Diagnosis of Arbitrary Defects Using Neighborhood Function Extraction*

Rao Desineni and R. D. (Shawn) Blanton
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA-15213.
Email: {rhd,blanton}@ece.cmu.edu

Abstract

We present a methodology for diagnosing arbitrary defects in digital integrated circuits (ICs). Rather than using one or a set of fault models in a cause-effect or effect-cause approach, our methodology derives defect behavior from the test set, the circuit and its response, and the physical neighbors that surround a potential defect location. The defect locations themselves are identified using a model-independent stage. The methodology enables accurate identification of defect location and behavior through validation via simulation using passing and additional diagnostic test patterns. A byproduct of our methodology is the distinction that can be made among stuck-fault equivalencies which results in improved diagnostic resolution. Several types of shorts and opens are used to demonstrate the applicability of our approach to the diagnosis of arbitrary defects.

Keywords: *Diagnosis, defects, failure analysis, test generation, yield enhancement.*

1 Introduction

Failure analysis (FA) is used to characterize defects that occur during the fabrication of an IC so that the manufacturing process and/or design can be corrected to improve yield or test escape. Fault diagnosis has typically augmented the complex task of physical analysis of the failure (PFA) by acting as a first step towards locating defects. However, because PFA is becoming increasingly complex and time-consuming, diagnosis must take on a more important role in FA [1]. Most defects exhibit logic-level misbehavior, and therefore, can be modeled as a logical fault. Our objective in fault diagnosis is to identify where and under what logical conditions does the logical misbehavior of a defect manifest.

Past approaches to fault diagnosis include techniques for fault localization [2,3] and those that attempt to identify a particular type of fault [4–6]. It has been argued that localization alone is not sufficient [7–9] and that using accurate fault models for diagnosis improves accuracy [4, 9]. The latter approach of using specific fault models for fault diagnosis works well when the defect behavior can be conjectured fairly accurately. However, it has been shown that the commonly used fault models may be insufficient to model the complex behaviors of defects in nanoscale technologies [7, 10]. One approach is to use several fault models and response matching algorithms for fault diagnosis [8, 11, 12]. However, those approaches require vital information regarding the logical behavior of defects for correct diagnosis.

In contrast to past approaches, the diagnosis methodology presented in this paper attempts to derive the behavior of defects that manifest as a logical fault from the test data regardless of the types of defects. The extracted logical behavior is exhaustively verified before being declared a diagnosis candidate and contains both location and behavior of the defect. By explicitly remaining independent of any particular defect type (and its assumed logical behavior), our methodology implicitly deals with arbitrary defects of both known and unknown types. Additionally, rather than finding if a set of modeled faults is the cause of a failure, our methodology identifies all possible causes of the failure. For example, a defect that resembles a 2-line short may in fact be a 3-line short; our methodology will identify both as possible candidates. In this paper, we discuss the applicability of our methodology for diagnosis of single-cycle defects, *i.e.*, sequence- and delay-dependent defects are not considered but are the focus of our current work.

The rest of the paper is organized as follows. A brief overview of our diagnosis methodology is provided in Section 2. The diagnosis framework consisting of the circuits, defects and test data used in this paper are introduced in Section 3. The details of the diagnosis methodology and the results are described in the context of the diagnosis framework in Section 4. Finally, conclusions are provided in Section 5.

2 Methodology Overview

Our diagnosis methodology is targeted towards identifying both the site and defect type. The methodology uses a very general set of assumptions regarding defect

*This work is supported by the Pittsburgh Digital Greenhouse under agreement no. A002838.

behavior that are much weaker than those employed by existing fault models.

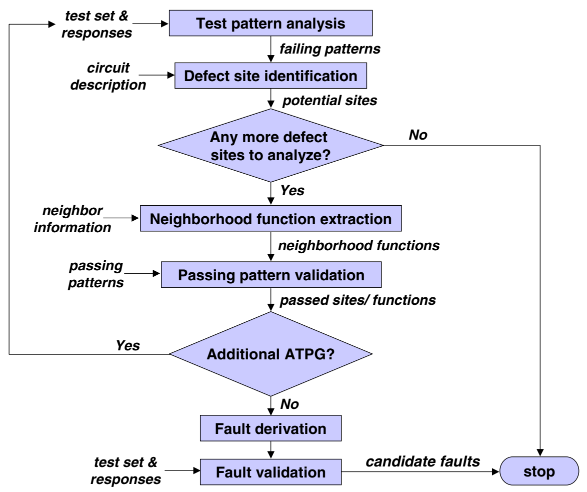


Figure 1: Our diagnosis methodology.

In this work, we assume defects manifest as a logical fault affecting one or more signal lines when certain logical conditions on the physical neighbors surrounding the faulty lines are satisfied. As shown in Figure 1, our methodology first identifies the faulty signal lines and then attempts to extract the set of logical conditions for the physical neighbors responsible for fault excitation for each faulty line. Once the conditions have been identified, the fault and the excitation conditions (called neighborhood functions) for each location are represented using fault tuples [13], which is subsequently verified via simulation to validate that the fault accurately matches the behavior of the defects. The validated neighborhood functions for each fault location are then combined into a “macrofault” [14] to represent the defects in the fault derivation step. Finally, the derived faults are validated via simulation using existing and possibly additional test patterns [15]. The validated faults are the output of our diagnosis.

When defects do not meet our assumptions, it is possible that incorrect locations and/or neighbor functions are derived early in the diagnosis methodology. In order to account for error in the early stages, feedback from the validation stages to the location identification stage can be used to backtrack and relax some of the assumptions.

It is known that industry has typically correlated fault locations identified by logic diagnosis tools with layout information. However, this correlation is largely manual and in general is restricted to identifying lines that are close enough to be part of the defect. We propose an automated approach that uses restricted layout information to derive a precise fault model based on the logical misbehavior of defects. Additionally, the faults derived by our methodology can be utilized for fast identification of similar defects in the future. In summary, our methodology attempts to discover defect behavior rather than assume it in the form of a finite et of fault models.

In this paper, we first describe the important details of our methodology and discuss our set of assumptions. We then demonstrate one application of our diagnosis methodology: we show how identification of accurate logic behavior from the test data can be utilized to reduce the number of fault locations for failure analysis. Specifically, we present results showing the reduction in the number of diagnosis candidates for several types of defects. For lack of space, we do not present results for the additional ATPG, fault model derivation, and validation stages.

3 Diagnosis Framework

In this section, we describe the circuits and defects used in this paper. We use five benchmark circuits; although each analyzed circuit carries the name of an ISCAS’85 [16] circuit, the actual gate-level circuits differ significantly from their original form due to synthesis. The circuits are first logically optimized and then technology-mapped using a commercial physical synthesis tool for a 0.18μm standard-cell library. For logic diagnosis purposes, a gate-level netlist consisting of primitive gates is extracted from the standard-cell representation. For each circuit, we use a 100% stuck-at test set generated by a commercial ATPG tool. The basic characteristics of the five circuits are listed in Table 1.

Circuit name	No. of lines	No. of gates	No. of tests
c432	179	420	61
c880	868	385	64
c1196	1106	480	138
c1355	1091	489	91
c3540	2433	1104	155

Table 1: Benchmark characteristics.

Our methodology utilizes lines in the physical neighborhood of potentially faulty lines. The physical neighbors for each line in the circuit are obtained using critical area analysis. Specifically, for each signal line in the circuit, the set of lines that have a critical area for a defect of radius $0.5\mu m$ are deemed its physical neighbors. Critical area extraction for the layout is performed using a commercial critical area analysis tool.

For each circuit, we simulated various types of two- and three-line shorts and opens to generate test responses. The shorted lines are selected based on critical area analysis. We created bridge models where one line dominates the other based on the relative driving strengths of the two lines [17]. Driving strengths of lines are determined based on the number of ON and OFF

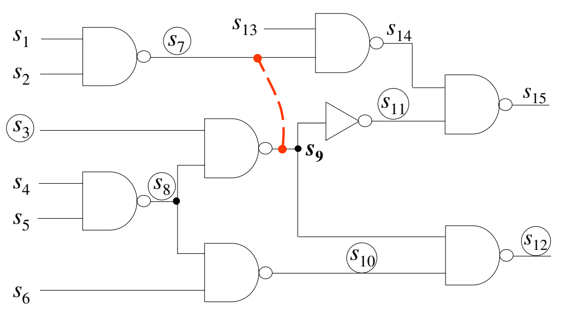


Figure 2: Two-line short between victim s_9 and a physical neighbor s_7; physical neighbors of s_9 are circled.

transistors. An example of a driver-based dominating bridge fault is shown in Figure 2, where s_7 imposes its value on s_9 only when both s_1 and s_2 are driven to logic-0. We also included the commonly used AND/OR and dominating bridge models. We modeled three-line shorts using a majority function, that is, a line is forced to v if the other two lines are driven to v. Further, we created models of opens based on the analysis presented in [6].

4 Diagnosis Methodology

An overview of our diagnosis methodology is shown in Figure 1. In this section, we provide the details of the important steps in the methodology.

4.1 Defect Site Identification

The first step involves identification of potential defect sites. Initially, we partition the set of test patterns into failing and passing patterns by comparing the output response of the circuit-under-diagnosis (CUD), called a *test response*, and its fault-free response for every test pattern. We then reduce the search space for faulty signal lines by performing a path-trace procedure [11] from the failing outputs for all the failing patterns. Path-trace is very conservative and is guaranteed to include all possible faulty signal lines, even when multiple lines are simultaneously faulty. The output of our path-trace procedure is a list of stuck-at faults, represented by the set $\mathbf{S_p}$. The signal lines associated with the faults in $\mathbf{S_p}$ are marked as lines that are potentially affected by the defect.

After path-trace, failing patterns are categorized as either SLAT [3] or non-SLAT patterns. SLAT patterns are obtained by simulating the stuck-at faults in $\mathbf{S_p}$ and finding those failing patterns that can be explained by at least one fault in $\mathbf{S_p}$. A fault is said to *explain* a test pattern if the CUD output response for the pattern exactly matches the simulation response of the fault. All failing patterns that are not SLAT patterns are called non-SLAT patterns [3]. In this work, we use only SLAT patterns for characterizing potential defect sites. At this stage, we make the following assumption based on our experience in dealing with various defect types and empirical data published in the literature [3].

Assumption 1. *For each line l_i affected by an arbitrary defect in a CUD, there exists at least one SLAT pattern explained by a stuck-at fault on l_i.*

Non-SLAT patterns represent the failing patterns that cannot be explained by any one stuck-at fault, and therefore, must be due to the presence of a defect that simultaneously affects multiple lines[1]. Although, we do not use non-SLAT patterns for defect site identification, they are used for fault model validation as shown in Figure 1. For the defects analyzed in this paper, all the failing patterns are SLAT patterns.

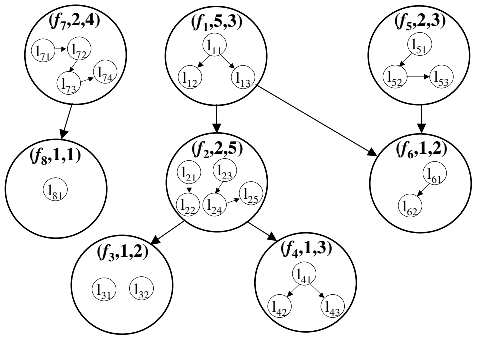
Even though the set of suspect lines returned by path-trace is much smaller than the set of all circuit lines, $\mathbf{S_p}$ is still quite large. In order to further reduce the size of $\mathbf{S_p}$, we adopt an approach similar to per-test diagnosis [2, 3, 12]. Per-test diagnosis treats each individual test response as an independent diagnosis. For each SLAT pattern, the CUD exhibits a certain logical misbehavior. Per-test diagnosis attempts to identify the cause of misbehavior for each SLAT pattern. We first collapse the faults in $\mathbf{S_p}$ using structural equivalence and then fault simulate the representative faults of each equivalence class using SLAT patterns to identify the patterns explained by each class. Only the classes that explain at least one SLAT pattern are retained.

The result of per-test diagnosis is a set of stuck-at fault classes along with the SLAT patterns that each class explains. The stuck-at faults in these classes represent potential defect locations and are represented using a graphical data structure we call a *cover forest*. A cover forest is a directed acyclic graph in which each class of equivalent (under the SLAT patterns) stuck-at faults are represented as a vertex of the graph. A directed edge from vertex a to a different vertex b exists, if and only if, the faults represented by a explains all the failing patterns explained by b but not vice-versa[2].

An example of a cover forest is shown in Figure 3. Each vertex of the cover forest is labeled with a tuple (f_i,s,n), where f_i is the representative of the equivalence class of faults in the vertex, s is the number of SLAT patterns explained by each fault belonging to the vertex, and n is the number of faults in the vertex. The top of each tree in the forest is called a *root* vertex and contains a fault set that explains the largest number of

[1]It is possible that multiple faults cause a failing pattern to appear as a SLAT pattern due to error masking.

[2]If a and b explain the same set of SLAT patterns, the two classes, by definition, should be combined.



(f_7,2,4)
l_{71} l_{72} l_{73} l_{74}
(f_1,5,3)
l_{11} l_{12} l_{13}
(f_5,2,3)
l_{51} l_{52} l_{53}
(f_8,1,1)
l_{81}
(f_2,2,5)
l_{21} l_{23} l_{22} l_{24} l_{25}
(f_6,1,2)
l_{61} l_{62}
(f_3,1,2)
l_{31} l_{32}
(f_4,1,3)
l_{41} l_{42} l_{43}

Figure 3: An example of a cover forest.

SLAT patterns. The graph in Figure 3 has three trees rooted at (f_7,2,4), (f_1,5,3) and (f_5,2,3). The relation among faults with respect to SLAT patterns can easily be deduced from the cover forest. For example, it can be observed that two of the five SLAT patterns explained by f_1 are also explained by f_2, and one SLAT pattern explained by f_1 is explained by f_6. However, since there is no edge from f_2 to f_6, the SLAT pattern explained by f_6 must be different from those explained by f_2.

As shown in Figure 3, a sub-circuit graph representing the structural relationship among the fault sites in a vertex is also contained in the forest. The sub-circuit graph is a directed acyclic graph where each fault site is represented as a vertex of the graph. A directed edge exists between two vertices l_i and l_j, if l_j can be reached from l_i. The structural connections between different fault sites can be utilized for generating fault distinguishing test patterns for increasing diagnostic resolution, a stage of our methodology that is not elaborated upon in this paper.

Generation of a cover forest involves the accumulation of defect behaviors per failing pattern. Even if non-SLAT patterns are present, Assumption 1 ensures that all the affected signal lines have at least one SLAT pattern and thus, will be included in the cover forest. This means that every vertex in the cover forest provides a snapshot of the behavior caused by the defect. Heuristically, the faults belonging to the root vertices represent the strongest evidence of defect manifestation. However, every vertex must be examined to ensure completeness. For the 583 defects analyzed in this paper, cover forest sizes are listed in Table 2.

Circuit name	Avg. no. of SLAT patterns	No. of vertices		No. of faults	
		avg.	min/max	avg.	min/max
c432	9.3	9.6	1/41	42.6	2/118
c880	8.8	15.6	1/99	101.8	2/492
c1196	18.3	12.2	1/78	69.8	3/365
c1355	21.6	95.1	1/551	557.7	1/1599
c3540	19.3	20.3	1/116	107.9	9/552

Table 2: Sizes of cover forests for the analyzed circuits and defects.

Past per-test diagnosis approaches attempt to rank the stuck-at faults by finding minimal covers of faults that explain all failing [2] or all SLAT [3] patterns. The sites associated with the stuck-at faults in the covers are then reported as defect locations. An extension to the diagnosis approach in [3] is presented in [18], where the authors attempt to extract defect behavior from the test data using the defect locations identified in [3]. Specifically, the approach in [18] looks for logic conditions responsible for dominant bridge behavior. The derived conditions are subsequently verified against the entire test set.

In [12], the authors rank covers of faults using Bayesian probabilities rather than using minimal covers. Further, they attempt to correlate the top-ranking covers with a set of common fault models. While the approach in [12] is a significant step towards characterization, it has two limitations: (1) the approach is dependent upon the accuracy and applicability of the fault models, and (2) the approach does not validate the behavior of the defect conjectured after the correlation step, and therefore, does not check if the selected cover is completely consistent with observed defect behavior.

4.2 Neighborhood Function

For each potential defect site in the cover forest to manifest as stuck-line, certain logic-level conditions on the physical neighbors *may be* required[3]. Neighborhood function extraction (NFE) attempts to identify the logic-level conditions associated with defect activation [19]. For example, if the two-line short in Figure 2 behaves like a dominant bridge with s_9 stuck-at-1 being in the cover forest, NFE should identify s_7=1 (where, s_7 is a physical neighbor of s_9) as the excitation condition responsible for the fault on s_9.

For NFE, we make one reasonable assumption regarding defect misbehavior, an assumption that is central to the proposed diagnosis method.

Assumption 2. *Whenever the defect causes a logical error (i.e., a logic 0 flips to a logic 1 or vice-versa) on a signal line l_i, the physical neighbors (and the drivers of l_i and its neighbors) are the only lines (if any) that cause l_i to be erroneous.*

Given that defects are generally localized, Assumption 2 is both conservative and practical. In case the single defect in the CUD affects several signal lines (*e.g.*, the defect manifests as a multiple stuck-line fault) or the CUD has multiple defects, each faulty line must satisfy Assumption 2. The two important failure mechanisms

[3]If the defect shorts a line l_i to one of the power rails, then the neighbors other than the power rail are assumed to have no effect on l_i.

in CMOS circuits, shorts and opens, certainly satisfy Assumption 2 [4–6].

The signal lines driving a faulty line l_i and those driving the physical neighbors of l_i are also important for neighborhood function extraction. The logic values of the drivers may be an important factor that determines when l_i becomes faulty [17].

4.2.1 Function Extraction

Given a stuck-at-v fault affecting line l_i (represented as l_i/v, $v\epsilon\{0,1\}$), a set of SLAT patterns $T=\{t_1, t_2, ..., t_j\}$ explained by l_i/v, and the set of signal lines $N=\{n_1, n_2, ..., n_k\}$ that are l_i's physical neighbors (including drivers of l_i and its neighbors), a *neighborhood function* is defined as a boolean expression that describes the logic values on the "important" signal lines in N for every $t_i \epsilon T$.

Some lines in N are not important for neighborhood function extraction since their logic values are fixed for every SLAT pattern explained by the fault l_i/v. A neighbor whose logic value is implied due to the detection of l_i/v, or always lies on a sensitized path from l_i/v, *i.e.*, whose value is directly controlled by the fault can be removed for NFE. This is called *imply-based pruning* and results in a substantial reduction in the number of lines in N for each fault l_i/v. Note, only physical neighbors present in N are imply-pruned; when a neighbor is pruned, its corresponding drivers are also eliminated from N. Imply-pruned neighbors may actually play an important role in defect activation, however, their significance is reduced since logic conditions on them are implicit in l_i/v detection.

Clearly, the lines in N that can be imply-pruned based on fault detection is a function of the circuit structure as well as the fault under consideration. The average number of neighbors (along with their drivers) per fault for the 583 CUDs after employing imply-based pruning are shown in Figure 4. We use two more pruning techniques for NFE that will be described in Section 4.2.2.

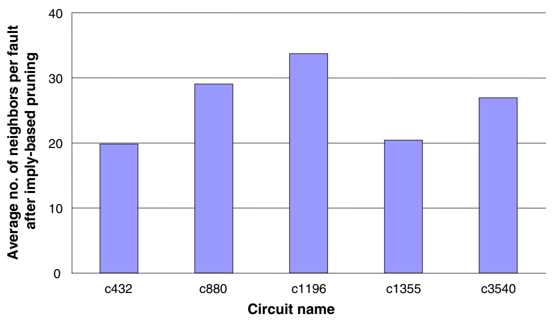


Figure 4: Average number of neighbors per fault after imply-based pruning.

For deriving a neighborhood function for l_i/v, logic values on the lines in N are obtained using good-value simulation of the SLAT patterns in T. The set of values on the lines in N for each SLAT pattern in T is called a *neighborhood state*. A neighborhood function describes any trend in the neighborhood states that may cause defect activation, specifically, that cause l_i to be stuck-at-v for the SLAT patterns in T. The task of identifying a trend in the neighborhood states is equivalent to finding a function that covers all the states, *i.e.*, the neighborhood states can be viewed as minterms of a truth table. All the other states not exhibited by the SLAT patterns in T are treated as maxterms. Boolean minimization techniques are then applied to the truth table to derive a minimum sum-of-products (SOP) expression. We use SIS [20] for deriving the SOP expression. The final boolean expression is the neighborhood function for the fault l_i/v. It is possible that some of the states declared as maxterms are exhibited by some passing patterns that detect l_i/v, resulting in the removal of l_i/v from the cover forest. A detailed analysis of this point is discussed as part of the validation stage in Section 4.3.

We can safely use good-simulation values on the physical neighbors to derive the neighborhood functions NFE only if the following assumption is valid.

Assumption 3. *For every SLAT pattern t_i explained by a fault l_i/v, the logic value on every neighbor n_i of l_i is assumed to be fault-free unless n_i lies on a sensitized path from l_i/v.*

If a defect violates Assumption 3, that is, one or more neighbors has an erroneous value, the neighborhood function derived by NFE may be incorrect.

To illustrate NFE, let us again consider the two-line short shown in Figure 2. For the fault $s_9/1$, there are six physical neighbors – s_3, s_7, s_8, s_{10}, s_{11} and s_{12}. Including the drivers of s_9 and its neighbors makes the total number of neighborhood lines to be $|N|$=11 (s_1 and s_2 drive the neighbor s_7; s_4 and s_5 drive s_8 that in turn drives s_9 and s_{10}; s_6 drives the neighbor s_{10}). However, detection of $s_9/1$ requires s_3=s_8=1 for fault activation. Also, the logic value on the neighbor s_{11} is directly implied by $s_9/1$ since s_{11} always lies on the sensitized path from $s_9/1$. Thus, s_3, s_8, and s_{11} and their drivers s_4 and s_5 are imply-pruned, leaving only six meaningful neighbors.

SLAT patterns	**Imply-pruned lines**					
	s_1	s_2	s_6	s_7	s_{10}	s_{12}
t_1	0	0	1	1	0	1
t_2	1	0	0	1	1	1

Table 3: Neighborhood states for $s_9/1$ for the defect shown in Figure 2.

Table 3 shows logic values on the six remaining neighbors for two SLAT patterns t_1 and t_2 explained by $s_9/1$. A quick analysis of the Table 3 reveals that s_7 and s_{12} are always driven to logic-1 whenever the defect manifests as $s_9/1$. It can be hypothesized at this stage that the defect is probably a short involving s_7, s_9 and s_{12}, where s_7 and s_{12} adversely affect s_9. However, no hypothesis is correct unless validated. Therefore, our methodology includes a validation step after NFE as described in Section 4.3.

4.2.2 Neighborhood Pruning

Imply-based pruning was described in Section 4.2.1. We use two other pruning techniques to reduce the list of physical neighbors for a fault location. First, given a stuck-at fault l_i/v, any neighbor n_i on a sensitized path from l_i/v is eliminated on a per-test basis because we assume that the logic value on n_i cannot enable the activation of the fault on l_i in the same clock period. We call this type of pruning *sensitization-based pruning*. Similar to the imply-based pruning, drivers of any neighbor removed using sensitization-based pruning are also eliminated from the list of neighbors (if they are not neighbors themselves).

For the defect in Figure 2 and the test patterns in Table 3, the neighbor s_{12} lies on the sensitized path of $s_9/1$ for only the SLAT pattern t_2. Since s_{12} cannot act as an aggressor causing $s_9/1$, it does not need to be considered in the neighborhood state shown in Table 3. Combining the two neighborhood states using SIS yields $Z{=}s_1s'_2s'_6s_7s_{10} + s'_1s'_2s_6s_7s'_{10}s_{12}$ as the neighborhood function for $s_9/1$.

The second type of pruning called *polarity-based pruning* involves eliminating neighbors that are not driven to the same logic value as the polarity of the fault on a per-test basis. Specifically, for a fault l_i/v that explains a SLAT pattern t_i, any neighbor that is not driven to v is assumed unable to cause a stuck-at-v fault on l_i and thus, can be removed for t_i. Applying polarity-based pruning to the neighborhood states in Table 3, and minimizing using SIS yields $Z{=}s_1s'_2s_7 + s'_2s_7s_{12}$.

All pruning techniques are heuristics that work for most defects. We can conceive of defects however, where the heuristics do not hold. For example, for a defect that manifests as $l_i/1$, neighbor n_i will be polarity-pruned if it is driven to logic-0. However, n_i driven to 0 in a particular way (say, weakly driven) may cause a test to fail, while a strongly driven 0 means the test passes. In this case, the weakly driven 0 on n_i is essential to describe the defect activation.

4.3 Validation

Inadvertently, incorrect faults are included in the cover forest due to stuck-fault equivalence and dominance. In this section, we describe how the stuck-fault relationships can be severed to eliminate incorrect faults. Specifically, we use passing patterns (and possibly additional diagnostic test patterns) to validate the correctness of each fault in the cover forest.

4.3.1 Passing Pattern Validation

The process of removing incorrect faults and the subsequent reduction of a cover forest is referred to as *deforestation*. For each fault l_i/v in the forest, deforestation starts with identification of passing patterns that detect l_i/v. If a passing pattern that detects l_i/v creates a neighborhood state that is also caused by some SLAT patterns that explain l_i/v, two different strategies can be used to decide the status of l_i/v. We can either be aggressive and remove l_i/v from the cover forest, or we can be conservative and just not consider the particular neighborhood state for NFE.

In this paper, we use the aggressive approach as a heuristic to maximize deforestation. It turns out that this heuristic works well in that the actual faults are never dropped for the defects analyzed in this paper. The heuristic assumes that neighborhood states for l_i/v derived from SLAT patterns (that explain l_i/v) represent the precise fault activation conditions. Therefore, a pattern that detects l_i/v cannot have the same excitation conditions and be a passing pattern if the defect in the CUD really manifests as l_i/v. For example, if a passing pattern creates one of the two states shown in Table 3 and also detects $s_9/1$, $s_9/1$ is removed from the forest.

The aggressive approach to deforestation is enabled by representing each fault l_i/v and its associated neighborhood function as a macrofault using fault tuples [13]. By definition, the macrofault must be detected by every SLAT pattern explained by l_i/v. If the macrofault is detected by a passing pattern, l_i/v is removed. We use FATSIM [14] to simulate the fault tuple macrofaults against passing patterns.

The aggressive approach to deforestation may cause removal of a real fault location for certain situations of defect behavior. In particular, a defect that causes multiple faulty lines that are equivalent for a given test can create a situation where one or more of the lines are removed.

Deforestation reduces the size of a cover forest, which means the number of faulty behaviors investigated during FA is also reduced. Deforestation can potentially remove all the faults belonging to a vertex, resulting in vertex elimination. The adjusted cover forest is called a *pruned* cover forest. For example, if deforestation removes the root vertex $(f_1, 5, 3)$ in Figure 3, the vertex $(f_2, 2, 5)$ becomes a root vertex and the edge between $(f_1, 5, 3)$ and $(f_6, 1, 2)$ is deleted.

To illustrate deforestation, let us again consider the short between s_7 and s_9 shown in Figure 2, where s_7 is assumed to dominate s_9. Assume that s_7 is not a neighbor of s_{11}. Therefore, while the neighborhood function for $s_9/1$ includes s_7, the function for the equivalent fault $s_{11}/0$ does not. Now, if there is a test t_i that detects $s_{11}/0$ (as well as $s_9/1$) and does not set s_7 to logic-1, t_i will be a passing pattern. When t_i is used for validation, the macrofault representing $s_{11}/0$, but not the one associated with $s_9/1$, may be detected which will cause $s_{11}/0$ to be eliminated. In this fashion, the equivalency between two otherwise indistinguishable faults can be severed, resulting in higher diagnostic resolution.

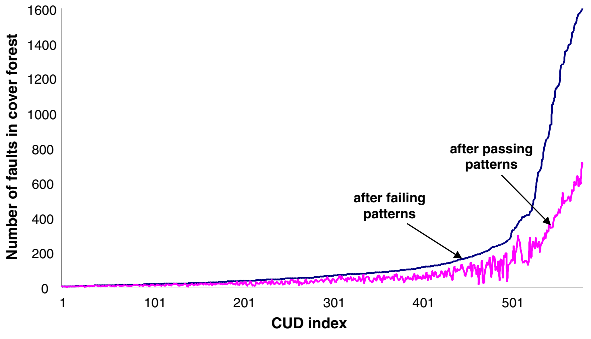


Figure 5: Reduction in the number of cover forest faults.

Figure 5 illustrates all the results for 353 two-line shorts, 123 three-line shorts, and 107 opens simulated for the five benchmarks. All the results are combined to show the overall effectiveness of deforestation. Figure 5 is a histogram comparing the number of faults in the cover forest before and after passing pattern validation. The CUDs on the x-axis are ordered from left to right based on the number of faults in their forests before deforestation.

For 583 diagnosed CUDs, there is a reduction in the number of faults for all but ten CUDs after passing patterns are utilized. For the ten CUDs, deforestation is not able to remove any fault from the cover forest since the passing patterns do not possess any macrofault-distinguishing capability. Additional ATPG is required to increase diagnostic resolution for such faults. Overall, deforestation resulted in an average reduction of over 43% with the maximum reduction being 95% for one CUD that had a dominant bridge defect. For the CUD with maximum reduction, the number of faults reduced from 21 to a single, correct fault. Passing patterns contained excellent diagnostic resolution for this particular dominant bridge case. In general, Figure 5 reveals that the greatest reduction takes place where it is most needed, that is, for large cover forests. For example, the average reduction for cover forests with more than 500 faults is 62%. As a final point, it is important to note that the real fault locations are never dropped in any of the 583 diagnosed CUDs.

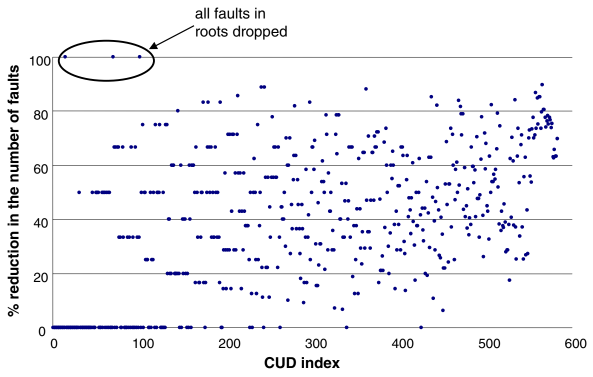


Figure 6: Ratio of initial to final number of root faults.

As discussed earlier, the stuck-at faults belonging to the root vertices in a cover forest heuristically represent the strongest evidence of defect manifestation. We conducted another experiment where only the faults in the roots of cover forests were considered for deforestation. The histogram for faults in roots reveals that on average, number of root faults reduced from 33 to 13. Figure 6 represents the reduction in number of faults as a percentage of the number of faults present in the roots before deforestation.

As highlighted in Figure 6, deforestation removed all the faults in the roots (*i.e.*, eliminated all the roots) for a few CUDs. Removal of a root implies that the most probable faults, the ones that explain the greatest number of failing patterns, may in fact be incorrect and exist initially in the forest due to equivalence and dominance. A diagnosis algorithm that uses only failing patterns may report these faults as candidates - an imprecise result. Upon further investigation of these specific CUDs, it was revealed that the real faults were indeed present in the cover forest, but not in the roots. Deforestation severed the fault relationships that caused an incorrect stuck-at fault to be present in a root.

4.3.2 Additional ATPG

Deforestation using passing patterns may not always result in a substantial reduction in cover forest size. For a passing pattern to be effective in removing an incorrect fault, the pattern must detect that fault in simulation. In the example described in Section 4.3.1, if no passing pattern detects $s_{11}/0$ without setting s_7 to logic-1, $s_{11}/0$ cannot be removed.

In order to alleviate this shortcoming, our methodology allows application of additional test patterns. The primary objective of additional ATPG is to distinguish macrofaults that remain after passing pattern validation. One way to distinguish macrofaults is described in [9]. Another objective for additional test patterns is to maximize the number of unique neighborhood states reached by test patterns [19] for each fault in the cover forest. This may, for example, lead to generation of a test pattern that detects $s_9/1$ (and $s_{11}/0$) while setting s_7=0, which can remove $s_{11}/0$.

While not a common industry practice, the generation of new test responses for increasing diagnosis accuracy has been proposed in [21] and more recently, in [9, 15, 22].

5 Conclusions

A generalized methodology for diagnosis of arbitrary defects in logic circuits has been presented. The proposed methodology addresses both defect localization and characterization. Characterization can lead to a significant reduction or altogether elimination of the effort involved in physical failure analysis. Unlike past diagnosis approaches for identifying defect types, our methodology does not use assumptions regarding defect behavior in the form of fault models; the methodology attempts to derive defect behavior from the test data.

The neighborhood functions derived in the methodology allows accurate representation of defect behavior per failing test pattern. The defect behaviors are then validated using passing patterns. Results indicate that passing pattern validation allows stuck-fault relationships (*i.e.*, equivalence under the applied test set) to be severed, leading to a reduction in the number of faults analyzed for FA.

The diagnosis methodology proposed in this paper makes certain assumptions regarding defects. These assumptions are used to keep the problem tractable. Current work focusses on relaxing some of these assumptions in order to expand the scope of defects for which the diagnosis methodology can be applied.

Acknowledgements

We would like to thank Jeff Nelson for the layouts and critical area data for the circuits used in this paper.

References

[1] Semiconductor Industry Association, "The International Technology Roadmap for Semiconductors," 2003 edition, International SEMATECH: Austin, TX, 2003.

[2] J. A. Waicukauski and E. Lindbloom, "Logic Diagnosis of Structured VLSI," *IEEE Design and Test of Computers*, vol. 6, no. 5, pp. 49–60, Aug. 1989.

[3] L. M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using Single Location at a Time (SLAT)," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 91–101, Jan. 2004.

[4] R. C. Aitken and P. C. Maxwell, "Better Models or Better Algorithms? Techniques to Improve Fault Diagnosis," *Hewlett-Packard Journal*, pp. 110–116, Feb. 1995.

[5] S. Chakravarty and Y. Gong, "An Algorithm for Diagnosing Two-Line Bridging Faults in Combinational Circuits," in *Proc. of Design Automation Conference*, pp. 520–524, June 1993.

[6] Y. Sato *et al.*, "A Persistent Diagnostic Technique for Unstable Defects," in *Proc. of International Test Conference*, pp. 242–249, Oct. 2002.

[7] W. Maly *et al.*, "Deformations of IC Structure in Test and Yield Learning," in *Proc. of International Test Conference*, pp. 856–865, Oct. 2003.

[8] I. Pomeranz *et al.*, "Defect Diagnosis Based on Pattern-Dependent Stuck-At Faults," in *Proc. of International Conference on VLSI Design*, pp. 475–480, Jan. 2004.

[9] R. Desineni *et al.*, "A Multi-Stage Approach to Fault Identification Using Fault Tuples," in *Proc. of International Symposium for Testing and Failure Analysis*, pp. 496–505, Nov. 2003.

[10] T. Vogels *et al.*, "Benchmarking of Defect-Oriented Diagnosis with a Diverse Set of IC Misbehaviors," in *Proc. of International Test Conference*, pp. 508–517, Oct. 2004.

[11] S. Venkataraman and S. B. Drummonds, "POIROT: A Logic Fault Diagnosis Tool and Its Applications," in *Proc. of International Test Conference*, pp. 253–262, Oct. 2000.

[12] D. B. Lavo, I. Hartanto and T. Larrabee, "Multiplets, Models, and the Search for Meaning: Improving Per-Test Fault Diagnosis," in *Proc. of International Test Conference*, pp. 250–259, Oct. 2002.

[13] R. D. Blanton, "Methods for Characterizing, Generating Test Sequences for, and Simulating Integrated Circuit Faults Using Fault Tuples and Related Systems and Computer Program Products," Dec. 2004, U.S. Patent No. 6,836, 856.

[14] K. N. Dwarakanath and R. D. Blanton, "Universal Fault Simulation Using Fault Tuples," in *Proc. of Design Automation Conference*, pp. 786–789, June 2000.

[15] R. D. Blanton, "Failure Diagnosis Using Fault Tuples," in *Proc. of IEEE Latin American Test Workshop*, pp. 253–257, Feb. 2001.

[16] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," in *Proc. of International Symposium on Circuits and Systems*, pp. 695–698, June 1985.

[17] P. C. Maxwell and R. C. Aitken, "Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Thresholds," in *Proc. of International Test Conference*, pp. 63–72, Oct. 1993.

[18] T. Bartenstein and J. Bhawnani, "SLAT Plus: Work in Progress," in *International IEEE Workshop on Yield Optimization and Test*, Nov. 2001.

[19] R. D. Blanton, K. N. Dwarakanath and A. B. Shah, "Analyzing the Effectiveness of Multiple-Detect Test Sets," in *Proc. of International Test Conference*, pp. 876–885, Oct. 2003.

[20] E. Sentovich, *Sequential Circuit Synthesis at the Gate Level (Chapter 5)*, Ph.D. thesis, University of California Berkeley, 1993.

[21] P. Song *et al.*, "S/390 G5 CMOS Microprocesor Diagnostics," *IBM Journal of Research and Development*, vol. 43, no. 5/6, pp. 899–913, Sept. 1999.

[22] D. Appello *et al.*, "Understanding Yield Losses in Logic Circuits," *IEEE Design and Test of Computers*, vol. 21, no. 3, pp. 208–215, May-June 2004.