

# An Efficient and Effective Methodology on the Multiple Fault Diagnosis

Zhiyuan Wang<sup>1</sup>   Kun-Han Tsai<sup>2</sup>   Malgorzata Marek-Sadowska<sup>1</sup>   Janusz Rajski<sup>2</sup>

<sup>1</sup>Department of Electrical and Computer Engineering  
University of California, Santa Barbara, CA, 93106

<sup>2</sup>Mentor Graphics Corporation  
Wilsonville, OR, 97070

## Abstract

*In this paper, we analyze failing circuits and propose a multiple-fault diagnosis approach. Our methodology has been validated experimentally and has proved to be highly efficient and effective in diagnosing multiple faults. We do not consider the multiple-fault behavior explicitly, but rather use an incremental simulation-based approach to diagnose failures one at a time. Furthermore, to improve the diagnosability, we propose a failing-primary-output partitioning algorithm. Experimental results show that our approach has approximately linear time complexity, and it achieves high diagnosability and resolution. Our approach has also been validated on data collected from manufactured chips. The diagnosis time is within minutes for real industrial chips that failed because of multiple faults.*

## 1. Introduction

The purpose of fault diagnosis is to determine the cause of failure in a manufactured, faulty chip. A good diagnostic tool should effectively assist a designer in quickly and accurately locating the reason for failure. The quality of a diagnosis impacts directly the time-to-market and the total product cost. Most of the studies on failure analysis have assumed a single defect. However, for present technologies and chip sizes, this assumption may not be true. Multiple defects on a failing chip often better reflect the reality. It is also possible that certain single-location defects behave as multiple faults. Defects of this kind are bridging defects which affect two fault locations, or via defects which may affect multiple branches and which can be modeled as multiple faults at those branches.

Diagnostic algorithms can be quantified by several measures. *Resolution* of an algorithm is measured as a ratio of true faults and the total number of reported candidates. *Diagnosability* of an algorithm is a measure reflecting the percentage of defects which can be correctly identified.

The existing diagnosis algorithms can be divided into two main classes. The first one applies the cause-effect principle [13]. The second one traces the effect-cause dependencies [4][6][8][12]. The former methods build the simulation

response database for the modeled faults and compare this database with the observed failure responses to determine the probable cause of the failure. This method is sometimes referred to as the *fault dictionary method*. For the assumed fault model, whose defect behavior is similar to the modeled fault behavior, this method can give a very good resolution. Otherwise the resolution may be drastically reduced. However, because this method requires a huge fault behavior database, it may have difficulty with large designs.

The effect-cause-based algorithms analyze the actual responses and determine which fault(s) might have caused the observed failure effect. This class of methods does not build the fault-response database. They trace backward from each primary output to determine the error-propagation paths for all possible fault candidates. Compared with the cause-effect methods, effect-cause techniques are more memory-efficient and can cope with larger designs.

Although most of the published papers on diagnosis describe methods which explicitly or implicitly assume existence of a single fault, some research on multiple-fault diagnosis [2][4][5][9] [11][14][15][18] has been published. In [15] the authors propose generating test patterns which improve the multiple-fault diagnosis resolution. However, in practice, a customer may use for diagnosis only the same test patterns which were generated by an ATPG tool to test single stuck-at-faults. Generating new patterns for diagnosis may not be an acceptable solution. Some other relevant research can be found in [4][5][9][12][18]. In [18], the authors first apply a path-tracing algorithm[17] and find some sites with possible candidate faults. Then they sequentially try each candidate site in the circuit, simulating about 6K-10K random vectors along with the test vectors from [7] to see how many failing POs can be corrected by each candidate simulation. The candidates are ranked by their correcting potential. After some good candidates have been determined, they are injected into the circuit, and the iteration proceeds. This method requires exhaustive searching and comparisons, which may be impractical for big industrial designs. In [14], the authors proposed a simulation-based method (multiple-fault and single-fault simulation) for multiple-fault diagnosis. All faults which could be detected by the failing patterns are the initial fault candidates. Then the algorithm uses the results of each passing

pattern simulation to remove the impossible candidates and construct the most probable multiple fault groups. However, this method requires many fault simulations (long diagnostic time) and produces too many final fault candidate sites.

Due to limited tester storage space, we cannot store many failing patterns for the purpose of future diagnosis. Normally, fewer than one hundred failing patterns can be stored. If we address directly the multiple fault problem, the number of suspected faults grows exponentially with the number of defects: error space = (# of lines)<sup>(# of defects(errors))</sup> [20]. In [4], the authors assume that the multiple design-errors are caused by close proximity defects, and they use three-value simulation (0,1,X) to locate them. This method is very efficient in computation time and has good resolution when the defects are grouped into clusters. But in big designs, defects may be widely scattered and unrelated. In [5], the authors address the problem of multiple defects from the design error point of view. They proposed three error models (independent, simple merging, and simple successive) to model the multiple errors. When the errors in the design are few (two in their experimental results) and errors behave like at least one of their error models, their methods are very efficient. However, they have difficulties in handling large designs when errors have complex dependencies. In [9], the author uses two matching mechanisms (vector match and failing primary output match) to rank each possible candidate error. When a circuit does not have a single-fault behavior, then every pair of signals in the circuit is considered. But for big designs affected by multiple faults, exhaustive enumeration of all fault combinations is infeasible. In [12], the authors attempt to find a minimal set of modifications which correct the faulty design.

In recent work [2], the authors apply a single-location-at-a-time (SLAT) pattern to perform the diagnosis for failure responses caused by multiple faults. The SLAT pattern attributes each failure response to a single fault. Acting on this assumption, using SLAT patterns, the algorithm tries to find all single-fault locations. Each candidate can explain some failing patterns. Then after finding these candidates, the algorithm seeks to group them into a minimum cardinality groups that could explain all the failing patterns. The results in [2] show that many failing patterns are indeed SLAT patterns even though multiple faults are present in the circuit. However, in reality, due to limited tester storage space, we could obtain information on only a few failing patterns. When multiple faults are present in the circuit, not every fault could have a SLAT pattern.

In this paper we propose an incremental approach based on multiple-fault simulation for combinational and full-scan sequential circuits. Also, we analyze and compare our method against another incremental algorithm [18] to show the greater efficiency of multiple-fault diagnosis.

Our technique requires information concerning only a few failing patterns (typically only 30 to 50), to accurately diagnose the given chip failure responses. Utilization of the

*Detection Pattern Set* (the concept explained in section 2) and *Failing-PO Partition* algorithm (presented in section 3.3) can significantly help to avoid the inherent exponential time-complexity problem in multiple-fault diagnosis. We have developed a diagnostic framework which can handle any specific fault model and invoke any diagnostic algorithm. The experiments performed on two industrial circuits (100K and 300K gates) with multiple faults took only minutes.

The rest of the paper is organized as follows. In section 2, we analyze failing patterns. In section 3, we propose our diagnosis algorithm. In section 4, we discuss the limitation of our proposed algorithm, offer some solutions, and compare against the work proposed in [18]. In section 5, for industrial and ISCAS circuits we show experimental results in terms of diagnosability, resolution, and performance. Section 6 concludes the paper.

## 2. Failing Pattern Analysis

Suppose that  $T$  is a test set and  $f$  is a fault. Those patterns in  $T$  that can detect  $f$  will form the *detection pattern set of the fault  $f$* .

Any single-fault-based diagnostic algorithm uses single-fault simulation behavior to match the observed failure responses to the given test patterns. However, in reality, a pattern may activate multiple faults and create a multiple-fault behavior.

For the purpose of explanation, let us assume that only two faults exist in the circuit. Our analysis can be extended to situations with more than two faults.

Each failing pattern  $p$  in the given diagnostic test set  $T$  is classified into one of the following three types:

**Type-1:**  $p$  can activate only one fault and observe its effect. This type of pattern is also defined in [2] as a SLAT pattern.

**Type-2:**  $p$  activates both faults and observes their effects, but those effects are not correlated. We say that the faults have disjoint behavior on the pattern  $p$ .

**Type-3:**  $p$  activates both faults and observes their effects, but the faults interact on some primary outputs. The faulty effects may cancel each other out on some POs.

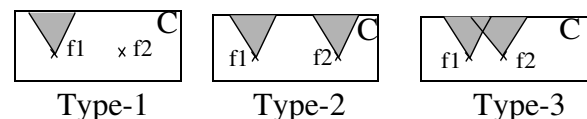


Fig. 1: Failing Pattern Types

Most of the published diagnostic algorithms are based on the single-fault assumption, no matter what kind of fault behavior they target (stuck-at, bridging[1], branch[16], transition[3], path delay...). These single-fault-based algorithms rely primarily on the type-1 patterns to perform the analysis and find the fault candidates. Each candidate can fully explain some of the failing patterns. If other faults are present in the circuit, they do not manifest themselves in the type-1 diagnostic test patterns. However, when the fault density increases, the probability decreases that the detec-

tion pattern set of every fault in the circuit contains a type-1 pattern. Furthermore, due to limited tester storage space, only very few failing patterns can be stored for future failure analysis. The probability is even lower that the detection pattern set of every fault has a type-1 pattern, which suggests that many faults may not be identified by a single-fault-based diagnosis algorithm.

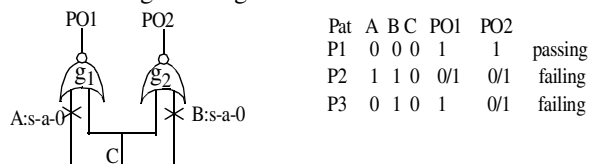


Fig. 2: Example of a single-fault assumption limitation.

Consider the example in Figure 2 which shows why an algorithm based on a single-fault-assumption may not work. Suppose we have 3 patterns, and pattern  $P_1$  is a passing pattern. Applying patterns  $P_2$  and  $P_3$  we observe some failing POs. The single-fault diagnosis algorithm applied on the failing pattern  $P_3$  will result in identifying a possible fault candidate B s-a-0, since the A s-a-0 fault is not activated by  $P_3$  (thus  $P_3$  is a type-1 pattern). This explains why single-fault assumption algorithms sometimes work even when multiple faults are present in the circuit. However, the failing pattern  $P_2$  (which is a type-2 pattern) cannot be explained by any single fault in the circuit. All single-fault-assumption-based algorithms will fail in this case. Only an algorithm using a partial-match mechanism could report A s-a-0 as another possible candidate[11][16].

Another important fact is that, in practice, we use a compact test pattern set to shorten the expensive test application time. If multiple faults occur and a compact test pattern set is used for testing and diagnosis, the majority of patterns will be types 2 and 3. For example, benchmark s35932 has only 21 test patterns for testing and diagnosis purposes. Patterns of types 2 and 3 usually happen when multiple faults are present in the circuit. In general, using compact test sets for diagnosis creates problems for single-fault-based algorithms.

Our work in this paper addresses the different failing pattern types mentioned above. If all the failing patterns available for diagnosis are of type-1, the situation can be handled by any single-fault-based diagnosis algorithm. For type-2 patterns, we propose the failing-PO partitioning technique explained in section 3.3. It changes type-2 patterns into type-1 patterns, allowing for easy handling of this situation without information loss. In the case of type-3 patterns, our incremental technique works when some of the faults activated by the type-3 pattern are also activated by type-1 or type-2 patterns. If there are no type-1 or type-2 patterns, the failing-PO partitioning algorithm, in most cases, still finds the real fault locations.

### 3. Proposed Algorithm

#### 3.1 Diagnosis Algorithm in [22]

Since we will be referring often to the diagnosis algorithm in [22], we describe it briefly here. This algorithm will be designated from here on as  $Alg_{single}$ .

1. Initialize the fault candidate list using path-tracing technique. Initial fault candidates satisfy the following requirements to reduce the search space and improve diagnosis efficiency:
  - 1.1 The fault must reside in the input cone of a failing PO of the given pattern.
  - 1.2 There must exist a parity-consistent path from the failing site to the failing PO.
  - 1.3 If a failing pattern affects more than one PO, the candidate fault must reside in the intersection of all the input cones of those failing POs (single fault per pattern assumption).
2. Simulate each fault on the initial candidate list and see if it explains perfectly any of the failing patterns. If it does, store it as a candidate fault, assign to it a weight equal to the number of patterns it explains in the current list, and remove the explained failing pattern(s).
3. After explaining the entire failing-pattern list, or when all faults in the initial list have been examined, the algorithm terminates and reports the possible candidate sites. Candidate faults are sorted by their weights. The fault with the greatest weight is reported first.

Note that even when the algorithm is based on a single-fault-per-pattern assumption, it is capable of identifying multiple fault candidates as long as the failing pattern is affected by one fault only (i.e. type-1 pattern).

#### 3.2 Proposed Diagnosis Algorithm

**Observation:** Type-1 failing pattern very often exists even if multiple-fault failure responses are also present.

The observation above implies that in most multiple-fault failure cases, some fault locations can be identified just by the single-fault diagnosis algorithm. Our algorithm is based on this observation and on the analysis in section 2. Instead of considering all possible multiple-fault combinations directly, which would be computationally very time-consuming and impractical for large designs, we isolate the problems and deal with them incrementally. For example, in figure 2, if we already knew (from analyzing failing pattern  $P_3$ ), that B s-a-0 is the most probable fault in the circuit, we could simply change the circuit by connecting net B to ground. Then we could perform analysis of the failing pattern  $P_2$  and repeat the single-fault diagnosis. It is obvious that when we do so, any single-fault-seeking algorithm would find the second fault candidate A s-a-0, since now pattern  $P_2$  has become a type-1 pattern in the modified circuit. In our algorithm, we also attempt to manage the complexity of multiple-fault diagnosis. In [18], the authors propose an incremental method to perform the error correction and stuck-at-fault diagnosis. The potentially good candidates are those which can explain the greatest number of failing POs for all the failing patterns. However, the authors do not consider those cases when a potential candidate fault causes some passing POs to fail, even if it explains the larg-

est number of failing POs. In our method, a fault (or a group of faults) can be a candidate only if its (their) simulation results can fully explain at least one failing pattern. And if more than one candidate can explain the same failing patterns, we use passing-pattern information to rank the candidates and to select the best among them.

### 3.2.1 $n$ -perfect Algorithm

**Definition 1:  $n$ -perfect candidate:** Suppose that after injecting  $n$  faults into a circuit and running multiple-fault simulations, we can perfectly explain some failing patterns. These  $n$  faults as a group are called the  $n$ -perfect candidate for those explained patterns. If  $n$  is equal to 1, we have a single-fault candidate.

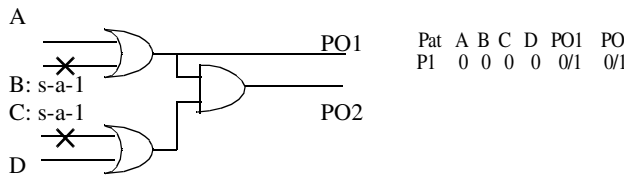


Fig. 3: Example of an  $n$ -perfect candidate.

Consider the example in figure 3. There is no single fault that can explain the observed failing responses, but B s-a-1 and C s-a-1 as a group can explain the observed failure. Thus {B s-a-1, C s-a-1} form a 2-perfect candidate.

The algorithm proceeds as follows.

1. Find a 1-perfect fault candidate(s):  
 $n = 1$ . Apply  $Alg_{single}$  (refer section 3.1). Eliminate the explained patterns from the failing patterns.
2. For each  $n$ -perfect candidate, inject it into the circuit and perform steps 3 and 4 until all  $n$ -perfect candidates have been tried.
3. For each unexplained failing pattern, initialize the possible fault candidates. We use a technique similar to that in [22] with some modifications in obtaining the initial candidate list, as discussed in section 3.2.2.
4. Perform  $Alg_{single}$  on the modified circuit and construct  $(n+1)$ -perfect candidates based on the targeted fault model.
5. Determine the  $(n+1)$ -perfect candidates, which can further explain some failing patterns not yet explained by those (1 through  $n$ )-perfect candidates.
6. Rank and weight the  $(n+1)$ -perfect candidates based on failing and passing pattern information. Eliminate those failing patterns which can be explained by  $(n+1)$ -perfect candidates from the failing pattern list. Increase  $n$  by 1.
7. Perform steps 2-6 for the remaining unexplained failing patterns until no fault candidate can be found, or until all failing patterns have been explained.
8. Post-process all possible  $k$ -perfect candidates ( $k$  is an integer between 1 and  $n$ ) to eliminate the candidates which cause many passing patterns to fail when multiple-fault candidates are injected into the circuit. This is a refinement step. Its purpose is to eliminate the cases of wrong candidates assumed in the very beginning, which may imply other wrong candidates.

### 3.2.2 Discussion

In the proposed algorithm, we use a weighting mechanism similar to that of [16] to rank potential candidates. For each

$n$ -perfect candidate, we use “match,” “mis-predict,” and “non-predict” weights.

In addition to the failing patterns, we simulate passing patterns. Each  $n$ -perfect candidate fault has 2 weights. The first weight reflects how well it explains the failing patterns, the second how well it explains the passing patterns. We use the failing pattern weights greedily to find possible candidates. Then, for those candidates we compute a combined weight, order them, and drop the worst candidates.

In the step initializing the fault candidates, we use a technique similar to that of [22] which allows us to reduce significantly the size of our initial fault candidate list. However, since we are constructing incrementally the multiple-fault behavior, we make some modifications in generating the initial fault candidate. First let's consider the simple example in figure 2:

From the failing pattern  $P_3$  we find B s-a-0 as a 1-perfect candidate. Since we still have an unexplained failing pattern  $P_2$ , we inject B s-a-0 into the circuit and perform the single-fault diagnosis again. However if we directly apply the technique in [22], we would back-trace from  $PO_1$  and  $PO_2$ , find the initial fault lists, and then intersect them. However, this intersection could be empty. To overcome this problem, we first record the effect of injecting faults (i.e. B s-a-0) on this failing pattern, and then we ignore all POs which are affected by injecting those faults. We do back-tracing on the remaining failing POs and perform cone intersection to determine the next possible fault candidates.

Our proposed algorithm and the majority of other published diagnosis algorithms proceed from this basic assumption: “Use the minimum number of faulty locations to explain as many failing responses as possible.” The probability that a failure is caused by one faulty site, or even by a few, is greater than the probability of multiple sites failing simultaneously. So the algorithm always starts with a single-fault candidate before moving toward multiple-fault locations. It may happen that the initial fault list does not include any of the injected faults, which means we have a wrong candidate at the beginning. If multiple faults behave exactly like a single fault, no algorithm can distinguish that group from an equivalent single fault candidate. On the other hand, if the fault candidates are incorrectly identified at the beginning due to our incremental heuristic approach, most of them will be pruned out after our failing- and passing-pattern simulation with the help of the weighting mechanism at the final step (step 8).

### 3.3 Failing-PO Partitioning Technique

Again we consider the example in figure 2. But this time suppose we have one less pattern, and we consider only stuck-at faults.  $P_1$  is a passing pattern, and  $P_2$  is the only failing pattern we have in this case. No single fault can explain the fault behavior. The single-fault-assumption algorithm will stop and report an inconclusive result because the pattern set for diagnosis is composed of type-2 patterns. However, if we partition the failing POs into

groups, we can easily find the fault candidates. After proper partition of failing POs, each group of failing POs captures the faulty behavior of one or a very small number of faults, such that for each fault and partitioned failing-PO group, the patterns in the pattern set become type-1 patterns, an easy case to diagnose. In this example, we partition POs into 2 groups:  $PO_1$  is in one group,  $PO_2$  in the other. Then we perform the diagnosis for each failing-PO group separately on each failing pattern. For the failing pattern  $P_2$  and the group  $\{PO_1\}$ , we find A stuck-at 0 as a candidate. Similarly, we use the pattern  $P_2$  and the  $\{PO_2\}$  group to get another fault candidate, B stuck-at 0. After finding some candidates, we perform our algorithm again, obtain more candidates, and refine the results.

### 3.3.1 Failing-PO Partitioning Algorithm

This partitioning technique is very useful especially for the current big industrial designs, most of which are full-scanned and very flat, with a large number of observation points. It very often happens that different faults have disjoint observation points (type-2) or share only a few of them. Our goal is to identify those type-2 and type-3 failing patterns with few overlapped failing POs. Then, partition the failing POs into groups such that each of them is affected by only one or few faults. After the partitioning, a majority of the type-2 and type-3 patterns are transformed into several type-1 patterns so that our  $n$ -perfect algorithm can be further applied. The algorithm referred as  $Alg_{po\_partition}$  is described as below:

1. Back-trace from each failing PO following the approach described in section 3.1. If back-tracing from  $PO_i$  finds a possible fault candidate, we mark it as *reachable* from  $PO_i$ .
2. For each failing pattern  $P_i$  create the failing-PO connectivity graph,  $g_{P_i}$ . Each PO corresponds to a vertex in  $g_{P_i}$ . If two failing POs can reach the same fault by the back-tracing performed in step 1, there is an edge between them in  $g_{P_i}$ .
3. Collect the  $g_{P_i}$ s created for all failing patterns. The resulting graph,  $G_p$ , has vertices corresponding to POs. There is an edge in  $G_p$ , if there is an edge between these vertices in any of the  $g_{P_i}$ s. We assign a weight to an edge in  $G_p$  equal to the number of corresponding  $g_{P_i}$ s which contain that edge. If  $G_p$  is a *disjoint* graph, the heuristic stops. We perform the diagnosis separately on each sub-graph without losing any failing pattern information.
4. Partition  $G_p$  by removing low-weight edges.
5. Use each group of failing POs induced by the partition to filter out the original failure responses and generate the new failure responses.
6. Diagnose each group of POs separately and obtain the fault candidates.

We observe that if the initial failing-PO connection graph has more than one component, each of them can be diagnosed independently without loss of any failing pattern information. If in the current sub-graph we cannot find an initial fault candidate, we apply the lowest weight edge-cutting-based partitioning algorithm by (Step 4 - Step 6). The failing pattern information corresponding to the cutting

edges is lost. However, if we can get the initial candidate applying other failing patterns on the sub-graph, those "lost" failing patterns still can be used for further incremental diagnosis.

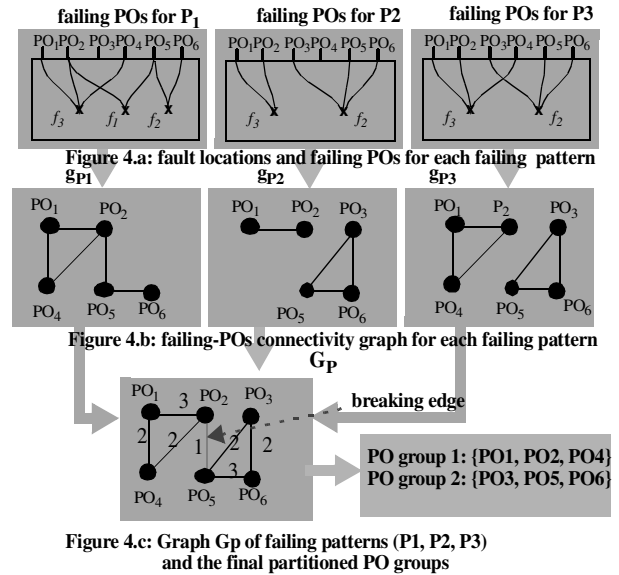


Fig. 4: Example of partitioning POs into groups.

Figure 4 shows an example of how the partition algorithm works to improve diagnosis. Suppose there are four faults ( $f_1, f_2, f_3, f_4$ ) and three failing patterns ( $P_1, P_2, P_3$ ). Additionally, each failing pattern is affected by two faults observed at some POs. This situation is depicted in Figure 4.a. For each failing pattern we build its connectivity graph,  $g_{P_i}$ , and a union of those graphs forms the graph  $G_p$ . After we break the least-weighted edge ( $PO_2, PO_5$ ) in  $G_p$ , the graph consists of two disjoint sub-graphs which induce a partition on POs. For each such a group of POs, we consider all the failing patterns. On those groups we perform diagnosis separately and ignore responses to particular input vectors on POs not contained in the current group. This PO partition effectively transforms the pattern  $P_2$  (and  $P_3$ ) from a type-2 into two type-1 patterns so that fault  $f_2$  and  $f_3$  ( $f_2$  and  $f_4$ ) can be separately identified by the  $Alg_{single}$ . Note that due to the removal of edge ( $PO_2, PO_5$ ), the pattern  $P_1$  lost some of its observation information and can not be used by the  $Alg_{single}$ . But after correctly diagnosing  $P_2$  (or  $P_3$ ) and identifying the fault candidate  $f_2$ , by applying the  $n$ -perfect algorithm,  $P_1$  can be diagnosed to further identify  $f_1$  as a possible fault candidate.

The very fact that full-scan circuits are very shallow results in many type-2 patterns. This is good for our partition algorithm since we can easily find the disjoint sub-graphs even without cutting edges, and partition the failing POs into groups without information loss. The failing-PO partition improves the diagnostic resolution and solves many inconclusive cases caused by multiple-fault behavior. After add-

ing the failing-PO partition algorithm, we get the final diagnosis flow as shown in figure 5. The enhanced flow applies  $Alg_{po\_partition}$  for each  $n$  when no  $n$ -perfect candidates are found yet unexplained failing patterns are still present. The statement: “Inject each  $n$ -perfect candidate into circuit” means that we inject candidates into the circuit one at a time (instead of injecting them all at once) and continue the diagnostic process.

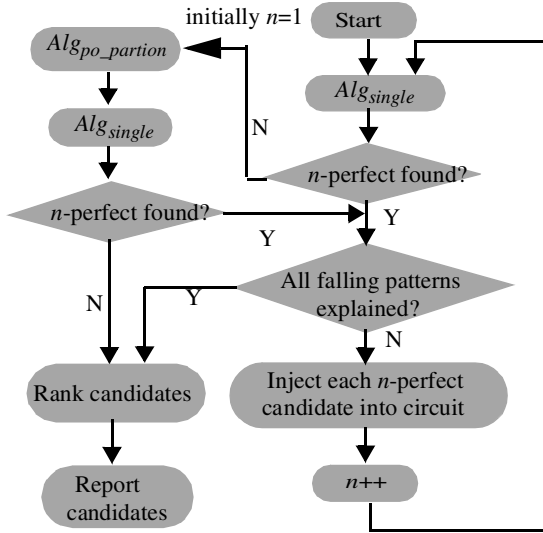


Fig. 5: Diagnosis flow including the failing-PO partition.

#### 4. Limitation Analysis and Comparisons

Our proposed algorithm can handle all the multiple-fault location cases with type-1 and type-2 failing patterns. The algorithm experiences certain limitations on the type-3 patterns. Below we discuss this case in more detail. Our analysis is performed in the context of the pseudo-random fault testability. We also explain the relationship between pseudo-random fault testability and diagnosability.

##### 4.1 Analysis of type-3 patterns

A fault is *hard to test* or its *testability is low*, when it is resistant to pseudo-random test patterns. An *easy*, or *highly testable fault*, is a fault which is easily testable by pseudo-random test patterns. We have experimentally observed that for typical test sets, easy faults are detected many times, while hard faults are detected only once or a few times. For simplicity of the description, a fault is defined as a *hard fault* when it is only detected by one pattern. A fault detected by multiple patterns is defined as an *easy fault*.

Let us consider the situation of a type-3 pattern affected by two faults,  $f1$  and  $f2$ . There are three combinations of testability measures possible for those two faults as shown in table 1.

In case 1,  $f1$  and  $f2$  are both detected multiple times. The probability is high that we can find another type-1 or type-2 test pattern for one or both of them. So this case is most likely to be diagnosable by our algorithm (or other single-fault diagnosis algorithm).

For a case 2 situation, suppose that  $f1$  is an easy fault and  $f2$  is a hard fault. Since  $f1$  is likely to be detected multiple times, it is very likely that we can find another pattern which detects only  $f1$ . In such a case our  $n$ -perfect algorithm can be applied and will correctly identify both faults.

Table 1: Three cases for type-3 patterns

	$f1$	$f2$
case 1	Easy	Easy
case 2	Easy/Hard	Hard/Easy
case 3	Hard	Hard

For a case 3 situation,  $f1$  and  $f2$  are detected only once by the same type-3 pattern. We are unlikely to resolve the problem as easily as in the earlier two cases, even though we can apply the failing-PO partitioning algorithm. A possible solution is to apply a multiple-detection test set for diagnosis to increase the chance of finding a pattern which activates only  $f1$  or  $f2$  but not both. Another possibility is to perform test generation with different fault ordering to minimize the chance that  $f1$  and  $f2$  will be activated by the same pattern. Also in practice, case 3 has low probability of occurrence compared to another two cases.

#### 4.2 Comparison with Algorithm [18]

The results presented in [18] show that the algorithm described there can achieve very good multiple-fault diagnostic resolution. However, our analysis and experimental results suggest that our algorithm can further improve the efficiency of multiple-fault diagnosis. We compare our algorithm with the algorithm proposed in [18] considering the following aspects.

##### 4.2.1 Test pattern set for diagnosis

In [18], around 6K-10K random vectors are used to determine the correction potential of each connection in the circuit. In practice, because of high test costs, people normally use compact test patterns for testing and diagnosis. For example, the ISCAS89 circuit s35932 has only 21 compact test patterns which achieve 100% test coverage for all 39.1K testable (after collapsing) single stuck-at faults. Another problem is that even though we could afford to run those 6K-10K random vectors for diagnostic purposes, we can get good error correction potential only for pseudo-random easy testable faults. The pseudo-random hard testable faults may never be activated, and their error correction potential may not be correctly assessed.

In our experiments, we use for diagnosis the compact test patterns generated by a commercial tool. By using failing and passing pattern information, our diagnostic algorithm achieves good resolution and diagnosability for multiple faults.

##### 4.2.2 Initial candidate selection

In [18] the initial candidates are selected using path-tracing techniques. Those connections with a high path-trace count (normally the top 15-20%) are chosen as initial candidates. In our method we use parity-consistent paths [22] and their

intersection which significantly reduces the number of initial faulty sites. In the worst case, if all the failing patterns are of type-2 and type-3, our method may end up with no initial candidate at the first step. However, our failing-PO partition algorithm could help to solve this difficulty. Our experimental results and the results in [2] show low probability that all the failing patterns are type-2 and type-3.

### 4.2.3 Algorithm’s complexity

In the worst case, our algorithm has exponential time complexity due to the inherent nature of the multiple-fault diagnosis problem.

The algorithm proposed in paper [18] uses three heuristics to avoid exponential complexity. However, the experimental results appear to exhibit such complexity, especially for 3- and 4-fault situations. Our algorithm utilizes the observation stated in section 3.2 combined with the analysis in section 2 to speed up the diagnostic process and to avoid in practice an exponential time complexity. From the experimental results on ISCAS’85 and ISCAS’89 benchmarks, and on the industrial designs, we have observed that typically in up to three iterations we can identify correctly the fault locations, provided that no fault-masking occurs. Here, by *fault masking* we refer to a situation when, for a given test pattern set, the fault effect cannot be observed due to the presence of other faults. Our algorithm’s diagnosis time is approximately linear with respect to the fault multiplicity.

## 5. Experimental Results

We evaluated the diagnostic capabilities of our algorithm using the multi-mix fault simulation framework briefly described in section 5.1, and real failure information collected from the tester. In section 5.2, for comparison purposes, we present experimental data for multiple stuck-at faults. In section 5.3, we show experimental results using the stuck-at fault model to diagnose failure responses caused by other types of fault behavior, such as transition faults.

### 5.1 Multi-Mix Fault Simulation Framework

One of the major difficulties in evaluating fault-diagnosis algorithms is the problem of collecting faulty chips and analyzing their defects. In order to emulate the faulty chip behavior for different defect populations and defect types, and to evaluate the relationship between the diagnostic algorithms and defect distribution, we have developed a multi-fault, mixed-type simulation environment. The framework has two applications:

First, it allows us to inject different combinations of faults into the circuit to determine the failure response. The failing patterns and failure responses are then used as inputs for the fault diagnosis algorithms. In addition, the injected fault list can be stored for later evaluation.

Second, the framework provides an environment to evaluate the quality of different diagnostic algorithms by comparing the injected fault list and the candidate list determined by the algorithm.

The fault simulation framework can be also used when the defect behavior of a chip is not fully captured by simple fault models. It is possible that the fault models reflect only some properties of the actual defects, and in such a case the framework shows their modeled behavior. How well the framework simulates the behavior of real defects depends on the types of fault models used.

The simulation framework has been implemented in such a way that other fault types can be easily added. Because one of the goals of our research is to study the deep-submicron-related defects, the environment can be enhanced to include new fault models. However, in this paper, we restrict our discussion to the fault models named below.

Currently, we use the following fault models: stuck-at, bridging (which includes bridging-AND and bridging-OR as modeled in [1]), and transition fault [3]. In addition, each fault, except for bridging faults, can occur on selected fanout branches of the same stem. We call this fault model a *branch fault*. The branch fault [16] property allows us to model open via defects, and defects located on wires or gates. We do not consider bridging faults on branches of the same stem because such faults can never be activated. Our simulation environment can handle combinational feedback loops caused by the bridging faults. However, in this paper, we investigate only how to diagnose the failure responses caused by multiple stuck-at faults or multiple transition faults.

For the evaluated algorithm and injected faults, the framework will compute a diagnosability measure reflecting the fraction of identified faults. Its numerical value is between 0 and 1. For given injected defects and test patterns, the framework allows us to evaluate systematically and compare the average diagnosability of different algorithms.

### 5.2 Experimental Results for Stuck-At Fault

First, in table 2, we list pertinent data for the circuits used in our experiments. The first column gives the circuit’s name, the second column lists the total number of stuck-at faults (SAFs) after fault collapsing, and the third column lists the number of patterns used for diagnosis in our experiments.

Table 2: Circuit Information

<i>circuit</i>	<i># SAFs</i>	<i># Pat.</i>	<i>circuit</i>	<i># SAFs</i>	<i># Pat.</i>
C880	942	30	C2670	2,747	67
C3540	3,428	133	C5315	5,350	74
C1908	1,879	115	C7552	7,550	105
C6288	7,744	22	s1488	1,488	110
s9234	6,927	168	s15850	11.7k	137
s35932	39.1k	21	s38417	31.1k	100
s38584	36.3k	156	<b>IND1</b>	534k	4k

In practice, multiple defects of large cardinality (more than four) do not happen very often. For this reason we report

only the results of experiments with one through four faults.

We define *diagnosability* as a measure reflecting the percentage of injected faults which can be correctly identified. Its numerical value is between 0 and 1.

For practical reasons (due to a huge multiple-fault space), it is infeasible to try exhaustively all multiple-fault combinations. When the cardinality of multiple faults increases, their search space increases exponentially. In our experiments, for each circuit and each different multiplicity, we perform 500 random fault injections to get different "faulty" circuits. For each multiplicity in different circuits, the *average diagnosability* is determined by averaging the diagnosability of 500 different injection test-cases. Experiments show that the averaged results are consistent for different initial pseudo-random seeds for random injection. We ran all the experiments on a SUN Blade 1000 workstation with 512MB of memory.

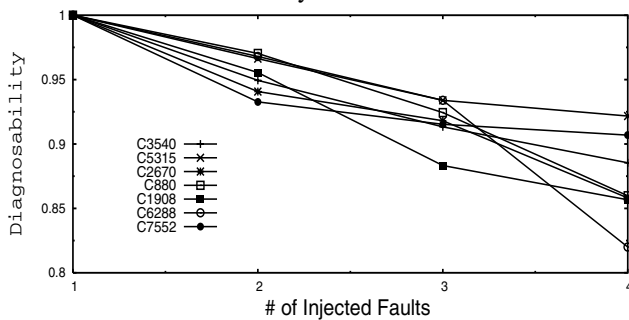


Fig. 6: Diagnosability for ISCAS'85 circuits.

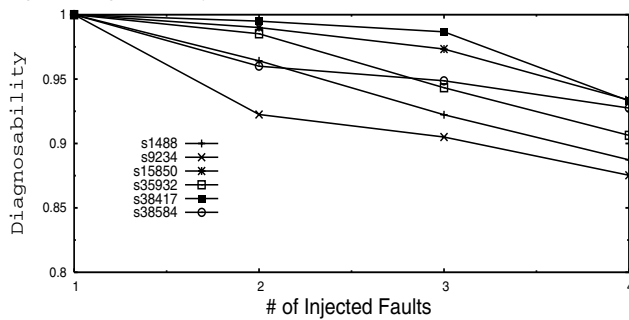


Fig. 7: Diagnosability for ISCAS'89 circuits.

Figures 6 and 7 show the average diagnosability for different fault densities for ISCAS'85 and full-scan versions of ISCAS'89 benchmark circuits. Almost all injected faulty locations can be correctly and quickly identified. However, in the fault density range, we have experimentally observed that fault masking sometimes occurs, though not very often for ISCAS'85 and ISCAS'89 benchmark circuits. Those masked faults decrease the diagnosability measure since there is no way to identify them or to observe their effects. Also, it happens sometimes that behavior of some faults cannot be observed using a particular compact pattern set, even though functionally those faults have not been masked by other injected faults. Since our algorithm assumes a single fault, when the number of injected faults is one, the diagnosability of our algorithm is always 1.

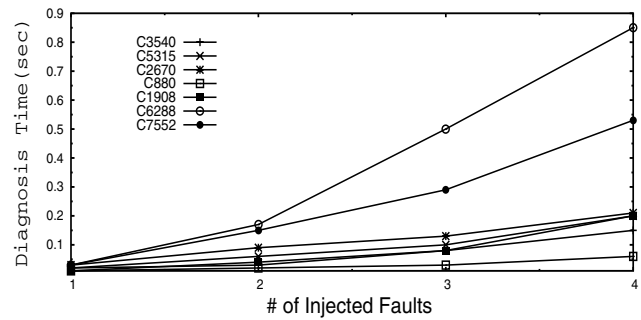


Fig. 8: Run-time for ISCAS'85 circuits.

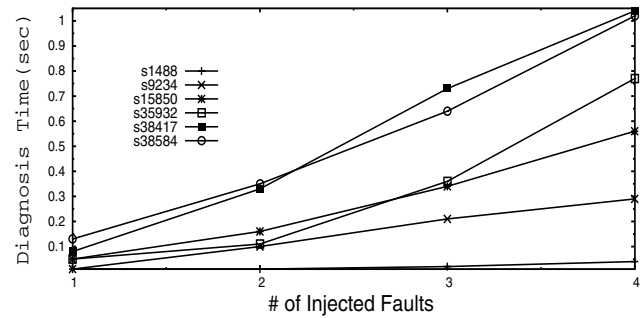


Fig. 9: Run-time for ISCAS'89 circuits.

Figures 8 and 9 show the performance (average runtime) of our algorithm. We observe that in most of the test-cases, the runtime is about linear. For bigger circuits, such as s38417 and four injected faults, the average time to identify all the injected faults is 1.04 seconds.

The data published in [18] have been obtained by running an algorithm on 10 randomly injected faults for each ISCAS circuit on a SUN Ultra 5 workstation with 128MB memory, which is slower than our experimental environment. However, the run-times in [18] appear to increase dramatically when the number of injected faults increases. Please refer to [18] for more details.

Diagnostic resolution is a very important feature of any failure analyzer. It translates to time savings during the chip microscope scanning. We use the total number of reported sites as a measure of resolution for our algorithm. We use a single fault location to represent the stuck-at fault equivalence class. In figures 10 and 11, the number of total sites is the average of the total number of reported representative fault classes we have to examine. Our results suggest that

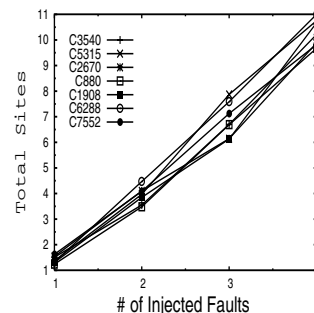


Fig. 10: Total reported sites for ISCAS'85 circuits.

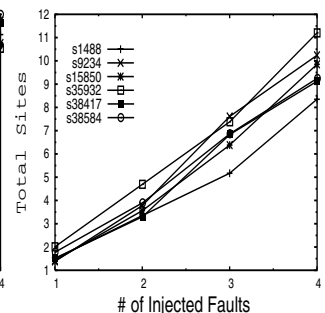


Fig. 11: Total reported sites for ISCAS'89 circuits.

we need to probe only a few candidates to locate the real defect locations. From the diagnosability data and the total reported sites information, we can see that our algorithm can identify almost all injected fault locations with good resolution.

Our diagnostic algorithm lists the identified faults in a specific order, with the most probable faults (based on matching) reported first. A property important from the practical standpoint is the rank, on the ordered list of faults found by the algorithm, of the first fault which matches an injected fault. This is often referred to as the *first hit rank* (FHR).

Besides reporting the total number of possible faulty sites, we use FHR to evaluate our algorithm. Since the next step of the diagnostic process is to use a microscope to examine the candidate sites in the reported order, we can save a lot of work for analyzers if the real cause of failure appears earlier in the candidate list. The authors of paper [9], whose algorithm enumerates all possible combinations to solve multiple gate-type error problem also use the first hit rank to evaluate their heuristic (*first hit index* in [9]).

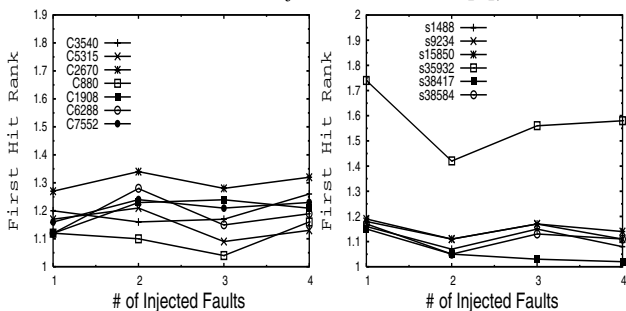


Fig. 12. FHR results for ISCAS'85 circuits.

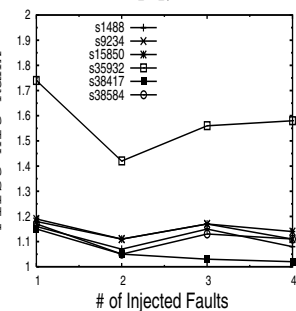


Fig. 13. FHR results for ISCAS'89 circuits.

Figures 12 and 13 show the first hit rank measure results. We observe that normally the first or second reported candidate corresponds to the real injected fault. The benchmark s35932 has the first hit rank much higher (worse) than the other test-cases. The reason is that the number of compact test patterns for this circuit is only 21 (100% test coverage). Working with too few failing patterns reduces the resolution of our algorithm. Also we note that the resolution of s35932 is worse than in the case of other circuits from figure 11. This is due to the very small size of the diagnosis pattern set.

We have applied our algorithm to an industrial circuit IND1 which is a 300K gate full-scan sequential design. For diagnosis we used a 4K compact stuck-at fault test pattern set (98.23% test coverage). The results in table 3 have been obtained by averaging 50 random injected test-cases for each fault density.

Table 3 shows comparisons of the diagnostic results for IND1. For different injected stuck-at fault densities, we report two diagnosability measures DA1 and DA2. DA1 is obtained when we use all the failing and passing patterns to perform the diagnosis. DA2 is obtained using the first 50 failing patterns as failure responses and all the passing patterns up to the 50th failing pattern. The purpose of performing this experiment is to emulate the real test-application

conditions. Due to limited tester storage space, we cannot store as much information about failing patterns as we would like for diagnosis purposes. This experiment shows that even when using only the first 50 failing patterns, our algorithm can identify almost every injected faulty location with a slightly worse resolution than that achievable by the large test set.

Table 3: Results for IND1 (300K full-scan sequential design)

# of inj. faults		1	2	3	4
All Pat.	DA1	1.00	0.99	0.98	0.98
	FHR	1.00	1.02	1.04	1.06
	# sites	1.06	3.23	5.57	7.58
	T (sec)	29.06	100.74	150.74	281.35
50 Fpat.	DA2	1.00	0.99	0.97	0.95
	FHR	1.01	1.07	1.10	1.26
	# sites	1.10	5.27	7.45	11.52
	T (sec)	25.71	87.21	130.34	200.11

We also validate our algorithm on manufactured failing chips. From two companies we obtained real tester failure information for their full-scanned circuits (100K and 300K gates designs) to validate our algorithm. We first applied a commercial tool to diagnose the chip failure responses. On those cases in which the commercial tool reported complete results, our algorithm produced the same results as the commercial tool did. For those cases in which the commercial tool reported incomplete or inconclusive results, our algorithm can identify the correct multiple-fault candidates.

### 5.3 Experimental Results for Transition Faults

The stuck-at fault model can be used to detect failure responses caused by many other types of defects which could be modeled better by other models, such as bridging- and transition-fault models. In this work, we investigate the feasibility of using the stuck-at fault model to diagnose failure responses caused by other types of faults (defects). We evaluated diagnostic capabilities of our algorithms using the multi-mix fault simulation framework. We used the stuck-at fault model to diagnose the failure responses caused by multiple bridging faults or multiple transition faults. We list only the results for transition faults in this paper. We do not list the results for bridging faults because the diagnosability of such faults is relatively low compared with stuck-at or transition-fault failure responses. In the future, we will investigate the feasibility of using the bridging-fault model for diagnosis in our incremental technique.

In this experiment, we used for diagnosis about 5.4K transition fault test patterns, which detect all testable single transition faults in the full-scanned sequential design IND1. For different fault densities we randomly generated 50 test-cases. The results are shown in table 4. To compare those results with the results for stuck-at fault behavior, we use the same format as in table 3.

Table 4: Transition fault behavior diagnosis results for IND1

# of inj. faults		1	2	3	4
All Pat.	DA1	1.00	0.98	0.94	0.89
	FHR	1.02	1.14	1.20	1.25
	# sites	1.08	4.14	6.24	10.30
	T (sec)	42.41	113.07	155.00	400.12
50 Fpat.	DA2	1.00	0.98	0.92	0.84
	FHR	1.12	1.24	1.22	1.40
	# sites	2.24	6.76	8.50	13.26
	T (sec)	30.29	100.35	154.74	240.79

From table 4 we observe that our algorithm can also identify most of the injected transition locations even though we use the stuck-at fault model and only the first 50 failing patterns. However, the resolution and diagnosability are somewhat decreased due to the behavior differences between the stuck-at fault model and the transition-fault model.

## 6. Conclusions and Future Work

Our diagnostic algorithm takes multiple-fault behaviors into account. To cope with design complexities and to improve the diagnostic resolution, it applies the failing-PO partition heuristic. Our algorithm achieves very good resolution and diagnosability with approximately linear time complexity.

In the future we will study the relationship between the quality of the test pattern set and diagnosability. The analysis and the experimental results of multiple-fault diagnosis suggest that the multiple detection pattern set should improve multiple-fault diagnosability.

Also layout information provides very useful guidance for analysis of defect behavior. It is especially valuable for bridging-fault and branch-fault diagnosis. We are also exploring the at-speed deep sub-micron (DSM) defect diagnosis, in particular the crosstalk and power/ground-bounce delay defects.

### Acknowledgement

This work was supported by the California MICRO Program through Mentor Graphics Corporation. We also acknowledge an equipment grant from Intel Corporation.

### References

- [1] M. Abramovici, M.A. Breuer, A.D. Friedman, "Digital Systems Testing and Testable Design", 1990, Computer Science Press, pp.94-95.
- [2] T. Bartenstein, D. Heaberlin, L. Huisman, D. Sliwinski, "Diagnosing combinational logic designs using the single location at-a-time (SLAT) paradigm", Proc. International Test Conference, 2001, pp. 287 -296.
- [3] Z. Barzilai and B. Rosen, "Comparison of ac self-testing procedures", Proc. International Test Conference, 1983, pp. 89-94.
- [4] B. Boppana, R. Mukherjee, J. Jain, and M. Fujita, "Multiple Error Diagnosis Based on Xlists", Proc. 36th Design Automation Conf., 1999, pp. 660 -665.
- [5] P.Y. Chung, I.N. Hajj, "Diagnosis and Correction of Multiple Logic Design Errors in Digital Circuits", IEEE Trans. on VLSI Systems, Vol. 5 Issue: 2, June 1997 pp. 233 -237.
- [6] H. Cox and J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis", IEEE Trans. on CAD, Vol. 7 No.7, 1988, pp. 813-833.
- [7] I. Hamzaoglu and J.H. Patel, "New Techniques for Deterministic Test Pattern Generation", Proc. 16th, IEEE VLSI Test Symp. 1998, pp. 446-452.
- [8] S.-Y. Huang, K.-T. Cheng, K.C. Chen, D.T. Chang, "Error-Tracer: A Fault Simulation Based Approach to Design Error Diagnosis", Proc. International Test Conference, 1997, pp. 974-981.
- [9] S.-Y. Huang, "On improving the Accuracy of Multiple Defect Diagnosis", Proc. 19th IEEE 2001 VLSI Test Symposium, pp. 34-39.
- [10] V. Krishnaswamy, A.B. Ma, P. Vishakantiah, "A study of Bridging Defect Probabilities on a Pentium(tm) 4 CPU", Proc. International Test Conference, 2001, pp. 688-695.
- [11] J. C.-M. Li and E.J. McCluskey, "Diagnosis of Sequence Dependent Chips", Proc. 20th IEEE VLSI Test Symposium, 2002, pp. 187-192.
- [12] I. Pomeranz and S.M. Reddy, "On Correction of Multiple Design Errors", IEEE Trans. on CAD of Integrated Circuits and Systems, Vol. 14 Issue: 2, Feb. 1995 pp. 255-264.
- [13] J. Richman and K.R. Bowden, "The modern Fault Dictionary", Proc. International Test Conference, 1985, pp. 696-702.
- [14] H. Takahashi, K.O. Boateng, K.K. Saluja, Y. Takamatsu, "On diagnosing multiple stuck-at faults using multiple and single fault simulation in combinational circuits", IEEE Trans. on CAD of Integrated Circuits and Systems, vol 21, no 3, Mar 2002, pp. 362 -368.
- [15] H. Takahashi, N. Yanagida, and Y. Takamatsu, "Enhancing Multiple Fault Diagnosis in Combinational Circuits Based on Sensitized Paths and EB testing", Proc. of the Fourth Asian Test Symp., 1995, pp. 58 -64.
- [16] S. Venkataraman, S.B. Drummonds, "Poirot: Applications of a Logic Fault Diagnosis Tool", IEEE Design & Test of Computers, vol 18 no 1, Jan.-Feb. 2001.
- [17] S. Venkataraman and W.K. Fuchs, "A Deductive Technique for Diagnosis of Bridging Faults." Proc. IEEE International Conference on Computer Aided Design, pp. 562-567, 1997.
- [18] A. Veneris, J.B. Liu, M. Amiri, M. S. Abadir, "Incremental Diagnosis and Correction of Multiple Faults and Errors", Proc. Design, Automation and Test in Europe Conference and Exhibition, 2002, pp. 716 -721.
- [19] A. Veneris, S. Venkataraman, I.N. Hajj, W.K. Fuchs, "Multiple design error diagnosis and correction in digital VLSI circuits", Proc. IEEE VLSI Test Symp. 1999, pp. 58-63
- [20] A. Veneris and I.N. Hajj, "Design Error Diagnosis and Correction via Test Error Simulation", IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 18, no 12, Dec. 1999 pp. 1803 -1816.
- [21] A. Veneris and I.N. Hajj, "A Fast Algorithm for Locating and Correcting Simple Design Errors in VLSI Digital Circuits", 1997, Proc. Great Lake Symp. on VLSI Design, pp.45-50.
- [22] J. Waicukauski and E. Lindbloom, "Failure Diagnosis of Structured VLSI", IEEE Design & Test of Computers, Aug. 1989, pp.49-60.