Multiple-Detect ATPG Based on Physical Neighborhoods*

J. E. Nelson, J. G. Brown, R. Desineni and R. D. Blanton
Carnegie Mellon University
Pittsburgh, PA 15232
blanton@ece.cmu.edu

ABSTRACT

Multiple-detect test sets detect single stuck line faults multiple times, and thus have a higher probability of detecting complex defects. But current definitions of what constitutes a new test for a single stuck line fault do not leverage defect locality. Recent work has proposed a new metric to capture quality of a multiple-detect test set based on the number of unique states on lines in the physical neighborhood of a targeted line. This paper presents a new ATPG strategy that uses this metric to generate higher quality multiple-detect test sets.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Testing

General Terms

Reliability, Verification, Algorithms

Keywords

Multiple-Detect, N-detect, neighborhoods, ATPG, defects

1. INTRODUCTION

With nanoscale integrated circuit manufacturing trends pushing the envelope of feature size and circuit complexity, many emerging defects are not detected by standard single stuck line (SSL) tests. However, deterministic test generation for more realistic defects requires a model of the defect, making generating test sets for an unbounded set of potential defects impractical. Even generalized fault modeling mechanisms, such as fault tuples [1] and pattern faults [2], must model specific defects and are therefore limited to generating tests for a small subset of all potential defects.

One solution that helps bridge the gap between the simplicity of the SSL fault model and the thoroughness of modeling complex defects is multiple-detect, or N-detect [3]. The original goal of multiple-detect simply required that every SSL fault in the circuit be detected by N unique tests. The principle behind multiple-detect is that given an arbitrary defect that affects the targeted stuck line, multiple tests are more likely to establish a circuit state that activates the defect. Since the SSL fault is detected, there is a guaranteed sensitized path (absent of any masking) from the defect site to at least one observable point. As more circuit states are created, more excitation conditions are explored, and the probability of detecting an arbitrary defect that cannot be modeled using an SSL fault increases.

To create unique circuit states during automatic test pattern generation (ATPG) for multiple-detect test sets, the only requirement is that the test patterns that detect the same SSL fault differ by at least one input bit. While this guarantees that the state of the circuit will vary between tests, the state of the *entire* circuit is considered.

Using the entire circuit state when determining uniqueness of a test is a poor metric given the local nature of many defects. For example, both bridge and open defects in the routing, as well as many front-end defects, have behavior that is influenced by lines surrounding the defect. An improved definition for determining test uniqueness confines circuit state to the subset of signal lines surrounding the targeted stuck line. Here, this subset is referred to as the *neighborhood* of a *victim* stuck line. Using the neighborhood of a victim line exploits defect locality and limits the additional tests to those that are likely to activate a localized defect.

Prior work attempts to characterize test sets using neighborhoods based on logic-level [4] and layout-level [5] circuit descriptions. While many have extended the basic ideas of multiple-detect (see, *e.g.*, [6, 7]), this paper presents a novel automatic test pattern generation algorithm based on physical neighborhoods for multiple-detect test sets.

2. BACKGROUND

The justification for multiple detect is that multiple tests for SSL faults increases the probability of catching hard-to-detect defects due to the increased activation conditions established in the circuit. While intuitive, this was shown experimentally using production data from a Pentium 4 processor [8]. It was found that approximately 3.3% of dice that failed a 6-detect test set *passed* a standard SSL test set. This indicates that there is still potential for structural tests to detect more defects.

One multiple-detect ATPG technique proposed generates what the authors refer to as "defect aware" test patterns [9]. Signal probability enhancing (SPE) cubes are used to increase the probability of a 1 (or 0) appearing on a specific line. An SPE cube is a partially-specified input pattern that is sufficient to force a particular line in the circuit to either a logic 1 (or 0). SPE cubes are then embedded in partially specified patterns resulting from test generation for those lines with low probability. In this way, the diversity of values

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

throughout the circuit is increased. In contrast to this work, there is no restriction of circuit state when choosing the most meaningful lines. That is, any line in the circuit with low signal probability can be enhanced.

A multiple-detect test set can be roughly characterized by its value, N, the minimum number of times each SSL fault is detected. However, more sophisticated methods of determining the quality of a multiple-detect test set have been used. One method uses surrogate fault simulation, where a multiple-detect set is generated using SSL faults, then simulated against a set of more complex faults such as bridge [8] or path delay faults [10]. Surrogate simulation gives an idea of how many complex, non-targeted defects would be detected by the multiple-detect test set, but only for the set of surrogate faults simulated.

Recently, more general methods of evaluating multiple-detect test sets have been proposed that measure the ability of the test set to exercise the circuit at the neighborhood level. These metrics evaluate overall quality of the multiple-detect test set with respect to arbitrary defects occurring within the neighborhood. X-lists [4], for example, are neighborhoods that include all lines within a certain logical distance from the victim line. An X-list is treated as a black box with inputs and outputs, where the total number of unique patterns applied to the inputs of the X-list out of the total number of possible input patterns defines the *activity level* of the applied test set. A greater activity level is more desirable since it should lead to a greater probability of activating defects local to the X-list neighborhood.

However, defects are localized at the layout level, not the logic level. Therefore, in order to properly exploit this fact, neighborhoods should be determined by physical proximity, not logical proximity. Physical neighborhoods (described in greater detail in Section 3) extracted from ISCAS benchmark layouts (for a radius of 0.2μm) were used with multiple-detect tests generated by a commercial tool to measure the percentage of neighborhoods that reached N unique states, where a state is the set of logical values established by a test on the neighboring signal lines. A similar experiment was performed in [5]. Each test set was simulated using a modified version of FATSIM—a fault simulator for fault tuples [11]—that records neighborhood states. The percentage of neighborhoods that reach N unique states (not including those neighborhoods where the number of neighbors is less than $log_2 N$) is then identified. For six ISCAS benchmark layouts, the average percentage of neighborhoods that reached 2 unique states for 2-detect test sets was 95.9%. For $N = 4$, 85%; $N = 6$, 80.1%; $N = 10$, 75.6%.

This indicates that current multiple-detect test sets include wasteful patterns that do not cause any new neighborhood states to be established. Ideally, every circuit would have 100% of its neighborhoods reach N unique states. The more states established, the more confident one can be that the source of escapes is not due to slow, structural testing. Thus, there is room for improvement in current ATPG approaches used to produce multiple-detect test sets.

3. PHYSICAL NEIGHBORHOODS

The neighborhood of a given victim signal line is the set of signal lines near the victim [5]. Here, it is defined more specifically as the set of lines within radius r of the victim in every routing layer that the victim exists. This means lines near the victim in a specific metal layer will be included in the neighborhood, as well as lines near the victim in other layers in which the victim is routed. Signals that are routed in layers above or below are not considered (in other words, only intra-layer proximity is considered, not inter-layer proximity). Methods currently exist to extract physical neighborhoods as they have been defined here. In particular, critical area extraction [12] can be used in a binary sense to determine proximity of polygons. Future work may take into account the *amount* of critical area.

Neighborhoods generated from critical area extraction consist of signal lines that are likely to influence the victim line in the event of a deffect. In theory, any line in the circuit can affect any other, but by selecting a practical value for r, we simply ignore defects at the tail end of the defect size distribution (larger defects are generally less likely).

X-lists, and other logic-level heuristics for generating neighborhoods such as logic cones, could be used as an inexpensive approximation of neighborhoods extracted from the layout. But, as shown in Table 1, neither correlates well to physical neighborhoods. The *coverage* of a neighborhood is the percentage of lines in the physical neighborhood that are captured by the logic neighborhood (*i.e.*, X-list or logic cone). The *accuracy* is the percentage of logic neighborhood lines that are also in the physical neighborhood. Formally, $coverage = \frac{\|S_P \cap S_X\|}{\|S_P\|}$, and $accuracy = \frac{\|S_P \cap S_X\|}{\|S_X\|}$, where S_P is the set of signals in the physical neighborhood and S_X is the set of signals in the logic neighborhood. $\|S_i\|$ is used to denote the number of signal lines in S_i.

	Logic cone		X-list (radius 2)	
Circuit name	**Average coverage (%)**	**Average accuracy (%)**	**Average coverage (%)**	**Average accuracy (%)**
c432	70	2	35	38
s1196	65	2	31	41
c3540	62	0	22	37
s13207	17	2	27	38
s15850	9	1	28	41

Table 1: The average coverage and accuracy of X-lists of radius 2 versus physical neighborhoods of radius 0.2μm for four ISCAS benchmark circuits.

These metrics describe how well X-lists and logic cones capture the physical neighborhoods. Table 1 shows the average coverage and accuracy values over all neighborhoods for several ISCAS benchmark circuits. It is clear that both coverage and accuracy is quite low, indicating that the intersection between physical and logical neighborhoods is small. Logic cones are particularly poor, capturing many lines that are in fact in the physical neighborhood, but also capturing an extraordinary number of lines that are not, resulting in extremely low accuracy.

The layouts used in this paper were created from ISCAS benchmarks using commercial automatic place and route tools and a $0.18\mu m$ standard cell library. The circuits are either combinational, or fully scanned and treated as combinational during test generation. Layout elements associated with the test logic and clock tree are ignored during neighborhood analysis.

4. NEIGHBORHOOD-BASED ATPG

The objective here is to improve ATPG for multiple-detect test sets by incorporating neighborhood-based metrics. Specifically, physical neighborhoods are used because they are the most accurate representation of signal proximity.

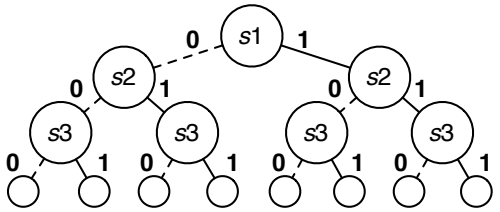

Figure 1: A sample neighborhood state tree.

Traditional multiple-detect requires that every SSL fault be detected at least N times. To improve test set quality, this definition is modified such that every SSL fault must be detected at least N times, where the test patterns create at least N unique states in its physical neighborhood (if possible). The improved ATPG algorithm is designed to work within the framework of fault tuples [1] and FATGEN, an ATPG system based on fault tuples [13].

A fault tuple is a three tuple $f = (l, v, T)$, where l is any signal line in the circuit, v is from the set {0, 1, X, U}, and T is an inter-clock cycle range that describes when l is equal to v. Fault tuples can be ANDed together to form products, and products can be ORed together to form macrofaults. A macrofault represents activation and observation requirements for the logic misbehavior of one or more defects. An example SSL fault, $s0$ stuck-at 0, is described by the macrofault $(s0, 0, i)^e$; meaning in any clock cycle, i, if in the good circuit $s0$ is driven to a logic one, the fault tuple indicates $s0$ will instead be an erroneous logic zero. If the error is propagated to an observable point, then the macrofault is detected.

A test can be generated for a macrofault that creates a specific neighborhood state surrounding an SSL fault by ANDing together tuples that require neighborhood lines to have specific values. For example, if a victim SSL fault $s0$ stuck-at 0 has neighbors $s1$, $s2$, and $s3$, test generation for the macrofault $(s0, 0, i)^e(s1, 0, i)^c(s2, 0, i)^c(s3, 1, i)^c$ will lead to a test pattern that detects $s0$ stuck-at 0 with $s1s2s3 = 001$. It should be noted that IBM pattern faults [2] can also be used in a similar manner.

4.1 Neighborhood State Tree

To keep track of unique neighborhood states, a binary tree is built to represent all possible neighborhood states. Only one neighborhood state tree exists at any given time; it is built only for the currently targeted neighborhood. Since the size of the tree must be limited by the effort required to search it, optimizing the data structure that stores the state tree has a negligible effect. Figure 1 illustrates a three-line neighborhood. In this case, there is a victim SSL fault, $s0$ stuck-at 0 (not shown in the figure), and three neighbors $s1$, $s2$, and $s3$. The path from the root node to any node in the tree represents a macrofault specifying a partial or complete state in the neighborhood. A partial state simply means that the macrofault requires a subset of the neighborhood signals to have specific values. For example, if the first two right edges are followed, this leads to the macrofault $(s0, 0, i)^e(s1, 1, i)^c(s2, 1, i)^c$. While the macrofault only specifies a partial state, should test generation be successful, the resulting (fully-specified) test pattern will create a complete state in the neighborhood.

By searching the tree shown in Figure 1 in a breadth-first fashion and performing ATPG on the macrofault representing each path, the entire state space of the neighborhood is guaranteed to be explored. However, if at any node a test cannot be generated for the corresponding macrofault, its entire subtree (with the node serving as the root) is removed and not explored because all macrofaults in the subtree will be redundant as well. Some states can be quickly and easily identified as impossible-to-reach using standard SSL implications before the state tree is built. With minimal effort, this reduces the overall size of the tree.

```
1   for(k=0; k < num_neighborhoods; k++)
2     num_atpg = max_atpg_per_neighborhood;
3     while((s_k < N) && (num_atpg > 0))
4       Select t from tree using BFS;
5       Construct mf to represent t;
6       num_atpg − −;
7       if(P = ATPG(mf) is successful)
8         random_fill(P);
9         state_dropping(P, all_neighborhoods);
10      else
11        remove subtree of state t;
12  state_dropping(P, nbrhds)
13    Simulate pattern P;
14    For each n_k in nbrhds
15      if(SSL for n_k is detected)
16        mark state;
17        s_k + +;
```

Figure 2: Pseudo-code for the NBATPG algorithm.

4.2 ATPG

The algorithm for neighborhood-based ATPG (NBATPG) is shown in Figure 2. It has an outside loop that iterates over all neighborhoods. On line 2, num_atpg is set to a user-specified threshold, $max\_atpg\_per\_neighborhood$; this value limits the number of ATPG calls for each neighborhood. Since neighborhood state trees can be large, this threshold prevents test generation from spending excessive time on a neighborhood for which unique states are difficult to establish. They can be revisited later with this limit increased.

For the neighborhood currently targeted, the inner loop defined on line 3 continues as long as $s_k < N$, and there have been fewer than the maximum number of ATPG calls allowed for this neighborhood, where s_k refers to the number of unique states reached for neighborhood k. For each iteration, a new state t is selected from the neighborhood state tree. By choosing states from the tree using a breadth first search, it is guaranteed that no state will be visited twice. A macrofault, mf, is constructed to represent state t for which test generation is performed using FATGEN. Should test generation fail, the entire subtree of the node targeted can be removed from the state tree since the partial state causing redundancy will be present in any state created from the subtree. If test generation is successful, unspecified inputs in the test pattern are first randomly filled, then state dropping is performed through fault simulation.

State dropping simulates a newly generated test pattern over the entire set of neighborhoods, where n_k represents neighborhood k. Any new states established for any neighborhood will be marked, and thus avoided during future test generation. Furthermore, when a new state is marked for a neighborhood, the state count for that neighborhood is incremented (as shown on line 17).

5. ATPG EXPERIMENT RESULTS

The neighborhood-based ATPG algorithm presented in Section 4 is applied to several ISCAS benchmarks to gen-

	c432	s1196	c3540	s13207	s15850
$N = 2$	99.11	90.46	98.88	97.22	97.48
	88.74	82.66	89.80	83.79	87.22
$N = 4$	93.47	84.24	95.2	89.95	92.94
	60.99	68.31	80.55	74.82	80.7
$N = 6$	87.8	78.64	94.23	88.96	91.7
	45.88	63.56	76.86	71.55	78.77
$N = 10$	79.26	72.3	91.81	88.29	89.87
	35.66	53.75	73.3	68.95	85.12

Table 2: Percentage of neighborhoods that reach N unique states for ISCAS benchmark circuits.

	c432	s1196	c3540	s13207	s15850
$N = 2$	294	629	927	2862	3730
	92	262	258	646	820
$N = 4$	663	1651	2420	8398	10857
	156	424	456	1064	1444
$N = 6$	1158	3316	4685	16859	22047
	156	636	648	1512	1992
$N = 10$	1889	6473	8738	31584	42060
	250	970	990	2190	3260

Table 3: Test set sizes for various ISCAS circuits and various values of N.

erate improved multiple-detect test sets for $N \in \{2, 4, 6, 10\}$. Additionally, a standard multiple-detect algorithm was used to generate multiple-detect test sets for those same values of N. The standard algorithm uses a classic definition to determine whether patterns are unique. That is, any difference between test vectors is sufficient to include the test, rather than examining neighborhood state.

Table 2 shows the percentage of neighborhoods that reach N unique states for NBATPG (top number) and the standard multiple-detect test set (bottom number). Ideally, this number would be 100% for all entries, indicating that every neighborhood in the circuit reached N unique states during testing. However, 100% may not be reached for NBATPG either due to state redundancy or ATPG aborts.

It can be seen from the table that NBATPG has significantly more neighborhoods that reach their full potential of at least N unique states. Additionally, as N increases, the difference between the NBATPG and the standard multiple-detect test set generally increases. This indicates that as it becomes necessary to find the neighborhood states that are harder to establish, NBATPG excels whereas standard multiple-detect falls significantly short.

In all cases, the more neighborhoods that achieve their full potential of N unique states, the more likely a defect occurring in the neighborhood is to be detected. This is due to the increased number of activation conditions explored, while always sensitizing a propagation path. These simulation results indicate that NBATPG test sets in all cases lead to more neighborhoods reaching their full potential.

It is important to note that, currently, this gain in neighborhood state exploration comes at the expense of test set size. Table 3 shows the sizes of test sets used to generate Table 2. In all cases, the NBATPG test set sizes are larger. We believe this is due to the absence of test set compaction in our algorithm. Note that on line `8` of the NBATPG algorithm, newly generated test patterns with unknown bits are filled with random logic values.

The NBATPG test sets could benefit greatly from test set compaction, but the compaction algorithm must be neighborhood aware. Typical compaction techniques only consider SSL faults detected by each test pattern, but now, states established by the test pattern must be taken into account.

6. SUMMARY

Prior work was described that both motivates the use of multiple-detect and outlines metrics for capitalizing on the localized nature of defects. Logic-level neighborhoods were shown to be an inexpensive but inaccurate approximation of physical neighborhoods. Subsequently, an ATPG system that generates higher-quality multiple-detect test sets based on physical neighborhoods was presented. Using a directed search, tests are generated for specific neighborhood states until N unique states are established, or no states remain.

Results were generated for several circuits that showed how using NBATPG leads to more neighborhoods reaching N unique states. It is hoped that the increased diversity of patterns will lead to a greater number of defects being detected at test time, resulting in a reduction of defective parts being shipped to the customer. Currently, experiments with real silicon are underway to investigate the effectiveness of these tests.

7. REFERENCES

[1] 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.

[2] B. Keller, "Hierarchical Pattern Faults for Describing Logic Circuit Failure Mechanisms," 1994, U.S. Patent no. 5,546,408.

[3] S. C. Ma, et al., "An Experimental Chip to Evaluate Test Techniques Experiment Results," *ITC*, pp. 663–672, Oct. 1995.

[4] A. Jain, "Arbitrary Defects; Modeling and Applications," *Master's Thesis, Rutgers Univ.*, 1999.

[5] R. D. Blanton, et al., "Analyzing the Effectiveness of Multiple-Detect Test Sets," *ITC*, pp. 876–885, 2003.

[6] S. Lee, et al., "A New ATPG Algorithm to Limit Test Set Size and Achieve Multiple Detections of all Faults," *DATE*, 2002.

[7] B. Benware, et al., "Impact of Multiple-Detect Test Patterns on Product Quality," *ITC*, 2003.

[8] S. Venkataraman, et al., "An Experimental Study of N-Detect Scan ATPG Patterns on a Processor," *VTS*, pp. 23–29, April 2004.

[9] H. Tang, et al., "Defect Aware Test Patterns," *DATE*, vol. 1, pp. 450–455, 2005.

[10] I. Pomeranz and S. M. Reddy, "On N-Detection Test Sets and Variable N-Detection Test Sets for Transition Faults," *IEEE Trans. on CAD of Integrated Circuits*, vol. 19, no. 3, pp. 372–383, March 2002.

[11] K. N. Dwarakanath and R. D. Blanton, "Universal Fault Simulation Using Fault Tuples," *DAC*, pp. 786–789, June 2000.

[12] W. Maly and J. Deszczka, "Yield Estimation Model for VLSI Artwork Evaluation," *Electronic Letters*, vol. 19, no. 6, pp. 226–227, March 1983.

[13] R. Desineni, et al., "Universal Test Generation Using Fault Tuples," *ITC*, pp. 812–819, Oct. 2000.