VLSI Design
1994, Vol. 2, No. 3, pp. 199–207
Reprints available directly from the publisher
Photocopying permitted by license only

An Efficient Automatic Test Pattern Generator for Stuck-Open Faults in CMOS Combinational Circuits

HYUNG K. LEE and DONG S. HA
The Bradley Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061 USA

(Received November 15, 1990; Revised March 8, 1991)

In this paper, we describe a highly efficient automatic test pattern generator for stuck-open (SOP) faults, called SOPRANO, in CMOS combinational circuits. The key idea of SOPRANO is to convert a CMOS circuit into an equivalent gate level circuit and SOP faults into the equivalent stuck-at faults. Then SOPRANO derives test patterns for SOP faults using a gate level test pattern generator. Several techniques to reduce the test set size are introduced in SOPRANO. Experimental results performed on eight benchmark circuits show that SOPRANO achieves high SOP fault coverage and short processing time.

Key Words: *SOP fault, Stuck-at fault, Potentially equivalent fault, Primary fault, Fault simulation, Test pattern generation, ATPG*

I. INTRODUCTION

CMOS technology has become a dominant technology in VLSI circuits. However, the testing of CMOS circuits is complex and time consuming. A major difficulty in testing CMOS circuits stems from the inadequacy of the line stuck-at fault model. Transistor stuck-open (SOP) faults, in which faulty transistors are turned off permanently, are not modeled properly in the line stuck-at fault model [1]. A combinational circuit under the presence of SOP faults may behave as a sequential circuit. A sequence of two test patterns is required to detect a SOP fault [2–14].

Several researchers have proposed various methods of deriving test patterns for SOP faults [2–14]. These methods can be classified into three categories depending on their approaches:

1. switch level test pattern derivation based on a graph or a switch model of circuits [2–5];
2. gate level test pattern derivation based on an equivalent gate level model of circuits [6–11]; and
3. derivation of stuck-at test patterns based on an equivalent gate level model first and then organization of the sequence of stuck-at test patterns to cover SOP faults [12–14].

Since a switch level model represents the behavior of a CMOS circuit accurately, the first approach achieves higher fault coverage than the other two approaches (based on a gate level model). However switch level test derivation algorithms are complex and time consuming, hence they may not be practical for large circuits. The last approach is simple and effective, but limited to CMOS circuits consisting of only primitive gates. For these reasons, we employed the second approach in implementing an automatic test pattern generator for SOP faults. We call the system SOPRANO. In general, gate level test pattern derivation algorithms are relatively simple when compared to switch level test pattern derivation algorithms. Moreover, they can use well established test generation algorithms developed for line stuck-at faults such as PODEM [15] and FAN [16].

In this paper, we consider only single SOP faults. Detection of a SOP fault requires the application of two test patterns in sequence [2–14]. The first pattern, called T_1, is used for the initialization of the faulty gate output. The second pattern, called T_2,

*The research reported was supported in part by the National Science Foundation under grant no. MIP-8809164.

detects the SOP fault. As explained later, T_2 is, in fact, a test pattern detecting the stuck-at fault corresponding to the SOP fault. The key idea of SOPRANO is to convert a CMOS circuit under test into an equivalent gate level circuit and SOP faults into stuck-at faults. Then, SOPRANO derives a T_2 pattern of a SOP fault using an efficient gate level test pattern generation algorithm, FAN [16], and a gate level fault simulator, a parallel pattern single fault simulator. Once a T_2 pattern is obtained, a T_1 pattern is obtained using fault free responses of the circuit. After a (T_1, T_2) pair of each SOP fault is obtained, SOPRANO minimizes the overall test length through operlapping T_1 and T_2 patterns.

Since SOPRANO does not consider gate delays, some (T_1, T_2) pairs obtained by SOPRANO could be invalidated when gate delays are considered [10]. However, our experimental results show that only a small portion of (T_1, T_2) pairs is invalidated. Currently, SOPRANO is limited to fully complementary CMOS (FCMOS) gates.

The paper is organized as follows. In Section II, the modeling technique for CMOS circuits and SOP faults is described. In Section III, the procedure for deriving SOP fault test patterns is described. Section IV reports experimental results and observations. Finally, Section V summarizes this paper.

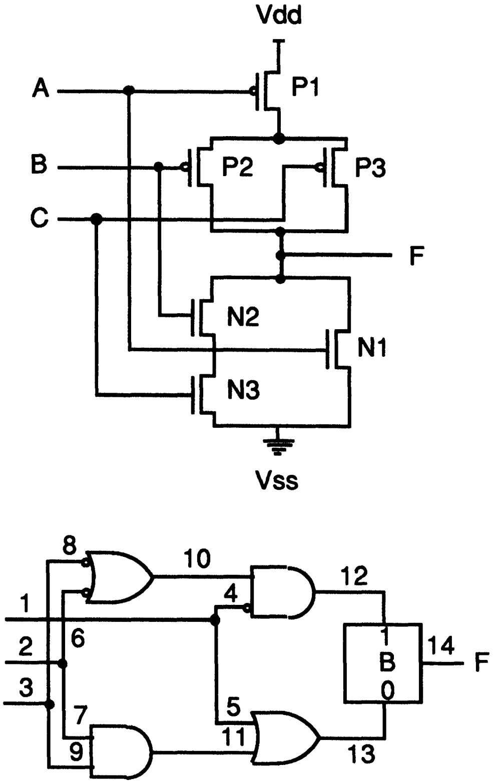

FIGURE 1 A FCMOS gate and Jain and Agrawal's gate model. a) A FCMOS gate; b) equivalent gate model.

II. CIRCUIT AND FAULT MODELING

In this Section, we describe the procedure to transform a CMOS gate into its equivalent gate level circuit. Our modeling is not confined to FCMOS gates with the dual structure, but, for clarity, we limit our discussion to dual FCMOS gates for the time being. Non-dual FCMOS gates are discussed later.

First, we briefly review the modeling procedure given by Jain and Agrawal [8]. In this procedure, a CMOS gate is described by three logic blocks, a memory element (called a B-block) and two gate level networks in which one represents the p-type transistor network and the other the n-type transistor network. Figure 1 shows a FCMOS gate and its equivalent gate level circuit obtained from the above method. A memory element is necessary to represent the memory state of the CMOS gate output under the presence of SOP faults. The introduction of a memory element in the above method makes the test pattern generation more complex.

Reddy, Agrawal and Jain further simplified the above model as described below [9]. When a T_2 test pattern of a SOP fault is applied to the faulty gate, the gate output floats and maintains the previous output of the gate. Hence, if the gate output is properly initialized by a T_1 pattern before the application of a T_2 pattern, the logic value of the gate output under the application of the T_2 pattern can be obtained. The knowledge of the logic value of the gate output enables us to eliminate the memory element. Since the functions of a p-type transistor network and an n-type transistor network of a FCMOS gate are dual, only one part, either a p-type or an n-type, transistor network is necessary to describe the function of the gate. In this paper, we employ this particular technique. The n-type network is used to describe an equivalent gate level circuit. An inverter is necessary due to the pull down operation of the n-type network. The n-type equivalent gate level circuit for the CMOS gate in Figure 1 is shown in Figure 2. For details of the modeling procedure, refer to [9].

Once an equivalent gate level circuit is constructed, the next problem is the representation of SOP faults in the equivalent circuit. We discuss this problem after giving a definition.

Definition [14]: A *potentially equivalent* fault of an n-type transistor SOP fault of a CMOS gate is the line stuck-at-0 (s-a-0) fault on the input line corre-

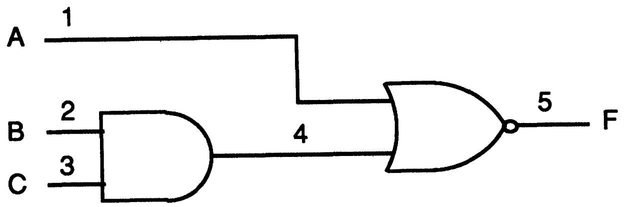

FIGURE 2 A gate level model by Reddy et al.

sponding to the faulty transistor. Similarly, a potentially equivalent fault of a p-type transistor SOP fault is the stuck-at-1 (s-a-1) fault on the corresponding input line.

In the above definition, we relate a SOP fault into a stuck-at fault using the terminology *potentially equivalent* to avoid confusion with the well known fault equivalence relation. Note that if an input of a CMOS gate is s-a-0 (s-a-1), it is equivalent to the n-type (p-type) transistor SOP fault connected to the input line assuming that the faulty gate is properly initialized. This implies that if a test pattern T_2 detects a potentially equivalent stuck-at fault, it also detects the corresponding SOP fault provided that an initialization pattern T_1 is applied prior to the application of the test pattern. If the structure of a CMOS gate is dual, there is always a corresponding p-type transistor for an n-type transistor. Hence, SOP faults on the p-type network are represented by s-a-1 faults in the n-type equivalent network. For example, potentially equivalent stuck-at faults of SOP faults in the CMOS gate shown in Figure 1 are given in Table I.

In the above, we explained a mapping procedure of a SOP fault into a stuck-at fault. Next, we discuss the reduction of the potentially equivalent stuck-at fault set size. Two stuck-at faults are equivalent if any test pattern detecting one also detects the other. Similarly, two SOP faults are equivalent if any (T_1, T_2) pair detecting one fault also detects the other. For two SOP faults in *a CMOS gate,* if T_2 patterns of the two gates are the same, then it can be easily seen that T_1 patterns are also the same. From the above discussion, we have the following property.

TABLE I
A Mapping of SOP Faults to Stuck-at Faults

SOP fault	Potentially Equivalent Stuck-At Fault
P1 SOP	line 1 s-a-1
P2 SOP	line 2 s-a-1
P3 SOP	line 3 s-a-1
N1 SOP	line 1 s-a-0
N2 SOP	line 2 s-a-0
N3 SOP	line 3 s-a-0

Property: Two SOP faults in a CMOS gate are *equivalent* if and only if the potentially equivalent stuck-at faults are equivalent.

For example, the line 2 s-a-0 fault and the line 3 s-a-0 fault in Figure 2 are equivalent. Hence, the N2 SOP fault and the N3 SOP fault of a CMOS gate in Figure 1 are equivalent. Note that the initializing values of two SOP faults are the same ($F = 1$ at the faulty gate output). Using the above property, the potentially equivalent fault set size can be reduced by eliminating all the equivalent stuck-at faults. We call the reduced potentially equivalent stuck-at fault set of a CMOS gate the *primary faults* of the gate. For example, a set of primary faults of the CMOS gate shown in Figure 1 are {line 1 s-a-1, line 2 s-a-1, line 3 s-a-1, line 1 s-a-0, line 2 s-a-0} in the equivalent gate model shown in Figure 2. Since every SOP fault is mapped into a corresponding stuck-at fault, a test set detecting all the primary faults of a CMOS circuit also detects all the SOP faults of the circuit, provided that the faulty gate outputs are properly initialized [14].

So far, we have confined our discussion to only dual FCMOS gates. If a CMOS gate is not dual in the structure, there are more than one corresponding p-type transistors for an n-type transistor or vice versa. If some p-type transistor SOP faults are not mapped into potentially equivalent stuck-at faults in an n-type equivalent circuit, both the p-type and n-type networks are used to represent the faults in SOPRANO. The use of both networks guarantees the representation of all the SOP faults using potentially equivalent stuck-at faults. However, only the n-type network is used for fault free simulations.

III. TEST PATTERN GENERATOR FOR SOP FAULTS

In this section, we describe the test pattern generation procedure employed in SOPRANO. Like other gate level test pattern generators [17], SOPRANO consists of three sessions: a random pattern testing (RPT) session, a deterministic test pattern generation (DTPG) session and a test compaction session. A parallel pattern single fault simulator (a parallel version of the single fault propagation method proposed by Harrel et al. [18]) is used for fault simulations during the RPT session and the test compaction session. A test pattern generator based on the

FAN algorithm [16] is used to derive test patterns of some primary faults in the DTPG session.

Suppose that a CMOS circuit under test and SOP faults are described by the equivalent gate level circuit and the primary faults as described in Section II. In the RPT session, a packet of 32 random patterns is applied to the circuit each time and fault simulated. If a test pattern tp detects a new primary fault f_i, which has not been detected by previous test patterns, the test pattern t_p is marked as a T_2 pattern of f_i. The fault f_i is marked as "T_2 FOUND." Then all the faults which can be initialized by the test pattern t_p are identified using fault free responses of the circuit. If the test pattern t_p is an initialization pattern of a primary fault f_j, t_p is marked as a T_1 pattern of f_j. The fault f_j is marked as "T_1 FOUND." A primary fault which is marked as both "T_1 FOUND" and "T_2 FOUND" is eliminated from the fault list. The RTP session terminates if either the fault list is empty or consecutive two packets of random patterns (i.e., 64 random patterns) do not detect any new fault. Test patterns of the remaining faults in the RPT session are derived in the DTPG session. Once a T_2 pattern of a primary fault f_i is derived by the FAN algorithm, the rest of the procedure is identical to the one described above. After (T_1, T_2) pairs are obtained for all the primary faults in the above two sessions, the pairs are concatenated to achieve maximum overlapping of T_1 patterns followed by T_2 patterns. The concatenated test sequence is fault simulated in both forward and backward order for further reduction of the length of the sequence, i.e., the number of test patterns.

The SOPRANO procedure is divided into five main steps as shown below. The first step transforms a CMOS circuit under test into its equivalent circuit and constructs a fault list containing all primary faults of the circuit. The second step is the RPT session and the third step is the DTPG session. In both steps, a T_1 pattern and a T_2 pattern of each primary fault are derived as explained above. If a fault is identified as redundant in step 3, the fault is marked as "REDUNDANT." Similarly, if the derivation of a test pattern is unsuccessful due to excessive backtrackings, the fault is marked as "ABORTED." In the fourth step, a procedure is applied to reduce the overall test sequence. Finally, the test sequence obtained in Step 4 is compacted through both forward and backward fault simulations. The procedure is given below.

```
PROCEDURE SOPRANO

Step 1: {Initialization}
        Transform the given circuit into the gate
        level circuit and set up a fault list, FL, con-
        taining primary faults.
Step 2: {RPT session}
        If all faults are detected, GOTO step 4.
        If consecutive two packets of random pat-
        terns do not detect any new fault, GOTO
        step 3.
        Generate a packet of random patterns and
        perform a fault simulation.
        FOR each fault $f_i$ in FL DO
          FOR each test pattern $t_p$ in the packet DO
            If $t_p$ detects $f_i$, mark $t_p$ as a $T_2$ of $f_i$.
          END FOR
        END FOR
        FOR each fault $f_j$ in FL DO
          FOR each test pattern $t_p$ in the packet DO
            If $t_p$ initializes $f_j$, mark $t_p$ as a $T_1$ of $f_j$.
          END FOR
          Eliminate $f_j$ if both $T_1$ and $T_2$ are found.
        END FOR
        GOTO Step 2.
Step 3: {DTPG session.}
        If all faults are considered, GOTO Step 4.
        Select a fault $f_i$ from FL.
        Generate a test pattern for the fault.
        IF a test $t_p$ is derived THEN
          If unspecified inputs exist in $t_p$, assign ran-
          dom patterns to the unspecified inputs.
          Apply $t_p$ to the circuit and perform a fault
          simulation.
          FOR each fault $f_j$ in FL DO
            If $t_p$ detects $f_j$, mark $t_p$ as a $T_2$ pattern
            of $f_j$.
          END FOR
          FOR each fault $f_k$ in FL DO
            If $t_p$ initializes $f_k$, mark $t_p$ as a $T_1$ pattern
            of $f_k$.
            Eliminate $f_k$ if both $T_1$ and $T_2$ are found.
          END FOR
        ELSE IF $f_i$ is identified as redundant,
          Mark $f_i$ as REDUNDANT.
        ELSE
          Mark $f_i$ as ABORTED.
        END IF
        GOTO step 3.
Step 4: {This step arranges the test patterns to min-
        imize the overall test set size.}
Step 5: {This step compacts the test sequence de-
        rived in step 4 through forward and back-
        ward simulations.}

END SOPRANO
```

In Step 3 of the above procedure, we selected a fault at a check point (i.e., a primary input or a fanout branch) first, if possible. Otherwise, a fault is selected arbitrarily. It should be noted that the fault free responses of the circuit (which are necessary to determine initialization patterns of SOP faults in Step 2 and Step 3) are obtained during the fault simulations. In the following, we explain Step 4 and Step 5 in detail.

Step 4. Minimization of The Test Sequence

Once two test patterns for each fault are derived, the overall test sequence is minimized in this step. Let us denote each distinct test pattern as $t_1, t_2, \ldots, t_n$ and each test pattern pair as $s_1, s_2, \ldots, s_m$. Each s_i represents an ordered pair $\langle T_i, t_j\rangle$ where t_i is an initialization pattern and t_j a test pattern. Our aim is to find a minimal length test sequence S where S contains each test sequence s_i as a subsequence. This problem is an instance of the problem of finding a minimal length superstring (where the length of each substring is two) [19]. The linear time algorithm proposed by Gallant et al. is employed in SOPRANO [19]. The algorithm guarantees to find a minimal length test sequence for the given test patterns. The time complexity is linear with respect to the number of two pattern tests. Recently, Chakravarty and Ravi presented a similar method of compacting test sequences in [20].

In the following, we will illustrate the algorithm using an example. Let us suppose that ten pairs of two pattern tests, $\langle t_1, t_2\rangle$, $\langle t_1, t_5\rangle$, $\langle t_2, t_3\rangle$, $\langle t_2, t_5\rangle$, $\langle t_3, t_4\rangle$, $\langle t_4, t_2\rangle$, $\langle t_5, t_6\rangle$, $\langle t_6, t_1\rangle$, $\langle t_7, t_8\rangle$, $\langle t_8, t_7\rangle$, are derived. A directed graph model of these test patterns is shown in Figure 3. Each node denotes a test pattern. A directed edge from a node t_i to a node t_j denotes a test pair $\langle t_i, t_j\rangle$. In the given graph, the traversing starts at the node t_1 since it is the only node which has more outgoing edges (2) than incoming edges (1). Suppose that we traversed edges following the path $\langle t_1, t_2, t_5, t_6, t_1, t_5\rangle$. The traversing is stopped at the node t_5 since it has no more outgoing edges. The path traversed is deleted from the graph G and added to the test sequence S. The remaining paths $\langle t_2, t_3, t_4, t_2\rangle$ and $\langle t_7, t_8, t_7\rangle$, form cycles. Pick a node t_2 and traverse the cycle $\langle t_2, t_3, t_4, t_2\rangle$. Since the cycle has a common node t_2 with S, insert it to the test sequence S without duplicating the intersecting node t_2. Then S becomes $\langle t_1, t_2, t_3, t_4, t_2, t_5, t_6, t_1, t_5\rangle$. Since there is no common node between the cycle $\langle t_7, t_8, t_7\rangle$ and S, the cycle is added at the end of S. The final test sequence S is $\langle t_1, t_2, t_3, t_4, t_2, t_5, t_6, t_1, t_5, t_7, t_8, t_7\rangle$.

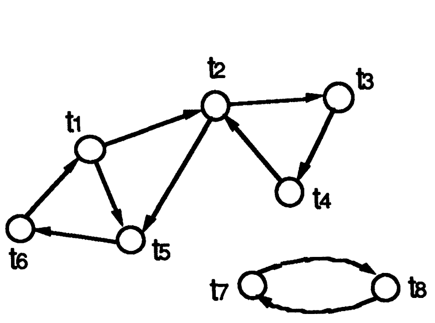

FIGURE 3 Graph model of two pattern tests.

The above minimization procedure is formally described in the following. The first step of the algorithm is to construct a directed graph from the given test patterns. In the second step, a test sequence S is constructed by traversing the directed graph.

```
PROCEDURE TEST_SEQUENCE_MINIMIZATION

Step 1: Construct a direct graph $G$ with the given
        test patterns.
        Test sequence $S$ be initialized to be empty.
        FOR each node $v$ in $G$, compute the numbers of the outgoing edges and the incoming
        edges.
Step 2: WHILE there exists a node $v$ in $G$ in which
        the number of the outgoing edges is larger
        than that of the incoming edges DO
          Traverse edges as far as possible starting
          from $v$.
          Delete the edges from $G$ and add them
          into $S$.
        END WHILE
        {The remaining edges form cycles.}
        WHILE there exist a node $v$ in $G$ DO
          Traverse a cycle $C$ starting from $v$ and
          delete it from $G$.
          IF $C$ intersects with $S$ THEN
            Insert $C$ after an intersecting node of $S$
            without duplicating the intersecting
            node.
          ELSE
            Add $C$ at the end of $S$.
          END IF
        END WHILE

END TEST_SEQUENCE_MINIMIZATION
```

Step 5: Compaction of The Test Sequence

In this step, SOPRANO compacts the test sequence by applying the test patterns in the forward and back-

TABLE II
Compaction of a Test Sequence
(a) Forward Simulation

test sequence	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
detected faults		X	X			X			X	X

TABLE II
Compaction of a Test Sequence
(b) Backward Simulation

test sequence	t_7	t_8	t_9	t_4	t_5	t_0	t_1	t_2
detected faults		X	X	X			X	

ward order. In the following, we illustrate the procedure using an example. Suppose that a test sequence of ten patterns $\langle t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\rangle$ is obtained in Step 4 of SOPRANO. It should be noted that a pattern t_i may be identical to t_j as shown in the example of Step 4. Then SOPRANO applies the test sequence in the given (forward) order and performs fault simulations. Through the fault simulations, SOPRANO checks if each test pattern detects at least one new fault. Suppose that only five test patterns marked with "X" detect one or more new faults as shown in Table II-(a). Since a test pattern t_i could be an initialization pattern of t_{i+1}, t_i can be eliminated only if t_i and t_{i+1} do not detect any new fault. For example, t_3 and t_6 are eliminated from the test sequence, but t_0, t_4 and t_7 are not. The remaining test sequence is divided into three subsequences, $\langle t_0, t_1, t_2\rangle$, $\langle t_4, t_5\rangle$ and $\langle t_7, t_8, t_9\rangle$ such that the first test patterns of the subsequences (i.e., t_0, t_4 and t_7) do not detect any new fault, but serve as initialization patterns. The subsequences are applied in the backward order as shown in Table II-(b). Similarly, test patterns t_5 and t_2 are eliminated from the test sequence. The final test sequence after the two simulations is $\langle t_7, t_8, t_9, t_4, t_0, t_1\rangle$. The above procedure is formally described in the following.

```
PROCEDURE TEST_COMPACTION

Step 1: {Forward Simulation}
        Apply the given test patterns in the forward
        order and perform fault simulations.
        Discard the test patterns which serve as nei-
        ther initialization patterns nor test patterns.
Step 2: {Backward Simulation}
        Divide the obtained test sequence into sub-
        sequences such that the first pattern of each
        subsequence serves as only an initialization
        pattern.
        Apply the subsequences in the backward or-
        der and perform fault simulations.
        Discard the test patterns which serve as nei-
        ther initialization patterns nor test patterns.

END TEST_COMPACTION
```

As a conclusion of this section, we note the differences between automatic test pattern generators (ATPGs) for stuck-at faults and SOPRANO, an ATPG for SOP faults. SOPRANO requires the identification of initialization patterns in Step 2 and Step 3 and the organization of test patterns in Step 4, which are unnecessary for an ATPG for stuck-at faults. In Step 5, SOPRANO requires both forward and backward fault simulations for test compaction, while an ATPG for stuck-at faults requires only backward fault simulation.

IV. EXPERIMENTAL RESULTS

SOPRANO has been implemented using approximately 5000 lines of *C* language code. It currently runs on SUN 386i workstations. As explained in the previous section, SOPRANO derives test patterns using a gate level test pattern generator and a gate level fault simulator in which zero gate delays are implicitly assumed. When gate delays are considered, some initialization patterns may fail to set the intended values at the faulty gate outputs [10]. As a result, the following T_2 patterns fail to detect intended faults, which otherwise would be detected. To measure the effective fault coverage under non-

TABLE III
Gate Delay Assignment

gate	number of inputs				
	1	2	3	4	≥ 5
inverter	1	-	-	-	-
buffer	2	-	-	-	-
NAND/NOR	-	2	3	4	5
AND/OR	-	3	4	5	6

zero gate delays, we implemented a SOP fault simulator in which gate delays are considered [21]. The transport delay model is employed in the simulator. The delay values used in the simulator are shown in Table III.

To measure the performance of SOPRANO, we used eight benchmark circuits presented in ISCAS85 [22]. We assume that the circuits are composed of only primitive FCMOS gates (i.e., AND, OR, NAND, NOR and inverter) and buffers. Buffers are assumed to be composed of a series of two inverters. The backtracking limit of the deterministic test pattern generator is set to ten. Experimental results are given in Table IV. The results are the average of ten experiments with different initial random seeds. The fault coverage is compared with that of Cox and Rajski presented in [7]. In Cox and Rajski's method, (T_1, T_2) pairs of test patterns are generated randomly and T_1 and T_2 patterns of a pair differ in only one bit position. (Cox and Rajski's results are chosen for comparison, since, to our knowledge, they are the only experiments performed on benchmark circuits which are available in the open literature). In the table, the column headings fc and fc_d are the SOP fault coverages assuming zero gate delays and modeled gate delays, respectively. The CPU time is measured on a SUN 386i workstation.

From Table IV, the average SOP fault coverage of SOPRANO is 97.90% under zero gate delays and 94.57% under the modeled gate delays. The corresponding SOP fault coverage of Cox and Rajski's method is 85.91% and 79.27%, respectively. Except for the smallest benchmark circuit (C880), the SOP fault coverage of SOPRANO is much higher than that of Cox and Rajski's method. CPU times of SOPRANO for all the circuits except the largest one are less than about one minute. (The CPU times for Cox and Rajski's experiments are not available in [7].) The reduction of SOP fault coverage of SOPRANO (by 3.3%) under gate delays is due to the invalidation of initialization patterns. The switch level test pattern generation considering delays could increase the fault coverage. However, the switch level test generation is very time consuming.

Another observation to be noted is that the average number of test patterns of SOPRANO is reduced by 50.8% on the average through the forward and backward simulations in Step 5. However it in-

TABLE IV
Experimental Results of SOPRANO

name	SOPRANO				Cox & Rajski	
	no. of SOP faults	CPU time (secs)	fc (%)	fc_d (%)	fc (%)	fc_d* (%)
C880	1112	5.0	100.00	98.97	99.59	99.59
C1355	1610	12.8	98.38	92.17	92.86	85.90
C1908	2378	19.1	99.52	92.12	89.42	87.29
C2670	3269	54.5	94.25	92.70	61.32	59.04
C3540	4608	73.0	95.27	93.20	89.04	84.11
C5315	6693	29.5	99.09	94.95	83.10	82.43
C6288	7216	68.8	99.04	94.82	97.87	67.32
C7552	9656	267.9	97.68	93.47	70.81	68.46**
average	4568	66.3	97.90	94.05	85.51	79.27

* This is the fault coverage of robust test patterns.
** Experiments did not run to completion.

TABLE V
Comparison with an ATPG for Stuck-At Faults and SOPRANO

name	no. of faults		no. of tests		fault coverage (%)		aborted faults		redundant faults		CPU time (secs)	
	SAT	SOP	SAT	SOP	SAT	SOP	SAT	SOP	SAT	SOP	SAT	SOP
C880	942	1112	66	202	100.00	98.97	0	0	0	0	3.4	5.0
C1355	1574	1610	83	337	98.87	92.17	9	18	8	8	9.8	12.8
C1908	1879	2378	127	401	99.49	92.12	2	4	7	7	11.6	19.1
C2670	2747	3269	126	397	95.74	92.70	31	69	86	119	41.3	54.5
C3540	3428	4608	174	703	96.00	93.20	0	2	137	216	33.3	73.0
C5315	5350	6693	141	609	98.90	94.95	0	1	59	60	17.2	29.5
C6288	7744	7216	37	253	99.56	94.82	0	18	34	51	33.8	68.8
C7552	7550	9656	239	793	98.25	93.47	60	129	71	95	184.8	267.9

curred the increase of processing time. The average CPU time spent for the simulations is 27.4% of the total CPU time.

Since SOPRANO considers equivalent gate level circuits and potentially equivalent stuck-at faults (rather than switch level CMOS circuits and SOP faults), it is interesting to compare the performance of SOPRANO with that of an ATPG for stuck-at faults. We implemented an ATPG for stuck-at faults, called SAT-ATPG, for comparison. SAT-ATPG uses the same FAN algorithm and the same parallel pattern single fault simulator used in SOPRANO. Experimental results of SOPRANO and SAT-ATPG are compared in Table V. For both experiments, the backtracking limit is set to ten. The results are again the average of ten experiments. The CPU time is measured on a SUN 386i workstation. The column headings of the table are self-explanatory.

From Table V, the average size of test patterns of SOPRANO is 461 and that of SAT-ATPG is 124. The average size of the SOP fault test set is about 3.7 times larger than that of the stuck-at test set. The average CPU time of SOPRANO is 66.3 seconds and that of SAT-ATPG is 41.9 seconds. SOPRANO takes about 58% more CPU time on the average than SAT-ATPG. The larger test set size and both the forward and backward simulations for the test compaction are the main reasons for the longer CPU times of SOPRANO. The average fault coverage (considering gate delays) of SOPRANO is 94.57% and that of SAT-ATPG is 98.35%. The lower SOP fault coverage for SOPRANO is due to the invalidation of some initialization patterns. Notice that the fault coverage of SAT-ATPG is the stuck-at fault coverage, not the SOP fault coverage. When we directly apply the test patterns derived by SAT-ATPG to detect SOP faults, the SOP fault coverage is much lower than the stuck-at fault coverage. For example, only 85.78% of SOP faults ar detected by stuck-at test sets for C3540 (while 93.20% of SOP faults are detected by SOPRANO). Hence, SOPRANO achieves much higher SOP fault coverage than SAT-ATPG. A detailed analysis on the SOP fault coverage of stuck-at test sets is available in [14]. It is interesting to note that the number of aborted and redundant faults for SOPRANO is always larger than that of SAT-ATPG. The reason for the larger redundant and aborted faults for SOPRANO is due to larger number of non-equivalent SOP faults. (Conversely, more redundant and aborted faults are collapsed into equivalent faults in SAT-ATPG). Finally, C7552 has many redundant and hard-to-detect faults. Hence, SOPRANO as well as SAT-ATPG perform many backtrackings to derive test patterns or identify the redundant faults. This explains long CPU times of SOPRANO and SAT-ATPG.

V. SUMMARY

In this paper, we described an automatic test pattern generator, SOPRANO, for stuck-open (SOP) faults in CMOS combinational circuits. The essential idea of SOPRANO is to convert a CMOS circuit and SOP faults into an equivalent gate level circuit and equivalent stuck-at faults, respectively. Then SOPRANO derives test patterns using a gate level test pattern generator and a gate level fault simulator. Several heuristics are also introduced in SOPRANO to reduce test set size. The experimental results on the eight benchmark circuits show that SOPRANO achieves high SOP fault coverage and short processing time. The average SOP fault coverage is 97.90% assuming zero gate delays and 94.57% considering gate delays. When compared to an ATPG for stuck-at faults, the average CPU time of SOPRANO is only 58% longer and the average test set size 3.7 times larger.

As explained in Section II, SOPRANO currently deals with CMOS circuits consisting of only FCMOS gates. CMOS circuits with pass transistors cannot be modelled into a gate level circuit directly. A future research area is to expand the capability of SOPRANO to include general CMOS combinational circuits which may include pass transistor logic.

References

[1] R.L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell Sys. Tech. J.*, Vol. 57, No. 5, pp. 1449–1474, May–June 1978.

[2] K.W. Chiang and Z.G. Vranesic, "On Fault Detection in CMOS Logic Networks," 20th Design Automation Conference, Miami Beach, FL, pp. 50–56, June 1983.

[3] M.K. Reddy, S.M. Reddy and P. Agrawal, "Transistor Level Test Generation for MOS Circuits," Proc. 22nd Design Automation Conference, Las Vegas, NV, pp. 825–828, June 1985.

[4] H.H. Chen, R.G. Mathews and J.A. Newkirk, "An Algorithm to Generate Tests for MOS Circuits at the Switch Level," Proc. 1985 International Test Conference, Philadelphia, PA, pp. 304–312, Nov. 1985.

[5] R.I. Damper and N. Burgess, "MOS Test Pattern Generation Using Path Algebra," *IEEE Trans. on Computers,* Vol. C-36, No. 9, pp. 1123–1128, Sept. 1987.

[6] J. Rajski and H. Cox, "Stuck-Open Fault Testing in Large CMOS Networks by Dynamic Path Tracing," Proc. International Conference on Computer Design, Rye Brook, NY, pp. 252–255, Oct. 1986.

[7] H. Cox and J. Rajski, "Stuck-Open and Transition Fault Testing in CMOS Complex Gates," Proc. 1988 International Test Conference, Washington D.C., pp. 688–694, Sept. 1988.

[8] S.K. Jain and V.D. Agrawal, "Test Generation for MOS Circuits Using D-Algorithm," Proc. 20th Design Automation Conference, Miami Beach, FL, pp. 64–70, June 1983.

[9] S.M. Reddy, V.D. Agrawal and S.K. Jain, "A Gate Level Model for CMOS Combinational Logic Circuits with Application to Fault Detection," 21st Design Automation Conference, Albuquerque, NM, pp. 504–509, June 1984.

[10] S.M. Reddy, M.K. Reddy and V.D. Agrawal, "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits," Proc. 14th International Symposium on Fault-Tolerant Computing, Orlando, FL, pp. 44–49, June 1984.

[11] K. Kinoshita, Private Communication.

[12] Y.M. El-Ziq and R.J. Cloutier, "Functional-Level Test Generation for Stuck-Open Faults in CMOS VLSI," Proc. 1981 International Test Conference, Philadelphia, PA, pp. 536–546, Oct. 1981.

[13] R.J. Chandramouli, "On Testing Stuck-Open Faults," Proc. 13th International Symposium on Fault-Tolerant Computing, Milan, Italy, pp. 258–265, June 1983.

[14] H.K. Lee, D.S. Ha and K. Kim, "Test Pattern Generation for Stuck-Open Faults Using Stuck-At Test Sets in CMOS Combinational Circuits," Proc. 26th Design Automation Conference, Las Vegas, NV, pp. 345–350, June 1989.

[15] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers,* Vol. C-30, No. 3, pp. 215–222, March 1981.

[16] H. Hujiwara and T.K. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. on Computers,* Vol. C-32, No. 12, pp. 1137–1144, Dec. 1983.

[17] M.H. Schulz, E. Trischler and T.M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. on Computer-Aided Design,* Vol. 7, No. 1, pp. 126–137, Jan. 1988.

[18] D. Harrel, R. Sheng and J. Udell, "Efficient Single Fault Propagation in Combinational Circuits," Proc. International Conference on Computer Aided Design, Santa Clara, CA, pp. 2–5, Nov. 1987.

[19] J. Gallant, D. Maier and J.A. Storer, "On Finding Minimal Length Superstring," *Journal of Computer and System Sciences,* Vol. 20, No. 1, pp. 50–58, Feb. 1980.

[20] S. Chakravarty and S.S. Ravi, "Computing Optimal Test Sequences from Complete Test Sets for Stuck-Open Faults in CMOS Circuits," *IEEE Trans. on Computer Aided Design,* Vol. 9, No. 3, pp. 329–331, March 1990.

[21] H.K. Lee, D.S. Ha and K. Kim, "A CMOS Stuck-Open Fault Simulator," 1989 Southeascon, Columbia, SC, pp. 1151–1155, April 1989.

[22] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Bench Mark Circuits and a Target Translator in FORTRAN," Special Session on ATPG and Fault Simulation, 1985 International Symposium on Circuits and Systems, Kyoto, Japan, June 1985.

Biographies

HYUNG KI LEE received the B.S. degree in electronic engineering from Hanyang University, Seoul, Korea, in 1980 and the M.S. and Ph.D. degrees in electrical engineering from the Virginia Polytechnic Institute and State University in 1989 and 1994, respectively. From 1980 to 1986, he served as a research engineer in the Agency for Defense Development, Korea, where he was involved in the development of computer systems for a naval weapon control system. His research interests are in the area of VLSI testing including fault simulation, test pattern generation and design verification.

He is a student member of the IEEE Computer Society.

DONG SAM HA was born in Milyang, Korea. He received the B.S. degree in electrical engineering from Seoul National University, Korea, in 1974 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Iowa, Iowa, in 1984 and 1986, respectively.

Since the fall of 1986 he has been a faculty in the Department of Electrical Engineering of the Virginia Polytechnic Institute and State University. Currently, he is an associate professor of the Department. From 1975 to 1979, he served as a research engineer in the Agency for Defense Development, Korea, During the period, he was involved in various projects, including the development of airborne equipment for communications and navigation and the development of data communication systems.

Dr. Ha is a member of the IEEE Computer Society and the Communications Society.