Technical Report No. 105
May, 1998

Combining GAs and Symbolic Methods for High Quality Tests of Sequential Circuits

Martin Keim   Nicole Drechsler   Rolf Drechsler   Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Am Flughafen 17
79110 Freiburg im Breisgau, Germany
email: <keim, ndrechsler, drechsler, becker>@informatik.uni-freiburg.de

Abstract

A symbolic fault simulator is integrated in a Genetic Algorithm (GA) environment to perform Automatic Test Pattern Generation (ATPG) for synchronous sequential circuits. In a two phase algorithm test length and fault coverage as well are optimized. However, there are circuits, that are hard to test using random pattern sequences, even if these sequences are genetically optimized. Thus, deterministic aspects are included in the GA environment to improve fault coverage. Experiments demonstrate that both a priori time consuming strategies can be combined at reasonable costs: Tests with higher fault coverages and considerably shorter test sequences than previously presented approaches are obtained.

1 Introduction

Automatic Test Pattern Generation (ATPG) for synchronous sequential circuits is of large interest in computer aided design for integrated circuits and has been studied intensively. Two methodologies have been primarily used: Deterministic algorithms [22] and simulation based algorithms [29]. Deterministic ATPG algorithms have the advantage that they find a test sequence for a target fault if there exists one, or prove the fault as undetectable. However, deterministic ATPG algorithm tend to produce large test sequences. On the other hand side, simulation based ATPG algorithms usually consider all faults simultaneously, thus calculating considerably shorter test sequences. But these algorithms cannot guarantee to find a test sequence, nor can they prove a fault as undetectable. Finally, the quality of the test sequence computed by these algorithms is determined by the test sequence optimization algorithms. In this field of simulation based approaches *Genetic Algorithms* (GAs) [12] as an optimization paradigm have been demonstrated to be an interesting alternative (see e.g. [31, 30, 29, 7, 6]). In general, GAs provide optimization and search strategies which are used more and more in the area of VLSI CAD [10]. They are derived from observations in nature where living beings are improved by evolutionary mechanisms. Thus, each solution is denoted by an *individual* which is represented as a string over a fixed alphabet (*genotype*). A set of individuals is called a *population*. To each individual in a population a *fitness* is assigned based on an *objective function*. The objective function measures the quality of the solution that is encoded in an individual with respect to the problem formulation. Analogously to nature, the individuals in a population can reproduce themselves. This is simulated by recombining the genotypes of two selected parent elements. An element is selected proportional to its fitness, thus individuals (solutions) of higher quality are preferredly chosen.

In this paper we propose an ATPG algorithm based on simulation and a GA. Thus, the problem formulation is ATPG and a solution (an individual) represents a test pattern sequence. We measure the quality of a solution (the fitness) by means of fault coverage and test pattern sequence length. The main focus of the GA is quality of the tests at the expense of a reasonable amount of runtime. The idea is to split the ATPG into two phases: We use a "fast but inexact" simulation during the first ATPG-run to obtain tests for easy to detect faults. Thereafter an "exact" simulation is used during a second ATPG-run to determine tests for the remaining "hard" faults. Additionally, this second ATPG-run can start a probabilistic or a deterministic ATPG sub-routine to find a test sequence for a target fault. Thereafter this sequence is genetically optimized to increase the coverage of other faults. Experiments showed that a deterministic ATPG sub-routine is necessary to further improve the quality of the solution [28, 13], since basic GA showed not to be able to find good test sequences for all circuits [31, 29].

Exact simulation methods for sequential circuits are normally based on symbolic methods. The problem with symbolic approaches is in general, that they offer high evaluation quality on the one hand, but are rather costly on the other hand. Thus, at a first sight, combination of GA methods and symbolic methods seems to be even more problematic. Nevertheless, we will show in this paper for the case of ATPG of sequential circuits that symbolic methods can be advantageous also in a GA environment, if they are used in a flexible way.

As a symbolic fault simulator we use the hybrid BDD based fault simulator presented in [18] that supports not only the common SOT *(Single Observation Time Test Strategy)* but also the *Multiple Observation Time Test Strategy* (MOT) [23, 27, 24, 25] presented in [19], respectively. By this, the fault simulator gains not only from an accurate evaluation by BDDs, but also from an accurate test strategy, since MOT includes SOT. The experimental results show that MOT further improves the number of detected faults in particular in comparison to symbolic SOT. Additionally, with a restricted version of MOT (rMOT) this improvement in quality comes without any drawback in test execution. Unfortunately, the BDDs used for symbolic simulation can not be constructed for all functions, since too many BDD nodes are needed. For this, the simulator is able to switch between several simulation modes, i.e. three-valued, symbolic, and mixed (symbolic fault-free simulation / three-valued explicit fault simulation).

The GA used consists of two phases: At first, a test sequence is optimized with respect to fault coverage and test length for a given sequence of time frames. Then in the second phase the test sequence is enlarged to improve the fault coverage. Since we make use of the more time consuming and more accurate fault simulation procedure we additionally reduced the number of evaluations (fault simulation) during the GA run. This is done by an alternative selection strategy which allows to evaluate only "essential" individuals.

We describe experimental results to show the efficiency of the approach in comparison to previous work and show the effect of using a three-valued simulation or a hybrid method.

The paper is structured as follows: In Section 2 basic definitions concerning the underlying fault model, test pattern generation in general, and the symbolic fault simulator are described. Section 3 introduces the GA including representation of the elements and objective function. After the probabilistic and deterministic ATPG sub-routine is described, the complete ATPG tool is presented at the end of Section 3. Then the experiments are described in Section 4. Finally, the work is summarized.

2 Preliminaries

In this section we introduce the notations used and review some basic definitions. First, we define synchronous sequential circuits and the underlying fault model. Then the problem domain is formulated. Finally, we describe the fault simulator that is used in our approach.

2.1 Definitions

A synchronous sequential circuit can be described using a *Finite State Machine* (FSM). A FSM is a 5-tuple $M = (I, O, S, \delta, \lambda)$, where I is the input set, O is the output set and S is the set of states. $\delta : I \times S \to S$ is the next-state function and $\lambda : I \times S \to O$ is the output function. Since we consider a gate level realization of the FSM, we have $I = \mathbb{B}^k$, $O = \mathbb{B}^l$, and $S = \mathbb{B}^m$ with $\mathbb{B} = \{0, 1\}$. k denotes the number of primary inputs, l denotes the number of primary outputs, and m denotes the number of memory elements. The functions δ and λ are computed by a combinational circuit C. The inputs (outputs) of the combinational circuit, which are connected to the outputs (inputs) of the memory elements, are called secondary inputs (outputs). Sometimes the secondary inputs are called *present state variables* and the secondary outputs are called *next state variables*.

For the description of our algorithms we use the following notations: $Y = y(1), \ldots, y(n)$ denotes an input sequence of length n. $s(s_0, 0), s(s_0, 1), \ldots, s(s_0, n)$ denotes the state sequence defined by $s(s_0, 0)$ and Y. $s(s_0, 0) = s_0$, $s_0 \in S$, is called the initial state. In our experiments, we do not presume a given reset state or reset sequence, nor do we assume a (fault free) full scan environment. Notice, that this is different to several other ATPG approaches.

$o(s_0, 1), o(s_0, 2), \ldots, o(s_0, n)$ denotes the output sequence defined by the initial state s_0 and Y. $o_i(s_0, t)$, $1 \le i \le l$, denotes the value at the i-th primary output after simulation step t. Using these notations the next

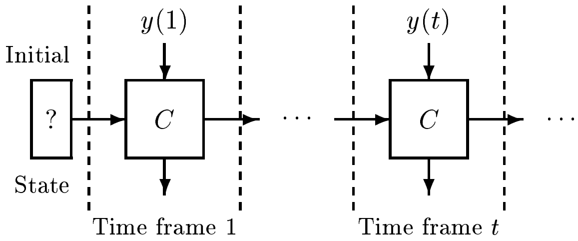

Figure 1: Iterative description of a sequential circuit.

state is given by

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

Analogously, the output $o(s_0, t)$ is defined by the function λ. In doing so, we consider a synchronous sequential circuit as an iterative network (see Figure 1).

2.2 Fault Model and ATPG Problem Formulation

It is well-known that, even if the circuits are correctly designed, a fraction of them will behave faulty because of physical defects caused by imperfections during the manufacturing process. Fault models, which cover a wide range of the possible defects, have to be defined, and tests for faults in the fault models have to be constructed. The fault model adopted in this paper is described in the following.

The *Stuck-At Fault Model* (SAFM) [11, 3] is well-known and used throughout the industry. We use the single-fault assumption, i.e., we assume that at most one fault can occur in the circuit. 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 the states s^f and outputs o^f are defined analogously.

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

Definition 1 *: A single stuck-at fault is detectable with respect to SOT by an input sequence $Y = (y(1), y(2), \ldots, y(n))$ if*

$$\exists b \in \{0,1\}, \exists t \leq n, \exists i \leq l, \forall p, q \; : o_i(p,t) = b \text{ and } o_i^f(q,t) = \overline{b}$$

with p an initial state of the fault-free machine and q an initial state of the faulty machine.

The definition of MOT seams very similar. We will point out the difference afterwards.

Definition 2 *: A single stuck-at fault is detectable with respect to MOT by an input sequence $Y = (y(1), y(2), \ldots, y(n))$ if*

$$\forall p, q \, \exists t \leq n, \exists i \leq l, \exists b \in \{0,1\} \; : o_i(p,t) = b \text{ and } o_i^f(q,t) = \overline{b}$$

with p an initial state of the fault-free machine and q an initial state of the faulty machine.

According to SOT, a fault is detectable, if there is an *unique* point in time, such that independent of the initial states of both machines the boolean output values on a particular primary output are inverse to each other. According to MOT, there is an *individual* point in time for each possible initial state pair. This means that for the initial pair p_1, q_1 the fault could be detected, let's say in the third time frame, but for another pair p_2, q_2, the very same fault could be detected in time frame twenty. A fault is detectable, if for all possible pairs, there exists such a time frame. Hence, it is clear that MOT includes SOT. The example in Figure 2 is taken from [25]. It illustrates the advantage of MOT over SOT.

This figure shows two time frames of a circuit consisting of an AND and an EXOR gate. The circuit has one primary output at the bottom of the figure, and two primary inputs shown on top of the figure. A stuck-at 1

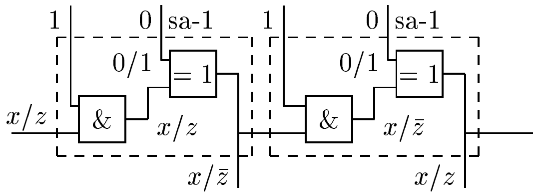

Figure 2: Example for MOT.

fault is located on the second primary input. Assume, the sequence to simulate is $Y = (\ (1{,}0),\ (1{,}0)\ )$. With x we denote the state of the correct machine; with z that of the faulty machine. There are four possible initial state pairs: $(x, z) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. These four pairs can be separated into two cases: $x = z$ and $x \neq z$: For $(p_1, q_1) \in \{(0, 0), (1, 1)\}$ there is a difference at the primary output at time frame one, but there is no difference at the primary output at time frame two. For $(p_2, q_2) \in \{(0, 1), (1, 0)\}$ there is a difference at the primary output at time frame two, but there is no difference at the primary output at time frame one. Hence, this fault is undetectable according to SOT, since there is no single time frame for that all initial state pairs cause an output to differ. But considering MOT, for all pairs (p_1, q_1) the fault is detectable at time frame one, and for all pairs (p_2, q_2) the fault is detectable at time frame two. This means that for all possible initial state pairs, there is a detecting time frame. Therefore the fault is detectable according to MOT after the second time frame.

The ATPG problem can now be formulated as:

> *How can we determine a test sequence such that a given fault f is detected by this sequence, starting from an unknown initial state ?*

To solve the problem we combined a genetic algorithm with an accurate fault simulator, as explained in the introduction. In the following, we introduce the components of our ATPG tool.

2.3 Hybrid Fault Simulator

Basically, the purpose of fault simulation is to determine, which faults are detected by a given sequence of input values, according to Definition 1 and 2. A common strategy for handling the problems induced by an unknown initial state is given by a fault simulation procedure based upon a three-valued logic $\mathbb{B}_X := \{0, 1, X\}$. X denotes the *unknown* or *undefined* value. It is used for encoding the unknown initial state, which is given by $s_0 = (X, X, \ldots, X)$.

As is well-known, fault simulation based upon the three-valued logic is rather inaccurate. Many papers mention this problem (see e.g [5, 21]). The inaccuracy can be explained on the one hand by the binary encoding of the states [21] and on the other hand by the logic realizing the next state function. This results in the following: There may exist detectable faults, that cannot be detected with respect to a given input sequence Y using fault simulation, based upon the three-valued logic. Therefore, fault simulators based upon this logic determine only a lower bound for the real fault coverage.

A lot of work has been done to overcome these problems. One approach is the application of symbolic traversal techniques to test generation [5, 32]. This approach is based upon *ordered Binary Decision Diagrams* (BDDs) [4]. Using this symbolic methods, the *exact* fault coverage with respect to a given input sequence Y according to Definitions 1 and 2 is determined.

Since we make use of features of the event-driven *Hybrid Fault Simulator* (*HFSim*) [18, 19], we shortly review its functionality in the following. In this context *hybrid* means, that the algorithm supports different logics for simulation and allows a dynamic automated switching to the logic used for each simulation step. More precisely, it consists of three kinds of fault simulation procedures:
(1) a fault simulation procedure based upon the three-valued logic,
(2) a symbolic fault simulation procedure based upon BDDs, and
(3) a combined fault simulation procedure, with a symbolic true value simulation and a three-valued based explicit fault simulation procedure.

Formally, the symbolic (fault) simulation is based upon the logic $\mathbb{B}_m = \{g : \mathbb{B}^m \to \mathbb{B}\}$. For the simulation, an element of the logic is represented by a BDD. To encode the unknown initial state a Boolean variable x_i,

$1 \leq i \leq m$, for each memory element is introduced. 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 according to Definition 1, if there exists an output for which M and M^f compute different constant functions $\mathbf{0}$ or $\mathbf{1} \in \mathbb{B}_m$.

To detect a fault according to Definition 2, it is necessary to compare *sets* of fault-free responses with *sets* of responses obtained in the presence of faults. This is especially complicated if long test sequences and a large number of memory elements exist. By the use of BDDs it becomes possible to handle these output sequences. Thus, we succeed in computing the *exact* MOT fault coverage for many of the considered benchmark circuits. Details can be found in [19].

HFSim possesses a restricted version of MOT, called rMOT. The restriction is, that the detection of a fault at a primary output is checked only if the fault-free machine's output value is Boolean. The rMOT strategy is very attractive if the circuit considered has a reset state or a reset sequence. In contrast to the general MOT strategy with the rMOT strategy there is no drawback in test execution: It allows a test evaluation by comparing the output sequence of the circuit-under-test with the *unique* output sequence of the fault-free circuit *as usually*. We show by experiments that the accuracy of fault simulation based on rMOT is identical with that based on MOT for nearly all circuits considered here, if the computed test sequence is applied to the circuit. With regard to the performance it can be observed that for some circuits a symbolic rMOT fault simulation works even more efficiently than a symbolic SOT fault simulation.

3 Test Pattern Generation based on Genetic Algorithms

In this section the GA that is used for generating test sequences for sequential circuits is described. Firstly, the basic idea of the method is given. Then we present the representation of test candidates and the objective function and provide some insight into the parameter settings of the GA. Before the complete ATPG tool is described, the probabilistic and deterministic approach included in the tool is shown.

3.1 The Main Idea

Our environment for ATPG makes use of two main GA phases. A first phase optimizes test sequences to yield high fault coverage. In the second phase the GA enlarges this optimized test sequence to improve the results. The approach is based on genetic operators and the hybrid fault simulator. Additionally, the performance with respect to runtime can be increased while using the *tournament selection* [9] which reduces the number of simulations tremendously.

3.2 Representation, Objective Function, Selection, and Genetic Operators

An individual in a population $\mathcal{P}$ is a binary string that represents a test sequence Y of n time frames. The maximum length of the individuals is fixed, i.e. each test sequence in a population has an upper bound in length. Initially, these binary strings are randomly chosen.

The objective function computes the fitness for each element in the present population. It measures the number of undetected faults and the length of the test sequence. Thus, the value of objective function has to be minimized. This objective function consists basically of the fault simulation procedure *HFSim* as described in Subsection 2.3. Notice, that our primary optimization goal is the improvement of the fault coverage. The minimization of the length of the test sequences is only a second optimization criterion. However, by this combination we get not only high fault coverages but the final test sequences are very short in comparison to other approaches, also (see the experimental results).

The selection of the elements is done by *2-tournament selection*: For each parent two elements are chosen randomly. Only these four elements are evaluated by the objective function. The elements with higher ranking are selected. By this for the selection of two parents only four elements are evaluated instead of the whole population.

For comparison we used the *roulette wheel selection* [9] in another series of experiments: Each element of the population is evaluated by the objective function. The parent elements are chosen randomly, where the chance of being selected corresponds to their fitness.

Genetic operators modify the individuals in the populations. In our GA applications we make use of "standard" operators like *crossover* [9] and several problem specific operators. All genetic operators generate only feasible solutions, if they are applied to binary strings. To make the paper self-contained, we here shortly review the genetic operators [16] used in the application:

Mutation (MUT): Construct one new element c from a parent p by copying the whole element and changing a value at a randomly chosen position.

2-time Mutation: Perform MUT two times on the same element.

Mutation with neighbor: Perform MUT at two adjacent positions on the same element.

Crossover: Construct two new elements c_1 and c_2 from two parents p_1 and p_2, where p_1 and p_2 are split in two parts at a cut position. The first (second) part of c_1 (c_2) is taken from p_1 and the second (first) part is taken from p_2. (Notice, that a special case of this operator is the *horizontal crossover* from [7], where the cut position is chosen only between two test vectors, i.e. test vectors are not split up.)

2-time Crossover: Construct two new elements c_1 and c_2 from two parents p_1 and p_2, where p_1 and p_2 are split in three parts. The first (second) part of c_1 (c_2) is taken from p_1 (p_2), the second part is taken from p_2 (p_1) and the last part is again taken from p_1 (p_2).

Uniform Crossover: Construct two new elements c_1 and c_2 from two parents p_1 and p_2, where at each position the content is taken with a certain probability from p_1 and p_2, respectively.

Free Vertical Crossover: Construct two new elements c_1 and c_2 from two parents p_1 and p_2. Determine for each test vector t of p_1 and p_2 a cut position. Divide each t in two parts at its cut position. The first (second) part of each test vector of c_1 (c_2) is taken from p_1 and the second (first) part is taken from p_2. (Notice, that the *vertical crossover* from [8] is a special case of this operator, if the split position is equal for all test vectors t.)

3.3 Genetic Algorithm and Parameter Setting

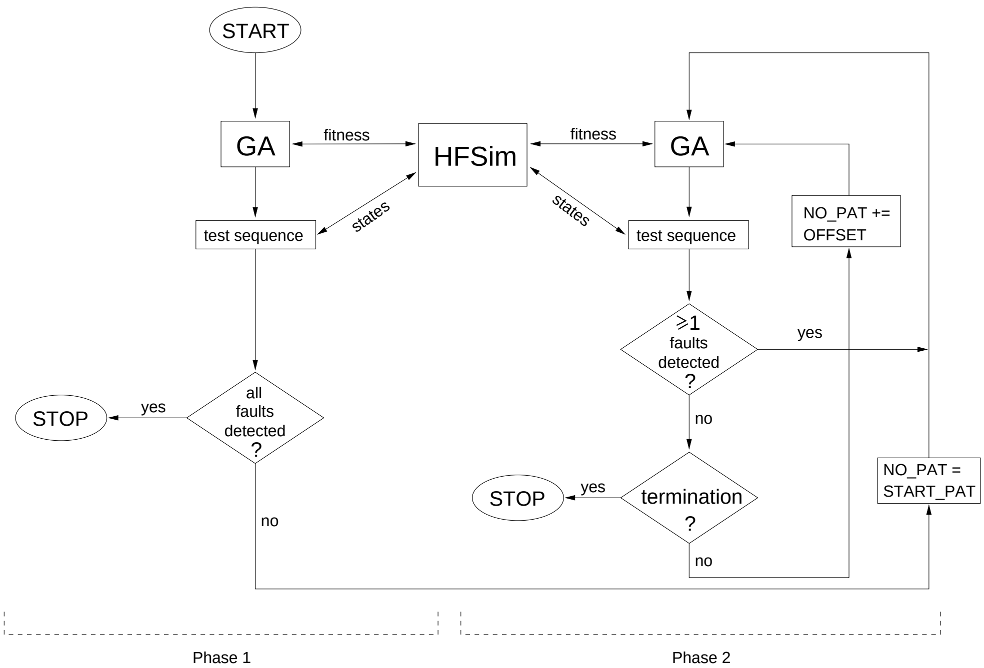

Figure 3: Structure of the GA.

As mentioned above the GA based ATPG algorithm consists of two phases. For illustration a structure of the GA is given in Figure 3.

Phase 1: The algorithm starts optimizing a test sequence of fixed length, beginning in a given starting state, e.g. the unknown initial state. This test sequence Y is optimized with respect to fault coverage and length of the test sequence.

Phase 2: Tests are generated for the remaining undetected faults, i.e. the test sequence Y from *Phase 1* is enlarged by test vectors. For this, the GA does not start in the given starting state but in the state $s(s_0, |Y|)$ that results from applying Y, the test sequence found so far, to the circuit.

The second phase of the GA is started using a number of additional test vectors. This number NO_PAT is initially set to START_PAT. If the GA terminates successfully, e.g. test vectors have been found, these test vectors are suffixed at the end of Y. If no fault has been detected, the length NO_PAT of the additional test sequence is incremented by OFFSET. This newly generated test sequence composed of Y and the new (test) vectors is denoted by Y_{new}. Then the GA of the second phase is continued in the new present state $s(s_0, |Y_{new}|)$. This is repeated, until no further fault is found for a given upper bound.

The size of the population in *Phase 1* is set to $|\mathcal{P}| = 32$ and in *Phase 2* it is set to $|\mathcal{P}| = 16$. If *Phase 2* runs in the three-valued mode the population size is enlarged to $|\mathcal{P}| = 32$, since the three-valued simulation runs much faster and thus more time can be spend for different evaluations of the GA. Each recombination operator is performed with a probability of 20% and the mutation rate is set to 5%. The magnitude of probabilities for the operators showed to be a good choice in several other occasions. (Thus, not such many experiments were necessary to determine parameters that work fine for all circuits.)

3.4 Probabilistic and deterministic ATPG

As mentioned, some circuits, e.g. *s820* and *s832*, cannot be tested sufficiently using random pattern sequences. Furthermore, as our experimental results show and those of other researchers [31, 29], no high fault coverages have been achieved for these circuits using basic GAs. On the other hand side, deterministic ATPG algorithms can achieve fault coverages of more than 93%. Recently, GAs have been presented that additionally use deterministic ATPG aspects to minimize this weakness, e.g. [28, 13]. In both works the deterministic test generator HITEC [22] have been used to determine a test sequence for initializing the individuals ([13]) of the GA or to excite and propagate the faults ([28]). Our approach, although it is much simpler, provides higher fault coverages and shorter test sequences, as the experimental results show: The initial population of *Phase 2* is determined according to an *example primary input sequence*. How this sequence is computed is explained next.

After the three-valued based GA-ATPG has determined a test sequence, this sequence is tried to be extended by a BDD-based forward propagation algorithm. (Details how the separate parts of the GA-ATPG tool are linked together are given in the next section.) Starting from the reached state, a fault detection BDD is constructed, i.e. a BDD for a circuit for that the primary outputs of the fault-free and of the target faulty circuit are XORed together and these XOR-outputs are ORed together; the primary inputs are shared. A fault is detectable according to SOT, iff there is an assignment to the primary input-variables such that for all secondary input-variables the fault detection BDD evaluates to 1. In other words, a test pattern equals a 1-path in the fault detection BDD, all secondary input-variables universally quantified out. If there is no such assignment, the fault cannot be detected in the current time frame. In this case, the fault detection BDD of the next time frame is constructed, defined over two sets of primary input variables and considering the according present state.

This found test pattern sequence (i.e. the 1-path in the quantified BDD) defines the initial values of the individuals of the current GA-population. Since usually only a few primary input-variables occur on this 1-path only the according positions on the bit-string representing an individual are initially set fixed to either 1 or 0. The value of all other non occurring variables is randomly determined.

To compute a test sequence according to MOT, a similar fault detection BDD must be build. However, there are two sets of secondary input-variables. One for the fault free and one for the faulty circuit. The fault detection BDD of the current time frame is paired with the fault detection BDDs of all former time frames. Thus, a fault is detectable, iff there is a primary input-variable assignment, such that the output sequence produced by the circuit-under-test cannot be determined by any initial state of the fault free circuit. Details on symbolic (r)MOT fault simulation can be found in [19].

Since the fault detection BDD is constructed using the current state of the circuits that is largely initialized, the BDD-forward simulation described above works fine for five up to several ten time frames depending on the complexity of the BDDs. Finally, the BDD-sizes will exceed a given size limit. In order to symbolically simulate a much larger number of time frames another algorithm is implemented. It bases on signal probabilities (i.e. the 1-probability of a signal line in a circuit) and fault detection probabilities, and the fact that both probabilities can be computed exactly and very easily using BDDs [17]: A single MOT fault detection BDD is constructed not considering the current state, i.e. secondary input-variables are assigned to all secondary inputs of the circuit. Also, for each secondary output of the fault free and for each secondary output of the faulty circuit a BDD is constructed. The 'forward simulation algorithm' works now as follows:

1. Compute the signal probability of all secondary outputs of the former time frame.
2. Assign this value to the current secondary input-variables.
3. Compute the 1-probability of the fault detection BDD, i.e. compute the fault detection probability.
4. Continue with step 1 for a given number of time frames.

Since the BDD is iteratively evaluated and no BDD-operations are performed, the algorithm works very efficiently. Its speed depends only on the static size of the BDDs. However, this algorithm does make an inherent error: Within each time frame all values are exactly computed, but the algorithm does not consider signal dependencies across time frames. Thus, the computed values are only an approximation. But for two reasons this approximation is sufficient:

1. Extensive experiments showed that the relative error between the approximation and the real value is (very) small in most cases: 23 circuits of the ISCAS'89 benchmark set have been examined (s27 – s1494). For 75.56% of all signal lines the relative error is less than 10%. For nearly 80% of all lines the relative error is less than 20%. (This experiments have been performed for the fault free circuit.)
2. The computed input probability vector is finally used to determine the initial population of *Phase 2* of the GA. This means, it is sufficient that the input probability vector gives a good *hint* in a promising direction.

Changing the 1-probability value of the primary input-variables (input probability) changes the fault detection probability. Thus, optimizing the fault detection probability means manipulating the vector of input probabilities. This optimization task is done by an instance of the GA, described in this section. Note, that the individuals are no longer bit-vectors, but vectors over floating point numbers. Each element of an individual represents the input probability of the according primary input at a distinct time frame.

Finally, the optimized input probability vector is handed to calling *Phase 2* of the GA. According to this vector the initial values of the bit-strings of the individuals are determined by comparing a random number with the approximated input probability: If the random number is smaller than the input probability, the according element of the individual is set to 1, otherwise it is set to 0.

3.5 The ATPG Tool

Now all components of our ATPG tool have been presented. In the following we show the possibilities of the sequence these tools are composed together to achieve high quality test pattern for sequential circuits. Figure 4 shows the different ways to compute such a test sequence. At first, the tool has the possibility to select a three-valued greedy reset heuristic [15]. If selected, the sequence computed by the heuristic (partially) resets the correct circuit-under-test, and thus is the first part of the test sequence. After that, the GA determines a test sequence, using the three-valued mode of *HFSim* only. Thereafter this sequence is symbolically simulated by one of the three symbolic modes of *HFSim* (SOT, rMOT, MOT). Furthermore, the test sequence is enlarged by the GA now using one of the symbolic modes. Note, that each next step in our ATPG starts in the state reached by applying the test sequence found so far to the circuit-under-test.

In the first iteration of *Phase 2* in the symbolic case, the elements of the test pattern are chosen randomly with a 1-probability of 50%. Since there are faults, that are hard to detect with such a sequence, *Phase 2* is iterated with different 1-probabilities. In our experiments, we let Phase 2 run eleven times, from 0% to 100% 1-probability in 10% steps.

In the case of the input probability vector construction and in the case of deterministic test pattern computation the GA skips *Phase 1* after the symbolic simulation of the test sequence. It begins with computing the first input probability vector or the first deterministic test sequence, respectively. The result is handed to *Phase 2* of the regular GA for pattern creation as described in the last section. In both cases, the probabilistic and the deterministic one, the GAs use the roulette-wheel selection method.

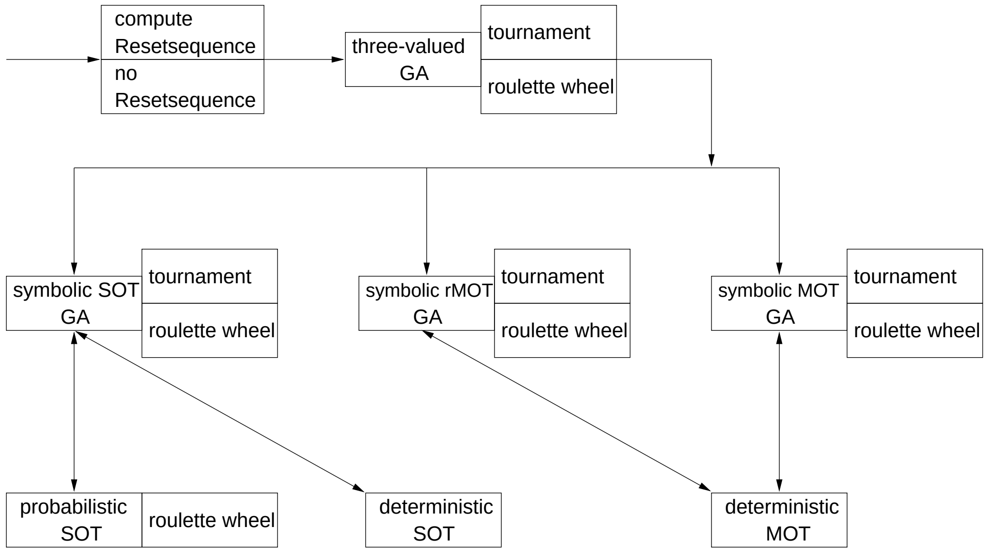

Figure 4: Structure of the ATPG algorithm

4 Experimental Results

To show the efficiency of the approach we apply the ATPG algorithm to several sequential benchmark circuits from *ISCAS 89*. We compare the algorithm to different other GA based methods ([29, 28, 13]) and a to *Best of* classical ATPG algorithms ([22, 2, 26, 14]). Note, that the results of [26] also include a test sequence compaction method. The GA-ATPG of [29] does not consider any deterministic ATPG aspects, whereas both other do [28, 13].

Since we do not assume a given fault free reset state or assume a given reset sequence, nor do we assume a (fault free) full scan environment, we do not compare with several other ATPG algorithms. Especially the genetic ATPGs [7, 8] are not not considered for comparison. Obviously, our algorithm can also be applied in this case, since assuming a fault free reset and performing then an ATPG is a subproblem of the one considered here. Table 1 summarizes the fault coverages (`FC`) and test length (`|Y|`) of the mentioned (GA-) ATPG algorithms found in literature. After the circuit name, the number of primary inputs and the number of secondary inputs is given.

The first set of experimental results is shown in Table 2. The columns denoted by $\texttt{GA}_{X01}$ shows the results obtained by our method with the restriction to a pure three-valued simulation, i.e. we do not make use of the symbolic techniques. The extension `to` denotes the experiments using the tournament selection, whereas the extension `ro` denotes the experiments using the roulette wheel selection. For these experiments *Phase 1* starts in the unknown initial state of the circuit. The columns labeled with `re` shows the experimental results using the greedy reset heuristic. Accordingly, *Phase 1* starts in the state reached after applying the sequence.

Note, that all results are achieved with *only one* parameter setting of the GA and no manual intervention is needed. However, for almost all circuits we achieve better results in fault coverage and/or test sequence length than [29]. This is especially true for the largest benchmarks considered, *s5378* and *s35932*. Usually our test sequence length is only a fraction of that of [29]. Nevertheless, there are two circuits, *s820* and *s832*, that are hard to test using the genetic approach, as the results of $\texttt{GA}_{X01}$ and [29] show. On the other hand, for *s526* our three-valued ATPG improves the fault coverage also in comparison to the deterministic ATPGs of Table 1.

Comparing the much faster tournament selection method with the roulette wheel selection method it can be observed, that the faster one achieves surprisingly good results. The roulette wheel selection method is only slightly better. However, for four circuits the tournament selection method is superior (Table 2).

If the GA no longer finds an improvement using the three-valued mode of *HFSim*, the test sequence Y is handed to the GA-ATPG using symbolic evaluation and the same selection method (tournament or roulette wheel) as before. Now *Phase 1* starts in the (symbolic) state achieved after the simulation of Y.

We made three different experiments: GA_{HFSim}, using the input probability computation routine and GA_{HFSim}, and finally using the deterministic test sequence computation routine and GA_{HFSim}. The experimental results

name	*PI*	*SI*	Classical ATPG Best of [22, 2, 26, 14] FC	$\|Y\|$	Genetic ATPG [29] FC	$\|Y\|$	[28] FC	$\|Y\|$	[13] FC	$\|Y\|$
s298	3	14	88.04	306	85.94	161	86.04	415	86.04	221
s344	9	15	96.20	37	96.20	95	95.91	169	96.20	75
s349	9	15	95.71	137	95.71	95	95.71	188	95.71	104
s386	7	6	81.77	121	76.88	154	81.77	359	81.77	241
s400	3	21	90.14	2424	85.70	280	81.22	704	90.14	2196
s444	3	21	89.45	1945	85.59	275	80.38	880	89.45	1046
s526n	3	21	81.80	2642	75.08	281	67.75	873	81.62	2109
s641	35	19	86.51	63	86.51	139	86.51	292	86.51	140
s713	35	19	81.93	176	81.93	128	81.93	294	81.93	185
s820	18	5	95.76	424	60.76	146	95.76	1108	95.76	1243
s832	18	5	94.02	701	61.95	150	94.02	1064	94.02	1109
s953	16	29	8.87	20	–	–	–	–	–	–
s1196	14	18	99.76	244	99.19	347	99.76	377	99.76	948
s1238	14	18	94.69	247	94.02	383	94.69	409	94.69	880
s1488	8	6	97.17	317	93.67	243	97.17	1369	97.17	1191
s1494	8	6	96.48	540	94.02	245	96.48	1224	96.48	985
s5378	35	179	79.06	11571	68.98	511	70.35	683	76.17	7270
s35932	35	1728	87.78	257	89.55	197	89.17	425	89.77	825

Table 1: Experimental results found in literature.

name	GA_{X01} to FC	$\|Y\|$	GA_{X01} ro FC	$\|Y\|$	GA_{X01} to,re FC	$\|Y\|$	GA_{X01} ro,re FC	$\|Y\|$
s298	86.04	128	86.04	79	86.04	135	86.04	95
s344	96.20	61	96.12	51	96.20	72	96.20	56
s349	95.71	61	95.71	65	95.71	64	95.71	57
s386	79.65	99	78.91	96	80.47	128	78.91	125
s400	56.13	82	81.12	209	57.78	73	90.09	443
s444	60.55	47	60.76	42	88.19	372	60.55	51
s526n	80.65	639	81.92	820	81.74	812	81.74	717
s641	86.51	99	86.51	67	86.51	95	86.51	64
s713	81.93	81	81.93	79	81.93	99	81.93	74
s820	58.12	121	68.35	106	67.76	145	64.71	143
s832	57.01	118	69.89	114	60.23	128	67.24	86
s953	8.34	17	8.34	19	8.34	16	8.34	17
s1196	99.52	346	99.60	243	99.28	335	99.60	233
s1238	94.46	458	94.46	342	94.17	318	94.61	231
s1488	97.17	367	96.97	280	96.90	335	97.11	301
s1494	96.28	332	72.21	305	96.15	267	96.35	299
s5378	70.15	222	72.50	180	70.19	274	70.11	203
s35932	89.78	163	89.78	163	89.78	170	89.78	191

Table 2: Experimental results using three-valued evaluation only.

are given in Table 3. After the name of the circuit, the best three-valued result from Table 2 is given for comparison. The next three columns belong to the corresponding symbolic mode of *HFSim*. The first column, labelled `Symb. Sim.` shows the fault coverage by simply simulating the three-valued test sequence symbolically. The next two columns give the improvement. A '–' denotes, that the according method obtains no further improvement in comparison to the symbolic simulation. Note, that this table also shows results for circuit s510, that is not included in former tables, since this circuit is untestable using basic three-valued ATPG. A '*' means that the according computation has been terminated due to too complex simulations or too many faults to consider.

As expected, the symbolic simulation of Y improves the fault coverage. However, application of the GA results in a further increase of fault coverage. But for both hard to test circuits *(s820, s832)*, GA_{HFSim} cannot

name	Best GA_{X01} FC	Best GA_{X01} $\|Y\|$	Symb. Sim. FC	GA_{HFSim} Genetic only FC	GA_{HFSim} Genetic only $\|Y\|$	GA_{HFSim} Probabilities FC	GA_{HFSim} Probabilities $\|Y\|$	GA_{HFSim} Deterministic FC	GA_{HFSim} Deterministic $\|Y\|$
s298	86.04	79	87.01	87.66	91	87.66	94	87.66	90
s344	96.20	56	97.37	–	–	–	–	–	–
s349	95.71	57	96.86	–	–	–	–	–	–
s386	80.47	128	80.47	80.73	142	81.25	172	81.77	160
s400	90.09	443	90.09	90.33	444	92.22	655	92.22	691
s444	88.19	372	88.19	88.40	373	90.08	529	90.08	557
s510	0.00	–	–	100.00	252	99.82	341	100.00	245
s526n	81.92	820	83.18	–	–	83.36	860	83.36	879
s641	86.51	64	87.37	–	–	–	–	–	–
s713	81.93	74	82.62	–	–	–	–	–	–
s820	68.35	106	68.47	68.59	131	85.06	727	95.88	412
s832	69.89	114	70.00	70.11	139	82.76	720	94.14	400
s953	8.34	16	26.04	99.07	253	99.07	294	99.07	196
s1196	99.60	233	99.60	–	–	–	–	99.76	236
s1238	94.61	231	94.61	–	–	–	–	94.69	233
s1488	97.17	367	97.31	–	–	–	–	–	–
s1494	96.48	305	96.61	–	–	–	–	–	–
s5378	72.21	180	72.50	72.52	182	*	*	74.99	303
s35932	89.78	163	89.78	–	–	*	*	–	–

Table 3: Experimental results using different symbolic evaluation methods (SOT).

SOT				
	Without s953		With s953	
Method	FC	$\|Y\|$	FC	$\|Y\|$
Det. GA_{HFSim}	100.00	100.00	100.00	100.00
Prob. GA_{HFSim}	98.35	108.72	98.45	110.26
GA_{HFSim}	96.19	76.72	96.42	78.68
Symb. Sim.	96.08	75.13	91.87	72.63
Best GA_{X01}	95.63	75.13	90.37	72.63
[13]	99.40	411.49	–	–
[28]	96.90	214.64	–	–
[29]	92.82	75.89	–	–
Best of [22, 2, 26, 14]	99.73	446.38	94.24	430.08

Table 4: Comparing the experimental results (SOT).

obtain an improvement, also. However both advanced method, the probabilistic and the deterministic, compute comparable fault coverages, that are even a little bit better than those shown in Table 1. Nevertheless, the probabilistic approach seams to mislead GA_{HFSim} sometimes with respect to the test length, see circuits *s298* and *s953*. Also, for circuit *s510* no 100% FC have been achieved. However comparing the probabilistic and the deterministic GA_{HFSim}, the first one determines shorter test sequences (if the FC equals) than the second one.

Table 4 compares the different SOT ATPG algorithms. Since not all researchers report results for *s953* two columns have been made. One does not consider the results for *s953*, the other does. For both columns the results for s510 have not been considered. Assume the FC and test length of the deterministic GA_{HFSim} to be 100.00%, that table shows the ratio of these values of the algorithms of Tables 1 and 3. Obviously, our GA-ATPG computes higher fault coverages than all other presented ATPGs, even if the advantage is not much compared to [13] and the best of [22, 2, 26, 14]. However, our test sequence length is only a fraction – less than 1/4 – of that shown in these two lines.

Experiments concerning MOT are shown in Table 5. It shows also that the change of the test method from SOT to rMOT further improves the fault coverage and shortens the test sequence in comparison to symbolic

name	MIX		Best GA_{X01}		rMOT Sim.	GA_{HFSim} rMOT				MOT Sim.	GA_{HFSim} MOT			
						Genetic only		Deterministic			Genetic only		Deterministic	
	FC	$\|Y\|$	FC	$\|Y\|$	FC	FC	$\|Y\|$	FC	$\|Y\|$	FC	FC	$\|Y\|$	FC	$\|Y\|$
s298	88.64	206	86.04	79	88.31	88.64	86	88.64	90	88.31	88.64	86	88.64	90
s344	97.95	76	96.20	56	97.95	–	–	–	–	97.95	–	–	–	–
s349	97.43	104	95.71	57	97.43	–	–	–	–	97.43	–	–	–	–
s386	81.77	226	80.47	128	80.47	80.73	142	81.77	160	80.47	80.73	142	81.77	160
s400	93.40	1170	90.09	443	92.22	92.45	444	93.63	706	92.22	92.45	444	93.41	706
s444	91.77	870	88.19	372	90.51	90.72	373	92.41	929	90.51	90.72	373	92.41	929
s510	100.00	587	0.00	–	–	100.00	337	*	*	–	100.00	309	*	*
s526n	83.24	1545	81.92	820	84.09	–	–	84.27	879	84.09	–	–	84.27	879
s641	87.37	131	86.51	64	87.37	–	–	–	–	87.37	–	–	–	–
s713	82.62	130	81.93	74	82.62	–	–	–	–	82.62	–	–	–	–
s820	95.88	838	68.35	106	68.47	68.59	131	95.88	412	68.47	68.59	131	95.88	412
s832	94.14	881	69.89	114	70.00	70.11	139	94.14	400	70.00	70.11	139	94.14	400
s953	99.07	563	8.34	16	27.90	99.07	217	99.07	188	27.90	99.07	212	99.07	188
s1196	99.76	306	99.60	233	99.60	–	–	99.76	236	99.60	–	–	99.76	236
s1238	94.69	347	94.61	231	94.61	–	–	96.69	233	64.61	–	–	96.69	233
s1488	97.31	918	97.17	367	97.31	–	–	–	–	97.31	–	–	–	–
s1494	96.61	759	96.48	305	96.61	–	–	–	–	96.61	–	–	–	–
s5378	79.14	1766	72.21	180	72.54	72.56	182	75.21	273	72.54	72.56	182	75.21	273
s35932	89.81	296	89.78	163	89.81	–	–	–	–	89.81	–	–	–	–

Table 5: Experimental results using different symbolic evaluation methods (rMOT and MOT).

rMOT Without s510		
Method	FC	$\|Y\|$
Det. GA_{HFSim}	100.00	100.00
GA_{HFSim}	94.51	73.03
Symb. Sim.	90.14	68.10
Best GA_{X01}	89.88	68.10
MIX	100.32	199.07

Table 6: Comparing the experimental results (MOT).

SOT for all but one circuit. Changing to MOT gives nearly no further improvement (only for circuit *s510*), since most correct circuits are early initialized by Y, thus MOT and rMOT do no longer differ. An advantage can be achieved only for circuits, that are hard or impossible to initialize, or *during* the reset sequence. Comparing to MIX [20] our (r)MOT GA_{HFSim} is better concerning the FC for only three out of 19 circuits and is worse for one (circuit *s5278*). However, in comparison to MIX our test sequence length is only half as long (Table 6).

The execution time of the symbolic part of the ATPG algorithm is, in general, only some minutes. The accelerated execution compared to a pure symbolic GA-ATPG from the beginning is due to the following facts:

- most faults have already been found
- the correct circuit and
- many faulty circuits are initialized
- therefore, the memory demand of the symbolic simulation is small due to small BDDs
- and, finally, the number of switchings of *HFSim* between several logics is considerably reduced.

Thus, the effort of simulation is small. For example circuit *s5378*: $\texttt{GA}_{X01}$ using tournament selection takes an execution time of 6h 50 minutes, whereas the GA_{HFSim} SOT (MOT) part needs time less than 11 (54) minutes on a Sparc Ultra 1. However, the cumulative three-valued simulation time is high. This is, because we do perform a single-fault single-pattern simulation until now. Thus, a parallel version would be helpful, and is focus of our current work.

5 Conclusions

In this paper we presented a *Genetic Algorithm* (GA) for *Automatic Test Pattern Generation* (ATPG) for sequential circuits. We outlined the advantages of using hybrid symbolic fault simulation concerning the quality of the fault coverages.

The experiments have demonstrated that our ATPG tool can generate high quality test patterns in both, fault coverage and test length: The test length is reduced to 1/4 with respect to test length of comparable ATPGs without neglecting the fault coverage. This quality is mainly based on a high quality GA. Applying symbolic methods, in particular rMOT, does further improve the quality of the test sequence.

References

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

[2] M. Abramovici, K.B. Rajan, and D.T. Miller. Freeze: A new approach for testing sequential circuits. In *Design Automation Conf.*, pages 22–25, 1992.

[3] M.A. Breuer and A.D. Friedman. *Diagnosis & reliable design of digital systems.* Computer Science Press, 1976.

[4] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] 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.

[6] F. Corno, P. Prinetto, M. Rebaudengo, and M.S. Reorda. Comparing topological, symbolic and GA-based ATPGs: an experimental approach. In *Int'l Test Conf.*, pages 39–47, 1996.

[7] F. Corno, P. Prinetto, M. Rebaudengo, and M.S. Reorda. GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits. *IEEE Trans. on CAD*, 15(8):991–1000, 1996.

[8] F. Corno, P. Prinetto, M. Rebaudengo, M.S. Reorda, and R. Mosca. Advanced techniques for ga-based sequential atpg. In *European Design & Test Conf.*, pages 375–379, 1996.

[9] L. Davis. *Handbook of Genetic Algorithms.* van Nostrand Reinhold, New York, 1991.

[10] R. Drechsler. Evolutionary algorithms for computer aided design of integrated circuits. In *Int'l Symposium on IC Technologies, Systems and Applications*, pages 302–311, 1997.

[11] R.D. Eldred. Test routines based on symbolic logical statements. *Journal of the ACM*, 6(1):33–36, 1959.

[12] J.H. Holland. *Adaption in Natural and Artifical Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[13] M.S. Hsiao, E.M. Rudnick, and J.H. Patel. Alternating strategies for sequential circuit atpg. In *European Design & Test Conf.*, pages 368–374, 1996.

[14] M.S. Hsiao, E.M. Rudnick, and J.H. Patel. Sequential circuit test generation using dynamic state traversal. In *European Design & Test Conf.*, pages 22–28, 1997.

[15] M. Keim, B. Becker, and B. Stenner. On the (non-) resetability of synchronous sequential circuits. In *VLSI Test Symp.*, pages 240–245, 1996.

[16] M. Keim, N. Göckel, R. Drechsler, and B. Becker. Test generation for (sequential) multi-valued logic networks based on genetic algorithm. In *Int'l Symp. on multi-valued Logic*, 1998.

[17] R. Krieger. PLATO: A tool for computation of exact signal probabilities. In *VLSI Design Conf.*, pages 65–68, 1993.

[18] R. Krieger, B. Becker, and M. Keim. A hybrid fault simulator for synchronous sequential circuits. In *Int'l Test Conf.*, pages 614–623, 1994.

[19] R. Krieger, B. Becker, and M. Keim. Symbolic fault simulation for sequential circuits and the multiple observation time test strategy. In *Design Automation Conf.*, pages 339–344, 1995.

[20] X. Lin, I. Pomeranz, and S.M. Reddy. Mix: A test generation system for synchronous sequential circuits. In *VLSI Design*, pages 456–463, 1998.

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

[22] T.M. Niermann and J.H. Patel. HITEC: A test generation package for sequential circuits. In *European Conf. on Design Automation*, pages 214–218, 1991.

[23] I. Pomeranz and S.M. Reddy. Test generation for synchronous sequential circuits using multiple observation times. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 52–59, 1991.

[24] I. Pomeranz and S.M. Reddy. The multiple observation time test strategy. *IEEE Trans. on Comp.*, pages 627–637, May 1992.

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

[26] I. Pomeranz and S.M. Reddy. On static compaction of test sequences for synchronous sequential circuits. In *Design Automation Conf.*, pages 215–220, 1996.

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

[28] E. Rudnick and J.H. Patel. Combining deterministic and genetic approaches for sequential circuit test generation. In *Design Automation Conf.*, pages 183–188, 1995.

[29] E.M. Rudnick, J.H. Patel, G.S. Greenstein, and T.M. Niermann. Sequential circuit test pattern generation in a genetic algorithm framework. In *Design Automation Conf.*, pages 698–704, 1994.

[30] D.G. Saab, Y.G. Saab, and J.A. Abraham. Iterative [simulation-based genetics+deterministic techniques] =complete atpg. In *Int'l Conf. on CAD*, pages 40–43, 1994.

[31] D.S. Saab, Y.G. Saab, and J.A. Abraham. Cris: A test cultivation program for sequential vlsi circuits. In *Int'l Conf. on CAD*, pages 216–219, 1992.

[32] T. Stanion and D. Bhattacharya. TSUNAMI: a path oriented scheme for algebraic test generation. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 36–43, 1991.