A Diagnosis Algorithm for Bridging Faults in Combinational Circuits*

Yiming Gong
Sreejit Chakravarty
Department of Computer Science
State University of New York
Buffalo, NY 14260

Abstract

A novel algorithm for diagnosing bridging faults in combinational circuits is presented. The voting model and its simplifications, the Wired-AND and Wired-OR models, are assumed. The diagnosis algorithm guarantees the "object bridging fault" (bridging fault in the circuit being diagnosed) to be in the final fault list. The novelty of the algorithm are: (i) unlike previous algorithms it does not use the full fault dictionary but uses only portions of the stuck-at fault dictionary which is computed dynamically; (ii) it enumerates the faults implicitly using a compact data structure; and (iii) fault dropping rules, using stuck-at fault simulation only, are used. The resulting algorithm is both time and space efficient. Experimental evaluation of the algorithm is presented.

1 Introduction

In a circuit when two or more distinct lines are unintentionally connected, due to a defect, we have a bridging fault. Bridging faults model 40% to 50% of physical faults in contemporary MOS technologies [14, 16, 21]. Diagnosis is the process of locating the fault(s) causing system (chip) failure which helps in identifying manufacturing and/or design errors.

We address the problem of diagnosing bridging faults in combinational circuits defined as follows: Given a set of test vectors $T = \{T_1, T_2, \cdots, T_N\}$, a combinational circuit C, and the response R of the faulty circuit to T (the difference between responses from the fault-free circuit and faulty circuit for each T_i), determine the set of bridging faults which can result in the response R. Note that we are

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

not addressing the problem of computing the diagnostic test set T but the problem of diagnosing the circuit given any diagnostic test set T.

Fault diagnosis has been discussed in the literature mostly for stuck-at faults [1, 3, 12, 13, 23, 28, 30, 31]. Two approaches for diagnosing bridging faults have been studied. The first uses I_{ddq} measurement [8, 11], and the second uses logic measurement [10, 26]. We consider logic measurement based diagnosis of bridging faults.

Our logic measurement based diagnosis algorithm is described in Section 4. The novelty of the algorithm are: (i) unlike previous algorithms it does not use the full fault dictionary for bridging faults but uses only portions of the stuck-at fault dictionary which are computed dynamically; (ii) it enumerates bridging faults implicitly using a compact data structure; (iii) fault dropping rules, based on stuck-at fault simulation only, are used. The net result is a time and space efficient algorithm.

The possible two-line bridging faults (TSBF) computed by the diagnosis algorithm are known as **Residual Faults**. The fault causing the failure is the **Object Fault**. For a correct diagnosis algorithm, the *Object Fault* must be a *Residual Fault*. Our algorithm guarantees the Object Fault to be a Residual Fault, given that the voting model correctly models the behavior of the faulty circuit.

The only other known logic measurement based diagnosis algorithm for bridging faults[26] uses heuristics to compute the diagnostic classes of TSBFs from stuck-at fault dictionaries. For each diagnostic class the test vectors that would detect it and the faulty responses are computed. The TSBFs are explicitly enumerated. Thus it requires a large amount of space to store the diagnostic classes of TSBFs and the computation time for computing the diagnostic classes is also very high. This restricts its usefulness to small circuits only.

To conserve space the algorithm approximates the diagnostic classes[26]. This approximate algorithm does not guarantee identification of the TSBFs which caused the failure[26]. However, it does use the reasonably accurate Voting Model [5], to model behavior of the faulty circuit in the presence of TSBFs.

Thus, our algorithm differs from the only known logic measurement diagnosis algorithm for TSBFs[26] in that unlike this method we do not: (i) make use of the diagnostic classes of TSBFs; (ii) use any static TSBF dictionary but only dynamic stuck-at fault dictionary[29]; and (iii) explicitly enumerate the TSBFs but uses an implicit enumeration technique.

In the next two sections we discuss the fault model and define some terms used. Next, we discuss the diagnosis algorithm. Finally we present experimental results for ISCAS85 [7] and ISCAS89 [6] benchmark circuits.

2 Fault Models

Models for modeling circuits with TSBFs in them have to provide a means to determine the voltage at the faulty nodes as well as interpretation of the voltage at these nodes by gates connected to the nodes. In the Voting model when lines X, Y are shorted, if $T(X) = T(Y)$ then the value at the shorted point $V(X,Y)$ equals $T(X)$. Else, $V(X,Y)$ depends on the ratio of pull-down, pull-up strength at the shorted point. In Figure 1, the circuit in (a) is modelled as the potential divider of (b) when X, Y are shorted. Let R_u (R_d) be the resistance of the pull-up (pull-down) network at X (Y). Then, the voltage at the shorted point $V_{br} = \frac{R_d}{R_d+R_u} \times V_{dd}$.

In the Voting Model, logic threshold, which is a function of the technology in use, for all gates is assumed fixed (eg $2.5V$ [25]). If $\frac{R_d}{R_d+R_u} \times V_{dd}$ is less (greater) than this threshold the logic value at the fault location is interpreted as 0(1) by all gates driven by it and the TSBF is said to exhibit **s-a-0 behavior**(**s-a-1 behavior**). If $\frac{R_d}{R_d+R_u} \times V_{dd}$ equals the threshold the TSBF is said to exhibit **neither s-a-1 nor s-a-0 behavior**. The effect of the intermediate level at the faulty point on the rest of the circuit is not known.

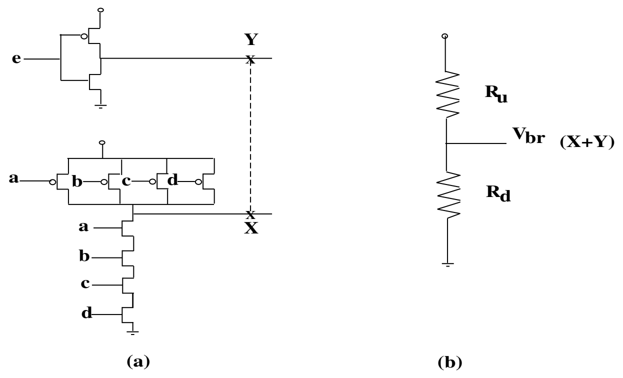

Figure 1: The Voting Model

The pull-up and pull-down resistances depend on the technology, the transistor sizes, and the number of conducting transistors. Two nFETs in series each with channel resistance r may not have a resistance of $2r$. The equivalent resistance is encoded in a table and used by our algorithm. Notation #p (#n) represent the number of pFETs (nFETs) in series, and p# (n#) represent the number of pFETs (nFETs) in parallel.

In Table 1[25] **nfactors** (**pfactors**) is the strength of an nFET (pFET) network relative to an nFET(pFET). Thus two nFETs in series (2n) sinks about 63% of the current of one nFET. Table 2[25] is derived from Table 1 by dividing the **pfactors** by the strength ratio of a pFET and an nFET assuming a $\pm 10\%$ variation in fabrication process (**max ratio** and **min ratio**, respectively). For example, the entry for p2/3n is computed as follow. The pull-up strength of $p2$ is 0.72. The pull-down strength of $3n$ is 0.46. Thus $\frac{R_u}{R_d} = \frac{0.46}{0.72} = 0.64 < 1$, implying the faulty point to be at logic 1.

Primary inputs are assumed to have infinite driving capability. Any line shorted with a primary input is driven to the value of the primary input. If two primary inputs (with different values) are shorted, the fault exhibits s-a-0 behavior.

In the Wired OR (AND) model, for the fault $\langle e, f \rangle$, the value at the faulty point is the OR(AND) of the two fault-free values at lines e and f. This simpler model correctly describes the behavior of circuits with TSBFs in some technologies.

3 Definitions

TSBFs are implicitly enumerated by representing them by a set of **Ordered Pairs of Sets** (**OPS**). To represent TSBFs between lines X, Y two forms of OPS are used: (i) if $X,\ Y \in A,\ X \neq Y$, form $\{\langle A, \Psi \rangle\}$ is used; and (ii) if $X \in A,\ Y \in B,\ A \bigcap B = \Phi$, form $\{\langle A, B \rangle\}$ is used.

The **Faultlist** is represented by a set of OPS (**SOPS**) of the form $\{\langle A, B \rangle, \cdots\}$, where $\langle A, B \rangle$ is an OPS. For example, the SOPS $\{\langle \{1,2\}, \{3,4\} \rangle, \langle \{5,6\}, \Psi \rangle\}$ represents TSBFs $\langle 1,3 \rangle,\ \langle 1,4 \rangle,\ \langle 2,3 \rangle,$ $\langle 2,4 \rangle$, and $\langle 5,6 \rangle$.

The **lines** of a circuit are its primary inputs and gate outputs. Let $T_i(X)$ (respectively, $T_i^f(X)$) denote the value at line X on application of T_i to the fault free circuit (respectively, circuit with the TSBF f in it). For any T_i the set of lines S in a fault free circuit has two subsets: **zero subset**, **one subset**. Zero subset (one subset) contain all lines set to 0(1) by T_i. T_i is known from context and dropped for brevity. Lines in one subset (zero subset) are called **one lines** (**zero lines**).

Tests which detect the fault are **faulty vector**s. For each faulty vector T_i, sets, $D_1(T_i)$, $D_0(T_i)$ and $D_c(T_i)$ are defined as follows: $D_1(T_i)$ (respectively, $D_0(T_i)$) is the set of stuck-at 1 (stuck-at 0) faults which, on application of T_i, have the same output pattern as the given faulty response. $D_c(T_i)$ is the set of lines X such that there exists a path from X to a primary output Y_j, where Y_j is a faulty output on application of T_i. These sets, constituting the modified stuck-at fault dictionaries, are computed dynamically during diagnosis.

In Figure 2 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, stuck-at-0 faults at lines *4, 9, 10, 11,* and *12* result in the response $< 8 = 0, 12 = 0 >$, $D_0(T) = \{4, 9, 10, 11, 12\}$. No stuck-at-1 fault can result in the output pattern $< 0, 0 >$. Thus, $D_1(T) = \Phi$. There exist paths from *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. Therefore, $D_c(T) = \{1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12\}$.

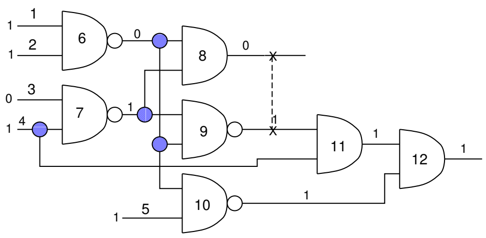

Figure 2: Example for computing D_1,D_0,D_c

For all lines X, $Cone(X)$ is the subcircuit of the given circuit such that $Y \in Cone(X)$ if and only if there is a path from Y to X. For all lines X, $Reach(X)$ is the subcircuit of the given circuit such that $Y \in Reach(X)$ if and only if there is a path from X to Y. In Figure 2, $Cone(8) = \{1, 2, 3, 4, 6, 7\}$, $Reach(9) = \{11, 12\}$.

A TSBF $\langle X, Y \rangle$ is a **non-feedback TSBF** (**NFTSBF**) if and only if there is no path either from X to Y or from Y to X. A TSBF that is not an NFTSBF is a **feedback TSBF** (**FTSBF**).

4 The Algorithmic Paradigm

The diagnosis algorithm consists of three major steps: Initialization, Computing Fault Dictionary, and Fault Dropping.

For each input vector

  Use SSA fault simulation to compute D_0*,* D_1 *and* D_c*;*

  Drop NFTSBFs from fault list;

Initialize FTSBF list;

For each input vector

  Use SSA fault simulation to compute D_0*,* D_1 *and* D_c*;*

Drop FTSBFs from fault list;

Initialization is based on SSA fault simulation and described in Section 5. D_0 and D_1 are computed using a modified SSA fault simulator as discussed in Appendix I. Section 6 contain rules, based on D_0, D_1 and D_c, used for **fault dropping**.

5 Initialization

To detect a TSBF, the fault must be activated and the faulty value propagated to the primary outputs. If the lines have different values then the TSBF is activated. For NFTSBFs the same input vector must also propagate the faulty value to primary outputs. FTSBF however can be activated by one input vector and detected later by another input vector [4].

The idea behind the initialization procedure is as follows. If a TSBF is detected by T_i, at least one of the two shorted lines must be in $D_c(T_i)$. If there are two faulty vectors T_i, T_j such that $D_c(T_i) \bigcap D_c(T_j) = \Phi$ then the *Object Fault* is in $\{\langle D_c(T_i), D_c(T_j)\rangle\}$. Further if $D_c(T_k) \bigcap D_c(T_i) \neq \Phi$ then one of the lines must be in both sets. A similar idea can be used for Wired models[9, 10].

InitialSet, for the Voting Model, is presented next. It uses the notion of an **Initialization Graph** $G = (V, E)$ defined below: $V = \{T_i | T_i \in T, T_i$ is faulty $\}$ and $(T_i, T_j) \in E$ if and only if $D_c(T_i) \bigcap D_c(T_j) \neq \Phi$. From $D_c(T_i)$ of Table 3 we get the initialization graph shown in Figure 3(a). Let I_{FTSBF}(I_{NFTSBF}) denote the initial set of residual FTSBFs(NFTSBFs).

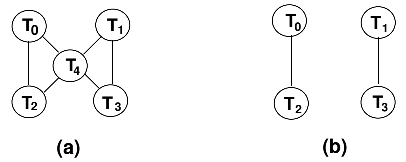

Figure 3: Initialization Graph: (a) $T = \{T_0, T_1, T_2, T_3, T_4\}$, (b) $T = \{T_0, T_1, T_2, T_3\}$

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 Initialization Graph $G = (V, E)$;
Compute the connected components of G;
/* G can have at most two components. See Appendix II, Lemma II.1.1 */
<u>Case 1</u>: G has two connected components $G1$, $G2$.

$S_1 \longleftarrow \bigcap_{T_i \in G1} D_c(T_i)$; $S_2 \longleftarrow \bigcap_{T_j \in G2} D_c(T_j)$;
$I_{NFTSBF} \longleftarrow \{\langle S_1, S_2 \rangle\}$;
$I_{FTSBF} \longleftarrow \{\langle \{X\}, Reach(X) \bigcap S_2 \rangle, \langle Cone(X) \bigcap S_2, \{X\} \rangle | X \in S_1\}$;

<u>Case 2</u>: G has only one connected component.

<u>Case 2.1</u>: G is not a complete graph.

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

$S_1 \longleftarrow \bigcap_{T_i \in G'} D_c(T_i)$; $S_2 \longleftarrow \bigcap_{T_j \in G''} D_c(T_j)$;
$I_{NFTSBF} \longleftarrow \{\langle S_1, S_2 \rangle\}$;
$I_{FTSBF} \longleftarrow \{\langle \{X\}, Reach(X) \bigcap S_2 \rangle, \langle Cone(X) \bigcap S_2, \{X\} \rangle || X \in S_1\}$;

<u>Case 2.2</u>: G is complete graph.

Let j be such that $\forall i,\ T_i \in T, ||D_c(T_j)|| \leq ||D_c(T_i)||$
$I_{NFTSBF} \longleftarrow \{\langle A, D_c(T_j) \rangle\}$;
$I_{FTSBF} \longleftarrow \{\langle \{X\}, Reach(X) \rangle, \langle Cone(X), \{X\} \rangle || X \in D_c(T_j)\}$;

$I_{NFTSBF} \longleftarrow I \bigcap I_{NFTSBF}$;
$I_{FTSBF} \longleftarrow I \bigcap I_{FTSBF}$;

End of Procedure

In Figure 4, let i and i' be the two shorted lines. For each test vector of Table 4, the good and faulty responses, and the set $D_c(T_i)$ are shown in Table 3.

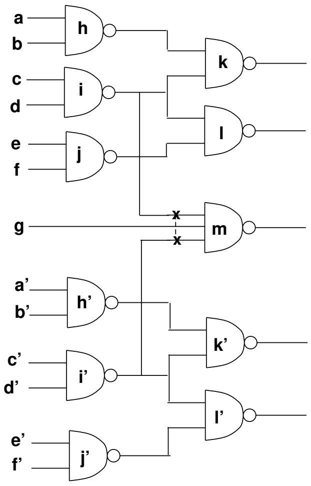

Figure 4: Initialization Example

The initialization graph of Figure 3(b) is for the test set $T = \{T_0, T_1, T_2, T_3\}$. There are two disconnected subgraphs $G_1 = (V_1, E_1) = (\{T_0, T_2\}, \{(T_0, T_2)\})$ and $G_2 = (V_2, E_2) = (\{T_1, T_3\}, \{(T_1, T_3)\})$. Thus, (Case 1), the initial set of NFTSBF residual faults is $\{\langle \bigcap_{T_i \in G_1} D_c(T_i),\ \bigcap_{T_j \in G_2} D_c(T_j) \rangle = \langle \{i', d', c'\}, \{i, d, c\} \rangle\}$, and the initial set of FTSBF residual faults is $\{\langle \{i'\}, \Phi \rangle, \langle \{d'\}, \Phi \rangle, \langle \{c'\}, \Phi \rangle\} = \Phi$. Note that the NFTSBF $\langle i, i' \rangle$ is in the list.

The initialization graph of Figure 3(a) is for the test set $T = \{T_0, T_1, T_2, T_3, T_4\}$. It has one component, but is not a complete graph. Thus, (Case 2.1 use Procedure Partition below) we remove vertex T_4 and all edges connected to it. The new graph is identical to the one in the previous example resulting in the same set of residual faults.

If the initialization graph has one component but is not complete it is converted into a graph with two components. In Appendix II.1 we show that if the following partitioning procedure is used then the set of faults obtained by procedure *InitialSet* contains the Object fault.

Procedure Partition

/*Given an initialization graph $G(V, E)$ which contains one component and is not complete, partition G 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

6 Fault Dropping

6.1 Dropping NFTSBFs

For each faulty input vector the procedure for dropping NFTSBFs consists of two major steps: (1) Dropping inactive faults; (2) Dropping inconsistent faults. We next explain these two steps in detail. Note that faults could have been dropped using both the faulty and the good vectors. However, as reported in Section 7 and in [10] this leads to a dramatic increase in the computaion time without any appreciable gain.

Dropping inactive faults: For NFTSBFs if a fault is detected by T_i, then it sets the two lines involved to opposite values. Faults $\langle X, Y\rangle$ such that $T_i(X) \neq T_i(Y)$ are retained. Using SOPS enables us to perform this using some set operations. For example, if $\{\langle A, B\rangle\}$ is the fault list, A^0 (respectively, B^0) the zero subset of A (B) and A^1 (B^1) the one subset of A (B). Then, faults $\{\langle A^0, B^0\rangle, \langle A^1, B^1\rangle\}$ are dropped leaving us with the new fault list $\{\langle A^0, B^1\rangle, \langle A^1, B^0\rangle\}$.

Dropping inconsistent faults: This step drops faults which are activated by T_i but the possible faulty response is inconsistent with the observed response. For example if, after computing the behavior of the fault, the fault exhibits s-a-1 behavior, the shorted line which is driven to 1 by the faulty value should be in the set $D_1(T_i)$. Constraints like this can be used to further trim the fault list as described below.

For the Voting model, let V_t be the logic threshold and V_{dd} the supply voltage. Assume the fault is detected by the current input vector T_i.

Rule 1. If a NFTSBF exhibits s-a-0 behavior, then $\frac{R_u}{R_d} \geq \frac{V_{dd}}{V_t} - 1$; and the shorted line with fault free logic value 1 must be in $D_0(T_i)$.

Rule 2. If a NFTSBF exhibits s-a-1 behavior, then $\frac{R_u}{R_d} \leq \frac{V_{dd}}{V_t} - 1$; and the shorted line with fault free logic value 0 must be in $D_1(T_i)$.

Rule 3. If a NFTSBF exhibits neither s-a-1 nor s-a-0 behavior, then $\frac{R_u}{R_d} = 1$; and at least one of the shorted lines must be in $D_c(T_i)$.

In the above example, if all faults exhibit s-a-0 behavior (called α **list**), we have following fault list : $\{\langle A^0, B^1 \bigcap D_0(T_i)\rangle, \langle A^1 \bigcap D_0(T_i), B^0\rangle\}$. From this list faults inconsistent with the observed response are dropped. Likewise, if we assume that all faults to exhibit s-a-1 behavior (called β **list**), we have following fault list : $\{\langle A^0 \bigcap D_1(T_i), B^1\rangle, \langle A^1, B^0 \bigcap D_1(T_i)\rangle\}$. Next faults inconsistent with the observed response are dropped from this list. Finally, if all faults exhibit neither s-a-1 nor s-a-0 behavior (called γ **list**), we have following fault list : $\{\langle A^0, B^1\rangle, \langle A^1, B^0\rangle\}$. Sublists of α, β and γ obtained after fault dropping are merged to form new set of Residual Faults. Explicit enumeration of TSBFs, as detailed below, are avoided during the consistency checking phase. Consistency checking is done by Procedure Reduce. The correctness of the algorithm is presented in Appendix II.2.

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 /* Computing fault dictionaries */
          $D_c(T_i)$, $D_0(T_i)$ *and* $D_1(T_i)$ *are computed using stuck-at fault Simulation;/* See Appendix I. */*
        Step 1 /* Drop inactive faults */
          Partition A *into* A^0 *(zero subset),* A^1 *(one subset);*
          If $B \neq \Psi$*, partition* B *into* B^0 *(zero subset),* B^1 *(one subset)*
          Else $B^0 \longleftarrow \Phi$, $B^1 \longleftarrow A^1$;
          $G_1 = \{< A^0, B^1 >, < B^0, A^1 >\}$;

Step 2 /* Make assumptions about fault behavior */

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

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

$\gamma \longleftarrow \{< A^0, B^1 >, < B^0, A^1 >\}$; /* neither s-a-0 nor s-a-1 */

Step 3 /* Check consistency */

$\alpha' \longleftarrow reduce(\alpha, sa0)$; /* If the object fault exhibits s-a-0, it should satisfy Rule 1. Otherwise, it is dropped from α. */

$\beta' \longleftarrow reduce(\beta, sa1)$; /* Fault that exhibit s-a-1 behaviour should satisfy Rule 2. Otherwise, it is dropped from β. */

$\gamma' \longleftarrow reduce(\gamma, san0n1)$; /* If the object fault exhibits neither s-a-0 nor s-a-1, it should not satisfy rule 3. Otherwise, it is dropped from γ. */

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

End

$F \longleftarrow F'$;

End

End of Procedure

Procedure Reduce(list, behavior)

/* Let *list* be the list to be processed; *behavior*, that is s-a-0, s-a-1 etc., be the desired behavior such that faults not exhibiting that behaviour is dropped; $< B^0, B^1 >$ the OPS in list that is being processed; B^0_j ($D(B^1_i)$ 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_t to be the logic threshold, V_{dd} to be the power source.*/

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

Begin

Step 1:

$B^0 \longleftarrow$ *Sort(B^0, Key:pull-down resistances);* /*Sort and group lines with same pull-down resistance together. */

$B^1 \longleftarrow$ *Sort(B^1, key:pull-up resistances);* /* Sort and group lines with same pull-up resistance together. */

Step 2:

For each maximal subset $B^0_j \subseteq B^0$ /* all lines in B^0_j 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

$\forall B^1_i$, $B^1_i \subseteq B^1$, $\frac{R^i_u}{R^j_d} \geq \frac{V_{dd}}{V_t}$*-1* /* the resistance of lines in B^1_i is R^i_u, if the relation between pull-down and pull-up resistance satisfy rule 1, the fault exhibits s-a-0 behavior. */

$tmp \longleftarrow < B^0_j, \bigcup_{\forall i} B^1_i >$;

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

$\forall B^1_i$, $B^1_i \subseteq B^1$, $\frac{R^i_u}{R^j_d} \leq \frac{V_{dd}}{V_t} - 1$ /* the resistance of lines in B^1_i is R^i_u, if the relation between pull-down and pull-up resistance satisfy rule 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*

$\forall B_i^1,\ B_i^1 \subseteq B^1,\ \frac{R_u^i}{R_d^j} = 1$ /* the resistance of lines in B_i^1 is R_u^i, if the relation between pull-down and pull-up resistance satisfy rule 3, 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

Example: Assume Table 4 to be the input vectors; $< i, i' >$ the Object Fault in Figure 4. For each input vector T_i, $D_c(T_i)$ and the good and faulty responses are shown in Table 3; the set $D_0(T_i)$ and $D_1(T_i)$ are shown in Table 5. $F = \{\langle A, \Psi \rangle\} = \{\langle \{a - m, a' - f', h' - l'\}, \Psi \rangle\}$ is the initial fault list. This is not the initial list computed by the procedure described in Section 5 but is assumed for our example. 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 by T_0. After dropping inactive faults the fault list is $\{\langle A^0, A^1 \rangle\} = \{\langle \{a, b, c, d, g, j, k, a', b', i', j'\}, \{e, f, h, i, l, m, c', d', e', f', h', k', l'\} \rangle\}$. The α, β and γ lists are:

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

Next we reduce these lists. Take list α as an example. 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, and $B_2^0 = \{j, k, i', j'\}$ — same pull-down resistance equivalent to two conducting nFET transistors in series. 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 and the assumptions about primary inputs, we see that only faults in $\langle B_1^0, B_2^1\rangle$ and faults in $\langle B_2^0, B_2^1\rangle$ exhibit s-a-0 behavior. Therefore α list can be reduced to $\alpha' = \{\langle\{a, b, c, d, g, j, k, a', b', i', j'\}, \{k'\}\rangle\}$. 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 $\beta' = \{\langle\{i'\}, \{h, i, e, f, c', d', m, h', e', f', l'\}\rangle\}$ and $\gamma' = \{\langle\{c', d'\}, \{a, b, c, d, g, a', b'\}\rangle\}$ list. The fault list F is now $\{\langle\{a, b, c, d, g, j, k, a', b', i', j'\}, \{k'\}\rangle, \langle\{i'\}, \{h, i, e, f, c', d', m, h', e', f', l'\}\rangle, \langle\{c', d'\}, \{a, b, c, d, g, a', b'\}\rangle\}$. The reader can verify that after all input vectors in Table 4 are processed, the Residual Fault list is $\{\langle\{c, d\}, \{c', d'\}\rangle, \langle\{i\}, \{i'\}\rangle\}$.

The above procedures can be simplified for the Wired models[10] because in this model TSBFs exhibit either s-a-0(Wired-AND) or s-a-1(Wired-OR). Thus, one of the following rules is used[10].

Rule 1 (Wired-AND): If a NFTSBF is active, the shorted lines with logic value 1 in fault-free circuit must be in the set of $D_0(T_i)$. /* Only α list is processed. */

Rule 2 (Wired-OR): If a NFTSBF is active, the shorted lines with logic value 0 in fault-free circuit must be in the set of $D_1(T_i)$./* Only β list is processed. */

6.2 Dropping 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 $< X, Y >$, such that there is a path from X to Y. Let $T(X)$ be the value at X on application of T to the fault free circuit. Then, excluding oscillations, if an input vector T detects the fault one of the following conditions must be satisfied[4].

1. $T(X) \neq T(Y)$ and the fault exhibits Y s-a-$T(X)$ behavior on application of T.
2. $T(X) \neq T(Y)$ and the fault exhibits X s-a-$T(Y)$ behavior on application of T.
3. $T(X) \neq T(Y)$ and the logic value at the faulty location cannot be determined either to be 1 or 0 on application of T.
4. $T(X) = T(Y)$. In this case, T may still detect the fault because the circuit can 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 the object fault exhibits TypeI behavior, while processing T 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. 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 set of Residual Faults. In order to eliminate such faults we perform detailed simulation.

The above discussion leads us to the following two-phase algorithm for diagnosing FTSBF using the Voting model (**Diag-FTSBF**). Phase I is similar to *Procedure Diag_NFTSBF* except a patch is added to retain faults exhibiting TypeII behavior. In Phase II we enumerate the remaining FTSBFs eliminating some using detailed simulation. For the Wired models modifications similar to Diag_NFTSBF are needed[10].

7 Experimental Results

Experiments to evaluate the proposed algorithms was done on ISCAS benchmark circuits. For each circuit 25 instances of the diagnosis problem were randomly generated. A random number generator was used to generate the the TSBF $< X, Y >$. The test set used was that computed by the test generator “atpg”[19] in **misII**. The Response was obtained by simulating the fault $\langle X, Y \rangle$. All data reported are average figures for the 25 instances.

7.1 Wired Models

The implementation consisted of the following steps.

1. Compute sets $D_0(T_i)$ (or $D_1(T_i)$) for all faulty input vector T_i.
2. Compute initial fault list (**InitialSet**).
3. Drop NFTSBF faults (**Diag_NFTSBF**).
4. Drop FTSBF faults (**Diag_FTSBF**).

We performed two experiments. In **Version 1**, for each step, we used *only* the *faulty vectors*. For **Version 2**, for each step, we used *both* the *good* and *faulty vectors*. For fault dropping rules for good vectors see [10]. The results for *Version 1* are tabulated in Table 6, Table 8 whereas the results for *Version 2* are tabulated in Table 7, Table 8. These tables are explained below.

In Table 6, *Size* is the total number of nodes in the circuit, *Detect* and *DetectPat* are the total number of nodes to be stored in the modified stuck-at fault-dictionary that we are using when only the *faulty vectors* are processed. *Time* is the total time, in cpu seconds on a SUN SPARC station 2, to compute these sets. Table 7 shows similar results when both the good and faulty vectors are processed. Note the remarkable difference in the preprocessing time and the time required for these two versions.

In Table 8, *Total* is the number of total faults in the given circuit. *NFTSBF* is the number of initial non-feedback faults, *FTSBF* is the number of initial feedback faults. They were computed by **InitialSet**. Note that this initialization step itself drops a lerge number of faults. The next two columns of this table show the final *Residual Faults* and the additional time required, after preprocessing, when *only the faulty vectors* are processed. The last two columns show the final *Residual Faults* and the additional time required, after preprocessing, when *all the vectors* are processed.

The main conclusion we draw from these table is that the overhead - time, memory requirement - for processing both good and faulty vectors is considerably larger when compared with the overhead when only faulty vectors are processed. However, there is very little improvement in the result when the good vectors are also processed. Henceforth, in our experiments we only consider faulty vectors.

7.2 The Voting Model

In this subsection we present experimental results obtained by running the proposed algorithm on the ISCAS85 [7] combinational, and full scan version of ISCAS89 [6] sequential benchmark circuits on a SUN 4/670 with 64M memory. The outline of our implementation is as follows.

1. Diagnose NFTSBFs (**Diag_NFTSBF**) with initial fault list F set to all TSBFs in the circuit. While diagnosing NFTSBFs 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 (step 2).
2. Compute initial set of candidate FTSBFs from all TSBFs in the circuit (**InitialSet**)
3. Drop faults from this initial set of FTSBFs.

Results are shown in tables 9, 10, 11, 12, 13 and 14. In 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. **Sim. Time** is the time used for computing $D_0(T_i)$, $D_1(T_i)$ and $D_c(T_i)$. The value under **N**(**F**) is the time required by Diag_NFTSBF(Diag_FTSBF). **Low** is the smallest residual fault list among 25 trials and **High** is the largest residual fault list among 25 trials. Note the large variation between the average and the largest size of the set of residual faults. We will come back to this issue later.

Although the final size of the set of residual faults is a small fraction of the total number of faults, as shown in Table 9 it is not as small as we had hoped for. Suspecting the stuck-at test set not to be a good diagnostic test set for TSBFs, we experimented with random test sets of the same size as the stuck-at test sets. The results are tabulated in Table 10. We see a dramatic improvement, for all but C1908 and 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 conclude that the final result depends on the diagnostic test set used.

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 such as C1908. This, we believe, is not so much a function of the effectiveness of the algorithm as it is a function of the test set used, the circuit involved and the object TSBF. 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 C1908. Table 15 shows the relationship between the number of faulty input vectors and the number of residual NFTSBFs for some of the shorts in C1908, when a stuck-at test set was used. Note that faults that have a 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 11. 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.

8 Discussion

We presented a novel algorithm for diagnosing TSBFs in combinational circuits. Although TSBFs are different from SSA faults, information provided by SSA fault simulation can help in diagnosing TSBFs. Our algorithm uses a set of rules, derived from the behaviors of TSBFs, and the information given by SSA fault simulation, resulting in a time efficient algorithm.

We also showed how to avoid explicit enumeration of TSBFs. To do that we used a compact representation of TSBFs and implicit enumeration technique making the algorithm space efficient.

In some recent papers[22, 27] 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 [22].

Our proposed algorithm can be modified to use Biased-Voting model. In procedure *Reduce*, based on different V_{ds} different resistance will be used for consistency checking. For those TSBFs which exhibits multiple stuck-at behavior, we can treat them as "neither stuck-at 0 nor stuck-at 1 behavior" and put them in the γ list. In phase II we performed 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. This will ofcourse incur an additional overhead.

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. Breuer. A Practical Approach to Fault Simulation and Test Generation for Bridging Faults. *IEEE Trans. on Computers*, C-34(7):658–663, July 1985.

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

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

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

[6] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of Sequential benchmark circuits. In *Proc. Int. Symp. Circuits and Systems*, pages 1929–1934, May 1989.

[7] 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.

[8] 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.

[9] S. Chakravarty and Y. Gong. An Algorithm for Diagnosing Two-line Bridging Faults in Combinational Circuits. In *Technical Report 92-26, Department of Computer Science, University as Buffalo, State University of New York*, 1992.

[10] S. Chakravarty and Y. Gong. An Algorithm for Diagnosing Two-Line Bridging Faults in Combinational Circuits. In *Design Automation Conference*, pages 520–524, 1993.

[11] 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.

[12] 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.

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

[14] F. J. Ferguson and Tracy Larrabee. Test Pattern Generation for Realistic Bridging Faults in CMOS ICs. In *Proceedings of the International Test Conference*, pages 492–499, 1991.

[15] J. Ferguson and J. P. Shen. Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis. In *Proceedings of the International Test Conference*, 1988.

[16] J. Galiay, Y. Crouzet, and M. Vergniault. Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability. *IEEE Trans. on Computers*, C-29(6), June 1980.

[17] Gary S. Greenstein and Janak H. Patel. E-PROOFS: A CMOS Bridging Fault Simulator. In *Proceedings of International Conference on Computer-Aided Design*, pages 268–271, 1992.

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

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

[20] F. Maamari and J. Rajski. A Method of Fault Simulation Based on Stem Regions. *IEEE Trans. on Computer Aided Design*, 9(2):212–, February 1990.

[21] W. Maly. Realistic Fault Modeling for VLSI Testing. In *Proc. 24th DAC*, pages 173–180, 1987.

[22] Peter C. Maxwell and Robert C. Aitken. Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Thresholds. In *Proceedings of the International Test Conference*, pages 63–72, 1993.

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

[24] K. C. Y. Mei. Bridging and Stuck-at Faults. *IEEE Trans. on Computers*, C-23(7):720–727, July 1974.

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

[26] 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.

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

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

[29] P. G. Ryan, S. Rawat, and W. K. Fuchs. Two-Stage Fault Location. In *Proceedings of the International Test Conference*, pages 963–968, 1991.

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

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

	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	0	0	0	0	0	0	0
3p	0	0	0	0	0	0	0
2p	0	0	0	0	0	0	0
p	N	0	0	0	0	0	0
p2	1	1	1	0	0	0	0
p3	1	1	1	1	0	0	0
p4	1	1	1	1	0	0	0

Table 2: Bridging Fault Result Analysis

	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 3: Response & set $D_c(T_i)$ for circuit in Figure 3

	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 4: Test Set

	$D_0(T_i)$	$D_1(T_i)$
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	Size(m)	Vectors(N)	Detect	DetectPat	Time
C1355	619	25.05	3329	421	4.97
C1908	938	42.58	6547	646	20.51
C2670	1566	98.79	29974	1133	47.28
C3540	1741	24.88	5031	371	37.30
C432	203	8.67	269	41	0.79
C499	275	39.90	3422	211	3.25
C5315	2608	72.21	38321	1362	88.65
C6288	2480	34.58	27845	1706	187.52
C7552	3827	50.64	24193	1562	136.17
C880	469	24.52	2934	317	2.84

Table 6: Wired Model, Faulty Input Only

Name	Size(m)	Vectors(N)	Detect	DetectPat	Time
C1355	619	99	14945	421	16.70
C1908	938	146	28455	646	63.99
C2670	1566	310	132436	1133	127.76
C3540	1741	227	61679	371	272.54
C432	203	74	1885	41	4.06
C499	275	100	9204	211	7.20
C5315	2608	370	253136	1362	369.19
C6288	2480	83	98890	1706	442.77
C7552	3827	278	201880	1562	625.38
C880	469	98	12703	317	7.84

Table 7: Wired Model, Good & Faulty Input

Name	Total	NFTSBF	FTSBF	Version 1		Version 2	
				Res. Faults	Time	Res. Faults	Time
C1355	191271	3366	118763	1367.54	1.34	1174.68	6.06
C1908	439453	2483	16119	693.58	3.23	624.54	7.44
C2670	1225395	8020	34823	6316.27	33.58	6202.27	84.63
C3540	1514670	2610	7403	6008.4	4.22	5567.45	37.61
C432	20503	210	747	236.57	0.23	197.42	0.61
C499	37675	168	463	139.55	0.34	139.25	0.52
C5315	3399528	7866	28256	2828.17	42.29	2772.05	85.00
C6288	3073960	10935	386934	12993.67	9.58	10479.66	35.43
C7552	7321051	19500	21714	3688.86	27.44	3561.72	143.09
C880	109746	306	789	125.08	0.47	115.88	0.87

Table 8: Wired Model Diagnosis Results

Name	Vectors(T/F)	Total	Residual(Ave)		Low		High		Sim. Time(S)		Time(S)	
			F	N	F	N	F	N	F	N	F	N
C432	74/9	20503	0	226	0	1	1	1939	0.48	0.44	11.18	0.75
C499	100/21	37675	4	185	0	1	52	3360	2.51	2.47	46.59	3.1
C880	98/24	109746	0	24	0	1	0	124	1.92	2.86	19.57	4.21
C1355	99/19	191271	2	62	0	1	64	486	4.6	4.55	203.8	5.78
C1908	146/44	439453	0	7128	0	1	0	176321	25.95	25.49	336.3	28.62
C2670	310/75	1225395	1	5353	0	15	8	35702	63.31	64.35	1939.5	77.54
C3540	227/28	1514670	0	1559	0	1	1	22699	41.84	41.40	603.2	48.4
C5315	370/49	3399528	3	10528	0	1	27	107478	79.87	80.34	4288.1	97.77
C6288	83/34	3073960	6	2691	0	1	94	30506	207.47	207.17	4339.3	212
C7552	278/56	7321051	2	12412	0	1	32	267087	177.1	175.9	3680.8	194.5

Table 9: ISCAS85 Results with Stuck-at Test Set

Name	Vectors(T/F)	Total	Residual(Ave)		Low		High		Sim. Time(S)		Time (S)	
			F	N	F	N	F	N	F	N	F	N
C432	74/10	20503	0	85	0	1	3	1031	0.51	0.51	16.1	0.861
C499	100/21	37675	4	360	0	1	26	85	2.43	2.41	15.5	3.11
C880	98/23	109746	0	19	0	1	1	177	1.60	2.66	14.7	3.77
C1355	99/24	191271	0	540	0	1	2	2370	5.3	5.3	255.6	6.5
C1908	146/39	439453	0	8028	0	1	23	199084	19.4	19.4	270.5	22.2
C2670	310/61	1225395	0	2411	0	1	0	32882	41.6	48.1	196.3	59.0
C3540	227/29	1514670	0	468	0	2	2	2636	40.8	40.8	336.3	48.1
C5315	370/64	3399528	0	97	0	1	1	2178	91.9	123.5	203.9	142.3
C6288	83/36	3073960	0	8	0	1	1	149	237.3	231.8	382.6	236.3
C7552	278/57	7321051	1	5634	0	1	40	140119	194.7	212.9	682.7	234.9

Table 10: ISCAS85 Results with Random Test Set

Name	Vectors(T/F)	Total	Residual(Ave)		Low		High		Sim. Time(S)		Time(S)	
			F	N	F	N	F	N	F	N	F	N
C432	74/10	20503	439	85	3	1	2447	1031	0.51	0.51	2.35	0.861
C499	100/21	37675	674	360	50	1	1311	85	2.43	2.41	5.98	3.11
C880	98/23	109746	175	19	0	1	1761	177	1.60	2.66	4.66	3.77
C1355	99/24	191271	5263	540	84	1	32728	2370	5.3	5.3	18.0	6.5
C1908	146/39	439453	2614	8028	309	1	43957	199084	19.4	19.4	50.6	22.2
C2670	310/61	1225395	717	2411	0	1	7665	32882	41.6	48.1	63.8	59.0
C3540	227/29	1514670	1286	468	26	2	8215	2636	40.8	40.8	86.5	48.1
C5315	370/64	3399528	355	97	0	1	6474	2178	91.9	123.5	122.6	142.3
C6288	83/36	3073960	359	8	4	1	4209	149	237.3	231.8	341.5	236.3
C7552	278/57	7321051	1698	5634	0	1	35421	140119	194.7	212.9	261.8	234.9

Table 11: ISCAS85 Results without Simulation(Random Test Set)

Name	Vectors(T/F)	Total	Residual(Ave)		Low		High		Sim. Time(S)		Time (S)	
			F	N	F	N	F	N	F	N	F	N
s298	59/10	12090	0	260	0	1	13	4283	0.11	0.16	1.0	0.41
s344	43/14	21945	0	63	0	1	0	323	0.35	0.31	1.61	0.52
s349	41/15	22155	0	106	0	2	5	840	0.36	0.35	2.28	0.57
s526	111/29	29646	0	35	0	1	0	454	0.76	1.18	2.13	1.93
s641	89/18	113050	17	286	0	2	142	2346	1.85	1.96	110.9	2.89
s713	92/16	119316	34	590	0	3	272	3439	1.89	1.89	202.1	2.89
s820	178/6	56280	0	682	0	41	0	14474	0.3	0.3	5.6	1.7
s832	176/22	55611	0	536	0	1	0	2888	1.32	1.32	11.8	2.0
s1238	196/25	163306	3	534	0	1	33	3833	5.4	5.5	175.9	8
s1423	132/23	341551	1	180	0	1	17	2043	11.22	11.61	647.7	14.2

Table 12: ISCAS89 Results (Full Scan) (SSA Test Set)

Name	Vectors(T/F)	Total	Residual(Ave)		Low		High		Sim. Time(S)		Time (S)	
			F	N	F	N	F	N	F	N	F	N
s298	59/10	12090	0	74	0	2	15	471	0.12	0.17	1.13	0.45
s344	43/14	21945	0	40	0	1	0	216	0.37	0.36	1.2	0.57
s349	41/13	22155	0	40	0	1	0	465	0.33	0.33	1.3	0.53
s526	111/21	29646	0	26	0	1	0	187	0.56	0.91	1.74	1.36
s641	89/21	113050	1	52	0	2	18	343	2.38	2.29	11.9	3.12
s713	92/20	119316	8	101	0	1	84	742	2.34	2.47	37.7	3.48
s820	178/9	56280	0	2636	0	37	0	16166	0.48	0.44	9.2	2.0
s832	176/20	55611	0	1209	0	4	0	15089	1.2	1.3	14.2	3.0
s1238	196/20	163306	2	574	0	1	32	3981	4.29	4.49	236.4	6.9
s1423	132/29	341551	0	4646	0	1	3	115006	14.6	16.7	60.7	19.4
s15850_1	117/31	61233711	0	11561	0	4	15	238349	720	1266	20490	1290

Table 13: ISCAS89 Results (Full Scan) (Random Test Set)

Name	Vectors(T/F)	Total	Residual(Ave)		Low		High		Sim. Time(S)		Time (S)	
			F	N	F	N	F	N	F	N	F	N
s298	59/10	12090	43	74	0	2	247	471	0.12	0.17	0.42	0.45
s344	43/14	21945	43	40	0	1	250	216	0.37	0.36	0.79	0.57
s349	41/13	22155	54	40	1	1	334	465	0.33	0.33	0.71	0.53
s526	111/21	29646	16	26	0	1	129	187	0.56	0.91	1.43	1.36
s641	89/21	113050	203	52	0	2	1570	343	2.38	2.29	4.55	3.12
s713	92/20	119316	498	101	0	1	5397	742	2.34	2.47	5.35	3.48
s820	178/9	56280	310	2636	1	37	942	16166	0.48	0.44	2.01	2.0
s832	176/20	55611	332	1209	0	4	950	15089	1.2	1.3	2.95	3.0
s1238	196/20	163306	1244	574	0	1	5192	3981	4.29	4.49	7.6	6.9
s1423	132/29	341551	321	4646	0	1	3301	115006	14.6	16.7	19.89	19.4

Table 14: ISCAS89 Results (Full Scan) without Simulation(Random Test Set)

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 15: C1908: Residual Faults VS. Faulty Input Vectors

Appendix

I Computing Fault Dictionaries

We used a modified stem region based simulator [20] to compute the response of SSA faults in the circuit. Once the response of each SSA fault is computed, the sets $D_0(T_i)$ and $D_1(T_i)$ can be computed by comparing the response of each fault with the given response R. If the two responses are the same and the SSA fault is a s-a-0 fault, it is in $D_0(T_i)$. Otherwise, it is in $D_1(T_i)$.

In a combinational circuit, fault simulation can be performed by simulating only the faults on reconvergent fan-out stems, while determining the detectability of faults on other lines by critical path tracing within fan-out-free regions. In the simulator presented in [20], for every reconvergent fan-out stem, a region of the circuit has been delimited. Outside of the region stem fault does not have to be simulated. Such a region is called **Stem Region** [20]. Lines on the boundary of such a stem region, called **exit lines**, have the following property: if the stem fault is detected on the line, and the line is critical with respect to a primary output, then the stem fault is detected at that primary output.

Based on the notion of stem regions, we have developed a labeling scheme to label each line with parities from that line to primary outputs. The detectability of a fault and, more important in our case, the response of the faulty circuit can be read out direct from this label. The following definitions are used in the description of our algorithm that follows.

Path parity: Notations D, $\overline{D}$ are used in the usual sense to distinguish between the good and faulty circuit [4]. Let x, y be two lines in a circuit, t an input vector, and $T(X)$ denote the fault free value of line x on application of t. Then, we say that there exists a **sensitizing path of even (odd) parity**, on application of t, from x to y iff t propagates a $D(\overline{D})$ to line y, when $T(X) = D$ [10]. For example, in Figure 5, the input vector shown sensitizes a path of even parity from line 1 to line 5 and a path of odd parity from line 4 to 6.

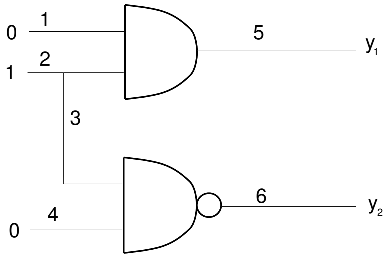

Figure 5: Example Circuit

Concatenation of path parities: We use E, O, and N to denote even, odd and non-sensitizing path, respectively. The parities of two connected paths can be concatenated into one using the operation defined in Table 4. For example, assume the parity of line X to line Y be E(even), line Y to line Z be O(odd), then the parity of line X to line Z is O. The inversion of odd parity is even parity, the inversion of N is still N, etc.

Offpath input, sensitizing value: Let G be a gate and X be an input of G. Then all other inputs of G are considered as offpath inputs of G with respect to X. An input vector is said to assign a sensitizing value to an input of a gate if either: (i) the gate is an OR or NOR gate and the value of the input is 0; (ii) the gate is a NAND or AND gate and the value of the input is 1; or (iii) the gate is an XOR gate.

	E	O	N
E	E	O	N
O	O	E	N
N	N	N	N

Table 16: Parity Concatenation

I.1 Algorithm

The algorithm consists of a backward labeling phase for the whole circuit, and a forward labeling phase for each stem region. Backward labeling labels each line in fan-out-free region with parities from that line to its nearest fan-out stem or primary outputs. Forward labeling of each stem region labels each fan-out stem with the parities from that fan-out stem to the stem region's exit lines. By concatenating the parities of a line X and its "nearest" fan-out stem we get the parities from X to the primary outputs. The response of each line can be read out by simply examining the value of the line and the parities.

Preprocessing: Preprocessing is performed before applying any input vector to extract the stem regions of the circuit [20].

Backward labeling: Backward labeling labels each line in fan-out-free region with parities from that line to its nearest fan-out stem or primary outputs, whichever comes first. Starting from primary outputs, breadth first, we first label all the primary outputs as E (even). For each input line of a gate we traversed, if the input vector assigns sensitizing values to all offpath inputs of the gate, the line has the same parity (inversed parity in the case the value of the line is different from the output of the gate) as the gate output has. If a fan-out stem is encountered, at this time, we label it as E(even).

For example for the circuit in Figure 6 (a), and the input vector shown, starting from primary output (line 9), we first label it as E. Since line 7 and 8 both have value 1 and gate C is an OR gate, the parity from line 7 and 8 to line 9 are both not sensitized (labeled as N). Traversing further, since line 7 and 8 are labeled as N, all lines beyond that point will be labeled as N. But since line 1 and 2 are two fan-out stems, they are labeled as E.

Forward labeling: Forward labeling labels fan-out stem of each stem region with the parities from that fan-out stem to the stem region's exit lines. For each stem region, starting from the fan-out stem, we try to propagate a D to all the exit lines. If a D reaches an exit line and remains unchanged, the parity from the fan-out stem to that exit line is E (even). If D changes to $\overline{D}$, the parity is O (odd). Otherwise, we cannot propagate the D to that exit line and label the parity as N (non-sensitizing).

For example, in Figure 6 (b), if we propagate a D from either line 1 or line 2 to primary output (line 9), we will get a D on line 9 unchanged. Therefore, there exits even parity path from line 1 and 2 to line 9 (primary output and its exit line), line 1 and 2 are labeled as E.

Concatenation of parities: After backward labeling and forward labeling and before computing the response of each fault, we need to compute the parities from each line to all primary outputs. This can be done by concatenate the parities of each segment, from the line to its nearest fan-out stem (if exists) (labeled by backward labeling), from that fan-out stem to its exit lines (labeled by forward labeling), and from those lines to their nearest fan-out stems, and so on and so forth. For example, in Figure 6 (a), line has a parity of E to itself. In Figure 6 (b), fan-out stem of line 1 has a parity of E to its exit line 9. Concatenating these two parities, the parity from line 1 to line 9 is E.

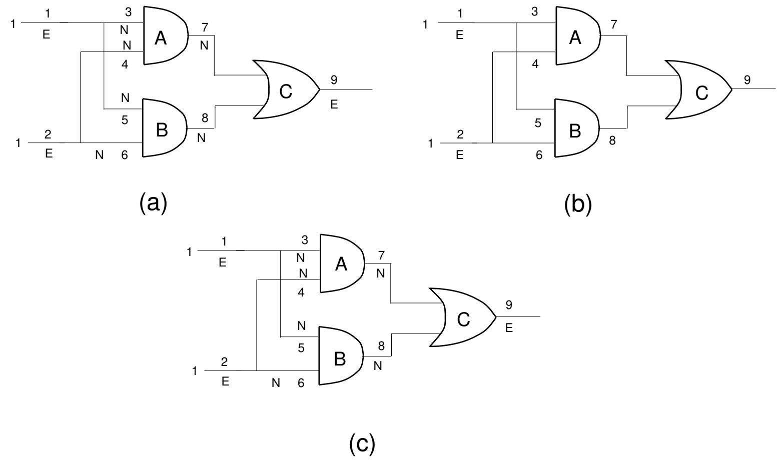

Figure 6: Example of Computing Response

Response: The relation between the faulty response of a line stuck-at fault and the parities of the line to primary outputs is simple. If, in fault free circuit, a line X has value $t(X)$, $t(X) \in \{0, 1\}$, then the fault "line X stuck-at-$t(X)$" has the fault free response. While the response of X stuck-at-$\overline{t(X)}$ fault can be obtained by flip the values of those primary outputs to which the parities from the line are E or O. For example, let $t(X) = 0$, parities of line X is (N, E, O, E, N, N). The fault free response of the circuit is $(1, 1, 0, 1, 1, 0)$. Then line X stuck-at-1 fault has the response of $(1, 0, 1, 0, 1, 0)$.

Data structure: To be space efficient, we use following data structure to store the parities of each line. Associated with each line is its parity to the nearest fan-out stem or primary output. Associated with each fan-out stem is its parities to the stem region's exit lines. To compute the parities of each line to all primary outputs, we recursively concatenate the parities, from the line to its fan-out stem, from the fan-out stem to its exit lines, from these lines to their fan-out stems, so on and so forth, till we reach the primary outputs.

A formal description of the algorithm follows.

Computing_Response(Circuit: C, input vector: t)
Begin
/* Assume stem regions have been extract */
Simulate the fault-free circuit;
Backward labeling;
For each stem region
Begin
Forward labeling;
End
For each line X in C
Begin
Concatenate the parities of X and its nearest fan-out point;
Compute the response from parities of X;
End
End

II Correctness of the Algorithm

II.1 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 II.1.1: The initialization graph defined in section 5 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. □

Lemma II.1.2: 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. □

Lemma II.1.3: 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. □

Theorem II.1.1: 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 II.1.2.

The proof for case 2.1 follows from Lemma II.1.3.

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. □

II.2 Correctness of Procedure Diag_NFTSBF and Diag_FTSBF

Lemma II.2.1: 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. Since the fault is a active fault, we know that the Object Fault must be in $\{< A^0, B^1 >, < A^1, B^0 >\}$. From rules in 6.1.1, 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. □

Lemma II.2.2: 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. □

Lemma II.2.3: 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.2.1 and II.2.2, 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'. □

Theorem II.2.1: 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.2.3, $< X, Y > \in F'$ 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.2.3, $< X, Y > \in F'$ and hence $< X, Y > \in F$ after processing T_i.

Therefore, after processing all faulty input vectors, the Object Fault $< X, Y > \in F$. □

Theorem II.2.2: If the Object Fault is a FTSBF, and I_{FTSBF} contains the Object Fault then the candidate fault list computed by procedure *Diag_FTSBF* also contains the object fault.

Proof: Proof follows the argument in Section 6.2. □

Theorem II.2.3: 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.2.2, it is in the Residual Fault list computed by *Diag_FTSBF*. If the Object Fault is a NFTSBF, from theorem II.2.1, 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. □