Verication of SwitchLevel Designs with ManyValued Logic

Reiner Hähnle* and Werner Kernig

Institute for Logic, Complexity and Deduction Systems,
Dept. of Computer Science, University of Karlsruhe, Germany
{haehnle,kernig}@ira.uka.de

Abstract. This paper is an approach to automated verication of circuits represented as switchlevel designs. Switchlevel models (SLM) are a wellestablished framework for modelling lowlevel properties of circuits. We use manyvalued propositional logic to represent a suitable variant of SLM. Logical properties of circuits (gatelevel) can be expressed in a standard way in the same logic. As a result we can express soundness of switchlevel designs wrt to gatelevel specications as manyvalued deduction problems. Recent advances in manyvalued theoremproving indicate that it is possible to handle real life examples. We report rst results obtained with an experimental theorem prover.

Introduction

Switchlevel models[1] (SLM) are well-established tools for representing a circuit on the transistor level in considerable detail. They can be used to model phenomena like propagation and resolution of undened values, hazard detection, degradation effects, varying capacities, pull-up transistors or depletion mode transistors; see [4] for a very exhaustive list. It is important to note, however, that all dimensions are symbolic values.

Traditionally, SLM have been used as a formal basis for the construction of simulation tools which can be used for testing the behaviour of a circuit before it is actually built. In this paper we present an approach to automated verication on the switch level based on automated deduction in propositional manyvalued logics. Related work was done by Bryant & Seger [1] and Büttner et al. [3]. The relationship between these approaches and our own is discussed in the conclusion.

Propositional manyvalued logics are particularly wellsuited for representing SLM. Given the fact that typical SLM of real circuits contain several hundreds of parts, there is no sense in trying full rstorder predicate logic. Also the expressivity of full rstorder predicate logic is not really needed. On the other hand, mere twovalued propositional logic is not advisable, too, since one has to introduce lots of auxiliary variables that bear no natural meaning, such that the logical representation of SLM would become unreadable and, more important, unfeasible through its many variables. We found that propositional manyvalued logics are just the right tool for adequate representation of

* Research supported by IBM Germany and Deutsche Forschungsgemeinschaft.

[1] Switchlevel models were introduced in the late 70s mainly by Bryant [2] and Hayes [10, 11] as a formal framework for modelling lowlevel properties of circuits.

SLM if truth values are interpreted as different levels of voltage. Together with a logical representation of the intended function of a circuit it becomes possible to establish its soundness in terms of a formal proof in manyvalued logic.

Recent research showed that it is well possible to build generic satisability checkers for propositional manyvalued logics that are quite efcient [5, 6, 8]. The logical basis for manyvalued deduction here is a suitable extension of analytic tableaux [12]. Moreover, manyvalued tableaux give rise to a reduction of manyvalued deduction problems to integer programming problems [7]. First results suggest that with this technique it is possible to build manyvalued satisability checking programs whose performance comes close to stateoftheart satisability checkers for classical logic.

We are condent that ultimately it is possible to prove properties of real life circuits with our approach. In this paper we sketch the rst steps towards this task. The organization of the paper is as follows: in Section 1 we describe the variant of switchlevel model we used for our experiments. In Section 2 we introduce a manyvalued deduction framework based on analytic tableaux. Section 3 presents the link between SLM and logic in terms of a translation from the former into the latter. This is illustrated by an example. Finally, we give statistical gures from rst tests, we discuss related work and we point out the next steps.

1 The Verication Model

1.1 SwitchLevel Representation of Circuits

In SLM we represent a MOS circuit as a set of nodes which are interconnected by switches (transistors). In this paper we consider only combinational and asychronous circuits with zerodelay elements.

Denition 1 SwitchLevel Network. A switchlevel network consists of a set $\mathbf{N}$ of nodes $\{n_1, \ldots, n_k\}$ which are interconnected by a set $\mathbf{T}$ of transistors $\{t_1, \ldots, t_l\}$. A subset $\mathbf{Q}$ of $\mathbf{N}$ is called the set of source nodes and the set $\mathbf{S} = \mathbf{N} - \mathbf{Q}$ is called the set of storage nodes.

Denition 2 MOSTransistor. A MOS-transistor is a bidirectional switch in a switch-level network which has three terminals called gate, source and drain.

A NMOStransistor is closed (conducting) iff the voltage at its gate terminal represents a logical `1`. The transistor is open (nonconducting) iff the voltage at its gate terminal represents a logical `0`. In all other cases, the state of the transistor is called unknown.

For a PMOStransistor, the conditions for the gate terminal are ipped.

From now on we use the more general term value instead of voltage to describe the state of the nodes (and the terminals of a transistor are nothing else than nodes).

We follow [11, 4] by using a sevenvalued logic in order to model one particular switch level phenomenon, namely degradation effects in CMOS circuits which occur due to the fact that transistors constitute nonideal switches that degrade the strength of the signals.

Denition 3 SwitchLevel Value. The set of switchlevel values (SLVs) consists of the elements E, D0, D1, DU, S0, S1, SU. These are the only possible values at the nodes of a switchlevel network. The meaning of these SLVs is as follows:

The strong values S1 and S0 are the values associated with the support voltage vdd and ground gnd.
D1 and D0 are values which result of a degradation effect, that is, the property that a closed transistor passes on the voltage of his drain (source) terminal diminished to his source (drain) terminal.
SU and DU are undened values, but correspond to a certain strength.
E is the value of all nodes which are not connected to a source node via a path through the network.

Remark. If we consider circuits in ratioed logic we can split the degraded values further to distinguish between depletion transistors and normal transistors.

To compute the value of a node in the network we rely on the # operator introduced by Hayes [10]. The semantics of the operator # corresponds to the computation of the supremum in a lattice if we order the SLVs as shown in Figure 1. For an exhaustive treatment of this topic see [10, 11].

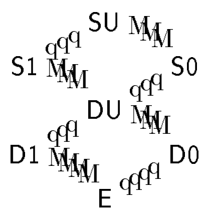

Fig. 1. SLV lattice.

Compared to SLM used in real simulation tools our model seems a bit simplicistic, however, it should be clear that with more truth values we can achieve a much more negrained model. To keep this paper readable we have taken the simplest SLM which is nontrivial.

1.2 GateLevel Specication of Circuits

The gatelevel specication of circuits is well known and we restrict ourselves to a short denition.

Denition 4 Gate. A gate is the smallest undividable switch element for the processing of binary signals. It is an unidirectional element which computes according to n inputs an output. Common gates are: AND, OR, NOT, NAND, NOR and XOR.

In other words, gates realize (certain) Boolean functions.

Denition 5 Gate-Level Network. A gatelevel network is a directed graph whose nodes are gates.

The modelling of digital systems exclusively with gate networks is regarded as unsufcient for several reasons:

1. circuits in ratioed logic cannot be modelled properly at the gate level.
2. the analysis of circuits on the gate level is too far from the actual layout of the circuit. Connections on the chip, for instance, the connection with vdd cannot be represented with the gate model.

3. gates are unidirectional elements.
4. a onetoone transformation of circuits described with gates to circuits constructed with transistors does usually not result in an efcient implementation of the desired function.

1.3 Vertical and Horizontal Verication

In general, the meaning of verication is the (formal) proof of a certain property, for example, the correctness of a hardware system. Since the construction of a hardware system is done by several design steps in which the designers lay down what the system should do and how it should do it, we have several tasks for verication.

We call the design of what the system should do the specication and the design of how it should do it the implementation.

One possible mode of verication is the proof that a single design level (specication or implementation) is correct in itself, for example, that a specication at the gate level does not produce hazards or similar kinds of errors. In a switchlevel design we can verify that only proper (dened) voltage levels occur at the output nodes provided that the voltages at the input nodes are proper. This kind of verication we call horizontal verication, a commonly used term for verications concerning only one design level.

In contrast we can also perform vertical verication which includes two levels of the design hierarchy. For instance, we can prove soundness of a gatelevel specication with respect to a switchlevel implementation. A complete stratication ranging from the physical level to highlevel functional properties of complex circuits would incorporate many other formalisms than propositional logic. The type of formal systems used is determined by the complexity of the circuit and the kind of properties which are to be veried. For functional verication of a CPU, for instance, rstorder or even higher order logic may be required. The drawback of these more complicated formalisms is that they are not amenable to full automatization. If the amount of automatization within the whole verication task is to be maximized, it is crucial that at each level the most adequate formalism is used. For the switch level this is manyvalued (temporal) propositional logic.

In the eld of hardware verication several meanings of the term verication are in common use: simulation, complete testing and formal verication. Our understanding is formal verication which means that the verication is mathematical and not experimental. Correctness is understood in this paper as a mathematical relation between two entities, for example, a specication/implementation pair. Formal verication allows a general proposition in contrary to simulation, where we can only prove the presence of bugs, but never their absence. Correctness as a relation can be classied as follows:

equality: $\mathtt{S} = \mathtt{I}$
equivalence: $\mathtt{S} \leftrightarrow \mathtt{I}$
logical implication: $\mathtt{S} \supset \mathtt{I}$
reverse logical implication: $\mathtt{I} \supset \mathtt{S}$
homomorphism: $\mathtt{M}(\phi(\mathtt{S})) = \mathtt{M}(\mathtt{I})$

We choose in the following the logical implication $\mathtt{I} \supset \mathtt{S}$ which denotes that a specication S is a behavioural abstraction, in other words, the formal verication that a switchlevel network realizes the same function as a given gatelevel network.

For a more detailed survey on formal verication of hardware correctness see [4].

2 Automatic Proof Search in ManyValued Logic

In this section we give a very brief introduction into the logical formalism underlying our verication approach. For more details we refer the reader to [5, 8].

2.1 ManyValued Logic

Denition 6 Syntax, Truth Values. Let L be a propositional language with propositional variables L_0 and connectives F. Let N be the set of truth values, for deniteness take equidistant rational numbers, i.e. $N = \left\{0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1\right\}$ and dene n to be the cardinality of N.

Denition 7 Semantics, ManyValued Logic. Connectives $F \in \mathbf{F}$ are interpreted as functions with nite range and domain, in other words, if k is the arity of F we associate a function $f : N^k \to N$ with F which we call the interpretation of F. Let f be the family of functions over N associated with connectives in F. Then we call f $n$valued matrix for L and the triple $\langle L, \mathbf{f}, N\rangle$ $n$valued propositional logic.

In practice we take always the same symbols for f and F.

Denition 8 Valuation. Let $\mathcal{L} = \langle L, \mathbf{f}, N\rangle$ be a $n$valued propositional logic. A valuation for $\mathcal{L}$ is a function $v : L_0 \to N$. As usual, v can be uniquely extended to a homomorphism from L to N via

$$v(F(\phi_1, \ldots, \phi_k)) = f(v(\phi_1), \ldots, v(\phi_k))$$

where f is the interpretation of F.

Denition 9 SSatisable, STautology. For $S \subseteq N$ and a $n$valued propositional logic $\mathcal{L}$ call a formula $\phi \in L$ Ssatisable iff there is a valuation such that $v(\phi) \in S$. Call ϕ a Stautology iff ϕ $v(\phi) \in S$ for all valuations.

2.2 Analytic Tableaux

Analytic Tableaux are a sound and complete proof procedure for classical rstorder predicate logic introduced in the 50s (a standard reference is [12]). The version we present here is modied for the efcient treatment of manyvalued logics, cf. [5, 8].

Denition 10 Signed Formula. Let $\phi \in L$, $S \subseteq N$. Then we call the expression $S : \phi$ signed formula. The set of signed formulas is denoted with L^*.

Signed formulas are a device for talking about manyvalued logics with only two truth values on the metalevel. In [5] we introduced systematically truth value sets as signs in order to achieve an adequate representation of the manyvalued search space. We coined this 'setsassigns' approach. In [8] it is demonstrated that using setsassigns in some way is crucial for the efciency of any manyvalued proof procedure.

Analytic tableaux are a refutation procedure. For our purposes it is sufcient to visualize a tableau proof as a nite labelled tree, whose node labels are signed formulas. To proof validity of a formula $S : \phi$ we begin with a tableau whose single node is labelled with the complement: $(N - S) : \phi$. Now this formula is analysed following its syntactic structure in a topdown manner to the atomic level. If we arrive at a contradiction in any case we have proved that no valuation can satisfy the root, in other words, ϕ is a Stautology. Rather than giving the formal denitions we illustrate the process with a small example from classical logic.

Example 1. We prove that the formula $(p \supset \neg p) \supset \neg p$ is a classical tautology, that is, $\{1\} : (p \supset \neg p) \supset \neg p$ holds. The initial tableau consists of the complemented theorem $\{0\} : (p \supset \neg p) \supset \neg p$.

Next, we analyze the truth conditions for $\{0\}$ and the top connective $\supset$. For each combination of signs and connectives there is a rule that characterizes it. In the present case we need the following rules:

$$\frac{\{1\}\ \phi \supset \psi}{\{0\} : \phi \,|\, \{1\} : \psi} \qquad\qquad \frac{\{0\} : \phi \supset \psi}{\begin{array}{c}\{1\} : \phi \\ \{0\} : \psi\end{array}}$$

Rule application to a formula $S : \rho$ in the tree means that we can append to any of the paths containing it as many new branches as there are extensions in the conclusion of the rule whose premise matches $S : \rho$. The new branches contain the formulas from the rule extensions. In our example we apply rst the rule on the right and then on the rst of the resulting formulas the rule on the left. Formulas within the same branch are conjunctively connected while formulas in different branches are disjunctively connected. We notice that each branch in the example contains a complementary pair of formulas, that is, $S_1 : \phi$, $S_2 : \phi$ with $S_1 \cap S_2 = \emptyset$. Such branches are called closed. A tableau represents a proof iff all its branches are closed.

$$\begin{tikzcd}[column sep=small]
 & \{0\} : (p \supset \neg p) \supset \neg p \arrow[d, no head] & \\
 & \{1\} : p \supset \neg p & \\
 & \{0\} : \neg p \arrow[d, no head] & \\
 & \{1\} : p \arrow[dl, no head] \arrow[dr, no head] & \\
\{0\} : p & & \{1\} : \neg p
\end{tikzcd}$$

The extension of this framework to manyvalued logics is more or less straight-forward. To prove that ϕ is a Stautology we simply construct a manyvalued tableau with root $(N - S) : \phi$. Manyvalued tableau rules can be stated very much like their twovalued counterparts. For instance, if we dene manyvalued conjunction as $i \wedge j = \min(i, j)$, where min is the natural minimum on N, we nd the following rule for $\{0, \frac{1}{2}\} : \phi \wedge \psi$ in threevalued logic:

$$\frac{\{0, \frac{1}{2}\} : \phi \wedge \psi}{\{0, \frac{1}{2}\} : \phi \,|\, \{0, \frac{1}{2}\} : \psi}$$

One difference between the twovalued and the manyvalued case is that in the latter more than two extensions in the rules may become necessary. Another important difference is the slightly more general notion of branch closure:

Denition 11 ManyValued Closure. A branch in a manyvalued tableau is closed iff (i) either it contains signed formulas $S_1 : \phi_1, \ldots, S_m : \phi_m$ such that $S_1 \cap \cdots \cap S_m = \emptyset$ or (ii) a single signed formula $S : F(\phi_1, \ldots, \phi_k)$ such that $\mathrm{rg}(f) \cap S = \emptyset$, where $rg(f) = \{i | i = f(j_1, \ldots, j_k);\ i, j_1, \ldots, j_k \in N\}$.

For some logics, including classical logic, $m = 2$ is sufcient for completeness and (ii) never occurs [6]. Then, of course, we have the old notion of closure.

Remark. It is in general not necessary to have all 2^n possible signs present to achieve a sound and complete system, see [8] for necessary conditions on the set of signs. On the other hand, the more signs are present, the fewer extensions the rules tend to have and, consequently, the shorter the proofs become.

Remark. Various improvements of analytic tableaux known from the twovalued case such as lemma generation, structure sharing, selection heuristics etc. carry over to the manyvalued case.

In [7] it is demonstrated that manyvalued tableaux (with a certain extension of the syntax) can be naturally translated into integer programming (IP) problems which can then be solved quite efciently with various algorithms. First results indicate that it is well possible to handle formulas with up to several hundred propositional variables and more than one thousand connectives that way.

3 Verication with ManyValued Logic

In this section we provide the connection between SLM and manyvalued logic. The basic idea is to treat switchlevel values as truth values and to represent nodes and transistors as manyvalued connectives. Thus we dene a manyvalued propositional logic $\mathcal{L}_{\mathrm{SLM}}$ called switchlevel logic as follows:

Denition 12 $\mathcal{L}_{\mathrm{SLM}}$. Let $\mathcal{L}_{\mathrm{SLM}}$ be the sevenvalued propositional logic with

truth values $N = \{\mathsf{E}, \mathsf{D0}, \mathsf{D1}, \mathsf{DU}, \mathsf{S0}, \mathsf{S1}, \mathsf{SU}\}$,
binary connectives
{#, `ntrs`, `ptrs`, `ntrd`, `ptrd`, AND, OR, XOR, NAND, NOR, `imp`, `spec`, `m_imp`},
unary connectives {`definite`, `vdd`, `gnd`, NOT}

and with the truth table semantics as given in Table 1 to 4.

The meaning of the connectives should be clear, for example, `ntrs` computes the value of the source terminal according to the given values at the gate and drain terminals.[2]

[2] `ptrs` and `ptrd`, as well as `ntrs` and `ntrd` have identical denitions in the current model. This is likely to change when models become more negrained. Moreover, the binary connectives suggest a unidirectional behaviour of transistors. A bidirectional model could be easily attained by taking ternary predicates `ptr(Gate,Source,Drain)` instead of binary ones (cf. [3]), however, we wanted to keep the denitions as simple as possible.

Table 1. Truth tables for `NOT`, `AND`, `OR`, `NAND`, `NOR` and `XOR`.

	`NOT`
0	1
1	0

`AND`	0	1
0	0	0
1	0	1

`OR`	0	1
0	0	1
1	1	1

`NAND`	0	1
0	1	1
1	1	0

`NOR`	0	1
0	1	0
1	0	0

`XOR`	0	1
0	0	1
1	1	0

Table 2. Truth tables for #, `ntrs` and `ptrs`. In the tables for the latter, rows correspond to the gate terminal and columns to drain.

#	E	D0	D1	DU	S0	S1	SU
E	E	D0	D1	DU	S0	S1	SU
D0	D0	D0	DU	DU	S0	S1	SU
D1	D1	DU	D1	DU	S0	S1	SU
DU	DU	DU	DU	DU	S0	S1	SU
S0	S0	S0	S0	S0	S0	SU	SU
S1	S1	S1	S1	S1	SU	S1	SU
SU	SU	SU	SU	SU	SU	SU	SU

`ntrs`	E	D0	D1	DU	S0	S1	SU
E	E	E	E	E	E	E	E
D0	E	E	E	E	E	E	E
D1	E	D0	D1	DU	S0	D1	SU
DU	E	DU	DU	DU	SU	DU	SU
S0	E	S1	E	E	E	E	E
S1	E	D0	D1	DU	S0	D1	SU
SU	E	DU	DU	DU	SU	DU	SU

`ptrs`	E	D0	D1	DU	S0	S1	SU
E	E	E	E	E	E	E	E
D0	E	D0	D1	DU	D0	S1	SU
D1	E	E	S1	E	E	E	E
DU	E	DU	DU	DU	DU	SU	SU
S0	E	D0	D1	DU	D0	S1	SU
S1	E	E	E	E	E	E	E
SU	E	DU	DU	DU	DU	SU	SU

`m_imp(I,S)` corresponds to $I \supset S$ and `imp(I,out)` is true iff the value of `I` equals the value of `out`.[3]

As can be seen, we have four kinds of connectives: Connectives associated with the gate level, connectives associated with the switch level (#, `ntrs`, `ptrs`, `ntrd`, `ptrd`), connectives used as a link between these two levels (`imp`, `spec`, `m_imp`) and connectives used for expressing facts at the switch level (`definite`, `vdd`, `gnd`) which have sevenvalued input and Boolean output.

In our rst approach only the connectives associated with the switch level have a sevenvalued semantic, whereas all others have a Boolean one. One can imagine our verication model consisting of several components, each with its own associated logic. Each logic can be embedded into the most general one with a suitable reinterpretation of the truth values. Hence, each of the component logics can be altered easily without

[3] `imp` stands for 'implements', not for 'implies' in contrast to `m_imp` which is material implication.

Table 3. Truth tables for `m_imp`, `imp` and `spec`.

`m_imp`	0	1
0	1	1
1	0	1

`imp`	0	1
0	1	0
1	0	1

`spec`	0	1
0	1	0
1	0	1

Table 4. Truth tables for `definite`, `vdd` and `gnd`.

	`definite`	`vdd`	`gnd`
E	0	0	0
D0	1	0	0
D1	1	0	0
DU	0	0	0
S0	1	0	1
S1	1	1	0
SU	0	0	0

changing the whole verication model. This can occur if, for example, one wants to analyze the gate level with a manyvalued logic. In $\mathcal{L}_{\mathrm{SLM}}$ we use only the SLVs as truth values, Boolean values are mapped to the SLVs. So the SLVs S1 and D1 are both mapped to the Boolean `1`, S0 and D0 are mapped to the Boolean `0` and for the other SLVs, the mapping is not dened.

Example 2 Interpretation of Truth Values. Take the signed formula $\{\mathsf{S1},\mathsf{D1}\} : \texttt{NOT}(\phi)$; its Boolean equivalent is $\{1\} : \texttt{NOT}(\phi)$. A signed formula $\{\mathsf{SU},\mathsf{DU}\} : \texttt{NOT}(\phi)$ never occurs during the proof procedure, because it has no Boolean equivalent and is therefore not generated by any of the rules.

Example 3 Tableau Rules, cf. Section 2.2. The upper left rule shown below is the one we always need for the initial tableau. The upper right rule is one of the rules for the # connective. The lower rule expresses the fact that the variable value has no undened or unknown value. This rule demonstrates the interconnection between the different logics: the premise has a Boolean semantic (it is true that value has a denite value), the conclusion has a 7-valued semantic (value has a truth value from the set {S1,S0,D1,D0}).

$$\frac{\{S0,D0\}:\texttt{m\_imp}(I,S)}{\begin{array}{c}\{S1,D1\}:I\\ \{S0,D0\}:S\end{array}} \qquad \frac{\{S1\}:\#(A,B)}{\begin{array}{c|c|c}\{S1\}:A & \{E,D1,D0,DU\}:A & \{S1\}:A\\ \{E,D1,D0,DU\}:B & \{S1\}:B & \{S1\}:B\end{array}}$$

$$\frac{\{S1,D1\}:\texttt{definite}(\mathrm{value})}{\{S1,S0,D1,D0\}:\mathrm{value}}$$

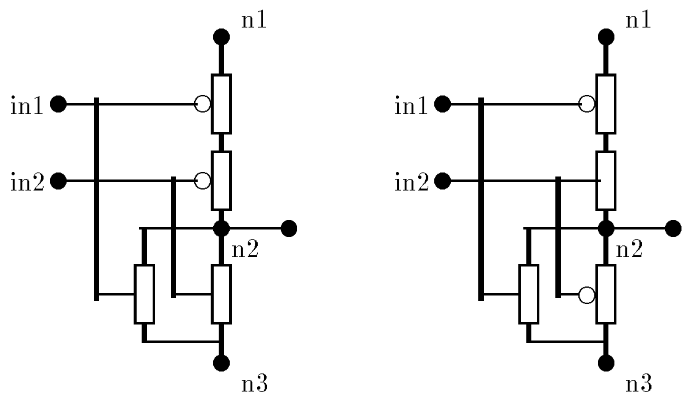

Fig. 2. A correct and an incorrect NOR implementation.

A simple example illustrates our ideas: Figure 2 shows a correct (on the left) and an incorrect (on the right) implementation of a NOR gate. We want to prove in the rst case

```
{S1,D1}m_imp(imp(#(#(ptrs(in2,ptrs(in1,n1)),
                    ntrd(in1,n3)),
                  ntrd(in2,n3)),out),
             spec(NOR(in1,in2),out))
```

and in the second case

```
{S1,D1}m_imp(imp(#(ntrs(in2,ptrs(in1,n1)),
                  #(ntrd(in1,n3),
                    ptrd(in2,n3)),out),
             spec(NOR(in1,in2),out))
```

In standard syntax this would amount to prove validity of

$$(f(in1, in2) \leftrightarrow out) \supset (g(in1, in2) \leftrightarrow out)$$

where $f(in1, in2)$ is the switchlevel design and $g(in1, in2)$ the gate level design of a circuit. The reasons not to use this notation are that (1) we wanted to use the same set of truth values for all levels and (2) the denitions of imp,spec etc are very likely to change when the verication model gets more negrained.

e start our proof procedure with the complemented theorem together with the following axioms:

```
{S1,D1} vdd(n1)                  {S1,D1} definit(in1)
{S1,D1} gnd(n3)                  {S1,D1} definit(in2)
```

we have our initial database for an automatic theorem prover. Figure 3 shows the rst two rule applications of the proof procedure to the initial tableau corresponding to the correct implementation.

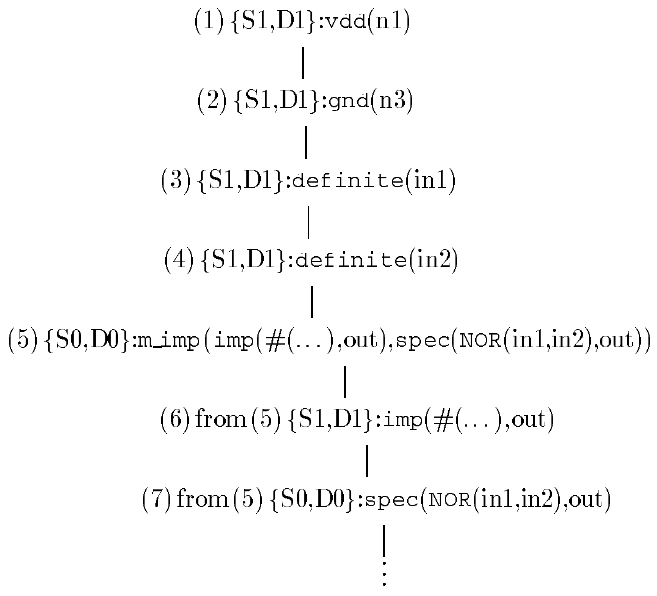
(1) {S1,D1}:vdd(n1)
(2) {S1,D1}:gnd(n3)
(3) {S1,D1}:definite(in1)
(4) {S1,D1}:definite(in2)
(5) {S0,D0}:m_imp(imp(#(. . .),out),spec(NOR(in1,in2),out))
(6) from (5) {S1,D1}:imp(#(. . .),out)
(7) from (5) {S0,D0}:spec(NOR(in1,in2),out)

Fig. 3. Tableau for verication of NOR after the rst two rule applications.

4 Conclusion

With an experimental tableaubased manyvalued theorem prover implemented in Prolog [9] we have veried the correct NOR implementation within 0.250 seconds and have shown the incorrectness of the second implementation within 0.37 seconds (on a SUN 4/75). Among the larger problems, we have veried a fulladder for two binary variables, with a specication consisting of 7 gates and an implementation using 24 transistors. There we have separated the computation of the sum and the computation of the carry. To verify the computation of the sum of two variables we need 18 seconds and for the verication of the carry computation, we need 5.6 seconds. Other experiments have shown that up to 30 transistors can be handled.

Sequential circuits (i.e., with feedback) can either be handled as in [3] using nite automata or by extending the manyvalued reasoner to a temporal model checker.

These gures seem not very impressive, however, they show that the approach is viable. As already noted recent experiments with a manyvalued propositional satisability checker based on integer programming techniques [7] showed that a speedup by a factor of several hundred may be obtained by using sophisticated implementation techniques. As demonstrated in [8] most inference techniques can be extended from classical to manyvalued logic in an efcient way. We expect that switchlevel circuits consisting of several hundered parts can be handled this way without modularizing the input.

The main contribution of this paper is to show that manyvalued theorem proving on the propositional level is a promising tool for hardware verication of SLM. This is the rst time that a genuine manyvalued theorem proving approach is used for this purpose. In [3] a similar verication problem is reduced to unication in a certain class of functionally complete nite algebras and from there to unication in Boolean algebras. In [8] it is pointed out that the possibility of handling a single class of functionally complete logics is not sufcient for manyvalued theorem proving in practice. Also the reduction of manyvalued logic to twovalued logic as done in [1] creates too much redundancy to be efcient. We argue that genuine manyvalued proof procedures based on 'setsassigns' [5, 8] and integer programming [7] are potentially more efcient and more general than reduction techniques.

The next steps would be to (i) implement a highperformance satisability checker for manyvalued logics, (ii) to develop manyvalued logics for a more negrained modelling of switchlevel designs, (iii) to verify larger circuits taken from real hardware designs and (iv) to extend the theorem prover for the treatment of sequential circuits.

References

1. R. E. Bryant and C.-J. H. Seger. Formal verication of digital circuits using symbolic ternary system models. In E. M. Clarke and R. P. Kurshan, editors, Computer-Aided Verication: Proc. of the 2nd International Conference CAV'90, pages 3343. Springer, 1991.
2. R. Y. Bryant. A switchlevel model and simulator for MOS digital systems. IEEE Transactions on Computers, C33:160 169, 1984.
3. W. Büttner, K. Estenfeld, R. Schmid, H.-A. Schneider, and E. Tidén. Symbolic constraint handling through unication in nite algebras. Applicable Algebra in Engineering, Communication and Computing, 1:97 118, 1990.
4. H. Eveking. Verikation digitaler Systeme:, eine Einführung in den Entwurf korrekter digitaler Systeme. LMI. Teubner, Stuttgart, 1991.
5. R. Hähnle. Towards an efcient tableau proof procedure for multiplevalued logics. In Proc. Workshop on Computer Science Logic, Heidelberg, pages 248 260. Springer, LNCS 533, 1990.
6. R. Hähnle. Uniform notation of tableaux rules for multiplevalued logics. In Proc. International Symposium on MultipleValued Logic, Victoria, pages 238 245. IEEE Press, 1991.
7. R. Hähnle. A new translation from deduction into integer programming. In Proc. Conf. on Articial Intelligence and Symbolic Mathematical Computations, Karlsruhe. Springer LNCS, 1992.
8. R. Hähnle. Automated Proof Search in MultipleValued Logics. Oxford University Press, forthcoming, 1993.
9. R. Hähnle, B. Beckert, S. Gerberding, and W. Kernig. The ManyValued TableauBased Theorem Prover $_3T^AP$. IWBS Report 227, Wissenschaftliches Zentrum Heidelberg, IWBS, IBM Deutschland, July 1992.
10. J. P. Hayes. A unied switching theory with applications to VLSI design. Proceedings of the IEEE, 70(10):1140 1151, October 1982.
11. J. P. Hayes. PseudoBoolean logic circuits. IEEE Transactions on Computers, C35(7):602 612, jul 1986.
12. R. Smullyan. FirstOrder Logic. Springer, New York, 1968.