A Hybrid Fault Simulator for Synchronous Sequential Circuits

Rolf Krieger[†] Bernd Becker[*] Martin Keim[†]

TR-94-008

Abstract

Fault simulation for synchronous sequential circuits is a very time consuming task. The complexity of the task increases if there is no information about the initial state of the circuit available. In this case, an unknown initial state is assumed which is usually handled by introducing a three-valued logic. It is known, that fault simulation based upon this logic only determines a lower bound for the fault coverage achievable by a test sequence. Therefore, we developed a hybrid fault simulator H-FS *combining the advantages of a fault simulator using the three-valued logic and of an exact symbolic fault simulator based upon binary decision diagrams.* H-FS *is able to handle even the largest benchmark circuits and thereby determines fault coverages much more accurately.*

[*]Fachbereich 20 - Informatik, J.W.Goethe-Universität, D-60054 Frankfurt, and International Computer Science Institute, Berkeley, CA 94707; email: becker@kea.informatik.uni-frankfurt.de

[†]Fachbereich 20 - Informatik, J.W.Goethe-Universität, D-60054 Frankfurt; email: <name>@kea.informatik.uni-frankfurt.de

This work was supported by DFG grant Be 1176/3-1.

1 Introduction

The automatic generation of test sequences for synchronous sequential circuits has proven to be a hard problem. To reduce the cost for test generation, it has to be intensively supported by a fault simulation procedure. The task of fault simulation is to determine all faults which can be detected by a given test sequence. We assume that a fault can be modeled as a single stuck-at fault. In opposite to fault simulation for combinational circuits, fault simulation for synchronous sequential circuits is a problem whose exact solution cannot be determined in polynomial time if there is no information about the initial state of the circuit available. Therefore, in most cases only a lower bound for the fault coverage is determined by performing the fault simulation using a three-valued logic [3, 12, 14, 19]. Unfortunately, there are a lot of circuits for which the gap between this lower bound and the exact fault coverage which can be achieved by a given test sequence is very large. The reason for that gap is the inherent inaccuracy of the three-valued logic. For instance, there are circuits whose synchronizing sequence cannot be verified using the three-valued logic [16]. (A synchronizing sequence is an input sequence which drives the circuit from any initial state into a single state).

A lot of work has been done to overcome these problems. On the one hand, a full or partial scan design [2, 7] or a sophisticated state assignment procedure [7, 10] can help. But a scan design involves additional circuitry such as multiplexers which increase the area and degrade the performance of the circuit [2]. A state assignment procedure can only be applied during the synthesis and not if an already designed circuit is considered. On the other hand, a fault simulation procedure based upon the multiple observation time test strategy can help [17]. But even for small circuits, the application of this strategy is very time consuming and should be restricted to faults which are redundant with respect to the single time observation strategy. Another proposal is the application of symbolic traversal techniques to test generation [9]. This approach is based upon reduced ordered binary decision diagrams (*OBDD*s) [6]. It is of great promise but up to now not feasible for larger circuits.

In this paper, we present an event-driven hybrid fault simulator based upon the single observation time test strategy which can help to solve these problems. Hybrid means that our algorithm supports different logics for simulation and allows a dynamic switching of the applied logic after each simulation step. More precisely, it consists of three kinds of fault simulation procedures: (1) a fault simulation procedure *X-FS* based upon the three-valued logic, (2) a symbolic fault simulation procedure *B-FS* based upon binary decision diagrams [6] and (3) a fault simulation procedure *BX-FS* which is hybrid itsself in the sense that a symbolic true value simulation and an explicit fault simulation procedure based upon the three-valued logic are combined. They differ in their time and space requirement and the accuracy of the fault coverage determined. The hybrid fault simulator *H-FS* tries to combine the advantages of these procedures by choosing a convenient logic before starting the simulation for the next test vector. For instance, if the space requirement of *B-FS* is becoming too large, the hybrid fault simulator will select *BX-FS* or if necessary *X-FS* based upon the three-valued logic. After the application of a few patterns that possibly initialize a large number of memory elements and reduce the space requirement of *B-FS*, it will try to start *B-FS* again. For that reason, the hybrid fault simulation strategy works also for the largest benchmark circuits [5], and it determines the exact fault coverage or at least a tighter lower bound. In some cases simulating random test sequences it achieves fault coverages which are higher than fault coverages achieved by up-to-date test generation systems [9, 18]. Consequently, using our fault simulator most of these systems can improve their performance considerably.

The paper is structured as follows: We start in section 2 by presenting some definitions and important properties of synchronous sequential circuits. In section 3, we describe the different fault

simulation procedures and compare their accuracy and time and space complexity. The hybrid fault simulation procedure is described in section 4. Finally, in section 5 experimental results are given, which demonstrate the efficiency of the presented hybrid fault simulation procedure for the ISCAS89-benchmark circuits [5].

2 Definitions

A synchronous sequential circuit can be considered as a *Finite State Machine* (FSM) as introduced in [13].

Definition 1: A finite state machine M is defined as a 5-tuple $M = (I, S, O, \delta, \lambda)$, where I is the input set, O is the output set and S is the set of states, $\delta : S \times I \rightarrow S$ is the next state function, and $\lambda : S \times I \rightarrow O$ is the output function.

Since we consider a gate level realization of the FSM as outlined in Fig. 1, we have $I = \mathbf{B}^k$, $O = \mathbf{B}^l$ and $S = \mathbf{B}^m$ with $\mathbf{B} = \{0, 1\}$. k denotes the number of primary inputs, l denotes the

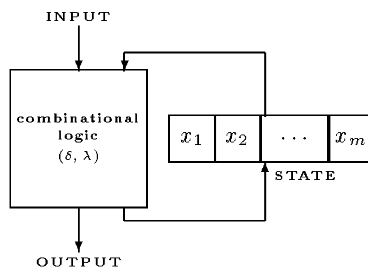

Figure 1: Model of a finite state machine

number of primary outputs and m denotes the number of memory elements. The functions δ and λ are computed by a combinational circuit. As it is well-known, the structure of a combinational circuit can be described as a directed acyclic graph $G = (V, E)$. Its nodes correspond to the gates, primary inputs and outputs of the circuit. Its edges correspond to the leads of the circuit. The inputs (outputs) of the combinational circuit which are connected to the outputs (inputs) of the memory elements are called secondary inputs (outputs).

For the description of our algorithms we use the following notations: $Y = y(1), \ldots, y(n)$ denotes an input sequence of length n and $y_i(t)$, $1 \leq i \leq k$, denotes the value that is assigned to the i-th primary input before starting simulation at time t, $1 \leq t \leq n$. $S(Y) = s(0), s(1), \ldots, s(n)$ denotes the state sequence defined by Y. $s_i(t)$, $1 \leq i \leq m$, denotes the state of the i-th memory element after simulation step t. $s(0) = s_0$, $s_0 \in S$, is called the initial state. $O(s_0, Y) = o(1), o(2), \ldots, o(n)$ denotes the output sequence defined by the initial state s_0 and Y. $o_i(t)$, $1 \leq i \leq l$, denotes the value at the i-th primary output after simulation step t. Using these notations the next state is given by

$$s(t) = \begin{cases} s_0 & \text{if } t = 0 \\ \delta(s(t-1), y(t)) & \text{otherwise} \end{cases}$$

The output $o(t)$ is defined by the function λ, analogously. To describe the simulation of the machine starting in an unknown initial state, we define a function *s_set* as follows:

$$s_set(t) = \begin{cases} \mathbf{B}^m & \text{if } t = 0 \\ \{\delta(s, y(t)) | s \in s_set(t-1)\} & \text{otherwise} \end{cases}$$

s_set(t) represents the set of states which can be reached at time t, $1 \leq t \leq n$, starting with an unknown initial state.

A stuck-at fault f transforms a machine M into a machine $M^f = (I, O, S, \delta^f, \lambda^f)$. The functions δ^f, λ^f and s_set^f are defined, analogously. Notice, that we get $s_set(0) = s_set^f(0)$ for any fault.

If there is no knowledge about the initial state of the machine using the single observation time strategy the detectability of a stuck-at fault with respect to an input sequence Y can be defined according to [1] as follows:

Definition 2: *A fault f is detectable by an input sequence $y(1), y(2), \ldots, y(n)$ if $\exists t \leq n$, $\exists i \leq l$, $\exists b \in \{0, 1\}$ such that (A) $\forall s \in \mathrm{s_set}(t-1)$: $o_i(t) = b$ with $o(t) = \lambda(s, y(t))$ and (B) $\forall s \in \mathrm{s_set}^f(t-1)$: $o_i^f(t) = \overline{b}$ with $o^f(t) = \lambda^f(s, y(t))$.*

Now, we formalize the problem, which is considered in the sequel.

Problem: *Given a synchronous sequential circuit and an input sequence Y, determine the detectability of all stuck-at faults (the fault coverage) according to definition 2.*

The problem is at least *NP*-hard. This can be proved by a reduction of the *non-tautology* problem to the problem described above. The non-tautology problem corresponds to the question wether a given Boolean expression ist not a tautology, i.e. is there a truth assignment that makes the Boolean expression false [11].

3 Fault Simulation

In this section, we discuss several strategies to solve the problem formulated in the previous section. All strategies are based upon the same algorithm for fault simulation. The only difference between the strategies is the logic used for the simulation. Therefore, we give a logic-independent description of the fault simulation procedure in the sequel.

3.1 SFP-based Fault Simulation Algorithm

During the last years, a lot of algorithms for fault simulation have been developed and their efficiency has been compared. Roughly spoken, they can be devided into two classes. The first class makes use of the machine word length for simulating 32 faults in parallel [8]. The second class is based upon the single fault propagation (*SFP*). Some approaches belonging to the second class use the machine word length for parallel evaluation of input patterns [3, 12]. The overall fault simulation procedure used in our approach correponds to *SFP*. Since different logics encoding the unknown initial state in different ways are to be used, we cannot take advantage of the machine word length for parallelization. In order to simulate a synchronous sequential circuit we replace each memory element by a secondary input and a secondary output. After this replacement the circuit can be described by an acyclic directed graph. The simulation of the circuit can be performed similar to the simulation of a combinational circuit by evaluating the gates in a topological order. The value of a secondary input at time t, $1 \leq t \leq n$, is defined by the value of the corresponding secondary output at time $t-1$. The fault simulation procedure receives a set of faults F, a test sequence Y and an encoding of the unknown initial state s_0 as inputs. First, a true value simulation is carried out which determines the value of each lead implied by the input vector and the present state vector. The implied values at the secondary outputs define the next state vector. Subsequently, an explicit fault simulation is carried out. For each fault f, $o^f(t)$ and $s^f(t)$ are determined by an event-driven single fault propagation. This means, that the faults are injected one by one. The effects of each fault are propagated towards the primary and secondary outputs. If thereby any primary output

changes from 0 to 1 (1 to 0) the fault will be marked as detectable with respect to s_0. Of course this fault is dropped and it will not be considered during following simulation steps. If a secondary output changes, the next state of the faulty machine is different from the next state of the correct machine and it has to be stored for the next simulation step. To reduce the memory requirement only the memory elements are stored whose values are different from that of the good machine.

3.2 Three-Valued Fault Simulation

A common strategy to handle the problems produced by an unknown initial state is given by a fault simulation procedure *X-FS* based upon a three-valued logic over $\mathbf{B}_X = \{0, 1, X\}$. 0 and 1 are called defined values. X denotes the unknown or undefined value. It is used for encoding the unknown initial state, which is given by $s_0 = (X, X, \ldots, X)$. The machine is initialized if all memory elements have defined values. A fault f is marked as detectable during the explicit fault simulation if $\exists t \leq n$, $\exists i \leq l$, $\exists b \in \{0,1\}$ such that $o_i(t) = b$ and $o_i^f(t) = \overline{b}$. This means there exists an output for which M and M^f compute different defined values.

As it is well-known, fault simulation based upon the three-valued logic is very inaccurate. There are circuits having synchronizing sequences that cannot be verified by a simulation based upon the three-valued logic. Many papers point out to this problem. The inaccuracy can be explained on the one hand by the binary encoding of the states [16] and on the other hand by the logic realizing the next state function. For instance, consider the circuit shown in Fig. 2. A simulation based upon the three-valued logic is not able to verify the synchronizing sequence which is given by the input vector [a=1,b=0,c=0].

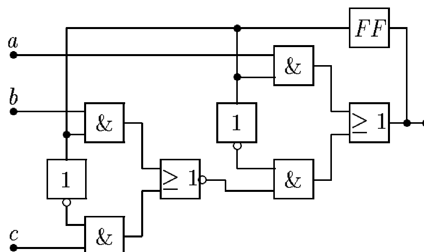

Figure 2: $\mathbf{B}_X$-uninitializable circuit

We conclude:

Theorem 1: *The procedure* X-FS *determines a lower bound for the exact fault coverage of a given test sequence Y.*

Proof: In proof of it we use the fact that a state vector $s(t) \in \mathbf{B}_X^m$ represents a set $s_set_X(t)$ containing all states $(a_1, \ldots, a_m) \in \mathbf{B}^m$ with $a_i = s_i(t)$ if $s_i(t) \neq X$ and $a_i \in \{0,1\}$ otherwise, $1 \leq i \leq m$. By induction it can be shown that $s_set(t) \subseteq s_set_X(t)$, $0 \leq t \leq n$. In addition, it can be shown that $s_set^f(t) \subseteq s_set_X^f(t)$ for all faults f, $0 \leq t \leq n$. Therefore, if a fault f is marked as detectable at time t because there is an output with $o_i(t) = b$ and $o_i^f(t) = \overline{b}$, $b \in \{0,1\}$, we get $\forall s \in s_set_X(t)$: $o_i(t) = b$ with $o(t) = \lambda(s, y(t))$ and $\forall s \in s_set_X^f(t)$: $o_i^f(t) = \overline{b}$ with $o^f(t) = \lambda^f(s, y(t))$. $\square$

The advantage of the fault simulation procedure *X-FS* is its time and space behaviour. Based upon the three-valued logic the fault simulation of a circuit C for a test sequence of length n can be performed in time $O(n \cdot |C|^2)$ logic where $|C|$ denotes the size of C. Moreover, the loss of accuracy is probably negligible if problems discussed above do not occur. But even for some of the ISCAS

benchmark circuits [5] it can be proved that their synchronizing sequence cannot be verified using a three-valued logic. Consequently, a fault simulation for these circuits based upon the three-valued logic only achieves a lower bound which turns out to be very inaccurate.

3.3 *OBDD*-Based (Symbolic) Fault Simulation

As outlined in the last section only a lower bound for the fault coverage is determined if fault simulation is based upon the three-valued logic $\mathbf{B}_X$. A symbolic fault simulation procedure called *B-FS* can help to overcome this problem. The simulation strategy used by *B-FS* is identical to the strategy used by the fault simulation procedure *X-FS*.

Formally, *B-FS* is based upon the logic $\mathbf{B}_m = \{g : \mathbf{B}^m \rightarrow \mathbf{B}\}$. The two constant functions contained in $\mathbf{B}_m$ will be abbreviated by $\mathbf{0}$ and $\mathbf{1}$. For the simulation, an element of the logic is represented by a reduced ordered binary decision diagram (*OBDD*) as introduced in [6]. Using an *OBDD*-package, *OBDD*s can be constructed and modified, efficiently [4]. To perform a symbolic fault simulation a component of an input vector will be interpreted as one of the constant functions $\mathbf{0}$ and $\mathbf{1}$. To encode the unknown initial state we introduce a boolean variable x_i, $1 \leq i \leq m$, for each memory element representing its unknown value at the beginning of the simulation. The initial state of the good machine as well as of the faulty machines is given by $s_0 = (x_1, x_2, \ldots, x_m)$. A fault f is marked as detectable during the explicit fault simulation if $\exists t \leq n$, $\exists i \leq l$, $\exists g \in \{\mathbf{0}, \mathbf{1}\}$ such that $o_i(t) = g$ and $o_i^f(t) = \overline{g}$. This means there exists an output for which M and M^f compute different constant functions. Using the definition 2 and the definition of the initial state, we can prove the following theorem:

Theorem 2: *The procedure* B-FS *as described above determines the exact fault coverage according to definition 2 for a given test sequence* Y.

Proof: A state vector $s(t)$ defines a function $s(t) : \mathbf{B}^m \rightarrow \mathbf{B}^m$. Consequently, a state vector $s(t)$ defines a set $s_set_B(t) = \{a \in \mathbf{B}^m | \exists z \in \mathbf{B}^m : a = s(t)(z)\}$. By induction it can be shown that $s_set(t) = s_set_B(t)$, $0 \leq t \leq n$, and $s_set^f(t) = s_set_B^f(t)$ for all faults f. Moreover, if a fault f is marked as detectable at time t because there is an output i, $1 \leq i \leq l$, with $o_i(t) = g$ and $o_i^f(t) = \overline{g}$, $g \in \{\mathbf{0}, \mathbf{1}\}$, we get $\exists b \in \{0,1\} : [\forall s \in s_set_B(t)$: $o_i(t) = b$ with $o(t) = \lambda(s, y(t))$ and $\forall s \in s_set_B^f(t)$: $o_i^f(t) = \overline{b}$ with $o^f(t) = \lambda^f(s, y(t))]$. □

The disadvantage of *B-FS* is its time and space behaviour. The worst case size of an *OBDD* representing a function in m arguments is nearly $O(2^m/m)$ [15]. The space requirement of *B-FS* may be much larger because the true value simulation assigns an *OBDD* to each lead of the circuit which must be stored for the event-driven explicit fault simulation. Moreover, the state vector of the good machine and all state vectors of the faulty machines which have to be stored for the next simulation step consist of symbolic values (*OBDD*s). Consequently, the space and time requirement may be very high and it forbids an application of a purely *OBDD*-based fault simulation algorithm to large circuits in most cases.

3.4 Mixed-Logic Fault Simulation

Up to now, we described two fault simulation procedure *X-FS* and *B-FS* which differ in their accuracy and their space and time requirement. Now, we go a first step towards a hybrid fault simulation procedure by combining *X-FS* and *B-FS*. The resulting mixed-logic fault simulation procedure is called *BX-FS* and it works as follows: The true value simulation which is carried out only once per input vector uses the accurate symbolic logic $\mathbf{B}_m$ (*OBDD*s). The expensive explicit fault simulation uses the three-valued logic $\mathbf{B}_X$. To this end the symbolic values of the leads computed by the

true value simulation have to be transformed into values of the three-valued logic. A mapping $\tau : \mathbf{B}_m \to \mathbf{B}_X$ with

$$\tau(g) = \begin{cases} 0 & \text{if } g = \mathbf{0} \\ 1 & \text{if } g = \mathbf{1} \\ X & \text{otherwise} \end{cases}$$

defines the transformation. Fig. 3 illustrates the transformation of a symbolic assignment determined

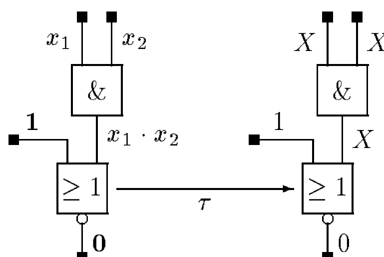

Figure 3: Transformation of a symbolic assignment to a three-valued assignment

by the true value simulation into a three-valued assignment. All symbolic values that are unequal to $\mathbf{0}$ and $\mathbf{1}$ are transformed to X according to τ. The explicit fault simulation uses the resulting three-valued assignment for a more efficient event-driven single fault propagation than in the pure symbolic case. Notice, that using this computation of the assignment instead of using a three-valued true value simulation increases the accuracy of the explicit fault simulation. A simple example illustrates the improvement. Consider the circuit shown in Fig. 2. A symbolic true value simulation for the input vector $[a = 1, b = 0, c = 0]$ initialized the good machine and assigns a $\mathbf{1}$ to the primary output of the circuit. A true value simulation based upon the three-valued logic assigns the value X to the primary output. Using *BX-FS* after the transformation of the assignment the output has the value 1 instead of X and a stuck-at 0 fault at the output is marked as detectable.

It turns out that using different logics for the true value simulation and the explicit fault simulation leads to a new problem. Consider, the situation shown in Fig. 4. It shows a symbolic assignment

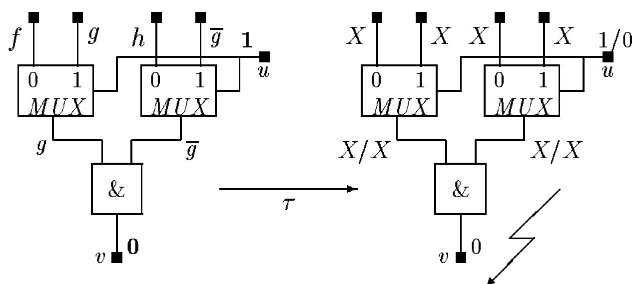

Figure 4: *BX-FS* event single fault propagation

determined by the true value simulation before and after the logic transformation. In Fig. 4, f, g, h, $\mathbf{0}$ and $\mathbf{1}$ are elements of $\mathbf{B}_m$ with f, g, $h \notin \{\mathbf{0}, \mathbf{1}\}$. Now, assume that a stuck-at 0 fault at u is injected and the resulting event are propagated towards v (see the right-hand side picture). First, the multiplexers are evaluated. Since no event is produced the single fault propagation stops and the value of v remains 0. But considering the symbolic assignment, we get $f \cdot h$ as value of v in the faulty

case. This corresponds to X if $f \cdot h \notin \{\mathbf{0}, \mathbf{1}\}$ and not to 0. Now assume, that v is connected to an *OR* gate whose second input assumes the value 1/0 during the event propagation. In combination with a 0/0 at v the stuck-at 0 fault at u is observable and is marked as detectable. In combination with a $0/X$ the fault is not observable. Therefore, the event-driven single fault propagation has to be modified to guarantee that *BX-FS* determines a lower bound for the fault coverage. Whenever a gate g is evaluated during the event-driven single fault propagation and the value at the output of the gate is equal to X in the good and in the faulty case the subsequent gates that get this value as input are also evaluated to determine the actual faulty values. Consider again Fig. 4. Evaluating the multiplexers we get two X/X events. Consequently, the succeeding *and* gate has to be evaluated and we get a $0/X$ event at v.

Lemma 1: Modification of the event-driven single fault propagation as described above yields $s_set^f_{BX}(t) \subseteq s_set^f(t)$, $0 \leq t \leq n$, for all faults f where $s_set^f_{BX}(t)$ denotes the set of states which can be reached at time t starting in unknown initial state.

Using this result, we get:

Theorem 3: *The procedure* BX-FS *as described above determines a lower bound for the fault coverage. The lower bound is tighter than the bound determined by* X-FS.

The proof of the theorem directly follows from Lemma 1, Theorems 1, 2 and the example in Fig. 2.

Since the procedure *BX-FS* only uses symbolic values for the true value simulation, the true value simulation can directly delete an *OBDD* assigned to a lead whenever all gates connected with the lead have been evaluated. Only, the *OBDD*s assigned to the secondary outputs must be stored for the further computation. The state vectors of the faulty machines consist of elements of the three-valued logic. Consequently, only the state vector of the good machine consists of symbolic values. Due to this fact, the space requirement of *BX-FS* is considerably smaller than that of *B-FS*. With regard to the accuracy, *BX-FS* lies between *X-FS* and *B-FS*. As we will see, in many cases the accuracy of *BX-FS* is better than that of *X-FS*. Therefore *BX-FS* represents a compromise with respect to space requirement and accuracy.

4 Hybrid Fault Simulation

As we saw in the previous section, fault simulation based upon the three-valued logic (*X-FS*) has some advantages compared with a fault simulation procedure based on *OBDD*s (*BX-FS* or *B-FS*) with respect to the space and time requirement. On the other hand, it only determines a lower bound for the exact fault coverage which can be achieved by a given test sequence. The gap between the bound and the exact fault coverage may be unacceptable. The *B-FS* determines the exact fault coverage according to definition 2. But the quality improvement must be paid for with an exponential worst case space requirement. Due to this fact, in general fault simulation based upon *OBDD*s (*BX-FS* and *B-FS*) is not feasible for large circuits. Conclusions drawn from the results presented in [9] emphasize this statetment. The authors of [9] propose the application of symbolic traversal techniques to test generation. Their experiments have shown that their symbolic methods are not applicable to the largest benchmarks presented in [5]. For that reason, we developed a hybrid fault simulation procedure *H-FS* which allows to select the logic for simulating the next input vector. Fig. 5 depicts the overall structure of *H-FS*. It incorporates the fault simulators *X-FS*, *BX-FS* and *B-FS*. Depending on the space requirement needed for a symbolic fault simulation *H-FS* selects a convenient fault simulation procedure. In doing so, the stored state vectors of the good and faulty machines have to be transformed from one logic into the other. The necessary logic transformation

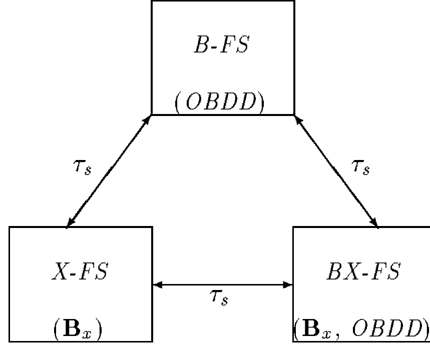

Figure 5: Hybrid Fault Simulator *H-FS*

is performed by two procedures computing the transformation functions $\tau_s : \mathbf{B}_X^m \rightarrow \mathbf{B}_m^m$ with $\tau_s(a_1, \ldots, a_m) = (g_1, \ldots, g_m)$ and

$$g_i \quad = \quad \begin{cases} \mathbf{0} & \text{if } a_i = 0 \\ \mathbf{1} & \text{if } a_i = 1 \\ x_i & \text{if } a_i = X \end{cases} ,$$

$1 \leq i \leq m$, for all $(a_1, \ldots, a_m) \in \mathbf{B}_X^m$ and $\tau_s^{-1} : \mathbf{B}_m^m \rightarrow \mathbf{B}_x^m$ with $\tau_s^{-1}(g_1, \ldots, g_m) = (a_1, \ldots, a_m)$ and

$$a_i \quad = \quad \begin{cases} 0 & \text{if } g_i = \mathbf{0} \\ 1 & \text{if } g_i = \mathbf{1} \\ X & \text{otherwise} \end{cases} ,$$

$1 \leq i \leq m$, for all $(g_1, \ldots, g_m) \in \mathbf{B}_m^m$. τ_s and τ_s^{-1} are used for transforming three-valued state vectors to *OBDD* state vectors and vice versa. For instance, consider Fig. 6. It shows the transformation

$s(t)$	X	0	1	X
$s^{f_1}(t)$	0	1	X	X
$s^{f_2}(t)$	X	0	1	1

$\xrightarrow{\tau_s}$

$s(t)$	x_1	$\mathbf{0}$	$\mathbf{1}$	x_4
$s^{f_1}(t)$	$\mathbf{0}$	$\mathbf{1}$	x_3	x_4
$s^{f_2}(t)$	x_1	$\mathbf{0}$	$\mathbf{1}$	$\mathbf{1}$

Figure 6: Transformation of three-valued state vectors into symbolic state vectors

of three-valued state vectors into the corresponding symbolic state vectors. $s(t)$ is the state vector computed by the good machine at time t and $s^{f_1}(t)$ and $s^{f_2}(t)$ are the state vectors computed by the faulty machine having fault f_1 or f_2, respectively. If the i-th component of a state vector is uninitialized, it is replaced by x_i, $1 \leq i \leq m$. In Fig. 6 this leads to the identical symbolic value x_1 after the transformation of the first component of $s(t)$ and $s^{f_2}(t)$ although they are uncorrelated. For shortness of the paper we omit the proof, that our algorithm remains correct if we use the same variables for transforming the state vectors of the good and the faulty circuit. Nevertheless it should be mentioned that this property helps to keep the memory requirements of the OBDD representations reasonable. Fig. 7 illustrates the transformation of symbolic state vectors into three-valued state vectors. f, g and h are elements of $\mathbf{B}_m - \{\mathbf{0}, \mathbf{1}\}$.

To perform fault simulation for synchronous sequential circuits with input sequence Y, *H-FS* receives as additional input a space limit $S_{\max}$. $S_{\max}$ bounds the memory which can be used by the

$s(t)$	f **0** **1** g
$s^{f_1}(t)$	**0** **1** h g
$s^{f_2}(t)$	h **0** **1** **1**

$\xrightarrow{\tau_s^{-1}}$

$s(t)$	X 0 1 X
$s^{f_1}(t)$	0 1 X X
$s^{f_2}(t)$	X 0 1 1

Figure 7: Transformation of symbolic state vectors into three-valued state vectors

OBDD-package. In its basic form *H-FS* works in three modes. The modes differ in their accuracy and space and time requirement as described above. *H-FS* attempts as long as possible to perform an accurate fault simulation.

Mode M_X : *H-FS* performs a fault simulation using the three-valued logic. In this mode, it works like *X-FS*.

Mode M_{BX} : *H-FS* tries to perform a fault simulation using *BX-FS*. If the space required by the *OBDD*-based true value simulation exceeds the space limit $S_{\max}$, *H-FS* selects *X-FS* and simulates the next Δ input vectors in mode M_X. Subsequently, *H-FS* works in mode M_{BX} again if necessary. A change to M_{BX} is unnecessary if all memory elements of the good machine are initialized.

Mode M_B : *H-FS* tries to perform fault simulation using *B-FS*. If the space required by *B-FS* exceeds the space limit $S_{\max}$, *H-FS* performs a logic transformation for the state vectors of the faulty machines and works in mode M_{BX} for the next Δ simulation steps. Subsequently, *H-FS* changes to mode M_B again if necessary. A change to M_B is unnecessary, if all memory elements of the good machine and of the faulty machines are initialized. Clearly, if the space limit is not exceeded by *B-FS*, *H-FS* determines the exact fault coverage working in mode M_B.

At the moment Δ is a constant defined by the user. It controls the number of simulation steps performed in a mode with lower accuracy after exceeding the space limit. For our experiments we used a value of 10 for Δ. Another possibility to control the number of simulation steps in a mode with a lower accuracy is the number of memory elements having an undefined value. For instance, if after a few simulation steps based upon the three-valued logic the number of uninitialized latches is reduced, *H-FS* changes to a mode with higher accuracy again because the reduction of uninitialized memory elements reduces the number of variables introduced by the transformation. Note that the number of variables has a strong influence on the memory requirements.

The value of Δ and the space limit have a strong influence on the run time and the accuracy of *H-FS*. For instance if no space limit is given *H-FS* started in mode M_B determines the exact fault coverage and has the same run time behaviour as *B-FS*. Using a small space limit and a large value for Δ the accuracy and the efficiency of *H-FS* converges to that of *X-FS*.

Clearly, the efficiency of *H-FS* depends on the selected mode. To support the different modes, *H-FS* checks after each simulation step whether there is any memory element left either in the good machine or in any faulty machine which is not initialized. Thereby *H-FS* automatically changes from mode M_B to mode M_X if all memory elements of the good machine and of the faulty machines are initialized because gate evaluations based upon the three-valued logic are much more efficient than gate evaluations based upon *OBDD*s. Working in mode M_{BX} only the good machine must be initialized.

Theorem 4: *The procedure* H-FS *as described above determines a lower bound for the exact fault coverage according to definition 2 for a given test sequence.*

		fault coverage[%]			CPU time [sec]		
Circ.	$\|T\|$	M_X	M_{BX}	M_B	M_X	M_{BX}	M_B
s298	162	85.71	85.71	87.01*	1.21	1.24/2	3.89
s349	91	95.71	95.71	96.86*	0.77	0.83/2	7.75
s526a	754	75.32	75.32	75.68*	23.11	23.32/1	325.41
s713	107	80.90	80.90	81.58*	1.02	1.10/6	1.40
s832	377	81.38	81.38	81.49*	5.18	5.26/1	3.74/23
s1238	349	94.69	94.69	94.69*	3.41	3.41/1	3.48/2
s1494	469	91.10	91.10	91.43*	17.24	17.40/1	8.29/9
s5378	408	74.02	74.02	74.26	50.09	59.78/47	77.93
s35932	86	87.99	87.99	87.99	154.81	160.56/1	647.52

Table 1: Simulation times in CPU seconds for deterministic vectors.

The proof of the theorem directly follows from theorem 1, 2 and 3.

H-FS can profit by the fact, that after some three-valued simulation steps in most cases the number of memory elements which are not initialized is drastically reduced. Consequently, the space required by *B-FS* or *BX-FS* may be reduced reduced because the number of variables introduced by τ_s may be less. Notifying that the space requirement may be exponential in the number of variables introduced the importance of the three-valued simulation steps is obvious. For instance, consider Fig. 6. The transformation only introduces 3 variables instead of 4 which are necessary for encoding the unkown initial state at the beginning of the simulation. The hybrid fault simulation procedure *H-FS* uses *X-FS* for reducing the space requirement of *B-FS* and *BX-FS*. Consequently, it partially allows a symbolic fault simulation even for very large circuits. Its accuracy can be controled by the space limit $S_{\max}$.

5 Experimental Results

To investigate the performance of our approach we implemented the hybrid fault simulation procedure *H-FS* based upon the procedures *X-FS*, *BX-FS* and *B-FS*. For the implementation, we used C++ and the *OBDD*-Package of [4]. The measurements were performed on a SUN SPARC station 10 with 32Mbytes of memory. For our experiments, we considered the ISCAS-89 benchmark circuits [5]. A space limit $S_{\max}$ of 2Mbytes was used to guarantee that the procedures of *H-FS* based upon *OBDD*s work efficiently. Tab. 1 shows the fault coverages and execution times for some circuits for deterministic test sequences also used in [12]. We investigated the performance of *H-FS* working in different modes as described in section 4. $|T|$ denotes the length of the test sequence. Exact fault coverages are indicated by an asterisk. Comparing the fault coverages determined by *H-FS* working in different modes, we observed that the fault coverage determined in mode M_B is higher than the lower bound determined in modes M_X or M_{BX}. The fault coverages determined by *H-FS* in modes M_X and M_{BX} are equal. This is not surprising, because using a deterministic test sequence after a few input vectors the good machine is initialized and *H-FS* automatically switches to mode M_X. For M_{BX} and M_B the simulation step after which *H-FS* works in mode M_X due to the initialized state vectors is given behind the bar. Note that we are now able to classify the accuracy of a fault simulation procedure based upon the three-valued logic by comparing the fault coverage determined in the mode M_X with the exact fault coverage determined in the mode M_B. For the first time it is possible to show that for the benchmark circuits considered in Tab. 1 the gaps between these values are very small.

	10^4 gate evaluations		
Circ.	M_X	M_{BX}	M_B
s298	15.27	15.59	6.96
s349	9.89	10.57	4.75
s526a	293.34	293.75	211.06
s713	8.35	9.38	5.75
s832	42.89	43.13	16.41
s1238	13.73	13.78	13.82
s1494	166.76	168.50	27.97
s5378	388.32	434.72	259.42
s35932	1060.67	1196.85	1177.96

Table 2: Number of gate evaluations during the explicit fault simulation.

		Fault Coverage [%]			CPU Time [sec.]		
circ.	$\|T\|$	M_X	M_{BX}	M_B	M_X	M_{BX}	M_B
510	128	0.00	4.08	51.06*	4.44	90.01	594.00
	512	0.00	21.10	96.45*	17.87	355.23	633.42
	1024	0.00	21.45	100.00*	35.73	699.48	633.94
953	128	8.34	39.39	54.96*	17.40	116.19	57.17
	512	8.34	59.59	86.28*	72.97	424.42	67.13
	1024	8.34	62.56	90.92*	148.05	807.50	74.30
5378	128	43.95	43.95	44.04	29.37	54.09	131.60
	512	59.68	59.68	59.90	80.06	118.07	238.00
	1024	63.20	63.20	63.46	135.84	177.99	297.96
9234.1	128	5.28	5.47	5.47	222.55	978.91	1465.22
13207.1	128	8.74	11.25	11.29	508.61	3908.92	2812.61
15850.1	128	17.54	17.72	17.77	382.42	620.68	1664.05
38417.1	128	3.43	4.80	4.84	979.39	4440.91	8670.67
38584.1	128	29.90	32.85	32.91	1826.41	2507.44	3216.54

Table 3: Simulation times in CPU seconds for random vectors.

Comparing the execution times, we observed that only in case of two circuits mode M_B is considerably slower than modes M_X and M_{BX}. At first sight, it should generally hold that M_B is slower than M_X and M_{BX}. But for circuits s832 and s1494, the mode M_B is faster than the other modes. This can be explained by Tab. 2, which shows the number of gate evaluations performed during the explicit fault simulation for the deterministic test sequences (given in units of 10 000 gate evaluations). For almost all circuits *H-FS* performs much less gate evaluations in mode M_B than working in the other modes, because besides the good machine most of the faulty machines are initialized during simulation. In the mode M_{BX}, *H-FS* performs the largest number of gate evaluations due to the modified single fault propagation. Of course, a gate evaluation performed during an *OBDD*-based simulation is much more expensive than a gate evaluation performed by a simulation based on the three-valued logic. But a smaller number of gate evaluations can neutralize these more expensive *OBDD* evaluation costs. From this it follows that the mode M_B can accelerate the fault simulation. Note that *H-FS* working in mode M_X is very fast for deterministic test sequences. In many examples its efficiency is approximately comparable with that of fault simulators published in [12, 3, 8]. Tab. 3 shows the fault coverages and execution times for circuits which are known to be hard to initialize. We considered the different modes of *H-FS* for random test sequences. For all these circuits, *H-FS* can increase the fault coverages considerably working in mode M_B. For

instance, consider the fault coverages obtained for circuit s510. After simulating a random sequence of length 1024 we get a fault coverage of 100%. This fault coverage is much better than the fault coverage determined by the test generation procedure VERITAS described in [9] which only achieves a fault coverage of 93.3% for a deterministic test sequence of length 3027. We are sure, that the performance of a tool like VERITAS can be improved considerably by using *H-FS*. The small difference in the execution times for circuit s510 for fault simulation with 512 and 1024 is explained by the fact that during the simulation after 533 input vectors the state vector of the good machine and the state vectors of all faulty machines were initialized and *H-FS* works in the mode M_X. Another important observation is that this circuit does not contain any fault that is redundant to the single observation time test strategy. Consequently, an application of the expensive multiple observation time test strategy as proposed in [17] is superfluous.

In opposite to other procedures using symbolic methods, we are also able to perform a more accurate fault simulation for the largest benchmark circuits. For instance, using *H-FS* in mode M_B, the fault coverage is approximately 3% higher than using M_X for circuit s13207. For a direct comparison of the accuracy of *H-FS* working in different modes consider Fig. 8. It shows the fault coverage as a function of the test sequence length for circuit s953 depending on the different modes. The resulting graph illustrates the gap between the exact fault coverage determined in mode M_B and the lower bounds computed in modes M_X or M_{BX}.

As a rule, the efficiency of *H-FS* depends largely upon the simulation step after which the memory elements are initialized. We are sure that simulating a synchronizing sequence before the test sequence would improve the performance of *H-FS* considerably.

6 Conclusions

In this paper, we presented a hybrid fault simulator for synchronous sequential circuits. It is able to select between different logics during the simulation. In particular, it can select between the well-known three-valued logic and *OBDD*s for fault simulation. Therefore, it can profit from the advantages, which are offered by the different logics. On the one hand, it may use the efficiency of the fault simulator based on three-valued logic and on the other hand it may use the accuracy of the symbolic fault simulator. The disadvantages of both strategies can be reduced. The experiments have shown, that the hybrid fault simulator is very fast. Moreover, for many benchmark circuits it computes the exact fault coverage. Even for the largest benchmark circuits for which the fault simulator based upon the three-valued logic is very inaccurate, to a certain degree the fault coverage can be increased by using the hybrid fault simulation strategy. The accuracy of the hybrid fault simulator depends on the memory offered by the working environment.

Since the fault simulator presented is based upon the single time observation test strategy, a further improvement of the accuracy can be achieved by incorporating the multiple observation time test strategy in our fault simulation procedure. Due to the accuracy of our symbolic fault simulation the expensive application of this strategy can be restricted to some few faults. Therefore, we are sure that the resulting procedure is working much more efficiently than the procedure presented in [17].

References

[1] M. Abramovici and M. Breuer. On redundancy and fault detection in sequential circuits. *IEEE Trans. on Comp.*, 28(11):864–865, 1979.

[2] V. Agrawal, K.-T. Cheng, D. Johnson, and T. Lin. Designing circuits with partial scan. *IEEE Design & Test of Comp.*, 5(4):8–15, 1988.

[3] B. Becker and R. Krieger. FAST-SC: Fast fault simulation in synchronous sequential circuits. In *VLSI Design Conf.*, pages 128–131, 1993.

[4] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[5] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[6] R. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 8:677–691, 1986.

[7] K.-T. Cheng and V. Agrawal. State assignment for initializable synthesis. In *IEEE Int'l Conf. on CAD*, pages 212–215, 1989.

[8] W. Cheng and J. Patel. PROOFS: A super fast fault simulator for sequential circuits. In *European Conf. on Design Automation*, pages 475–479, 1990.

[9] H. Cho, S. Jeong, F. Somenzi, and C. Pixley. Synchronizing sequences and symbolic traversal techniques in test generation. *Jour. of Electronic Testing, Theory and Applications*, 4:19–31, 1993.

[10] S. Devadas and K. Kreutzer. Boolean minimization and algebraic factorization procedures for fully testable sequential machines. In *IEEE Int'l Conf. on CAD*, pages 208–211, 1989.

[11] M. Garey and D. Johnson. *Computers and Intractability - A Guide to NP-Completeness.* Freemann, San Francisco, 1979.

[12] N. Gouders and R. Kaibel. A parallel pattern fault simulator for synchronous sequential circuits. In *IEEE Int'l Conf. on CAD*, volume 2, pages 542–545, 1991.

[13] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill Book Company, 1978.

[14] C. Kung and C. Lin. Parallel sequence fault simulation for synchronous sequential circuits. In *European Conf. on Design Automation*, pages 434–438, 1992.

[15] H. Liaw and C. Lin. On the obdd-representation of general boolean functions. In *IEEE Transaction on Computers*, volume 41, pages 661–664, 1992.

[16] A. Miczo. The sequential ATPG: A theoretical limit. In *IEEE Int'l Test Conf.*, pages 143–147, 1983.

[17] I. Pomeranz and S. Reddy. Fault simulation for synchronous sequential circuits under the multiple observation time testing approach. In *European Test Conf.*, pages 292–300, 1993.

[18] I. Pomeranz, S. Reddy, and L. Reddy. Increasing fault coverage for synchronous sequential circuits by the multiple observation time test strategy. In *IEEE Int'l Conf. on CAD*, pages 454–457, 1991.

[19] E. Rudnick, T. Niermann, and J. Patel. Methods for reducing events in sequential ciruit fault simulation. In *IEEE Int'l Conf. on CAD*, pages 546–549, 1991.

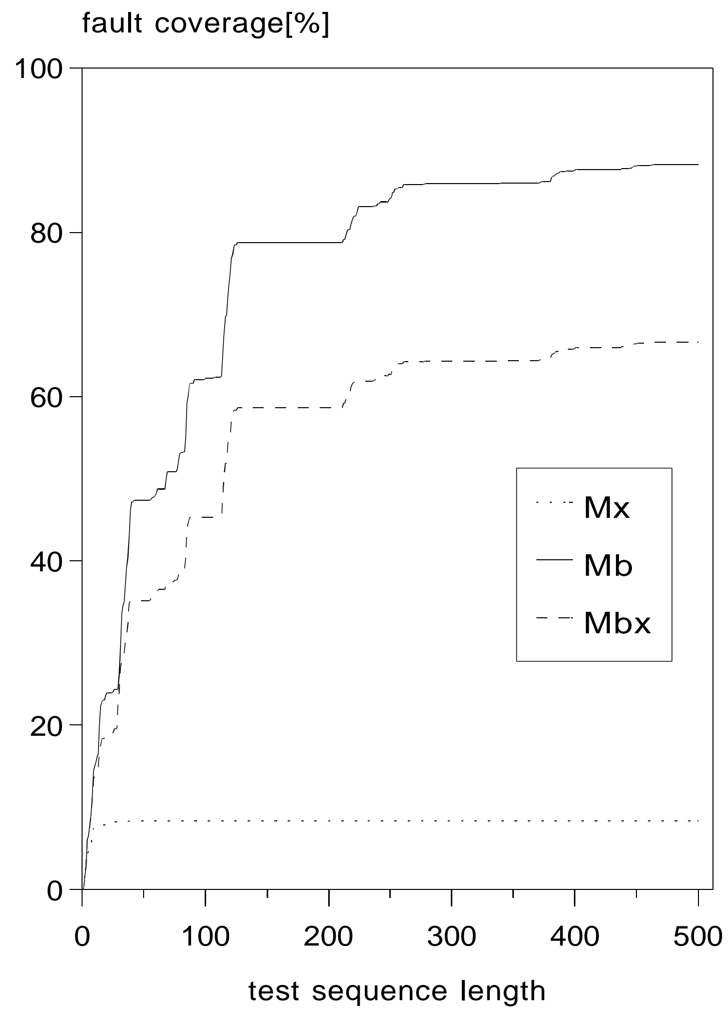

Figure 8: Fault coverage depending on the working mode for circuit s953.