A Diagnostic Simulation Algorithm for Stuck-at Faults in Combinational Circuits*

Sreejit Chakravarty
Yiming Gong
Department of Computer Science
the State University of New York at Buffalo
Buffalo, N.Y. 14260

March 29, 1993

Abstract

Two faults are said to be equivalent, w.r.t a test set T, iff they cannot be distinguished by any test in T. The sizes of the equivalence classes are used as a basis for comparing the diagnostic capability of two given test sets. A novel algorithm for computing the Equivalence Classes of stuck-at faults, in combinational(full scan) circuits, w.r.t a given test set is presented. Experimental results presented show the algorithm to be more time and space efficient than a previously known algorithm.

1 Introduction

Fault diagnosis is the process of determining the cause of a circuit failure. In most cases, diagnosis problem can be defined as: given a circuit, a set of test

*Research Supported by NFS Grant No. MIP9102509.

vectors, and a set of output vectors, diagnosis attempts to find the faults or combinations of faults which are consistent with the symptom.

Fault diagnosis is often more difficult than fault detection and usually requires the application of more test vectors. Test vectors used in fault diagnosis may differ from test vectors used in fault detection. In fault detection, input vectors which can cover more faults are required, but in fault diagnosis input vectors which can distinguish one fault from others are required. If all the faults in a circuit are distinguishable, to locate all the faults, at least $n/(2^m - 1)$ test vectors are needed, where n is the number of faults, m is the number of primary outputs.

Test sets differ in their capability to distinguish between sets of faults in a given circuit. This capability of a test set is usually quantified in terms of some parameters of the test set which are defined below. Evaluation of diagnostic test resolution is the process of computing the value of these parameters.

Three diagnostic resolution measures, *Diagnostic Resolution (DR)*, *Diagnostic Power (DP)*, and *Equivalence Class (EC)* have been proposed in [1, 2, 3]. DR is the fraction of fault pairs distinguished by a test set.

$$DR = \frac{number\ of\ fault\ pairs\ distinguished}{total\ number\ of fault\ pairs}$$

A fault that can be distinguished, by a test set, from all other faults is a *fully distinguishable fault* w.r.t that test set. DP is the fraction of fully distinguishable faults.

$$DP = \frac{number\ of\ fully\ distinguishable\ faults}{total\ number\ of\ faults}$$

Two faults are in one equivalence class iff no test vector in the test set can distinguish the two faults. EC, another measure, is the number of equivalence classes and the size of each class.

Let $\{F_1, F_2, \ldots, F_N\}$ be the set of equivalence classes. Then, DR and DP can be computed as following:

$$DR = \frac{\sum_{i \neq j} |F_i| \times |F_j|}{total\ number\ of\ fault\ pairs}$$

$$DP = \frac{\sum_{|F_i|=1} |F_i|}{total\ number\ of\ faults}$$

Initially, only DR and DP were proposed as diagnostic measures[1]. However, it was argued in [3] that these two measures are not a very good set of measures. A third measure, EC was then proposed. This measure contains more diagnostic information and both DP and DP can be computed from EC as stated above.

Note that the set of targeted faults is important while computing these measures. In all work on evaluation of diagnostic test resolution the set of targeted faults is the set of all single stuck-at faults. In this paper we will continue to make this assumption.

Evaluation of diagnostic test set have been discussed in several context [1, 2, 3, 4]. In [1], DR and DP are defined and used to evaluate the diagnostic test pattern generator presented in that paper. In [3], EC as a measure of diagnostic test resolution is introduced, and an algorithm for computing the equivalence classes for both combinational and sequential circuits are presented. In [4], an algorithm, which is a modification of PROOFS[9, 10], for computing DR and DP of sequential circuits is presented.

In this paper we present an algorithm for computing the Equivalence Classes of single stuck-at faults in a given circuit C w.r.t the given test set T. We assume C to be combinational. This algorithm starts by assuming that there is only one equivalence class consisting of all the single stuck-at faults. It then computes the equivalence classes by processing one input vector at a time.

This is a very different approach than the algorithm in [3] that uses N-ary Decision Diagram. As a result the algorithm in [3] potentially needs both exponential space and time. In contrast, our algorithm requires time no more than $O(n_i^2)$ to process each input vector where n_i is the size of the circuit. It also requires $O(n_i + n_f q)$ space where: q is the number of primary outputs, and n_f is the number of fanout points.

We present experimental results on the ISCAS85[11] as well as the ISCAS89[1] benchmarks and show that our algorithm is both faster and more space efficient than the algorithm in [3].

The rest of the paper is organized as follows. In section 2, we briefly introduce and explain concepts and definitions used in this paper. In section 3, we, through an example, give the overall structure of the algorithm, and then explain it in detail. Experiment results are given in section 4.

[1]Obtained from MCNC benchmark project

2 Definitions

2.1 Equivalence Classes

Let **C** denote the circuit under test, let n be the number of primary inputs, and m be the number of primary outputs.

Let $F = \{f_i\}$ be the set of all stuck at faults in C, $T = \{t_i\}$ be the set of test vectors. A response of a circuit is an m-tuple $r =< r(1), r(2), \ldots, r(m) >$, where $r(i)$ is the value of ith primary output. Let r_i denotes the response of C on application of t_i. Then $S^f = \{r_i^f | \ \forall i \quad t_i \in T, \quad f \in F\}$ is the set of responses, on application of the test set T, when fault f is present. Response of the fault free circuit is denoted by S^ϕ. For example, in Figure 2, let $T = \{< 0,1,0 >, < 1,1,1 >\}$, and f be *line 1 stuck at 1*. Then $S^f = \{r_1^f, r_2^f\} = \{< 1,1 >, < 1,0 >\}$ while $S^\phi = \{r_1^\phi, r_2^\phi\} = \{< 0,1 >, < 1,0 >\}$.

Let R be a relation defined over set F, where $R(f_x, f_y) = \{f_x, f_y \in F \ and \ S^{f_x} = S^{f_y}\}$. Clearly, $R(f_x, f_x)$; $R(f_x, f_y) \Longrightarrow R(f_y, f_x)$; and $R(f_x, f_y)$, $R(f_y, f_z) \Longrightarrow R(f_x, f_z)$. Therefore, R is an equivalence relation.

Since R is an equivalence relation, it partitions F into equivalence classes $\{F_i\}$ such that $F_i \subseteq F$; $F = \bigcup F_i$; and $F_i \cap F_j = \Phi, \forall i, j \quad i \neq j$. Since all the faults in one subset F_i have same responses, they cannot be distinguished by the test set T. If $S^f = S^\phi$, f is undetectable by test set T. If $|T| = 1$, S^f is the response of a single test vector.

In the sequel we use $F^j = \{F_i^j\}$ to denote the set of equivalence classes of F w.r.t first j vectors i.e. the set of vectors $\{t_1, t_2, \ldots, t_j\}$. Note: $|\{F_i^j\}| \geq |\{F_i^k\}| \quad \forall j, k \quad j \geq k$. Our algorithm computes $\{F_i^{j+1}\}$ from $\{F_i^j\}$ after processing t_{j+1}.

For the example in Figure 2 assume that x is a line in C, x_0 denotes x stuck at 0 and x_1 denotes x stuck at 1. Initially F contains only one class

$$F^0 = \{F_1^0\} = \{\{1_0, 1_1, 2_0, 2_1, 3_0, 3_1, 4_0, 4_1, 5_0, 5_1, 6_0, 6_1\}\}.$$

On application of $t =< 0,1,0 >$, F^0 is refined to

$$F^1 = \{F_1^1, F_2^1, F_3^1\} = \{\{1_0, 2_0, 2_1, 3_0, 3_1, 4_0, 5_0, 6_1\}, \{1_1, 5_1\}, \{4_1, 6_0\}\}.$$

Algorithms for computing F^{j+1} from F^j will be discussed later on.

2.2 Sensitized Path, Path Parity

The notation D and $\overline{D}$ are in the usual sense to distinguish between the good and faulty circuit. Let $t(x)$ denote the value of line x on application of input vector t. Let G be a gate and x an input of G. Then, all inputs z of G s.t $z \neq x$ constitutes the **offpath inputs of G w.r.t. x**. In Figure 1, 24 is an offpath input of 26 w.r.t. 22. An input vector t is said to assign a **sensitizing value** to an input z of a gate G iff either:

(i) G is an OR or NOR gate and $t(z)$ is 0;
(ii) G is a NAND or AND gate and $t(z)$ is 1;
or
(iii) G is an XOR gate.

Note that if t assigns sensitizing value to all offpath inputs of G w.r.t x, a D or $\overline{D}$ at x can be propagated to the output of G.

Let x, y be two lines in the circuit C and t an input vector. 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 $t(x) = D$[5]. For example, in Figure 1, the input vector shown sensitizes a path of even parity from 28 to 30 and a path of odd parity from 28 to 38.

For every line $x \in C$ and input vector t_i, we define **Parity**(x, t_i) to be an m-Tuple $< a_1, \ldots, a_m >$ where a_j equal $E(O)$, $\forall j \;\; j \leq m$, iff t_i sensitized an even (odd) path from x to the jth primary output y_j, or N otherwise[5]. In Figure 1 assume that $y_1 = 25, y_2 = 26, y_3 = 34, y_4 = 38$. Let t_1 be the input vector defined in Figure 1. In Figure 1 each line is marked by an ordered-pair (a, b). where, a is the value assigned by t_1. On application of t_1 to the circuit there are no sensitized paths from 28 to either 25 or 26, and there are sensitized paths of odd parity from 28 to 34 and 38. Therefore, Parity$(28, t_1) =< N, N, O, O >$.

Note that if $t(x)$ is 1(0), then t does not detect x stuck at 1(0) fault. Therefore, all faults x *s-a-b*,$b = t(x)$ have the response S^{ϕ}. We use f_x to denote the fault x *s-a-b*, $b = \overline{t(x)}$.

Observation 2.1 For a given input vector t, x a line in C, let $r_t^{\phi} =< r_t^{\phi}(1), r_t^{\phi}(2), \ldots, r_t^{\phi}(m) >$ and Parity$(x, t) =< a_1, a_2, \ldots, a_m >$. Then $r_t^{f_x} =<$

$r_t^{f_x}(1), r_t^{f_x}(2), \ldots, r_t^{f_x}(m) >$, where

$$r_t^{f_x}(i) = \begin{cases} r_t^{\phi}(i) & if\ a_i = N \\ \overline{r_t^{\phi}(i)} & if\ a_i \neq N \end{cases}$$

There is a strong correlation between the parity of a line and the equivalence classes. Later on, we will describe *Parity Labeling* that computes the parity of each line very efficiently and show how it can be used to compute the equivalence classes.

2.3 Stem Region, Exit lines

Our algorithm for computing equivalence classes uses the notation of a Stem Region[6]. In general, there is only one path between two nodes in a circuit unless the path passes through fanout point(s). Computing path parities is much easier if at most one path exists between any two lines in a circuit. To compute path parities in the presence of fanout, we use the concepts of *Stem Region* and *Exit Lines* [6].
Let

$$p1 =< g_1 = x, g_2, \ldots, g_{t-1}, g_t = y >$$

and

$$p2 =< h_1 = x, h_2, \ldots, h_{q-1}, h_q = y >$$

be two paths between lines x and y. We say they are **disjoint paths** iff $\forall i, j,\ 1 < i < t,\ 1 < j < q,\ g_i \neq h_j$; We say they are **distinct paths** iff $\exists j$, $g_j \neq h_j$.

In Figure 1, $< 28, 30, 31, 32, 34 >$ and $< 28, 30, 31, 33, 34 >$ are *distinct* but not *disjoint paths* whereas, $< 28, 35, 37, 38 >$ and $< 28, 36, 38 >$ are *distinct* as well as *disjoint paths*.

A gate G is a **reconvergence gate** of a fanout point x iff there are two distinct paths from x to G. A gate G is a **primary reconvergence gate** of a fanout point x iff exists two disjoint paths from x to G. The fanout point x is called **reconvergent fanout point**. In Figure 1, 34, 38 are reconvergence gates of 28, 38 is a primary reconvergence gate of 28 but 34 is not. 28 is a reconvergent fanout point.

If there are no reconvergent fanout points on the paths between a fanout point x and its primary reconvergence gates, then x is called a **narrow**

fanout point. Otherwise, it is called a **wide fanout point**. In Figure 1, 28 is a narrow fanout point whereas 14 is a wide fanout point.

Let A be a narrow fanout point. If A lies on one or more path(s) between a fanout point B and its primary reconvergence gates then all the primary reconvergence gates of A that are not primary reconvergence gates of B are **secondary reconvergence gates** of B. Also, if a fanout point X lies on a path between a fanout point Y and its primary reconvergence gates then all the primary and secondary reconvergence gates of X that are not primary reconvergence gates of Y are **secondary reconvergence gates** of Y.

In Figure 1, 14 is a wide fanout point. However, since 29 has no primary reconvergence gates 14 does not have any secondary reconvergence gates. Actually, none of the fanout points in Figure 1 have any secondary reconvergence gates.

A primary or secondary reconvergence gate X of a fanout point A that does not drive another primary or secondary reconvergence gate of A is a closing reconvergence gate for A. For example, in Figure 1, the set of closing reconvergence gates of the reconvergent fanout point 28 is 38.

The **stem region** of a reconvergent fanout point A is the subcircuit which consists of nodes that satisfies both (i) and (ii):

(i) the node is reachable from A;
(ii) a closing reconvergence gate of A is reachable from the node.

The stem regions of the reconvergent fanout point 28 of the circuit of Figure 1 are 35,36,37,and 38.

Let X be a line in the stem region of a fanout point A. Then, X is an **exit line** of A if X is a primary output or X satisfies both (i) and (ii):

(i) X is the output of a gate in the stem region of A;
(ii) X is an input of a gate that is not in the stem region of A.

The exit lines of the reconvergent fanout point 28 of Figure 1 are $< 28, 30 >$ and 38.

Corollary 2.4. If x and y, $x \neq y$, are exit lines of a reconvergent fanout point A, C_x, the circuit subnetwork driven by x, and C_y, the circuit subnetwork driven by y, are disjoint.[6]

Corollary 2.4 is important for computing parities of each line. We will use the algorithms discussed in [6], along with the modifications suggested in [7], to compute stem regions and exit lines.

3 Computing Equivalence Classes

In this section, we discuss how to compute the equivalence classes of stuck at faults in a given circuit w.r.t a given test set T. *Parity Labeling* will also be discussed.

3.1 Overall Structure

Given an input vector t_i, assume that $Parity(x, t_i) \quad \forall x,\ t_i, \quad x \in C, \quad t_i \in T$ are known. Then using observation 2.1, equivalence classes can be computed step by step as follow.

Assume that the set of equivalence classes $F^j = \{F_i^j\}$, for test vectors $t_1, t_2, \ldots, t_j$, and Parity(x, t_{j+1}) $\forall\ x\ \in\ C$ have been computed. Note that if $x \neq y$ and $G^{j+1} = \{G_l^{j+1}\}$ are equivalence classes over F w.r.t. $\{t_{j+1}\}$, then $\exists l$ s.t

$$f_x \in G_l^{j+1}\ and\ f_y \in G_l^{j+1} \Longleftrightarrow r_{j+1}^{f_x} = r_{j+1}^{f_y} \tag{1}$$

The equivalence classes over F after test vectors $t_1, t_2, \ldots, t_j, t_{j+1}$ are processed are

$$F^{j+1} = \{F_i^j \bigcap G_l^{j+1}\} \quad \forall i,\ l, \quad F_i^j \in F^j, \quad G_l^{j+1} \in G^{j+1} \tag{2}$$

We should be careful when we compute F^{j+1} using (2). A straight forward application of (2) leads to an inefficient algorithm. However, note that for each $F_i^{j+1} \in F^{j+1}$ there exists one response, that is $S^f = S^g\ \forall\ f, g \in F_i^{j+1}$. Therefore, sorting each F_i^j w.r.t $r_{j+1}^{f_x}$, $f_x \in F_i^j$ will group the partitions of F_i^j together. Since the responses are m-tuples over $\{0,1\}^m$ therefore we can use radix sort[12], that efficiently sorts a set of bit vectors, to speed up this step. This has the same effect as the computation implied by (2). This point is illustrated in the next subsection.

The algorithm for computing equivalence classes is as following:

Equivalence_classes(C,T)
Begin
Stem_Region_Extrac; / extracts stem regions to be used by Parity-Labeling */*
$F^0 = \{\{the\ set\ of\ all\ single\ s\text{-}a\text{-}f\ in\ C\}\};$

For each test vector $t_i \in T$

Begin

C*=Parity_Labeling(*t_i*);*

F^i*=Refine_F(*F^{i-1}, G^{t_i}*); /* formula (2) */*

End

End

In the next subsection, we present an example to illustrate the procedure.

3.2 An Example

The circuit C is given in Figure 2, set of test vector T is given in Table 1. Assume Parity(x, t_i) for all x in C are know(Table 2). Initially, F^0 equals $\{F\}$.

$$F^0 = \{\{1_0, 1_1, 2_0, 2_1, 3_0, 3_1, 4_0, 4_1, 5_0, 5_1, 6_0, 6_1\}\}$$

On application of t_1, faults 1_0, 2_1, 3_1, 4_0, 5_0, 6_1 are undetectable (response $< 0, 1 >$) because the values of these lines are the same as their stuck at values. From Table 2 we see that the parities of lines 2 and 3 are $< N, N >$, so faults 2_0 and 3_0 also have the responses $< 0, 1 >$. For the rest of the faults, Parity$(1, t_1) =< E, N >$ and Parity$(5, t_1) =< O, N >$. From observation 2.1 the responses of faults 1_1 and 5_1 are $< 1, 1 >$. Parity$(4, t_1) =< N, O >$ and Parity$(6, t_1) =< N, E >$, implies that the responses of faults 4_1 and 6_0 are $< 0, 0 >$. Sorting fault list w.r.t their responses, we have $F^1 = \{\{4_1, 6_0\}, \{1_0, 2_0, 2_1, 3_0, 3_1, 4_0, 5_0, 6_1\}, \{1_1, 5_1\}\}$.

Next t_2 is processed. Fault 4_1 has response $r_2^{4_1} =< 0, 1 >$. Fault 6_0 has response $r_2^{6_0} =< 0, 0 >$. Faults $1_0, 2_0, 3_0, 4_0, 5_0, 6_1$ have response $< 0, 1 >$. Faults 2_1 has response $< 1, 0 >$. Fault 3_1 has response $< 0, 0 >$. Fault 1_1 has response $< 0, 1 >$. And fault 5_1 has response $< 1, 1 >$. The lists $\{4_1, 6_0\}, \{1_0, 2_0, 2_1, 3_0, 3_1, 4_0, 5_0, 6_1\}$, and $\{1_1, 5_1\}$ are sorted and partitioned. we have $F^2 = \{\{6_0\}, \{4_1\}, \{3_1\}, \{1_0, 2_0, 3_0, 4_0, 5_0, 6_1\}, \{2_1\}, \{1_1\}, \{5_1\}\}$.

Continue to apply test vector t_3. Faults $1_1, 2_1, 3_1, 4_1, 5_1, 6_0$ have response $r_3^\phi =< 1, 0 >$. Faults $1_0, 5_0$ have the response $< 0, 0 >$. Faults $3_0, 4_0, 6_1$ have response $< 1, 1 >$. Fault 2_0 has response $< 0, 1 >$. Sorting each list in set F^2 w.r.t their responses, we have

$$F^3 = \{\{6_0\}, \{4_1\}, \{3_1\}, \{1_0, 5_0\}, \{2_0\}, \{3_0, 4_0, 6_1\}, \{2_1\}, \{1_1\}, \{5_1\}\}$$

There are 9 equivalence classes. From this we have $DP = \frac{7}{12}$ and $DR = \frac{62}{66}$. 7 faults are fully distinguishable, 2 faults fall into one class and 3 faults fall into another class — a very clear picture of the diagnostic capability of the test set.

3.3 Parity Labeling

Procedure *Parity_Labeling* labels all lines in C, on application of input vector t_i, with Parity(x, t_i). For example, in Figure 2, let the input vector be $(0, 1, 0)$. Then, Parity$(1, t) = (E, N)$, indicating that t can propagate a D from line 1 to y_0. However, a D cannot be propagated by t from line 1 to y_1. Parities of other lines are: Parity$(2, t) = (N, N)$, Parity$(3, t) = (N, N)$, Parity$(4, t) = (N, O)$, Parity$(5, t) = (E, N)$ and Parity$(6, t) = (N, E)$.

Parities for lines in a fanout-free circuit can be computed as follows. Assume parity of primary outputs are E. Starting from the primary outputs, traverse towards the primary inputs. When a gate G is encountered, we compute the parity of its inputs. Let x be an input of G. Check if the input vector assigns sensitizing value to offpath inputs of G or not. If not, parity of x is N, otherwise it is E or O, depending on gate type and parity of gate output. As an example, Figure 3 shows the labeling of a fanout-free circuit. Note that since Parity$(5, t) = (N)$, all gates before line 5 can not propagate anything to primary output and can be labeled N.

For every line x, FAN(x) is the primary output or fanout point y s.t. there exists a path p from x to y and p does not pass through any other fanout point. If x is a fanout point then FAN$(x) = x$. In Figure 1, FAN$(18) = 25$, FAN$(27) = 27$.

If C is not fanout-free but a reconvergence-gate-free circuit we can break C into subcircuits such that lines with same FAN(x) are in same subcircuits. The inputs of the subcircuits are either primary inputs or fanout branches and the outputs are FAN(x). For example, the circuit of Figure 4 can be broken into the subcircuits shown in Figure 5.

Each subcircuit can be labeled separately. Since each line knows its fanout point FAN(x), parities of each primary output can be computed very quickly by using $*$ operation, defined below.

A $*$ operation over (E, O, N) is defined as

$$a * b = \begin{cases} E & if\ a = E\ and\ b = E,\ or\ a = O\ and\ b = O \\ O & if\ a = E\ and\ b = O,\ or\ a = O\ and\ b = E \\ N & if\ a = N\ or\ b = N \end{cases}$$

It concatenates the parity along a path. For example, parity from a to b is E and from b to c is O, then parity from a to c is $E * O = O$.

In Figure 5, the parity from 5 to 10 is E, and parity from 10 to 12 is O, from 10 to 13 is E, so parity from 5 to 12 is $E * O = O$ and from 5 to 13 is $E * E = E$. There is no path from 10 to 11. Hence Parity$(5, t) = (N, O, E)$, assume y_1 is 11, y_2 is 12, and y_3 is 13.

If C contains reconvergence gate(s), it can not be simply broken into subcircuits as before, and a one pass backward labeling is not enough either. For example, in Figure 6, two branches of a fanout point lead to one gate, it can not be broken into two subcircuits. Also, if we simply traverse backward, we would have the label shown in Figure 6. But in fact, as shown in Figure 7, a D on line 2 can be propagated to its output. The reason for this is that gate 3 is a reconvergence gate, single stuck at fault affects both of its inputs.

Reconvergence gate only affects the parity of its reconvergent fanout points. To compute the parities of reconvergent fanout points, stem regions are extracted. The circuit is broken into stem regions and reconvergence-gate-free subcircuits. For reconvergence-gate-free subcircuit, parity labeling is done as stated above. For stem region, parity labeling is same as that reconvergence-gate-free subcircuit except for the reconvergent fanout points. For the reconvergent fanout points forward labeling, which starts from the fanout point and ends at exit lines is used. For example, in Figure 7, line 2 is a reconvergent fanout point, its exit line is 5. Initially, all lines in the stem region (3, 4, 5) are labeled N. Then, label 2 by E. Start from 2 and traverse the stem region of 2 breadth first. The input vector $t = (1, 1)$ assigns sensitizing value to the offpath inputs w.r.t line 2 of both gates 1 and 2. These off-path inputs are not in the stem region, therefore their outputs are labeled E. Since both inputs of gate 3 are labeled E and it is a NAND gate, its output is labeled E.

A formal description of the procedures for forward labeling and backward labeling are given in [5] (*PathParity* and *MarkLines* respectively). A procedure for region extraction has been discussed in [6, 7]. Procedure *Parity_labeling* uses these procedures to label all lines in C.

Procedure Parity_Labeling
/ Given t, C it computes Parity(x,t) for all $x \in C$.*/*
Step 1. Use PathParity to label each reconvergent fanout point x with Parity(x,t);
/ Start from x and use forward labeling */*
Step 2. Use MarkLines to mark each line x with the parity P_x of the path from x to FAN(x).
Step 3. Use * operation to concatenate parities as follows. Parity(x,t) = $P_x *$Parity(Fan$(x), t$). */* Note that Parity(Fan$(x),t)$ $\forall x$ has been computed in step 1 and P_x has been computed in step 2. */*

Note that steps 2 and 3 can be combined. This is done in our implementation.

3.4 Complexity Analysis

Let n_i be the the size of C, n_f be the number of reconvergent fanout point(s) in C, n_t be the number of test vector(s) in T and m the number of output(s) of C. We first determine the complexity of *Parity_Labeling*. In step 1, *PathParity*, for each reconvergent fanout point, traverses its stem region once. Therefore it has a complexity of $O(n_i \times n_f)$. In step 2, *MarkLines* traverses each line in C once. Therefore its complexity will be $\Theta(n_i)$. In step 3, the * operation takes $O(m)$ for each line. Total time taken by *Parity_Labeling* will be $O(n_i \times n_f + n_i \times m)$.

Next we determine the complexity of computing Equivalence Class. Let m_f denote the number of single stuck at faults in C. *Refine_F* is implemented by sorting as discussed previously. Note that responses are vectors over $\{0,1\}^m$. Therefore, we can use radix sort[12]. This step has complexity $O(m \times m_f)$.

The complexity of *stem_region_extract* is $O(n_i^2)$[6, 8]. Total time for this algorithm will be

$$O(n_i \times n_f \times n_t + m \times n_i \times n_t + m \times m_f \times n_t + n_i^2).$$

Note that this will always be better than $O(n_i^2 \times n_t)$.

4 Experimental Results

We have implemented the algorithm presented in this paper in C. ISCAS85 and ISCAS89 combinational benchmark circuits were used as target circuits in our experiment. The test sets were randomly generated. Experiments were conducted on a Sun SPARC station 2 with 32MB of memory.

The performance of the implementation is shown in Table 3. For each circuit, the table gives the number of faults, the number of test vectors and the execution time(in CPU seconds).

The diagnostic resolution provided by the test set are shown in Table 4. Equivalence classes are reported in order of its size. For example, the number of equivalence classes of size 1 of circuit C432 is 260.

We compare these results with results from [3]. Note that the same number of tests were used in both cases. They are shown in Table 5. Note that the times are CPU times in a sun 4/70 with 64MB of memory. Times for larger circuits like C5315, C6288, C7552 as well as some circuits in ISCAS89 were not reported. The ratio gives the speed of our algorithm compared to the one in [3]. This large speedup cannot be accounted for by the faster CPU speed of SPARC station 2.

5 Conclusion

We showed that by modifying stuck-at fault simulation algorithms for single stuck-at faults we can derive a time and space efficient algorithm for diagnostic. Experiments using this algorithm show that it is more efficient than existing algorithm.

References

[1] P. Camurati, D. Medine, P. Prinetto, and M. S. Reorda, "A diagnostic test pattern generation algorithm," *Proceedings of the International Test Conference*, pp. 52-58,1990.

[2] T. Grüning, H. Koopmeiners, and U. Mahlstedt, "DIAGTEST: A fast diagnostic test pattern generator for combinational circuits", *Digest of*

Papers, International Conference on Computer-Aided Design, pp. 194-197, 1991.

[3] K. Kubiak, S. Parkes, W. K. Fuchs, and R. Saleh, "Exact Evaluation of Diagnostic Test Resolution," *Proc. 29th Design Automation Conf.*, June 1992, pp. 347-352.

[4] Elizabeth M. Rudnick, W. Kent Fuchs, and Janak H. Patel, "Diagnostic Fault Simulation of Sequential Circuits", *Proc. Intn'l Test Conf.*, 1992, pp. 178-186.

[5] Sreejit Chakravarty and Yiming Gong, "An Algorithm for Diagnosing Two-Line Bridging Faults in Combinational Circuits", *Tech. Report 92-26, Dept. of Computer Science, the State University of New York, Buffalo,N.Y. 14260.*

[6] F. Maamari and J. Rajski, " A Method of Fault Simulation Based on Stem Regions", *IEEE Trans. on Comput.-Aided Design*, Vol. CAD-9, No. 2, pp. 212-220, Feb. 1990.

[7] Hyung K.Lee and Dong S.Ha, "Comments on "A Method of Fault Simulation Based on Stem Regions"", *IEEE Trans. on Comput-Aided Design*, Vol. 11, No. 3, pp. 407-408.

[8] D. Harel and B. Krishnamurthy, "Is there hope for linear time fault simulation?." in *Proc. 17th Fault Tolerant Computing Symp.*, pp. 28-33,1987.

[9] T. M. Niermann, W. -T. Cheng, and J. H. Patel, "PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator," *IEEE Trans. CAD*, pp. 198-207, Feb 1992.

[10] E. M. Rudnick, T. M. Niermann, and J. H. Patel, "Methods for Reducing Events in Sequential Circuit Fault Simulation," *Proc. Intn'l Conf. Computer-Aided Design*, 1991, pp. 546-549.

[11] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," *Special Session on ATPG and Fault Simulation, International Symposium on Circuits and Systems*, 1985.

[12] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms", *McGraw-Hill*, Second printing 1990.

Table 1

	1	2	4
t_1	0	1	0
t_2	1	0	1
t_3	1	1	1

Table 2

	Parity$(1, t_i)$	Parity$(2, t_i)$	Parity$(3, t_i)$	Parity$(4, t_i)$	Parity$(5, t_i)$	Parity$(6, t_i)$
t_1	(E,N)	(N,N)	(N,N)	(N,O)	(E,N)	(N,E)
t_2	(N,N)	(E,O)	(N,O)	(N,N)	(E,N)	(N,E)
t_3	(E,N)	(E,O)	(N,O)	(N,O)	(E,N)	(N,E)

Table 3			
Circuit	Faults	Vectors	Time(s)
C432	878	102	6.98
C499	1062	171	22.35
C880	1812	131	21.98
C1355	2774	169	63.28
C1908	3862	116	81.08
C2670	5616	249	560.98
C3540	7118	303	525.97
C5315	10876	350	2006.65
C6288	12640	350	2584
C7552	15320	350	2771.97
s1196	2456	500	159.866667
s1238	2540	478	160.983333
s13207	27938	500	35126.950000
s1423	3004	88	74.933333
s1488	3026	500	144.083333
s1494	3038	500	149.283333
s15850	33062	500	38252.450000
s298	636	287	11.166667
s344	722	109	6.366667
s349	732	77	4.583333
s526	1106	34	2.716667
s5378	11046	500	4975.816667
s641	1364	219	38.516667
s713	1510	175	34.916667
s820	1688	1135	135.050000
s832	1712	1027	121.950000
s9234	18968	500	7947.950000
s953	2010	500	173.500000

Table 4										
	Number of equivalence classes									
Circuit	1	2	3	4	5	6	7	8	9	> 9
C432	260	65	32	14	14	13	1	2	6	5
C499	552	52	53	11	2	-	-	13	-	2
C880	453	71	155	31	41	6	9	1	2	11
C1355	212	170	82	175	86	-	27	43	-	3
C1908	253	330	83	214	14	9	4	38	7	27
C2670	670	557	240	251	66	41	25	19	3	15
C3540	1108	717	197	406	103	40	6	23	15	43
C5315	2030	1088	513	893	123	72	9	6	3	14
C6288	3343	957	2316	49	33	1	-	-	-	1
C7552	2025	1356	950	1092	165	81	29	20	9	15
s298	138	57	37	28	16	7	2	2	1	-
s344	137	93	42	39	17	3	2	-	-	-
s349	136	92	39	38	12	5	2	2	-	1
s526	138	62	49	36	11	9	2	5	2	3
s641	146	56	93	21	30	17	30	6	3	12
s713	137	56	86	24	28	13	24	3	9	22
s820	453	73	73	34	26	5	12	12	8	12
s832	452	67	73	33	22	8	17	12	7	12
s1238	507	137	183	89	25	14	6	5	3	3
s1423	458	265	181	77	41	11	41	4	14	6

Table 5									
	C432	C499	C880	C1355	C1908	C2670	C3540	s298	
Time	4m03s	26m30s	12m36s	37m15s	35m59s	3h18m41s	2h56m29s	3m30s	
Ratio	34.81	71.14	34.39	35.32	26.63	21.25	20.13	18.0	
	s344	s349	s526	s641	s713	s820	s832	s1238	s1423
Time	2m00s	1m44s	17s	6m21s	7m09s	1h55m23s	1h48m90s	46m28s	2m56s
Ratio	18.84	22.66	6.25	9.89	12.29	51.26	53.87	17.32	2.34

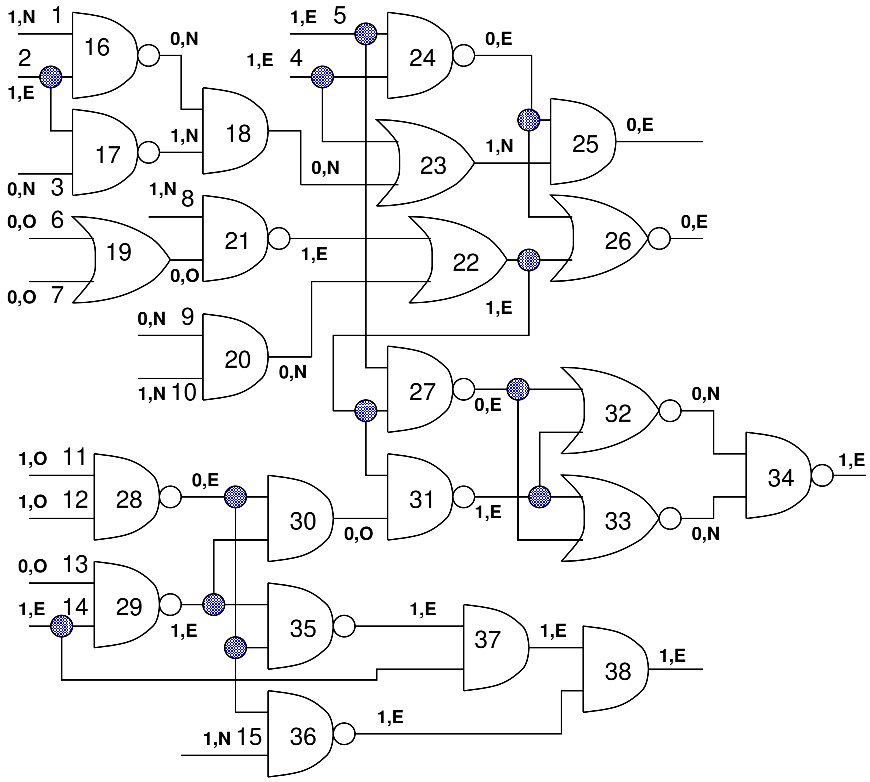

Figure 1:

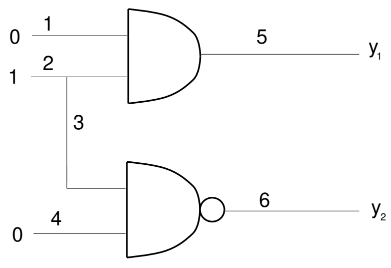

Figure 2:

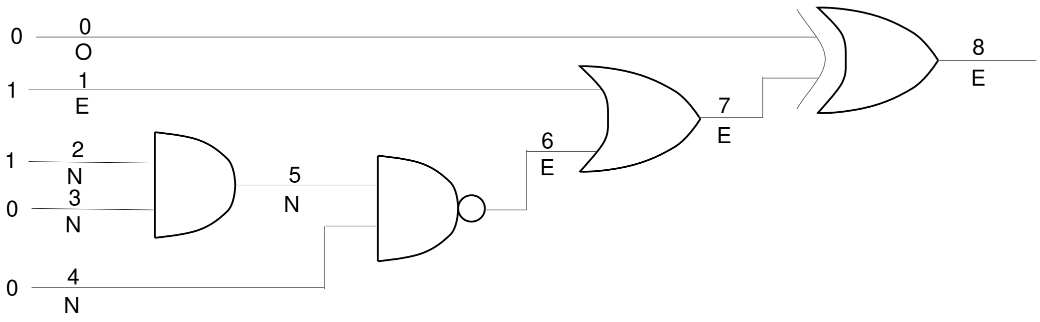

Figure 3:

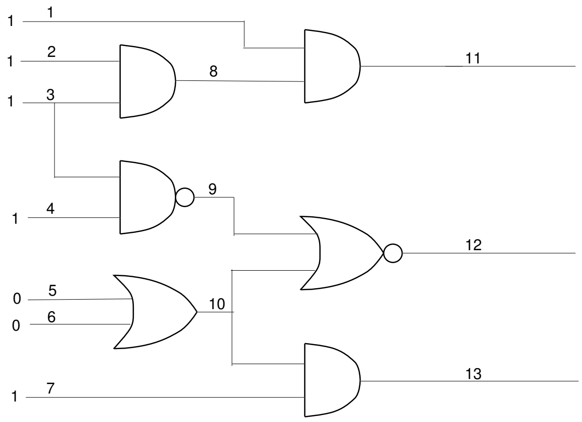

Figure 4:

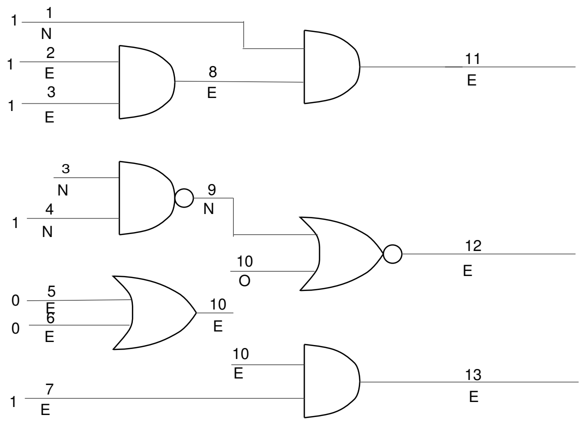

Figure 5:

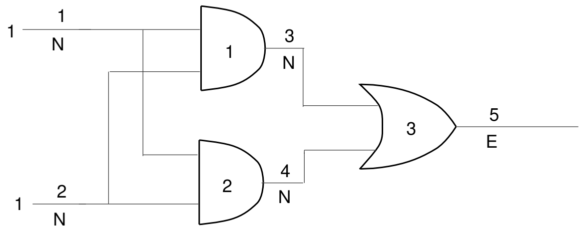

Figure 6:

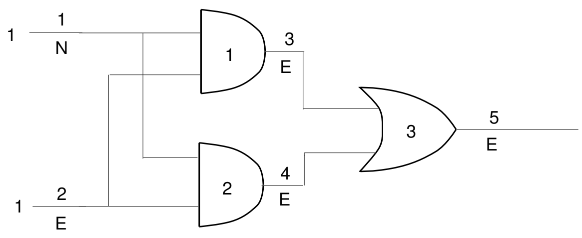

Figure 7: