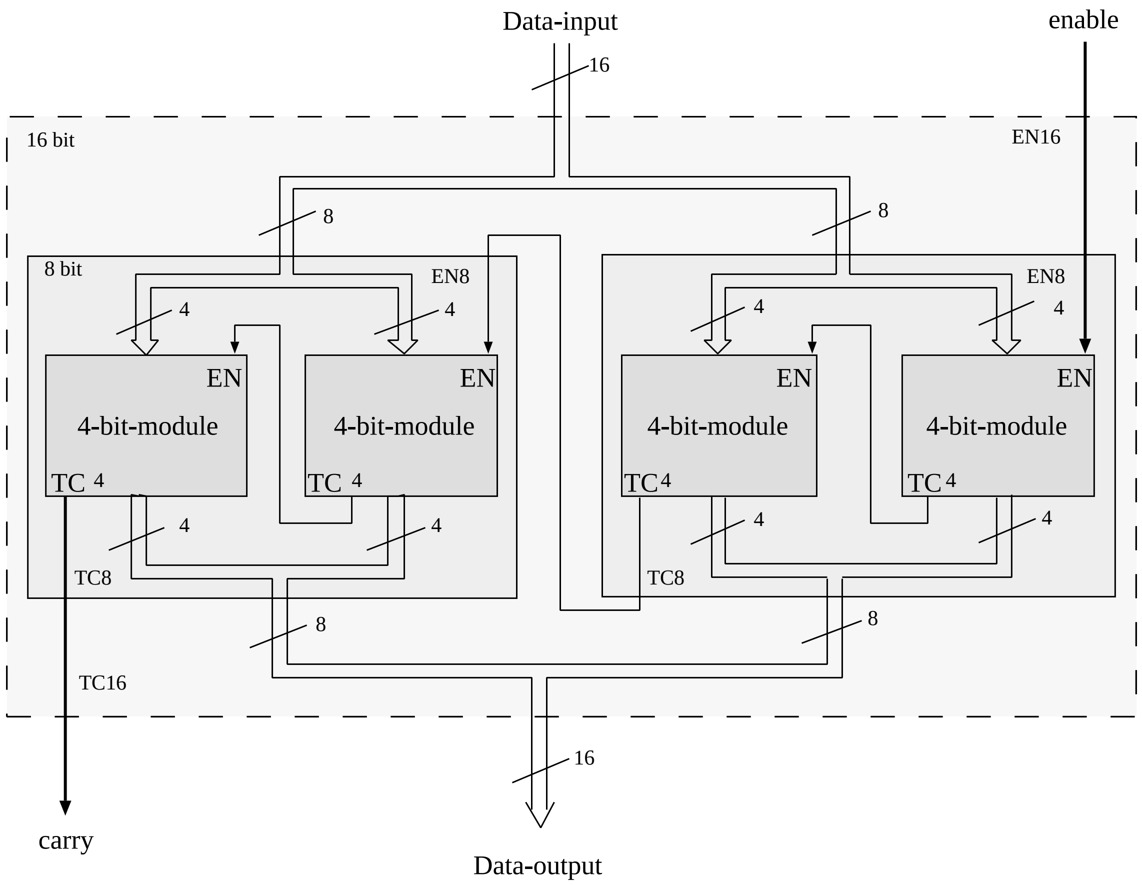

Figure 2: Hierarchical implementation of the counter

- In a **hierarchical strategy**, the structure of the description is exploited during the proof. Here we consider a hierarchy of 2n-bit counters, each composed of only two instances of an n-bit module, as pictured in Fig.2.

Results. Fig. 3 illustrates the performances on a *SUN SparcStation 1+* with 64 MB memory. They include the cost of the gate level verification of the 4-bit module (1.07s), which is negligible for all but the 8-bit counter.

The hierarchical approach avoids the exponential blowup in the number of states, and exibits a *linear* time complexity. This way, a circuit with $2^{256} \approx 10^{80}$ reachable states has been verified in a completely symbolic way. In contrast, the flat approach is exponential in the number of modules. Note that most verification tools must follow the flat strategy even for hierarchical designs.

3.2 Distinguishing formulas and Test Pattern Generation

The diagnostic capabilities of our system are illustrated wrt. three categories of faults:

- *stuck-at* faults on wires, the standard fault model for gate-level testing,
- *functional* faults, like the absence or erroneous substitution of a component, and
- *initialization* faults, which may be due to incorrect reset conditions.

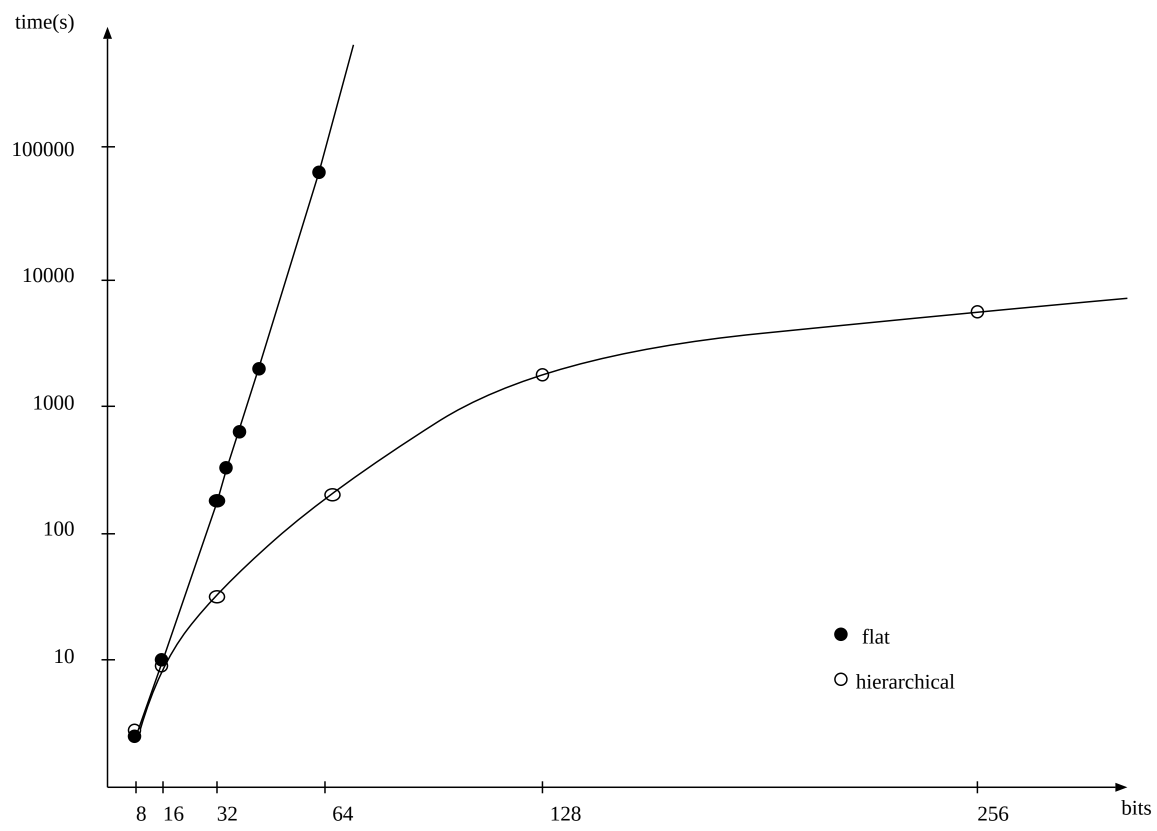

Figure 3: Comparative results

In our method, the last two classes of faults can be revealed either directly on the RT level description, or (more commonly) on the gate level structural implementation, by comparison with a (RT or gate level) description. Thus the test pattern generation becomes fully mixed-mode.

We apply the method to the 4-bit counter module here. In all three cases we observe the resulting behaviour on the 4-th data output $out_d(\overline{v})$.

Stuck-at faults. Here we assume that the $xLoad$ control input is stuck at 1 in the structural description. Instead of $L0(\overline{x}) = not(xLoad)$ we have now $L0(\overline{x}) = 0$.

The distinguishing formula shown in Fig. 4 characterizes in normal form the complete set of test vectors for the fault: every path that leads to 1 in the BDD reveals the difference of the behaviour of the two descriptions at output out_d. On the graph, x_i indicates the current input value on line i and $q_i(t) = x_i(t-1)$. Note that the signals $xClear$ and $xLoad$ are active low. As an example, the sequence of input patterns

$$[xClear, xLoad, xEnt, xEnp, x_d, x_c, x_b, x_a] \quad = < \quad \begin{array}{l} [-,-,-,-,1,1,1,1], \\ [\ 1,0,1,1,1,-,-,-] > \end{array}$$

corresponding to the highlighted path on the BDD, is a test vector (of length 2), while $<[0,-,-,-,-,-,-,-]>$, which corresponds to the path $xClear = 0$, executed during the *clear* operation, is not.

Functional faults. The *absence* of an inverter on the same $L0$ line can *not* be handled by standard testing methods. We assume now $L0(\overline{x}) = xLoad$.

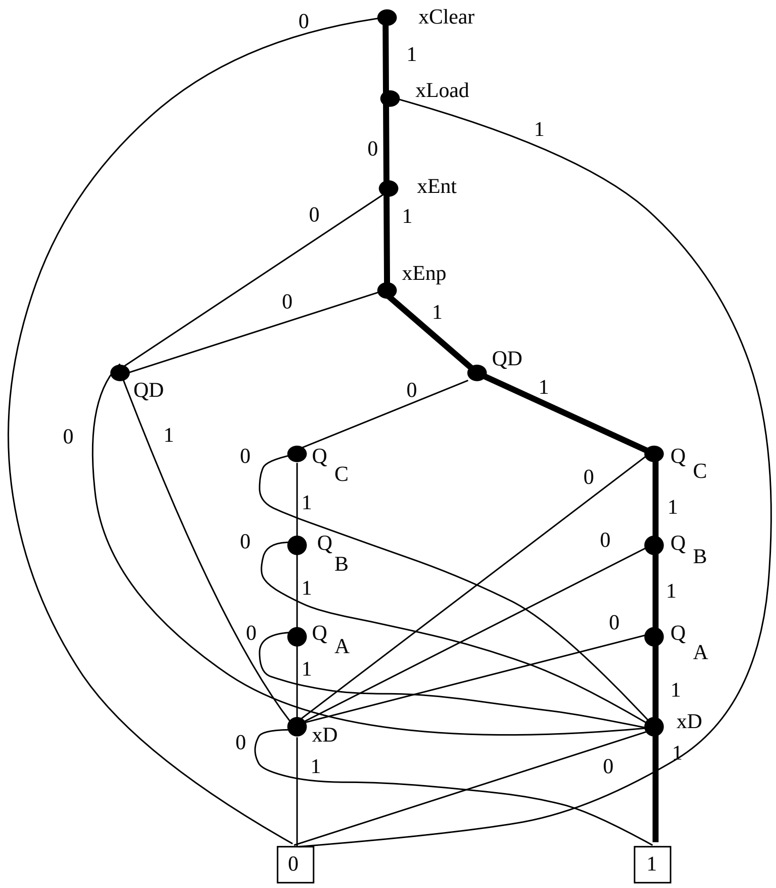

Figure 4: BDD of the distinguishing formula for fault $LO'1$.

The distinguishing formula shown in Fig. 5 characterizes in normal form the complete set of test vectors for this functional fault. It has been obtained in exactly the same way as the formula for the traditional stuck-at fault. As an example, the sequence

$$\begin{array}{lcl}[xClear, xLoad, xEnt, xEnp, x_d, x_c, x_b, x_a] & = < & [-,-,-,-,1,-,-,-], \\ & & [\ 1,-,0,-,0,-,-,-] > \end{array}$$

is a test pattern of length 2, and there is no shorter test pattern.

3.2.1 Initialization faults

The symbolic state variable Q_D is initialized to x in the specification, but, erroneously, to its complement in the implementation. Fig. 6 shows that the specification is only met when the initial configuration is overridden via a reset (clear) or a preset (load).

Reset and *preset faults* are handled exactly in the same way.

4 Conclusions

We have presented a system for the efficient verification of complex FSMs which is based on a compositional semantics encompassing both gate and RT abstraction levels in terms of First-Order logic formulas. Our verification methodology efficiently exploits the hierarchy and structural regularity of a design. As a consequence, traditionally exponential problems can be solved in linear time. Moreover, the method automatically provides distinguishing formulas which ex-

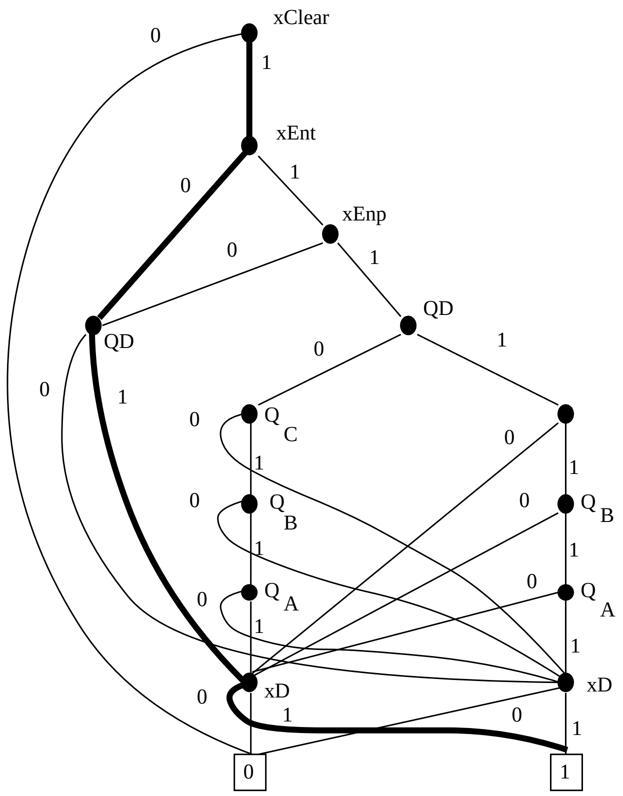

Figure 5: BDD of the distinguishing formula for the functional fault.

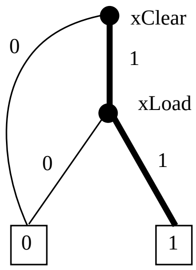

Figure 6: BDD of the distinguishing formula for an initialization fault.

press all test patterns that witness a behavioural difference between two descriptions. Besides covering the standard stuck-at faults, the method also addresses functional faults and initialization faults. A parallel implementation of our system, exploiting the independency of outputs, is under development.

References

[BEFP89] A. Bratsch, H. Eveking et al. : "LOVERT - A logic verifier of register-transfer level descriptions," IMEC-IFIP WG 10.2/10.5 Worksh. on "Applied Formal Methods For Correct VLSI Design", Leuven (Belgium), Nov. 1989, pp. 522-531.

[Brya86] R.E. Bryant: "Graph-based algorithms for boolean function manipulation," IEEE Trans. Computing, vol. C-35(8), August 1986, pp. 677-691.

[Brya89] R.E. Bryant: "Verification of synchronous circuits by symbolic logic simulation," Worksh. on Hardware Specification, Verification and Synthesis: Mathematical Aspects, LNCS 408, Springer Verlag, 1989, pp. 14-24.

[BoMo88] R.S. Boyer, J.S. Moore: "A Computational Logic Handbook," Academic Press, New York, 1988.

[ChLe73] C.L. Chang, R.C. Lee: "Symbolic logic and mechanical theorem proving," Academic Press, Inc., 1973.

[CMPr91] P.Camurati, T.Margaria, P.Prinetto: "Formal verification of design correctness of sequential circuits based on theorem provers," IEEE CompEuro'91: Advanced Computer Technology, Reliable Systems and Applications, Bologna (Italy), May 1991, pp. 322-326.

[CoBM89] O. Coudert, C. Berthet, J.C. Madre: "Verification of synchronous sequential machines based on symbolic execution," Workshop on Automatic Verification Methods for Finite State Systems, Grenoble (France), June 1989.

[Cons86] R. L. Constable et al.: "Implementing Mathematics with the Nuprl Proof Development System," Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Evek85] H. Eveking: "The application of CHDL's to the abstract specification of hardware," IFIP 7th Int. Symp. on Computer Hardware Description Languages and their Applications, Tokyo (Japan), August 1985, pp. 167-177.

[FiBr89] A.L. Fisher, R.E. Bryant: "Performance of COSMOS on the IFIP workshop benchmarks," IMEC-IFIP WG 10.2/10.5 Worksh. on "Applied Formal Methods For Correct VLSI Design", Leuven (B), Nov. 1989, pp. 595-599.

[Fili91] I. Filippenko: "VHDL Verification in the State Delta verification System (SDVS)," Proc. Int. Workshop on Formal Methods in VLSI Design, Miami, FL, (USA), ACM, January 1991.

[GaGu91] S.J. Garland, J.V. Guttag: "A guide to LP, the Larch Prover," Digital Equipment Corp. Systems Research Center Rep. n. 82, December 1991.

[Gord88] M. Gordon: "HOL: a proof generating system for Higher-Order Logic," in "VLSI Specification, Verification and Synthesis", G. Birtwistle, P. Subrahmanyam eds., Kluwer, 1988.

[Lang90] M. Langevin: "Automated RTL verification based on predicate calculus," Int. Worksh. on "Computer-Aided Verification", DIMACS Series in Discrete Mathematics and Th. Computer Science, AMS, 1990, pp. 577-590.

[MaBi88] J.C. Madre, J.P. Billon: "Proving circuit correctness using formal comparison between expected and extracted behaviour," DAC-25: 25th ACM/IEEE Design Automation Conference, Anaheim, CA (USA), June 1988, pp. 211-217.

[Marg91] T. Margaria: "First-Order theories for the verification of complex FSMs," Aachener Informatik-Berichte Nr.91-30, RWTH-Aachen, Dec. 1991.

[Marg92] T. Margaria: "Efficient RT-Level Verification by Theorem Proving", IFIP World Congr.'92, Madrid (Spain), Sept. 1992, North-Holland pp. 696-702.

[McCu90] W.W. McCune: "OTTER 2.0 User's Guide," Rep. ANL-90/9 Mathematics and Computer Science Division, Argonne Nat. Lab., Argonne, IL (USA).

[NKFT90] H. Nakamura, Y. Kukimoto, M. Fujita, H. Tanaka: "A data path verifier for Register Transfer level using Temporal Logic language TOKIO," Int. Workshop on "Computer-Aided Verification", DIMACS Series in Discrete Mathematics and Th. Computer Science, Vol.3, AMS, 1990, pp. 493-504.

[PrGa85] P.Prinetto, S. Gai: "ART∗: user manual," Int. Rep. DAI/CAD n.14/85, Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino (Italy), October 1985.

[SaWa88] "IEEE Standard VHDL Reference Manual," L. Saunders, R. Waxman eds., IEEE, 1988.

[Vemu89] R. Vemuri: "A formal model for Register-Transfer level structures and its applications in verification and synthesis," IMEC-IFIP Workshop on "Applied Formal Methods For Correct VLSI Design", Leuven (Belgium), November 1989, Vol. 2, pp. 77-96.