Multiple Stuck-at Fault Diagnosis in Logic Circuits

Younès KARKOURI, El Mostapha ABOULHAMID

Dép. d'informatique et de recherche opérationnelle
Université de Montréal, C.P. 6128, Succ. "A"
Montréal, (Québec), H3C-3J7, Canada.

ABSTRACT

A new method to fault diagnosis in combinational circuits is presented. We consider multiple stuck-at-(0/1) faults at the gate level. We introduce the concept of *frontier faults* which reduce the number of faults to consider and are equivalent to the set of all multiple faults; however, we do not enumerate all the possible multiple faults. The diagnosis is performed in two consecutive steps. Forward propagation that determines, for each line in the circuit, its fault free value and the potential effect(s) from other faulty lines that can propagate to it. Backward implication is performed from the primary outputs toward the primary inputs and determines, given the circuit response, the value(s) carried by each line. Some of the deduced values imply that either the line is not faulty, the subnetwork driving the line contains fault(s), or the line itself is faulty. The method uses the concept of *Parallel-Pattern-Multiple-Fault-Propagation* which allows to analyze simultaneously bit strings of responses. It was applied to benchmark circuits from ISCAS'85 and '89 sets, to locate faulty and fault-free lines in a reasonable cost.

1. INTRODUCTION

A digital circuit fails when its observed behavior is different from its expected one. If the circuit is to be repaired or the used designing process is to be corrected, we have to locate (diagnosis) the cause of the errors [2, 10]. First, we transform the circuit into a model that facilitates its analysis and simulation while preserving its logical nature. Second, to counteract the large magnitude of considering all physical failures affecting the circuit components, we model many of them as a set of logic lines permanently stuck-at the value 0 or 1 at the gate level [4]. Therefore, a circuit consisting of n lines may have up to 3^n-1 possible multiple stuck-at faults, making it impractical to deal with all the possible ones. Some of the works assume that, after manufacture, the circuits are frequently tested so they don't contain more that one fault at a time (single fault model). However, this frequent test strategy was shown to be inefficient in the presence of redundant faults [9], when a defect appears as a multiple fault, or when test sets are incomplete [11].

Since, there has been a great interest in multiple faults diagnosis in combinational circuits represented at the gate level, and the methods presented in the literature tackle the problem using classes of equivalent faults rather then the set of all multiples faults [3, 7]. In our case, we consider that a fault is easier to observe when it is closer to a primary output, so we delay the effect of equivalent faults as far as possible to be nearest the primary outputs [15]. In this way, we reduce considerably the number of multiple faults which have to be dealt with.

We present in this paper a method for multiple fault diagnosis based on the conservative approach to fault analysis [15]. The fault analysis method determines, for a given test set, the lines that are not faulty in order to observe the fault free response only, while this diagnosis method determines in addition the lines that are faulty when the observed response is erroneous.

Our approach to diagnosis uses an early detection of faults which makes it different from the previous ones [1, 3, 7, 8]. Given an input vector and the corresponding response, our method identifies faults that are either detected, or that may be detected if there sites are not hidden by still undetected faults. A fault is said *detected* if we determine its presence or absence in the circuit under test (CUT) in order to produce the corresponding erroneous response. This early detection of faults makes our algorithm much simpler than methods requiring fault enumeration [7], backtracking [1], or pairs of vectors [8]. Also, since all equations used in our system are Boolean, they are implemented on bit strings. This allows us to

The work is supported by NSERC MEF Grants No. MEF0040113 and OGP0000861. The experimental work is carried on workstations from the Canadian Microelectronics Corporation.

perform diagnosis with up to 32 bits simultaneously [16] (typically the length of a machine word) while considering all multiple faults, resulting in a *parallel-pattern-multiple-fault-propagation*, and thus a very efficient implementation. The method was applied to benchmark circuits from [5, 6] and gives an order of magnitude speed improvement over existing methods. None of the methods presented in [1, 3, 7] have reported benchmark results on these circuits.

The rest of this paper is organized as follows: in Section 2, the fault model is described, Section 3 presents the diagnosis method and a complete example, and finally, experimental results are presented in Section 4.

2. THE FAULT MODEL

Based on the circuit topology, we perform fault collapsing to reduce the number of faults to deal with. Therefore, restricted combinations of the remaining faults in the CUT constitute the set of frontier faults that are equivalent to all multiple faults.

2.1. Fault Collapsing

We assume that faults occur on circuit lines only; the gates are assumed to perform fault free functions. We consider one fault to be the representative of an equivalent class of faults. The effect of this fault is delayed to be closer to the primary outputs. For example, if there is a s-a-0 on an input of an AND gate, we remove it and place it on the output. We do not have to consider, for example, that all the inputs of an AND gate are simultaneously s-a-1, because we assume an equivalent s-a-1 fault at the output of the gate. The following faults are considered after collapsing [15]:

- s-a-1 (s-a-0) on all inputs of any AND/NAND (OR/NOR) gate (not present simultaneously).
- No fault on the input of an inverter or on a fanout stem.
- s-a-0 and s-a-1 on inputs of XOR and XNOR gates (not present simultaneously).
- Both s-a-0 and s-a-1 on primary outputs.

2.2. Frontier Fault Model

We consider all multiple faults consisting of all combinations of the above faults (excluding simultaneous faults on gate inputs). We then characterize frontier faults which are equivalent to all multiple faults.

In a circuit consisting of n lines, a multiple fault is represented by a tuple with at most n components and denoted by $f = (f_i^{\alpha}, f_j^{\beta}, \dots)$, $i \neq j$, where f_i^{α} represents the fault on line i : $\alpha = 0$ for s-a-0, $\alpha = 1$ for s-a-1. A line k missing in the tuple is not faulty in that specific multiple fault.

A path from line i to a primary output is said *normal* if all lines along this path are normal, i.e., faultless, but the other inputs to gates along this path may be faulty. Let $f = (f_i^{\alpha}, f_j^{\beta}, \dots)$ be a multiple fault; then f is a *frontier fault* iff for each fault f_i^{α}, there is a normal path from line i to a primary output. It is shown in [15] that every multiple fault is equivalent to a frontier fault, thus a test set that detects all frontier faults will detect all multiple faults.

According to the previous definition, a frontier fault f in the circuit under test partitions the lines into three categories, *Stuck Lines*, *Normal Lines* and *Hidden Lines*, defined as follows:

- Line i is *stuck* if $f_i^{\alpha} \in f$.
- Line j is *normal* if $f_j^{\alpha} \in f$ and there is a normal path from line j to a primary output.
- Line k is *hidden* if there is no normal path from k to a primary output.

The effective values (real values) on hidden lines are unknown since they are unobservable due to the frontier fault, and there is no algorithmic way to determine these values (Normal Path Theorem in [1]). In our case, we assume that *these lines carry fault free values*. As it will be seen in Section 3.2, this assumption does not invalidate the deductions made during the diagnosis.

3. FAULT DIAGNOSIS

Given an input vector and its corresponding circuit response (good or erroneous), the diagnosis is performed in two consecutive phases: *Forward Phase* and *Backward Phase*. During the forward phase, we evaluate the fault free circuit, and the CUT line values taking into account any possibility of propagating faulty values due to the remaining faults still to be considered. Starting from primary outputs, the backward phase attempts to deduce values on lines that produce the corresponding response. A fault on a line is detected if its deduced value imply that

either the line is not faulty, the network driving it contains fault(s), or the line itself is faulty. According to our fault model, the deductions can be made on lines that may be hidden. However, if we detect the absence of a fault, it is dropped and cannot be a component of any frontier fault. In this way, an entire class of frontier faults is dropped at once (implicit enumeration [1, 8] by proving that a line is not faulty in order to observe the specific circuit response).

3.1. Forward Phase

A line i propagates a fault effect if its actual value is different from its fault free value n_i. We model the effect of a fault on a line i using a *propagation bit* p_i. $p_i = 1$ if line i may be normal and may propagate a fault effect due to the fault(s) in the driving network. $p_i = 0$ if under the current input vector, line i carries the fault free value only or it is stuck or hidden. We represent the fault free value and the propagation bit on a line i by n_i / p_i. We also associate with each line a state variable s_i^1 (s_i^0). $s_i^\alpha = 1$ indicates that the fault s-a-α on line i (i.e., f_i^α) can be component of a frontier fault still to be considered.

First, all faults remaining after collapsing are considered simultaneously present in the CUT. The forward phase starts from the primary inputs (PIs), computes the fault free values, and propagates fault effects on the output of the encountered gates. The propagation of fault effects is based on a conservative evaluation [15]. PIs are directly controlled from the circuit environment, so we assume that they carry the fault free values only (their propagation bits are set to 0). The output of a gate that may propagate a fault effect is assumed normal, thus we take into account all possibilities of faulty values on its inputs and propagate them to its output according to the gate functionality. The propagation bit on the output of an AND gate is set to 1 in the following two cases:

(i) All its inputs have a fault free value of 1 and at least one of them may propagate a fault effect (the gate output is sensible to any change on its inputs which will change it to 0), or

(ii) All its inputs that have a fault free value of 0 may carry a fault effect or may be stuck at 1. Combining all possibilities of having 1 on the inputs, will change its output value to 1.

For all gate types, except inverters, buffers and fanout stems where the propagation bit is transmitted forward as is, the equations to compute the propagation bits are Boolean and all variables (i.e., n_i, p_i, s_i^1, s_i^0) may be vectors of up to 32 bits. The following equation determines the propagation bit p_{out} on the output *out* of a m-input AND gate (equations for all gate types are presented in the Appendix):

$$p_{out} := (\square_i\, n_i).(\square_i\, p_i) + (\square_i\, \overline{n}_i).(\square_i\, (n_i+p_i+s_i^1)).(\square_i\, (n_i+p_i))$$

Fig. 1 shows an example of the propagation bit computation for an AND gate. The applied input vector is $a = 0$ and $b = 1$. (For clarity purpose, we consider in the following a single vector; the equation uses patterns of up to 32 bits). The possibility of a s-a-1 fault on line a, combined with the fault free value 1 on line b will produce a faulty value of 1 rather then 0 on the gate output. This output is assumed to be normal in this case (consistency with the fault model) and then p_{out} is equal to 1 to indicate the presence of a possibly faulty value on line *out*.

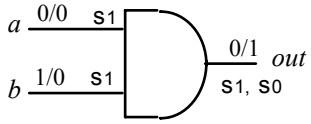

Fig. 1. Propagation bit computation for an AND gate.

3.2. Backward Phase

The forward phase propagates all fault effects to the primary outputs (POs). The backward phase then starts from the POs toward the PIs and attempts to deduce values on the CUT lines that are consistent with the observed response R on all the POs. The propagation bit p_o on each PO o is adjusted by setting it to either 0 or 1 according to its observed value r_o of R. To distinguish between a propagation bit that is set to 1 during the forward phase and deduced in the backward phase, we associate with each line i a bit variable f_i called *forced bit*. That is, $p_i = 1$ in the forward phase reflects the possibility of propagating a fault effect, while in the backward phase, $f_i = 1$ means that we determined the presence of a fault effect (i.e., p_i is deduced equal to 1) on line i. For example, assume that we obtained during the forward phase $n_o = 1$ and $p_o = 1$ on a PO o and the observed response is $r_o = 0$. Hence, the only situation to observe this value is the presence of the fault effect which changes the value of o to 0. In this case, the

propagation p_o is said to be forced and f_o is set to 1. The output is really s-a-v if no fault effect is propagated to it under the current input vector and the observed response r_o is different from its fault free value. This is computed using a Boolean state variable $faulty_o$.

There are cases where the observed response cannot be modeled in terms of a propagation bit and a line status (s_i^v), thus the observed response R could not be generated under the stuck-at fault model. In such a case, the fault in the CUT is not equivalent to any permanent multiple stuck-at fault and the diagnosis is aborted signaling the error.

When the propagation and/or the forced bit of a PO are reset to 0 or 1, the corresponding bits of the input lines to the gate feeding this PO are adjusted in consequence. These deductions are performed assuming that the gate output is normal because it may be hidden by a yet undetected fault. The consistency of this deductions is proven by considering all the possible status of the gate output according to the frontier fault model. For example, if we have reset to 0 the propagation bit on the output of an AND gate, we try to identify the input responsible of such fault effect. If such an input exists, say i, we reset its p_i to 0 and also drop its fault f_i^1 (i.e., by resetting s_i^1 to 0). On the other hand, if we determined the presence of a fault effect on the output of the gate, we try to identify the unique cause that propagates the faulty value to the output. This may result in either setting as faulty one of the gate inputs or in forcing to 1 its propagation bit. All this processing is Boolean equation based, thus it is performed on 32 input vectors simultaneously. The resulting equations for all gate types are presented in the Appendix.

In the example of Fig. 2, the fault free value on *out* is 0 and there is a possibility of a faulty value since p_{out} = 1. We assume that we observe the response r_{out} = 1. The s-a-0 on *out* is dropped, and we have the following cases:

(i) Line *out* is really s-a-1, then lines a and b are hidden, thus fault free (but any deduction on these lines will never be contradicted).

(ii) Line *out* is normal, then we force its propagation bit to be 1 (i.e., f_{out} = 1) in order to observe the current response. Therefore, the only way to observe such value on *out* is the presence of a fault effect or a fault on line a, thus its forced bit f_a is set to 1. Since line a is a primary input and p_a = 0, then $faulty_i$ is true and it is declared stuck at 1.

The deduction made on line a will never be invalidated, since a s-a-1 and *out* s-a-1 could not be components of the same frontier fault. In subsequent vectors, line *out* is still considered as possibly s-a-1, and if we observe a response r_{out} = 0, then line a s-a-1 is confirmed in order to observe a 1 on *out* when the test vector a = 0 and b = 1 were applied to the circuit. On the other hand, if we have determined, latter on, that *out* is really stuck at 1, lines a and b are hidden thus assumed fault free and the consistency of the deductions (according to the fault model) are still valid. Note also, that in order to preserve this consistency, the s-a-1 on line b is dropped when a is declared stuck line, because both faults cannot be components of the same frontier fault (we assume an equivalent s-a-1 fault on *out*).

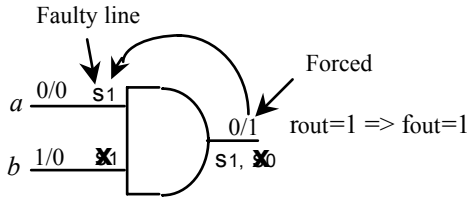

Fig. 2. Diagnosis deduction example.

According to our fault model, a deduction can be made on a line that may be hidden by others faults. However, this deduction will never be contradicted in subsequent test vectors, since the line is either declared no more hidden and then the deduction is valid, or determined hidden thus cannot influence the analyzed circuit behavior and the deduction has no effect. Hence, when a line i is declared as s-a-v (i.e., $faulty_i$ is true), the algorithm drops all faults in the fanout free region driving it because all the lines belong to such an area are hidden and thus fault free. For subsequent vectors, the detected stuck lines keep their stuck value to be consistent with the observed response. The overall diagnosis method can be summarized in the following procedure, assuming that we know the response to each of the vectors in the test set:

```
procedure Circuit_Diagnosis();
{ Pattern_Length := Read();   /* 1 ≤ Pattern_Length ≤ 32 */
  Pattern_Number := 0;
  while Input_Vectors do
  { Current_Pattern := Read_Pattern( Pattern_Length );
    Pattern_Number := Pattern_Number + 1;
    Observed_Response := Read_Response(Pattern_Number);
    while Faults_Detected do
    { Forward_Phase( Current_Pattern );
```

```
    Deduce_PO_Status( Observed_Response );
    if Inconsistency_of_the_Fault_Model then HALT;
    Backward_Phase( Observed_Response );
    /* Drop faults on hidden lines */
    for each i of the new detected stuck lines do
        Drop_FFR_Faults( i );
}}};
```

The input vectors are divided into patterns of *"Pattern_Length"* bits each. Each pattern is reapplied to the CUT as many times as it permits detecting faults. Benchmark experiments show that a pattern is reapplied at most 10 times. The forward and backward phases are performed at each iteration, and are linear time algorithms in the circuit size. At the end of the diagnosis, the results obtained are not invalidated in the presence of undetected or undetectable faults. A fault is detected only if it is not masked by another fault and its effect produces the observed response (when its site is not hidden). Some of the faults may remain undetected because either the inherent pessimism in the method (due to its conservatism) reduces the deduction power of the backward phase, the analyzed test set is not sufficient to detect all faults, or they are masked by the faults located in the CUT. In order to remedy the pessimism, we perform event analysis between adjacent vectors within the same pattern in order to retrace paths that have propagated an event, thus fault free [15]. This is similar to the event and stem region analysis of [8, 12], however, our method does not make explicit analysis using sets of pairs of vectors [8] which is impractical when considering patterns of 32 bits.

3.3. A Complete Diagnosis Example

In this section, we present a complete example of the diagnosis that makes the emphasis on the contribution of our method compared to the existing ones. First, fault collapsing were performed on the circuit in Fig. 3, and only restricted combinations of these faults (according to the definitions in Section 2) constitute the set of frontier faults. For example, the faults f_b^1, f_f^1 and f_g^1 cannot belong to the same frontier fault because there is no normal path from *b* to the output *j*.

In Fig. 3, the circuit is evaluated for the test vector *a* = 1, *b* = 0, *c* = 1 and *d* = 1. Line *e* carries a fault effect (p_e = 1) propagated from the combination of f_b^1 and n_c = 1. Line *h* also propagates a fault effect emanating from the combination of n_a = 1 and either p_f = 1 or f_f^1. Fault effects propagate also to *i* and finally to the primary output *j*.

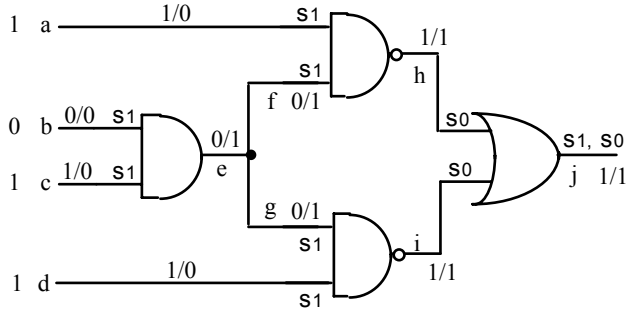

Fig. 3. Forward Propagation.

In the backward phase (Fig. 4), we first assume that we have observed the value 0 on the output *j*, i.e., r_j = 0. Thus, we have the following situations:

1: f_j^0 □ All the other lines are hidden, then any deduction will never be contradicted.

2: *j* normal □ *h* = 0 and *i* = 0:

2.1: f_h^0 and *i* normal:

□ *a* and *f* hidden.

□ *g* = 1: • f_g^1 □ *b*, *c* and *e* hidden.

• *g* normal □ *e* = 1 □ *b* is stuck at 1.

2.2 *h* normal and f_i^0:

□ *d* and *g* hidden.

□ *f* = 1: • f_f^1 □ *b*, *c* and *e* are hidden.

• *f* normal □ *e* = 1 □ *b* is stuck at 1.

2.3 *h* and *i* normal:

□ *f* = 1 and *g* = 1:

• f_f^1 and f_g^1 □ *b*, *c* and *e* are hidden.

• f_f^1 and *g* normal □ *e* = 1 □ *b* is stuck at 1.

• *f* normal and f_g^1 □ *e* = 1 □ *b* is stuck at 1.

• *f* and *g* normal □ *e* = 1 □ *b* is stuck at 1.

Since the CUT may contain at most one frontier fault, each time we reach a line in the deductions, it is assumed normal in order to deduce on possibly hidden lines. In this example, we reach line *b* while considering the path *j-h-f-e* or *j-i-g-e* as normal. When combining the deductions made in every case, we conclude that line *b* is either stuck at 1 or hidden. The equivalent deductions made by our diagnosis method are based one such cases without, however, considering each one of them. As illustrated in Fig. 4, these deductions are performed in 6 steps. For clarity in the

figure, the forced bit of each line is illustrated as an arrow when it is equal to 1, with circled numbers representing the step. For example, $n_i = 1$, $p_i = 1$ and if f_i is deduced to be equal to 1, then line $i = 0$ as done is step 2.

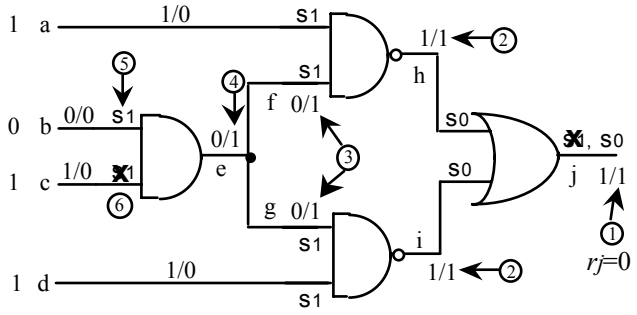

Fig. 4. Backward Deduction.

Table 1 shows the complete diagnosis of the circuit example in Fig. 3, for the test set $T = \{t_1 = 1011, t_2 = 0111, t_3 = 1110\}$ and the response $R = 011$. The table shows the current test vector, the corresponding observed response r_j, and the located and dropped fault(s). In fact, we analyzed a pattern of 3 bits simultaneously, which is $a = 101$, $b = 011$, $c = 111$ and $d = 110$ (to show the deduction made on each single vector, we separate them in Table 1).

Table 1. Diagnosis Example.

T	$a\ b\ c\ d$	r_j	Located	Dropped
t_1	1 0 1 1	0	f_b^1	f_j^1, f_c^1
t_2	0 1 1 1	1	-	f_j^0, f_h^0, f_a^1
t_3	1 1 1 0	1	-	f_i^0, f_d^1

After the analysis of this test set, the faults f_f^1 and f_g^1 are not detected. No algorithmic method can claim the absence or presence of these faults because they are redundant. In our case, we were able to declare line b stuck at 1. In fact, b can be hidden by the frontier fault (f_f^1, f_g^1) and this deduction has no effect on the circuit behavior, or in the absence of f_f^1 and/or f_g^1, our diagnosis is valid because line b must be s-a-1 in order to observe the response $R = 011$. This early detection of faults is allowed using the frontier fault model and makes our method to diagnosis efficient compared to the existing ones [1, 3, 7, 8] which cannot get such deductions.

4. EXPERIMENTAL RESULTS

We have define the fault coverage according to our fault model and fault detection strategy. For a circuit of n lines, there are $2n$ possible faults. Therefore, the coverage is defined as *the ratio of faults detected to 2n possible faults*. For example, in the circuit of Fig. 4, the total number of faults is 18. After the analysis of the responses, one fault was located and 15 faults were dropped, i.e., 16 faults were detected among the 18, leading to 88.9% fault coverage.

The diagnosis experiment is performed as follows: First, given a test set, a separate tool simulates the circuit with a randomly injected frontier fault consisting of several faults, and we collect the circuit response. The diagnosis is then started, assuming the possible presence of all faults, and for each input pattern of 32 bits and its response, we perform forward and backward sweeps. The goal is to locate correctly the faults of the injected frontier fault, and to drop all the other ones that are not present in order to produce the analyzed response. The faults remaining after diagnosis are either masked, dominated or equivalent to the located ones, if any.

Table 2 summarizes the results obtained from several experiments on some benchmark circuits from [5, 6]. Circuits in [5] are assumed to be fully scannable and are transformed into combinational ones. The table gives the circuit name, the number of faults remaining after collapsing, the number of randomly injected faults ("inject."), the test size corresponding to the total number of single vectors, the coverage as defined earlier and the total CPU time in seconds (on a SPARC-Station 2) to analyze the responses of the whole test set. For example, for the c880 we collected the responses of 160 input vectors when a frontier fault consisting of 5 faults was injected in the circuit. Then diagnosis analyzed these responses assuming the possibility of all faults. At the end, the faults of the injected frontier fault were correctly located, and all the other ones were dropped. This was performed with 5 patterns of 32 bits each (160□32), and it took only 2.51 seconds to get this diagnosis result.

Table 2. Diagnosis Experiments.

Circuit	Faults	Inject.	Test Size	Coverage	CPU Time
ALU	175	0	16	100%	0.10s

ALU	175	1	16	99.4%	0.15s
c432	246	1	166	98.1%	1.21s
c880	692	5	160	100%	2.51s
c1908	1109	2	1569	96.3%	37.0s
s838	589	4	118	99.5%	1.34s
s953	717	6	183	99.0%	2.89s
s1196	930	10	705	98.8%	11.42s
s1488	1334	0	282	100%	8.56s
s1488	1334	8	282	94.1%	10.21s
s38584	28407	9	495	95.0%	6mn 8.2s

For the ALU (74LS181), it is well known that 16 vectors detect all multiple faults [14]. When the observed response *R* is equal to the fault free one, the pattern of 16 vectors was repeated 5 times to cover all multiple faults in only 0.1 seconds. When a frontier fault consisting of a single fault is injected, the 16-bit pattern was repeated 9 times to locate the fault, and drop all the other ones except 2 which were masked by the located fault. For the other circuits, the test sets consist of compacted tests for single faults from [13]. Some vectors are repeated in the test set in order to increase the coverage. All the faults of the injected frontier fault were correctly located by our diagnosis method, except for the s1488 where 2 among the 8 faults were not located and they remain as possible in the circuit. Circuits of large sizes (e.g. s38584 which contains 38584 lines) can also be handled, and the diagnosis is performed in a very reasonable cost and is an order of magnitude faster then the existing methods such as the analysis using pairs of vectors reported in [8].

5. CONCLUSION

We have presented a fast method to perform diagnosis in large circuits. It uses concepts from the previously developed multiple fault analysis [15]. Fault collapsing reduces the number of faults to deal with and the frontier fault model allows to detect a fault even if its site may be hidden by yet undetected faults. This makes the method efficient enough to determine, nearly all the time, the faulty sites or equivalent ones in the network. It is also able to recognize responses not generated under the stuck-at fault model. The use of Boolean equations and patterns of vectors speeds up significantly the deductions and is less time consuming than method requiring clusters of vectors, backtracking, or fault enumeration.

If the CUT is to be repaired, it is probed using an electron beam tester [10], thus our method contributes in reducing considerably the target area of the IC where the behavior of interconnections have to be tested. When the analyzed responses of a test set are equal to the fault free ones, the method identifies the faults not covered by the given test set and the lines that are not faulty. In this case, it performs multiple fault analysis and it is about 10 times faster than the analysis method reported in [15], since it analyzes sequences of up to 32 vectors simultaneously.

REFERENCES

[1] M. Abramovici, M.A. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis.", *IEEE Trans. on Computers*, vol. C-29, 1980, pp. 451-460.

[2] M. Abramovici, M.A. Breuer, A.D. Freidman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[3] D.C. Bossen, S.J. Hong, "Cause-Effect Analysis for Multiple Fault Detection in Combinational Networks.", *IEEE Trans. on Computers*, vol. C-20, 1971, pp. 1252-1275.

[4] M.A. Breuer, A.D. Friedman, *Diagnosis & Reliable Design of Digital Systems*, Computer Science Press, 1976.

[5] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", *Proc. of the Intl. Symp. Circuits and Systems*, 1989, pp. 1929-1934.

[6] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", *Proc. of the Intl. Symp. Circuits and Systems*, 1985,

[7] C.W. Cha, "Multiple Fault Diagnosis in Combinational Networks", *Proc. of the 16th Design Automation Conf.*, 1979, pp. 149-155.

[8] H. Cox, J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis.", *IEEE Trans. on Computer-Aided Design*, vol. 7, no. 7, 1988, pp. 813-833.

[9] A.D. Friedman, "Fault Detection in Redundant Circuits.", *IEEE Trans. Electron. Comput.*, vol. EC-16, 1967, pp. 99-100.

[10] S. Gorlich, H. Harveck, P. Kebler, E. Wolfgang, K. Zibert, "Integration of CAD, CAT, and Electron Beam Testing for IC Internal Logic Verification", *Proc. of the Intl. Test Conf.*, 1987, pp. 566-574.

[11] J.L.A. Hughes, "Multiple Fault Detection Using Single Fault Test Sets.", *IEEE Trans. on Computer-Aided Design*, vol. 7, no. 1, 1988, pp. 100-108.

[12] F. Maamari, J. Rajski, "A Reconvergent Fanout Analysis for Efficient Exact-Fault Simulation of Combinational Circuits", *Proc. of the 18th Fault-Tolerant Computing Symp.*, 1988, pp. 122-127.

[13] I. Pomeranz, L.N. Reddy, S.M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits", *Proc. of the IEEE Intl. Test Conf.*, 1991, pp. 194-203.

[14] J. Rajski, "GEMINI - A Logic System for Fault Diagnosis Based on Set Functions", Technical Report TR-87-5R, McGill University, 1987.

[15] A. Verreault, E.M. Aboulhamid, Y. Karkouri, "Multiple Fault Analysis using a Fault Dropping Technique", *Proc. of the 21th Fault-Tolerant Computing Symp.*, 1991, pp. 162-169.

[16] J.A. Waicukauski, E.B. Eichelberger, D.O. Forlenza, E. Lindbloom, T. McCarthy, "Fault Simulation for Structured VLSI.", *VLSI Systems Design*, vol. 6, no. 12, 1985, pp. 20-32.

APPENDIX

DIAGNOSIS EQUATIONS

A. Propagation Bit Computation

- AND/NAND: $p_{out} := (\underset{i}{\square}\, n_i) \,.\, (\underset{i}{\square}\, p_i) + (\underset{i}{\square}\, \overline{n}_i) \,.\, (\underset{i}{\square}\, (n_i + p_i + s_i^1)) \,.\, (\underset{i}{\square}\, (n_i + p_i))$
- OR/NOR: $p_{out} := (\underset{i}{\square}\, \overline{n}_i) \,.\, (\underset{i}{\square}\, p_i) + (\underset{i}{\square}\, n_i) \,.\, (\underset{i}{\square}\, (\overline{n}_i + p_i + s_i^0)) \,.\, (\underset{i}{\square}\, (\overline{n}_i + p_i))$
- XOR/XNOR: $p_{out} := (\underset{i}{\square}\, p_i) + (\underset{i}{\square}\, (n_i \,.\, s_i^0 + \overline{n}_i \,.\, s_i^1))$

B. Deduction Equations

- AND: (for NAND gates, substitute $n_{out} \times \overline{n}_{out}$)

$$f_i := 1 \Leftrightarrow f_{out} \,.\, p_{out} \,.\, (\overline{n}_i + n_{out} \,.\, p_i \,.\, (\underset{j \neq i}{\square}\, \overline{p}_j))$$

$$p_i := 0 \Leftrightarrow \overline{p}_{out} \,.\, [(\overline{n}_i \,.\, (\underset{j \neq i}{\square}\, n_j \,.\, \overline{p}_j)) + (\underset{j}{\square}\, n_j)] + (f_{out} \,.\, p_{out} \,.\, \overline{n}_{out} \,.\, n_i)$$

$$s_i^1 := 0 \Leftrightarrow (\overline{p}_{out} \,.\, \overline{n}_i \,.\, (\underset{j \neq i}{\square}\, n_j \,.\, \overline{p}_j)) + (\underset{j \neq i}{\square}\, faulty_j) + (f_{out} \,.\, p_{out} \,.\, n_{out} \,.\, f_i \,.\, p_i)$$

$$Faulty_i := f_i \,.\, \overline{n}_i \,.\, \overline{p}_i \,.\, s_i^1$$

- OR: (for NOR gates, substitute $n_{out} \times \overline{n}_{out}$)

$$f_i := 1 \Leftrightarrow f_{out} \,.\, p_{out} \,.\, (n_i + \overline{n}_{out} \,.\, p_i \,.\, (\underset{j \neq i}{\square}\, \overline{p}_j))$$

$$p_i := 0 \Leftrightarrow \overline{p}_{out} \,.\, [(n_i \,.\, (\underset{j \neq i}{\square}\, \overline{n}_j \,.\, \overline{p}_j)) + (\underset{j}{\square}\, \overline{n}_j)] + (f_{out} \,.\, p_{out} \,.\, n_{out} \,.\, \overline{n}_i)$$

$$s_i^0 := 0 \Leftrightarrow (\overline{p}_{out} \,.\, n_i \,.\, (\underset{j \neq i}{\square}\, \overline{n}_j \,.\, \overline{p}_j)) + (\underset{j \neq i}{\square}\, faulty_j) + (f_{out} \,.\, p_{out} \,.\, \overline{n}_{out} \,.\, f_i \,.\, p_i)$$

$$Faulty_i := f_i \,.\, n_i \,.\, \overline{p}_i \,.\, s_i^0$$

- XOR: with two inputs i and j: (for XNOR gates, substitute $n_{out} \times \overline{n}_{out}$)

$$f_i := 1 \Leftrightarrow f_{out} \,.\, p_{out} \,.\, \overline{p}_j \,.\, [\overline{n}_{out} \,.\, (\overline{n}_i \overline{n}_j \overline{s}_j^1 + n_i n_j \overline{s}_j^0) + n_{out} \,.\, (\overline{n}_i n_j \overline{s}_j^0 + n_i \overline{n}_j \overline{s}_j^1)]$$

$$p_i := 0 \Leftrightarrow \overline{p}_{out} \,.\, \overline{p}_j \,.\, (n_j \,.\, \overline{s}_j^0 + \overline{n}_j \,.\, \overline{s}_j^1)$$

$$s_i^1 := 0 \Leftrightarrow \overline{p}_{out} \,.\, \overline{p}_j \,.\, \overline{n}_i + f_{out} \,.\, p_{out} \,.\, n_i \,.\, f_i \,.\, p_i$$

$$s_i^0 := 0 \Leftrightarrow \overline{p}_{out} \,.\, \overline{p}_j \,.\, n_i + f_{out} \,.\, p_{out} \,.\, \overline{n}_i \,.\, f_i \,.\, p_i$$

$$Faulty_i := f_i \,.\, \overline{p}_i \,.\, (n_i \,.\, s_i^0 + \overline{n}_i \,.\, s_i^1)$$

• Fanout Stem s with n branches $i = 1, ..., n$:

$$f_s := 1 \Leftrightarrow (\underset{i}{\square}\, f_i) + (\underset{i}{\square}\, (f_i \,.\, \exists \text{ normal path from } i \text{ to a po}))$$

$$p_s := 0 \Leftrightarrow (\underset{i}{\square}\, \overline{p}_i) + (\underset{i}{\square}\, (\overline{p}_i \,.\, \exists \text{ normal path from } i \text{ to a po}))$$
