Voting Model Based Diagnosis of Bridging Faults in Combinational Circuits*

Abstract

A novel algorithm for diagnosing all two-line single bridging faults in combinational circuits is presented. It uses the accurate voting model to model bridging fault. SOPS, a compact data structure is used to represent the set of possible faults. This makes the algorithm space efficient. A set of rules is used to reduce the fault list. These rules are based on stuck-at fault simulation, making the algorithm time efficient. The diagnosis algorithm also guarantees that the fault which causes the failure will remain in the final fault list. Experimental results are presented.

1 Introduction

In a circuit, when two or more distinct lines, due to a defect, are unintentionly connected we have a **bridging fault**. We assume that the fault is such that two lines are shorted and the resistance between the two shorted points, compared to the pull-up, pull-down resistances, is negligible. These faults are known as Two-line, Single, Bridging Faults(**TSBF**s).

The problem we address here is defined as follow: Given a set of test vectors $T = \{T_i | 0 \leq i < N\}$, the circuit C, and the faulty response R(the difference between responses from fault free circuit and faulty circuit for each T_i), determine the TSBFs which can result in the faulty response R.

The possible TSBFs, computed by the algorithm, are known as **Residual Faults**. The actual fault is called **Object Fault**. For a correct diagnosis algorithm, the *Object Fault* must be a Residual Fault. The smaller the number of Residual Faults the more effective is the diagnosis algorithm.

*Research supported by NSF Grant No. MIP-9102509.

Fault diagnosis has been discussed in the literature. Most of them are for stuck-at faults[1, 2, 9, 10, 16, 20, 22, 23]. However, failures that occur frequently in some technologies, such as CMOS, are bridging faults[11, 12, 21]. Two approaches for diagnosing bridging faults, using either I_{ddq} measurement[6, 8] or logic measurement [7, 18] have been studied.

In this paper we will be concerned only with logic measurement based diagnosis of bridging faults. In [18], heuristics are used to compute the diagnostic classes of bridging faults from stuck-at fault dictionaries. For each diagnostic class the test vectors that would detect it and the faulty vectors are computed. Consequently, the algorithm requires a large space and the computation time for computing the diagnostic classes is also very high. It can therefore be used for small circuits. However, it uses a very accurate fault model, viz. the Voting Model[4], to model the faulty behavior.

The algorithm in [7] uses a novel data structure to represent the bridging faults. In addition, a modified version of the stuck-at fault dictionary is computed dynamically. The resulting algorithm was shown to be very time and space efficient and at the same time it was an effective algorithm. However it uses Wired-Or(Wired-And) model which limits its application to some subclasses of CMOS circuits.

In this paper we present an algorithm that uses the good features of both [18, 7]. Like [18] the voting model is used to accurately model the bridging faults. Unlike [18] and as in [7], the algorithm uses a compact data structure to represent all TSBFs, making it space efficient. Like [7], the paradigm it uses is to derive a set of rules which are used to reduce the fault list. These rules are based only on stuck-at fault simulation making it time efficient. The diagnostic algorithm also guarantees that the *Object Fault* will remain in the residual fault list. Modifications required to use it for the Biased-Voting Model[15] are addressed in the discussion section.

The results obtained from experiments on ISCAS85 and ISCAS89[1] benchmark circuits are very promising. In most cases it reduces the set of faults to a reasonably small number in a reasonable amount of CPU time.

In next few sections, we first discuss the fault model, and then give the definitions

[1]Obtained from MCNC benchmark project

we will use. After that we will discuss the diagnostic algorithm. Finally we will present the experimental results on ISCAS85 and ISCAS89 benchmark circuits.

2 Fault Model

A *TSBF* between lines X, Y, denoted by $< X, Y >$, is a **Non-Feedback TSBF** (**NFTSBF**) iff there does not exist any path either from X to Y or from Y to X in the fault free circuit. A TSBF that is not an NFTSBF is a **Feedback TSBF** (**FTSBF**). TSBFs will also be referred to as *faults* in the rest of the paper.

There are different models for modeling the behavior of faulty circuits in the presence of bridging faults. Two commonly used models are *Wired-OR* and *Wired-AND* model. These models model the behavior of TTL circuits in the presence of TSBFs, some important classes of static CMOS and some classes of nMOS circuits, in the presence of TSBFs[7]. In this paper we use the voting model[4], which is a more accurate model and can model the behavior of a considerably larger class of CMOS faulty circuits, in the presence of TSBF. For the completeness of the paper, we next discuss the voting model. This discussion is based on [17]. For more detail about the voting model, please refer to [4].

In the voting model, when two lines A, B are shorted, the value at the shorted point $V(A, B)$ on application of an input vector T is determined as follows. If $T(A) = T(B)$ then $V(A, B) = T(A)$. If $T(A) \neq T(B)$ then $V(A, B)$ is computed as follows. Assume $T(A) = 1$ and $T(B) = 0$. Then $V(A, B)$ depends on the ratio of pull-down strength at B and pull-up strength at A. For example in Figure 1, we can model the circuit in (a) as the potential divider of (b) when A and B are shorted. Here R_u is the equivalent resistance of the pull-up network at A, R_d is the equivalent resistance of the pull-down network at B, and V_{BF} is the voltage at the shorted point. Let V_0 be the threshold of logic zero, V_1 be the threshold of logic one, $\alpha = \frac{V_0}{V_{dd}}$, $\beta = \frac{V_1}{V_{dd}}$. If

$$\frac{R_d}{R_u + R_d} \geq \beta \ then \ \ V_{BF} \geq V_1. \tag{1}$$

In this case, the logic level at the shorted point is 1, and we say that the fault **exhibits s-a-1 behavior**. If

$$\frac{R_d}{R_u + R_d} \leq \alpha \ then \ \ V_{BF} \leq V_0. \tag{2}$$

In this case, the logic level at the shorted point is 0 and we say that the fault **exhibits s-a-0 behavior**. In other words, the resulting signal value at the shorted point depends on the relative strengths of R_d, R_u. This is equivalent to a vote, where the network that can source or sink the most current wins.

The resistance of the pull-up and pull-down networks depend on the technology, the transistor sizes and the number of conducting transistors. For example, in Figure 1, assume that all the transistors are minimum-sized, and the n-channel transistors are 2.5 times stronger than the p-channel transistors(i.e n-channel transistors can sink 2.5 times more current than p-channel transistors can source)[17]. If input a is low, and inputs b, c, d and e are high, the ratio $\frac{R_u}{R_d}$ is 2.5, and $\frac{R_d}{R_u+R_d}$ is 0.29. Assume the threshold for logic zero to be $2V$ out of $5V$, i.e $\alpha = 0.4$. Therefore, according to the voting model, the logic level at faulty point is 0. On the other hand, if inputs a, b, c and d are low, e is high, then $\frac{R_d}{R_u}$ is 0.625, and $\frac{R_d}{R_u+R_d}$ is 0.615. Assume the threshold for logic one is $3V$ out of $5V$, i.e β is 0.6. Then the logic level at faulty point is 1. Thus, the same fault may exhibit either Wired-Or or Wired-And behavior depending on the input vector.

It is possible that on application of some input vectors, the pull-down strength is not strong enough to pull the faulty point to 0, and the pull-up strength is not strong enough to pull the faulty point to 1. For example, in Figure 1, if inputs a, b are low, c, d and e are high, the ratio $\frac{R_u}{R_d}$ is 1.25, and $\frac{R_u}{R_u+R_d}$ is 0.44. The value is greater than α and less than β. Therefore, we cannot model V_{BF} either as logic 1 or 0. Thus, the faulty behavior can not be modeled either as stuck-at 1 or stuck-at 0 fault. We say that the fault **exhibits neither s-a-1 behavior nor s-a-0 behavior**. The effect of the intermediate level at the faulty point on the rest of the circuit is not known. Therefore, using a somewhat pessimistic assumption, we are not going to drop such faults during diagnosis when we encounter such a situation.

While computing R_u, R_d we have to take into account the fact that transistors are non-linear devices. For example, if two nFETs, each with channel resistance r, are connected in series then the resulting pull-down network may not have a resistance of $2r$. The actual value is dependent on the technology and fabrication process. For example[17], for some processes the actual equivalent resistance is about $1.58r$.

Following the approach in [17] we will encode this information in a table, which will be used subsequently by our algorithm. We use #p(#n) to represent the number of p-transistors(n-transistors) in series, and p#(n#) to represent the number of p-transistors(n-transistors) in parallel.

Table 1(from [17]) shows such a table based on above assumption. It shows the relative strengths of various configurations of nFETs and pFETs. The **nfactors** are the results of transistor-level simulation that determine the strength of a network of nFETs relative to a single nFET. For example, two nFETs in series, denoted by 2n, will sink about 63% of the current of a single nFET. Two nFETs in parallel, denoted by n2, will sink twice as much current as a single nFET. Likewise, the **pfactors** represent the strength of a network of pFETs relative to a single pFET. Finally, the **pfactors** are converted to strengths relative to a single nFET's strength by dividing by the strength ratio, with the consideration of $\pm 10\%$ variations in fabrication process. Table 2, also from [17] then is created using the values computed in Table 1 and shows the logic levels at faulty point when different input combinations are considered. In Table 2, an L stands for a logic zero, H stands for a logic one, and N stands for a non-determined value.

We would like to point out that we are using the same model for modeling the behavior of faulty circuits in the presence of bridging faults as was done in [17]. However, our algorithm is very different from the one in [17].

3 Preliminaries

As mentioned in previous section, we are considering two-line bridging faults in combinational circuits. A *line* here is either the output of a gate or a primary input. Fanout branches are considered as a single line. Input vectors are processed one at a time.

Let T_i be the current vector being processed and A a set of lines in C. Then $A^j \subseteq A$ is the set of lines in A that are set to $j \in \{0,1\}$ by T_i. For example, in Figure 2, lines $1,2,4,5,7,9,10,11,12$ are set to one and lines $3,6,8$ are set to zero by the input vector shown. If $A = \{1,2,3,5,8\}$, then $A^0 = \{3,8\}$ and $A^1 = \{1,2,5\}$. $A^0(A^1)$ is the <u>zero(one) subset</u> of A.

A line set to 0(1) by input vector T_i is a **zero(one) line** of T_i. In subsequent discussion, the current vector will be assume to be T_i and the shorter term zero(one) line will be used instead of the longer phrase *zero(one) line of* T_i. Associated with each zero(one) line is its pull-down(pull-up) resistance. Note that each line is the output of a gate or a primary input. The pull-down and pull-up resistances of a line are the resistances of pull-down and pull-up networks of its driving gate. These pull-up and pull-down resistances depend on the number of conducting FETs in these networks. They can be readily computed using logic simulation and Tables 1, 2. We only compute the pull-up(pull-down) resistance of one(zero) lines, the pull-down(pull-up) resistance of a one(zero) line is assumed to be infinite. Primary input are considered to have infinite driving capability. Any line shorted with a primary input will be driven to the value of the primary input. If two primary inputs (with different values) are shorted, the fault exhibits s-a-0 behavior.

TSBFs are represented by Set of **Ordered Pairs of Sets**(**OPS**) in the algorithm. The collection of the OPS is referred to as an SOPS. SOPS has the form $\{< A, B >, \cdots\}$, where $< A, B >$ is an OPS. There are two forms of OPS. To represent TSBFs between lines X and Y s.t $X,\ Y \in A,\ X \neq Y$, form $< A, \Psi >$ is used. To represent TSBFs between lines X and Y s.t $X \in A,\ Y \in B,\ A \bigcap B = \Phi$, form $< A, B >$ is used. For example, if A is the set of all lines in a circuit, all TSBFs in the circuit can be represented by $< A, \Psi >$. On the other hand, an SOPS $\{< \{1, 2\}, \{3, 4\} >\}$ represents TSBFs $< 1, 3 >$, $< 1, 4 >$, $< 2, 3 >$, and $< 2, 4 >$, i.e all TSBFs between lines in sets $\{1, 2\}$ and $\{3, 4\}$. A **Fault list** is a set of TSBFs which are represented by an SOPS.

From the given test set, our diagnosis algorithm uses only those input vectors that gave a faulty response. These vectors are called **Faulty Vectors**. For each faulty vector T_i, three sets, $D_1(T_i)$, $D_0(T_i)$ and $D_c(T_i)$ are defined as follows: $D_1(T_i)$ ($D_0(T_i)$) is the set of stuck-at 1(stuck-at 0) faults which, on application of T_i, has the same output pattern as the given faulty response. $D_c(T_i)$ is the set of lines X such that $\exists$ a path from X to a primary output Y_j, and Y_j is a faulty output on application of T_i. These three sets can be considered as modified stuck-at fault dictionaries. They are computed as required during the diagnosis process. For example, in Figure

3 assume lines 8, 9 are shorted, the faulty response is $< 8 = 0, 12 = 0 >$ and $T =< 1 = 1, 2 = 1, 3 = 0, 4 = 1, 5 = 1 >$. Since the single stuck-at-0 faults at lines 4, 9, 10, 11, and 12 will result in the faulty response $< 8 = 0, 12 = 0 >$, therefore, $D_0(T) = \{4, 9, 10, 11, 12\}$. Since no stuck-at-1 fault can result in the output pattern $< 0, 0 >$, therefore, $D_1(T) = \Phi$. Since there exist paths from lines 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12 to primary output 12, and 12 is a faulty output on application of T, $D_c(T) = \{1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12\}$.

In section 4, we use the following observations to compute an initial set of residual faults and in section 5 to formulate rules for dropping faults from the fault list.

Observation 3.1: If input vector T_i detects a NFTSBF in circuit C, one of the shorted line must be a *zero line* and the other must be a *one line.*

Observation 3.2: For a NFTSBF $< X, Y >$, if T_i detects the fault, one of the following holds:

1. The shorted line with 1 value(in fault free circuit) is in D_0, and the fault exhibits stuck-at-0 behavior;
2. The shorted line with 0 value(in fault free circuit) is in D_1, and the fault exhibits stuck-at-1 behavior;
3. At least one of the shorted lines is in D_c, and the fault exhibits neither stuck-at-0 nor stuck-at-1 behavior.

4 Computing Initial Set of Residual Faults

In this section we discuss how to initialize the set of *Residual Faults*. Once again we note that we use only the set of faulty vectors. Let $T = \{T_0, \ldots, T_{N-1}\}$ be the set of *faulty* test vectors. From an initial fault list, which could be either all TSBFs, a subset of all TSBFs given by user or a fault list extracted using systems like CARAFE[13], procedure *InitialSet* computes the initial set of residual faults. It uses the sets $D_c(T_i)$, $0 \leq i \leq N - 1$.

An **Initialization Graph** $G = (V, E)$ is defined as follows:$V = \{T_i || T_i \in T\}$ and $(T_i, T_j) \in E$ iff $D_c(T_i) \bigcap D_c(T_j) \neq \Phi$. For example, from the $D_c(T_i)$ of Table 4 we get the initial graph shown in Figure 5(a).

For all line X, $Cone(X)$ is the subcircuit of the given circuit s.t. $Y \in Cone(X)$ iff $\exists$ a path from Y to X. For all line X, $Reach(X)$ is the subcircuit of the given circuit s.t. $Y \in Reach(X)$ iff $\exists$ a path from X to Y. In Figure 3, $Cone(8) = \{1, 2, 3, 4, 6, 7\}$, $Reach(9) = \{11, 12\}$.

We first give the algorithm for procedure *InitialSet*. Next we illustrate it by an example. Finally, we establish the correctness of the algorithm. Let I_{FTSBF} and I_{NFTSBF} denote the initial set of residual FTSBFs and NFTSBFs respectively.

Procedure InitialSet

/* Let A be the set of all lines in C; I be the initial fault list that may have been extracted either from the layout or by some other means; T be the test set. It is assumed that for all $T_i \in T$, $D_c(T_i)$ has been computed. */

Compute the Initial Graph $G = (V, E)$*;*

Compute the connected components of G*;*

/* G can have at most two components. See proof below. */

Case 1: G has two connected components $G1$, $G2$.

$S_1 \longleftarrow \bigcap_{T_i \in G1} D_c(T_i)$;

$S_2 \longleftarrow \bigcap_{T_j \in G2} D_c(T_j)$;

$I_{NFTSBF} \longleftarrow \{< S_1, S_2 >\}$;

$I_{FTSBF} \longleftarrow \{< \{X\}, Reach(X) \bigcap S_2 >, < Cone(X) \bigcap S_2, \{X\} > || X \in S_1\}$;

Case 2: G has only one connected component.

Case 2.1: G is not a complete graph.

Partition G into two subgraph G' *and* G'' *by deleting some vertices using Procedure Partition discussed below.*

$S_1 \longleftarrow \bigcap_{T_i \in G'} D_c(T_i)$;

$S_2 \longleftarrow \bigcap_{T_j \in G''} D_c(T_j)$;

$I_{NFTSBF} \longleftarrow \{< S_1, S_2 >\}$;

$I_{FTSBF} \longleftarrow \{< \{X\}, Reach(X) \bigcap S_2 >, < Cone(X) \bigcap S_2, \{X\} > || X \in S_1\}$;

Case 2.2: G is complete graph.

Let j *be such that* $\forall i,\ T_i \in T, ||D_c(T_j)|| \leq ||D_c(T_i)||$

$I_{NFTSBF} \longleftarrow \{< A, D_c(T_j) >\}$;

$I_{FTSBF} \longleftarrow \{< \{X\}, Reach(X) >, < Cone(X), \{X\} > || X \in D_c(T_j)\}$;
$I_{NFTSBF} \longleftarrow I \bigcap I_{NFTSBF}$;
$I_{FTSBF} \longleftarrow I \bigcap I_{FTSBF}$;
End of Procedure

For example, in Figure 4, let i and i' be the two shorted lines. The test vectors are shown in Table 3, the corresponding response, both good and faulty, and the set $D_c(T_i)$ for each input vectors are shown in Table 4.

Let's first examine the case where the test set consists of the first four vectors of Table 3, i.e. $T = \{T_0, T_1, T_2, T_3\}$. The initial graph $G = (V, E)$ is shown in Figure 5(b). We can see that there are two disconnected subgraphs $G_1 = (V_1, E_1) = (\{T_0, T_2\}, \{(T_0, T_2)\})$ and $G_2 = (V_2, E_2) = (\{T_1, T_3\}, \{(T_1, T_3)\})$. According to the algorithm(Case 1), the initial set of NFTSBF residual faults $I_{NFTSBF} = \{< \bigcap_{T_i \in G_1} D_c(T_i), \bigcap_{T_j \in G_2} D_c(T_j) >=< \{i', d', c'\}, \{i, d, c\} >\}$, and the initial set of FTSBF residual faults $I_{FTSBF} = \{< \{i'\}, \Phi >, < \{d'\}, \Phi >, < \{c'\}, \Phi >\} = \Phi$. Note the shorted lines i and i' are in the list, and the fault is not a feed back fault.

Next consider the case with all five test vectors, i.e. $T = \{T_0, T_1, T_2, T_3, T_4\}$. The initial graph $G(V, E)$ is shown in Figure 5(a). It has only one component, but is not a complete graph. Case 2.1 of the algorithm is used. If we remove vertex T_4 (as determined by Procedure Partition below) and all the edges connected to it, we have a new graph, which is same as the one in previous case. According to the algorithm, we will have the same initial set of residual faults of NFTSBFs and FTSBFs as in the previous case.

Since the size of two intersected sets is always less or equal to both sets, we note that initial graph with two components can always yield smaller initial set of residual faults. Interestingly enough, if the initial graph has only one component and is not a complete graph, we can always take away some vertices and make it a graph with two components. We will prove that if the following partitioning procedure is used then the resulting initial set of faults obtained by procedure *InitialSet* will contain the Object fault.

Procedure Partition

/*Given an initial graph $G(V, E)$ s.t it has one component and is not complete, we partition it into two disconnected subgraphs $G'(V', E')$ and $G''(V'', E'')$ */

Find two vertices T_i, T_j which are not adjacent; /* Can always be found, since G is not complete */

$G'(V', E'), V' = \{$*all vertices which are adjacent to T_i(including T_i) but not adjacent to T_j*$\}$, $E' = \{(T_i, T_k) \in E,\ T_i, T_k \in V'\}$;

$G''(V'', E''), V'' = \{$*all vertices which are adjacent to T_j(including T_j) but not adjacent to T_i*$\}$, $E'' = \{(T_j, T_t) \in E,\ T_j, T_t \in V''\}$;

End of Procedure

The correctness of the algorithm is given by following theorem. For the proof of the theorem, please refer to Appendix I.

Theorem 4.1: If the initial fault list I contains the *Object Fault*, the initial set of residual TSBFs computed by Procedure *InitialSet* contains the *Object Fault.*

5 Diagnosing TSBFs

In fault diagnosis a set of possible faults, which must contain the object fault, is computed. We start with an initial set of residual fault as the possible set of faults. Next, by processing vectors one at a time faults, which are determined not to be equivalent to the object fault, are dropped. The stuck-at fault dictionary $D_0(T_i)$, $D_1(T_i)$ and $D_c(T_i)$ of input vector T_i and a set of rules, derived from Observations 3.1, 3.2, are used for dropping fault. In addition, to save space we do not explicitly list all the faults but use the implicit representation SOPS. FTSBFs and NFTSBFs are processed separately. In the next three subsections we discuss how to drop NFTSBFs and FTSBFs from the fault list.

5.1 Diagnosing NFTSBFs

The algorithm starts from an initial set of residual fault, which could be all TSBFs in circuit C, or the initial fault list I_{NFTSBF} computed by *Procedure InitialSet*, or any fault list containing the object fault. The initial fault list is represented by an SOPS. We first present the algorithm, and then explain it with the help of an example.

Procedure Diag_NFTSBF

/* Given an initial fault list F, represented by an SOPS; the faulty test set T, the circuit C and the observed responses R. The algorithm returns the residual faults containing the object fault if it is a NFTSBF. */

For each faulty input vector $T_i \in T$

Begin

For each OPS $< A, B > \in F$

Begin

$F' \longleftarrow \Phi$;

Step 0

$D_c(T_i)$, $D_0(T_i)$ *and* $D_1(T_i)$ *are computed using Logic Simulation;*

Step 1

/* From Observation 3.1, fault must be between a zero line and a one line. Drop other faults. */

Partition A *into* A^0, A^1;

If $B \neq \Psi$, *partition* B *into* B^0, B^1

Else $B^0 \longleftarrow \Phi$, $B^1 \longleftarrow A^1$;

$G_1 = \{< A^0, B^1 >, < B^0, A^1 >\}$;

Step 2

/* From Observation 3.2, the object fault will exhibit either s-a-0 behavior, or s-a-1 behavior or neither s-a-0 nor s-a-1 behavior. We compute the subsets of G_1, that could potentially exhibit s-a-0, s-a-1, neither s-a-0 nor s-a-1 behaviors. */

/* Faults potentially exhibiting s-a-0 behavior. */

$\alpha \longleftarrow \{< A^0, B^1 \bigcap D_0(T_i) >, < B^0, A^1 \bigcap D_0(T_i) >\}$;

/* Faults potentially exhibiting s-a-1 behavior. */

$\beta \longleftarrow \{< A^0 \bigcap D_1(T_i), B^1 >, < B^0 \bigcap D_1(T_i), A^1 >\}$;

/* Faults potentially exhibiting neither s-a-0 nor s-a-1 behavior. */

$\gamma \longleftarrow \{< A^0, B^1 >, < B^0, A^1 >\}$;

Step 3

/* Based on the voting model, from each sublist α, β, γ of faults, faults not exhibiting the desired behavior are dropped. */

/* If the object fault exhibits s-a-0, it should satisfy inequality (2)(section 2). If fault does not satisfy (2) then it is dropped from α. */

$\alpha' \longleftarrow reduce(\alpha, sa0)$;

/* If the object fault exhibits s-a-1, it should satisfy inequality (1)(section 2). If fault does not satisfy (1) then it is dropped from β. */

$\beta' \longleftarrow reduce(\beta, sa1)$;

/* If the object fault exhibits neither s-a-0 nor s-a-1, it should not satisfy either inequality (1) or (2) (section 2). If fault does satisfy (1) or (2) then it is dropped from γ. */

$\gamma' \longleftarrow reduce(\gamma, san0n1)$;

$F' \longleftarrow F' \bigcup \alpha' \bigcup \beta' \bigcup \gamma'$;

End

$F \longleftarrow F'$;

End

End of Procedure

In step 0 of Procedure Diag_NFTSBF, $D_c(T_i)$, $D_0(T_i)$ and $D_1(T_i)$ are computed using a modified fault simulation algorithm[7]. The stuck-at fault simulation algorithm we modified is given in [7]. Next in step 1, since we are only considering faulty input vectors, for each OPS=$< A, B > \in F$ the fault must be between a zero line and a one line. Thus, after dropping faults between one lines and faults between zero lines, the faults represented by $< A, B >$ reduces to the set of faults given by $\{< A^0, B^1 >, < B^0, A^1 >\}$ (Observation 3.1).

We can reduce the fault list further. Let's consider the OPS $< A^0, B^1 >$. Based on the behavior of fault, there are three possible outcomes(Observation 3.2). (i) If the fault behaves as a stuck-at-1 fault, i.e. the logic level at faulty point goes to 1, the OPS $< A^0, B^1 >$ could be replaced by $< A^0 \bigcap D_1(T_i), B^1 >$. (ii) If the fault behaves as a stuck-at-0 fault, i.e. the logic level at faulty point goes to 0, the OPS $< A^0, B^1 >$ could be replaced by $< A^0, B^1 \bigcap D_0(T_i) >$. (iii) If the faulty point goes to an intermediate level then the OPS $< A^0, B^1 >$ remain unchanged. Similarly we can replace OPS $< B^0, A^1 >$, with respect to (i), (ii) and (iii), by $< B^0 \bigcap D_1(T_i), A^1 >$, $< B^0, A^1 \bigcap D_0(T_i) >$, and $< B^0, A^1 >$. We call $\{< A^0, B^1 \bigcap D_0(T_i) >, < B^0, A^1 \bigcap D_0(T_i) >\}$ the α list, $\{< A^0 \bigcap D_1(T_i), B^1 >, < B^0 \bigcap D_1(T_i), A^1 >\}$ the β list, and $\{< A^0, B^1 >, < B^0, A^1 >\}$ the γ list.

Since at this point we don't know what the fault is, i.e. which value the faulty point takes, all three cases are possible. We will retain all three lists α, β, and γ. At this point, α, β and γ are not disjoint. In particular, α, β are subsets of γ. But after next step, these three lists will be disjoint.

In step 3, we further reduce the fault list by dropping more faults from α, β, γ lists. Consider the list $\alpha = \{< A^0, B^1 \bigcap D_0(T_i) >, < B^0, A^1 \bigcap D_0(T_i) >\}$. While retaining these faults, we had tacitly assumed that all faults of the form $< X, Y > \in \alpha$ exhibit stuck-at 0 behavior on application of T_i. But this is not necessarily true for all $< X, Y >$. We can therefore delete such faults $< X, Y >$ from the α list that cannot exhibit stuck-at 0 behavior. This is done by examining the ratio of pull-down and pull-up resistance of each fault $< X, Y >$ in the fault list α. Using *Procedure Reduce* faults that cannot exhibit s-a-0 behavior are dropped from the α list. *Procedure Reduce* is also used for dropping faults from $\beta(\gamma)$ list that do not exhibit s-a-1(neither s-a-1 nor s-a-0) behavior.

Assume *list* to be the fault list we want to reduce, *behavior* is the potential behavior the faults in the *list* exhibit. The basic idea in *Procedure Reduce* is to examine each fault in the *list*. If the fault exhibits the *behavior*, it is retained, otherwise it is dropped from the *list*. The algorithm is presented below and a detailed discussion follows.

Procedure Reduce(list, behavior)

/* Let *list* be the list to be processed, *behavior* be the desired behavior we want to extract; $< B^0, B^1 >$ the OPS in list that is being processed; $B_j^0(B_i^1)$ be a maximal subset of $B^0(B^1)$ with pull-down(up) resistance $R_j^d(R_i^u)$; f be the extracted fault list derived from *list* after dropping faults. Assume $V_0(V_1)$ to be the threshold of logic zero(one), V_{dd} to be the power source.*/

For each OPS $< B^0, B^1 > \in$ *list*

Begin

Step 1:

/* By sorting, group lines with same pull-down resistance together. */

$B^0 \longleftarrow$ *Sort(*B^0*, Key:pull-down resistances);*

/* By sorting, group lines with same pull-up resistance together. */

$B^1 \longleftarrow$ *Sort(*B^1*, key:pull-up resistances);*

Step 2:

For each maximal subset $B_j^0 \subseteq B^0$

/* all lines in B_j^0 have resistance R_j^d and nothing else in B^0 have the same resistance. */

Begin

<u>*Case 1:*</u> *The desired behavior "behavior" is s-a-0*

$\forall B_i^1,\ B_i^1 \subseteq B^1,\ \frac{R_j^d}{R_i^u + R_j^d} \leq \frac{V_0}{V_{dd}}$

/* the resistance of lines in B_i^1 is R_i^u, if the relation between pull-down and pull-up resistance satisfy (2)(section 2), the fault exhibits s-a-0 behavior. */

$tmp \longleftarrow < B_j^0, \bigcup_{\forall i} B_i^1 >;$

<u>*Case 2:*</u> *The desired behavior "behavior" is s-a-1*

$\forall B_i^1,\ B_i^1 \subseteq B^1,\ \frac{R_j^d}{R_i^u + R_j^d} \geq \frac{V_1}{V_{dd}}$

/* the resistance of lines in B_i^1 is R_i^u, if the relation between pull-down and pull-up resistance satisfy (1)(section 2), the fault exhibits s-a-1 behavior. */

$tmp \longleftarrow < B_j^0, \bigcup_{\forall i} B_i^1 >;$

<u>*Case 3:*</u> *The desired behavior "behavior" is neither s-a-0 nor s-a-1*

$\forall B_i^1,\ B_i^1 \subseteq B^1,\ \frac{V_0}{V_{dd}} < \frac{R_j^d}{R_i^u + R_j^d} < \frac{V_1}{V_{dd}}$

/* the resistance of lines in B_i^1 is R_i^u, if the relation between pull-down and pull-up resistance satisfy neither (1) nor (2)(section 2), the fault exhibits neither s-a-1 nor s-a-0 behavior. */

$tmp \longleftarrow < B_j^0, \bigcup_{\forall i} B_i^1 >;$

$f \longleftarrow f \bigcup tmp;$

End

End

Return f;

End of Procedure

In *Procedure Reduce*, we first use sort to group all lines with same pull-down resistance for zero line(pull-up resistance for one lines) together. Then for each maximal subset of B_j^0 of same resistance in B^0, find all maximal subsets of B_j^1 of same resistance in B^1 s.t. fault $< X, Y >$, $X \in B_j^0$, $Y \in B_i^1$ exhibits the desired behavior. All such B_i^1 are merged together to form a new OPS with B_j^0 and appended to the return fault list.

Similarly, we use *Procedure Reduce* to extract faults which exhibit s-a-1 behavior, and faults which exhibit neither s-a-1 nor s-a-0 behavior from β and γ list, respectively. The fault list is reduced to $F = \alpha' \bigcup \beta' \bigcup \gamma'$. At this point the three new list α', β', and γ' are disjoint. Since a faulty point could only either go to 1 or 0 or an intermediate level, all possible faults are covered by α', β' and γ' lists.

In summary, by applying each faulty input vector one at a time to the circuit, we use three steps to reduce the existing fault list to a smaller one. After all the input vectors have been processed the fault list we are left with constitutes the set of Residual Faults.

5.2 Example for Diag_NFTSBF

We next illustrate *Procedure Diag_NFTSBF* using Figure 4. Assume Table 3 to be the input vectors; i and i' to be the two shorted lines(the object fault). For each input vector T_i, $D_c(T_i)$ and both good and faulty responses are shown in Table 4; the set $D_0(T_i)$ and $D_1(T_i)$ are shown in Table 5. $F = \{< A, \Psi >\} = \{< \{a - m, a' - f', h' - l'\}, \Psi >\}$ is the initial fault list. This is not the initial list computed by *Procedure InitialSet* but is assumed for our example. Logic simulation of T_0 implies that lines a, b, c, d, g, j, k, a', b', i', j' are set to zero and lines e, f, h, i, l, m, c', d', e', f', h', k', l' are set to one. Therefore $A^0 = \{a, b, c, d, g, j, k, a', b', i', j'\}$ and $A^1 = \{e, f, h, i, l, m, c', d', e', f', h', k', l'\}$. The α, β and γ lists are:

$$\begin{aligned}
\alpha &= \{< A^0, A^1 \bigcap D_0(T_0) >\} \\
&= \{< \{a, b, c, d, g, j, k, a', b', i', j'\}, \{e, f, h, i, l, m, c', d', e', f', h', k', l'\} \bigcap \{k', d', c'\} >\} \\
&= \{< \{a, b, c, d, g, j, k, a', b', i', j'\}, \{k', d', c'\} >\} \\
\beta &= \{< A^0 \bigcap D_1(T_0), A^1 >\} \\
&= \{a, b, c, d, g, j, k, a', b', i', j'\} \bigcap \{i'\}, \{e, f, h, i, l, m, c', d', e', f', h', k', l'\} >\} \\
&= \{< \{i'\}, \{e, f, h, i, l, m, c', d', e', f', h', k', l'\} >\} \\
\gamma &= \{< A^0, A^1 >\} \\
&= \{< \{a, b, c, d, g, j, k, a', b', i', j'\}, \{e, f, h, i, l, m, c', d', e', f', h', k', l'\} >\}
\end{aligned}$$

Next we use *Procedure Reduce* to drop faults from the three lists α, β, γ. Take list α as an example. While retaining these faults, we had tacitly assumed that all faults in the list exhibit s-a-0 behavior. But this is not true for all the faults in α. Using *Procedure Reduce*, we first sort the zero set $B^0 = \{a, b, c, d, g, j, k, a', b', i', j'\}$ to $B^0 = \{\underbrace{a, b, c, d, g, a', b'}_{0}, \underbrace{j, k, i', j'}_{2n}\}$, and the one set $B^1 = \{k', d', c'\}$ to $B^1 = \{\underbrace{d', c'}_{0}, \underbrace{k'}_{p}\}$. The two maximal subsets of zero set are $B_1^0 = \{a, b, c, d, g, a', b'\}$ — all primary inputs with the pull-down resistance equivalent to zero[2], and $B_2^0 = \{j, k, i', j'\}$ — same pull-down resistance equivalent to two conducting nFET transistors. The two maximal subsets of one set are $B_1^1 = \{d', c'\}$ — with pull-up resistance equivalent to zero, and $B_2^1 = \{k'\}$ — with pull-up resistance equivalent to one conducting pFET. From Table 2, we see that only faults in $< B_1^0, B_2^1 >$ and faults in $< B_2^0, B_2^1 >$ exhibit s-a-0 behavior. Therefore α list can be reduced to $\alpha' = \{< \{a, b, c, d, g, j, k, a', b', i', j'\}, \{k'\} > \}$. Note that we have dropped 22 faults out of 33 from the α list. In a similar way, we can reduce β and γ list to $\beta' = \{< \{i'\}, \{h, i, e, f, c', d', m, h', e', f', l'\} > \}$ and $\gamma' = \{< \{c', d'\}, \{a, b, c, d, g, a', b'\} >\}$ list. The fault list F is now $\{< \{a, b, c, d, g, j, k, a', b', i', j'\}, \{k'\} >, < \{i'\}, \{h, i, e, f, c', d', m, h', e', f', l'\} >, < \{c', d'\},$

[2]Note that we are assuming that the primary inputs have a strong pulldown(if set to 0) and a strong pullup(if set to 1) capability.

$\{a, b, c, d, g, a', b'\} >\}$. After all input vectors in Table 3 are processed, the residual fault list is $\{< \{c, d\}, \{c', d'\} >, < \{i\}, \{i'\} >\}$.

The correctness of the algorithm follows from the following theorem. For the proof of this theorem please refer to Appendix B.

Theorem 5.1: If the object fault is a NFTSBF and the initial fault list contains the object fault then, the residual fault list computed by procedure *Diag_NFTSBF* also contains the object fault.

5.3 Diagnosing FTSBFs

The diagnosis of FTSBF is considerably more difficult because feedback loops can turn the circuit into a sequential circuit. To understand the diagnosis algorithm we need to have a closer look at conditions that an input vector must satisfy to detect an FTSBF.

Consider the FTSBF $< X, Y >$, s.t. a path from X to Y exists. Let $T_i(X)$ be the value at X on application of T_i to the fault free circuit. Then, excluding oscillations, if an input vector T_i detects the fault one of the following conditions must be satisfied[3].

1. $T_i(X) \neq T_i(Y)$ and the fault exhibits Y s-a-$T_i(X)$ behavior on application of T_i.

2. $T_i(X) \neq T_i(Y)$ and the fault exhibits X s-a-$T_i(Y)$ behavior on application of T_i.

3. $T_i(X) \neq T_i(Y)$ and the logic value at the faulty location cannot be determined either to be 1 or 0 on application of T_i.

4. $T_i(X) = T_i(Y)$. In this case, T_i may still detect the fault because the circuit may memorize a faulty value from an earlier vector.

In general, rules developed for diagnosing NFTSBFs can not be used for diagnosing FTSBFs. However, a closer examination of the above conditions suggests that the first three faulty behavior is similar to a NFTSBF. We call them **TypeI** faulty behavior. The fourth behavior is a **TypeII** faulty behavior.

It is not difficult to see that if for any input vector T_i the object fault exhibits TypeI behavior, while processing T_i the rules used for diagnosing NFTSBFs will include it in the residual fault list. But since the rules used for NFTSBFs assume that the two lines involved have different fault free values, they will discard the object fault if it exhibits TypeII behavior on application of T_i. In order to include object fault which may exhibit TypeII behavior, we modify our algorithm for NFTSBF and retain all faults which have same fault free value in our fault list. Thus, some FTSBF which are not equivalent to the object fault may be retained in the Residual Set of Faults. In order to eliminate such faults we perform detailed simulation.

The above discussion leads us to the following algorithm for diagnosing FTSBF. This is a two-phase algorithm. Phase I of the algorithm is similar to *Procedure Diag_NFTSBF* except a patch is added to retain faults exhibiting TypeII behavior. Since we do not know the behavior of object fault, both behaviors should be considered for each fault in fault list. In Phase II we enumerate FTSBFs one at a time and eliminate some of these using detailed simulation.

Procedure Diag_FTSBF

/* Given an initial fault list F, in the form of SOPS; the faulty test set $T = \{T_0, T_1, \cdots, T_{N-1}\}$ and the corresponding faulty responses $R = \{R_0, R_1, \cdots, R_{N-1}\}$. The algorithm returns the residual FTSBFs list. The initial fault list is I_{FTSBF} computed by procedure *InitialSet*. Note that each OPS in I_{FTSBF} has either the form $< \{X\}, A >$ or $< B, \{X\} >$, where X is a line in C and $A(B)$ is a subset of $Reach(X)(Cone(X))$. */

Phase I

For each faulty input vector $T_i \in T$

Begin

For each OPS $< A, B > \in F$

Begin

$F' \longleftarrow \Phi$;

Step 0

Logic Simulation;

Step 1

Partition A into A^0, A^1*;*

If $B \neq \Psi$, *partition B into* B^0, B^1

Else $B^0 \longleftarrow \Phi$, $B^1 \longleftarrow A^1$*;*

Step 2

$\alpha \longleftarrow \{< A^0, B^1 \bigcap D_0(T_i) >, < B^0, A^1 \bigcap D_0(T_i) >\}$;

$\beta \longleftarrow \{< A^0 \bigcap D_1(T_i), B^1 >, < B^0 \bigcap D_1(T_i), A^1 >\}$;

$\gamma \longleftarrow \{< A^0, B^1 >, < B^0, A^1 >\}$

Step 3

$\alpha' \longleftarrow reduce(\alpha, sa0)$;

$\beta' \longleftarrow reduce(\beta, sa1)$;

$\gamma' \longleftarrow reduce(\gamma, san0n1)$;

Step 4

/ Patch for accommodating type II faults */*

If $< A, B > = < \{X\}, B >$ *and* $T_i(X) = 0$ *then*

$f \longleftarrow \{< \{X\}, B^0 >\}$

Else If $< A, B > = < \{X\}, B >$ *and* $T_i(X) = 1$ *then*

$f \longleftarrow \{< \{X\}, B^1 >\}$

Else If $< A, B > = < A, \{X\} >$ *and* $T_i(X) = 0$ *then*

$f \longleftarrow \{< A^0, \{X\} >\}$

Else If $< A, B > = < A, \{X\} >$ *and* $T_i(X) = 1$ *then*

$f \longleftarrow \{< A^1, \{X\} >\}$

$F' \longleftarrow F' \bigcup \alpha' \bigcup \beta' \bigcup \gamma' \bigcup f$;

End

$F \longleftarrow F'$;

End

<u>Phase II</u>

/* Initially, all inputs = 0 the other lines have the corresponding steady state value. */

for each fault $f \in F$

Begin

for each input vector $T_i \in T$

Begin

Fault simulation with fault f present;

If the response does not equal R_i

Drop f from fault list;

Goto next fault;

End

End

End of Procedure

Theorem 5.2: If the object fault is a FTSBF, and I_{FTSBF} contains the object fault then the residual fault list computed by procedure *Diag_FTSBF* also contains the object fault.

6 Experimental Results

In this section we present our experimental results obtained by running the proposed algorithm on the ISCAS85[5] combinational, and ISCAS89 benchmark circuits on a SUN 4/670 with 64M memory. The outline of our implementation is as follows.

1. Use **Diag_NFTSBF** to diagnose NFTSBF. The initial fault list F is set to all the TSBFs in the circuit.

2. While diagnosing NFTSBF using **Diag_NFTSBF**, all $D_c(T_i)$s for each input vector T_i are computed and stored. They are used in procedure InitialSet.

3. Use **InitialSet** to compute initial set of residual FTSBFs from all TSBFs in the circuit.

4. Use **Diag_FTSBF** to diagnose FTSBF from the initial set of residual FTSBFs.

In our implementation, we first diagnose NFTSBF from all TSBFs. Concurrently, we compute and store all the $D_c(T_i)$s. This is used to compute initial set of residual FTSBFs. Then, from the initial set of residual FTSBF list, *Diag_FTSBF* is used to diagnose FTSBF.

For each circuit we generated 25 instances of the diagnosis problem as follows. Random number generator were used to generate the numbers X, Y to get the TSBF $< X, Y >$.The test set is obtained from the test generator "atpg"[14] in misII.

The results are shown in Table 6, 7, 8, 9 and 10. In Table these tables, **Vectors** is the number of input vectors used (**T** – total number of input vectors, **F** – number of faulty input vectors); **Total** is the total number of TSBFs in the circuit, and **Residual** is the average number of residual faults for NFTSBF(**N**) and FTSBF(**F**) faults. **Time** is the average CPU seconds required by the diagnosis algorithm. The value under **N**(**F**) is the time required by Diag_NFTSBF(Diag_FTSBF). All measurements were done on a SUN 4/670 with 64M memory. Averages are computed for the 25 instances. **Residual(Low)** is the smallest residual fault list among 25 faults and **Residual(High)** is the largest residual fault list among 25 faults. Note the large variation between the average and the highest size of the set of residual faults. We will come back to this issue later on.

Table 9 and 10 show the results on some of ISCAS89 sequential circuits. All these circuits are converted to combinational circuits, where all fanins to D flip-flops are considered as primary outputs, and all fanouts from D flip-flops are considered as primary inputs.

Although the final size of the set of residual faults is a small fraction of the total number of faults it is not as small as we had hoped for. We suspected that the stuck-at test set is not an effective diagnostic test set for TSBFs. We therefore carried out a second set of experiments. In this we used randomly generated test sets. The size of the random test set used was same as the size of the stuck-at test set. The results are tabulated in Table 7. We see a dramatic improvement, for all but C1908, C1355, in the final size of the set of residual faults. The running time also improved for these cases. The most noticeable improvement was for C5315. From this we can only conclude that the final result depends on the diagnostic test set used and random test sets are at least as good, if not better, than stuck-at test sets.

There is a large variation in the largest size of the residual faults and the average value. In addition the residual faults could not be reduced for some circuits like 1908. This, we believe, is not so much a function of the effectiveness of the algorithms as much as it is a function of the test set used, the circuit involved and the particular bridging fault assumed. Consequently, for similar data all other diagnosis algorithms will also exhibit similar behavior. To support our intuition on this we present some detailed results for 1908. Table 11 shows the relationship between the number of faulty input vectors and the number of residual NFTSBFs for some of the shorts in C1908, when stuck-at test set was used. Note that faults that have huge number of residual NFTSBFs usually have one or two faulty input vectors. For example, for fault number 8 only one of the input vectors detected it resulting in only one faulty vector. This is consistent with the fact that this circuit is known to be difficult to test because of a large number of hard to detect stuck-at faults.

The total time required is quite reasonable. However, FTSBF time comprised a disproportionate amount of the total time. This was because of the detailed simulation done in Phase II of Diag_FTSBF. By eliminating Phase II we get the results shown in Table 8. There is a considerable reduction in the total time. However, the number of residual FTSBF faults increases quite a bit. So, we propose that if one can afford the additional CPU time Phase II of Diag_FTSBF should be used.

7 Discussion

We studied the diagnosis problem for bridging faults. The voting model was used to model the behavior of the faulty circuit in the presence of bridging faults. An Algorithm for this problem was discussed. The experimental results obtained are encouraging.

In some recent papers[15, 19] some weaknesses of the voting models were pointed out. They are: the resistance of the FET channel is a function of the voltage between the source and drain; and the voltage level at the faulty point could turn on some nFET (pFET) while it may not turn on other nFETs (pFET). The first is a consequence of the well know Vds-Ids characteristic of FETs while the second one is a consequence of the variations in threshold voltage of the different FETs. The conductance table we used[16] are correct for Vds near 2.5V[15].

Our proposed algorithm can be modified by making it somewhat more pessimistic during Phase I. The values of α, β can be computed taking into account the variation in the threshold voltages. In addition, we can precompute the range of FET conductance for each FET and take their upper bounds.

In phase II we perform detailed simulation for FTSBFs using the voting model. Now we need to perform detailed simulation using the Biased-Voting model for the residual NFTSBFs and FTSBFs.

For lack of realistic cell library data we are unable to measure the effect that these modifications will have on the proposed algorithm.

In closing we would like to point out that so far as the FETs are driven past the "knee" of the Vds-Ids curve and the threshold voltage of the FETs do not vary too much the voting model can still be used.

References

[1] M. Abramovici. A Hierarchical, Path-Oriented Approach to Fault Diagnosis in Modular Combinational Circuits. *IEEE Trans. on Computers*, C-31(7):672–677, July 1982.

[2] M. Abramovici and M. A. Breuer. Multiple Fault Diagnosis of MOS Combinational Networks. *IEEE Trans. on Computers*, C-29(6):451–460, June 1980.

[3] M. Abramovici, M. A. Breuer, and A. D. Friedman. *DIGITAL SYSTEMS TESTING AND TESTABLE DESIGN*. Computer Science Press, 1990.

[4] John M. Acken. Deriving accurate fault models. In *CSL-TR-88-365, Computer System Laboratory, Standford University, Oct.*, 1988.

[5] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Special session on ATPG and Fault simulation, IEEE Int'l. Symposium on Circuits and Systems*, 1985.

[6] D. J. Burns. Locating High Resistance Shorts in CMOS Circuits by Analyzing Supply Current Measurement Vectors. *Int'l Symposium for Testing and Failure Analysis*, pages 231–237, 1989.

[7] S. Chakravarty and Y. Gong. An Algorithm for Diagnosing Two Line Bridging Fault in Combinational Circuits. In *Design Automation Conference*, 1993.

[8] S. Chakravarty and M. Liu. I_{DDQ} Measurement Based Diagnosis of Bridging Faults. In *Journal of Electronic Testing: Theory and Application (Special Issue on I_{DDQ} Testing)*. Kluwer Academic Publishers, 1993.

[9] H. Cox and J. Rajski. A Method of Fault Analysis for Test Generation and Fault Diagnosis. *IEEE Trans. on Computer Aided Design*, 7(7):813–833, July 1988.

[10] Y. M. El-Ziq and S. Y. H. Su. Fault Diagnosis of MOS Combinational Networks. *IEEE Trans. on Computers*, C-31(2), February 1982.

[11] J. Ferguson and J. P. Shen. A CMOS fault Extractor for Inductive Fault Analysis. *IEEE Trans. on Computer Aided Design*, (11), 1988.

[12] J. Ferguson and J. P. Shen. Extraction and Simulation of Realistic Faults in CMOS Circuits. In *Proceedings of the International Test Conference*, 1988.

[13] A. Jee and J. Ferguson. Carafe: An Inductive Fault Analysis Tool for CMOS VLSI Circuits. In *IEEE Int'l Test Symposium*, 1993.

[14] T. Larabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Trans. on Computer Aided Design*, 11(1):4–15, January 1992.

[15] P. Maxwell and R. Aitken. Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Threshold. In *Proceedings of the International Test Conference*, pages 63–72, 1993.

[16] E. J. McCluskey. Test and Diagnosis Procedure for Digital Networks. *Computer*, 4(1):17–20, January 1971.

[17] Steven D. Millman and Sr. James P. Garvey. An Accurate Bridging Fault Test Pattern Generator. In *Proceedings of the International Test Conference*, pages 411–418, 1991.

[18] Steven D. Millman, Edward J. McCluskey, and John M. Acken. Diagnosing CMOS Bridging Faults with Stuck-at Fault Dictionaries. In *Proceedings of the International Test Conference*, pages 860–870, 1990.

[19] J. Rearick and J. Patel. Fast and Accurate Bridging Fault Simulation. In *Proceedings of the International Test Conference*, pages 54–62, 1993.

[20] J. Richman and K. Bowden. The Modern Fault Dictionary. In *Proc. International Test Conference*, pages 696–702, 1985.

[21] Maly W, J. P. Shen, and F. J. Ferguson. Systematic Characterization of Physical Defects for Fault Analysis of MOS IC Cells. In *Proceedings of the International Test Conference*, 1984.

[22] J. Waicukauski, J. V. Gupta, and S. Patel. Diagnosis of BIST failures by PPSFP Simulation. In *Proc. International Test Conference*, pages 480–484, 1987.

[23] J. Waicukauski and E. Lindbloom. Failure Diagnosis of Structured VLSI. *IEEE Trans. on Design & test of Computers*, 6(4):49–60, August 1989.

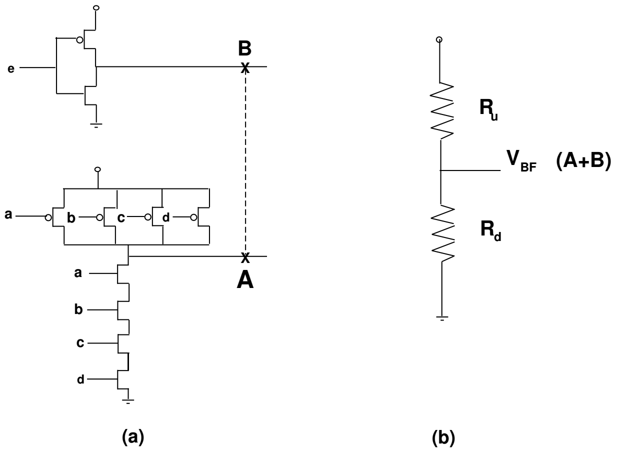

Figure 1: An Example Circuit.

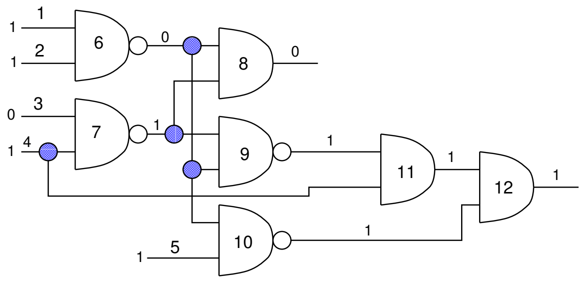

Figure 2: An Example Circuit.

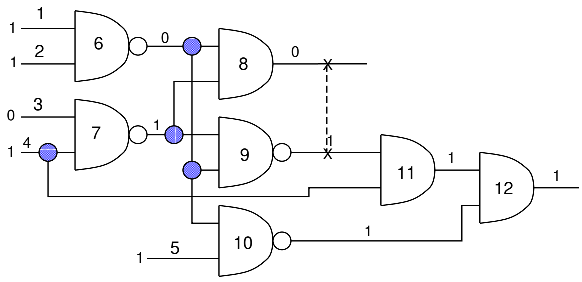

Figure 3: An Example Circuit.

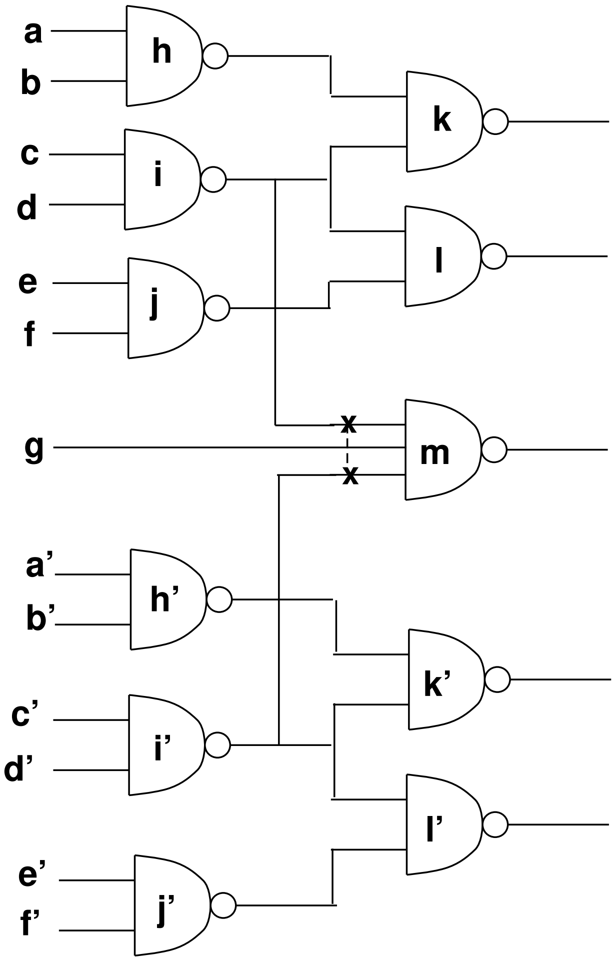

Figure 4: Initializtion Example

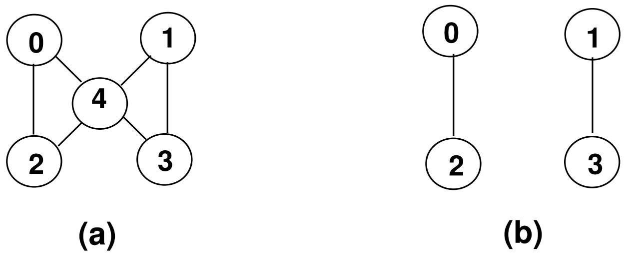

Figure 5: Initial Graph: (a) $T = \{T_0, T_1, T_2, T_3, T_4\}$, (b) $T = \{T_0, T_1, T_2, T_3\}$

	max ratio	min ratio	nfactors		pfactors	
4p	0.11	0.14	4n	0.36	4p	0.30
3p	0.14	0.18	3n	0.46	3p	0.39
2p	0.20	0.25	2n	0.63	2p	0.56
p	0.36	0.45	n	1.00	p	1.00
p2	0.72	0.91	n2	2.00	p2	2.01
p3	1.07	1.37	n3	3.00	p3	3.01
p4	1.43	1.82	n4	4.00	p4	4.01

Table 1: Strength Analysis Table

p/n	4n	3n	2n	n	n2	n3	n4
4p	L	L	L	L	L	L	L
3p	L	L	L	L	L	L	L
2p	L	L	L	L	L	L	L
p	N	L	L	L	L	L	L
p2	H	H	H	L	L	L	L
p3	H	H	H	H	L	L	L
p4	H	H	H	H	L	L	L

Table 2: Bridging Fault Result Analysis

	a	b	c	d	e	f	g	a'	b'	c'	d'	e'	f'
T_0	0	0	0	0	1	1	0	0	0	1	1	1	1
T_1	0	0	1	1	1	1	0	0	0	0	0	1	1
T_2	1	1	0	0	0	0	0	1	1	1	1	0	0
T_3	1	1	1	1	0	0	0	1	1	0	0	0	0
T_4	0	0	0	0	1	1	1	0	0	1	1	1	1

Table 3: Test Set

	Good Res.					Faulty Res.					$D_c(T_i)$
	k	l	k'	l'	m	k	l	k'	l'	m	
T_0	0	1	1	1	1	0	1	0	1	1	$\{k', i', h', d', c', b', a'\}$
T_1	1	1	0	1	1	0	1	0	1	1	$\{k, i, h, d, c, b, a\}$
T_2	1	0	1	1	1	1	0	1	0	1	$\{l', j', i', f', e', d', c'\}$
T_3	1	1	1	0	1	1	0	1	0	1	$\{l, j, i, f, e, d, c\}$
T_4	0	1	1	1	1	0	1	0	1	0	$\{m, g, k', i', h', d', c', b', a', i, d, c\}$

Table 4: Response & set $D_c(T_i)$

	$D_0(T_i)$	$D_1(T_i)$
T_0	$\{k', d', c'\}$	$\{i'\}$
T_1	$\{k, d, c\}$	$\{i\}$
T_2	$\{l', d', c'\}$	$\{i'\}$
T_3	$\{l, d, c\}$	$\{i\}$
T_4	$\{d', c'\}$	$\{i'\}$

Table 5: Set of $D_0(T_i)$, $D_1(T_i)$

Name	Vectors(T/F)	Total	Residual(Ave)		Residual(Low)		Residual(High)		Time	
			F	N	F	N	F	N	F	N
C432	67/9	20503	0	226	0	1	1	1939	26.02	1.02
C499	84/21	37675	4	185	0	1	52	3360	154.3	4.25
C880	87/24	109746	0	24	0	1	0	124	50.66	5.31
C1355	95/19	191271	2	62	0	1	64	486	706.6	7.33
C1908	146/44	439453	0	7128	0	1	0	176321	957.4	35.6
C2670	304/75	1225395	1	5353	0	15	8	35702	6002	111.6
C3540	208/28	1514670	0	1559	0	1	1	22699	1328	56.2
C5315	341/49	3399528	3	10528	0	1	27	107478	20193	146.25
C6288	78/34	3073960	6	2691	0	1	94	30506	5934	253.56
C7552	267/56	7321051	2	12412	0	1	32	267087	18162	270.46

Table 6: ISCAS85 Results with Stuck-at Test Set

Name	Vectors(T/F)	Total	Residual(Ave)		Residual(Low)		Residual(High)		Time	
			F	N	F	N	F	N	F	N
C432	68/9	20503	0	85	0	0	3	1031	26.07	1.131
C499	76/16	37675	4	20	0	1	26	85	50.25	3.84
C880	82/23	109746	0	19	0	1	1	177	39.61	5.15
C1355	87/21	191271	0	148	0	1	2	2370	856.0	8.24
C1908	146/39	439453	0	8028	0	1	23	199084	757.7	32.54
C2670	292/56	1225395	0	2411	0	1	0	32882	1649.0	89.19
C3540	199/27	1514670	0	468	0	2	3	2636	787.0	53.73
C5315	327/61	3399528	0	93	0	1	1	2178	993.3	189.13
C6288	78/35	3073960	0	8	0	0	1	149	509.4	279.42
C7552	273/57	7321051	1	5634	0	1	40	140119	3450	290.48

Table 7: ISCAS85 Results with Random Test Set

Name	Vectors(T/F)	Total	Residual(Ave)		Residual(Low)		Residual(High)		Time	
			F	N	F	N	F	N	F	N
C432	68/9	20503	439	85	3	0	2447	1031	2.709	1.131
C499	76/16	37675	512	20	50	1	1311	85	6.318	3.84
C880	82/23	109746	175	19	0	1	1761	177	5.95	5.15
C1355	87/21	191271	4631	148	84	1	32728	2370	20.15	8.24
C1908	146/39	439453	2614	8028	309	1	43957	199084	69.04	32.54
C2670	292/56	1225395	717	2411	0	1	7665	32882	94.27	89.19
C3540	199/27	1514670	1286	468	26	2	8215	2636	99.14	53.73
C5315	327/61	3399528	341	93	0	1	6474	2178	161.59	189.13
C6288	78/35	3073960	345	8	4	0	4209	149	404.31	279.42
C7552	272/57	7321051	1698	5634	0	1	35421	140119	322.41	288.44

Table 8: ISCAS85 Results without Simulation(random)

Name	Vectors(T/F)	Total	Residual(Ave)		Residual(Low)		Residual(High)		Time	
			F	N	F	N	F	N	F	N
s298	54/10	12090	43	82	0	2	217	471	0.5213	0.6107
s344	39/14	21945	43	40	0	1	250	216	1.0307	0.79
s349	37/13	22155	54	40	1	1	334	465	0.929	0.748
s526	100/21	29646	16	27	0	1	129	187	1.718	2.252
s641	78/21	113050	203	56	0	0	1570	343	5.847	4.123
s713	86/20	119316	498	112	0	1	5397	742	6.951	4.693
s820	175/9	56280	310	2742	13	0	942	16166	2.467	2.296
s832	172/20	55611	332	1247	0	0	941	15089	3.650	3.595
s1238	177/20	163306	1244	585	0	1	5192	3981	9.672	8.401
s1423	128/29	341551	321	4657	0	1	3306	115006	27.434	26.191

Table 9: ISCAS89 Results without Simulation(random)

Name	Vectors(T/F)	Total	Residual(Ave)		Residual(Low)		Residual(High)		Time	
			F	N	F	N	F	N	F	N
s298	54/10	12090	56	268	0	1	249	4283	0.485	0.584
s344	39/13	21945	83	58	2	0	272	323	0.970	0.717
s349	39/14	22155	140	121	2	2	702	840	1.064	0.823
s526	98/29	29646	29	35	0	1	274	454	2.245	2.861
s641	82/17	113050	1028	366	0	2	5695	2346	5.331	3.965
s713	80/16	119316	1860	757	0	3	7337	3439	7.160	4.249
s820	178/6	56280	203	814	56	41	600	14474	2.214	2.172
s832	176/22	55611	391	722	31	1	925	2888	4.528	4.100
s1238	177/24	163306	1069	539	0	1	4723	3833	12.11	10.196
s1423	129/23	341551	1107	320	0	1	13960	2043	25.912	21.287

Table 10: ISCAS89 Results without Simulation(Stuck)

Fault	Faulty Inputs	Residual
1	5	11
2	14	317
3	69	35
4	61	318
6	47	1
8	1	176321
24	40	218

Table 11: C1908: Residual Faults VS. Faulty Input Vectors

Appendix

I Correctness of Procedure InitialSet

In following lemmas, theorem, and their proofs, we assume the Object fault f is between lines X and Y, i.e. $f =< X, Y >$. Initial graph is $G = (V, E)$.

Lemma I.1: The initialization graph defined in section 4 contains at most two disconnected subgraphs.
Proof. The proof is by contradiction. Assume that we have an initialization graph G with $l \geq 3$ disconnected subgraphs $G_1, \cdots, G_l$. Pick any three arbitrary components G_i, G_j and G_k. Let $T_{i'}$ be a vertex in G_i, $T_{j'}$ be a vertex in G_j, and $T_{k'}$ be a vertex in G_k, $1 \leq i < j < k \leq l$. Note that, as defined, all vertices of G corresponding to faulty vectors. Therefore, either X or Y or both X, Y should be reachable from the faulty output. Therefore, from the definition of $D_c(T_i)$, $\forall T_i \in V$, $D_c(T_i)$ contains either X or Y or both. Thus either $D_c(T_{i'}) \bigcap D_c(T_{j'}) \neq \Phi$ or $D_c(T_{i'}) \bigcap D_c(T_{k'}) \neq \Phi$, or $D_c(T_{j'}) \bigcap D_c(T_{k'}) \neq \Phi$. Hence, at least two of the three subgraphs G_i, G_j, G_k are connected. This contradicts our assumption $\Box$
Lemma I.2: If an initial graph G has two disconnected subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, $\forall T_i \in V_1$, $D_c(T_i)$ contains only one of the two faulty lines, and $\forall T_j \in V_2$ $D_c(T_j)$ contains only the other faulty line.
Proof. From the definition of $D_c(T_i)$, we know that $\forall T_i \in G$, $D_c(T_i)$ contains either X or Y or both X, Y. Now assume $T_i \in V_1$, $T_j \in V_2$. Since G_1 and G_2 are two disconnected subgraphs, $D_c(T_i) \bigcap D_c(T_j) = \Phi$. Therefore none of X, Y can be in both $D_c(T_i)$, $D_c(T_j)$. Since i, j are arbitrary, therefore $\forall T_i \in V_1$, $D_c(T_i)$ must contain one of the lines, say X. $\forall T_j \in V_2$ $D_c(T_j)$ must contain the other line, i.e. Y. $\Box$
Lemma I.3: Let $G' = (V', E')$, $G'' = (V'', E'')$ be the two subgraphs obtained by using *Procedure Partition*. $\forall T_i \in G'$, $D_c(T_i)$ contains only one of the two faulty lines, and $\forall T_j \in G''$ $D_c(T_j)$ contains only the other faulty line.
Proof. As we know that $\forall T_i, T_i \in V$, $D_c(T_i)$ either contains X or Y or both X, Y. Assume that Procedure Partition selects non-adjacent vertices T_i, T_j to construct

G', G''. Since T_i and T_j are not adjacent to each other, T_i and T_j can not contain the same line. Without loss of generality, assume that $X \in D_c(T_i)$ and $Y \in D_c(T_j)$. Then for all vertices T_k adjacent to T_i but not adjacent to T_j, $D_c(T_k)$ must contain X and not Y. Therefore, $\forall T_i \in G'$, $D_c(T_i)$ contains line X but not Y. Using the same argument, $\forall T_j \in G''$ $D_c(T_j)$ contains line Y but not line X. $\Box$

Theorem I.1: If the initial fault list I contains the *Object Fault*, the initial set of residual TSBFs computed by Procedure *InitialSet* contains the *Object Fault.*

Proof. The proof for case 1 follows from Lemma I.2.

The proof for case 2.1 follows from Lemma I.3.

For case 2.2, since $\forall i,\ T_i \in T$, $D_c(T_i)$ contains either X, or Y or both line, therefore $D_c(T_j)$ contains at least one of the shorted line. A is all the lines in the circuit, therefore $< A, D_c(T_j) >$ contains the Object fault.

Therefore, the initial set of residual TSBFs computed by procedure *InitialSet* contains the Object TSBFs. $\Box$

II Correctness of Procedure Diag_NFTSBF

Lemma II.1: In procedure Diag_NFTSBF, while processing T_i, if the object fault is a NFTSBF and the OPS $< A, B >$ contains the object fault, then either α, or β or γ list contains the object fault.

Proof: Since we only process faulty vectors, the object fault is detected by the current test vector T_i. Thus the object fault either exhibits stuck-at-1, or stuck-at-0 or neither stuck-at-0 nor stuck-at-1 behavior. From observation 3.1, we know that the object fault must be in $\{< A^0, B^1 >, < A^1, B^0 >\}$. From observation 3.2, we see that if the object fault exhibits stuck-at-1 behavior, the list

$\alpha = \{< A^0 \bigcap D_1(T_i), B^1 >, < B^0 \bigcap D_1(T_i), A^1 >\}$ contains the object fault. If the object fault exhibits stuck-at-0 behavior, the list

$\beta = \{< A^0, B^1 \bigcap D_0(T_i) >, < B^0, A^1 \bigcap D_0(T_i) >\}$ contains the object fault. If the object fault exhibits neither stuck-at-1 nor stuck-at-0, γ list contains the object fault. Since these are the only possible behaviors the object fault can have, it is contained in at least one of three lists. $\Box$

Lemma II.2: Given a fault list x and a desired behavior b, procedure *Reduce(x,b)* returns a list of fault which exhibits the desired behavior, on application of T_i.

Proof: The proof follows from the constructive algorithm.$\Box$

Lemma II.2: If the object fault is a NFTSBF and OPS $< A, B >$ contains the object fault, then after processing T_i using steps 1-3 in procedure *Diag_NFTSBF* F' contains the object fault.

Proof: Step 0 and step 1 will not affect the fault list. From lemmas II.1 and II.2, we know that regardless of what the behavior of the object fault is, it will be in $\alpha' \bigcup \beta' \bigcup \gamma'$. Therefore, after step 2 and 3, the object fault will be in F'.$\Box$

Theorem II.1: If the object fault is a NFTSBF, the residual fault list computed by procedure *Diag_NFTSBF* contains the object fault.

Proof: Without loss of generality, assume that the object fault is $< X, Y >$. Since initial fault list is either all TSBFs in C or computed by procedure *InitialSet*, $< X, Y > \in F$. Therefore, for faulty input vector T_0, $\exists < A, B > \in F$ s.t $< X, Y > \in < A, B >$. From lemma II.3, $< X, Y > \in F'$ and hence $< X, Y > \in F$ after processing T_0. Assume that after processing faulty input vector T_{i-1}, $< X, Y > \in F$. For faulty input vector T_i, $\exists < A, B > \in F$ s.t. $< X, Y > \in < A, B >$. From lemma II.3, $< X, Y > \in F'$ and hence $< X, Y > \in F$ after processing T_i. Therefore, after processing all faulty input vectors, the object fault $< X, Y > \in F$.$\Box$

Theorem II.3: The object fault will be either in residual fault list computed by procedure *Diag_NFTSBF* or in the residual fault list computed by procedure *Diag_FTSBF*.

Proof: If the object fault is a FTSBF, from theorem II.2, it is in the residual fault list computed by *Diag_FTSBF*. If the object fault is a NFTSBF, from theorem II.1, it is in the residual fault list computed by *Diag_NFTSBF*. Since a bridging fault could either be FTSBF or NFTSBF, the object fault is in one of the fault list.$\Box$