SOCRATES: A Highly Efficient Automatic Test Pattern Generation System

MICHAEL H. SCHULZ, ERWIN TRISCHLER, AND THOMAS M. SARFERT

Abstract—**This paper presents Siemens' automatic test pattern generation system SOCRATES. SOCRATES includes several new concepts and techniques aiming at a significant improvement and acceleration of the automatic test pattern generation process for combinational and scan-based circuits. Based upon the sophisticated strategies of the FAN algorithm, an improved implication procedure, an improved unique sensitization procedure, and an improved multiple backtrace procedure are described. The application of the proposed techniques leads to a considerable reduction of the number of backtrackings and an earlier recognition of conflicts and redundancies. Several experiments using a set of combinational benchmark circuits demonstrate the efficiency of SOCRATES and its cost-effective applicability even in a workstation environment.**

I. Introduction

THE ADVANCES IN VLSI technology during the last decade have had a great impact on testing. Because of the increase in circuit size and the limited accessibility to the internal nodes of a circuit, the costs of testing a chip have become a substantial part of the overall chip costs. To keep chip testing costs within reasonable bounds, design for testability (DFT) methods and, in particular, the various scan-design techniques [1] have been developed. The most important advantage of those techniques is that they transform sequential circuits into combinational ones for testing purposes. Thus, the introduction of the scan-design techniques, as e.g., the LSSD approach, reduces the general logic testing problem to that of combinational circuits. Consequently, well established methods and algorithms for automatic test pattern generation (ATG) and fault simulation can be used, in order to facilitate the testing of VLSI circuits.

The problem of deterministic test pattern generation for combinational circuits has been discussed in a number of papers during the last few years. Based upon Roth's conceptual solution [2], a number of other approaches and improvements, e.g., PODEM [3], FAN [4], FAST [5], and ATWIG [6], have been proposed. Although considerable progress has been achieved, especially by Fujiwara's FAN algorithm [4], the computational resources required for ATG are still immense. As a matter of fact, new concepts and cost-effective methods are imperative, in order to cope efficiently with circuits with several tens of thousands of gates in a workstation environment.

Manuscript received July 11, 1987; revised August 25, 1987. The review of this paper was arranged by Associate Editor T. W. Williams.

M. H. Schulz is with the Institute of Computer Aided Design, Department of Electrical Engineering, Technical University of Munich, D-8000 Munich 2, West Germany.

E. Trischler and T. M. Sarfert are with Siemens AG, Department E STE 4 CAD/CAT, D-8000 Munich 70, West Germany.

IEEE Log Number 8717435.

In this paper, we present a deterministic test pattern generation approach including several new techniques which significantly advance the state of the art for combinational and scan-based circuits. Based upon the FAN algorithm, an improved implication procedure, an improved unique sensitization procedure, and an improved multiple backtrace procedure are described. In particular, our proposals aim at a reduction of the number of backtrackings, at an earlier recognition of conflicts, and at improvements of the heuristics which are applied at several stages of the deterministic ATG process. Moreover, they considerably support the identification of redundant faults. The techniques presented in this paper are implemented in Siemens' ATG system SOCRATES (Structure-Oriented Cost-Reducing Automatic TESt pattern generation system), which will be available as part of the SITEST® CAD/CAT system. SOCRATES also incorporates the highly efficient fault simulation approach proposed recently by Antreich and Schulz [7]–[9].

The following section gives an overview of the SOCRATES system. Section III briefly deals with the fault simulation algorithm which is part of SOCRATES. In Section IV we introduce an abstract representation of the problem of deterministic test pattern generation. Subsequently, we present the improved implication procedure, the improved unique sensitization procedure, and the improved multiple backtrace procedure in detail. Our motivation for extending SOCRATES to high-level primitives is discussed in Section V. Section VI summarizes the experimental benchmark results that have been achieved by SOCRATES. Finally, a number of illustrative comparisons demonstrates the advantages that can be gained from the application of the techniques proposed in this paper.

II. System Overview of SOCRATES

Fig. 1 shows the system overview of SOCRATES. The input to SOCRATES is a description of the circuit for which ATG has to be performed. Moreover, if ATG is desired for certain faults only, the user can specify them in a target faults list. The output of SOCRATES comprises the generated test patterns, three lists containing the undetected, the redundant, and the aborted faults, a summary file providing all interesting information concerning the ATG process, e.g., CPU time, fault cover-

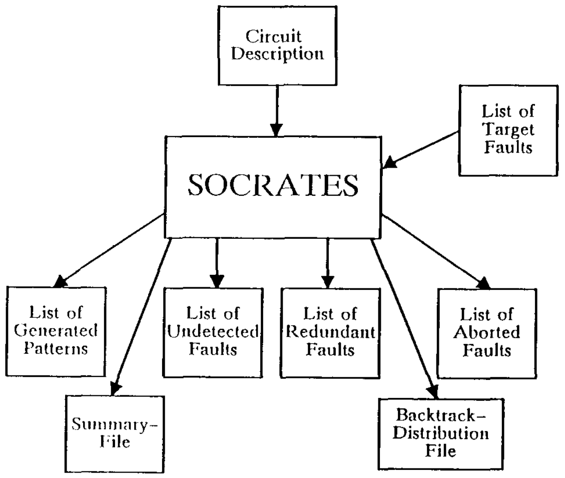


Fig. 1. SOCRATES system overview.

age, etc., and a distribution of the backtrackings having occurred.

The efficiency of SOCRATES is a result of combining several distinct techniques, which are briefly outlined in the following and illustrated by the system flowchart of SOCRATES (Fig. 2).

- After reading the circuit description and constructing the target faults list, SOCRATES executes at first a preprocessing phase, which includes a *learning procedure* and an *analysis of the circuit structure*.
- Based upon the concepts described in [10] and [11], SOCRATES assesses the random pattern testability of the circuit under consideration with the aid of the COP testability measures. In particular, it delivers projections of the numbers of random patterns, which are necessary to achieve 80-percent, 90-percent, 95-percent, and 98-percent fault coverage, respectively. The purpose of this *random testability assessment* is to assist the user in deciding whether or not he wants to apply random test pattern generation (RTG).
- As is common in most ATG systems, SOCRATES presently performs ATG in *two phases*:

  Phase 1: Random Test Pattern Generation;

  Phase 2: Deterministic Test Pattern Generation.

  Phase 1 can be skipped for special applications, for example, if the preceding random testability assessment indicated the circuit to be extremely random pattern resistant or if ATG has to be performed for faults specified in a target faults list.
- A *highly efficient fault simulation* approach, which has been described in [7]–[9] and benchmarked in [12] and [13], is used for both grading the generated random patterns as well as in the typical iterative loop, in which a test pattern is generated by the deterministic ATG algorithm, followed by the determination of all detected faults.
- An *improved deterministic test pattern generation* approach, which is based upon the FAN algorithm [4], has been developed. The improvements result from several new concepts, which will be described in detail in Section IV of this paper.

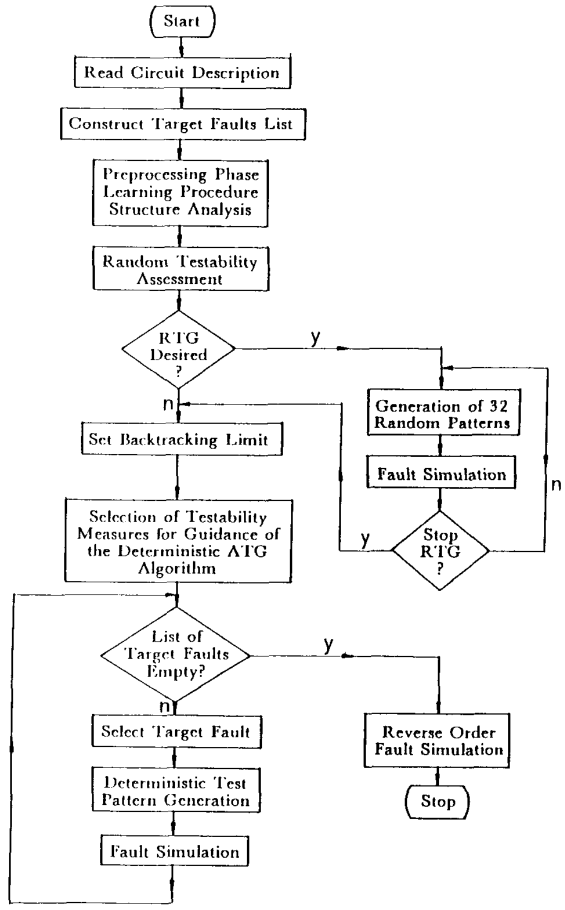

Start
Read Circuit Description
Construct Target Faults List
Preprocessing Phase Learning Procedure Structure Analysis
Random Testability Assessment
RTG Desired ?
y
n
Generation of 32 Random Patterns
Fault Simulation
Stop RTG ?
y
n
Set Backtracking Limit
Selection of Testability Measures for Guidance of the Deterministic ATG Algorithm
List of Target Faults Empty?
y
n
Select Target Fault
Deterministic Test Pattern Generation
Fault Simulation
Reverse Order Fault Simulation
Stop

Fig. 2. SOCRATES system flowchart.

- *Various testability measures*, e.g., the approach described in [5], COP [10], LEVEL [14], and SCOAP [15], can be used for guiding and supporting the decision making process in the presented deterministic ATG algorithm.
- In addition to simple gates (AND, OR, etc.), SOCRATES is also capable of handling XOR and XNOR gates as well as *high-level primitives*, e.g., adders, multiplexers, encoders, etc.
- In order to achieve a *compaction of the generated test set*, a fault simulation in reverse order is carried out at the final stage of the ATG process.

III. The Fault Simulation Algorithm

The fault simulation algorithm is based upon the concepts described in [7]–[9], [12], and [13]. Its basic strategy is to apply parallel processing of patterns at all stages of the fault simulation procedure and to avoid all unnecessary operations. Therefore, explicit fault simulations are performed for fanout stems only [16]. Moreover, a significant acceleration is achieved by targeting the stem fault simulation very carefully and by taking advantage of structural characteristics of the circuit, e.g., the dominance relationships between the fanout stems [8], [9].

The combination of these concepts results in a highly efficient fault simulation approach. It has been benchmarked using random patterns for both single stuck-at as well as single transition faults in [12] and in [13], respectively. Moreover, the fault simulation cost per pattern has been shown to decrease rapidly when a sufficiently high fault coverage has been achieved [12].

The reasons for choosing RTG as test generation strategy in phase 1 become obvious from the discussion above and can be summarized as follows:

- RTG is *straightforward*.
- Since the fault simulation algorithm is capable of grading as many patterns as there are bits in a machine word in *parallel*, the random patterns can be generated in *portions of 32 patterns* on an APOLLO DN 3000 workstation.
- The number of new faults detected per test is *initially large*. Thus, after a few trials, the fault coverage achieved is sufficiently high to cause a significant *decrease* in the *fault simulation cost per pattern*.
- The deficiencies of RTG are that the number of new faults detected per pattern decreases and that the fault simulation effort for the evaluation of those random patterns, which do not detect any additional fault, is wasted. However, those deficiencies of RTG *are compensated* by the efficiency of the fault simulation algorithm. As a consequence of the facts that patterns are simulated in parallel and that the fault simulation cost per pattern decreases rapidly, the fault simulation effort wasted for the evaluation of those random patterns that do not cover any additional fault becomes very small.

IV. The Deterministic Test Pattern Generation Algorithm

A. Problem Representation

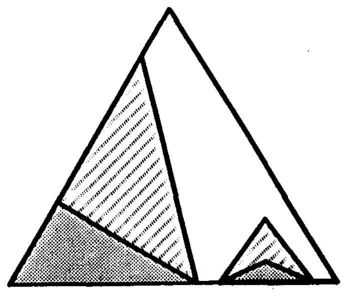
The problem of deterministically generating a test pattern for a given fault (target fault) is to find a combination of assignments of logic values (0 or 1) to the primary inputs which

simulate the target fault and

monitor the target fault at at least one of the primary outputs.

According to [3], the problem of ATG, which is known to belong to the class of NP-complete problems, can be viewed as a finite space search problem. For a circuit with N primary inputs, there exist 2^N combinations of input assignments. These 2^N combinations represent all points the finite *search space* consists of. Since, in general, only a small portion of those 2^N combinations of input assignments fulfill the above requirements, i.e., are test patterns for the target fault, deterministic ATG can be viewed as the search problem of finding a point in the search space that corresponds to a test pattern and, consequently, to a solution of the search problem.

Since the properties of deterministic ATG fulfill the requirements for a *systematic search* and the *split-and-prune* method [17], ATG algorithms usually build a *decision tree* and apply a *backtracking search* procedure [3], [4], [7], in order to find a solution of the problem. The decision tree can be described by a triangle (Fig. 3), which is usually divided up into one *solution area* and into several *non-solution areas*. The latter ones are shaded in Fig. 3. It is of particular importance to note that, whenever one of the nonsolution areas is entered during the search process, no solution can be found by additional assignments of logic values to the primary inputs of the circuit. In this case, the nonsolution area can only be left by backtracking, i.e., by rejecting assignments already made and choosing a different alternative which has not been examined during the search process up to now [3], [17]. Moreover, the decision tree can be interpreted as a projection of the total search space into the plane.



☐	solution area of the decision tree
▨	unidentified non – solution areas
▩	identified non – solution areas

Fig. 3. Abstract problem representation of deterministic ATG.

The main problem deterministic ATG algorithms suffer from is that, in general, they are not able to identify the entire nonsolution areas, but only parts of them. Whenever a deterministic ATG algorithm gets into one of the areas (Fig. 3), i.e., an *identified non-solution area*, it will recognize that no solution can be reached from the current point and immediately resort to backtracking. Contrary to the identified nonsolution areas, the *unidentified non-solution areas* (lightly shaded in Fig. 3) cause the most serious problems to deterministic ATG algorithms. In this case, a lot of backtracking and CPU time are wasted in order to leave the non-solution area. As a consequence, improvements of deterministic ATG for both combinational as well as sequential circuits have to aim at two global goals:

(1) minimizing the unidentified non-solution areas and
(2) avoiding all non-solution areas during the search process.

Those global goals can be achieved by special techniques that

- optimize the pruning of the search space [17],
- reduce the number of backtrackings,
- assist in recognizing conflicts as early as possible, and especially
- avoid useless backtracking in unidentified nonsolution areas.

B. Background and Basic Strategies

The deterministic test pattern generation approach is based upon the FAN algorithm [4]. Like FAN, our deterministic ATG algorithm uses the five-valued logic containing the values 0, 1, X, D, and $\overline{D}$ [2]. Moreover, it is a complete algorithm in the sense that, given enough time, it will generate a test pattern for each testable, i.e., nonredundant, fault. In the following we assume that the reader is familiar with definitions and terminologies, e.g., decision tree, D-frontier, implication, backtrace, line justification, etc. (for definitions, see previous work [2]–[4]).

The FAN algorithm has achieved considerable progress in the area of ATG for combinational circuits by several distinct techniques aiming successfully at the global goals (1) and (2) elaborated on in the preceding subsection. The most important strategies of the FAN algorithm to achieve those goals are [4]:

- In each step of the algorithm, the logic values of as many signals as possible are immediately determined by a powerful *implication procedure*.
- When the D-frontier consists of a single signal, a *unique sensitization* procedure is invoked.
- In order to reduce the computational cost required for backtracing objectives through the circuit, a *multiple backtrace* procedure using very sophisticated heuristics is employed instead of a single backtrace procedure [3].
- The backtrace is stopped at *head lines* [4]. Since, by definition, the line justification of the head lines can always be done without any backtracking, it is *postponed to a final stage* of the deterministic test pattern generation process.

The techniques applied in FAN, in order to improve and accelerate deterministic ATG, can be categorized into

- deterministic and
- heuristic.

While the multiple backtrace procedure is heuristic in nature, all other techniques listed above belong to the first category. The difference between those two categories is that the deterministic techniques *definitely* lead to improvements of ATG in terms of backtrackings and recognition of conflicts. From the heuristic techniques, in general, the ATG process benefits only *on average*. In particular, situations may exist in which the applied heuristics do not result in optimal performance, which might have been achieved by the application of different heuristics. Consequently, deterministic techniques to improve and accelerate the ATG process are more valuable and more desirable than heuristic ones. For this reason, we have developed an improved implication procedure (Section IV-C) and an improved unique sensitization procedure (Section IV-D). On the other hand, in spite of their deficiencies, good heuristics are indispensable for performing ATG efficiently. Therefore, we are going to elaborate on an improved multiple-backtrace procedure as well (Section IV-E).

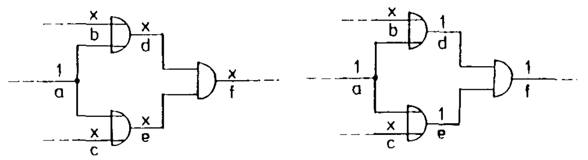


Fig. 4. Implications from $a = 1$.

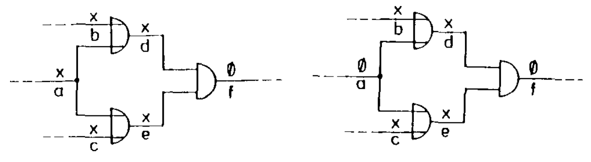


Fig. 5. Implications from $f = 0$.

C. The Improved Implication Procedure

Fig. 4 shows a small part of a larger circuit, in which the logic value 1 has been assigned to signal a. Let us assume that the injected target fault is not located in this part of the circuit and that the effects of the target fault cannot propagate to it as well. Then, we are sure that the logic values in this part of the circuit are all 0, 1, or X, and not D or $\overline{D}$. Clearly, the implication procedure used in FAN [4] would immediately imply the assignments $d = 1$, $e = 1$, and $f = 1$.

Fig. 5 shows the same part of the circuit as Fig. 4. However, the situation in Fig. 5 is different from that in Fig. 4. Now, the logic value 0 has been assigned to signal f. Since f is the output of an AND gate and the logic values of both gate inputs d and e are X (don't care), no implications can be performed by the implication procedure used in FAN. On the other hand, there exists a *logical identity* called the *contrapositive* [18]:

$$(P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P). \tag{1}$$

In (1) P and Q are assertion variables of arbitrary assertions. If P and Q are replaced by the assertions "the logic value of signal a is 1" and "the logic value of signal f is 1", respectively, then the left side of (1) becomes

$$(a = 1) \Rightarrow (f = 1) \tag{2}$$

which is obviously a valid implication, as shown in Fig. 4. Making use of identity (1), we can deduce from (2)

$$\neg (f = 1) \Rightarrow \neg (a = 1). \tag{3}$$

Executing the negations in (3), we get

$$(f = 0) \Rightarrow (a = 0). \tag{4}$$

Considering Fig. 5, it becomes obvious that the logic value 0 at signal f can be justified if, and only if, the logic value 0 is assigned to signal a as well. Therefore, it would be desirable to find a way to immediately imply the assignment $a = 0$ if the logic value 0 has been assigned to signal f. The immediate execution of those additional implications during the deterministic ATG process would result in an improved pruning of the search space [17] and thus in a reduction of the number of backtrackings and an earlier recognition of conflicts and redundancies.

In order to take advantage of those additional implications during the deterministic ATG process, a *learning procedure* is performed during the preprocessing phase.

```
learn ()
{
   for every signal i
   {
      assign_value ( i,0 )
      implication ()
      analyze_result ( i )
      assign_value ( i,1 )
      implication ()
      analyze_result ( i )
   }
}
```

Fig. 6. Learning procedure.

```
analyze_result ( i )
{
  v_i = value of signal i
  for every signal j with value of j ≠ X and
                  with j ≠ i
  {
   v_j = value of signal j
   g  = gate with output signal j
   t_g = type of gate g
   if ( v_j = 1 and ( t_g = AND or t_g = NOR  ) or
        v_j = 0 and ( t_g = OR   or t_g = NAND ) or
        t_g = XOR or t_g = XNOR ) then
   {
    if ( check_path ( j,i ) = 0 ) then
    {
     save_implication ( j = v̄_j ⇒ i = v̄_i )
    }
   }
  }
}
```

Fig. 7. Outline of the procedure *analyze_result*.

The learning procedure for simple gates as well as for XOR and XNOR gates is outlined in Figs. 6 and 7.

The overall strategy of the learning procedure is to assign a logic value to a certain signal of the circuit, to perform all implications from that assignment, and to learn from the results of those implications. This is done for all signals of the circuit for both logic values 0 and 1.

Among all implications from the assignment by which the current learning step has been initialized, we have to pick out those which are worthwhile. Obviously, it does not make sense to explicitly learn those implications that can be performed by the conventional implication procedure [4] as well, since it would result in a wastage of storage resources. Therefore, for the small example shown in Figs. 4 and 5, we do not learn the implications

$$(d = 0) \Rightarrow (a = 0) \tag{5a}$$

and

$$(e = 0) \Rightarrow (a = 0) \tag{5b}$$

since d and e are the output signals of OR gates, and the implications procedure used in FAN would immediately execute these *local* implications. On the other hand, as elaborated above, the conventional implication procedure does not recognize the implication

$$(f = 0) \Rightarrow (a = 0) \tag{5c}$$

which is consequently worthwhile learning. Moreover, (5c) can be viewed as a *global* implication over reconvergent fan-out. Unfortunately, it would require prohibitive CPU times and would not be feasible for VLSI circuits to check in the described manner which implications the conventional implication procedure is able to perform. In order to decide efficiently whether or not an implication is worthwhile learning, we have developed a *learning criterion*.

• *Learning Criterion:*

Let i be the signal at which the current learning step has been initialized by the assignment $i = v_i$, and let j be a signal to which the fixed logic value v_j(0 or 1) has been assigned during the implication procedure, i.e.,

$$(i = v_i) \Rightarrow (j = v_j). \tag{6}$$

Furthermore, let j be the output signal of gate g. If

(1) v_j requires all inputs of g to have non-controlling values and
(2) a forward implication [4] has contributed to the assignment $j = v_j$, then the implication

$$(j = \bar{v}_j) \Rightarrow (i = \bar{v}_i) \tag{7}$$

is considered to be worthwhile learning.

The learning criterion is implemented by the procedure *analyze_result*. Condition (1) is fulfilled if v_j is 1 and g is an AND or a NOR gate, if v_j is 0 and g is an OR or a NAND gate, or if g is an XOR or a XNOR gate. If the result of the procedure *check_path* indicates that no directed path from signal j to signal i exists, we are sure that a forward implication has contributed to the assignment $j = v_j$ and consequently, condition (2) of the learning criterion is satisfied.

It should be mentioned that the learning criterion represents a sufficient, but not a necessary, condition to establish that an implication cannot be performed by the implication procedure used in FAN. Moreover, a forward implication may have contributed to the assignment $j = v_j$, even if a structural path from j to i exists. Consequently, one can find examples for which *analyze_result* will not identify all those implications which the conventional implication procedure is unable to recognize. On the other hand, from practical experience it can be confirmed that, for the most circuits, the presented learning criterion and its implementation by *analyze_result* provide a very efficient way of learning all those implications.

Considering high-level elements, the implementation of the learning criterion is much more costly. On the other hand, it can assist in facilitating the handling of complex primitives, e.g., buses. Finally, the presented concept of learning implications during a preprocessing phase can also be successfully applied in order to improve and accelerate the ATG process for sequential circuits.

D. The Improved Unique Sensitization Procedure

In [4], a unique sensitization procedure has been introduced with the goal of handling those situations in an optimal manner in which

- the D-frontier consists of a single signal and
- all paths from the D-frontier to the primary outputs of the circuit pass through distinct gates.

In order to formulate the instructions for the application of the unique sensitization procedure exactly, we introduce the notion of dominance between signals of a combinational circuit.

Definition 1: A signal y is said to *dominate* signal x, i.e. $y \in \mathrm{dom}(x)$, if all directed paths from x to the primary outputs of the circuit pass through y. Thereby, $\mathrm{dom}(x)$ denotes the set of signals that dominate signal x. The elements of $\mathrm{dom}(x)$ are called dominator signals or, for brevity, dominators of x.

With the concept of dominance, instruction 1 for the application of the unique sensitization procedure can be formulated as follows.

- *Instruction 1 for the Application of the Unique Sensitization Procedure:* Let x be the only signal the D-frontier consists of. Furthermore, let the elements of $\mathrm{dom}(x) = \{y_1, y_2, \cdots, y_n\}$ be the output signals of their corresponding gates, combined in the set $G = \{g_1, g_2, \cdots, g_n\}$. Then, for all gates $g \in G$, the *noncontrolling value* (determined by the type of g) is assigned to all those inputs of g that *cannot be reached* from x on a path of signals.

For this application, it is sufficient to determine the *immediate dominator* of every signal.

Definition 2: A signal y is said to be the immediate dominator of signal x, if $y \in \mathrm{dom}(x)$ and y is that element of $\mathrm{dom}(x)$ with the lowest circuit level.

Obviously, if the immediate dominators of all signals are known, *all* dominators of signal x can be determined *recursively*.

While it is trivial to identify the immediate dominators of those signals that have only one successor signal, the task of determining the immediate dominators of signals with more than one successor signal is more difficult. Note that, in the case of high-level elements, even signals which are no fan-out stems may have more than one successor signal if they affect more than one output signal of the high-level primitive they feed into. The identification of the immediate dominators is performed during the preprocessing phase. Note that it is advantageous to start the identification of the immediate dominators with signals close to the primary outputs in terms of circuit levels, and to proceed in decreasing order of circuit levels towards the primary inputs. In this case, the determination of the immediate dominators of signals at lower circuit levels can capitalize on the information already gained from the identification of the immediate dominators of signals at higher circuit levels.

It should be mentioned that the dominance relationships are exploited by both the fault simulation approach [8],

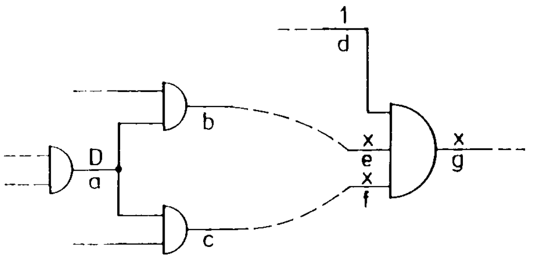


Fig. 8. Use of dominance relationships in the unique sensitization procedure.

[9] as well as the unique sensitization procedure of the deterministic ATG algorithm. Fig. 8 shows an example for the application of the unique sensitization procedure. For simplicity, the gates are named in accordance with their output signals. In Fig. 8, the D-frontier consists of signal a only. Since fanout stem a branches out to the gates b and c, none of the signals b and c is a dominator of a and, consequently, instruction 1 cannot be applied to the gates b and c. However, all paths from fanout stem a to the primary outputs of the circuit pass through signal g. That means that, according to Definition 1, signal g is a dominator signal of fanout stem a. In order to apply instruction 1 correctly, we have to distinguish between those gate inputs of g which can be reached from fanout stem a and those which cannot be reached from a.

In our example, the logic value 1 is assigned to signal d, because d cannot be reached from fan-out stem a and g is an AND gate. No logic values are assigned to the signals e and f, because both can be reached from signal a, and the value D or $\overline{\mathrm{D}}$ may be assigned to one or both of them during the further ATG process. Moreover, it is worth mentioning that, in the case of XOR and XNOR gates as well as high-level primitives, sometimes no fixed values can be assigned, not even to those gate inputs that cannot be reached from the signal representing the D-frontier. The reason for this is that either the non-controlling value or the combination of the non-controlling values is not unique for the corresponding primitive. For example, in the case of an XOR gate, both values 0 and 1 will sensitize a path through the gate.

Fig. 9 shows a different situation, which is not suitable for the application of instruction 1. Moreover, this type of situation is not considered by the unique sensitization procedure used in FAN [4]. In Fig. 9, assume that the D frontier originally consisted of signal a only. Obviously, it is necessary to assign the logic value 1 to signal b in order to sensitize a path through at least one of the gates d, e, or f. The reason for this is that signal b branches out to all those gates to which signal a branches out, and all gates (d, e, and f) require the same non-controlling value 1. On the other hand, signal c feeds into gate d only. Therefore, a fixed logic value must not be assigned to signal c in the unique sensitization procedure.

The *improved unique sensitization procedure* takes situations like that in Fig. 9 into account. They are also identified by an analysis of the circuit structure during the

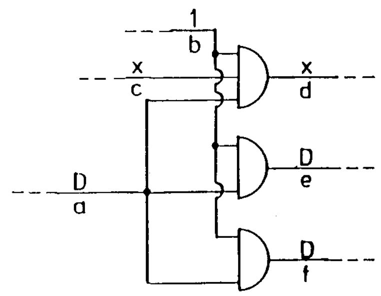


Fig. 9. Example for the improved unique sensitization procedure.

preprocessing phase. Thus, in addition to instruction 1, an additional instruction for the application of the improved unique sensitization procedure can be formulated.

• *Instruction 2 for the Application of the Improved Unique Sensitization Procedure:* Let x be the only signal the D-frontier consists of or a dominator of the only signal the D-frontier consists of. Furthermore, let signal x branch out to the gates $g_1, g_2, \cdots, g_n$, and let *all* of them require the *same noncontrolling value* 0 or 1. Then, the noncontrolling value is assigned to all those signals $y \neq x$ which branch out to *all* gates $g_1, g_2, \cdots, g_n$, too.

As the improved implication procedure, the improved unique sensitization procedure results in an optimized pruning of the search space, and assists in reducing the number of backtrackings and in detecting conflicts as well as redundancies earlier. Moreover, both procedures belong to the category of deterministic improvements and acceleration techniques for ATG. In particular, the application of the improved unique sensitization procedure significantly increases the performance of SOCRATES for PLA-structured circuits, where situations to which instruction 2 can be applied occur extremely often.

E. *The Improved Multiple-Backtrace Procedure*

We have extended the *multiple-backtrace* procedure proposed in [4] in order to allow handling of XOR and XNOR gates as well as high-level primitives. In our multiple backtrace procedure an *objective* is defined in accordance with [4] as an ordered triplet:

$$(s, n_0(s), n_1(s)). \tag{8}$$

In (8), s denotes the number of the signal at which the objective occurs, and $n_0(s)$ and $n_1(s)$ indicate the number of times the logic values 0 and 1 are required at signal s, respectively [4]. Based upon this definition, Fig. 10 shows the multiple backtrace procedure for a XOR gate with two inputs x_1 and x_2 and the output signal y. Moreover, it should be mentioned that SOCRATES is also capable of handling XOR and XNOR gates with more than two inputs.

In order to support and guide the multiple-backtrace procedure, we have implemented the following testability analysis methods:

• COP [10],
• LEVEL [14],
• SCOAP [15], and

```
multi_btr_xor2 ( y,n_0(y),n_1(y) )
{
    c_00 = cost ( x_1 = 0 , x_2 = 0 )
    c_01 = cost ( x_1 = 0 , x_2 = 1 )
    c_10 = cost ( x_1 = 1 , x_2 = 0 )
    c_11 = cost ( x_1 = 1 , x_2 = 1 )
    if ( c_00 < c_11 ) then
    {
        n_0(x_1) = n_0(y)
        n_0(x_2) = n_0(y)
    }
    else
    {
        n_1(x_1) = n_0(y)
        n_1(x_2) = n_0(y)
    }
    if ( c_01 < c_10 ) then
    {
        n_0(x_1) = n_0(x_1) + n_1(y)
        n_1(x_2) = n_1(x_2) + n_1(y)
    }
    else
    {
        n_1(x_1) = n_1(x_1) + n_1(y)
        n_0(x_2) = n_0(x_2) + n_1(y)
    }
}
```

Fig. 10. Multiple backtrace for two-input XOR gates.

• and the approach used in the LAMP2 system [5].

Each of these testability analysis approaches provides both controllability and observability measures. While the former ones indicate the relative difficulty of setting a signal to a logic value (0 or 1), the latter ones reflect on the relative difficulty of observing a signal at the primary outputs. Consequently, the controllability measures of either testability analysis approach are used for guiding the line justification process (as indicated by cost ($x_1 = 0, x_2 = 0$), etc., in Fig. 10), and the observability measures are employed for selecting that signal of the D frontier for propagation which is supposed to be the best one.

When implementing the multiple-backtrace procedure and selecting the *fanout objectives* as *final objectives* according to the flowcharts presented in [4], we have gained the experience that, for some circuits, too many fan-out objectives become final objectives. As a consequence, the decision tree contains an inadequately large number of nodes. This, in turn, can lead to a significant increase in the number of backtrackings.

For this reason, we have introduced an *initialization flag* which is set *true* whenever the *backtrace flag* [4] is set *true*. That means, the initialization flag is set true whenever the multiple backtrace starts from the set of *initial objectives*, which consists of the unjustified lines and that signal of the D-frontier which is assumed to be the best one for propagation. When the set of the *current objectives* becomes *empty for the first time*, we consider the set of the fan-out objectives and satisfy those that fulfill the conditions described in [4]. If no such fanout objectives

```
satisfy_headobj ( h,n_0(h),n_1(h) )
{
  if ( n_0(h) > n_1(h) or
      n_0(h) = n_1(h) and cost(h = 0) < cost(h = 1) ) then
  {
    assign_value ( h,0 )
  }
  else
  {
    assign_value ( h,1 )
  }
}
```

Fig. 11. Procedure *satisfy_headobj*.

exist, or if all of them have already been satisfied, the initialization flag is reset to *false*. Fanout objectives can only become final objectives as long as the initialization flag is true. After the initialization flag has been reset to false, all objectives containing all requirements are traced back in a *breadth-first manner* until the *head lines* are reached. At the head lines, the objectives are satisfied by the procedure *satisfy_headobj*, which is shown in Fig. 11. Furthermore, in our experience, the order in which the head objectives are satisfied is of substantial impact to the performance of the deterministic ATG algorithm in terms of backtrackings and in terms of CPU time. In our current implementation, we satisfy those head objectives first, for which the sum of the 0 and 1-requirement is largest.

The presented strategy of tracing objectives through the circuit and satisfying them at fanout or head lines results in a reduction of the number of nodes in the decision tree of the deterministic ATG process and in a reduction of the number of backtrackings in many cases.

V. High-Level Primitives

Most ATG systems support only standard gate types [3]-[5], others also high-level primitives [6]. In addition to simple gate types (AND, NAND, OR, NOR, BUF, INV), SOCRATES is capable of handling XOR and XNOR gates with an arbitrary number of inputs as well as high-level primitives, e.g., adders, multiplexers, demultiplexers, encoders, decoders, etc. Consequently, SOCRATES can deal successfully with circuits for which no gate-level description is available, and take advantage of the concept of hierarchy, In addition, when comparing ATG, which uses high-level primitives (denoted in the following by ATG*), with ATG at the gate level, a number of advantages of ATG* can be observed:

- Since most testability analysis programs (e.g., [10]) assume a circuit to be free of reconvergent fanout, the calculation of the testability measures is only an estimate. In general, testability analysis using high-level primitives delivers *more exact testability measures* than testability analysis performed at the gate level. As a consequence, ATG* will presumably produce *fewer conflicts* than ATG.
- The *implication procedure* of ATG* is *more powerful* as compared to ATG, since fixed values can be

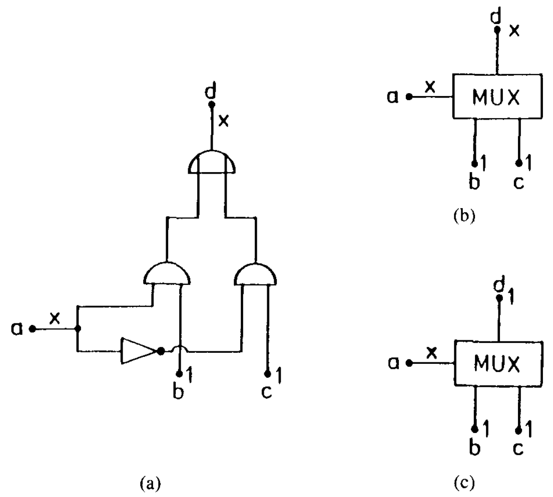


Fig. 12. Comparison of gate-level and high-level implication. (a) MUX at the gate level. (b) MUX before implication. (c) MUX after implication.

assigned to more signals than in ATG. Fig. 12 shows a multiplexer at different levels of description. The situation in Fig. 12(a) is identical to that in Fig. 12(b). In both cases, the logic value 1 has been assigned to the signals *b* and *c*. At the gate level (Fig. 12(a)), no implications can be performed, not even by the improved implication procedure described in Section IV-C of this paper. However, signal *d* will have the logic value 1 regardless whether the logic value 0 or 1 will be assigned to signal *a*, which is the select input of the multiplexer. Therefore, the logic value 1 can be immediately assigned to signal *d* at the high level, where the function of the primitive is known (Fig. 12(c)). From these observations it can be confirmed that ATG* prunes the search space in an improved manner. Consequently, ATG* will *solve a conflict faster* than ATG, and *fewer backtrackings* will occur in ATG* as compared to ATG.
- The *multiple backtrace procedure* of ATG* performs *more efficiently* as compared to ATG, since it can propagate an objective over a complex primitive by a single backtrace operation. On the contrary, many backtrace operations may be required at the gate level in order to propagate the objective over the part of the circuit corresponding to the complex primitive. In addition, at the high level the propagation of the objectives can capitalize on information about the function of the primitive and need not rely exclusively on the guidance by the testability measures.
- For the high-level circuit description, *less memory* is required to create the internal model of the circuit.

However, the use of high-level primitives involves not only advantages but also some disadvantages. The most important ones are:

- A fault within a primitive cannot always be detected if the internal structure of the primitive is not known.
- The implementation of ATG* is much more costly than the implementation of ATG.

To conclude the discussion above, in spite of the deficiencies we expect that the introduction of high-level primitives will accelerate the overall test generation process and further improve the performance of SOCRATES.

VI. Experimental Benchmark Results

SOCRATES has been implemented on an APOLLO DN 3000 workstation. We have applied SOCRATES to the ten well-known combinational benchmark circuits [19] and performed two experiments. In both experiments, the backtracking limit has been set to 10, and the COP [10] controllability measures in combination with the LEVEL [14] observability measures have been used for guiding the deterministic ATG algorithm. The only difference between the two experiments is that RTG has been employed in the first one, while phase 1 has been skipped in the second one. In experiment one, RTG has been stopped if 64 succeeding random patterns did not detect any additional fault. The results are summarized in Tables I and II, respectively. The fault coverage data refers to a simply collapsed fault set. The aborted faults column contains the number of those faults, for which the backtracking limit of 10 has been exceeded without successfully generating a test pattern and without proving them to be redundant. All CPU times have been measured on an APOLLO DN 3000 workstation. The CPU time required for the final reverse order fault simulation is included in the fault simulation time.

The results of Tables I and II demonstrate the efficiency of SOCRATES. Especially when comparing our results with other published results using the same benchmark circuits [5], [20], we can conclude that SOCRATES requires much less CPU time achieving the same or better fault coverage. For all benchmark circuits ranging in size between 160 and 3500 gates, fairly well compacted test sets have been generated in significantly less than 5 CPU minutes on an APOLLO DN 3000 workstation (Table I). Moreover, most circuits feature no aborted faults and a remarkably small number of backtrackings. Finally, as we would have expected, SOCRATES performs significantly better in terms of CPU time when RTG is applied during phase 1.

The reverse order fault simulation at the final stage of the ATG process requires between 17 percent and 48 percent of the total fault simulation time and results in compressing the generated test sets to about 65 percent of their original size in all cases considered.

Table III shows the number of implications that are learned by the learning procedure; the number of those situations in the benchmark circuits, to which instruction 2 of the improved unique sensitization procedure can be successfully applied; and the CPU time required to perform the entire preprocessing phase.

From Table III it becomes obvious that a surprisingly large number of implications is learned for all benchmark circuits. Moreover, although none of the benchmark circuits feature, PLA-structures, there exists a considerable number of those situations, which are suited for the application of instruction 2 of the improved unique sensitization procedure. The CPU times used for the execution of the entire preprocessing phase, which also includes the identification of the dominance relationships between the fan-out stems, appear to be small in comparison to the CPU times required for performing the total ATG process (Tables I and II).

TABLE I
Results of ATG by SOCRATES Applying RTG in Phase 1 (APOLLO DN 3000)

Circuit Name	Fault Coverage [%]	Test Patterns	Aborted Faults	Redundant Faults	Backtrackings	CPU-Time [sec.] ATG	CPU-Time [sec.] Fault Sim.	CPU-Time [sec.] Total
c432	99.24	58	3	1	31	2.2	1.5	3.7
c499	98.94	56	0	8	8	5.6	2.5	8.1
c880	100.00	60	0	0	0	1.6	4.1	5.7
c1355	99.49	88	0	8	15	12.9	9.0	21.9
c1908	99.52	125	0	9	6	15.1	18.0	33.1
c2670	95.49	127	11	106	218	46.9	22.4	69.3
c3540	96.00	171	0	137	102	22.0	40.0	62.0
c5315	98.90	143	0	59	63	12.3	23.0	35.3
c6288	99.56	38	0	34	18	1.3	113.5	114.8
c7552	98.25	231	56	77	694	197.7	86.6	284.3

TABLE II
Results of ATG by SOCRATES without Applying RTG in Phase 1 (APOLLO DN 3000)

Circuit Name	Fault Coverage [%]	Test Patterns	Aborted Faults	Backtrackings	CPU-Time [sec.] ATG	CPU-Time [sec.] Fault Sim.	CPU-Time [sec.] Total
c432	99.24	53	3	31	5.3	3.5	8.8
c499	98.94	58	0	38	24.9	5.7	30.6
c880	100.00	67	0	0	5.7	8.6	14.3
c1355	99.49	93	0	28	34.3	21.1	55.4
c1908	99.52	120	0	28	63.1	32.0	95.1
c2670	95.49	123	11	235	61.1	38.1	99.2
c3540	95.95	172	2	126	89.0	83.5	172.5
c5315	98.88	141	1	90	45.4	70.4	115.8
c6288	99.56	43	0	40	32.8	138.0	170.8
c7552	98.25	231	57	755	243.5	158.1	401.6

TABLE III
Number of Learned Implications; Number of Situations Suited for the Application of Instruction 2 of the Improved Unique Sensitization Procedure; CPU Time for Preprocessing Phase (APOLLO DN 3000)

Circuit Name	Number of Learned Implications	Number of Situations Suited for Instruction 2	CPU-Time [sec.]
c432	57	2	0.8
c499	2	0	1.1
c880	97	7	2.5
c1355	170	0	8.3
c1908	298	131	7.9
c2670	917	24	14.7
c3540	4107	158	44.4
c5315	1384	9	26.0
c6288	944	0	12.2
c7552	4095	52	61.5

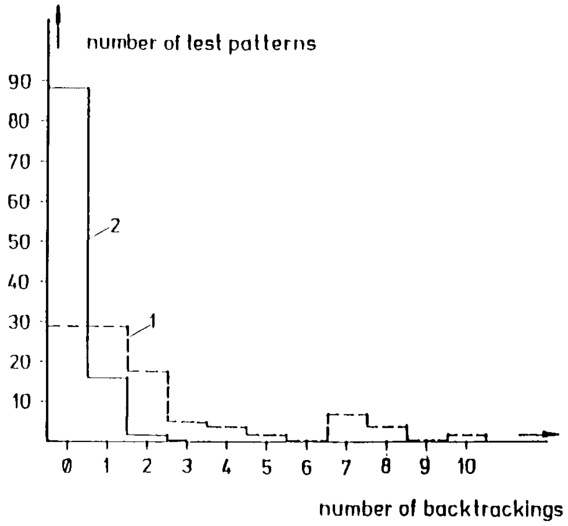


Fig. 13. Impact of the improved implication procedure.

Fig. 13 illustrates the effect of the improved implication procedure. For circuit c1355, ATG has been performed by two different implementations of SOCRATES without applying RTG during phase 1. The implementations differ in the implication procedure only. While implementation 1 uses the same implication procedure as described in [4], implementation 2 capitalizes on the improved implication procedure. Fig. 13 shows the backtracking distributions which have been obtained by the implementations 1 and 2, respectively, and demonstrates the impact of the improved implication procedure. Distribution 2 indicates that significantly more test patterns have been generated without any backtracking by exploiting the improved implication procedure. Moreover, the maximum number of backtrackings went down from 10 to 3, the total number from 201 to 28, and the average number (defined as the quotient of the total number of backtrackings and the number of generated test patterns) from 2.21 to 0.28.

To demonstrate the impact of the improved unique sensitization procedure, we have peformed two more experiments using an 8-bit ALU which is realized as a PLA with an input and an output decoder. The characteristics of the ALU are:

Number of signals:	173
Number of primary inputs:	19
Number of primary outputs:	11
Number of product terms:	53
Number of gates:	154
Number of stuck-at faults:	671
Thereof redundant:	22

In experiment I, a modified implementation of SOCRATES relying on the conventional unique sensitization procedure [4] has been applied to the ALU. Experiment II refers to the current version of SOCRATES, which capitalizes on the improved unique sensitization procedure as well as on all other techniques presented in this paper. RTG has been applied during phase 1 in both experiments. The results of those two experiments, which are summarized in Table IV, point out that the application of the improved unique sensitization procedure results in a significant reduction of both the number of backtrackings as well as the number of aborted faults.

In addition, the improved implication and unique sensitization procedures presented in this paper lead to identifying redundant faults much earlier as compared to a straightforward implementation of FAN. For example, SOCRATES proved two faults of circuit c1908 and eight faults of circuit c3540 to be redundant with less than 10 backtrackings, whereas an implementation relying on the conventional implication and unique sensitization procedures [4] requires by far more than 10 000 backtrackings in the most cases, and even more than 100 000 backtrackings in some cases. Moreover, SOCRATES identifies a considerable number of redundant faults in the benchmark circuits without any backtracking (e.g., circuit c3540).

Fig. 14 illustrates this effect with the aid of the abstract description of the decision tree introduced in Section IV of this paper. Since no test pattern exists for a redundant fault, no solution of the search problem can be found. Consequently, the total decision tree consists of a single nonsolution area. When relying on the conventional implication and unique sensitization procedures (Fig. 14(a)), a large part of it is not identified to be a non-solution area, and many backtrackings are required to prove the fault to be redundant. On the other hand, the unidentified nonsolution area is successfully minimized by the application of the improved implication and unique sensitization procedures (Fig. 14(b)), and the fault can be proved to be redundant with a significantly smaller number of backtrackings.

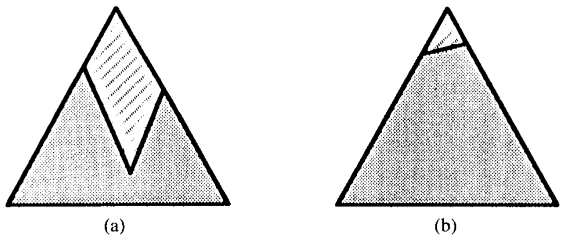


Fig. 14. Abstract problem representation of deterministic ATG for a redundant fault; impact of the improved implication and unique sensitization procedures. (a) Conventional procedures. (b) Improved procedures.

TABLE IV
IMPACT OF THE IMPROVED UNIQUE SENSITIZATION PROCEDURE

Ex-peri-ment	Fault Cover-age [%]	Test Pat-terns	Aborted Faults	Back-track-ings	CPU - Time [sec.]		
					ATG	Fault Sim.	Total
I	96.72	89	5	88	11.1	3.6	14.7
II	96.72	87	0	27	5.3	3.5	8.8

VII. Conclusions and Future Work

We have presented the highly efficient automatic test pattern generation system SOCRATES. Several techniques to significantly improve and accelerate the ATG process have been described in detail. The experimental benchmark results have pointed out the efficiency of SOCRATES and its cost-effective applicability even in a workstation environment. Moreover, the impact of the newly developed techniques presented in this paper has been illustrated by several experiments. Finally, it should explicitly be mentioned that most of the improvements are based upon the concepts of the FAN algorithm [4] and would have been impossible without this preceding work.

Our future work will concentrate on a number of investigations, which are briefly described as follows:

- Other low-cost test generation strategies, which can be used in the first phase of the overall ATG process, will be taken into account, in order to provide *alternatives to RTG*, especially for random pattern resistant circuits, as, e.g., c2670 and c7552.
- Since the CPU time required for performing deterministic ATG depends not only on the number of backtrackings, but also on the number of backtraces, it would be advantageous to *reduce the number of multiple-backtraces*. On the other hand, we have found that there is a *tradeoff* between the number of multiple backtraces and the performance of SOCRATES in terms of the number of backtrackings and aborted faults. Therefore, it would be favorable to minimize the number of times the multiple backtrace is started again at the unjustified lines and at the presumably best signal of the D frontier, while retaining the good performance of SOCRATES.
- In our experience, none of those testability measures that can be used in SOCRATES for guiding the deterministic ATG algorithm provides uniformly better results in terms of CPU time, aborted faults, fault coverage, etc., than all the others for all circuits considered. Therefore, it would be desirable to develop techniques capable of *automatically selecting* that kind of *testability measure* which promises best performance and the smallest number of conflicts. In addition, all testability measures implemented in SOCRATES up to now are *static* in nature. An interesting question appears to be whether benefits can be gained from the application of *dynamic testability measures*.
- We will try to further *improve the heuristics* which are used at several stages of the deterministic ATG algorithm, as, e.g.,

  strategy of the multiple-backtrace procedure and selection of head objectives to be satisfied.

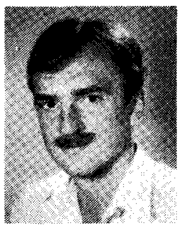
Acknowledgment

The authors are very grateful to Professor K. Antreich of Technical University of Munich for his valuable suggestions and his helpful advice and discussions, and to Dr. A. Sauer of Siemens's CAD/CAT project for his continuous support and encouragement. Moreover, M. Schulz wants explicitly to appreciate the cooperation of Prof. H. Fujiwara of Meiji University, Kawasaki, Japan.

References

[1] E. J. McCluskey, *Logic Design Principles.* Englewood Cliffs, NJ: Prentice-Hall, 1986.
[2] J. P. Roth, *Computer Logic, Testing, and Verification.* Potomac, MD: Computer Science Press, 1980.
[3.] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 215-222, Mar. 1981.
[4] H. Fuijiwara, and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Trans. Comput.*, vol. C-32, pp. 1137-1144, Dec. 1983.
[5] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test generation for VLSI scan-design circuits," *IEEE Design & Test*, pp. 43-54, Aug. 1986.
[6] E. Trischler, "ATWIG, An automatic test pattern generator with inherent guidance," in *Proc. 1984 Int. Test Conf.*, Oct. 1984, pp. 80-87.
[7] K. J. Antreich, and M. H. Schulz, "Fast fault simulation in combinational circuits," presented at *IEEE Int. Conf. Computer-Aided Design ICCAD-86*, Nov. 1986.
[8] K. J. Antreich, and M. H. Schulz, "Fast fault simulation for scan-based VLSI-logic, in *Proc. European Conf. Circuit Theory and Design 1987.*
[9] K. J. Antreich and M. H. Schulz, "Accelerated fault simulation and fault grading in combinational circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 704-712, Sept. 1987.
[10] F. Brglez, P. Pownall, and R. Hum, "Application of testability analysis: From ATPG to critical delay path tracing," in *Proc. 1984 Int. Test Conf.*, Oct. 1984, pp. 705-712.
[11] H.-J. Wunderlich, "PROTEST: A tool for probabilistic testability analysis," in *Proc. 22nd Design Automat. Conf.*, June 1985, pp. 204-211.
[12] M. H. Schulz, "Automatic test pattern generation and fault grading in combinational circuits," in *Proc. CompEuro 1987*, May 1987, pp. 382-385.
[13] M. H. Schulz and F. Brglez, "Accelerated transition fault simulation," in *Proc. 24th Design Automat. Conf.*, June 1987.
[14] A. Lioy and M. Mezzalama, "On parameters affecting ATPG performance," in *Proc. CompEuro 1987*, May 1987, pp. 394-397.
[15] L. H. Goldstein, and E. L. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program," in *Proc. 17th Design Automat. Conf.*, June 1980, pp. 190-196.
[16] S. J. Hong, "Fault simulation strategy for combinational logic networks," in *Proc. 8th Int. Symp. Fault-Tolerant Comput.*, 1978, pp. 96-99.
[17] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley, 1984.
[18] D. F. Stanat, and D. F. McAllister, *Discrete Mathematics in Computer Science.* Englewood Cliffs, NJ: Prentice-Hall, 1977.
[19] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," in *Proc. IEEE Int. Symp. Circuits Syst.*, June 1985.
[20] "Special Session on Recent Algorithms for Gate-Level ATPG with Fault Simulation and Their Performance Assessment," in *Proc. IEEE Int. Symp. Circuits Syst.*, June 1985, pp. 663-698.

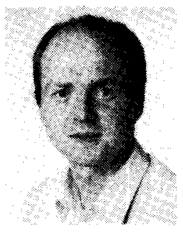
*



Michael H. Schulz was born in Hof, West Germany, on October 1, 1959. He received the Dipl.-Ing. degree from the Technical University of Munich, Munich, Germany, in 1984.

Since October 1984, he has been a research assistant at the Institute of Computer Aided Design, Department of Electrical Engineering, Technical University of Munich. His research interests are in the computer-aided design of electronic circuits and systems, with particular emphasis on automatic test pattern generation, fault simulation, fault grading, testability analysis, and design for testability. He is currently working on his Dr.-Ing. thesis on the topic of automatic test pattern generation and fast fault simulation for highly complex VLSI circuits. He has been a member of the ITG Testing Methods Group since 1986.

*



Erwin Trischler was born in Esseg, Yugoslavia, on September 20, 1943. He received the Dipl.-Ing. degree in electrical engineering in 1967 from the University of Zagreb, Zagreb, Yugoslavia.

He joined Siemens in Munich, West Germany, in 1969. Between 1969 and 1980 he was involved in research and development of design automation systems, especially in the areas of simulation, data base, test generation, and design for testability. Between 1980 and 1983 he was with Siemens Research and Technology Laboratories, Princeton, NJ, performing research in the areas design for testability, testability analysis, testability optimization, testability economics, and test generation. Since the end of 1984 he has been with Siemens' CAD/CAT project, responsible for research and development in the areas design for testability, test generation, and built-in self-test. He is currently also Siemens' representative at the Joint Test Action Group to promote boundary scan design for testability methodology.

*

Thomas M. Sarfert was born in Hamburg, Germany, on October 22, 1958. He received the Dipl.-Ing. degree in electrical engineering in 1986 from the University of Hannover, Hannover, West Germany.

Since 1986 he has been with Siemens's CAD/CAT project, where he is currently involved in the development of the SOCRATES test pattern generation system. His current interests are in the areas of logic synthesis, automatic test pattern generation, and redundancy checking.