Electronic Notes in Theoretical Computer Science 89 No. 4 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume89.html 15 pages

SAT-Based Methods for Sequential Hardware Equivalence Verification without Synchronization

Zurab Khasidashvili and Ziyad Hanna

Logic and Validation Technology
Design Technology Division
Intel Development Center, Haifa, Israel
{zurab.khasidashvili, ziyad.hanna}@intel.com

Abstract

The BDD- and SAT-based model checking and verification methods normally require an initial state. Here we are concerned with sequential hardware verification, where an initial state must be one of the reset states. In practice, a reset state is not always given by the designer, and computing a reset state of a circuit is a hard problem. In this paper we propose a method allowing usage of SAT-based verification methods without a need for a user-given or a computed initial state. The idea is to employ a binary encoding of 3-valued modeling of circuits, and use the undefined state X as a reset state.

1 Introduction

In the theory of Finite State Machines (FSM) [Koh78], one assumes an *initial* state (or a set of initial states), from which the machine starts operating. Here we will be concerned with sequential verification of synchronized hardware (circuit) models. In the practice of hardware verification, an initial state s_i of a circuit C is a state where all state elements (latches and flip-flops) have a binary value (T or F), and there is an *initializing sequence* π_i that brings C from the X state to that binary state s_i [CA89]. A *reset* or a *synchronization* sequence for C, on the other hand, is a sequence π_r that brings C from any *binary* state to a unique state s_r, called a *reset* or a *synchronization* state (π_r and s_r are independent from the state from which C starts operating) [Koh78]. Any initializing sequence is clearly a reset sequence, but the converse does not hold [CA89].

Classic BDD-based model checking and verification algorithms require a reset state [CBM89,CM90,TSLBS90,CCQ96,CC97,McM93]. The same is true for well known SAT-based model checking algorithms such as *Bounded Model Checking* [BCC99,BCCFZ99] or the *induction* method [SSS00]. Computation of reset sequences is a hard problem [CJSP93,PB94,PJH94,LP96,KBS96,CPRSS97,RH02].

Therefore in this work we are looking for verification methods that can work without a reset state.

Unlike SAT and BDD-based methods, the ATPG methods do not require a reset state [HCCG96,HCC96,HCC01]. There, one assumes the outputs to differ, and looks for a *justifying assignment*. The circuit modeling is *ternary* – besides the two binary values T and F, one considers an *unknown* value, X (elsewhere also denoted by $\perp$ or u). A justified assignment gives an input vector sequence that, if applied to the circuits starting at the unknown state X (or at *any* binary state), brings them to a state where their outputs differ.

In order to take advantage of the rapidly developing SAT-based verification technology, here we propose a SAT-based method for verifying 3-valued equivalence of sequential circuits without initialization. Our method is based on the *dual-rail* modeling of circuits, where every ternary value is represented with a pair of binary values (see [Bry87,BS94,SB95,KR03]). Via dual-rail encoding, we can arrive to ordinary (2-valued) propositional logic formulation of the verification problem.

The novelty of our approach is to show that the dual-rail X state can be used as a reset state in the (forward as well as backward) SAT-based algorithms mentioned above (the BMC and induction algorithms). We first present an algorithm for checking 3-valued equivalence which uses the X state as a reset state, and prove its correctness and completeness. We then discuss the applicability of our method to verification with respect to other concepts of sequential equivalence, such as *alignability* or *post-synchronization* equivalence [Pix92], and *steady-state* equivalence [KMH02].

The paper is structured as follows. In the next section, we quickly recall some basic definitions used in this work. In Section 3, we recall a backward ATPG based algorithm for verifying 3-valued equivalence [HCCG96,HCC96,HCC01] and explain its drawbacks. In Section 4, we give a light introduction to a binary encoding, called *dual-rail* encoding, of 3-valued logic into Boolean logic, originally developed for the purpose of efficient symbolic simulation and more direct modeling of circuit operation [BS94]. We also refer to more recent results on usage of the dual-rail encoding in SAT-based sequential verification [KR03]. In section 5, we propose a SAT-based method for 3-valued equivalence verification, and discuss how it relates to the ATPG algorithm mentioned above. In Section 6 we discuss how our method can be extended to steady-state and alignability sequential equivalence verification. Experimental results are discussed in Section 7. Conclusions appear in Section 8.

2 Preliminaries

Without restricting generality, we will assume that any circuit C has exactly one output, o. We denote by C_1 and C_2 our specification and implementation circuits (with outputs o_1 and o_2, respectively), and assume that they have the same set of inputs (dummy inputs can be added, if necessary). We denote by C_{xor} the combined circuit with shared inputs and XORed output $o = o_1 \, XOR \, o_2$. And we denote by

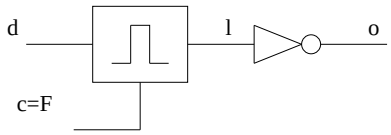

Fig. 1. Negated Latch

C_{xnor} the combined circuit (the *product machine* [HS98]) with shared inputs and XNORed output $o = o_1\, XNOR\, o_2$.

We consider *ternary* modeling of circuit node values. The values could be one of the *binary* values – T or F, or an *undefined* value – $\perp$ (elsewhere also denoted by X or u). Given a ternary (or binary) input vector sequence π, $n(s, \pi)$ will denote the value of node n of a circuit C after 3-valued simulation of C with π, starting at state s. Similarly, $C(s, \pi)$ denotes the (ternary) state into which π brings C, from state s.

A circuit C is specified as a collection of *next-state functions* (NSFs) of the latches as well as of the output. An NSF is a function of current and next-state values of inputs and latches. [1] This collection of NSFs defines a *sequential instance* corresponding to C, denoted $Inst(C)$. We denote by $Pins(C)$ the set of inputs, latches, and the output of C. Every pin variable p can be viewed as a sequence $(p[m])_{m\geq 0}$ of Boolean variables, each $p[m]$ representing value of pin p at phase m (thus the next state of $p[m]$ is $p[m+1]$).

Proof obligations can be added to an instance. They represent properties whose validity in all (relevant) phases we intend to check. The proof obligations we will be interested in are safety properties related to the validity of $o_{xnor} \Leftrightarrow T$.

Unrolling to depth k of the instance $Inst(C)$ yields a *combinational instance*, denoted $C[0, k]$, consisting of variables $\{p[i] \mid 0 \leq i \leq k, p \in Pins(C)\}$, and the relations on them are induced by the NSFs. The function of the output o in $C[0, k]$ at time phase k will be denoted by $o[0, k]$. We assume it is a partial function on all Boolean variables in the instance; partial because some value combinations are illegal as they contradict the NSF relations imposed on the instance. Alternatively, $o[0, k]$ can be seen as the conjunction of all NSF relations in $C[0, k]$.

Intuitively, falsification of a proof obligation expressing $o_1[k] \Leftrightarrow o_2[k]$ in $C[0, k]$ corresponds to k iterations of an ATPG procedure of finding a counter-example (CE) to the proof obligation $o_1 \Leftrightarrow o_2$. We will see in the later sections that this correspondence is not as tight as it may seem from the first sight.

The following example clarifies the above definitions.

Example 2.1 Consider a circuit C that consists of a negated latch l, with data d and clock which is always false: $c = F$ (see Figure 1). Let $o = \neg l$ denote the output. Then $Inst(C)$ consists of two NSFs: $l' = c'\&d' + \neg c'\&l = l$ and $o' = \neg l'$, where x' denotes next state value of x. Unrolling $Inst(C)$ to depth 1 yields combinational instance $C[0, 1]$ consisting of variables $o[0], o[1], l[0], l[1], d[0], d[1]$ and relations $o[i] = \neg l[i]$ for $i = 0, 1$, and $l[1] = l[0]$.

[1] Thus, the circuits that we consider are Mealy machines.

3 A backward ATPG based method for verification without initialization

Huang et al [HCC96,HCC01] developed an ATPG based method for verifying 3-*valued safe replacement* as well as *3-valued equivalence* for sequential circuits with or without an initial state. To define 3-valued equivalence, they introduced a *covering* relation on signals with ternary values: signal v_1 covers signal v_2 iff whenever $v_1 = T$ or $v_1 = F$, then $v_1 = v_2$.

Definition 3.1 • Circuit C_2 with output o_2 is called 3-*valued safe replacement* of circuit C_1 with output o_1 iff for any input sequence π, $o_1(\perp, \pi)$ *covers* $o_2(\perp, \pi)$. (That is, when o_1 has a binary value, then o_2 must have the same binary value.)

• Circuits C_1 and C_2 are *3-valued equivalent*, written $C_1 \cong_3 C_2$, iff for any input sequence π, $o_1(\perp, \pi) = o_2(\perp, \pi)$.

The values $\{(T,T),(F,F),(\perp,\perp)\}$ for the output pair (o_1, o_2) are called 3-*valued equal-pairs* of C_1 and C_2; in this case, C_{xnor} is in 3-*valued equal-state*. The remaining pairs $\{(T,F),(F,T),(\perp,T),(\perp,F)\},(T,\perp),(F,\perp)\}$ are called 3-*valued differ-pairs*, and C_{xnor} is in 3-*valued differ-state* in this case.

An input vector sequence π such that $(o_1(\perp,\pi), o_2(s_2,\pi)) \in \{(T,F),(F,T)\}$ is called a *partial test* for C_1 and C_2 in [HCC01] (this definition is not symmetric). Note that such a π brings C_{xnor} from state $\perp$ into a 3-valued differ-state.

To check for 3-valued safe replacement, the authors propose to use an ATPG solver in the following way:

The backward justification for the $o_{xor} = T$ (on C_{xor}) stops whenever one of the following two conditions is satisfied:

• *(Unjustifiable condition)*: All state requirements generated during the search of a partial test sequence are proven unjustifiable. Then C_2 is 3-valued safe replacement of C_1.

• *(Justified condition)*: A state requirement that does not have requirements on C_1 is reached. Then a partial test sequence has been found, and C_2 is not a 3-valued safe replacement of C_1.

Similarly, Huang et al [HCC96] proposed to disprove 3-valued equivalence by generating a state requirement that has no requirements on C_1 or on C_2; And to prove 3-valued equivalence by showing that all those state requirements that are generated while searching for a partial test for C_1 and C_2 and for a partial test for C_2 and C_1, are unjustifiable.

The above algorithm needs a termination criterion, based on some sort of 'diameter' or a fix-point, to be complete (not duscussed in [HCC01]). For example, let both C_1 and C_2 be negated latches, l_1 and l_2, with control F (like the circuit C in Example 2.1). Then $C_{xor}[0,k]$ will depend on variables $l_1[0]$ and $l_2[0]$ for any k, and neither of the two stopping conditions will ever be satisfied.

There is also another reason why the above algorithm is not complete: If an input vector sequence that can bring C_{xor} from X state to a differ state (with output

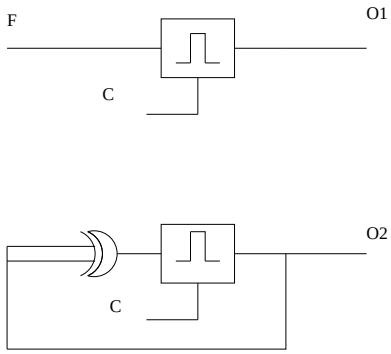

Fig. 2. 3-valued inequivalent circuits.

T) exists, a partial test for C_{xor} that the backward justification algorithm above is looking for may not exist:

Example 3.2 Consider two circuits C_1 and C_2 (see Figure 2), each consisting of a single latch with clock signal c, with pattern say $c = T, F, T, F, \ldots$. The input of the first latch is constant F, while the input of the other latch is $o_2\, XOR\, o_2$. Starting from the X state, o_1 behaves as $o_1 = X, F, F, \ldots$, and o_2 behaves as $o_2 = X, X, X, \ldots$. Thus these circuits are not 3-valued equivalent (and C_2 is not 3-valued safe replacement of C_1). However, o_{xor} can never become T in a non-0 phase (the only two concretizations of the sequence $o_2 = X, X, X, \ldots$ are $o_2 = F, F, F, \ldots$ and $o_2 = T, F, F, \ldots$), thus a partial test doesn't exist neither for C_1 and C_2 nor for C_2 and C_1.

Remark 3.3 *The above example was pointed out to us as a counter-example to (the sufficiency part of) Lemma 2 of [HCC01], which states that C_2 is a 3-valued safe replacement of C_1 iff there is no partial test for C_1 and C_2. While we believe the above example is not a counter-example to Lemma 2 of [HCC01]* [2] *, the correctness of the lemma does not affect the correctness of the above algorithm or our results below, and we will not elaborate on this issue further.*

Note that, intuitively, work with X values in a circuit corresponds to work with QBFs (Quantified Boolean Formulas): latch values are universally quantified in a predicate expressing a stop condition in the ATPG procedure above. Abdulla et al [ABE00] investigated ways to simplify QBF translation into quantifier free propositional formulae to facilitate SAT solvers on QBF, for the purpose of SAT-based model checking. Here we pursue a different path: To develop a SAT-based verification algorithm for 3-valued equivalence checking, we consider a dual-rail encoding of the ternary values. In the next section, we give a brief introduction to the subject. We will later explain why this approach can work well with certain SAT

[2] The authors state that they use an *enhanced* 3-valued logic simulation in Lemma 2; such simulation is based on approximating 3-valued simulation by 2-valued simulation. Note that the circuits C_1 and C_2 above cannot be distinguished if enhanced 3-valued logic simulation was used instead of the usual (conservative) one (see also Example 4.1). In the definitions however they use the usual 3-valued simulation.

solvers, and how it can be extended to verifying sequential equivalence without initialization with respect to other useful concepts of sequential equivalence.

4 Verification using dual-rail modeling of circuits

Dual-rail modeling of circuits was introduced by Bryant [Bry87]. It was used in [BS94] to enable a more precise modeling of circuit operation, and to enable representation of all *ternary* values with BDDs via a binary encoding. It resulted in a more efficient symbolic simulator, as more complex behaviors could be modeled with a single simulation run. We refer to [SB95,Jon02] for more information.

Each ternary value v is encoded as a pair of binary values (v_h, v_l), called the *high* and the *low* values. The undefined value $\perp$ is encoded as a pair $\perp = (T, T)$. The truth constants are encoded by $T = (T, F)$ and $F = (F, T)$. The pair $\top = (F, F)$ encodes a *contaminated* or *over-specified* value. To avoid any confusion, we use F^{dr}, T^{dr}, and $\perp^{dr}$ to denote the dual-rail encoding of T, F and $\perp$, respectively. And $v^{dr} = (v_h, v_l)$ will denote the dual-rail encoding of a ternary variable v.

Sequential logic can be expressed by using Boolean logic connectives such as $\&, +$, and $\neg$, and a *phase-delay* or *next state* operation, $'$. Thus in order to model sequential logic in dual-rail, it is enough to have dual-rail rules for these operations. We overload these logic connectives to denote the corresponding dual-rail counterparts as well. These dual-rail rules are as follows: Let $x^{dr} = (x_h, x_l)$ and $y^{dr} = (y_h, y_l)$ be dual-rail encoding of ternary variables x and y. Then

- $(x_h, x_l)\&(y_h, y_l) = (x_h \& y_h, x_l + y_l)$;
- $(x_h, x_l) + (y_h, y_l) = (x_h + y_h, x_l \& y_l)$;
- $\neg(x_h, x_l) = (x_l, x_h)$;
- $(x_h, x_l)' = (x'_h, x'_l)$.

Thus a dual-rail NSF is a pair of NSFs of the high and low values. We denote by $C^{dr}[0, k]$ the unrolled, to depth k, dual-rail sequential instance, and denote by $o^{dr}[0, k]$ the value of o in that instance (cf. definition of $o[0, k]$ in $C[0, k]$.

Example 4.1 Let us compute $x^{dr}\, XOR\, x^{dr}$ for $x^{dr} = \perp^{dr}$, as in Example 3.2:

$$\begin{aligned}
\perp^{dr} XOR \perp^{dr} &= ((T,T)\&\neg(T,T)) + (\neg(T,T)\&(T,T)) \\
&= ((T,T)\&(T,T)) + ((T,T)\&(T,T)) \\
&= (T\&T, T\&T) + (T\&T, T\&T) \\
&= (T,T) + (T,T) \\
&= (T+T, T+T) \\
&= \perp^{dr}.
\end{aligned}$$

We can see that dual-rail computation corresponds to usual 3-valued logic. [3]

[3] With enhanced 3-valued simulation as in the proof of Lemma 2 of [HCC01], we get $\perp XOR \perp =$

To ensure that in a sequential instance the inputs are always binary, one needs to add, for any input variable i, an assumption $i_h = \neg i_l$. This in particular will guarantee that we do not introduce (F, F) values in the instance. Further, if (F, F) values are not introduced in assumptions or in proof obligations, the NSFs cannot introduce them either (because the above four operations cannot result in an (F, F) value if the arguments are not over-constrained). Thus, for example, over-constrained values should not appear in a satisfying assignment found by a SAT-solver. An appearance of (F, F) in a satisfying assignment indicates a bug (in the design or in the tool), that is why we don't add to the instance an assumption forbidding over-constrained values on all variables.

We demonstrate dual-rail computation on another example:

Example 4.2 Let C be a circuit as in Example 2.1. Then $Inst(C)$ consists of four NSFs: $l'_h = (c'_h \& d'_h) + (c'_l \& l_h) = l_h$, $l'_l = (c'_l + d'_l) \& (c'_h + l_l) = l_l$, $o'_h = l'_l$, and $o'_l = l'_h$. Besides, we assume that d, as an input, is always binary, by adding $d_h = \neg d_l$ as an assumption to $Inst(C)$.

Dual-rail modeling is currently used in an alignability verification engine, Insight, in the formal verification group at Intel. Despite the double number of variables, experimental results show that the dual-rail implementation is $1.5x$ faster than a single-rail implementation based on the initialization flow reported in [RH02]. Among other factors, this is due to the fact that the dual variables 'behave similarly', and our SAT solver can exploit this similarity without a significant overhead [KR03]. For example, SAT solvers based on the *saturation* method [SS00] are known to perform well when there are many equivalent (up to negation) variables in the instance.

5 A SAT-based method for checking 3-valued equivalence

In this section, we show how the BMC algorithm [BCC99,BCCFZ99] and the induction method [SSS00] can be adapted to enable verification without a reset state, by using the dual-rail state $\perp$ as an initial state. Unlike the original ATPG based algorithm of Huang et al [HCC01], our algorithm is (sound and) *complete*. We will also see that a more direct encoding of the ATPG algorithm into SAT based dual-rail formalism results in an incomplete algorithm.

Algorithm 1 describes our 3-valued equivalence verification procedure without a reset state.

Theorem 5.1 *Algorithm 1 is a sound and complete procedure for checking 3-valued equivalence.*

Proof. *The situations when the proof obligation can be falsified are exactly the situations where the pair (o_1, o_2) is a 3-valued differ-pair:*

$$(o_1{}^{dr}, o_2{}^{dr}) \in \{(T^{dr}, F^{dr}), (F^{dr}, T^{dr}), (\perp^{dr}, T^{dr}), (\perp^{dr}, F^{dr}), (T^{dr}, \perp^{dr}), (F^{dr}, \perp^{dr})\}.$$

$\overline{F}$, since $T\,XOR\,T = F\,XOR\,F = F$.

Algorithm 1 SAT-based algorithm for 3-valued equivalence verification w/o reset state

1: Check 3-valued equivalence of o_1 and o_2 $\{$
2: Create a dual-rail sequential instance corresponding to C_{xnor} ;
3: Bind to T high and low latch values in phase 0 ;
4: Add proof obligation expressing ${o_{xnor}}^{dr} \Leftrightarrow T^{dr}$;
5: Apply a complete SAT-based method to the instance ;
6: **if** a counter-example is generated **then**
7: Report 'DIFFER' and EXIT ;
8: **end if**
9: **if** the proof obligation is proved **then**
10: Report 'EQUAL' and EXIT ;
11: **end if**
12: **else** report 'INDETERMINATE' and EXIT ;
13: $\}$

Thus the algorithm returns 'DIFFER' exactly when $C_1 \not\cong_3 C_2$, and the counter-example brings C_{xnor} from state $\perp$ to a 3-valued differ state. By the same argument, the algorithm returns 'EQUAL' iff $C_1 \cong_3 C_2$. (Only) in case the run terminates without resolving the instance, the algorithm returns 'INDETERMINATE'. □

In Algorithm 1, we mainly use induction based algorithms [SSS00], since they perform better when a full proof is sought. (We use the BMC based methods in algorithms that require initialization – the counter-examples become (part of) the initializing or synchronizing sequences [RH02,KR03].) We recall briefly that in the induction method, unrolling with increasing depths is performed, till a counter-example (to the proof-obligation) is found, or induction step can be proved (see also [BC00] for a nice description on why a simple induction, with depth 1, is not enough). In [SSS00], termination conditions for induction step are presented that reflect both forward and backward state space traversal methods, thus our algorithm also can be made forward or backward (or combined), depending on which kind of induction is used.

A direct encoding of the ATPG algorithm of [HCC01] into SAT-based model checking problem would correspond to

- Considering the set of (combined) states where all latches of C_1 or all latches of C_2 are in state $\perp^{dr}$ as the set of *initial* states;
- Considering the states where ${o_{xor}}^{dr} = T^{dr}$ as the *bad* states;
- And applying the backward induction scheme of [SSS00].

Counter-examples found by such an algorithm would be the correct ones, but the algorithm would miss counter-examples in situations like in Example 3.2. We therefore abandon this algorithm in favor of Algorithm 1 above.

6 Verification with respect to other concepts of equivalence

In this section, we comment on the applicability of our methods for equivalence checking with respect to some other concepts of equivalence, namely steady-state equivalence and alignability equivalence.

6.1 Verifying steady-state equivalence

We recall definition of steady-state equivalence from [KMH02]. In steady-state equivalence, we compare the outputs only in time phases where both outputs have binary values. Values in other time phases are don't cares. Thus circuits that are 3-valued equivalent are also steady-state equivalent, but not vice versa.

Definition 6.1 ([KMH02])

- An input vector sequence π is called a *steady-state* sequence for a circuit C if $o(\perp, \pi)$ is binary.
- Circuits C_1 and C_2 with outputs o_1 and o_2 are called *steady-state equivalent*, written $C_1 \cong_{ss} C_2$, iff for any input sequence π that is a steady-state sequence for both C_1 and C_2, $o_1(\perp, \pi) = o_2(\perp, \pi)$.

In order to develop a verification procedure for verifying steady-state equivalence without a reset state, we can simply change the proof obligation in Algorithm 1 to the following one: $(binary(o_1)\ \&\ binary(o_2)) \Rightarrow (o_1 \Leftrightarrow o_2)$, where $binary(o_i)$ denotes the property that o_i has a binary value (that is, $o_{ih} = \neg o_{il}$), $i = 1, 2$.

6.2 Verifying alignability equivalence

We recall definition of *alignability* or *post-synchronization* equivalence from [Pix92].

Definition 6.2 • State (s_1, s_2) of the combined circuit C_{xnor} is an *equivalent* state if for any input sequence π, $o_1(s_1, \pi) = o_2(s_2, \pi)$. [4]

- A binary input sequence π is an *aligning* sequence for a combined state (s_1, s_2) of C_{xnor} if it brings C_{xnor} from state (s_1, s_2) into an equivalent state.
- Circuits C_1 and C_2 are *alignable*, written $C_1 \cong_{aln} C_2$, if every state of C_{xnor} has an aligning sequence (or equivalently, there is a sequence, called a *universal aligning sequence*, that aligns any state of C_{xnor}).

Lemma 6.3 (i) *If circuits C_1 and C_2 are synchronizable and $C_1 \cong_{ss} C_2$, then $C_1 \cong_{aln} C_2$.*

(ii) *If $C_1 \cong_{aln} C_2$, then $C_1 \cong_{ss} C_2$.*

Proof.

[4] The concepts of equal- and differ-states should not be mixed with equivalent and inequivalent states. In this definition, all states are binary.

(i) *Let C_1 and C_2 be steady-state equivalent. Consider sequence π that synchronizes both C_1 and C_2, say into a state pair (s_1, s_2). Then for any sequence π', the concatenation of π and π' is a steady-state sequence, thus π' ends in a state (s_1', s_2') where o_1 and o_2 have equal binary values. Thus (s_1, s_2) is an equivalent state pair, implying that $C_1 \cong_{aln} C_2$.*

(ii) *Now let C_1 and C_2 be alignable. Suppose on the contrary that C_1 and C_2 are not steady-state equivalent. Then there is a steady-state sequence π that brings any state into a differ state (with outputs different binary values). Such a sequence can distinguish any pair of states, thus C_1 and C_2 do not have an equivalent state pair, and they cannot be alignable – a contradiction.*

□

Alignability equivalence is a widely used concept of equivalence. Therefore, to show the importance of our methods, it is important to clarify the relevance of our methods for alignability equivalence verification. Indeed, there are a number of ways allowing to infer alignability or non-alignability of circuits by using the methods of checking steady-state or 3-valued equivalence presented in the early sections. We mention a few of them, based on the above lemma and a result in [HCC01].

- If our steady-state verification algorithm proves circuits C_1 and C_2 inequivalent, then it returns a counter-example π_d that brings C_{xnor} from state $\perp$ to *binary* differ-state. Such a sequence π_d is actually a universal counter-example demonstrating that $C_1 \not\cong_{aln} C_2$ (as it can distinguish any pair of states of C_1 and C_2).
- If on the other hand $C_1 \cong_{ss} C_2$, then from the SAT procedure proving this, it is possible to extract information whether the part $binary(o_1)\&binary(o_2)$ becomes true in some phase. Such a procedure depends on the particular strategy used to resolve the sequential instance, and goes beyond the scope of this paper. (Of course initializability of C_1 and C_2 can be checked separately.) If yes, we have actually proven $C_1 \cong_{aln} C_2$ as well. If not, we cannot claim $C_1 \not\cong_{aln} C_2$, as synchronizing but not initializing sequence may exist that brings C_{xnor} into an equivalent state. For such not 3-*valued initializable circuits* [HCC01] we use a formal initialization method, briefly discussed in [RH02], to find an aligning sequence when it exists.
- It is shown in [HCC01] that if both C_1 and C_2 are initializable, then $C_1 \cong_3 C_2$ implies $C_1 \cong_{aln} C_2$. Actually, it is enough to show that one of the circuits is initializable and the other one is its 3-valued safe replacement [HCC01].
- Since 3-valued equivalence requires o_1 and o_2 to match in all time phases, the above sufficient condition may not be practical to infer alignability from 3-valued equivalence. Instead, a *$(k-)$ delayed* 3-valued equivalence can be used, which requires o_1 and o_2 to match from phase k onward. Still, usage of delayed 3-valued equivalence in proving alignability is limited.

			Steady-state equivalence			Alignability equivalence		
Ckt	Latches	Gates	Passed	Problematic	Time (sec.)	Passed	Problematic	Time (sec.)
C_1	712	7838	9	0	1067	9	0	1106
C_2	1208	38259	0	1	1331	0	1	1728
C_3	100	1202	28	0	1128	28	0	2701
C_4	826	6260	7	1	2697	6	2	3921
C_5	154	1730	35	0	2008	35	0	3251
Total			**79**	**2**	**8231**	**78**	**3**	**12707**

Table 1
Comparison of performance.

7 Experimental results

We have implemented Algorithm 1 and its modified version for checking 3-valued and steady-state equivalences. Most of our circuits are resetable, thus in practice this algorithms performs alignability check as well.

Experiments reported below were performed on 550MHz dual CPU Linux machine with 2GB memory. A timeout of 300 seconds was used in the SAT solver. Experimental results show that say the steady-state equivalence algorithm is $1.5x$ faster than a dual-rail alignability equivalence algorithm that first performs synchronization of the specification and implementation circuits (see Table 1; there, numbers of latches and gates represent an average per output). And as already mentioned, the latter in turn is $1.5x$ faster than a corresponding single-rail implementation of alignability checking engine (despite the fact that dual-rail modeling requires twice as much NSFs) [KR03]. Furthermore, the counter-examples returned by the steady-state engine are in average $2x$ shorter than those found by the alignability engine, which is much more important (for debugging) than the above reported speed-up (see Table 2, where circuits C_1 – C_7 contain loops, while circuits C_8 – C_{14} are loop-free; all data is given per single outputs).

8 Conclusions

Thus far, SAT-based verification methods have been mainly concentrated on property checking, and for the good reason: It is well understood that circuit equivalence verification can be performed by the model-checking of properties that express equivalence of the circuit outputs. Indeed, in this work, we have demonstrated how SAT-based methods (such as the BMC or the induction method) can be used for proving sequential equivalence in accordance with a number of important sequential equivalence concepts.

In particular, we have developed SAT-based verification methods for verifying

Steady-state equivalence								Alignability equivalence			
Ckt	Latches	Gates	CE len	Ckt	Latches	Gates	CE len	Ckt	CE len	Ckt	CE len
C_1	526	2509	9	C_8	151	1747	4	C_1	21	C_8	8
C_2	18	160	6	C_9	173	2037	6	C_2	8	C_9	8
C_3	18	92	6	C_{10}	107	1263	6	C_3	11	C_{10}	12
C_4	18	415	4	C_{11}	112	1317	6	C_4	5	C_{11}	9
C_5	24	207	4	C_{12}	67	744	4	C_5	6	C_{12}	9
C_6	25	1121	8	C_{13}	57	619	4	C_6	10	C_{13}	10
C_7	704	7660	11	C_{14}	98	726	3	C_7	65	C_{14}	6
Total							**81**				**188**

Table 2
Comparison of counter-example length.

sequential circuits with respect to 3-valued, steady-state and (partly) alignability equivalence. The novelty of our approach is that it does not require a reset state. Instead, we can use the undefined state as a reset state, after encoding the latter into a binary representation. Unlike the ATPG-based method of [HCC01], from which our approach emerged, our algorithms for checking 3-valued and steady-state equivalence are complete. We hope that our work sheds further light on the relationship between the ATPG- and SAT-based sequential verification.

An advantage of our approach is that the verification procedure becomes relatively simple conceptually, thus it is easy to implement and maintain it. Our method *compliments* earlier methods for which synchronization is an essential part of verification, as our algorithms outperform (in a number of dimensions) similar algorithms that need to compute reset states. Clearly, this does not decrease the importance of initialization based methods. In particular, synchronization methods when initializing sequences do not exist are indispensable.

Actually, because of the importance of short counter-examples for debugging at early stages of design, steady-state verification is entering a default flow in our verification methodology, which was previously based on initialization. We remark also that the ability to find counter-examples quickly is important in the framework of model abstraction refinement (see e.g. [CGJLV00]). There, because one works with pruned models, there is a higher probability of (false) negatives, till a right pruning is found. And synchronization can be checked on correctly pruned models only, when the pruned models are steady-state equivalent.

Despite the rapid development and success of SAT-based model checking, there is still a long way to go. As an example, we mention that, on loop-free circuits, SAT-based equivalence methods (both with or without initialization) perform very poorly compared to the method developed in [KMH02] for loop-free circuits. Both

steady-state and alignability checks time out after thousands of seconds on tests that can be verified in less than a minute with the method in [KMH02]. SAT-based model checking will profit from the development of alternative ways of translating model-checking problems into SAT problems.

Acknowledgments We thank R. Fraer, A. Jas, D. Kaiss, J. Moondanos, A. Rosenmann and G. Wolfovitz for careful reading, and S.-Y. Huang and K.-T. Cheng for clarifying the subtleties of their ATPG method for checking 3-valued equivalence.

References

[ABE00] P. A. Abdulla, P. Bjesse, N. Eén. *Symbolic reachability analysis based on SAT solvers*, Tools and Algorithms for the Construction and Analysis of Systems, TACAS'00, Springer LNCS, 2000.

[BC00] P. Bjesse, K. Claessen. *SAT based verification without state space traversal*, FMCAD'00, Springer LNCS, 2000.

[BCC99] A. Biere, A. Cimatti, E. Clarke. *Symbolic model checking without BDDs*, Tools and Algorithms for the Construction and Analysis of Systems, 1999.

[BCCFZ99] A. Biere, A. Cimatti, E. Clarke, M. Fujita. Y. Zhu. *Symbolic model checking using SAT procedures instead of BDDs*, DAC 1999.

[Bry87] R. E. Bryant. *Boolean Analysis of MOS Circuits*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. CAD-6, No. 4, 1987

[BS94] R.E. Bryant, C.-J.H. Seger. *Digital circuit verification using partially-ordered state models*, Twenty-Fourth International Symposium on Multiple-Valued Logic, 1994.

[BCLMD94] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, D.L. Dill. *Symbolic model checking for sequential circuit verification*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 13, n. 4, 1994.

[CCQ96] G. Cabodi, P. Camurati, S. Quer. *Improved reachability analysis of large finite state machines*, ICCAD 1996.

[CC97] G. Cabodi, P. Camurati. *Symbolic FSM traversals based on the transition relation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, n. 5, 1997.

[CA89] K.-T. Cheng, D. Agrawal. *State assignment for initializable synthesis*, ICCAD'89.

[CJSP93] H. Cho, S.-W. Jeong, F. Somenzi, C. Pixley. *Synchronizing sequences and symbolic traversal techniques in test generation*, J. Electron. Test.: Theory & Applications, vol. 4, n. 2, 1993.

[CGJLV00] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. *Counterexample-guided Abstraction Refinement*, CAV'00, Springer LNCS, 2000.

[CGP99] E. Clarke, O. Grumberg, D. Peled. *Model Checking*, MIT Press, 1999.

[CPRSS97] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda and G. Squillero. *A new approach for initialization sequences computation for synchronous sequential circuits*, IEEE VLSI in Computers and Processors, 1997.

[CBM89] O. Coudert, C. Berthet, J.C. Madre. *Verification of synchronous sequential machines based on symbolic execution*, Workshop of Automatic Verification Methods for Finite State Systems, 1989.

[CM90] O. Coudert, J.C. Madre. *A Unified framework for the formal verification of sequential circuits.*, ICCAD 1990.

[HS98] G.D. Hachtel, F. Somenzi. *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1998.

[HCCG96] S.-Y. Huang, K.-T. Cheng, K.-C. Chen, U. Glaeser. *An ATPG based framework for verifying sequential equivalence*, International Test Conference, 1996.

[HCC96] S.-Y. Huang, K.-T. Cheng, K.-C. Chen. *On verifying the correctness of retimed circuits*, Great-Lake symposium on VLSI, 1996.

[HCC01] S.-Y. Huang, K.-T. Cheng, K.-C. Chen. *Verifying sequential equivalence using ATPG techniques*, ACM Transactions on Design Automation of Electronic Systems, 2001.

[Jon02] R.B. Jones. *Symbolic Simulation Methods for Industrial Formal Verification*, Kluwer Academic Publishers, 2002.

[KR03] D. Kaiss, A. Rosenmann. *Dual rail modeling for SAT based sequential verification.* (In preparation.)

[KBS96] M. Keim, B. Becker and B. Stenner. *On the (non)-resetability of synchronous sequential circuits*, IEEE VLSI Test Symposium, 1996.

[KMH02] Z. Khasidashvili, J. Moondanos, Z. Hanna. *TRANS: Efficient Sequential Verification of Loop-Free Circuits*. HLDVT'02, IEEE Computer Society Press, 2002.

[Koh78] Z. Kohavi. *Switching and Finite Automata Theory*, McGrawHill, New York, 1978 (second edition).

[LP96] Y. Lu and I. Pomeranz. *Synchronization of large sequential circuits by partial reset*, IEEE VLSI Test Symposium, 1996.

[McM93] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*, Kluwer Academic Publishers, 1993.

[Pix92] C. Pixley. *A theory and implementation of sequential hardware equivalence*, IEEE transactions on Computer-Aided Design, vol. 11, no. 12, 1992.

[PB94] C. Pixley, G. Beihl. *Calculating resetability and reset sequences*, ICCAD, 1991.

[PJH94] C . Pixley, S.-W. Jeong, G.D. Hachtel. *Exact calculation of synchronizing sequences based on binary decision diagrams*, IEEE transactions on Computer-Aided Design, vol. 13, 1994

[PR96] I. Pomeranz, S.M. Reddy. *On removing redundancies from synchronous sequential circuits with synchronizing sequences*, IEEE transactions of computers, vol. 45, no. 1, 1996.

[RH02] A. Rosenmann, Z. Hanna. *Alignability equivalence of synchronous sequential circuits*, IEEE International High Level Design Validation and Test Workshop, HLDVT'02, IEEE Computer Society Press, 2002.

[SB95] C.-J. H. Seger, and R. E. Bryant. *Formal verification by symbolic evaluation of partially-ordered trajectories*, Formal Methods in System Design, vol. 6, no. 2, 1995.

[SS00] M. Sheeran, G. Stålmarck. *A tutorial on Stålmarck's method of propositional proof*. Formal Methods In System Design, 16 (1), 2000.

[SSS00] M. Sheeran, S. Singh, G. Stålmarck. *Checking safety properties using induction and a SAT-solver*, FMCAD, 2000.

[TSLBS90] H. Touati, H. Savoj, B. Lin, R.K. Brayton, A. Sangiovanni-Vincentelli. *Implicit enumeration of finite state machines using BDDs*, CAD'90, 1990.