An Efficient Method for Multiple Fault Diagnosis

Khushboo Sheth
Department of Electrical and Computer Engineering
Auburn University, Auburn, AL

Abstract: *In this paper, failing circuits are analyzed and a multiple-fault diagnosis approach is proposed. An incremental simulation-based approach is used to diagnose failures one at a time. Furthermore, to improve the diagnosability, a failing-primary-output partitioning algorithm is proposed. Experimental results show that this approach has approximately linear time complexity, and it achieves high diagnosability and resolution. 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. 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. The second one traces the effect-cause dependencies. 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 [2]*. 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.

In this paper an incremental approach based on multiple-fault simulation for combinational and full-scan sequential circuits is proposed. This 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* and *Failing-PO Partition* algorithm can significantly help to avoid the inherent exponential time-complexity problem in multiple-fault diagnosis. A diagnostic framework which can handle any specific fault model and invoke any diagnostic algorithm is also developed [7].

2.FAILING PATTERN ANALYSIS [6]

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. This 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 [6]:

Type-1: p can activate only one fault and observe its effect. This type of pattern is also defined as a single-location-at-a-time (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.

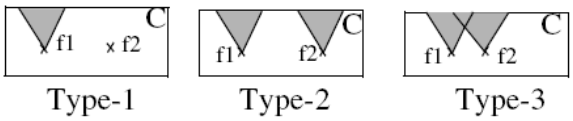
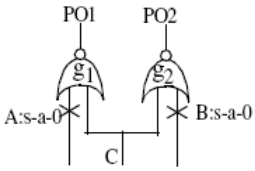


Fig. 1: Failing Pattern Types

Most of the diagnostic algorithms are based on the single-fault assumption. 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 detection 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.



Pat	A	B	C	PO1	PO2	
P1	0	0	0	1	1	passing
P2	1	1	0	0/1	0/1	failing
P3	0	1	0	1	0/1	failing

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 P1 is a passing pattern. Applying patterns P2 and P3 we observe some failing POs. The single-fault diagnosis algorithm applied on the failing pattern P3 will result in identifying a possible fault candidate B s-a-0, since the A s-a-0 fault is not activated by P3 (thus P3 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 P2 (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 [1][3]. Another important fact is that, in practice, a compact test pattern set is used 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.

3.PROPOSED ALGORITHM

3.1 DIAGNOSIS ALGORITHM [5]

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 faulty 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 [7]

It is observed that: *Type-1 failing pattern very often exists even if multiple-fault failure responses are also present.* This implies that in most multiple-fault failure cases, some fault locations can be identified just by the single-fault diagnosis algorithm. The algorithm given here 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 it is already known (from analyzing failing pattern P3), that B s-a-0 is the most probable fault in the circuit, then the circuit could be simply changed by connecting net B to ground. Then we could perform analysis of the failing pattern P2 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 P2 has become a type-1 pattern in the modified circuit. In the algorithm given here, there is also an attempt to manage the complexity of multiple-fault diagnosis. 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, passing-pattern information is used to rank the candidates and to select the best among them.

3.2.1 *n-perfect* ALGORITHM [6]

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.

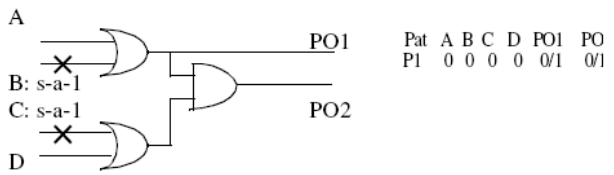


Fig. 3: Example of an *n*-perfect candidate.

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}. 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.
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, a weighting mechanism [3] is used to rank potential candidates. For each *n*-perfect candidate, "match," "mis-predict," and "non-predict" weights is used.

In addition to the failing patterns, passing patterns are also simulated. 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. The failing pattern weights are greedily used to find possible candidates. Then, for those candidates we compute a combined weight, order them, and drop the worst candidates.

The proposed algorithm and the majority of other 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 the incremental heuristic approach given, most of them will be pruned out after the 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 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, partition POs into 2 groups: PO_1 is in one group, PO_2 in the other. Then perform the diagnosis for each failing-PO group separately on each failing pattern. For the failing pattern P_2 and the group {PO_1}, A stuck-at 0 is found as a candidate. Similarly, the pattern P_2 and the {PO_2} group is used to get another fault candidate, B stuck-at 0. After finding some candidates, we perform the algorithm again, obtain more candidates, and refine the results.

3.3.1 FAILING-PO PARTITIONING ALGORITHM [7]

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. The goal here 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 the *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 Pi create the failing-PO connectivity graph, g_{Pi}. Each PO corresponds to a vertex in g_{Pi}. 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_{Pi}.
3. Collect the g_{Pi}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_{Pi}s. We assign a weight to an edge in G_p equal to the number of corresponding g_{Pi}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.

It is observed 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, the lowest weight edge-cutting-based partitioning algorithm is applied 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.

Figure 4 shows an example of how the partition algorithm works to improve diagnosis. Suppose there are fthe faults ($f1$, $f2$, $f3$, $f4$) 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 its connectivity graph, g_{Pi}, is build and a union of those graphs forms the graph G_P. After breaking 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 group of POs, all the failing patterns are considered. On those groups diagnosis are performed separately and responses to particular input vectors on POs not contained in the current group are ignored. This PO partition effectively transforms the pattern P_2 (and P_3) from a type-2 into two type-1 patterns so that fault $f2$ and $f3$ ($f2$ and f4) can be separately identified by the Algsingle. Note that due to the removal of edge (PO2, PO5), the pattern P1 lost some of its information and cannot be used by the *Alg*$_{single}$. But after correctly diagnosing P_2 (or P_3) and identifying the fault candidate $f2$, by applying the *n*-perfect algorithm, P_1 can be diagnosed to further identify $f1$ as a possible fault candidate.

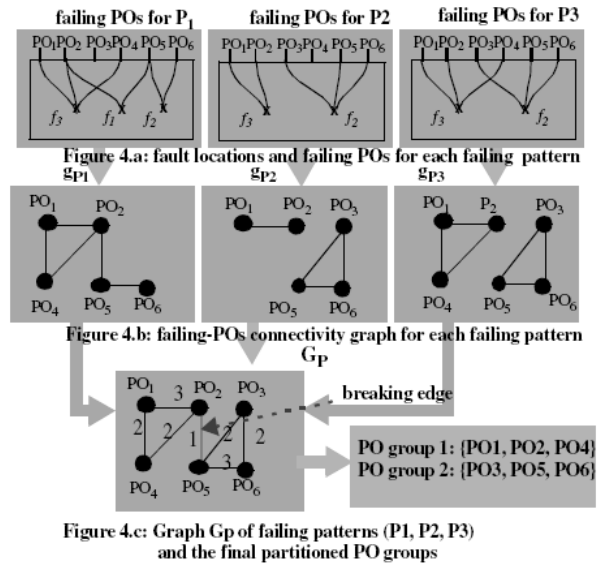


Fig. 4: Example of partitioning POs into groups.

The very fact that full-scan circuits are very shallow results in many type-2 patterns. This is good for partition algorithm since it 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 adding 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.

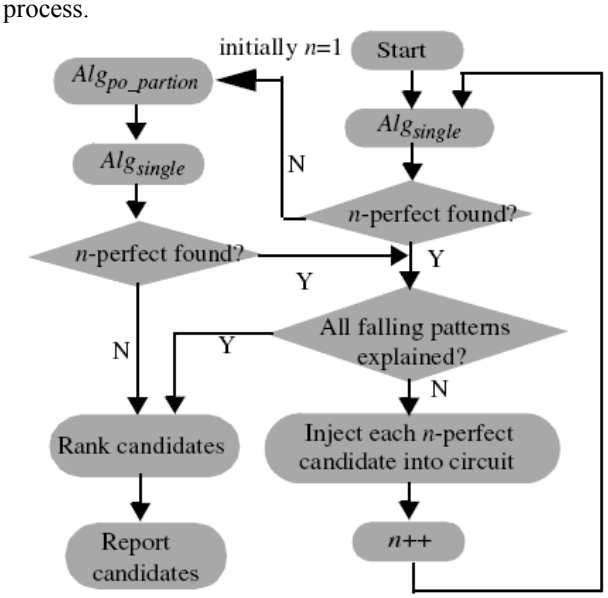


Fig. 5: Diagnosis flow including the failing-PO partition.

4.LIMITATION ANALYSIS

The 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. The discussion of this case in more detail is given below. The analysis is performed in the context of the pseudo-random fault testability. The relationship between pseudo-random fault testability and diagnosability is also explained.

4.1 ANALYSIS OF TYPE-3 PATTERNS [7]

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. It has been 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 the algorithm proposed (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 *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.

5. EXPERIMENTAL RESULTS [7]

The diagnostic capabilities of the proposed algorithm are evaluated 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, experimental data for multiple stuck-at faults are presented. In section 5.3, experimental results, using the stuck-at fault model to diagnose failure responses caused by other types of fault behavior, such as transition faults, are shown.

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, a multi-fault, mixed-type simulation environment has been developed. 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. The following fault models are used here: stuck-at, bridging (which includes bridging-AND and bridging-OR), and transition fault.

5.2 EXPERIMENTAL RESULTS FOR STUCK-AT FAULT

First, in table 2, pertinent data for the circuits used in the experiments is given. 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 the experiments. In practice, multiple defects of large cardinality (more than fthe) do not happen very often. For this reason only the results of experiments with one through fthe faults are reported.

Diagnosability is defined 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 the experiments here, for each circuit and each different multiplicity, 500 random fault injections are performed 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. All the experiments are ran on a SUN Blade 1000 workstation with 512MB of memory.

Table 2: Circuit Information

circuit	*# SAFs*	*# Pat.*	*circuit*	*# SAFs*	*# Pat.*
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	**IND1**	534k	4k

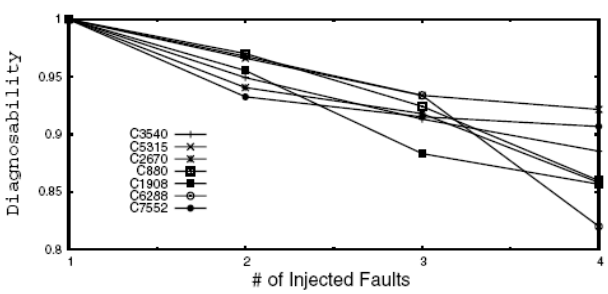


Fig. 6: Diagnosability for ISCAS'85 circuits.

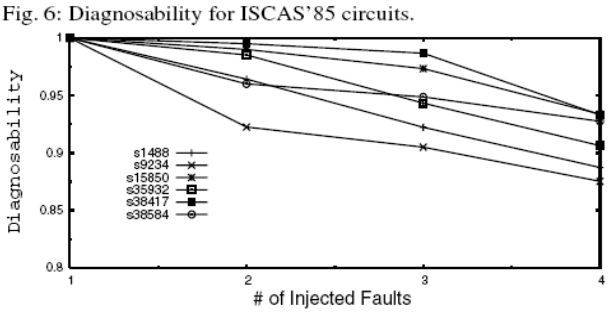


Fig. 7: Diagnosability for ISCAS'89 circuits.

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, it has been 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 the given algorithm assumes a single fault, when the number of injected faults is one, the diagnosability of this algorithm is always 1.

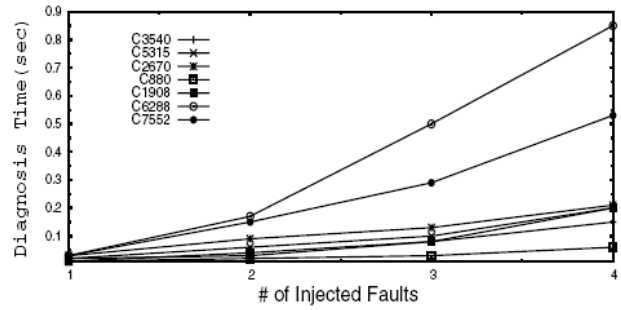


Fig. 8: Run-time for ISCAS'85 circuits.

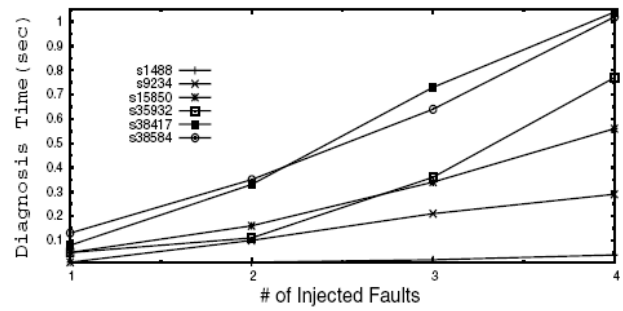


Fig. 9: Run-time for ISCAS'89 circuits.

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

Diagnostic resolution is a very important feature of any failure analyzer. It translates to time savings during the chip microscope scanning. The total number of reported sites is used as a measure of resolution for this algorithm. The given algorithm can identify almost all injected fault locations with good resolution. It 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).

The given algorithm has also been applied to an industrial circuit IND1 which is a 300K gate full-scan sequential design. For diagnosis a 4K compact stuck-at fault test pattern set (98.23% test coverage) is used. 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, two diagnosability measures DA1 and DA2 are reported. 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, the given 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

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. And so the stuck-at fault model is used here to diagnose the failure responses caused by multiple bridging faults or multiple transition faults. Only the results for transition faults are listed. The results for bridging faults are not listed because the diagnosability of such faults is relatively low compared with stuck-at or transition-fault failure responses.

In this experiment, about 5.4K transition fault test patterns, which detect all testable single transition faults in the full-scanned sequential design IND1are used for diagnosis. For different fault densities randomly 50 test cases are generated. The results are shown in table 4. To compare those results with the results for stuck-at fault behavior, the same format as in table 3 is used. From table 4 we observe that the given algorithm can also identify most of the injected transition locations even though the stuck-at fault model and only the first 50 failing patterns is used. 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. CONCLUSION

The proposed 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. This algorithm achieves very good resolution and diagnosability with approximately linear time complexity.

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

REFERENCES

1) J. C.-M. Li and E.J. McCluskey, “Diagnosis of Sequence Dependent Chips”, Proc. 20th IEEE VLSI Test Symposium, 2002, pp. 187-192.

2) 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.

3) 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.

4) 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.

5) J. Waicukauski and E. Lindbloom, “Failure Diagnosis of Structured VLSI”, IEEE Design & Test of Computers, Aug.1989, pp.49-60.

6) Zhiyuan Wang, Malgorzata Marek-Sadowska, Kun-Han Tsai, Janusz Rajski, “Multiple Fault Diagnosis Using *n*-Detection Tests”, IEEE Computer Design 2003 Proc., 21st International Conference, pp. 198-201

7) Zhiyuan Wang, Kun-Han Tsai, Malgorzata Marek-Sadowska, Janusz Rajski, “Efficient and Effective Methodology on the Multiple Fault Diagnosis”, IEEE Test Conference, 2003. Proc. ITC 2003. International ,Volume 1, pp. 329 - 338